BC FT-672
CLAUDIO GARCIA

**ESCALAÇÃO ESTÁTICA SUB-ÓTIMA DE TAREFAS PARCIALMENTE
ORDENADAS EM REDES DE TRANSPUTERS**

Tese apresentada à Escola
Politécnica da Universidade de
São Paulo para obtenção do
título de Doutor em Engenharia.

São Paulo
1992

CONSULTA
FT-672

BC FT 682

CLAUDIO GARCIA

**ESCALAÇÃO ESTÁTICA SUB-ÓTIMA DE TAREFAS PARCIALMENTE
ORDENADAS EM REDES DE TRANSPUTERS**

Tese apresentada à Escola
Politécnica da Universidade de
São Paulo para obtenção do
título de Doutor em Engenharia.

Área de Concentração:
Engenharia de Computação e
Sistemas Digitais

Orientador:
Moacyr Martucci Júnior

1992

Garcia, Claudio

Escalação estática sub-ótima de tarefas parcialmente ordenadas em redes de Transputers. São Paulo, 1992.

324p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Escalação de tarefas 2.Modelamento dinâmico de processadores 3. Transputers - Aplicação 4.Processamento paralelo I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t

A minha filha **Natalia**,
exemplo vivo do amor eterno.

AGRADECIMENTOS

A meu orientador Prof. Dr. Moacyr Martucci Jr. pela constante presença e aconselhamento sincero.

A meu amigo Dr. Antonio Carlos de Oliveira Barroso pela compreensão e apoio.

A meus companheiros Luis Fernando Silva Rodrigues Teixeira e Renato Massucci Pessoa Cesar pela inestimável ajuda.

A minha esposa e filhos, pelos muitos momentos de minha ausência que sempre foram retribuídos com amor e carinho.

SUMÁRIO

Lista de figuras

Lista de tabelas

Lista de abreviaturas

Resumo

"Abstract"

1	INTRODUÇÃO	1
1.1	Objetivo do Trabalho	2
1.2	Motivação para a Realização deste Trabalho	4
1.3	Metodologia de Trabalho Empregada	4
1.3.1	Metodologia para Geração de Software Paralelizável	5
1.3.2	Método Científico Utilizado	6
1.4	Abrangência do Trabalho	7
1.5	Organização do Trabalho	7
2	EMBASAMENTO TEÓRICO SOBRE ESCALAÇÃO DE TAREFAS	9
2.1	Geração de Programas Multiprocessados	9
2.1.1	Particionamento de Programas	9
2.1.2	Escalação de Tarefas	10
2.2	Problema Geral de Escalação	11
2.2.1	Definição de Termos e Estabelecimento de Conceitos Aplicáveis a Problemas de Escalação	11
2.2.1.1	Definição de Termos	12
2.2.1.2	Estabelecimento de Conceitos e Definições	13
2.2.2	Descrição do Modelo Básico a ser Empregado	18
2.2.2.1	Recursos	18
2.2.2.2	Sistemas de Tarefas	20

2.2.2.3 Restrições Impostas	28
2.2.2.4 Imprecisões no Modelo Considerado	28
2.2.3 Descrição das Funções-objetivo mais Utilizadas	30
2.2.4 Dados para Criação e de Entrada/Saída de um Algoritmo Escalador	33
2.2.4.1 Informações para Desenvolver um Algoritmo Escalador	33
2.2.4.2 Dados de Entrada/Saída de um Algoritmo Escalador	34
2.3 Escalação em Multiprocessadores Fracamente Acoplados	36
2.3.1 Conflito entre Balanceamento da Carga Computacional e Custo da Comunicação Interprocessadores	36
2.3.2 Alternativa para a Forma Clássica de Comunicação entre Tarefas	40
2.4 Métodos de Solução de Problemas de Escalação	41
2.4.1 Dificuldades Encontradas na Solução de Problemas de Escalação	43
2.4.1.1 Problemas Intratáveis	44
2.4.1.2 Classe de Problemas NP-completos	49
2.4.2 Técnicas Heurísticas de Escalação de Tarefas	52
2.4.2.1 Anomalias em Escalações em Lista	55
2.4.2.2 Exemplos de Escalação em Lista para Tarefas Parcialmente Ordenadas	57
2.4.2.3 Escalação com Tempos de Execução Decrescentes (LPT)	59
2.4.2.4 Escalação em Nível ou do Caminho Crítico (CP)	59
2.4.2.5 Método Melhorado do Caminho Crítico (CP/MISF)	60
2.4.2.6 Escalação em Lista para dois Casos Especiais	62
2.4.2.7 Escalação em Sistemas com Tempo de Comunicação Interprocessadores	64
2.4.3 Métodos de Avaliação de Algoritmos de Aproximação	65
2.4.3.1 Análise de Algoritmos de Aproximação pela Técnica do Desempenho Garantido	66

2.4.3.2	Definição da Técnica do Desempenho Garantido para Problemas de Otimização Combinatorial	68
2.4.3.3	Razões de Desempenho Conhecidas para Problemas de Escalação	69
2.4.3.4	Análise de Algoritmos de Aproximação pelo Método do Desempenho Esperado	73
2.4.3.5	Limites Probabilísticos no Desempenho de Escalação em Lista para Tarefas Independentes	75
2.4.3.6	Comparação entre Escalação de Desempenho Esperado e de Desempenho Garantido para Método LPT	77
2.4.4	Técnicas Enumerativas de Escalação de Tarefas	78
2.4.4.1	Programação Dinâmica	79
2.4.4.2	Método Ramifique-e-Delimita (branch-and-bound)	80
2.4.4.3	Relação entre os Métodos RD e Programação Dinâmica	82
2.4.5	Limites no Número Mínimo de Processadores e no Tempo Mínimo	82
2.5	Evolução Histórica dos Estudos sobre Escalação	84
2.5.1	Cronologia da Resolução de Problemas Gerais de Escalação	85
2.5.1.1	Escalação em Sistemas Fracamente Acoplados	91
2.5.2	Cronologia dos Métodos de Delimitação	93
2.5.3	Evolução dos Métodos de Solução	94
2.5.4	Pesquisas Recentes sobre Escalação em Transputers	96
3	AMBIENTE DISPONÍVEL DE HARDWARE DE SOFTWARE	98
3.1	Estrutura Funcional de Multiprocessadores	98
3.1.1	Multiprocessadores Fracamente Acoplados	99
3.2	Conceitos sobre a Linguagem de Programação Occam	100
3.2.1	Processos em Occam	101
3.2.2	Comunicação e Sincronização	102

3.2.3	Construções em Occam	103
3.2.4	Alocação de Dados em Memória	104
3.3	Características Relevantes do Transputer	107
3.3.1	Arquitetura dos Transputers	107
3.3.2	Concorrência e Comunicação	108
3.3.3	Escalador Interno	111
3.3.4	Configuração para Multiprocessamento	113
3.4	Processamento de Tarefas em Transputers	114
3.4.1	Escalador Interno	115
3.4.2	Alteração da Velocidade de Processamento em PAR e PRI PAR	119
3.4.3	Análise de Intervalos de Tempo Relevantes para o Programa Escalador	121
3.5	Roteamento de Mensagens	123
3.5.1	Processos Roteador e Distribuidor	124
3.5.2	Cabeçalho das Mensagens	127
3.5.3	Influência dos Processos Roteador/Distribuidor nos Tempos de Processamento	128
3.5.4	Tempos Envolvidos no Roteamento de Mensagens	129
3.5.5	Comunicação entre Processo Roteador e Processos de Baixa Prioridade	130
3.6	Escalação Convencional versus Escalação em Transputers	131
4	CONCEPÇÃO DO PROGRAMA ESCALADOR	134
4.1	Apresentação do Problema a ser Resolvido	134
4.1.1	Especificação do Problema a ser Resolvido	134
4.1.2	Classificação do Problema a ser Resolvido	136
4.2	Especificação de Requisitos do Programa Escalador	137
4.2.1	Classificação do Programa Escalador	137
4.2.2	Requisitos do Programa Escalador	138
4.2.3	Interfaces do Programa Escalador	139
4.2.4	Seleção do Método para Resolver o Problema de Escalação Proposto	140

4.2.5	Descrição do Programa Escalador	141
4.2.6	Descrição das Funções Executadas pelo Programa Escalador	144
4.2.7	Especificação dos Testes de Aceitação do Programa Escalador	170
4.3	Diferenças entre o Programa Escalador e os Métodos Convencionais de Escalação Estática	172
5	PROJETO DO PROGRAMA ESCALADOR	174
5.1	Projeto Básico do Programa Escalador	174
5.1.1	Detalhamento das Funções do Gerenciador do Programa Escalador	174
5.1.1.1	Formatação dos Dados de Entrada (formata_entrada)	175
5.1.1.2	Seleção da Melhor Escalação Sub-ótima (seleciona_escalção)	177
5.1.1.3	Formatação dos Dados de Saída (formata_saída)	178
5.1.2	Detalhamento das Funções do ESCEXT	179
5.1.2.1	Iniciação dos Parâmetros para Geração de Escalações (inicia_escalador)	180
5.1.2.2	Geração de Escalação pelo Algoritmo CP/NSI (algoritmo_CP/NSI)	183
5.1.2.3	Geração de Escalação pelo Algoritmo CP/NSI/PAR (algoritmo_CP/NSI/PAR)	185
5.1.2.4	Geração de Escalação pelo Algoritmo N/pSEQ/CP (algoritmo_N/pSEQ/CP)	186
5.1.2.5	Geração de Escalação pelo Algoritmo N/pSEQ/CP/PAR (algoritmo_N/pSEQ/CP/PAR)	187
5.1.2.6	Geração de Escalação pelo Algoritmo N/CP (algoritmo_N/CP)	187
5.1.2.7	Geração de Escalação pelo Algoritmo N/CP/PAR (algoritmo_N/CP/PAR)	188
5.1.2.8	Geração de Escalação pelo Algoritmo N/CPn/NSI (algoritmo_N/CPn/NSI)	189
5.1.2.9	Geração de Escalação pelo Algoritmo N/CPn/NSI/PAR (algoritmo_N/CPn/NSI/PAR)	189
5.1.2.10	Geração de Escalação pelo Algoritmo LPT (algoritmo_LPT)	190

5.1.2.11	Geração de Escalação pelo Algoritmo MTP (algoritmo_MTP)	191
5.1.3	Detalhamento das Funções do GEREDE	193
5.1.3.1	Coordenação da Simulação Dinâmica da Rede de Transputers (coordena_ simulação)	194
5.1.3.2	Iniciação da Simulação Dinâmica da Rede de Transputers (inicia_simulação)	196
5.1.3.3	Controle de IRFE (controla_IRFE)	197
5.1.3.4	Controle de ITQT (controla_ITQT)	198
5.1.3.5	Controle de ICMT (controla_ICMT)	199
5.1.4	Detalhamento das Funções do MSEQ/PAR	200
5.1.4.1	Controle da Ordem de Chamada e do Regime de Operação dos Processos (controla_chamada)	200
5.1.4.2	Manipulação da Recepção de Mensagens (recebe_mensagem)	201
5.1.4.3	Manipulação da Transmissão de Mensagens (transmite_mensagem)	203
5.1.4.4	Manipulação da Reativação de Processos (reativa_processo)	205
5.1.4.5	Manipulação de Fim de Processamento de uma Tarefa (finaliza_processamento)	206
5.1.4.6	Manipulação de Temporização (temporiza) ..	208
5.1.4.7	Manipulação de Término de Quantum de Tempo (manipula_quantum)	209
5.1.4.8	Funções Auxiliares do MSEQ/PAR	210
5.1.5	Diagramas de Fluxo de Dados	211
5.1.6	Definição da Base de Dados	214
5.1.6.1	Variáveis Globais do Programa Escalador ..	214
5.1.6.2	Variáveis Utilizadas pelo Gerenciador do Programa Escalador	214
5.1.6.3	Variáveis Utilizadas pelo ESCEXT	215
5.1.6.4	Variáveis Utilizadas pelo SIMREDE	216
5.2	Projeto Detalhado do Programa Escalador	217
5.2.1	Módulos do Gerenciador do Programa Escalador (programa_escalador)	218

5.2.2	Módulos do ESCEXT	221
5.2.3	Módulos do GEREDE	232
5.2.4	Módulos do MSEQ/PAR	235
6	IMPLEMENTAÇÃO E TESTES DO PROGRAMA ESCALADOR	244
6.1	Comentários sobre a Implementação do Programa Escalador	244
6.2	Entrada e Saída dos Dados no Programa Escalador	245
6.2.1	Geração do Arquivo de Entrada de Dados dos Processos	246
6.2.1.1	Inserção dos Dados de Entrada dos Processos	246
6.2.1.2	Coleta dos Dados de Entrada dos Processos que Envolvem Tempo	250
6.2.2	Geração do Arquivo de Entrada de Dados dos Transputers	260
6.2.3	Formato do Arquivo de Saída de Dados	261
6.3	Testes de Aceitação do Programa Escalador	261
6.3.1	Testes de Aceitação dos Módulos do Programa Escalador	262
6.3.2	Testes de Aceitação dos Algoritmos Heurísticos de Escalação	262
6.3.3	Testes de Validação do Modelo Dinâmico da Rede de Transputers	263
6.3.3.1	Avaliação de Desempenho do SIMREDE	264
6.3.3.2	Conclusões a Respeito do Desempenho do SIMREDE	286
6.3.4	Teste de Aceitação do Programa Escalador	288
6.3.4.1	Apresentação do Sistema de Tarefas e da Rede de Transputers	288
6.3.4.2	Montagem do Arquivo de Dados de Entrada dos Processos	293
6.3.4.3	Montagem do Arquivo de Dados de Entrada dos Transputers	300
6.3.4.4	Apresentação dos Resultados	300
6.3.4.5	Avaliação dos Resultados do Teste de Aceitação	301

6.3.4.6	Observações Importantes a Respeito do Teste de Aceitação	303
6.3.5	Comentários Finais a Respeito do Programa Escalador	308
7	CONSIDERAÇÕES FINAIS	310
7.1	Análise Crítica	310
7.2	Colaboração deste Trabalho no Campo de Escalação Estática	311
7.3	Continuidade do Trabalho	312
	ANEXO - FORMA DE TRATAMENTO DOS TEMPOS ENVOLVIDOS NO MODELAMENTO DE SEQ E PAR.....	315
	REFERÊNCIAS BIBLIOGRÁFICAS	318

LISTA DE FIGURAS

Fig. 2.1 Representação de um grafo acíclico dirigido de $(P, <, \{r_i\}, \{O_{ij}\})$	25
Fig. 2.2 Exemplo de escalação de tarefas em três processadores usando carta de Gantt	35
Fig. 3.1 Arranjo de ocupação da memória em programas Occam	106
Fig. 3.2 Mapa de memória utilizado pelo compilador Occam	106
Fig. 3.3 Programa em Occam-2 descrevendo os processos Roteador e Distribuidor	126
Fig. 3.4 Formato do cabeçalho das mensagens	128
Fig. 3.5 Exemplo para comparar tempos de processamento de tarefas	129
Fig. 3.6 Grafo de precedência entre três tarefas interdependentes	132
Fig. 3.7 Codificação dos processos P1, P2 e P3	132
Fig. 3.8 Exemplo de diferentes escalações de P1, P2 e P3	133
Fig. 4.1 Diagrama esquemático das partes constituintes do Programa Escalador	142
Fig. 4.2 Estrutura do vetor de listas de eventos VLEV	146
Fig. 4.3 Estrutura do registro EVENTO	147
Fig. 4.4 Forma de apresentação dos dados de saída ao usuário	148
Fig. 4.5 Estrutura do vetor de listas VLLESC	150
Fig. 4.6 Estrutura do elemento REGIME	150
Fig. 4.7 Grafo de precedência mostrando o comprimento da rota crítica de cada tarefa	151
Fig. 4.8 Grafo de precedência mostrando o nível das tarefas	153
Fig. 4.9 Exemplo de escalação pelo método MTP sem resíduo	158
Fig. 4.10 Exemplo de escalação parcial pelo método MTP com resíduo	158
Fig. 4.11 Exemplo de escalação pelo método MTP com resíduo	159

Fig. 4.12 Composição dos registros RIRM e RICMT	163
Fig. 4.13 Composição do registro RMTAS	164
Fig. 4.14 Composição do registro RIRFE	170
Fig. 5.1 Diagrama da interrelação entre as funções do SIMREDE	193
Fig. 5.2 Fluxo de dados com o usuário	212
Fig. 5.3 Fluxo de dados entre as partes do Programa Escalador	213
Fig. 5.4 Fluxo de dados do SIMREDE	213
Fig. 5.5 Diagrama de blocos do Programa Escalador	218
Fig. 5.6 Diagrama de blocos do módulo formata_entrada	219
Fig. 5.7 Diagrama de blocos do módulo pega_número(ARQ)	220
Fig. 5.8 Diagrama de blocos do módulo formata_saída	221
Fig. 5.9 Diagrama de blocos do módulo inicia_escalador	223
Fig. 5.10 Diagrama de blocos do módulo inicia_VCP	223
Fig. 5.11 Diagrama de blocos do módulo calcula_VCP(i)	223
Fig. 5.12 Diagrama de blocos do módulo inicia_VCPn	224
Fig. 5.13 Diagrama de blocos do módulo inicia_VN	224
Fig. 5.14 Diagrama de blocos do módulo calcula_VN(i)	225
Fig. 5.15 Diagrama de blocos do módulo escala(ALGOR)	225
Fig. 5.16 Diagrama de blocos do módulo ordena(VPRIOR,ALGOR)	225
Fig. 5.17 Diagrama de blocos da função de comparação cp_nsi_cmp(i,j)	226
Fig. 5.18 Diagrama de blocos da função de comparação n_tseq_cp_cmp(i,j)	226
Fig. 5.19 Diagrama de blocos da função de comparação n_cp_cmp(i,j)	227
Fig. 5.20 Diagrama de blocos da função de comparação n_cpn_nsi_cmp(i,j)	227
Fig. 5.21 Diagrama de blocos da função de comparação lpt_cmp(i,j)	228
Fig. 5.22 Diagrama de blocos do módulo distribui_SEQ(VPRIOR)	228

Fig. 5.23 Diagrama de blocos do módulo seleciona_transputer (VTAPT)	228
Fig. 5.24 Diagrama de blocos do módulo aloca_SEQ(i,t,VLLESC)	229
Fig. 5.25 Diagrama de blocos do módulo distribui_compactado (VPRIOR)	229
Fig. 5.26 Diagrama de blocos do módulo aloca_PAR(i,t,VLLESC)	230
Fig. 5.27 Diagrama de blocos do módulo distribui_PAR(VPRIOR)	230
Fig. 5.28 Diagrama de blocos do módulo distribui_MTP(VPRIOR)	231
Fig. 5.29 Diagrama de blocos do módulo coordena_simulação	233
Fig. 5.30 Diagrama de blocos do módulo inicia_simulação	234
Fig. 5.31 Diagrama de blocos do módulo controla_IRFE	234
Fig. 5.32 Diagrama de blocos do módulo controla_ITQT	235
Fig. 5.33 Diagrama de blocos do módulo controla_ICMT	235
Fig. 5.34 Diagrama de blocos do módulo controla_chamada(NTS)	237
Fig. 5.35 Diagrama de blocos do módulo recebe_mensagem (NTS,NPE)	238
Fig. 5.36 Diagrama de blocos do módulo transmite_mensagem (NTS,NPE)	239
Fig. 5.37 Diagrama de blocos do módulo reativa_processo(NPR)	240
Fig. 5.38 Diagrama de blocos do módulo finaliza_ processamento (NTS)	240
Fig. 5.39 Diagrama de blocos do módulo temporiza(NTS,NPE)	241
Fig. 5.40 Diagrama de blocos do módulo manipula_quantum(NTS)	241
Fig. 5.41 Diagrama de blocos do módulo atualiza_VIPPEV/VTPEV (NTS,NPE)	242
Fig. 5.42 Diagrama de blocos do módulo escala_processo(NTS)	242
Fig. 5.43 Diagrama de blocos do módulo insere_ordem_ crescente (RGT,LST)	243
Fig. 6.1 Exemplo de arranjo para a coleta de tempos de processo em SEQ	253
Fig. 6.2 Exemplo de arranjo para a coleta de tempos de processo em PAR	254
Fig. 6.3 Exemplo de arranjo para a coleta de pDMEXT	255
Fig. 6.4 Exemplo de arranjo para a coleta de pDMINT	256

Fig. 6.5 Exemplo de cálculo do intervalo de duração de cada evento	257
Fig. 6.6 Exemplo de arquivo de saída mostrando a escalação gerada	261
Fig. 6.7 Grafo de precedência utilizado para testar os algoritmos heurísticos	263
Fig. 6.8 Grafo de precedência utilizado para testar os algoritmos heurísticos	263
Fig. 6.9 Topologia da rede de Transputers utilizada para o quinto teste	280
Fig. 6.10 Grafo de precedência usado para testar o Programa Escalador	289
Fig. 6.11 Apresentação do resultado da escalação gerada no teste de aceitação do Programa Escalador	301
Fig. 6.12 Escalações geradas pelos algoritmos CP/NSI, CP/NSI/PAR, N/pSEQ/CP e N/pSEQ/CP/PAR	303
Fig. 6.13 Escalações geradas pelos algoritmos N/CP, N/CP/PAR, N/CPn/NSI e N/CPn/NSI/PAR	304
Fig. 6.14 Escalações geradas pelos algoritmos LPT e MTP	304
Fig. 6.15 Duração dos eventos das tarefas na execução de um novo teste	306

LISTA DE TABELAS

Tab. 2.1	Comparação de funções tempo-polinomiais e tempo-exponenciais	46
Tab. 2.2	Efeito de tecnologias mais avançadas em diversos algoritmos tempo-polinomiais e tempo-exponenciais	47
Tab. 2.3	Exemplos da complexidade de alguns problemas típicos de escalação	51
Tab. 2.4	Comparação de dois limites para WCG/WOPT	72
Tab. 2.5	Valores de d_n, ϵ obtidos do teste estatístico de Kolmogorov-Smirnov	76
Tab. 2.6	Limite superior na relação WLPT/WOPT com limite de confiança de 99%	77
Tab. 3.1	Características básicas dos Transputers	107
Tab. 3.2	Taxas efetivas de troca de dados entre Transputers	108
Tab. 4.1	Tabela do fator de correção FC para cálculo dos comprimentos normalizados das rotas críticas das tarefas	152
Tab. 5.1	Tabela de escolha de algoritmos heurísticos de acordo com valor de ALGOR	180
Tab. 5.2	Tabela de escolha de chamada de funções de acordo com o próximo evento previsto em cada Transputer da rede	195
Tab. 5.3	Variáveis globais do Programa Escalador	214
Tab. 5.4	Variáveis restritas ao Gerenciador do Programa Escalador	215
Tab. 5.5	Variáveis globais utilizadas pelo Gerenciador do Programa Escalador	215
Tab. 5.6	Variáveis restritas ao ESCEXT	215
Tab. 5.7	Variáveis globais utilizadas pelo ESCEXT	216
Tab. 5.8	Variáveis restritas ao SIMREDE	216
Tab. 5.9	Variáveis globais utilizadas pelo SIMREDE	217

Tab. 5.10	Relação dos módulos componentes do Gerenciador do Programa Escalador	219
Tab. 5.11	Relação dos módulos componentes do ESCEXT	222
Tab. 5.12	Relação dos módulos que compõem o GEREDE	232
Tab. 5.13	Relação dos módulos que constituem o MSEQ/PAR	236
Tab. 5.14	Relação dos submódulos auxiliares do MSEQ/PAR	236
Tab. 6.1	Relações de precedência das tarefas da figura 4.7	247
Tab. 6.2	Exemplo de entrada de dados dos tempos de processamento das tarefas	248
Tab. 6.3	Forma de representação e dados descritores de cada tipo de evento	250
Tab. 6.4	Exemplo de inserção dos dados da matriz MNNI	260
Tab. 6.5	Valores de tempo coletados no primeiro teste	266
Tab. 6.6	Valores de tempo coletados no segundo teste	270
Tab. 6.7	Valores de tempo coletados no terceiro teste	274
Tab. 6.8	Valores de tempo coletados no quarto teste	277
Tab. 6.9	Valores de tempo coletados no quinto teste	283
Tab. 6.10	Compilação dos resultados obtidos nos testes de avaliação do SIMREDE	287
Tab. 6.11	Relações de precedência dos vinte processos usados no teste de aceitação do Programa Escalador	293
Tab. 6.12	Tempo de processamento das vinte tarefas usadas no teste de aceitação do Programa Escalador	294
Tab. 6.13	Matriz MNNI relativa ao teste de aceitação do Programa Escalador	300
Tab. 6.14	Tempo real de processamento em quatro Transputers da escalação proposta na figura 6.11	302
Tab. 6.15	Tempo de processamento das tarefas para execução de um novo teste	305
Tab. 6.16	Comprimentos das escalações obtidas no novo teste para os dez algoritmos	308

LISTA DE ABREVIATURAS

- ALGOR: designa o algoritmo heurístico sendo testado no instante atual;
- CP: comprimento da rota crítica de um processo;
- CPAPAR: contador do número de processos atuantes (não concluídos) no regime PAR;
- CPn: comprimento normalizado da rota crítica de um processo;
- CQUAN: contador do número de quanta de tempo consumido por um processo escalado em um Transputer;
- CTA: contador do número de Transputers ativos;
- FC: fator de correção para cálculo do comprimento normalizado da rota crítica de cada tarefa;
- FE: fila de espera dos processos atuantes em regime PAR;
- ICMT: instante de chegada de uma mensagem em trânsito;
- IPPEV: instante previsto do próximo evento;
- IRFE: instante de retorno à fila de espera de um processo desescalado por temporização;
- ITQT: instante de término de um quantum de tempo;
- LESC: lista dos processos escalados em um dado regime de um Transputer;
- LESCOBT: lista que contém a melhor escalação obtida para um Transputer até o presente momento;
- LEV: lista que contém a relação seqüencial dos eventos de um processo;
- LLESC: lista dos regimes escalados em um Transputer. Contém listas do tipo LESC;
- LRICMT: lista de registros que contém elementos do tipo RICMT;
- LRIRFE: lista de registros que contém elementos do tipo RIRFE;
- LRMTAS: lista de registros que contém elementos do tipo RMTAS;

- m: número de Transputers existente na rede para uma determinada escalação;
- MNNI: matriz quadrada (m por m) e simétrica que contém como elementos os valores NNI;
- MPREC: matriz de precedência entre as tarefas com dimensão (n+2 por n+2);
- n: número de processos a serem alocados em uma determinada escalação;
- N: nível de um processo;
- NNI: número de nós intermediários entre dois Transputers quaisquer da rede;
- NPE: número (identificação) do processo escalado ocupando a UCP de um Transputer em um dado instante;
- NPR: número (identificação) do processo a ser reativado por sincronização ou término de temporização;
- NSI: número de tarefas sucessoras imediatas de uma dada tarefa;
- NSM: número seqüencial de mensagens recebidas por um determinado processo receptor;
- NTS: número (identificação) do Transputer simulado;
- PEVA: ponteiro para evento atual da lista LEV;
- P_j: designa o processo j;
- PLREGESC: ponteiro para o regime atual de operação na lista de regimes de operação de um Transputer;
- P_R: denota o processo receptor de uma mensagem;
- P_T: denota o processo transmissor de uma mensagem;
- P_{TEMP}: identificação de um processo desescalado por temporização;
- REG: regime de operação (construção) atuante em um Transputer. Pode assumir os valores SEQ ou PAR;
- RICMT: registro do instante para chegada de mensagens em

trânsito;

- RIRFE: registro do instante para retorno à fila de espera;
- RIRM: registro do instante para recepção da mensagem;
- RMTAS: registro de mensagem transmitida aguardando sincronização;
- RPACM: registro de processo aguardando chegada de mensagem;
- SARD: somatória dos atrasos no tempo causados pelo roteamento (τ_{ROT}) e pela distribuição (τ_{DMINT}) das mensagens nos Transputers origem e destino das mesmas;
- TAPT: tempo acumulado de processamento em um Transputer;
- T_i : designa o Transputer i da rede;
- TPEV: tipo do próximo evento de uma tarefa (recepção ou transmissão de mensagem, temporização ou fim de processamento) previsto de ocorrer em um Transputer;
- T_R : denota o Transputer que contém o processo receptor de uma mensagem;
- TRP: tempo real de processamento, correspondente ao tempo consumido na rede de Transputers desde o início da simulação até o momento atual;
- T_T : denota o Transputer que contém o processo transmissor de uma mensagem;
- T_{TEMP} : identificação de um Transputer que teve um de seus processos desescalados por temporização;
- $VALOCA(P_j)$: vetor de alocação dos processos nos Transputers de dimensão n , que contém elementos do tipo NIT. Para cada processo P_j determina o Transputer T_i em que ele está alocado;
- $VCP(P_j)$: vetor de dimensão $n+2$ que contém elementos do tipo CP para cada processo P_j ;
- $VCPAPAR(T_i)$: vetor de dimensão m que contém elementos do tipo

CPAPAR para cada Transputer T_i da rede;

- $VCP_n(P_j)$: vetor de dimensão n que contém elementos do tipo CP_n para cada processo P_j ;
- $VCQUAN(T_i)$: vetor de dimensão m que contém elementos do tipo $CQUAN$ para cada Transputer da rede;
- $VIPPEV(T_i)$: vetor de dimensão m que contém elementos do tipo $IPPEV$ para cada Transputer T_i da rede;
- $VLEV(P_j)$: vetor de dimensão n que contém elementos do tipo LEV para cada processo P_j ;
- $VLFE(T_i)$: vetor de listas de dimensão m que contém elementos do tipo FE para cada Transputer T_i ;
- $VLLESC(T_i)$: vetor de dimensão m que contém elementos do tipo $LLESC$ para cada Transputer T_i ;
- $VLLESCOBT(T_i)$: vetor de dimensão m que contém elementos do tipo $LESCOBT$ para cada Transputer T_i ;
- $VLRMTAS(P_j)$: vetor de dimensão n que contém elementos do tipo $LRMTAS$ para cada processo P_j ;
- $VN(P_j)$: vetor de dimensão $n+2$ que contém elementos do tipo N para cada processo P_j ;
- $VNPE(T_i)$: vetor de dimensão m composto de elementos do tipo NPE para cada Transputer T_i ;
- $VPEVA(P_j)$: vetor de dimensão n que contém elementos do tipo $PEVA$ para cada processo P_j ;
- $VPLREGESC(T_i)$: vetor de dimensão m que contém elementos do tipo $PLREGESC$ para cada Transputer T_i ;
- $VPRIOR(i)$: vetor de prioridade dentre as tarefas, de dimensão n , gerado com base em algum critério de ordenação e contendo elementos do tipo NIP ;
- $VREG(T_i)$: vetor de dimensão m que contém elementos do tipo REG para cada Transputer T_i ;

- VRPACM(P_j): vetor de dimensão n que contém elementos do tipo RPACM para cada processo P_j ;
- VSARD(T_i): vetor de dimensão m que contém elementos do tipo SARD para cada Transputer T_i da rede;
- VTA(T_i): vetor de dimensão m dos Transputers ativos, que contém para cada Transputer T_i um valor verdadeiro se o Transputer ainda está ativo ou falso se o Transputer já encerrou seu processamento;
- VTAPT(T_i): vetor de dimensão m que contém elementos do tipo TAPT para cada Transputer T_i da rede;
- VTPEV(T_i): vetor de dimensão m que contém elementos do tipo TPEV para cada Transputer T_i da rede;
- $V\tau_{PAR}(P_j)$: vetor de dimensão n que contém elementos do tipo τ_{PAR} para cada processo P_j ;
- $V\tau_{PEV,PAR}(P_j)$: vetor de dimensão n que contém elementos do tipo $\tau_{PEV,PAR}$ para cada processo P_j ;
- $V\tau_{SEQ}(P_j)$: vetor de dimensão n que contém elementos do tipo τ_{SEQ} para cada processo P_j ;
- WCR: comprimento da rota crítica do grafo de precedência das tarefas;
- WESC: comprimento resultante da escalação representada por VLLESC. Corresponde ao valor final de TRP obtido pelo GEREDE ao finalizar a simulação de uma escalação;
- WESCOBT: comprimento da melhor escalação até então obtida;
- W_R : comprimento real da escalação em μs obtido nos testes de avaliação do SIMREDE;
- W_S : comprimento simulado da escalação em μs obtido nos testes de avaliação do SIMREDE;
- τ_{DMEXT} : intervalo de duração da transferência de uma mensagem entre dois Transputers diretamente interligados;

- τ_{DMINT} : intervalo de duração da transferência de uma mensagem internamente em um Transputer. Corresponde ao tempo dispendido pelo processo Distribuidor para encaminhar uma mensagem dentro do Transputer destino da mesma;
- $\tau_{EV,PAR}$: intervalo de tempo compreendido entre dois eventos desescaladores consecutivos do processo P_j , apontados por PEVA e PEVA-1, aplicável ao regime PAR;
- $\tau_{EV,SEQ}$: intervalo de tempo compreendido entre dois eventos desescaladores consecutivos do processo P_j , apontados por PEVA e PEVA-1, aplicável ao regime SEQ;
- τ_{PAR} : tempo de processamento de uma tarefa na construção PAR;
- $\tau_{PEV,PAR}$: intervalo de tempo compreendido entre o evento atual e o próximo evento, aplicável ao regime PAR;
- τ_{ROT} : tempo dispendido pelo processo Roteador em cada nó para direcionar o trajeto de mensagens; e
- τ_{SEQ} : tempo de processamento de uma tarefa na construção SEQ.

RESUMO

Em certas aplicações científicas, a redução no tempo de processamento de programas é fundamental. Na área de processamento paralelo e, em particular, em multiprocessamento, busca-se minimizar esse tempo através de algoritmos que provêem escalação estática de tarefas em processadores. Esses algoritmos podem ser exatos, significando que o tempo de processamento resultante é o mínimo para uma dada arquitetura de "hardware" ou aproximados, indicando que a solução obtida é sub-ótima.

Este trabalho apresenta uma biblioteca de algoritmos heurísticos que fornecem escalações sub-ótimas e um modelo que simula a operação de qualquer rede de Transputers, os quais constituem a ferramenta computacional aqui denominada Programa Escalador. Cada escalação gerada é testada na rede simulada e seu tempo de processamento é avaliado. Após ter verificado o desempenho de todos os algoritmos heurísticos disponíveis, o Programa Escalador fornece como resposta a melhor escalação testada e seu tempo simulado de processamento. O usuário pode então alocar as tarefas nos Transputers, obedecendo a ordem e o regime de operação (SEQ ou PAR) que lhe foram fornecidos.

O Programa Escalador foi testado com diversos arranjos de tarefas parcialmente ordenadas em diferentes arquiteturas de Transputers, para verificar a conformidade do modelo da rede de processadores com a realidade. Os resultados dos testes atenderam plenamente aos requisitos de desempenho, apresentando desvios máximos em torno de 1%.

ABSTRACT

In certain scientific applications the reduction in the processing time is fundamental. In the parallel processing area and, particularly, in multiprocessing, the minimization of this time through algorithms that provide static scheduling of tasks in processors is aimed. These algorithms can be exact, meaning that the resulting processing time is minimum for a certain hardware architecture or approximate, indicating that the solution obtained is sub-optimal.

This work presents a library of heuristic algorithms that provide sub-optimal scheduling and a model that simulates the operation of any Transputers network. Together they constitute the computational tool here nominated Scheduling Program. Each schedule created is tested in the simulated network and its processing time is evaluated. After verifying the performance of every available heuristic algorithms, the Scheduling Program provides as response the best tested schedule and its simulated processing time. Then the user can allocate the tasks on the Transputers, following the order and the construction (SEQ or PAR) provided.

The Scheduling Program was tested on several partially ordered tasks arrangements in different Transputers architectures to check the conformity of the processors network model with reality. The results of the tests complied completely to the performance requirements, presenting a maximum deviation of about 1%.

1 - INTRODUÇÃO

Sistemas de processamento paralelo MIMD (Multiple Instruction stream, Multiple Data stream) ou sistemas multiprocessados têm sido empregados em uma extensa faixa de aplicações, não somente no campo do processamento de informações, mas também para o controle de robôs e para a simulação em tempo real de sistemas dinâmicos, devido a seu excelente desempenho e custos relativamente baixos (KASAHARA; NARITA, 1984b).

A geração de código para multiprocessadores envolve diversas operações que não possuem correspondência com a geração de código para monoprocessadores. Em particular, o programa deve ser particionado em módulos escaláveis, os quais devem ser atribuídos a processadores específicos. Esses passos devem ser executados na ordem apresentada acima.

Para tornar eficiente o uso de multiprocessadores, é desejável obter um tempo mínimo de execução com um número mínimo de processadores paralelos. Para tanto, um algoritmo eficiente deve ser desenvolvido, que permita alocar um conjunto de tarefas aos processadores e determinar a ordem ou sequência de execução das tarefas em cada processador. Esse tipo de problema é conhecido como problema de escalação de tarefas em multiprocessadores para obter o mínimo tempo de execução (ou comprimento de escalação), tratando-se, fundamentalmente, de um problema de otimização combinatorial, para o qual se busca um mínimo.

O problema de escalação de tarefas em multiprocessadores é extremamente difícil de se resolver e é geralmente intratável, recaindo na classe de problemas NP-completos. A dificuldade da

solução varia com a inclusão ou exclusão de preempção, com o número de processadores em paralelo e com atributos das tarefas, tais como: a topologia do grafo de tarefas e a uniformidade dos tempos de processamento das mesmas (KASAHARA; NARITA, 1984b).

As opções para particionamento e escalação dependem do programa a ser paralelizado, do processador utilizado e da arquitetura de hardware empregada (POLYCHRONOPOULOS; KUCK, 1987), (TERRANO et al., 1989).

No presente trabalho utilizou-se o Transputer para a realização da escalação das tarefas. Esse processador combina uma arquitetura RISC com memória RAM embutida na pastilha, quatro vias de comunicação para operação em multiprocessamento e, no caso do modelo T800, uma unidade de ponto flutuante de alto desempenho. O Transputer já implementa internamente a solução para o problema de comunicação/sincronismo interprocessadores. Ele provê um ambiente de hardware para implementar sistemas multiprocessados em tempo real. Seus atributos de alta capacidade computacional e modularidade tornam possível a implementação desse tipo de sistemas. São disponíveis facilidades de comunicação sincronizada, interrupções externas e por tempo, temporização, atribuição de prioridades a processos, etc (HULL; ZAREA-ALIABADI, 1989).

1.1 Objetivo do Trabalho

O intuito deste trabalho é criar uma ferramenta auxiliar no desenvolvimento de software multiprocessado para sistemas em tempo real, com ênfase para sistemas em que o conjunto e a

seqüência de execução das tarefas sejam bem definidos, como ocorre, por exemplo, em simuladores de vôo, de processos industriais, de centrais nucleares, etc. Para tanto, criou-se um algoritmo para escalção estática sub-ótima de tarefas em redes de Transputers, denominado **Programa Escalador**, que visa obter o menor tempo total de processamento do conjunto de tarefas. A função do Programa Escalador é prover a alocação dos módulos do programa nos processadores, orientando a configuração em tempo de compilação executada pelo Occam nos Transputers.

A complexidade do problema de escalção tem conduzido a comunidade de Informática a adotar enfoques heurísticos para cada novo processador. O algoritmo a ser desenvolvido neste trabalho empregará métodos heurísticos para encontrar a escalção desejada.

O método aqui apresentado difere dos métodos convencionais de escalção pelas características intrínsecas dos Transputers, como por exemplo a possibilidade da tarefa que está aguardando para se comunicar (tanto receber quanto enviar dados) ser "desescalada" enquanto espera a sincronização, dando lugar a outra tarefa.

O problema de escalção tratado neste trabalho, onde as relações de precedência são do tipo geral (qualquer arco transitivo é permitido), os tempos de processamento das tarefas são diferentes, o número de processadores paralelos é arbitrário e o tempo de transferência de dados entre processadores é contabilizado, é um dos problemas mais intratáveis que se conhece (KASAHARA; NARITA, 1984b). Aliado a essa dificuldade, deve-se acrescentar que se trata de desenvolver um algoritmo escalador que deve considerar as ações do Escalador Interno dos Transputers.

1.2 Motivação para a Realização deste Trabalho

O motivo que levou à elaboração deste trabalho, foi a necessidade encontrada, na prática, de escalar tarefas parcialmente ordenadas, com tempos arbitrários de execução, em redes de Transputers (GARCIA; MARTUCCI, 1990). No caso que originou este estudo, o conjunto dessas tarefas constitui um sistema de simulação dinâmica em tempo real de uma planta de processo (BARROSO; GARCIA, 1988).

A literatura sobre escalção é extensa, mas a grande maioria dos casos resolvidos se referem apenas a sistemas multiprocessados fortemente acoplados, não havendo nenhuma referência à escalção estática em redes de Transputers, que se enquadram como sistemas fracamente acoplados. Um sistema fracamente acoplado apresenta atrasos bem definidos na troca de mensagens entre processos residentes em processadores distintos.

Premido pela necessidade de resolver um problema e perante a inexistência de trabalhos publicados sobre o assunto, optou-se por criar uma ferramenta própria para resolver o problema.

1.3 Metodologia de Trabalho Empregada

Apresenta-se nesta seção a metodologia utilizada para gerar software passível de paralelização e o método científico adotado neste trabalho.

1.3.1 Metodologia para Geração de Software Paralelizável

Procede-se a uma análise da metodologia para criar arquiteturas de software passíveis de serem implementadas em multiprocessadores fracamente acoplados (STEIN, 1988), buscando enquadrar este trabalho dentro de um projeto mais amplo de software, envolvendo multiprocessamento.

O primeiro passo para criar tal arquitetura é identificar os processos componentes do sistema. Mais importante do que conhecer quais são os processos do sistema, é saber como eles interagem. A determinação das interfaces entre os processos é o segundo passo nesta metodologia. As interfaces interprocessos definem o formato preciso para a troca de informações. Se as entradas e saídas estão definidas, os processos estão isolados uns dos outros e a descrição lógica da concorrência do sistema está completa. Uma definição concisa das interfaces entre os processos comunicantes é essencial para executar a transformação de concorrência lógica para concorrência física, a qual é o ponto crucial no desenvolvimento de sistemas multiprocessados. Estes dois passos podem ser aglutinados em uma atividade denominada particionamento de programas.

O terceiro passo da metodologia consiste em escalar as tarefas. O quarto e último passo, necessário para completar a transformação lógica para física, é prover a configuração dos processos, isto é, sua efetiva alocação nos processadores, realizando assim a escalação. Tendo feito isso, conclui-se o trabalho de converter um sistema com processos concorrentes em um software implementado em processadores paralelos.

Neste trabalho, parte-se do princípio que o particionamento do programa já tenha sido efetuado, estando disponíveis as tarefas já fracionadas e as suas interrelações de dependência definidas através dos dados de entrada.

1.3.2 Método Científico Utilizado

A seqüência de atividades desenvolvidas para executar este trabalho iniciou-se com uma ampla busca na literatura correlata existente, para pesquisar o que havia de disponível sobre escalção de tarefas parcialmente ordenadas em multiprocessadores. Tendo-se concluído essa busca, constatou-se a inexistência de qualquer literatura referente à escalção estática de tarefas em redes de Transputers. Procurou-se então agrupar os métodos de escalção existentes, chegando-se aos resultados expostos no capítulo 2.

Ao avaliar os diversos métodos de solução disponíveis, optou-se pelo que apresentava maior flexibilidade de utilização e melhores condições de atender os quesitos de projeto. Tendo-se escolhido o método, partiu-se para o projeto e a implementação do algoritmo, baseado no modelo do modo de operação em multiprocessamento dos Transputers, que foi criado e validado no decorrer deste trabalho. Tendo-se concluído o algoritmo, partiu-se para a avaliação de seu desempenho, criando-se um conjunto de casos, compostos de sistemas de tarefas parcialmente ordenadas com tempos arbitrários de execução.

1.4 Abrangência do Trabalho

O Programa Escalador é aplicável nos seguintes casos:

- qualquer número de tarefas;
- tarefas ordenadas por qualquer grafo dirigido acíclico;
- tarefas com quaisquer tempos individuais de processamento; e
- ambiente de hardware composto de uma rede qualquer de Transputers do mesmo tipo.

1.5 Organização do Trabalho

O trabalho foi dividido em duas partes: na primeira apresenta-se o embasamento teórico a respeito da escalação de tarefas/redes de Transputers e na segunda o desenvolvimento do projeto do Programa Escalador.

A estrutura global do trabalho é delineada a seguir, descrevendo-se sucintamente o que cada capítulo apresenta.

a) Parte I - Embasamento teórico sobre escalação de tarefas/redes de Transputers

Visa fornecer os subsídios técnicos necessários para o desenvolvimento do projeto, sendo composta dos seguintes capítulos:

- Capítulo 2: apresenta definições e conceitos necessários à compreensão do que é um problema de escalação, cita os métodos de solução normalmente empregados para resolver esse tipo de problema e mostra uma visão histórica dos estudos sobre escalação, através da menção dos principais

trabalhos publicados sobre o assunto e da evolução dos métodos de solução; e

- Capítulo 3: são introduzidos conceitos sobre Transputers/ Occam, necessários à criação do modelo do modo de operação dos Transputers em multiprocessamento.

b) Parte II - Desenvolvimento e testes do Programa Escalador

Tem por objetivo apresentar o projeto conceitual, básico e detalhado do Programa Escalador e os testes a que o mesmo foi submetido, sendo composta dos seguintes capítulos:

- Capítulo 4: enuncia-se o problema a ser resolvido, definem-se os objetivos a serem alcançados, estipulam-se as restrições impostas à solução do problema, estabelecem-se os critérios para avaliar a eficiência do método proposto e, por fim, concebe-se o método de solução;
- Capítulo 5: desenvolvem-se os projetos básico e detalhado do Programa Escalador; e
- Capítulo 6: comentam-se alguns detalhes da implementação do Programa Escalador, apresenta-se a forma de entrada e saída dos dados no mesmo e discutem-se os testes realizados, os resultados obtidos, as avaliações efetuadas e as conclusões obtidas.

Por fim, no capítulo 7, efetua-se uma análise crítica do trabalho, enfoca-se qual é a sua colaboração dentro do campo de escalação estática de tarefas em multiprocessadores e delinea-se um plano de pesquisas futuras baseado nele.

2 - EMBASAMENTO TEÓRICO SOBRE ESCALAÇÃO DE TAREFAS

Este capítulo define o problema de escalação, apresenta conceitos e terminologia sobre o assunto, relaciona métodos típicos de solução de problemas de escalação e traça a evolução histórica dos métodos de solução desses problemas.

2.1 Geração de Programas Multiprocessados

Há muitas características peculiares a sistemas multiprocessados, que não aparecem em um contexto seqüencial, conforme é mostrado em (BERTSEKAS; TSITSITLIS, 1989). Uma característica marcante é o **particionamento** e a **escalação** de tarefas, que constituem os passos necessários para gerar um software passível de implementação em multiprocessadores, conforme a metodologia proposta no item 1.3.1.

2.1.1 Particionamento de Programas

O particionamento (partitioning) ou empacotamento (packaging) de tarefas é responsável pelo fracionamento do código (e/ou estruturas de dados) de um programa em módulos identificáveis, que são tratados como unidades na escalação, além de definir um sumário de dependências de dados entre tarefas. Assumindo que somente programação estruturada é permitida, o resultado do particionamento é um grafo dirigido acíclico. O tamanho dos módulos resultantes do particionamento está associado com o conceito de granularidade dos mesmos.

Uma diretriz geral para o particionamento é baseada no conceito de equilíbrio computação/comunicação. Um dos motivos para considerar a comunicação no projeto de programas paralelos, é que os tempos de comunicação podem ser maiores que os tempos de computação. Dessa forma, a maneira com que os algoritmos são particionados define o volume de comunicação intermódulos, que por sua vez afeta o tempo de término do processamento das tarefas (HWANG; BRIGGS, 1988).

A granularidade exerce uma influência direta na escalação, pois uma vez que o sistema de tarefas (vide sub-item 2.2.2.2) foi fracionado, deve-se conviver com os módulos resultantes, qualquer que seja o hardware a ser utilizado. Tarefas muito pequenas não são favoráveis, do ponto de vista dos requisitos de transferência de dados, porque há a possibilidade de distribuir um conjunto fortemente acoplado de tarefas, aumentando muito a comunicação interprocessadores. Tarefas muito grandes, por outro lado, podem ocasionar um decréscimo no grau de paralelismo e a melhora da resposta do sistema paralelo pode não ser alcançada (GAJSKI; PEIR, 1985), (KASAHARA; NARITA, 1984b), (MCCREARY; GILL, 1989).

2.1.2 Escalação de Tarefas

Uma vez concluído o particionamento do programa, executa-se a escalação das tarefas, a qual se refere à alocação e ao seqüenciamento das mesmas nos processadores, visando satisfazer uma ou mais funções-objetivo.

Na seção 2.2 aprofunda-se o estudo a respeito da escalação de tarefas.

2.2 Problema Geral de Escalação

Um problema de escalação consiste em duas partes: um **modelo** e uma **função-objetivo**. O modelo descreve o sistema, incluindo os tipos de tarefas e as restrições em seu processamento, os tipos de máquinas e seu número e todas as outras propriedades necessárias para especificar uma escalação. A função-objetivo corresponde ao critério de avaliação de desempenho especificado, atribuindo um "valor" a cada escalação exeqüível. Dado um problema I, o modelo indica quais são as escalações exeqüíveis para I e a função-objetivo indica quais são seus valores (GAREY et al., 1978).

O objetivo das escalações é otimizar um ou mais dos critérios de avaliação de desempenho. A motivação para esse objetivo é que em muitas situações, uma escalação fraca pode conduzir a uma resposta inaceitável em termos de requisitos de tempo ou a uma utilização inaceitável de recursos.

Esta seção aborda o modelo e as funções-objetivo normalmente empregados em problemas de escalação. Antes disso, no entanto, serão expostos conceitos aplicáveis à área de escalação de tarefas.

2.2.1 Definição de Termos e Estabelecimento de Conceitos Aplicáveis a Problemas de Escalação

Ao longo deste trabalho serão utilizados termos e conceitos próprios da área de escalação de tarefas, cuja terminologia é apresentada a seguir.

2.2.1.1 Definição de Termos

Os termos escalacão (scheduling), atribuição (assignment), alocação (allocation), mapeamento (mapping), seqüenciamento (sequencing) e configuração (configuration) são muito empregados em textos sobre escalacão de tarefas, cabendo aqui definí-los, da forma com que normalmente são encontrados na literatura correlata. O conceito de escalacão foi enunciado de diferentes formas pelos autores: "escalacão é a descrição do trabalho feito por cada processador como uma função do tempo" (MUNTZ; COFFMAN, 1969); ou "escalar tarefas corresponde a atribuir cada operação a uma posição específica na escala temporal de cada máquina" (CONWAY et al., 1967) ou ainda, "escalacão é a tarefa de atribuir cada operação a uma determinada máquina em uma seqüência estabelecida" (CONWAY et al., 1967).

Os termos atribuição, alocação e mapeamento são usados indistintamente para designar a distribuição das tarefas nos processadores (MUNTZ; COFFMAN, 1970). Define-se seqüenciamento como sendo o ordenamento de operações dentro de uma única máquina (CONWAY et al., 1967). Por fim, a operação de efetivamente alocar os módulos aos processadores é designada por configuração.

O termo escalacão corresponde à associação dos conceitos introduzidos pelos termos atribuição, seqüenciamento e configuração (MUNTZ; COFFMAN, 1970). Utilizam-se aqui os termos máquina e operação, visto que as definições mencionadas anteriormente foram criadas na década de 60, quando quase toda a literatura de escalacão utilizava a terminologia da manufatura. Ao longo deste trabalho, os termos máquina e processador, bem como operação, tarefa, processo e módulo serão utilizados

equivalentemente. O termo sistema de tarefas ou "job" (vide sub-item 2.2.2.2) será utilizado para designar um conjunto de tarefas que realiza alguma função bem definida.

Problemas de escalação em máquinas podem ser genericamente formulados como segue:

"Suponha que n tarefas interdependentes P_j ($j=1, \dots, n$), com tempos arbitrários de execução, devam ser processadas em m máquinas M_i ($i=1, \dots, m$). Assume-se que cada máquina pode processar no máximo uma tarefa por vez e que cada tarefa pode ser processada no máximo em uma máquina por vez. Qual, de acordo com algum critério selecionado de avaliação de desempenho, é a melhor alocação das tarefas nas máquinas e a seqüência ótima de processamento das tarefas em cada máquina?"

A escalação em multiprocessadores visa atribuir e seqüenciar n tarefas em m processadores paralelos, que podem ser idênticos, uniformes (idênticos em capacidade funcional mas diferentes em velocidade) ou distintos (diferentes tanto em função quanto em velocidade) (GRAHAM et al., 1979).

2.2.1.2 Estabelecimento de Conceitos e Definições

Definem-se, a seguir, conceitos e definições utilizados ao longo deste trabalho.

a) Configuração em tempo de compilação, carga e execução

Um programa paralelo pode ser configurado em qualquer um de seus estágios de operação: compilação, carga ou execução (POUNTAIN, 1990).

A configuração em tempo de compilação implica que se deva conhecer a topologia exata do hardware quando o programa é compilado. A linguagem Occam utiliza esse tipo de configuração, visto que as instruções de alocação dos módulos nos Transputers são disponíveis na própria linguagem e se tornam parte do programa.

A configuração em tempo de carga significa que se pode executar o mesmo código compilado em diferentes topologias, suprindo informações sobre elas ao carregador do programa antes de cada execução do mesmo. O carregador então dirige os módulos do programa aos processadores corretos.

A configuração em tempo de execução implica que enquanto o programa está sendo processado, ele pode se adaptar à topologia do hardware ou então que seus módulos podem ser realocados.

A configuração em tempo de compilação é a menos flexível, mas oferece a maior eficiência, pois otimizações podem ser efetuadas com pleno conhecimento do hardware. A configuração em tempo de execução apresenta maior flexibilidade mas impõe sobrecargas, pois consome mais memória e aumenta o tempo de processamento. A configuração em tempo de carga representa um compromisso entre esses dois extremos.

b) Escalação estática e dinâmica

A maneira com que as tarefas chegam para ser processadas provê uma distinção entre escalação estática e dinâmica (CONWAY et al., 1967). No caso estático, um certo número de tarefas chega simultaneamente e as máquinas estão todas livres e disponíveis

para trabalho imediato. Nenhuma tarefa adicional chegará, de modo que toda atenção pode ser focalizada em escalar esse conjunto de tarefas completamente conhecido e disponível. No caso dinâmico, a chegada das tarefas é um processo contínuo e intermitente.

Na escalação estática, os processos são alocados aos processadores antes da execução ser iniciada, em tempo de compilação ou em tempo de carga, baseado no conhecimento de informações globais a respeito do programa. Quando o programa inicia a execução, cada processador sabe exatamente quais tarefas irá executar (POLYCHRONOPOULOS; KUCK, 1987).

A escalação estática é normalmente aplicada nos casos em que um programa deva ser executado muitas vezes, como por exemplo, no caso de se ter um multiprocessador operando com um conjunto de tarefas bem definido, como ocorre em sistemas de tempo real. Nesse caso, a menos de interrupções assíncronas, conhece-se bem o encadeamento lógico das tarefas, além do tempo de processamento das mesmas. Torna-se então necessário atribuir os processos a processadores e seqüenciá-los de uma forma tal, que toda a potencialidade do sistema multiprocessado seja utilizada, visando otimizar a medição do desempenho da escalação.

Na escalação dinâmica as tarefas são atribuídas aos processadores durante a execução do programa. Assim, para minimizar o tempo dispendido em tempo de processamento, os enfoques dinâmicos devem ser menos sofisticados (POLYCHRONOPOULOS, 1988).

A escalação dinâmica é mais apropriada para multiprocessadores com memória compartilhada, de forma que o código do programa não necessite ser transferido ao processador encarregado de executá-lo. No caso de multiprocessadores com memória distribuída, é necessário ter uma cópia do código de cada tarefa e seu respectivo segmento de dados em cada processador, utilizando o sistema de transferência de mensagens para distribuir palavras de controle e dados, o que complica a utilização de escalção dinâmica (LEPPALA, 1987).

c) Tempo médio e máximo de processamento de uma tarefa

Com respeito aos tempos de execução das tarefas, assumidos como conhecidos na escalção estática, há duas interpretações que valorizam a utilidade dos resultados obtidos, quando esses tempos não são conhecidos exatamente (MUNTZ; COFFMAN, 1970).

Primeiro, os tempos de execução podem ser encarados como tempos máximos de processamento. Nesse caso, obtem-se o tempo máximo para completar o conjunto de tarefas. Segundo, os tempos de execução podem ser interpretados como médias dos tempos de processamento, considerados como variáveis aleatórias. Com essa interpretação, o tempo total obtido é uma estimativa (otimista) do tempo médio de computação das tarefas.

d) Disciplina preemptiva e não-preemptiva

No modo de operação não-preemptivo, uma vez que um processador é designado para uma tarefa, ele deve executá-la até o fim, sem ser interrompido. Isto significa que apenas um único intervalo de tempo τ deve ser atribuído a cada tarefa, onde τ

equivale ao tempo de processamento da tarefa. Esse procedimento para atribuição de processos a processadores é designado por escalção básica.

O caso oposto, em que se permite interromper qualquer processador em qualquer instante e redesigná-lo para outra tarefa é denominado escalção preemptiva (MUNTZ; COFFMAN, 1969). Com disciplinas preemptivas, nunca é benéfico introduzir períodos ociosos nos processadores, quando há tarefas disponíveis para execução. Esse não é o caso com disciplinas não-preemptivas (COFFMAN; DENNING, 1973).

e) Atribuição fixa e flexível

Dentro da escalção estática, pode-se ter a atribuição a processadores autônomos do tipo fixa, em que cada tarefa pode ser realizada apenas no processador para o qual foi designada, ou então pode-se ter a atribuição do tipo flexível, em que cada tarefa pode ser realizada em qualquer um dos processadores (assumidos como idênticos), dependendo apenas de sua disponibilidade (SCHWARTZ, 1964).

f) Escalção determinística e estocástica

Na escalção determinística assumem-se os tempos de processamento como sendo fixos e conhecidos de antemão. Na escalção estocástica, os tempos de execução são variáveis aleatórias. É comum não se conseguir determinar o tempo exato de execução de uma tarefa computacional devido aos desvios condicionais e às malhas de repetição (loops) nela contidos.

Pode-se derivar uma escalação determinística de uma estocástica, assumindo-se que a duração de uma tarefa é constante e igual a seu tempo médio de execução (ADAM et al., 1974). Para estimar o tempo médio de execução de cada tarefa, deve-se medir os tempos ao processá-la diversas vezes com diferentes conjuntos de dados de entrada.

2.2.2 Descrição do Modelo Básico a ser Empregado

O modelo geral a ser estudado assume um conjunto de tarefas e um conjunto de recursos a ser utilizado na execução das mesmas. O modelo é determinístico, no sentido de que as informações descrevendo as tarefas são assumidas como conhecidas "a priori". Essas informações incluem tempos de execução das tarefas e relações operacionais de precedência.

O modelo de escalação, a partir de onde os estudos subseqüentes serão desenvolvidos, é descrito considerando (COFFMAN, 1976):

- recursos;
- sistemas de tarefas; e
- restrições impostas.

2.2.2.1 Recursos

Na maioria dos modelos de escalação estudados, os recursos consistem simplesmente em um conjunto de processadores $T = \{T_1, \dots, T_m\}$. Dependendo do problema, eles podem ser idênticos, uniformes ou distintos.

Nos modelos mais gerais, há também um conjunto de tipos de recursos adicionais $R=\{R_1, \dots, R_S\}$, requeridos durante a execução de uma tarefa em algum processador. Em aplicações computacionais, tais recursos podem representar, por exemplo, armazenamento em memória principal ou de massa, dispositivos de entrada/saída, bibliotecas de subrotinas, etc. A maior parte dos trabalhos publicados sobre escalção assume que recursos adicionais suficientes são disponíveis para a execução das tarefas, analisando apenas a distribuição das tarefas nos processadores.

Para caracterizar o tempo de transferência de unidades de mensagens de um processador T_i a um processador T_j , é introduzido o parâmetro $\mu(T_i, T_j)$. Variando os valores de $\mu(T_i, T_j)$, o modelo pode ser usado para simular diversos tipos de sistemas, tais como sistemas totalmente conectados, redes locais ou hipercubos. Para acomodar a abordagem de escalção determinística, assume-se adicionalmente que o subsistema de comunicação é livre de contenção (HWANG et al., 1989).

Um processador, neste contexto, é um dispositivo capaz de executar o que for necessário para realizar uma tarefa. Abstratamente, um processador equivale a uma escala temporal $(0, t)$ com certos intervalos disponíveis (CONWAY et al., 1967).

As seguintes condições se aplicam aos processadores:

- os processadores são funcionalmente indivisíveis, de forma que cada processador pode manipular no máximo uma tarefa por vez;
- o número de processadores disponíveis é constante, não se considerando problemas de falhas ou manutenção; e

- não há tempo de preparação (set-up time) nos processadores, significando que um processador não gasta tempo se preparando para iniciar uma nova tarefa.

2.2.2.2 Sistemas de Tarefas

Um conjunto de processos relacionados é conhecido como um sistema de tarefas ou um "job" (HWANG; BRIGGS, 1988).

Um sistema geral de tarefas para um dado conjunto de recursos pode ser definido como o sistema $(P, <, [\tau_{ij}], \{R_j\}, \{F_j\})$ da seguinte forma (COFFMAN, 1976):

- $P = \{P_1, \dots, P_n\}$ é um conjunto de tarefas a ser executado. A tarefa é a unidade básica do processo de escalação. Entende-se tarefa como sendo uma operação elementar a ser realizada;
- $<$ é um ordenamento parcial (irreflexivo) definido em P , que especifica restrições operacionais de precedência entre as tarefas, isto é, $P_i < P_j$ (lê-se P_i precede P_j) significa que P_i deve ser completado antes que P_j possa ser iniciado.
Uma vez definidas as restrições de precedência entre tarefas, costuma-se expressá-las graficamente, podendo surgir três tipos de grafos, em função do grau de interrelacionamento entre as tarefas:
 - grafo dirigido acíclico arbitrário;
 - grafo em forma de árvore; e
 - grafo nenhum, quando as tarefas são independentes;
- $[\tau_{ij}]$ é uma matriz m por n dos tempos de execução das tarefas, onde $\tau_{ij} > 0$ é o tempo requerido para executar P_j , $1 \leq j \leq n$, no processador T_i , $1 \leq i \leq m$. Supõe-se que se $\tau_{ij} = \infty$, significa que P_j

não pode ser executado em T_i e que para cada j existe pelo menos um i , tal que $\tau_{ij} < \infty$. Quando todos os processadores são idênticos, τ_j denota o tempo de execução de P_j comum a todos os processadores;

- $R_j = [R_1(P_j), \dots, R_s(P_j)]$, $1 \leq j \leq n$, especifica o volume de recursos do tipo R_i , $1 \leq i \leq s$, requerido para a execução de P_j ; e
- os pesos F_j , $1 \leq j \leq n$, são interpretados como taxas de custos, que em geral podem ser funções arbitrárias das propriedades da escalação influenciando P_j . Por exemplo, assumindo-se F_j como constante, o custo de terminar P_j no tempo t é dado por $F_j * t$.

O modelo apresentado anteriormente foi proposto por COFFMAN (1976) e não considera a sobrecarga devida à comunicação interprocessadores. O modelo expandido proposto por HWANG et al. (1989) considera o relacionamento temporal entre duas tarefas, dado por um número inteiro positivo $\cap(P_i, P_j)$, que corresponde ao número de mensagens enviadas de um predecessor imediato P_i a um sucessor imediato P_j , após o término do processamento de P_i . O grafo passa a ser denominado grafo dirigido acíclico arbitrário expandido (HWANG et al., 1989).

O sistema de tarefas passa a ser designado pela sêxtupla $(P, <, [\tau_{ij}], [\cap_{ij}], \{R_j\}, \{F_j\})$, onde $[\cap_{ij}]$ é uma matriz n por n do número de mensagens enviadas da tarefa i para a tarefa j .

O tempo para levar $\cap(P_i, P_j)$ unidades de mensagens do processador T_k a T_l é $\mu(T_k, T_l) * \cap(P_i, P_j)$, um valor determinístico. Assumir a inexistência de contenção na comunicação é indispensável para manter o enfoque determinístico, mas, essa premissa é válida somente em sistemas plenamente conectados ou sistemas com protocolos livres de contenção (HWANG et al., 1989).

As seguintes condições se aplicam às tarefas utilizadas neste modelo genérico:

- elas são indivisíveis, de modo que no máximo um processador pode estar executando uma dada tarefa em qualquer instante; e
- não há ociosidade plena do sistema de processadores, significando não se permitir que todos os processadores estejam ociosos quando tarefas incompletas existem, isto é, não há nenhum intervalo de tempo durante o processamento das tarefas em que nenhum processador esteja executando nenhuma tarefa.

Deve-se notar que o ordenamento parcial $<$ é convenientemente representado por um grafo acíclico dirigido, sem arcos transitivos (redundantes), que corresponde ao grafo de fluxo de dados, o qual simboliza a estrutura de dependência de dados de um programa (McCREARY; GILL, 1989). Assume-se que $<$ é dado como uma lista de arcos em tal grafo.

Entretanto, nos problemas práticos de escalação, utilizam-se métodos gráficos para definir sistemas de tarefas, ao invés de definí-los como sêxtuplas. O relacionamento entre tarefas é normalmente mostrado por um grafo de precedência.

a) Grafos de precedência

Um grafo consiste em um conjunto de nós (vértices) e um conjunto de arcos que conectam os nós.

O nome dos arcos nesse conjunto é designado através do par ordenado (X,Y) , onde X e Y correspondem, respectivamente, à denominação do nó de origem e de destino. Para os problemas considerados, é coerente assumir que um grafo não possa conter arcos da forma (X,X) ou mais do que um arco da forma (X,Y) .

Um caminho ou trilha (path) é uma seqüência de arcos distintos $(a_1, a_2, \dots, a_k)$, tal que dois arcos adjacentes na seqüência tenham um nó em comum. Um caminho é dito caminho dirigido ou rota, caso se estipule adicionalmente, que para dois arcos adjacentes na seqüência, o nó terminal do primeiro arco corresponda ao nó inicial do segundo, como por exemplo: $(W, X)(X, Y)(Y, Z)$. O comprimento de um caminho dirigido corresponde ao número de vértices no caminho.

Um nó não-terminal está no nível k se o caminho mais longo dele ao nó terminal do grafo tiver comprimento k e, inversamente, está no co-nível j se o caminho mais longo dele ao nó inicial do grafo tiver comprimento j .

Um grafo dirigido pode ser usado para ilustrar relações de precedência entre nós. Em um grafo de precedência, um arco descreve uma relação de precedência entre um par de nós. O arco (X, Y) significa que o nó X imediatamente precede o nó Y , indicado por $X \ll Y$, de forma que a operação X deve ser completada antes da operação Y ser iniciada. Se há uma trilha dirigida de X a Y , X precede Y , indicado por $X < Y$.

Equivalentemente, se $X < Y$, pode-se também escrever $Y > X$ e dizer que a operação Y sucede a operação X . Diz-se que X é um predecessor de Y e que Y é um sucessor de X . A relação de precedência é transitiva ($X < Y$ e $Y < Z$ implica em $X < Z$) e não-reflexiva ($X < X$ não tem sentido).

A operação X diretamente precede a operação Y (ou Y diretamente sucede X) se $X < Y$ e não há nenhuma operação Z , tal que $X < Z < Y$. Neste caso X é chamado um predecessor imediato de Y e Y um sucessor imediato de X .

Caso as operações X e Y não possuam nenhum interrelacionamento, diz-se que elas são compatíveis (FUJII et al., 1969) ou incomparáveis (GRAHAM, 1972) e se escreve $X \sim Y$. Duas operações compatíveis podem ser processadas simultaneamente.

As relações de precedência podem ser descritas por uma matriz de precedência $MPREC$, quadrada n por n , cujas linhas e colunas contêm a identificação dos vértices e cujos elementos correspondem a variáveis booleanas. Se o nó I precede o nó J , um "1" aparece na i -ésima linha e na j -ésima coluna de M , mas não o contrário, pois se $m_{ij}=1$, então $m_{ji}=0$. Pela propriedade da não-reflexividade, $m_{ii}=m_{jj}=0$ para todos i e j .

A relação de precedência só é definida em um caminho dirigido aberto, não tendo, portanto, sentido em ciclos dirigidos (loops), pois a condição que controla o número de iterações só pode ser resolvida durante o tempo de execução. Requer-se, nesses casos, que o grafo seja alterado para formar apenas caminhos dirigidos abertos (MARTIN; ESTRIN, 1967). A maioria dos trabalhos publicados em escalação de processadores, explícita ou implicitamente ignora as dificuldades apresentadas por ciclos, assumindo que o ciclo inteiro possa estar contido em um único nó do grafo. Os grafos também não devem conter nós condicionais ou de decisão. O resultado de um nó de decisão durante o processamento pode afetar o fluxo de controle em um programa (GONZALEZ, 1977).

Um exemplo de um grafo acíclico dirigido é mostrado na figura 2.1.

Legenda:

- a simbologia utilizada na designação do nó indica tarefa/tempo de execução;
- junto ao arco aparece um número que caracteriza o número de mensagens enviadas de uma tarefa a outra.

Fig. 2.1 Representação de um grafo acíclico dirigido de $(P, <, \{\tau_i\}, \{\cap_{ij}\})$

Propriedades referentes à figura 2.1:

- P_1, P_2, P_3, P_{10} são vértices iniciais;
- P_8, P_9, P_{10} são vértices terminais;
- P_7 é um sucessor de P_1, P_2, P_3, P_4, P_5 mas é um sucessor imediato apenas de P_4 e P_5 ;
- P_5 é um predecessor de P_7, P_8, P_9 mas é um predecessor imediato apenas de P_7 e P_8 ;
- níveis ou comprimento dos caminhos críticos de cada nó:
 $P_1=8$; $P_2=9$; $P_3=8$; $P_4=7$; $P_5=7$; $P_6=3$; $P_7=5$; $P_8=6$; $P_9=2$ e $P_{10}=1$;
- caminhos críticos: P_2, P_5, P_8 e P_2, P_4, P_7, P_9 .

b) Conversão de programas em grafos dirigidos

Pode-se associar as cartas de fluxo (flow charts) e os programas de computadores a grafos, modelando-os como grafos dirigidos e descrevendo suas propriedades em termos de matrizes (SCHWARTZ, 1961).

Um arco implica na existência de uma ou mais dependências de dados entre duas tarefas. O ordenamento parcial ou relação de precedência é definido pela dependência dos dados nas instruções de um programa.

Dessa forma, é natural associar um grafo com cada sistema de tarefas, de tal forma que os nós correspondam a operações, ao passo que os arcos dirigidos que os unem simbolizam as relações de precedência direta definindo o ordenamento parcial entre as tarefas. O custo da comunicação intermódulos entre cada par de módulos é representado pelo peso do arco conectando dois nós. Uma tarefa constitui a unidade de atividade computacional. Uma tarefa é especificada em termos de seu comportamento externo, isto é, suas entradas, saídas e seu tempo de execução. As operações internas de uma tarefa não são de interesse neste contexto. Os tempos de processamento das tarefas são representados pelos pesos dos nós. Assim, em termos da terminologia de grafos, o enfoque deste trabalho se concentra em grafos dirigidos, acíclicos e ponderados, satisfazendo as restrições impostas pelas relações de precedência.

c) Rota ou caminho crítico

Uma tarefa pode ser denominada crítica, se um pequeno acréscimo em seu tempo de processamento ou um atraso em seu início, causar um atraso de mesmo valor no processamento de suas tarefas sucessoras. Uma rota crítica é uma rota em que todas as tarefas representam tarefas críticas.

Sejam S e T as tarefas que denotam, respectivamente, os nós inicial e terminal do grafo de precedência, onde S ou T podem corresponder a pseudo-operações, com tempo nulo de processamento. O problema consiste em encontrar uma rota de S a T, de tal forma que qualquer acréscimo no tempo de processamento de qualquer tarefa contida nessa rota, aumente o tempo de encerramento da operação T.

Dito em outras palavras, encontrar uma rota crítica equivale a localizar a rota ou rotas de S a T, para as quais a soma dos tempos de processamento das tarefas na rota é máxima, significando que o nó S está no nível mais alto do grafo.

Sempre existe pelo menos uma rota crítica e podem haver diversas. O objetivo da análise da rota crítica é identificá-la(s). Durante o processamento do sistema de tarefas, uma atenção especial deve ser dirigida às tarefas das rotas críticas, pois qualquer atraso em seu processamento estenderá o tempo de encerramento do sistema de tarefas.

Define-se o tempo para percorrer a rota crítica como (KASAHARA; NARITA, 1984b):

$$W_{CR} = \max_k \sum_{i \in \phi_k} \tau_i \quad (2.1)$$

onde ϕ_k representa o k-ésimo caminho do nó de entrada ao nó de saída.

W_{CR} é igual ao mínimo tempo possível para executar as tarefas envolvidas em um dado grafo de tarefas em um conjunto de processadores paralelos. Em outras palavras, qualquer que seja o método de escalção empregado, o comprimento da escalção gerado não pode ser menor que W_{CR} .

2.2.2.3 Restrições Impostas

As seguintes restrições podem ser impostas a uma escalção:

- disciplina de interrupção de tarefas (operação preemptiva ou não-preemptiva); e
- possibilidade de alocação variável de uma tarefa (escalção do tipo fixa ou flexível).

2.2.2.4 Imprecisões no Modelo Considerado

O modelo do sistema de tarefas considerado neste item apresenta imprecisões, conforme detalhado a seguir.

- a) Em Multiprocessadores Fortemente Acoplados (POLYCHRONOPOULOS; KUCK, 1987)

Os métodos clássicos de escalção são imprecisos, porque eles consideram uma forma idealizada do problema, onde os tempos de execução das tarefas são fixos e conhecidos de antemão e por ignorarem "efeitos colaterais" (comunicação e sincronização interprocessadores). Na realidade, instruções de desvio em

programas, interferências no acesso à memória, número variável de iterações em instruções de repetição, etc, fazem com que os tempos precisos de execução de tarefas sejam praticamente impossíveis de serem avaliados.

- b) Em Multiprocessadores Fracamente Acoplados (INDURKHYA et al., 1986)

Modela-se o tempo de execução de uma atribuição de tarefas em um sistema fracamente acoplado, através da soma dos tempos de comunicação interprocessadores com o tempo de execução do processador mais ocupado.

Neste modelo, os custos de comunicação são diferentes de zero quando uma tarefa em um processador se comunica com outra tarefa em outro processador. Caso as tarefas estejam no mesmo processador, tais custos são considerados nulos. Portanto, os tempos de comunicação neste modelo refletem o tempo requerido para se comunicar através de um canal, que excedem o tempo requerido para comunicação dentro de um único processador. Esse excesso inclui a sobrecarga (overhead) requerida para iniciar e terminar uma mensagem, bem como atrasos na transmissão devidos à largura de banda finita de um elo de comunicação.

A idéia de somar os tempos de comunicação interprocessadores é precisa, quando todas as comunicações são efetuadas em um elo comum, como um barramento compartilhado ou um anel de comunicação que atua como um barramento. Essa soma não é precisa quando o sistema de comunicação tem elos ponto-a-ponto independentes. Um sistema com diversos elos tem uma largura de banda agregada potencialmente maior, mas em determinados

momentos durante uma computação, pode haver saturação de alguns elos. O modelo proposto trata tal sistema como se todos os elos fossem agregados em um só e como se todas as mensagens fossem transmitidas através desse elo único. Portanto, o modelo apresenta uma visão altamente otimista dos tempos de comunicação, porque não se considera a saturação de elos individuais.

A estimativa para o tempo de processamento das tarefas é uma avaliação otimista, porque o modelo não leva em conta os atrasos atribuídos à sincronização entre elas. Neste modelo, nenhuma tarefa tem que esperar por nenhuma outra. Em realidade, a espera pode ocorrer muito freqüentemente entre um conjunto de tarefas cooperantes.

Finalmente, a estimativa para o tempo de execução simplesmente acrescenta os atrasos da comunicação aos tempos de processamento e não considera situações em que a comunicação ocorre em paralelo com o processamento da tarefa. Daí, essa imprecisão do modelo tende a superestimar os tempos de execução e, portanto, tende a compensar as imprecisões mencionadas anteriormente.

2.2.3 Descrição das Funções-objetivo mais Utilizadas

Especificar uma função-objetivo em um problema de escalação equivale a especificar o critério de avaliação de desempenho (COFFMAN, 1976) a ser utilizado, correspondendo a determinar um conjunto de classes de equivalência de escalações e um ordenamento de preferência entre essas classes. Por exemplo,

especificar o tempo de término como critério de desempenho para um problema específico significa que:

- todas as escalasções que possuem um tempo de término W' são equivalentes, de forma que é indiferente qual delas se escolha; e
- uma escalação com tempo de término W' é preferida a uma outra com tempo de término W'' se e somente se $W' < W''$.

Em quase todos os trabalhos de escalação, critérios de desempenho muito simples são empregados. Dentre eles se destacam os seguintes, em ordem decrescente de utilização (GONZALEZ, 1977), (COFFMAN; DENNING, 1973):

- minimização do tempo de término;
- minimização do número de processadores requeridos;
- minimização do tempo médio de fluxo; e
- cumprimento de datas previstas (due dates) ou datas limite, também chamadas de linhas críticas (deadlines).

a) Minimização do tempo de término

O tempo de término, também denominado tempo total de processamento ou comprimento da escalação, mede o intervalo de tempo decorrido desde o início do processamento da primeira tarefa até a conclusão da última tarefa, estando relacionado com a utilização dos processadores e com as exigências de completar todas as tarefas em tempos bem definidos, típicas de aplicações em tempo real.

A minimização do tempo de término, também chamada escalação de mínimo comprimento, maximiza o "throughput", definido como o número de tarefas processadas por unidade de tempo.

Como o comprimento de uma escalação depende do tempo total de execução das tarefas e do tempo total durante o qual os processadores estão ociosos, resulta que minimizar o comprimento de escalação minimiza o tempo ocioso e, em consequência, conduz à máxima utilização dos processadores.

b) Minimização do número de processadores requeridos

Minimizar o número requerido de processadores pode ser justificado por pelo menos duas razões. A primeira é custo e a segunda é que se o número de processadores requeridos para processar um conjunto de tarefas em um dado tempo é menor que o número total de processadores disponíveis, então os processadores restantes podem ser usados como reserva para obter uma disponibilidade maior e como processadores de "background" para computações não-críticas (GONZALEZ, 1977).

c) Minimização do tempo médio de fluxo

O critério de tempo médio de fluxo, também denominado tempo médio no sistema ou tempo de "turnaround", se relaciona com o desempenho do sistema, quando visto pelo lado dos usuários. O tempo de fluxo de uma tarefa é igual ao tempo em que sua execução é completada. O tempo de fluxo de uma escalação é definido como a soma dos tempos de fluxo de todas as tarefas na escalação. O tempo médio de fluxo é obtido dividindo o tempo de fluxo da escalação pelo número de tarefas.

A principal razão da importância deste critério de desempenho reside em sua conexão com o número médio de tarefas incompletas, pois ele freqüentemente minimiza esse número.

d) Cumprimento de datas previstas

Os critérios vistos anteriormente se referem ao comportamento do conjunto de tarefas como um todo, que corresponde à maioria dos casos estudados. Há casos, no entanto, em que linhas críticas (deadlines) ou tempos de término da escalação são estabelecidos para membros individuais do conjunto de tarefas.

Caso não se permita que cada tarefa ultrapasse uma determinada linha crítica, tem-se a escalação de tempo real dura (hard real-time schedule) ou escalação de linha crítica dura (hard deadline schedule). Escalações com linhas críticas aparecem mais freqüentemente em conexão com escalações de tarefas periódicas (GONZALEZ, 1977).

2.2.4 Dados para Criação e de Entrada/Saída de um Algoritmo Escalador

Este item apresenta as informações básicas necessárias para poder desenvolver e aplicar um algoritmo escalador e quais são as saídas que o mesmo deve fornecer.

2.2.4.1 Informações para Desenvolver um Algoritmo Escalador

O projetista de um algoritmo escalador deve conhecer, no mínimo, os seguintes dados para poder desenvolvê-lo:

- tipo dos processadores (idênticos, uniformes ou distintos), indicando a capacidade de processamento de cada tipo de processador;

- peculiaridades dos processadores que afetem a escalação, como por exemplo, a existência de um escalador interno em cada processador, como acontece nos Transputers;
- consideração ou não de atrasos na comunicação interprocessadores e, em caso afirmativo, uniformidade ou não dos tempos de transferência das mensagens. Caso haja variação nesses tempos, deve-se informar a maneira como eles variam. Por exemplo, se a variação no tempo de transferência de mensagens é uma função do número de nós intermediários interligando os processadores na rede, deve-se fornecer essa função;
- consideração ou não da utilização de recursos adicionais e, em caso afirmativo, a relação dos recursos a serem empregados (memória, dispositivos de E/S, etc);
- igualdade ou não nos tempos de execução das tarefas;
- relacionamento entre as tarefas (tarefas independentes ou parcialmente ordenadas);
- interruptibilidade das tarefas (pode-se ter disciplinas preemptivas ou não-preemptivas);
- flexibilidade da escalação (fixa ou flexível); e
- função(ões)-objetivo a ser(em) atendida(s).

2.2.4.2 Dados de Entrada/Saída de um Algoritmo Escalador

Ao se utilizar um algoritmo escalador, diversos dados de entrada/saída são trocados com o usuário, conforme indicado a seguir (HWANG et al., 1989).

a) Dados de entrada:

- número de processadores a ser efetivamente utilizado, discriminando a quantidade de cada tipo nos casos uniforme ou distinto;
- utilização de recursos adicionais: deve-se mencionar, para cada tarefa, quando aplicável, o tipo e a necessidade de utilização de cada recurso;
- número e duração das tarefas, indicando, quando aplicável, o tempo de processamento das tarefas em cada tipo de processador;
- número de mensagens trocadas entre tarefas; e
- grafo de precedência relacionando as tarefas.

b) Dados de saída:

- sequência de chamada das tarefas em cada processador.

Uma forma gráfica de apresentar uma escalação é exemplificada na figura 2.2, a qual mostra a alocação e o seqüenciamento de cinco processos (P1 a P5) em três processadores (T1 a T3).

OBS.: O símbolo ϕ indica processador ocioso

Fig. 2.2 Exemplo de escalação de tarefas em três processadores usando carta de Gantt

A carta de Gantt é um mapa temporal das atividades de cada processador, consistindo em um eixo com escala temporal em que se assinalam os módulos correspondentes a cada processador. Os intervalos de tempo são marcados externamente, ao passo que na linha correspondente a cada processador se rotula cada intervalo de tempo com o nome da tarefa sendo processada naquele determinado instante.

2.3 Escalação em Multiprocessadores Fracamente Acoplados

Enfoca-se, nesta seção, o conflito existente entre o balanceamento de carga computacional e o custo da comunicação interprocessadores ao se realizar a escalação em multiprocessadores fracamente acoplados. Compara-se também a abordagem clássica de comunicação realizada no início/término das tarefas com uma abordagem alternativa, em que a comunicação é realizada assim que o dado está disponível.

2.3.1 Conflito entre Balanceamento da Carga Computacional e Custo da Comunicação Interprocessadores

Nos problemas clássicos de escalação em multiprocessadores, o único tempo considerado é o de processamento das tarefas nos processadores, pois os sistemas considerados são multiprocessadores fortemente acoplados, para os quais não há atraso na comunicação interprocessadores (KASAHARA; NARITA, 1984b), (MANACHER, 1967). No entanto, essa premissa não é válida para multiprocessadores com passagem de mensagens ou redes de

computadores, visto que a sobrecarga devida à comunicação interprocessadores é um importante aspecto em tais sistemas e não é desprezível (HWANG et al., 1989). Portanto, em sistemas computacionais fracamente acoplados, deve-se considerar o tempo da comunicação interprocessadores, referente à troca de mensagens entre as tarefas residentes nos mesmos (STONE, 1977). Essa consideração adicional aumenta o grau de dificuldade da solução de problemas de escalação em multiprocessadores fracamente acoplados.

Os elos de comunicação entre processadores em um sistema fracamente acoplado são "gargalos" inerentes. Em face da sobrecarga devida à comunicação interprocessadores, a solução prática para usar um sistema fracamente acoplado é executar o mínimo possível de comunicação interprocessadores (STONE; BOKHARI, 1978).

Um problema típico de sistemas fracamente acoplados é a degradação em desempenho causada pelo efeito "saturação". Em um ambiente composto de multiprocessadores ideais, a expectativa é que o desempenho cresça linearmente com o número de processadores, desde que os módulos sendo processados sejam independentes uns dos outros. Na prática, o desempenho em sistemas fracamente acoplados cresce significativamente apenas para os primeiros processadores adicionais. Em algum ponto, o desempenho efetivamente começa a decrescer com cada processador adicional. Exemplos desse fenômeno são documentados em (CHU et al., 1980) e (JENNY, 1977).

O decréscimo no desempenho é devido à excessiva comunicação interprocessadores (CIP), causada pela necessidade de comunicação entre módulos residentes em processadores diferentes. Essa

degradação no desempenho para acréscimos nos recursos computacionais disponíveis é o chamado efeito "saturação".

Para evitar esse efeito, deve-se investigar a natureza da CIP. O custo da CIP ocorre quando dados são transferidos de um módulo a outro, ambos residentes em processadores distintos. Portanto, a CIP é uma função da comunicação intermódulos (CIM) e da alocação dos módulos. A CIM corresponde ao volume de dados transferidos entre cada par de módulos, sendo independente da alocação dos mesmos. Se dois módulos não são corresidentes, resulta $CIP = CIM$. Quando módulos são corresidentes em um processador, não existe custo de CIP (assume-se que o tempo dispendido na comunicação entre módulos corresidentes é desprezível) (CHU et al., 1980).

O particionamento e a alocação de tarefas são os dois passos necessários para minimizar a CIP. O particionamento de um sistema de tarefas busca gerar módulos com CIM mínima. A alocação de tarefas procura atribuir os módulos a processadores, minimizando a sobrecarga de tempo devida à CIP.

A CIM entre qualquer par de módulos é determinada pelo projeto de "software" e é um atributo fixo dos módulos. Visto que o volume de CIM pode variar consideravelmente de um par de módulos para outro, a maneira com que os módulos são alocados a processadores pode mudar drasticamente o custo global de processamento. Se dois módulos têm CIM alta e são atribuídos a processadores diferentes, a carga de CIP será alta. Os custos totais de processamento seriam decrescidos alocando esses dois módulos ao mesmo processador. Por outro lado, se dois módulos têm pouca CIM e nenhuma relação de precedência, designá-los para processadores diferentes pode acelerar o tempo total de processamento.

Assumindo módulos independentes, a intuição sugere que a estratégia de alocação que provê o máximo desempenho é aquela que atribui os módulos aos processadores, de tal forma que todos os processadores no sistema estejam com uma carga computacional aproximadamente igual. Supondo que todos os processadores iniciem a execução ao mesmo tempo, o último a terminar é o que determina o tempo de execução total. Em uma situação ideal, onde todos os processadores terminam ao mesmo tempo, pode-se considerar tal evento como o tempo ótimo de execução, correspondendo ao balanceamento perfeito da carga computacional (POLYCHRONOPOULOS, 1988).

Caso se tente maximizar o desempenho do sistema, minimizando a CIP, sem considerar o balanceamento de carga, todos os módulos serão atribuídos a um único processador. Há, portanto, dois fatores conflitantes: balanceamento de carga e CIP, os quais influenciam a estratégia de alocação para obtenção do desempenho ótimo do sistema. O máximo desempenho é alcançado pelo balanceamento de carga (na ausência de CIP), que tenta distribuir os módulos tanto quanto possível, mas a sobrecarga devida à CIP dirige a estratégia de alocação para agrupar módulos no menor número possível de processadores.

O objetivo da alocação de módulos é atribuir módulos a processadores para obter o máximo desempenho, representado pelo tempo de completar o programa, definido como a soma dos tempos de processamento das tarefas e da CIP, através de uma solução de compromisso entre o balanceamento de carga e a CIP (CHU et al., 1980), (EFE, 1982).

2.3.2 Alternativa para a Forma Clássica de Comunicação entre Tarefas

As formas com que se pode modelar a comunicação entre processos, se resumem a duas variantes:

a) Comunicação nos extremos do processamento (forma clássica)

A comunicação entre os processos ocorre no final do processamento do predecessor e no início do processamento do sucessor, de modo que o processo sucessor somente pode ser iniciado após o término de seu predecessor. Esta hipótese corresponde ao enfoque clássico dos problemas de escalção estática de tarefas, tanto para sistemas fortemente como fracamente acoplados, não admitindo qualquer paralelização entre tarefas predecessoras e sucessoras imediatas.

Esta abordagem reduz o volume de dados necessário para executar a escalção e, conseqüentemente, simplifica a coleta dos mesmos, pois, caso contrário, seria necessário saber em que instantes do processamento de cada tarefa há entrada ou saída de dados. Na prática, no entanto, a solicitação dos dados de entrada e o envio dos dados de saída pode ser feito em qualquer instante durante a execução da tarefa.

b) Comunicação durante o processamento (forma alternativa)

A comunicação entre os processos ocorre assim que os dados estão disponíveis no processo predecessor, de forma que o processo sucessor fica liberado para execução antes do término do processamento de seu predecessor. Analogamente, o processo sucessor somente solicita comunicação quando efetivamente

necessita do dado, de forma que se criam condições para gerar um certo paralelismo entre as tarefas comunicantes.

Na realidade, o ideal é sempre solicitar o dado de entrada apenas no momento em que ele for necessário e liberar o dado de saída tão logo tenha sido gerado. Esta forma de comunicação permite uma otimização na escalação, ficando escolhida como opção para este trabalho.

2.4 Métodos de Solução de Problemas de Escalação

O valor ótimo da escalação para um problema I , aqui denotado por $W_{OPT}(I)$, é mínimo ou máximo (dependendo da função-objetivo) dentre os valores de todas as escalações exeqüíveis. Uma escalação ótima é uma escalação exeqüível para I , com valor $W_{OPT}(I)$.

Um algoritmo de escalação A , para um problema específico, é um procedimento que, dado o problema I , produza uma escalação exeqüível para I . Seja $W_A(I)$ o valor da escalação encontrado por A quando aplicado a I . Se $W_A(I)$ sempre iguala $W_{OPT}(I)$, então A é chamado um algoritmo de otimização. Caso contrário, A é chamado um algoritmo de aproximação, subentendendo-se implicitamente que A fornecerá escalações sub-ótimas (GAREY et al., 1978).

Dentre os possíveis métodos disponíveis para resolver problemas de escalação, destacam-se os seguintes:

a) Métodos heurísticos ou aproximados

Baseiam-se no conhecimento e habilidade do projetista em prover a melhor escalação das tarefas. Normalmente obtêm-se

resultados aproximados (sub-ótimos). Os algoritmos heurísticos ou de aproximação são mais rápidos e mais simples que as técnicas que buscam soluções ótimas.

b) Métodos enumerativos

Asseguram uma resposta ótima, obtendo todas as soluções possíveis e selecionando a melhor. Correspondem a métodos de programação matemática (programação dinâmica, ramifique-e-delimite (branch-and-bound), etc).

c) Métodos híbridos

Utilizam técnicas heurísticas para encontrar uma solução aproximada e a utilizam como ponto de partida para métodos enumerativos.

Estes métodos podem gerar soluções classificáveis em uma das seguintes categorias, de acordo com (COFFMAN, 1976):

- soluções exatas;
- soluções aproximadas simples, em que se desconhece o grau de precisão da solução obtida; e
- soluções aproximadas com precisão garantida, isto é, obtem-se uma solução cuja precisão desejada é conhecida.

A obtenção de soluções exatas só é assegurada quando se utilizam os métodos enumerativos ou híbridos. Soluções aproximadas simples são obtidas usando métodos heurísticos. Por fim, as soluções aproximadas com precisão garantida ou pré-especificada são geradas através de qualquer um dos três métodos de solução relacionados anteriormente.

Antes de discorrer sobre os métodos de solução de problemas de escalação, procurar-se-á abordar os problemas de escalação dentro de um universo mais amplo, relativo aos problemas de otimização combinatorial, apresentando-se as dificuldades de resolver tais problemas. Esta introdução tem a finalidade de apresentar os obstáculos existentes na solução de problemas de escalação e as razões que conduziram os pesquisadores a desenvolver os métodos de solução apresentados neste capítulo.

2.4.1 Dificuldades Encontradas na Solução de Problemas de Escalação

Procura-se responder aqui às seguintes perguntas: "por que alguns problemas de otimização combinatorial dispõem de algoritmos eficientes de solução? São alguns problemas desse tipo intrinsecamente mais difíceis que outros ou será que algoritmos eficientes para esses problemas ainda não foram encontrados? Se diferentes problemas têm diferentes graus de dificuldade, podem os problemas ser classificados de acordo com seu grau de dificuldade?" (HU, 1982).

O objetivo deste item é analisar o conceito de grau de dificuldade de problemas. Há pelo menos duas razões para classificar problemas. Primeiro, caso se saiba que um problema pertence a uma classe muito difícil, não se deve concentrar em obter soluções ótimas, devendo-se tentar obter soluções aproximadas. Segundo, quando dois problemas pertencem a uma classe, a técnica de resolver um problema pode, algumas vezes, ser transferida para o outro problema (HU, 1982).

2.4.1.1 Problemas Intratáveis

Em geral, se está interessado em encontrar o mais eficiente algoritmo para resolver um problema, sendo que por eficiente subentende-se aqui o mais rápido (GAREY; JOHNSON, 1979). No entanto, verifica-se que procedimentos simples (não-enumerativos) para gerar escalasões ótimas só existem para casos muito restritos (vide sub-item 2.4.2.6) e, mesmo nesses casos especiais, as provas que os procedimentos realmente geram escalasões ótimas são árduas (COFFMAN; DENNING, 1973).

Os requisitos de tempo de um algoritmo são convenientemente expressos em termos de uma única variável, o "tamanho" do problema ou "comprimento da entrada", que pretende refletir a quantidade de dados de entrada necessários para descrever o problema (GAREY; JOHNSON, 1979).

Mas que quantidade de processamento um problema deve requerer antes de ser classificado como realmente difícil? Há um consenso geral de que se um problema não pode ser resolvido em menos que um tempo exponencial em função do comprimento da entrada, então o problema poderia ser considerado intratável.

A implicação desse esquema de classificação é que problemas possuindo algoritmos de solução limitados polinomialmente no tempo são tratáveis (AHO ET AL. 1975). Dito de outra forma, significa que seria razoável considerar que tais problemas seriam satisfatoriamente resolvidos, quando um algoritmo para sua solução é encontrado, o qual termina dentro de um número de passos limitado por um polinômio, função do comprimento da entrada.

Há um grande interesse na existência de algoritmos, que seguramente terminam em um número de passos limitado por um polinômio no comprimento da entrada. Há diversos problemas combinatoriais que são equivalentes, no sentido de que um algoritmo limitado polinomialmente para algum deles poderia efetivamente produzir um algoritmo limitado polinomialmente para todos eles (KARP, 1972).

A função de complexidade temporal para um algoritmo expressa seus requisitos de tempo, dando, para cada possível comprimento da entrada, a maior quantidade de tempo necessária pelo algoritmo para resolver um problema desse tamanho (GAREY; JOHNSON, 1979).

Diferentes algoritmos possuem uma ampla variedade de diferentes funções de complexidade temporal. A caracterização de quais deles são "suficientemente eficientes" e quais são "ineficientes demais" sempre dependerá da situação em análise. No entanto, reconhece-se uma distinção simples que oferece considerável discernimento dentre essas questões. Trata-se da distinção entre algoritmos tempo-polinomiais e algoritmos tempo-exponenciais (GAREY; JOHNSON, 1979).

Se $W(n)$ é o tempo para executar um algoritmo com comprimento da entrada n , então se escreve $W(n)=O(f(n))$, significando que o tempo é delimitado superiormente pela função $f(n)$ (HOROWITZ; SAHNI, 1984).

Um algoritmo é dito de complexidade $O(f(n))$ se para todas as entradas de comprimento n , sempre exista uma constante c , tal que o tempo de processamento do algoritmo $W(n) \leq c \cdot f(n)$ para todos os valores de $n \geq 0$, onde $O(\cdot)$ corresponde à notação de ordem de magnitude (GAREY; JOHNSON, 1979). Analogamente, $W(n)=\Omega(g(n))$

significa que o tempo é limitado inferiormente pela função $g(n)$. Em um algoritmo tempo-polinomial, $f(n)$ é um polinômio, enquanto que problemas para os quais nenhum algoritmo tempo-polinomial é conhecido, $g(n)$ é maior que qualquer polinômio (HOROWITZ; SAHNI, 1984).

Um algoritmo tempo-polinomial é definido como possuindo uma função de complexidade temporal $O(p(n))$ para alguma função polinomial p (GAREY; JOHNSON, 1979). Assim, caso se diga que um algoritmo tem complexidade $O(n^2)$, significa que há uma constante c , tal que a função $c \cdot n^2$ delimita o tempo de execução.

A distinção entre esses dois tipos de algoritmos tem particular significado na solução de grandes problemas. A tabela 2.1 (GAREY; JOHNSON, 1979) ilustra as diferenças nas taxas de crescimento entre funções de complexidade de ambos os tipos.

Função complex. temporal	Comprimento n					
	10	20	30	40	50	60
n	0,00001 seg	0,00002 seg	0,00003 seg	0,00004 seg	0,00005 seg	0,00006 seg
n^2	0,0001 seg	0,0004 seg	0,0009 seg	0,0016 seg	0,0025 seg	0,0036 seg
n^3	0,001 seg	0,008 seg	0,027 seg	0,064 seg	0,125 seg	0,216 seg
n^5	0,1 seg	3,2 seg	24,3 seg	1,7 min	5,2 min	13,0 min
2^n	0,001 seg	1,05 seg	17,9 min	12,7 dias	35,7 anos	366 séculos
3^n	0,059 seg	58,0 min	6,5 anos	3855 séculos	2 E8 séculos	1,3 E13 séculos

Tab. 2.1 Comparação de funções tempo-polinomiais e tempo-exponenciais

Deve-se notar as altíssimas taxas de crescimento para as duas funções de complexidade exponencial apresentadas na tabela 2.1.

Mais revelador ainda é um exame dos efeitos de tecnologias mais avançadas de computadores, em algoritmos possuindo essas funções de complexidade temporal. A tabela 2.2 (GAREY; JOHNSON, 1979) mostra como o tempo de processamento do maior problema solucionável em uma hora mudaria, caso se dispusesse de um computador 100 ou 1000 vezes mais rápido que as máquinas atuais. Observe que com o algoritmo 2^n , um crescimento de 1000 vezes na velocidade computacional acrescenta apenas cerca de 10 entradas ao tamanho do maior problema solucionável em uma hora, ao passo que com o algoritmo n^5 esse tamanho quase que quadruplica.

Função de complexidade temporal	Com computadores atuais	Com computad. 100 vezes mais rápidos	Com computad. 1000 vezes mais rápidos
n	N1	$100 \cdot N1$	$1000 \cdot N1$
n^2	N2	$10 \cdot N2$	$31,6 \cdot N2$
n^3	N3	$4,64 \cdot N3$	$10 \cdot N3$
n^5	N4	$2,51 \cdot N4$	$3,98 \cdot N4$
2^n	N5	$6,64 + N5$	$9,97 + N5$
3^n	N6	$4,19 + N6$	$6,29 + N6$

Tab. 2.2 Efeito de tecnologias mais avançadas em diversos algoritmos tempo-polinomiais e tempo-exponenciais

A tabela 2.2 indica que a intratabilidade de um problema é essencialmente independente do modelo do computador utilizado (GAREY; JOHNSON, 1979).

A maioria dos algoritmos tempo-exponenciais são meras variações de buscas exaustivas, ao passo que algoritmos tempo-polinomiais geralmente se tornam possíveis através de um entendimento mais profundo da estrutura do problema. Há um consenso que um problema não foi "bem resolvido", até que um algoritmo tempo-polinomial seja encontrado para sua solução. Conseqüentemente, um problema será denominado intratável se ele é tão difícil que nenhum algoritmo tempo-polinomial o resolva (GAREY; JOHNSON, 1979).

Assim, a noção de algoritmo "eficiente" pode ser associada com algoritmos não-enumerativos, ao passo que algoritmos "ineficientes" são aqueles que efetivamente requerem uma busca (enumeração) do espaço de soluções e têm uma complexidade que é pelo menos alguma função exponencial em n (COFFMAN, 1976).

A distinção entre algoritmos tempo-polinomiais "eficientes" e algoritmos tempo-exponenciais "ineficientes" admite muitas exceções quando os problemas de interesse têm tamanho limitado. Mesmo na tabela 2.1, o algoritmo 2^n é mais rápido que o algoritmo n^5 para $n \leq 22$. Mas, deve-se considerar que, muito embora uma função exponencial como 2^n cresça mais rápido que qualquer polinômio em n , um algoritmo limitado em tempo com grau de complexidade $O(2^n)$, para pequenos valores de n , pode ser mais eficiente que muitos algoritmos limitados em tempo por polinômios. Não obstante, a taxa de crescimento de uma função exponencial é tão acentuada, que um problema é dito intratável se todos os algoritmos para resolvê-lo são, pelo menos, de complexidade exponencial no tempo (AHO et al., 1975).

2.4.1.2 Classe de Problemas NP-completos

Uma certa classe de problemas, denominada não-determinística tempo-polinomial completa (non-deterministic polynomial-time complete) ou, como é mais conhecida, NP-completa, muito provavelmente contém somente problemas intratáveis. Essa classe de problemas inclui muitos problemas combinatoriais clássicos, sendo que todos os problemas nessa classe são equivalentes, no sentido de que se um problema é tratável, então todos são (KARP, 1972).

Visto que muitos desses problemas têm sido estudados por matemáticos e especialistas em teoria da computação ao longo de décadas e nenhum algoritmo limitado polinomialmente no tempo foi encontrado para qualquer um deles, é natural conjecturar que nenhum problema nessa classe possa ser resolvido eficientemente, de uma forma não-enumerativa e, conseqüentemente, encarar todos os problemas nessa classe como sendo intratáveis. Apesar da evidência empírica esmagadora em contrário, ainda é uma questão aberta se os problemas NP-completos admitem soluções eficientes (AHO et al., 1975), (KARP, 1986).

Como uma regra geral, pode-se considerar problemas com uma solução tempo-polinomial como sujeitos a uma solução exata; problemas NP-completos são "intratáveis", significando não estarem sujeitos a uma solução exata (COFFMAN, 1976). Estabelecer a NP-completeza de um problema pode evitar uma busca infrutífera de algoritmos eficientes e ser interpretado como uma justificativa formal para o desenvolvimento de métodos enumerativos de otimização ou então para se concentrar na busca de algoritmos heurísticos, visto ser provável que nenhum método

substancialmente melhor exista (LENSTRA et al., 1977), (LENSTRA; RINNOOY KAN, 1978).

Outra indicação da complexidade de um problema pode ser baseada na análise de algoritmos de aproximação. Para problemas NP-completos relativamente simples, freqüentemente existem técnicas heurísticas cujo desempenho é arbitrariamente próximo do ótimo; por outro lado, há situações em que mesmo o problema de encontrar uma solução exeqüível dentro de qualquer faixa de aproximação do ótimo provou ser NP-completo (LENSTRA et al., 1977).

2.4.1.3 A Intratabilidade de Problemas de Escalação

Todos os métodos gerais, atualmente conhecidos, para computar uma escalação ótima requerem uma busca combinatorial. Há teoremas que fortemente sugerem, mas não afirmam, que o problema de escalação bem como uma série de outros de análise combinatorial, permanecerão intratáveis perpetuamente (KARP, 1972). ULLMAN (1973 e 1975) mostrou que o problema geral de escalação, correspondente a encontrar uma escalação de mínimo comprimento para n tarefas parcialmente ordenadas, com tempos arbitrários de execução, em m processadores idênticos é NP-completo, de forma que é improvável que se encontre um algoritmo geral do tipo tempo-polinominal para resolvê-lo (GAREY; JOHNSON, 1979).

A complexidade em problemas de escalação é derivada principalmente da intrincada estrutura de restrições de precedência entre as tarefas e das complicadas relações entre os tempos de execução das mesmas. Limitando um ou ambos desses fatores, pode resultar nos tipos de casos especiais para os quais

escalações ótimas podem ser encontradas eficientemente (vide subitem 2.4.2.6) (GRAHAM, 1978).

Há muito poucos algoritmos tempo-polinomiais para determinar escalões de mínimo comprimento, mesmo quando restrições severas são impostas no ordenamento parcial e nos tempos de execução de um sistema de tarefas. Basicamente há dois casos para os quais algoritmos tempo-polinomiais são conhecidos, ambos assumindo que os processadores são idênticos (COFFMAN; DENNING, 1973):

- quando o grafo de precedência do sistema de tarefas é uma árvore (HU, 1961); e
- quando há somente dois processadores disponíveis (COFFMAN; GRAHAM, 1972).

Quando escalões não-preemptivos são construídas, há uma restrição adicional nos dois casos acima, de que todas as tarefas tenham o mesmo tempo de execução.

O grau de complexidade de alguns problemas não-preemptivos de escalação são mostrados na tabela 2.3 (KASAHARA; NARITA, 1984a).

Nº de process. idênticos (m)	Tempo de execução das tarefas (τ_i)	Relações de precedência	Grau de Complexidade
arbitrário	igual	árvore	$O(n)$
2	igual	arbitrário	$O(n^2)$
arbitrário	igual	arbitrário	NP-completo
arbitrário	$\tau_i=1$ ou 2 p/ todo i	arbitrário	NP-completo
arbitrário	arbitrário	arbitrário	NP-completo

Tab. 2.3 Exemplos da complexidade de alguns problemas típicos de escalação

Para uma classe restrita de problemas de otimização combinatorial, correspondente aos problemas determinísticos de escalação, 4.536 casos foram catalogados de acordo com sua complexidade computacional. 416 (cerca de 9% deles) foram reconhecidos como "fáceis" (isto é, solúveis em tempo polinomial); 3730 (82%) foram classificados como NP-completos e 390 (9%) ficaram em aberto. Os problemas nesta última classe requerem maiores investigações. Um exemplo notório dessa classe de problemas em aberto é o de minimizar o comprimento de escalação de tarefas com tempo unitário de execução em três processadores idênticos, sujeitas a restrições gerais de precedência (LAWLER; LENSTRA, 1981), (JOHNSON, D.S., 1983).

2.4.2 Técnicas Heurísticas de Escalação de Tarefas

As descobertas acerca de NP-completeza alteraram o rumo das pesquisas em escalação. Esforços anteriores se concentravam em encontrar soluções ótimas (exatas). Posteriormente, as atenções dos pesquisadores recaíram sobre soluções aproximadas, garantidas de fornecer resultados próximos dos ótimos. Visto que um algoritmo heurístico nem sempre obtém a solução ótima, duas questões que se apresentam e que ao longo do texto que se segue são respondidas, são:

- quando o algoritmo heurístico obtém a solução ótima? e
- se o algoritmo heurístico falha em obter a solução ótima, qual é o máximo erro gerado?

Para os propósitos deste trabalho, um algoritmo eficiente para escalar tarefas significa um algoritmo essencialmente não-

enumerativo, isto é, que requeira um número de passos que varie como um polinômio no número de tarefas.

Infelizmente, devido ao exposto no item anterior, verifica-se que a obtenção de uma solução ótima em um intervalo razoável de tempo é intangível e que, portanto, deve-se utilizar técnicas heurísticas, que se assentam basicamente na intuição humana, alicerçada em procedimentos empíricos criados pelos pesquisadores e que produzem uma solução aproximada (LAWLER; LENSTRA, 1981).

Técnicas heurísticas relativamente simples podem prover um desempenho considerado adequado para muitos problemas de escalação classificados como NP-completos (COFFMAN, 1976). Para a maioria dos problemas práticos, é suficiente obter boas soluções, não necessariamente ótimas (RINNOOY KAN, 1979). Muito embora os algoritmos heurísticos gerem apenas soluções sub-ótimas, eles são úteis em aplicações em que soluções ótimas não são necessárias (CHU et al., 1980) ou em que o comprimento da entrada possa se tornar grande (MA et al., 1982).

ADAM et al. (1974) arbitrariamente definiram uma solução sub-ótima, como sendo aquela em que o tempo de execução obtido pela escalação está dentro de uma faixa de $\pm 5\%$ do tempo ótimo de execução, em 90% dos casos.

As técnicas heurísticas podem genericamente produzir escalações enquadráveis como escalações em lista. Os princípios básicos de algumas técnicas de escalação em lista são abordados a seguir.

São dados m processadores idênticos T_i , $1 \leq i \leq m$, um conjunto de n tarefas $P = \{P_1, \dots, P_n\}$, um ordenamento parcial $<$ entre as tarefas e o tempo de execução de cada tarefa τ_j , $1 \leq j \leq n$. É

fornecida uma lista de tarefas (ou lista de prioridades) L , relacionando as n tarefas em ordem decrescente de prioridade, de forma que as tarefas mais prioritárias recebam primeiro o direito de acesso ao processador (COFFMAN; DENNING, 1973), (COFFMAN, 1976), (GRAHAM, 1966), (GRAHAM-1969), (GRAHAM, 1972). Se há mais de uma tarefa de mesmo nível de prioridade, os impasses são resolvidos aleatoriamente (ADAM et al., 1974). Sempre que um processador completa uma tarefa e fica livre, realiza-se imediatamente na lista L uma busca e se atribui ao processador livre (T_i) a primeira tarefa encontrada que ainda não tenha sido executada (P_j), cujas predecessoras tenham sido todas completadas e caso haja recursos suficientes para satisfazer $R_k(P_j)$ para cada $1 \leq k \leq s$.

Se alguma dessas condições não é satisfeita, busca-se a próxima tarefa em L que ainda não tenha sido executada e assim por diante. Se a busca ao longo de L termina sem encontrar uma tarefa para executar em T_i , então o mesmo fica ocioso e se diz que T_i está processando uma tarefa vazia. T_i fica ocioso até que algum outro processador T_j complete uma tarefa, quando T_i e T_j imediatamente buscam em L possíveis tarefas para executar.

Se dois processadores T_i e T_j , $i < j$, simultaneamente tentam executar a mesma tarefa P_k , convencionou-se atribuir P_k ao processador com menor índice (neste caso T_i). Todos os processadores iniciam a busca em L no tempo $t=0$ e prosseguem nessa operação até que todas as tarefas tenham sido completadas. Não se permite que nenhum processador fique ocioso quando há tarefas prontas para serem processadas. A escalação termina quando a tarefa final é completada. O tempo de término é denotado por W_L (GRAHAM, 1978).

Se uma lista é usada para gerar uma escalação, o ordenamento das tarefas (atribuição das prioridades) é o ponto mais importante de ser estabelecido (GONZALEZ, 1977).

Visto que para qualquer problema dado, há um número finito de listas possíveis, poder-se-ia sugerir averiguar todas elas e selecionar a melhor (GRAHAM, 1972). Caso se decidisse analisar todas as possíveis listas de prioridades, seria necessário testar $n!$ listas, o que representaria um número muito grande de opções, mesmo para um número pequeno de tarefas n . Por exemplo, para $n=20$ tarefas, se um computador pudesse verificar um milhão de listas por segundo, levaria mais de 77.000 anos para analisar todas elas (GRAHAM, 1978).

A escalação em lista é uma classe especial de escalação não-preemptiva, sendo que a diferença essencial entre ambas é que na escalação em lista um processador não pode permanecer ocioso quando há uma tarefa pronta para ser executada, conforme é discutido no capítulo 2 de (COFFMAN, 1976).

2.4.2.1 Anomalias em Escalações em Lista

Verifica-se que o comprimento W_L de uma escalação em lista é função da lista de prioridades L , do tempo de execução das tarefas τ_j , do ordenamento parcial $<$ entre as tarefas e do número de processadores m , além das regras que regem o modo de operação dos processadores T_i .

Há certas anomalias que ocorrem em escalações em lista, que vão de encontro à intuição e que refletem uma série de cuidados a serem tomados ao se executar as mesmas. O comportamento anômalo

se manifesta nos casos em que **acrêscimos** no comprimento da escalação são provocados por:

- aumento no número de processadores (m);
- redução nos tempos de execução das tarefas (τ_j);
- remoção de certas relações de precedência ($<$); e
- uso de diferentes listas de prioridade (L).

Dois fatores são responsáveis por esse comportamento anômalo:

- não se permite que nenhum processador fique ocioso se há uma tarefa pronta para ser executada; e
- uma vez que um processador inicia a execução de uma tarefa, ele não pode abandoná-la até que ela seja completada.

Como resultado, os processadores são forçados a iniciar a execução de tarefas relativamente sem importância (tarefas de curta duração ou tarefas envolvidas em poucas relações de precedência) e uma vez que eles tenham começado, eles não podem interromper a execução de nenhuma tarefa para iniciar tarefas mais urgentes que fiquem prontas posteriormente (GRAHAM, 1978).

Apresenta-se, a seguir, um limite superior no fator com que W_L pode crescer, devido aos efeitos gerados pelas quatro causas listadas anteriormente, agindo individual ou coletivamente.

Suponha que se execute um conjunto P de tarefas duas vezes, a primeira vez usando-se tempos de execução τ , um ordenamento parcial $<$, uma lista L e um sistema de multiprocessadores composto de m processadores e a segunda vez utilizando-se tempos de execução $\tau' \leq \tau$ (onde $\tau' \leq \tau$ significa $\tau'_j \leq \tau_j$, $1 \leq j \leq n$), um ordenamento parcial $<'$ contido ou igual a $<$ (onde $<'$ contido ou igual a $<$ significa que se $P_i <' P_j$, implica

que $P_i < P_j$ para todos P_i, P_j contidos em P), uma lista L' e um sistema com m' processadores. Sejam W_L e $W_{L'}$ os respectivos comprimentos de escalação. Obtem-se então:

$$\frac{W_{L'}}{W_L} \leq 1 + \frac{m-1}{m'} \quad (2.2)$$

O limite obtido em (2.2) é o melhor possível, no sentido que ele não pode ser substituído por nenhuma função menor em m e m' .

Para $m=m'$, o limite se torna:

$$\frac{W_{L'}}{W_L} \leq 2 - \frac{1}{m} \quad (2.3)$$

A inequação (2.3) indica que, no pior caso, a relação entre os comprimentos de duas escalações em lista quaisquer, realizadas em m processadores, é menor que $(2-1/m)$. Como uma dessas escalações pode ser ótima, significa que a escalação em lista gera, no pior caso, um comprimento que é $(2-1/m)$ vezes maior que o comprimento ótimo.

2.4.2.2 Exemplos de Escalação em Lista para Tarefas Parcialmente Ordenadas

Mostram-se a seguir cinco exemplos de escalação em lista para tarefas parcialmente ordenadas.

a) HLFET (Highest Levels First with Estimated Times)

Neste algoritmo, a lista de prioridades é formada a partir do nível da tarefa no grafo de precedência, assumindo-se como conhecido o tempo de execução de cada tarefa no caso

determinístico ou o tempo médio de execução de cada tarefa no caso estocástico.

b) HLFNET (Highest Levels First with No Estimated Times)

Assume-se (provavelmente de uma forma incorreta), que todas as tarefas tenham o mesmo tempo de execução, sendo que a prioridade atribuída a uma tarefa é seu nível, computado sob essa premissa. HLFNET pode ser usado quando HLFET é desejável, mas nenhuma estimativa para os tempos de execução das tarefas é disponível.

Um caso particular deste método, é o algoritmo de escalção para grafos de precedência em forma de árvore com tempos unitários de execução das tarefas (HU, 1961).

c) Lista Aleatória

As prioridades são atribuídas aleatoriamente.

d) SCFET (Smallest Co-levels First with Estimated Times)

A prioridade de uma tarefa é o negativo de seu co-nível, assumindo-se, como conhecidos, os tempos de processamento das tarefas.

e) SCFNET (Smallest Co-levels First with No estimated Times)

Este caso é similar a SCFET, exceto que se assume (provavelmente de uma forma incorreta) que todas as tarefas tenham o mesmo tempo de execução.

Resultados de testes indicaram que a ordem decrescente de precisão entre os métodos propostos é: HLFET, HLFNET, SCFNET, Lista Aleatória e SCFET. HLFET apresentou resultados sub-ótimos nos casos determinísticos e estocásticos estudados, não se afastando mais de 4,4% em relação aos valores ótimos (ADAM et al., 1974).

2.4.2.3 Escalação com Tempos de Execução Decrescentes (LPT)

Este algoritmo é também conhecido como LPT (Largest Processing Time) ou LF (Largest First). Neste tipo de escalação, a lista de prioridades L^* é formada arranjando-se as tarefas em ordem decrescente dos tempos de execução. Desempates são feitos de maneira arbitrária. Quando há relações de precedência entre as tarefas, este algoritmo tem um fraco desempenho; entretanto, quando as tarefas são independentes, o algoritmo LPT é garantido de fornecer resultados bem melhores, conforme é demonstrado em 2.4.3.3, alínea b (COFFMAN; DENNING, 1973), (GRAHAM, 1978).

2.4.2.4 Escalação em Nível ou do Caminho Crítico (CP)

Existe uma classe de algoritmos de escalação em nível, também denominados de "caminho crítico" (CP - "Critical Path") ou de "caminho mais longo" (LP - "Longest Path"), que são tipos particulares de algoritmos em lista, nos quais as prioridades mais altas são atribuídas às tarefas que encabeçam o mais longo caminho de tarefas não-executadas. Se $P_i, P_{k1}, P_{k2}, \dots, P_{kn}, P_j$ é um caminho dirigido de P_i a P_j , então o comprimento desse caminho é definido como:

$$\tau_i + \sum_{l=1}^n \tau_{kl} + \tau_j \quad (2.4)$$

O comprimento do caminho crítico para a tarefa P_i é definido como o comprimento do caminho mais longo de P_i a um nó terminal.

Algoritmos em nível constroem escalações de mínimo comprimento somente quando os tempos de execução das tarefas são iguais e o conjunto de relações de precedência forma uma árvore (vide 2.4.2.6, alínea "a") ou quando há dois processadores no sistema e os tempos de execução das tarefas são iguais (vide 2.4.2.6, alínea "b"). Em três ou mais processadores, o algoritmo em nível não gera escalações ótimas para relações de precedência arbitrárias (HORVATH et al., 1977).

O algoritmo HLFET (vide sub-item 2.4.2.2, alínea "a") pode ser enquadrado como um algoritmo de caminho crítico (CP), o que comprova a eficiência de escalações CP quando a medição de desempenho é o comprimento mínimo da escalação. A obtenção de resultados sub-ótimos nesse tipo de escalação em lista foi também confirmado por KOHLER (1975), que demonstrou que o desempenho dessa técnica melhora conforme o número de processadores cresce.

2.4.2.5 Método Melhorado do Caminho Crítico (CP/MISF)

Um método criado por KASAHARA; NARITA (1984a), chamado CP/MISF (Critical Path/Most Immediate Successors First) é uma versão melhorada do método CP, o qual, até então (1984) era considerado o mais eficiente algoritmo heurístico para a solução de problemas de escalação para tarefas parcialmente ordenadas. Visto que a lista de prioridades não pode ser determinada de forma única

quando há uma pluralidade de tarefas com o mesmo nível, implica que a eficiência da escalação pode variar dependendo da lista gerada (KASAHARA; NARITA, 1984b).

O método CP/MISF resolve esse problema, incorporando uma versão modificada do algoritmo ótimo de Coffman e Graham (vide 2.4.2.6, alínea b). Quando há duas tarefas P_i e P_j com o mesmo nível no algoritmo CP/MISF, a tarefa tendo o maior número de sucessoras imediatas recebe a prioridade mais alta.

O método CP/MISF consiste nos seguintes passos:

- determinar o nível N_i e o número de sucessoras imediatas NSI_i de cada tarefa;
- construir a lista de prioridades em ordem decrescente do nível N_i . Caso haja níveis iguais, atribuir maior prioridade às tarefas que possuam o maior número de sucessoras imediatas NSI_i ; e
- executar escalação em lista baseada nessa lista de prioridades.

A complexidade temporal desse algoritmo é $O(n^2 + m \cdot n)$, onde n corresponde ao número de tarefas e m ao número de processadores.

Para examinar a eficiência do método, duzentos casos foram aleatoriamente gerados (KASAHARA; NARITA, 1984a), com o número de tarefas em cada caso variando de 10 a 200. Soluções ótimas foram obtidas para cerca de 67% dos casos testados. Soluções aproximadas com erros menores que 5% foram obtidas para 87% dos casos e soluções com erros menores que 10% totalizaram 98,5% dos casos.

2.4.2.6 Escalação em Lista para dois Casos Especiais

Para os dois casos mencionados a seguir, conhecem-se algoritmos tempo-polinomiais que fornecem escalasões ótimas.

- a) Escalação para grafos de precedência em forma de árvore (HU, 1961)

Dadas n tarefas com tempos iguais de execução, arranjadas em um grafo em forma de árvore, com grau de saída=1, buscar a escalação não-preemptiva que produza o mínimo tempo total de processamento das tarefas em m processadores idênticos.

O algoritmo assume que o conjunto de tarefas na árvore tenha sido previamente processado, de forma que associado com cada tarefa P_j está seu nível N_j , disposto em uma lista L , ordenada decrescentemente em relação aos valores dos níveis das tarefas. Se há duas tarefas com o mesmo nível, então escolhe-se uma delas arbitrariamente para colocar primeiro em L .

A complexidade deste algoritmo é $O(n)$.

- b) Escalação em Dois Processadores Idênticos (COFFMAN; GRAHAM, 1972)

Trata-se do caso em que se têm n tarefas arranjadas de acordo com um grafo dirigido acíclico arbitrário G , escaladas não-preemptivamente em dois processadores idênticos, assumindo-se tempos iguais de execução para todas as tarefas. Busca-se a escalação de mínimo comprimento.

A solução para este problema envolve um algoritmo rotulador, o qual associa com cada tarefa no grafo G um número inteiro

positivo (rótulo), iniciando em 1. O algoritmo rotulador é um procedimento recursivo, que atribui rótulos crescentes às tarefas em ordem ascendente no grafo, principiando pelos vértices terminais, cujo conjunto de tarefas sucessoras é vazio. A atribuição prossegue "para cima" no grafo, de uma maneira que considera como candidatas para a atribuição do próximo rótulo, todas as tarefas cujas sucessoras já tenham sido rotuladas. O rótulo é efetivamente atribuído à tarefa, cujo conjunto de rótulos de suas tarefas sucessoras seja lexicograficamente mínimo, sendo que em caso de empate é arbitrária a atribuição do rótulo. A ordem lexicográfica corresponde à ordem do dicionário, de forma que $(5,4,2)$ é menor que $(6,2)$, $(5,4,3,2)$, $(5,4,3)$ e $(5,4,2,1)$.

O ordenamento linear de seqüências decrescentes de números inteiros positivos é formalmente descrito da seguinte forma: sejam $N=(n_1, n_2, \dots, n_t)$ e $N'=(n'_1, n'_2, \dots, n'_{t'})$ duas seqüências desse tipo, isto é, $n_i > n_{i+1}$ e $n'_j > n'_{j+1}$, para $1 \leq i < t$ e $1 \leq j < t'$. Diz-se que $N < N'$ se:

- para algum i , $1 \leq i \leq t$, tem-se $n_j = n'_j$ para todo j satisfazendo $1 \leq j \leq i-1$ e $n_i < n'_i$; ou
- $t < t'$ e $n_j = n'_j$, $1 \leq j \leq t$.

O algoritmo de escalação é definido em termos dos rótulos nas tarefas, os quais são colocados em uma lista L em ordem decrescente do valor do rótulo.

A complexidade deste algoritmo é $O(n^2)$.

Uma variante deste algoritmo, mais moderna e quase-linear no tempo, foi proposta por GABOW (1982). O algoritmo de Gabow se

baseia na idéia do HLF (Highest Level First), executando os nós nos caminhos mais longos do grafo de precedência. Este método apresenta resultados ótimos para dois processadores com relações arbitrárias de precedência e com tempos unitários de execução para as tarefas.

Um algoritmo HLF é uma escalação em nível que sempre salta para o mais alto nível possível; quando há uma escolha de nós em um nível, o algoritmo seleciona o nó que permite que os saltos subseqüentes ocorram para o nó de nível mais alto (GABOW, 1982).

O algoritmo de Gabow é processado em um tempo $O(e + n\alpha(n))$, onde "e" é o número de arcos no grafo dirigido acíclico e $\alpha(n)$ é o inverso da função de Ackermann, cujo crescimento é muito lento com n.

2.4.2.7 Escalação em Sistemas com Tempo de Comunicação Interprocessadores

Para resolver o problema de escalar tarefas em sistemas em que os tempos de comunicação interprocessadores são considerados, foi proposto por HWANG et al. (1989) um algoritmo denominado ETF (Earliest Task First). Esse algoritmo consiste basicamente em escalar primeiro a tarefa escalável mais cedo, conforme definido a seguir.

Uma tarefa é dita disponível quando todas suas predecessoras já tenham sido escaladas. O tempo de início de uma tarefa disponível é determinado por diversos fatores: quando suas tarefas predecessoras terminam, quanto tempo os atrasos na

comunicação consomem e onde a tarefa e suas predecessoras estão alocadas. A tarefa escalável mais cedo corresponde à tarefa disponível que primeiro recebe todas as mensagens de suas predecessoras, habilitando-a a ser iniciada.

Conflitos podem ocorrer durante o processo de escalação, quando duas tarefas disponíveis têm o mesmo tempo de início. Para resolver esse conflito, adota-se uma lista de prioridades L . Tal estratégia é denominada ETF/LS (list scheduling). Se a lista de prioridades L é obtida através de uma análise do caminho crítico, então o método é chamado ETF/CP.

A complexidade temporal do algoritmo ETF é $O(m \cdot n^2)$, onde m e n são o número de processadores e o número de tarefas, respectivamente.

Uma versão simplificada deste algoritmo que despreza os atrasos na comunicação entre processadores, por assumir que a razão entre o tempo de computação e o tempo de comunicação é muito alta, foi proposta por SHIRAZI; WANG (1989), na qual a seguinte premissa básica é seguida: atribuem-se as tarefas que já tenham predecessoras ou cujas predecessoras já foram todas alocadas aos processadores que possuam o menor tempo acumulado de processamento, escalando-se primeiro as tarefas com maior tempo de duração, denominadas tarefas mais pesadas. O título deste algoritmo é HNF (Heavy Node First).

2.4.3 Métodos de Avaliação de Algoritmos de Aproximação

Para produzir algoritmos de complexidade tempo-polinomial para resolver problemas NP-completos de otimização, é necessário

atenuar o significado de "resolver". Duas formas de atenuação são possíveis. Uma primeira forma é remover o requisito que o algoritmo de solução do problema I de otimização deva sempre gerar uma solução ótima. Esse requisito será substituído pelo requisito que o algoritmo para resolver I deva sempre gerar uma escalação exequível com valor próximo ao valor da solução ótima, intitulada solução aproximada.

Um algoritmo de aproximação para I é um algoritmo que gera soluções aproximadas para I. Enquanto em uma primeira análise pode-se menosprezar uma solução aproximada, deve-se considerar que freqüentemente os dados para o caso do problema sendo resolvido são apenas conhecidos aproximadamente. Então uma solução aproximada pode ser tão significativa quanto uma solução exata (ou ótima), desde que seu valor esteja suficientemente próximo ao da solução exata. Além disso, uma solução aproximada pode ser tudo que se consiga obter em um intervalo de tempo razoável (SAHNI; HOROWITZ, 1978), (HOROWITZ; SAHNI, 1984).

Outra forma de atenuação consiste em encontrar um algoritmo para I que "quase sempre" gere soluções ótimas. Algoritmos com essa propriedade são denominados algoritmos probabilisticamente bons (HOROWITZ; SAHNI, 1984). Serão estudadas a seguir as duas formas propostas, intituladas, respectivamente, técnica do desempenho garantido e técnica do desempenho esperado.

2.4.3.1 Análise de Algoritmos de Aproximação pela Técnica do Desempenho Garantido

A dependência em técnicas de aproximação ou heurísticas tem aberto caminhos para novas direções de pesquisa, que tentam

responder à seguinte questão: "se uma técnica heurística é usada, qual é o pior erro que pode resultar de sua aplicação?" (GAREY; JOHNSON, 1978), (KASAHARA; NARITA, 1984b). Este sub-item busca responder essa pergunta.

Os resultados obtidos com a teoria de NP-completeza provaram que a grande maioria dos problemas de otimização combinatorial redundam em tempos excessivos de computação. Como então lidar com tais problemas na prática? Uma abordagem possível resulta do fato que soluções sub-ótimas são com frequência suficientemente boas. Seguindo essa orientação, os pesquisadores partiram em busca de algoritmos tempo-polinomiais que fossem garantidos de produzir soluções sub-ótimas para problemas de otimização combinatorial NP-completos (KARP, 1986).

Houve época (até meio da década de 70), em que o desempenho dos algoritmos heurísticos era medido somente em testes computacionais, aplicando-os a amostras selecionadas de problemas, através da comparação da solução heurística com soluções ótimas conhecidas. No entanto, essa abordagem apresenta as seguintes desvantagens:

- baseado nesses testes, não se pode prever como o algoritmo se comportará para outros casos do problema. Pode-se apenas esperar que o algoritmo tenha um bom desempenho para outros conjuntos de dados, mas não se pode ter certeza desse resultado, principalmente, porque na maioria dos casos a solução ótima não é conhecida;
- é difícil escolher um conjunto conveniente de amostras realistas de problemas. Sempre há o risco de se omitir, acidentalmente, os casos para os quais o algoritmo em estudo

apresenta um fraco desempenho; e

- embora a técnica do desempenho garantido possa dar alguma indicação do desempenho médio do algoritmo A , ela não informa nada sobre quão próximo $W_A(I)$ estará de $W_{OPT}(I)$ em um caso específico, onde $W_A(I)$ corresponde ao comprimento de escalação gerado pelo método aproximado (GAREY et al., 1978).

As garantias de desempenho para algoritmos heurísticos são normalmente apresentadas através do estabelecimento de limites na diferença máxima entre o valor de uma solução ótima e o valor da solução gerada pelo algoritmo heurístico. Essas garantias são quantificadas determinando que um algoritmo específico constrói soluções que nunca diferem do ótimo mais que alguma constante ou, mais freqüentemente, examinando a razão entre o comprimento da escalação gerada pelo algoritmo e o comprimento da escalação ótima (GAREY; JOHNSON, 1976).

O desempenho no pior caso de um algoritmo heurístico é dado pelo maior valor que essa razão pode assumir para todos os possíveis sistemas de tarefas. Essa razão é maior ou igual a 1 e provê uma indicação de como o desempenho de uma abordagem heurística se compara com o desempenho de uma solução ótima.

2.4.3.2 Definição da Técnica do Desempenho Garantido para Problemas de Otimização Combinatorial

Seja $W_A(I)$ o resultado da aplicação de um algoritmo de aproximação A a um caso de um problema I de otimização combinatorial e seja $W_{OPT}(I)$ o resultado ótimo obtido para esse

mesmo caso do problema I. Para um problema de minimização, para todos os casos do problema I, tem-se que:

$$W_A(I) \leq r * W_{OPT}(I) + d \quad (2.5)$$

onde $r \geq 1$ e d são constantes especificadas.

Em geral, a constante aditiva d será assintoticamente desprezível (em muitos casos será 0). O fator dominante será a razão r (GAREY et al., 1978).

Ao analisar um algoritmo, deseja-se encontrar o menor r para o qual a inequação (2.5) possa ser provada, isto é, a melhor garantia possível. Pode-se provar que nenhum limite melhor pode ser encontrado, construindo casos do problema em que o algoritmo se porte essencialmente tão fracamente quanto o limite permite.

O valor r é a menor razão que pode ser garantida pelo algoritmo, sendo chamado de "razão de desempenho no pior caso" ou simplesmente "razão de desempenho" para o algoritmo (GAREY et al., 1978).

2.4.3.3 Razões de Desempenho Conhecidas para Problemas de Escalação

Fornecem-se, a seguir, razões de desempenho garantido para diversos problemas de escalação.

a) Escalação em Lista

Consideram-se, a seguir, resultados oriundos de tentativas de encontrar listas que mantenham a razão W_L/W_{OPT} próxima de 1.

Pode-se estabelecer a seguinte relação entre o comprimento da escalação em lista W_L e o comprimento da escalação ótima W_{OPT} (GRAHAM, 1966), (GRAHAM, 1969), (GRAHAM, 1972):

$$\frac{W_L}{W_{OPT}} \leq 2 - \frac{1}{m} \quad (2.6)$$

onde m corresponde ao número de processadores.

Para $m=3$, por exemplo, a relação W_L/W_{OPT} fica limitada a $5/3$. Este resultado significa que para qualquer conjunto de tarefas escaladas em três processadores, mesmo utilizando a pior lista possível, não se consegue obter uma escalação que seja mais de 66,67% pior que a escalação ótima. O limite $(2 - 1/m)$ é uma garantia de desempenho, pois ele assegura que, não importa quão complicado um sistema de tarefas (relações de precedência e tempos de execução) possa ser, e nem quão displicentemente uma lista é escolhida, a razão dentre os tempos de término W_L/W_{OPT} nunca é maior que $2 - 1/m$ (GRAHAM, 1978).

Não há nenhum procedimento eficiente conhecido para construir listas de tarefas parcialmente ordenadas, cujas escalações para três ou mais processadores são garantidas de terminar em $(2-\epsilon)$ vezes o tempo ótimo de término, para $1/m < \epsilon \leq 1$. A inequação (2.6) mostra, que a escalação para a pior lista possível tem um tempo de término que é menor que duas vezes o valor ótimo (GRAHAM, 1978).

b) Escalação de Tarefas Independentes com Algoritmo LPT

No caso em que o ordenamento parcial < entre tarefas é vazio (tarefas independentes), o limite apresentado na inequação (2.6) pode ser melhorado para:

$$\frac{W_{LPT}}{W_{OPT}} \leq \frac{4}{3} - \frac{1}{3*m} \quad (2.7)$$

onde W_{LPT} corresponde ao comprimento de escalação gerado pelo método LPT (vide 2.4.2.3).

c) Escalação do Caminho Crítico (CP)

As seguintes relações são válidas para determinar a razão de desempenho garantido em escalações do tipo caminho crítico (vide 2.4.2.4):

- Ordenamento parcial entre tarefas e tempo de execução arbitrários (COFFMAN, 1976):

$$\frac{W_{CP}}{W_{OPT}} \leq 2 - \frac{1}{m} \quad (2.8)$$

- Ordenamento parcial arbitrário entre tarefas e tempos unitários de execução para todas as tarefas (COFFMAN, 1976), (GRAHAM et al., 1979), (CHEN; LIU, 1975):

$$\frac{W_{CP}}{W_{OPT}} \leq \frac{4}{3} \quad \text{para } m=2 \quad (2.9)$$

$$\frac{W_{CP}}{W_{OPT}} \leq 2 - \frac{1}{m-1} \quad \text{para } m \geq 3 \quad (2.10)$$

d) Escalação com Algoritmo CP/MISF

O desempenho no pior caso do algoritmo CP/MISF (vide 2.4.2.5), é dado por (KASAHARA; NARITA, 1984b):

$$\frac{W_{CP/MISF}}{W_{OPT}} \leq 2 - \frac{1}{m} \quad (2.11)$$

e) Escalação utilizando o Algoritmo de Coffman e Graham

A aplicação do algoritmo de Coffman e Graham, conforme consta na alínea "b" de 2.4.2.6, para escalação de n tarefas com tempos unitários de execução em m ($m \geq 2$) processadores idênticos, conduz ao seguinte limite (LAM; SETHI, 1977):

$$\frac{W_{CG}}{W_{OPT}} \leq 2 - \frac{2}{m} \quad (2.12)$$

Conforme mostrado na tabela 2.4 (LAM; SETHI, 1977), a diferença entre $2 - 2/m$ e $2 - 1/m$ é significativa quando m é pequeno. Em particular, $2 - 2/m$ implica em uma escalação ótima para dois processadores.

m	$2 - 2/m$	$2 - 1/m$
2	1,00	1,50
3	1,33	1,67
4	1,50	1,75
5	1,60	1,80
10	1,70	1,90

Tab. 2.4 Comparação de dois limites para W_{CG}/W_{OPT}

f) Escalação com Algoritmo ETF

O desempenho garantido de ETF (vide 2.4.2.7) satisfaz a seguinte inequação (HWANG et al., 1989):

$$W_{ETF} \leq (2 - 1/m) * W_{OPT,i} + C \quad (2.13)$$

onde $W_{OPT,i}$ corresponde à escalação de mínimo comprimento sem considerar atrasos de comunicação e C se relaciona com os tempos de comunicação.

À medida que os processadores estão processando enquanto a comunicação está ocorrendo, o valor de C diminui. Se toda a comunicação é "escondida", isto é, ocorre em paralelo enquanto os processadores estão processando as tarefas, chega-se a $C=0$ e o limite na inequação (2.13) coincide com o limite descoberto por Graham (vide inequação (2.6)).

Em todos os casos citados, os limites fornecidos pelas relações são os melhores possíveis, representando os piores valores de escalação que os algoritmos em questão podem gerar quando comparados com valores produzidos por escalações ótimas.

2.4.3.4 Análise de Algoritmos de Aproximação pelo Método do Desempenho Esperado

Como uma alternativa para a abordagem de garantia de desempenho no pior caso, pode-se cogitar de utilizar análise de desempenho de um ponto de vista de "desempenho médio ou esperado". Freqüentemente, isso envolve arquitetar um conjunto de casos supostamente "típicos" do problema, processando o algoritmo nesses casos escolhidos e comparando os resultados. Naturalmente, a utilidade de tais experimentos depende de quão "típico" os casos-amostra realmente são, e nem sempre é fácil gerar uma coleção apropriada de casos. Experimentos feitos nessa linha, confirmaram a tendência de que o comportamento médio dos

algoritmos é geralmente muito melhor que seu comportamento no pior caso (GAREY; JOHNSON, 1979).

A desvantagem de utilizar o desempenho médio é que os resultados de casos "típicos" não são suficientes para afirmar nada sobre como o algoritmo irá se comportar para casos específicos do problema, ao passo que garantias do pior caso pelo menos provêem um limite para esse desempenho. Assim, a recomendação para usar algoritmos de aproximação, é analisá-los tanto do ponto de vista do pior caso quanto do desempenho médio (GAREY; JOHNSON, 1979).

A análise probabilística de algoritmos combinatoriais foi incentivada pelo bom desempenho dos algoritmos heurísticos na prática, em casos cujos dados de entrada são tirados de alguma distribuição razoável de probabilidades, diferente do que ocorre no estudo do pior caso, em que os casos de um problema são construídos de uma forma direcionada a mostrar o pior desempenho do algoritmo, através de uma seleção cuidadosa dos dados de entrada. Na abordagem estocástica não há como saber quais entradas serão apresentadas ao algoritmo (KARP, 1986).

Na análise do desempenho médio de algoritmos de aproximação para problemas de escalação, o que se faz não é uma escalação probabilística, visto a escalação ser feita com informações completas e disponíveis "a priori" relativas aos tempos de processamento das tarefas. O aspecto probabilístico se revela na amostragem dos tempos de processamento das tarefas de uma distribuição estatística (uniforme, exponencial, etc), antes de tomar decisões de escalação. Isso significa que a técnica do desempenho esperado consiste na análise probabilística de métodos determinísticos de escalação.

O método de análise probabilística do desempenho provê informações sobre a razão $W_m/W_{OPT,m}$ (onde o subscrito m denota valor médio) e sobre o formato da distribuição dos tempos de término W dos algoritmos heurísticos utilizados quando os tempos de execução das tarefas são uniformemente ou exponencialmente distribuídos (BRUNO; DOWNEY, 1986).

Idealmente, o objetivo da análise probabilística de algoritmos combinatoriais é provar que alguns algoritmos simples produzem soluções ótimas ou sub-ótimas com alta probabilidade. Naturalmente tal resultado é significativo somente se a distribuição de probabilidades assumida tem semelhança com a população de casos que existem no mundo real (KARP, 1986).

Um fenômeno interessante na teoria da probabilidade é a lei dos grandes números, que diz que o efeito cumulativo de um grande número de eventos aleatórios é altamente previsível, muito embora os resultados de eventos individuais sejam imprevisíveis. A análise probabilística revelou que o mesmo fenômeno governa o comportamento de muitos algoritmos de otimização combinatorial, quando os dados de entrada são coletados de uma distribuição de probabilidades simples. Com uma probabilidade muito alta, a execução do algoritmo ocorre em uma forma altamente previsível e a solução produzida é quase-ótima.

2.4.3.5 Limites Probabilísticos no Desempenho de Escalação em Lista para Tarefas Independentes

O seguinte resultado, atribuído a BRUNO; DOWNEY (1986), mostra a distribuição do desempenho esperado para qualquer escalação em lista de tarefas independentes.

Para qualquer $\epsilon > 0$ e para qualquer n inteiro positivo, se $\tau_1, \dots, \tau_n$ são tempos de processamento de tarefas, amostrados de uma distribuição uniforme em qualquer intervalo $[A, B]$ com $A > 0$, então para qualquer escalação em lista L , tem-se:

$$P \left[\frac{W_L(\bar{\tau})}{W_{OPT}(\bar{\tau})} < 1 + \frac{2*(m-1)}{n*(1-2*d_{n,\epsilon})} \right] \geq 1-\epsilon \quad (2.14)$$

onde $\bar{\tau} = (\tau_1, \dots, \tau_n)$ denota uma lista de variáveis aleatórias independentes.

O valor de $d_{n,\epsilon}$ para alguns valores de n e ϵ pode ser obtido da tabela 2.5.

n	$1-\epsilon = 0,95$	$1-\epsilon = 0,99$
5	0,509	0,627
10	0,369	0,457
15	0,304	0,377
20	0,265	0,329
25	0,238	0,295
30	0,218	0,270
35	0,202	0,251
40	0,189	0,235
>40	$\frac{1,22}{\sqrt{n}}$	$\frac{1,52}{\sqrt{n}}$

Tab. 2.5 Valores de $d_{n,\epsilon}$ obtidos do teste estatístico de Kolmogorov-Smirnov

2.4.3.6 Comparação entre Escalação de Desempenho Esperado e de Desempenho Garantido para Método LPT

Visto que o método LPT (vide sub-item 2.4.2.3) é um caso particular da escalação em lista, estende-se os resultados obtidos no sub-item 2.4.3.5 para o algoritmo LPT e se comparam os resultados obtidos do desempenho probabilístico com os resultados do desempenho garantido (BRUNO; DOWNEY, 1986).

A título de exemplo, suponha que se tenha $m=2$ processadores. O desempenho no pior caso, conforme visto na inequação (2.7) para $m=2$ é $7/6=1,167$, com probabilidade 1 para qualquer número de tarefas n . No caso probabilístico, suponha que se deseje 99% de limite de confiança ($\epsilon=0,01$) para $W_{LPT}(\bar{\tau})/W_{OPT}(\bar{\tau})$. Da inequação (2.14) sai:

$$\frac{W_{LPT}(\bar{\tau})}{W_{OPT}(\bar{\tau})} < \beta_m(n, \epsilon) = 1 + \frac{2*(m-1)}{n*(1-2*d_{n, \epsilon})} \quad (2.15)$$

que resulta nos valores apresentados na tabela 2.6.

n	$\beta_2(n; 0,01)$
20	1,292
40	1,094
60	1,055
80	1,038
100	1,029

Tab. 2.6 Limite superior na relação W_{LPT}/W_{OPT} com limite de confiança de 99%

O limite probabilístico passa a fornecer um melhor resultado do que o determinístico a partir de $n=28$, quando o valor de $\beta_2(28;0,01)$ é 1,162. Com $n=100$, por exemplo, o limite melhora para 1,029 com probabilidade mínima de 0,99.

Conclui-se que, dado o intervalo $[A,B]$ dos tempos de processamento das tarefas, o resultado da escalação em lista W_{LPT} , para o caso estocástico, se aproxima do resultado da escalação ótima W_{OPT} conforme n cresce, sendo que para o caso do desempenho garantido, esse valor é fixo. O preço pago por essa liberdade é que o caso probabilístico não vale para toda escolha de τ , como por exemplo quando se tem o pior caso, mas somente com alta probabilidade se os tempos de processamento das tarefas são distribuídos uniformemente (BRUNO; DOWNEY, 1986).

2.4.4 Técnicas Enumerativas de Escalação de Tarefas

Para todos os problemas de escalação, um número finito de escalações precisa ser considerado em busca de um resultado ótimo. Assim, todos esses problemas são efetivamente solucionáveis por métodos enumerativos, como programação dinâmica, delimite-e-ramifique (branch-and-bound) e outras técnicas de programação matemática.

Deve-se reconhecer a utilidade dos métodos enumerativos de otimização, na solução de problemas de tamanho moderado, podendo-se usar artifícios para atrasar ou atenuar o crescimento exponencial nos tempos de processamento desses métodos (KARP. 1975). Um dos artifícios utilizados para empregar métodos enumerativos é partir de condições iniciais determinadas por algum método heurístico.

Visto que se pretende utilizar métodos heurísticos neste trabalho, o aprofundamento nos métodos enumerativos será pequeno.

2.4.4.1 Programação Dinâmica

A programação dinâmica é uma técnica matemática para resolver certos tipos de problemas seqüenciais de decisão. Caracteriza-se um problema seqüencial de decisão como sendo um problema em que uma seqüência de decisões deve ser tomada, com cada decisão afetando futuras decisões (HOWARD, 1966).

A idéia permeando essa técnica é representar o problema por um processo que evolui de estado a estado, em resposta a decisões. Um problema típico de otimização se converte então em guiar o sistema a um estado terminal sob custo mínimo. Quando a estrutura de custos é apropriada, a determinação de uma política ótima (seqüência de decisões) pode ser reduzida à solução de uma equação funcional, na qual as variáveis de estado aparecem como variáveis independentes (KARP; HELD, 1967).

A abordagem da programação dinâmica é a seguinte:

- considerar o problema como sendo constituído de muitos subproblemas. Se o problema é tomar muitas decisões simultaneamente, a idéia é formular o problema como um processo de decisão seqüencial, onde algumas das decisões já tenham sido tomadas e então decidir qual seria a melhor decisão sob essas condições; e
- quando todos os subproblemas forem resolvidos ou quando todas as decisões parciais ótimas forem conhecidas, pode-se combinar essas soluções parciais para obter a solução ótima do problema

completo. A relação dos subproblemas com o problema completo pode usualmente ser definida recursivamente por uma equação funcional (HU, 1982).

A programação dinâmica atua como uma transformação que recebe um processo de decisão seqüencial ou multi-estágios, contendo muitas variáveis interdependentes e o converte em uma série de problemas de decisão mono-estágio, cada um contendo apenas algumas variáveis de estado. Essa transformação é invariante, significando que o número de soluções exeqüíveis e o valor da função-objetivo associado com cada solução exeqüível são preservados. A transformação é baseada no "Princípio de ser Ótimo" enunciado por BELLMAN (1957, 1962):

"Uma política ótima tem a propriedade de que qualquer que seja o estado inicial e a decisão inicial, as decisões restantes devem constituir uma política ótima com respeito ao estado resultante da primeira decisão."

A implicação dessa afirmação é que uma política pode ser desmembrada em muitas subpolíticas, onde o fato da política ser ótima acarreta que todas as subpolíticas são ótimas também, isto é, qualquer segmento de uma seqüência ótima de decisões deve ser ótimo (HU, 1982), (PAPADIMITRIOU; STEIGLITZ, 1982).

2.4.4.2 Método Ramifique-e-Delimite (branch-and-bound)

A utilidade do método ramifique-e-delimite (RD) deriva do fato que, em geral, somente uma pequena fração das soluções possíveis necessitam efetivamente ser enumeradas e conseqüentemente inspecionadas, sendo as soluções restantes eliminadas através da

aplicação de limites que estabelecem que tais soluções não podem ser ótimas (MITTEN, 1970).

A idéia permeando um algoritmo RD é decompor um dado problema em diversos problemas parciais, de dimensões menores. A decomposição é repetidamente aplicada, até que cada problema não-decomposto esteja resolvido ou provado não produzir uma solução ótima. A ordem com que os problemas parciais são testados (e possivelmente decompostos) é especificada por uma estratégia de busca (IBARAKI, 1976). Para cada problema parcial é calculado o seu custo, o qual é usado para descartar certos subconjuntos de problemas parciais (BALAS, 1979). Os nós eliminados são denominados nós mortos ou inativos. Os nós restantes, a partir dos quais a ramificação é ainda possivelmente prolífera, são intitulados nós vivos ou ativos (PAPADIMITRIOU; STEIGLITZ, 1982).

Na prática, para evitar a análise de todas as soluções exeqüíveis, torna-se a enumeração eficiente, usando tantas informações quanto possível, para reduzir a árvore de busca, conforme sugerido em (COFFMAN, 1976).

O método da força bruta (enumeração completa) consistiria em formar todas as ênuplas (permutações) possíveis e avaliar o valor da função-objetivo para cada uma, registrando aquelas que gerassem a solução ótima. A virtude do algoritmo RD se baseia em sua habilidade em produzir a mesma resposta, gerando menos permutações. Sua idéia básica é construir o vetor ótimo, um componente por vez, e usar funções-objetivo modificadas, normalmente chamadas de funções limitantes, para testar se o vetor sendo formado tem alguma chance de sucesso. A principal vantagem deste método é que, caso se perceba que o vetor parcial não pode, de forma nenhuma, conduzir a uma solução ótima, então

pode-se desprezar todos os demais vetores, cuja permutação tenha como base esse vetor parcial (HOROWITZ; SAHNI, 1984).

2.4.4.3 Relação entre os Métodos RD e Programação Dinâmica

Há um forte relacionamento entre os métodos RD e de programação dinâmica. Em ambos os casos, o espaço de busca de soluções pode ser modelado como uma árvore dirigida finita de soluções parciais e o objetivo é localizar e isolar uma solução ótima, enumerando explicitamente apenas uma pequena porção da árvore completa de busca (COFFMAN, 1976).

Apesar de ambos os métodos executarem uma enumeração "inteligente" de todos os pontos exeqüíveis de um problema, eles o fazem de maneira diferente. Enquanto no método de programação dinâmica utiliza-se "o princípio de ser ótimo", no método RD procura-se gerar nós na árvore de busca e eliminar os nós que não conduzem a soluções ótimas (PAPADIMITRIOU; STEIGLITZ, 1982).

Uma possível vantagem do método RD é a possibilidade de se poder obter, além da solução ótima, soluções com precisão garantida, com a conseqüente redução na utilização de recursos computacionais.

2.4.5 Limites no Número Mínimo de Processadores e no Tempo Mínimo

O comprimento W de qualquer escalação deve ser maior ou igual a W_{CR} ($W \geq W_{CR}$). Caso contrário, qualquer que seja o número de processadores, seria impossível executar as n tarefas em um tempo menor que W_{CR} (GRAHAM, 1972).

Se n' denota o número máximo de tarefas mutuamente compatíveis em P , então nunca é necessário ter mais que n' processadores no sistema, visto que não mais que n' tarefas estarão ativas simultaneamente em qualquer instante (GRAHAM, 1972).

Dois problemas fundamentais para a escalação de tarefas em multiprocessadores compostos de unidades idênticas são os seguintes (RAMAMOORTHY et al., 1972), (FERNÁNDEZ; BUSSELL, 1973), (FERNÁNDEZ; LANG, 1975):

- dado um conjunto P de tarefas parcialmente ordenadas, representado por um grafo dirigido acíclico, encontrar o número mínimo de processadores para executá-las em um tempo que não exceda o comprimento do caminho crítico do grafo (W_{CR}); e
- determinar o tempo mínimo para processar esse conjunto de tarefas, quando um número fixo de processadores é disponível.

Para resolver esses dois problemas, são calculados os seguintes limites:

- limites inferior e superior para o número mínimo de processadores requerido para executar P , tal que $W = W_{CR}$, onde W é o comprimento de qualquer escalação gerada pelo método sendo empregado; e
- limite inferior no tempo mínimo para executar P , quando se tem um número fixo de processadores m .

Em todos os métodos usados para resolver problemas de escalação, é necessário ter um valor de partida para o número de processadores ou para o tempo total de execução. As fórmulas estabelecendo esses limites, fornecendo as faixas mais estreitas que se conhecem para esses valores, foram criadas por FERNÁNDEZ;

BUSSELL (1973) e posteriormente aprimoradas por FERNÁNDEZ; LANG (1975) e YANG (1976).

2.5 Evolução Histórica dos Estudos sobre Escalação

Dentro do campo da otimização discreta, a área de escalação é bastante ampla, sendo objeto de pesquisas desde o princípio da década de 50. Os primeiros estudos sobre o assunto foram feitos por especialistas em Pesquisa Operacional (Operations Research) e Ciência do Gerenciamento (Management Science), envolvendo conjuntos de tarefas parcialmente ordenadas, com o interesse voltado para linhas de produção. Esses trabalhos lidavam com análises de caminhos críticos e de sensibilidade a mudanças na estrutura de precedência, custo ou disponibilidade de recursos e tempos de execução das tarefas (COFFMAN, 1976).

As pesquisas pioneiras se concentravam principalmente na escalação em máquinas individuais, "flow-shop" e "job-shop", sendo que os estudiosos da área de pesquisa operacional desenvolveram os primeiros algoritmos tempo-polinomiais, utilizando regras heurísticas simples (RINNOY KAN, 1979).

Dentro da Ciência da Computação, os artigos pioneiros abordando esse tema foram publicados no início da década de 60. Os pesquisadores dessa área, motivados pela aplicabilidade de modelos de escalação em sistemas operacionais, passaram então a estudar essa área. Esses estudiosos utilizaram amplamente a teoria de complexidade computacional e desenvolveram diversos algoritmos novos, para problemas em processadores únicos e em paralelo. Para problemas difíceis (NP-completos), o desempenho no

pior caso de muitos algoritmos de aproximação foi analisado (RINNOOY KAN, 1979).

Esta seção aborda os principais trabalhos na área de escalação estática publicados desde a década de 50 e, em particular, aqueles que se relacionam mais diretamente com a solução do problema abordado neste trabalho.

Deve-se ressaltar que muitas das técnicas de escalação de tarefas em processadores, em uso atualmente, foram adaptadas de resultados mais antigos, estabelecidos em estudos desenvolvidos por especialistas em Ciência do Gerenciamento e Pesquisa Operacional, pois as contribuições feitas nessa área por esses pesquisadores foi proeminente.

2.5.1 Cronologia da Resolução de Problemas Gerais de Escalação

Um dos primeiros artigos publicados sobre a solução de problemas de escalação, hoje considerado um artigo clássico, foi escrito por JOHNSON, S.M. (1954), a respeito de um caso típico de "flow-shop". Tratava-se de um algoritmo para encontrar a escalação ótima de n operações em duas máquinas, de forma que cada operação tinha que passar primeiro em uma e depois na outra máquina, supondo haver apenas uma máquina de cada tipo e que apenas uma operação pudesse estar sendo realizada em uma dada máquina em cada instante. O objetivo era minimizar o tempo de término do processamento das n operações. O mesmo artigo ainda sugeria uma solução para um caso de três máquinas, submetidas a certas restrições adicionais.

A escalação do empacotamento em caixas (bin packing) foi utilizada por McNAUGHTON (1959) e ROTHKOPF (1966) para construir escalações preemptivas de mínimo comprimento, para tarefas independentes em $m \geq 1$ processadores idênticos.

No início da década de 60 surgiram os primeiros artigos sobre escalação em revistas da área de Computação. Um desses artigos, redigido por SCHWARTZ (1961) e publicado no "Journal of the Association for Computing Machinery" apresentava um procedimento automático de seqüenciamento para atribuir conjuntos de instruções a unidades autônomas de multiprocessadores, objetivando minimizar o tempo total de processamento, mas satisfazendo certas relações de precedência, representadas por restrições parciais de ordenamento entre módulos. O artigo pressupunha que já houvesse sido feita a atribuição dos conjuntos de instruções (módulos) aos processadores, cabendo ao algoritmo apenas prover a melhor forma de seqüenciá-los dentro de cada processador, através de uma escalação do tipo fixa e não-preemptiva.

Em 1961, HU publicou um artigo relacionado com o problema de "flow-shop", que também se tornou um clássico. Dadas n tarefas com tempos iguais de processamento e relacionadas através de um grafo de precedência do tipo árvore, com grau de saída=1, o autor sugeria um algoritmo para distribuir as tarefas entre m processadores idênticos de modo a:

- completar os módulos no menor tempo possível dados m processadores; e
- completar as tarefas em um tempo pré-determinado W , de forma que se utilizasse o menor número possível de processadores.

Este caso difere do de JOHNSON, S.M. (1954), pois enquanto o algoritmo de Johnson posicionava n tarefas em uma sequência única, o algoritmo de Hu arranjava as n tarefas em diversas seqüências que iniciavam simultaneamente, consistindo em um seqüenciamento paralelo.

Em 1966, GRAHAM publicou um artigo que serviu de base aos estudos de métodos aproximados com desempenho garantido. Posteriormente, em 1969, 1972 e 1976 (COFFMAN, 1976), GRAHAM deu prosseguimento a seus estudos nessa área, publicando novos trabalhos.

Em 1967 surgiu o livro "Theory of Scheduling" (CONWAY et al., 1967) que, segundo os autores, foi uma primeira tentativa em organizar o trabalho que até então havia sido feito por diversos estudiosos do assunto. Nesse livro, os autores procuraram criar uma consistência na terminologia utilizada na área e uma taxinomia razoável, além de sumarizar muito do que havia sido escrito a respeito de escalação.

No final da década de 60, surgiram dois artigos importantes. O primeiro deles, de autoria de FUJII et al. (1969), apresentava um algoritmo para escalar n tarefas com uma relação arbitrária de precedência entre elas em dois processadores idênticos, com cada tarefa requerendo uma unidade de tempo para sua execução. O objetivo era completar todas as tarefas no menor tempo possível. Este algoritmo difere do de HU (1961), pois enquanto este último pressupunha uma relação de precedência na forma de árvore, mas com m processadores disponíveis, o de Fujii et al. aceitava a relação de precedência dada por um grafo dirigido acíclico arbitrário, mas com apenas dois processadores disponíveis.

O segundo artigo, assinado por MUNTZ; COFFMAN (1969), considerava a escalação preemptiva de n tarefas em dois processadores idênticos, objetivando obter a escalação de mínimo comprimento. Assumia-se conhecer o tempo de processamento de cada tarefa e suas relações de precedência correspondiam a um grafo dirigido acíclico.

A principal diferença entre os dois trabalhos é que o último não permitia a preempção (escalação básica), ao passo que o de Muntz e Coffman a permitia (escalação preemptiva).

Em 1970, MUNTZ e COFFMAN publicaram um artigo, no qual apresentava-se um algoritmo para encontrar a escalação preemptiva para qualquer número de processadores, que prouvesse o menor tempo total de processamento para n tarefas, cuja estrutura de precedência era representável por um grafo em forma de árvore, cujos pesos dos nós fossem mutuamente comensuráveis. Este trabalho difere do anterior dos mesmos autores, publicado em 1969, na medida em que este último era aplicável para relações de precedência representadas por grafos dirigidos acíclicos e visava a escalação em dois processadores, ao passo que o artigo de 1970 se restringia a relações de precedência em forma de árvore mas expandia a quantidade de processadores para qualquer número.

Em 1972 foi publicado um artigo clássico por COFFMAN e GRAHAM, que expunha um algoritmo para encontrar uma escalação básica em dois processadores que minimizasse o tempo total de execução das tarefas. Supunha-se que todas as tarefas tinham o mesmo tempo de execução, assumido como unitário. As relações de precedência eram representadas por um grafo dirigido acíclico arbitrário. Este trabalho se propunha a resolver o mesmo problema que o apresentado por Fujii et al. em 1969, tendo como vantagem o

processamento do algoritmo de escalação mais rapidamente que no caso de Fujii et al.. O algoritmo de Fujii et al. resolvia o problema em um tempo proporcional a $O(n^3)$, enquanto que o algoritmo de Coffman e Graham resolvia o mesmo problema em um tempo proporcional a $O(n^2)$, onde n corresponde ao número de tarefas a serem escaladas.

Um outro artigo publicado em 1972 por RAMAMOORTHY et al. foi considerado muito importante, por ser o primeiro a apresentar uma solução para o problema de prover uma escalação de comprimento mínimo para n tarefas em m processadores. Assumia-se uma disciplina de operação não-preemptiva. O grafo de dependência era dirigido acíclico arbitrário. Consideravam-se duas opções: tarefas de duração igual ou diferente.

Os autores apresentaram e compararam três possíveis algoritmos de solução, um baseado em programação dinâmica que, se concluído, produzia uma solução ótima, e outros dois algoritmos heurísticos. Os resultados indicaram que os algoritmos heurísticos eram precisos e que os tempos de processamento do programa de escalação dos dois algoritmos heurísticos propostos ficaram bastante aquém dos tempos do algoritmo de programação dinâmica, em relações que variaram de 2,4:1 a 14,4:1.

Em 1974, BUSSELL et al. apresentaram um artigo que tratava do caso de atribuir não-preemptivamente n tarefas parcialmente ordenadas, relacionadas através de um grafo dirigido acíclico, com tempos arbitrários de execução, a m processadores idênticos. Buscavam-se os seguintes objetivos: um limite (deadline) no tempo de execução total devia ser satisfeito, usando um número mínimo de processadores a ser determinado; ou um número fixo de processadores devia ser usado para executar o conjunto de tarefas

em um tempo mínimo a ser determinado. Este artigo e o de RAMAMOORTHY et al. (1972) se propunham a resolver o problema geral de escalção de mínimo comprimento, atribuindo e seqüenciando n tarefas interdependentes em m processadores.

Em 1974, ADAM et al. apresentaram um artigo comparando cinco diferentes filosofias de escalção em lista. Esse enfoque aplicando técnicas heurísticas apresentou ótimos resultados, pois a melhor das técnicas de escalção estudada foi quase-ótima, apresentando resultados nos casos analisados que divergiram no máximo de 4,4% em relação ao valor ótimo.

Em 1976, COFFMAN foi editor de um livro que se tornou um clássico na teoria de escalção. Esse livro proveu um tratamento teórico abrangente e profundo de problemas de escalção existentes em ambientes computacionais e de manufatura.

Em 1982, GABOW publicou um artigo em que melhorava ainda mais o algoritmo para escalar n tarefas parcialmente ordenadas, com tempos diferentes de processamento, em dois processadores idênticos, visando minimizar o comprimento da escalção. Este algoritmo era um aprimoramento em relação aos trabalhos de FUJII et al. (1969) e de COFFMAN; GRAHAM (1972), pois o tempo de processamento do algoritmo de escalção era quase linear no número de tarefas n .

Todos os algoritmos apresentados neste item empregavam algoritmos heurísticos para resolver o problema de escalção, exceto o trabalho de RAMAMOORTHY et al. (1972), onde foi também proposto um algoritmo utilizando programação dinâmica.

2.5.1.1 Escalação em Sistemas Fracamente Acoplados

Até o meio da década de 70, os trabalhos publicados sobre escalação desprezavam os tempos gastos em comunicação entre processadores. A partir dessa época, começaram a surgir artigos considerando sistemas computacionais distribuídos fracamente acoplados, interligados por uma via de comunicações do tipo rede local ou via de dados, nos quais os tempos de comunicação interprocessadores não são desprezíveis em face dos tempos de processamento das tarefas.

Um primeiro trabalho de relevo nessa área foi publicado no princípio de 1977 por STONE, o qual enfatizava que a atribuição de tarefas independentes a processadores em um sistema distribuído, devia considerar dois tipos de custo: custo de processamento em um processador e custo de comunicação interprocessadores. O problema era encontrar uma atribuição que minimizasse a composição desses dois custos.

No final de seu artigo, Stone levantou uma série de problemas abertos para pesquisa, dentre os quais se mencionava o caso de se efetuar uma escalação de tarefas relacionadas através de restrições de precedência, mas considerando os tempos gastos na comunicação interprocessadores, o que até então não era feito. Constata-se que os artigos de diversos autores considerando o tempo gasto na comunicação entre processadores, que sucederam este trabalho de Stone, não levavam em conta as relações de precedência entre tarefas.

Um dos primeiros artigos a considerar o problema de escalação de tarefas parcialmente ordenadas, considerando o tempo de comunicação interprocessadores, foi publicado em 1982 por

KASAHARA e NARITA. Nesse trabalho, os autores distinguem dois níveis de problemas relacionados com alocação de recursos e escalação de tarefas em sistemas distribuídos: o problema Interblocos e o problema Intrablocos.

O problema Interblocos visava a minimização do tempo total de processamento de um programa, encarado como a soma dos tempos de computação das tarefas nos processadores com o tempo de comunicação interprocessadores. O problema Intrablocos se preocupava em como alocar tarefas a processadores, visando minimizar o tempo total de processamento, obedecendo às relações de precedência existentes entre as tarefas e considerando o tempo de comunicação interprocessadores. Entende-se por bloco um conjunto de tarefas agrupadas para realizar alguma função bem definida.

O problema Interblocos é um caso típico de alocação de tarefas independentes em sistemas distribuídos; já o problema Intrablocos visa, além da alocação, o seqüenciamento de tarefas parcialmente ordenadas em sistemas distribuídos, objetivando a minimização do tempo total de processamento das tarefas. Para esses dois problemas, os autores propuseram a solução através de algoritmos heurísticos.

Em 1984, KASAHARA e NARITA publicaram dois artigos, no qual afirmavam que, até então, dentro do que conheciam, não havia nenhum trabalho publicado (exceto o deles mesmo de 1982), para prover a escalção de tarefas parcialmente ordenadas no caso do tempo de comunicação interprocessadores ser considerado, devido à extrema dificuldade de solução do problema. Nesses dois artigos, os autores forneceram um algoritmo heurístico e um híbrido para solucionar o problema de escalção, sendo que para ambos os

algoritmos haviam duas versões: uma desconsiderando os tempos de comunicação e outra considerando.

Foi publicado em 1989 por HWANG et al. um artigo propondo um algoritmo para escalar não-preemptivamente n tarefas parcialmente ordenadas, com tempos diferentes de processamento, em m processadores idênticos, sujeitos a atrasos na comunicação entre si, objetivando minimizar o tempo de término. Este artigo foi uma evolução dos métodos propostos por KASAHARA; NARITA (1984a e 1984b), pois enquanto neste último, o tratamento de casos em que o tempo de comunicação interprocessadores era tratado como uma extensão dos algoritmos CP/MISF e DF/IHS, o trabalho de Hwang et al. se preocupava exclusivamente com esse tipo de problema.

2.5.2 Cronologia dos Métodos de Delimitação

Para a solução dos problemas de limites no tempo mínimo e no número mínimo de processadores (vide item 2.4.5), os seguintes trabalhos de destaque foram publicados:

- em 1961, HU resolveu os dois problemas para o caso das tarefas terem o mesmo tempo de execução e as relações de precedência corresponderem a uma árvore com raiz;
- em 1972, RAMAMOORTHY et al. apresentaram resultados para grafos dirigidos acíclicos com tarefas de tempos iguais, determinando limites inferior e superior no número mínimo de processadores e limite inferior no tempo mínimo; e
- em 1973, FERNÁNDEZ e BUSSELL apresentaram soluções parecidas com as de Ramamoorthy et al., mas com resultados mais precisos

que os anteriormente obtidos, isto é, com os limites inferior e superior mais próximos.

Em um artigo posterior, FERNÁNDEZ; LANG (1975) implementaram métodos eficientes para calcular as fórmulas apresentadas em (FERNÁNDEZ; BUSSELL, 1973), de forma que o número de operações necessário para obter os limites foi reduzido. Novos aperfeiçoamentos foram introduzidos em 1976 por YANG visando reduzir a complexidade computacional do método de FERNÁNDEZ; BUSSELL (1973).

2.5.3 Evolução dos Métodos de Solução

Os métodos de solução de problemas de escalação trilharam uma seqüência lógica de desenvolvimento, que pode ser descrita pelas filosofias de pesquisa descritas a seguir.

Originalmente buscavam-se soluções exatas, muito embora sem uma metodologia bem definida, onde cada autor criava um algoritmo de solução, normalmente heurístico, voltado para um determinado problema. Essa etapa foi iniciada na década de 50 e se prolongou aproximadamente até o meio da década de 70. Os resultados ótimos para problemas de escalação, solúveis em tempo-polinomial, obteníveis via algoritmos heurísticos, foram todos descobertos nessa fase, como por exemplo os trabalhos de JOHNSON, S.M. (1954), McNAUGHTON (1959), HU (1961), FUJII et al. (1969), COFFMAN; GRAHAM (1972), entre outros.

As outras duas etapas foram uma consequência dos estudos de complexidade computacional dos problemas de otimização discreta, desenvolvidos na década de 70, iniciados com os trabalhos de COOK

em 1971 e de KARP em 1972. Esses estudos criaram o conceito que a grande maioria dos problemas de escalação era NP-completo. Essas conclusões levaram a uma busca mais sistemática de métodos de solução, que desencadearam dois caminhos paralelos de estudos, cada um deles gerando uma metodologia de pesquisa, conforme descrito a seguir.

Um dos ramos de pesquisa consistiu em estudar métodos heurísticos com desempenho garantido, a partir de 1976, através dos trabalhos de Garey e Johnson ((GAREY; JOHNSON, 1976, 1979), (GAREY et al., 1978)); Horowitz e Sahni ((SAHNI, 1976), (HOROWITZ; SAHNI, 1976), (SAHNI, 1977), (SAHNI; HOROWITZ, 1978), (HOROWITZ; SAHNI, 1984)), Lam e Sethi ((HORVATH et al., 1977), (LAM; SETHI, 1977)), entre outros, e métodos heurísticos com desempenho esperado, cujos principais trabalhos foram publicados a partir de 1984, tais como (COFFMAN et al., 1984), (COFFMAN; GILBERT, 1985), (BRUNO; DOWNEY, 1986), entre outros.

O outro ramo de pesquisa se caracterizou pelo uso de métodos enumerativos, cuja complexidade temporal é tempo-exponencial, mas que utilizados com uma solução inicial sub-ótima, gerada por um algoritmo heurístico, podem fornecer a solução ótima em tempos que poderiam ser proporcionais à complexidade tempo-polinomial. Esta técnica mista, apresentou como vantagem adicional, a possibilidade de não se necessitar buscar a resposta ótima, podendo-se delimitar a precisão da solução dentro de um desvio preespecificado. Os estudos a respeito deste assunto foram iniciados no meio da década de 70, a partir dos estudos de Kohler e Steiglitz (KOHLER; STEIGLITZ, 1974, 1975), (KOHLER, 1975), (COFFMAN, 1976).

2.5.4 Pesquisas Recentes sobre Escalação em Transputers

A busca de algoritmos escaladores para o Transputer tem sido realizada em outros centros de pesquisa, como por exemplo na Universidade de Berna (POUNTAIN, 1989). Nessa universidade estão sendo avaliados três estratégias de mapeamento de processos em Transputers. Uma delas, denominada "Adapter", utiliza um algoritmo de busca em grafo, gerando uma árvore de todos os mapeamentos possíveis e tentando minimizar, para cada um, os custos de comunicação interprocessadores e distribuir a carga computacional o mais equitativamente possível. Visto que esse é um algoritmo tempo-exponencial, sua aplicação se restringe a pequenas redes de Transputers.

As outras duas estratégias, denominadas "Evalconfig" e "Mapper" usam variantes de algoritmos probabilísticos e evolutivos. Algoritmos evolutivos recebem esse nome por se assemelharem à evolução darwiniana das espécies por seleção natural. Tais algoritmos encontram uma solução sub-ótima para o problema utilizando alguma estratégia heurística e tentam então melhorar essa solução, fazendo pequenas alterações aleatórias nas partes (mutações na analogia) e verificando se a solução geral piorou ou melhorou. Soluções melhores são conservadas e soluções piores são rejeitadas (sobrevivência do mais adaptado).

Algoritmos como esse podem produzir soluções excelentes em curtos intervalos de tempo. O fato da aleatoriedade estar envolvida implica que se pode chegar a diferentes soluções, embora próximas, a cada vez que o algoritmo é processado. Para um algoritmo evolutivo poder funcionar, ele deve estar apto a medir a qualidade de cada solução usando uma função-custo. Os dois

algoritmos em questão empregam uma função-custo que incorpora conceitos de balanceamento da carga computacional e de custo de comunicação interprocessadores.

No "Mapper", uma primeira solução é encontrada pela atribuição aleatória de processos a Transputers e deixando cada Transputer otimizar suas próprias conexões locais com seus vizinhos pela aplicação da função-custo. O mapeamento final produzido por essa soma de otimizações locais não irá, em geral, ser muito boa.

O algoritmo então comuta pares de processos e mede o efeito dessa mudança na qualidade da escalação. Tal comutação pode melhorar ou piorar a qualidade do mapeamento. Somente as comutações favoráveis são incorporadas. Quando todo par possível de processos foi comutado, considera-se a resposta obtida como definitiva.

Infelizmente, essa estratégia de comutação nem sempre converge suavemente para a melhor solução. A qualidade pode ter parado de progredir porque a melhor solução possível foi atingida, mas pode-se também ter alcançado um caminho evolutivo sem saída. Para escapar de tais situações, o algoritmo simula uma catástrofe ambiental em termos evolutivos, através da comutação de um número aleatório de processos, para gerar um novo mapeamento, o qual é muito provavelmente um passo a muito provavelmente um passo a uma evolução a partir desse ponto. O algoritmo se encerra se essa evolução adicional não produz nenhuma melhoria. O melhor mapeamento encontrado é então retornado como resultado.

Segundo os autores, o trabalho mais difícil consiste em melhorar a função-custo, que implica em medir mais precisamente a carga computacional e o volume de comunicação (POUNTAIN, 1989).

3 - AMBIENTE DISPONÍVEL DE HARDWARE E SOFTWARE

A plataforma de hardware a ser utilizada para implementar as escalasções geradas é constituída de Transputers, cuja linguagem original de programação é o Occam. Este capítulo visa fornecer características relevantes a respeito do par Transputer/Occam.

3.1 Estrutura Funcional de Multiprocessadores

Em multiprocessadores, os processadores podem se comunicar e cooperar em diferentes níveis para resolver um dado problema (HWANG; BRIGGS, 1988). A comunicação pode ocorrer através do envio de mensagens de um processador a outro ou através do compartilhamento de uma memória comum.

Dois diferentes conjuntos de modelos de arquitetura para multiprocessadores são possíveis (NI; WU, 1989), (HAYES, 1988):

- multiprocessadores fortemente acoplados, que se caracterizam pela comunicação através de uma memória compartilhada ou centralizada. Em sistemas desse tipo, a memória compartilhada é acessável por todos os processadores, em um único espaço de endereçamento global, sendo necessária uma rede de interconexão processador/memória para permitir o compartilhamento da memória física entre os processadores; e
- multiprocessadores fracamente acoplados, que se caracterizam pela comunicação através da troca de mensagens. Neste caso, a memória é distribuída, tanto a nível lógico quanto físico, e cada processador é fisicamente associado com uma memória local,

a qual é fisicamente separada e logicamente independente das memórias dos outros nós, não havendo um espaço global de endereçamento de memória. Um espaço global de endereçamento de memória não existe. Os processadores são interconectados através de uma rede de comunicação interprocessadores.

3.1.1 Multiprocessadores Fracamente Acoplados

A comunicação interprocessos em um multiprocessador fracamente acoplado ocorre através do roteamento de mensagens através de uma rede. A mensagem é a unidade lógica de troca de informações entre processos e o processo é a unidade abstrata usual em que uma computação é formulada, sendo parte de um programa. Um processo contém seu próprio código e variáveis particulares e coordena suas atividades com outros processos através do envio e recebimento de mensagens (ATHAS; SEITZ, 1988).

A via de comunicação provê meios para cada processador acessar dados ou código em processadores remotos, embora com um atraso maior e com uma largura de banda menor que para acessar dados e código locais. Sistemas típicos exibem larguras de banda de comunicação de 10 a 100 vezes menor que larguras de banda da memória de um nó.

Computadores implementados com arquitetura distribuída provêem execução distribuída de programas, no sentido que cada programa é dividido em tarefas e cada tarefa é atribuída a um processador. Conseqüentemente, um programa distribuído pode ser executado em paralelo. Se a largura de banda da via de comunicação é muito baixa comparada com a largura de banda da memória, o custo de comunicação entre tarefas se torna muito alto e anula os ganhos

atribuídos à execução paralela. Quando as larguras de banda das vias de comunicação são altas e se aproximam das larguras de banda da memória local, torna-se viável distribuir as tarefas entre os processadores (INDURKHYA et al., 1986).

Embora muitas linguagens de programação tenham sido propostas, dois enfoques gerais para expressar concorrência entre processos em multiprocessadores fracamente acoplados podem ser distinguidos (ATHAS; SEITZ, 1988):

- estrutura estática de processos, em que o número de processos e sua conectividade são conhecidos antes do programa ser executado. Em Occam, por exemplo, todos os processos e canais para uma computação são identificados no programa fonte. Cada canal conecta exatamente dois processos e a comunicação através dos canais é síncrona, isto é, o término da ação de enviar uma mensagem em um extremo do canal é sincronizada com o término da ação de recebimento no outro extremo; e
- estrutura dinâmica de processos, em que os mesmos são criados e destruídos sob demanda.

3.2 Conceitos sobre a Linguagem de Programação Occam

Occam é a linguagem de programação originalmente utilizada pelos Transputers, sendo que atualmente já há outras linguagens de alto nível disponíveis, como por exemplo C, Pascal, Fortran, etc. É possível diferentes partes da mesma aplicação serem implementadas em diferentes linguagens de alto nível e fazer com que elas se comuniquem.

Occam incorpora as duas importantes funções de sistemas concorrentes: processamento paralelo de atividades independentes e comunicação entre essas atividades (TAYLOR; WILSON, 1982).

Occam permite que um sistema seja expresso através de processos concorrentes, que na maior parte do tempo operam independentemente. Processos concorrentes se comunicam através de mensagens enviadas através de canais de comunicação. A comunicação é sincronizada e ocorre quando um processo de entrada e de saída estão ambos prontos para se comunicar no mesmo canal. Occam pode ser usado para programar um único ou uma rede de Transputers, cada um executando um ou mais processos.

3.2.1 Processos em Occam

O bloco construtivo de um programa em Occam é o processo. Os sistemas são projetados em termos de um conjunto interconectado de processos. Cada processo pode ser encarado como uma unidade independente de projeto (INMOS, 1988c).

Cada processo pode ser visualizado como uma "caixa preta", que pode se comunicar com outros processos usando canais de comunicação ponto-a-ponto. A comunicação entre processos é sincronizada, dispensando qualquer mecanismo separado de sincronização, sendo que as próprias mensagens servem como sinais sincronizadores (DAS; FAY, 1988).

Os processos são finitos, no sentido que cada processo começa, executa algumas ações e termina. Uma ação pode ser um conjunto de processos seqüenciais, como em uma linguagem convencional de programação ou um conjunto de processos paralelos (executados

simultaneamente). Visto que um processo é composto de outros processos, alguns dos quais podem ser executados em paralelo, resulta que um processo pode conter qualquer volume de concorrência interna e isso pode variar com o tempo, conforme os processos começam e terminam (INMOS, 1988c).

Em última instância, todos os processos em Occam são construídos a partir de três processos primitivos: atribuição, entrada e saída. Uma atribuição computa o valor de uma expressão e o transfere para uma variável. Entrada e saída são usadas para comunicação entre processos.

3.2.2 Comunicação e Sincronização

Um par de processos concorrentes se comunica usando um canal unidirecional que os conecta, jamais utilizando variáveis compartilhadas. Um processo envia uma mensagem para o canal e o outro recebe a mensagem desse mesmo canal. O canal é o elemento central nos processos de entrada e saída, sendo o elemento básico de comunicação que permite que processos concorrentes se comuniquem. Um canal é uma estrutura não "bufferizada" que conecta dois processos. Ele se comporta como um elemento de simples leitura para um processo receptor e como um elemento de simples escrita para um processo transmissor (TAYLOR; WILSON, 1982). Toda a comunicação interprocessos ocorre através de mensagens e o Occam implementa o paralelismo entre processos baseado nesse conceito de comunicação (DAS; FAY, 1988).

Quando dois processos concorrentes em Occam se comunicam, o seguinte procedimento é seguido: o processo que está preparado

para transferir dados aguarda até que o processo receptor indique estar pronto para receber. Quando o receptor está pronto, os dados são transferidos. Uma situação similar acontece se o receptor estiver preparado para receber antes do transmissor estar pronto. Conseqüentemente, o processo que fica pronto primeiro, deve esperar até que o outro também fique pronto. Uma vez que um processo pode ter concorrência interna, ele pode ter muitos canais de entrada e saída executando comunicação "simultaneamente" (INMOS, 1988c).

3.2.3 Construções em Occam

Os processos podem ser combinados para formar uma construção, a qual é, em si mesma, um processo e pode, portanto, ser usada como componente de outra construção. Há diversos tipos de construções, a saber: seqüencial (SEQ), paralela (PAR), condicional (IF), alternativa (ALT), malha (WHILE), seleção (CASE) e replicação, a qual é usada com uma construção SEQ, PAR, IF ou ALT para replicar o processo componente um certo número de vezes.

Para este trabalho, as construções que apresentam maior interesse são detalhadas a seguir:

a) Construção seqüencial

É uma construção composta por processos P_i , $1 \leq i \leq n$, que são executados seqüencialmente na ordem estabelecida. Construções seqüenciais em Occam são similares a programas escritos em linguagens convencionais de programação.

A construção seqüencial é representada por:

```

SEQ
  P1
  P2
  P3
  .
  .
  .

```

b) Construção paralela

É uma construção composta por processos P_i , $1 \leq i \leq n$, que são executados em paralelo. A construção paralela é exclusiva do Occam, correspondendo a operar com paralelismo explícito em uma linguagem de programação. Este recurso provê uma maneira direta de escrever programas que reflitam a concorrência existente em sistemas reais.

Uma construção paralela é representada por:

```

PAR
  P1
  P2
  P3
  .
  .
  .

```

3.2.4 Alocação de Dados em Memória

Buscando eficiência em tempo de execução, o compilador Occam aloca processadores e memória de forma estática, em tempo de compilação. Assim, o Transputer não necessita implementar a alocação dinâmica de memória (INMOS, 1989a).

Para alcançar o máximo desempenho de um Transputer é importante utilizar criteriosamente a memória embutida. Por exemplo, no Transputer modelo IMS T800-20, usado como base para este trabalho, os ciclos internos de memória são de 50 ns, enquanto os ciclos de memória externa são de 100 ns. Esse fator

de degradação de dois na velocidade de acesso à memória pode se refletir no desempenho do programa, se endereços de memória freqüentemente acessados estiverem na memória externa (INMOS, 1989b).

A memória embutida é melhor empregada para valores escalares e ponteiros, ao invés de código e matrizes de dados. O Transputer busca instruções na memória em palavras de 32 bits. Como a maioria das instruções é de um byte, cada busca de código na memória trará normalmente quatro instruções. Portanto, acessos à memória em busca de código geralmente ocorre menos freqüentemente que acessos em busca de dados. Todo acesso a uma estrutura de dados requer dois ou mais valores escalares e ponteiros para determinar o endereço de um componente da matriz (INMOS, 1989b).

O compilador Occam e o programa carregador do Transputer tentam colocar valores escalares e ponteiros na memória embutida do Transputer. As três seguintes áreas de armazenamento são alocadas, a partir do endereço livre mais baixo na memória embutida:

- área que contém os espaços de trabalho dos processos, que é normalmente colocada na memória embutida;
- área que contém o código do programa e é colocada acima dos espaços de trabalho dos processos, sendo que sua maior parte é alocada na memória externa; e
- área que contém as matrizes, a qual é quase sempre colocada na memória externa.

A figura 3.1 mostra o arranjo de ocupação de memória em programas escritos em Occam (INMOS, 1989b).

Fig. 3.1 Arranjo de ocupação da memória em programas Occam

A fim de comparar o arranjo mostrado na figura 3.1 com o endereçamento utilizado pelo compilador Occam, apresenta-se a figura 3.2 (INMOS, 1989c).

Endereço	Conteúdo	Observação
#0400		Início da memória externa
#1C		Início da memória disponível ao usuário
#00		Base da memória

Fig. 3.2 Mapa de memória utilizado pelo compilador Occam

3.3 Características Relevantes do Transputer

Um Transputer é um microprocessador com memória local própria e com vias de comunicação para conectar um Transputer a outro. Ele pode ser usado para implementar sistemas monoprocesados ou multiprocessados fracamente acoplados.

3.3.1 Arquitetura dos Transputers

A família dos Transputers é composta pelos componentes T212, T414 e T800, cujas características básicas são mostradas na tabela 3.1 (INMOS, 1988c).

CARACTERÍSTICA	MODELO			
	T212	T414	T800-20	T800-30
Frequência de operação (MHz)	20	20	20	30
Tamanho do barramento (bits)	16	32	32	32
Memória SRAM embutida (50 ns)	2 kB	2 kB	4 kB	4 kB
Espaço de memória diretamente endereçável	64 kB	4 GB	4 GB	4 GB
Velocidade de processamento (MWhet/s)	Não consta	0,663	4,0	6,0
Desempenho da Unidade de Ponto Flutuante (MFLOPS)	Não possui	Não possui	1,50	2,25

Tab. 3.1 Características básicas dos Transputers

3.3.2 Concorrência e Comunicação

A comunicação entre Transputers ocorre através de suas vias (links) de comunicação ponto-a-ponto. Todo Transputer possui quatro vias bidirecionais de comunicação, que provêem conexão com até quatro outros Transputers. Cada uma das quatro vias compreende dois canais unidirecionais de comunicação para transmissão serial de dados, que correspondem a dois canais em Occam, um em cada direção. O Transputer se comunica enviando mensagens como uma sequência de bytes através do canal de saída de uma via, sendo que cada byte deve ser individualmente reconhecido antes do próximo ser transmitido.

A comunicação pode ocorrer simultaneamente nas quatro vias e em ambas as direções. As vias de comunicação dos Transputers provêm a taxa de comunicação padrão de 10 Mbits/s. Além disso, eles podem ser ajustados para operar a taxas de 5 ou 20 Mbits/s, independentemente do modelo ou frequência de operação do Transputer. Assim, Transputers diferentes podem ser diretamente conectados. As taxas de transmissão de dados efetivamente alcançadas pelos Transputers são mostradas na tabela 3.2 (INMOS, 1988c).

Modelo	Taxa de transmissão de dados (kbytes/s)					
	Unidirecional			Bidirecional		
	5Mbits/s	10Mbits/s	20Mbits/s	5Mbits/s	10Mbits/s	20Mbits/s
T212	200	400	800	400	800	1600
T414	200	400	800	400	800	1600
T800	450	910	1740	670	1250	2350

Tab. 3.2 Taxas efetivas de troca de dados entre Transputers

Todo Transputer implementa os conceitos de concorrência e comunicação do Occam. Processos concorrentes em Transputers se comunicam exclusivamente através de canais (INMOS, 1988c). Como resultado, o Occam pode ser usado para programar um único Transputer ou uma rede deles.

Quando o Occam é usado para programar um único Transputer, o Transputer compartilha seu tempo entre os processos concorrentes e a comunicação através dos canais é implementada movendo dados dentro da memória RAM, tanto a embutida quanto a externa, através de uma única palavra. Tem-se, nesse caso, um canal interno. Quando o Occam é usado para programar uma rede de Transputers, cada Transputer executa os processos que lhe são alocados. A comunicação entre processos em diferentes Transputers ocorre através de canais externos, implementados pelas vias de comunicação do Transputer.

O processador dispõe das instruções de entrada e saída para prover comunicação interprocessos. A transferência de mensagens entre processos concorrentes é sumarizada da seguinte forma em Occam (INMOS, 1988a):

```
PAR
  canal ! msg1
  canal ? msg2
```

Este trecho de programa equivaleria à seguinte atribuição em Occam:

```
msg2:=msg1,
```

onde msg1 e msg2 podem corresponder a mensagens constituídas de dados escalares ou vetoriais, compostas de qualquer tipo de dado disponível no Occam.

As instruções de E/S usam o endereço do canal para determinar se o canal é interno ou externo. Isto significa que a mesma sequência de instruções pode ser usada para canais internos ou externos, permitindo que um processo seja escrito e compilado sem prévio conhecimento de como seus canais serão conectados. Assim, o mesmo programa em Occam pode ser implementado em uma variedade de configurações de Transputers, inclusive em um único processador.

Mesmo que a aplicação utilize somente um Transputer, o programa pode ser projetado como um conjunto de processos concorrentes que poderia ser processado em um número maior de Transputers. Esse estilo de projeto segue a metodologia da programação estruturada: os processos operam independentemente com suas próprias variáveis, exceto onde eles explicitamente interagem, através de canais.

Há uma diferença no tratamento entre a comunicação usando canal interno e externo para processos de baixa prioridade. Na comunicação utilizando canal interno, o processo que fica pronto primeiro é desescalado para esperar o outro processo, permitindo ao processador executar outros processos que não estão esperando por comunicação, de forma a não consumir tempo do processador. Quando o segundo processo fica pronto, ocorre a transferência da mensagem, o processo que havia sido desescalado é colocado na lista de espera e o segundo processo a ficar pronto continua seu processamento. No caso da mensagem ser passada através de um canal externo, o primeiro processo a ficar pronto é desescalado. Ao ficar pronto o segundo processo, ele também é desescalado e ocorre, então, a transferência da mensagem. Em seguida, ambos os

processos são recolocados na fila de espera de seus respectivos processadores.

3.3.3 Escalador Interno

O Transputer possui um Escalador Interno em microcódigo, que compartilha o tempo do processador entre os processos concorrentes. O Escalador Interno provê dois níveis de prioridade: qualquer processo de alta prioridade que possa ser processado o será, em detrimento de qualquer processo de baixa prioridade.

Em qualquer instante, um processo concorrente pode estar nos seguintes estados:

- ativo: quando está sendo executado ou está em uma lista de espera para ser executado; e
- inativo: quando está pronto para entrada de dados, para saída de dados ou esperando até um tempo especificado.

O Escalador Interno opera de forma que processos inativos não consomem nenhum tempo do processador, pois aloca uma porção de tempo do processador a cada processo ativo. Processos ativos, esperando para ser executados, são mantidos em duas listas ligadas: uma dos processos de alta prioridade e outra dos processos de baixa prioridade.

A formação das listas, dentro de cada um dos dois níveis de prioridade, é feita de acordo com a ordem de chegada dos processos, de forma que quando um processo é tornado ativo, ele é adicionado ao fim da lista de espera de seu respectivo nível de

prioridade, constituindo, portanto, uma fila de espera, o que impede que processos de mesmo nível causem mútua preempção.

A preferência na escalação é dada aos processos contidos na fila de alta prioridade. Os processos contidos na fila de baixa prioridade somente são executados quando não há processos ativos de alta prioridade.

Um processo de alta prioridade é desescalado quando ocorre um dos seguintes eventos:

- sua ação é completada;
- quando tem que esperar por comunicação com outro processo; ou
- por estar aguardando por um atraso de tempo gerado pelo temporizador embutido nos Transputers.

Ao assumir o processador, um processo de baixa prioridade pode ser desescalado por uma das três causas apontadas ou ainda pela utilização contínua do processador por um certo intervalo de tempo.

Quando um processo de alta prioridade se torna ativo enquanto um processo de baixa prioridade está sendo executado, o processo de baixa prioridade é apenas interrompido, não sendo colocado no final da fila de espera. O processo de baixa prioridade reassume o controle da UCP do Transputer tão logo o de alta prioridade conclua sua execução (INMOS, 1988b).

Para que diversos processos operem em paralelo, um processo de baixa prioridade somente pode assumir o processador por um máximo de dois "intervalos" de tempo, antes de ser forçosamente desescalado no próximo ponto de desescalação. Cada intervalo de

tempo consiste em 5120 ciclos do relógio externo de 5 MHz, o que corresponde a cerca de 1 ms.

Como resultado dessa política de desescalação, assegura-se que nenhum processo monopolize o Transputer.

3.3.4 Configuração para Multiprocessamento

Occam permite expressar um problema como um certo número de processos concorrentes $P_1, P_2, \dots, P_n$, que são executados separadamente, mas que podem interagir entre si. O alto grau de correspondência entre o Occam e o Transputer faz com que qualquer problema que possa ser definido por um programa em Occam em termos de processos possa ser processado em um sistema com 1, 2 ... até n Transputers.

Se menos que n Transputers forem usados, dois ou mais processos residirão em um único Transputer e o fluxo de dados entre eles será interno. Isto é transparente ao programador, exceto que, no início de um programa em Occam, uma seqüência de instruções de configuração é requerida para determinar o número de Transputers no sistema, os canais usados para o fluxo de dados e a distribuição dos processos nos Transputers. Sendo assim, a alocação dinâmica de processos em Transputers não é possível de ser executada, quando se utiliza o Occam.

Uma construção paralela pode ser configurada para uma rede de Transputers pelo uso da construção PLACED PAR, a qual substitui a construção PAR, aplicável a um único processador. Nesse caso, cada processo componente é executado por um Transputer diferente.

Em cada Transputer individual, a construção paralela mais externa pode ser configurada para priorizar seus componentes, através da construção PRI PAR, em que o primeiro processo tem a prioridade mais alta e o segundo a mais baixa. Convém observar que na versão atual do Transputer/Occam, só se permite a atribuição de dois níveis de prioridade: alto e baixo.

3.4 Processamento de Tarefas em Transputers

As principais fontes de consulta sobre Transputers, disponíveis (INMOS, 1988b, 1988c, 1989a, 1989b, 1989c), são omissas ou insuficientemente claras sobre certas informações a respeito do modo de operação desses processadores, originando a necessidade de pesquisas adicionais, com a realização de experimentos práticos com os Transputers.

Para a realização dos experimentos, optou-se pelo modelo IMS T800, por ser o mais completo à época da execução deste trabalho.

Os seguintes tópicos abordados nesta seção, foram estudados em busca de esclarecimentos adicionais:

- modo de operação do Escalador Interno;
- alterações da velocidade de processamento de um processo em função do regime em que o mesmo opera; e
- intervalos de tempo consumido pelo Transputer em algumas operações.

3.4.1 Escalador Interno

A principal característica que diferencia a escalação em Transputers de outros casos de escalação estática de tarefas é a possibilidade de executar processamento paralelo dentro de cada um dos processadores da rede, através da aplicação da construção PAR. Surge então uma dificuldade inexistente em outros casos de escalação, que é a presença do Escalador Interno para gerenciar o processamento das tarefas colocadas em PAR.

Para que diversos processos operem em paralelo em um único Transputer, um processo pode utilizar o processador continuamente, antes de ser obrigatoriamente desescalado, até que uma das situações mencionadas no item 3.3.3 ocorra. O responsável por cuidar da escalação/desescalção de processos em um Transputer é o seu Escalador Interno, o qual executa compartilhamento do tempo (time-sharing) entre os processos em PAR.

O Escalador Interno, programado em microcódigo e inacessível aos usuários, tem características próprias, que o colocam dentro da seguinte classificação:

- dinâmico: a atribuição de tarefas ao processador é feita em tempo de execução;
- determinístico: a desescalção de tarefas por exceder seus quanta de tempo disponíveis ocorre pela contagem efetiva dos intervalos de tempo que as mesmas consomem do processador;
- fixo: o Escalador Interno somente consegue atribuir os processos ao próprio Transputer em que está localizado; e

- preemptivo: os processos são desescalados em função de solicitação de E/S, espera de contagem de tempo do temporizador interno e término de seus quanta de tempo, sendo que esta última causa só se aplica a processos de baixa prioridade.

O Programa Escalador deve reproduzir as ações do Escalador Interno e, para tanto, ele necessita assimilar, de forma completa, o funcionamento do Escalador Interno. Portanto, o que se propõe é que o Programa Escalador modele o Escalador Interno, para poder simular a operação em processamento paralelo executada por um único Transputer.

As principais questões a serem elucidadas, acerca do modo de operação do Escalador Interno, são as seguintes:

- como é feita a seqüência de alocação dos processos paralelos, listados na construção PAR, ao processador? Seguindo a mesma ordem em que eles aparecem em PAR?
- como são compartilhados os quanta de tempo entre processos paralelos?
- quanto tempo recebe o primeiro processo escalado?

Para poder responder a essas e outras questões, foram realizadas diversas baterias de testes, cujos resultados estão sumarizados a seguir.

Convenciona-se tratar aqui o tempo em ticks dos temporizadores internos, que são as unidades de tempo disponíveis nas medições com os Transputers. Para processos de alta prioridade, cada tick equivale a 1 microsegundo e para processos de baixa prioridade a 64 microsegundos. Quando nada for mencionado no texto, tratar-se-á de ticks não prioritários, referentes a processos de baixa

prioridade. Quando se desejar indicar ticks prioritários, relativos a processos de alta prioridade, será empregada a sigla TP.

a) Ordem de escalação de processos em PAR

Em processos paralelos residentes em um mesmo Transputer, arrançados dentro da construção PAR, o último processo da fila do PAR é o primeiro a ser escalado. Daí em diante, a seqüência de escalação segue a ordem de aparecimento dos processos dentro do PAR. O mesmo se aplica a processos em PAR dentro de PRI PAR (quer sejam prioritários ou não).

b) Compartilhamento dos quanta de tempo entre processos em PAR

O contador de pulsos do Escalador Interno nunca é zerado, sempre contando até 5120 pulsos do relógio externo de 5 MHz para formar um quantum de tempo, que equivale a 16 ticks do temporizador de baixa prioridade. Cada processo de baixa prioridade tem direito a até dois quanta de tempo antes de ser desescalado por excesso de tempo.

Se um dado processo de baixa prioridade for escalado e a sua duração for menor ou igual a 32 ticks (dois quanta de tempo) o próximo processo de baixa prioridade a ser escalado completa o quantum de tempo deixado pelo processo anterior e tem direito a um segundo quantum de tempo, ocorrendo o compartilhamento de um quantum de tempo entre eles. Constata-se que qualquer processo de baixa prioridade tem direito a assumir o processador continuamente por, no mínimo, 17 e no máximo 32 ticks.

Por outro lado, se um processo de baixa prioridade está sendo processado em PAR e no final de seus quanta de tempo não há nenhum outro processo na fila de espera, ele recebe um quantum adicional de tempo. Ainda que durante esse quantum de tempo chegue um novo processo na fila de espera, o processo escalado continua nessa situação até o final do quantum adicional de tempo que ele recebeu.

c) Atuação do Escalador Interno perante o primeiro processo escalado em PAR

Ao iniciar o processamento do primeiro processo em PAR, não se sabe como está a contagem para o quantum de tempo. Como cada processo pode assumir continuamente o processador por até dois quanta de tempo, significa que o primeiro processo a ser escalado poderá ter disponível para si de 5.121 a 10.240 ciclos do relógio, correspondendo a 16 a 32 ticks do temporizador de baixa prioridade. Isto implica que processos com duração de 16 ticks ou menos serão processados por inteiro, mas processos com duração superior a 16 ticks poderão ser interrompidos para uma posterior continuação.

Para poder simular o funcionamento do Escalador Interno, resolveu-se adotar uma abordagem determinística do problema, trabalhando-se com valores médios. Isto significa dizer que, como o tempo mínimo de processamento é 16 ticks e o máximo é 32, optou-se pelo valor médio de 24 ticks disponíveis para o primeiro processo escalado. Com isso, qualquer processo com tempo de processamento menor ou igual a 24 ticks e que for o primeiro a ser escalado em PAR, será processado por inteiro. Processos com tempo de processamento maior que 24 ticks serão

interrompidos e continuarão posteriormente, quando forem reescalados. Este enfoque gera um erro máximo de $\pm 0,5$ ms, correspondente a ± 8 ticks.

Deve-se notar que essa hipótese do tempo de processamento do primeiro processo a ser escalado não afeta em absoluto o tempo total de processamento dos n processos em paralelo, pois, no final, tanto a escalação aqui preconizada quanto a escalação real realizada pelo Transputer, terminarão no mesmo tempo. O que pode ser afetado é algum processo residente em outro Transputer, que se comunique com o primeiro processo escalado mencionado acima. Nesse caso, o primeiro processo escalado poderá ter uma mensagem a ser transferida via canal externo, que poderá afetar a temporização dos processos em outro Transputer.

3.4.2 Alteração da Velocidade de Processamento em PAR e PRI PAR

Foram efetuados diversos testes para comparar a velocidade de processamento de um processo em SEQ, PAR e PRI PAR, com os resultados descritos a seguir.

a) Velocidade de processamento em PAR

Através da análise dos resultados obtidos nas baterias de testes, constatou-se o seguinte:

- o número de processos em paralelo na construção PAR não afeta seu tempo de processamento. Assim, é indiferente ter 2, 3, 4 ou mais módulos em PAR, que sua duração individual será sempre a mesma;

- quando resta apenas um processo em PAR, pois todos os outros já encerraram seu processamento, seu tempo de processamento permanece o mesmo que teria caso estivesse sendo processado com outros processos simultaneamente;
- há uma alteração na velocidade de processamento de cada tarefa, quando comparado com o tempo de processamento em SEQ;
- a alteração na velocidade de processamento do regime PAR em relação ao SEQ é função do tipo de operações efetuadas nas tarefas, sendo impossível, portanto, se estabelecer uma relação fixa entre esses tempos; e
- o tempo de processamento de tarefas alocadas na memória RAM embutida do Transputer difere do tempo de processamento de tarefas alocadas na memória RAM externa.

b) Velocidade de processamento em PRI PAR

Estes testes são necessários para verificar a velocidade de processamento das tarefas de baixa prioridade que serão processadas em paralelo com os processos de alta prioridade, responsáveis pela comunicação entre Transputers.

Realizados os testes, concluiu-se que:

- dentro do PRI PAR, tarefas prioritárias são processadas com a mesma velocidade que tarefas não prioritárias;
- a utilização de PAR dentro de PRI PAR leva a uma redução de velocidade de processamento das mesmas tarefas em PAR;

- a razão dos tempos de processamento entre as tarefas em PAR dentro de PRI PAR e das tarefas em PAR não segue uma relação fixa; e
- o tempo de processamento de tarefas processadas em PAR é o mesmo que em SEQ dentro de PRI PAR.

3.4.3 Análise de Intervalos de Tempo Relevantes para o Programa Escalador

Este item analisa alguns tópicos relevantes para o modelamento do modo de operação de uma rede de Transputers, relacionados com intervalos de tempo dispendidos pelos Transputers em certas operações.

a) Comunicação entre Transputers através de nós intermediários

O objetivo deste sub-item é avaliar o tempo que uma mensagem consome para ir de um Transputer a outro, no caso deles não estarem diretamente interligados. Uma primeira opção para definir esse tempo seria tentar encontrar equações que relacionassem o tempo de transferência de mensagens através da rede com o número de nós envolvidos, o tipo de mensagem (escalar ou vetorial), o comprimento da mesma (no caso vetorial) e o tipo de dado constituinte da mensagem. Este método esbarra no problema que para velocidades diferentes das vias de comunicação dos Transputers haveria necessidade de ajustar as equações, de uma forma difícil de ser quantificada.

Uma segunda opção, independente dos parâmetros mencionados, seria relacionar o tempo de trânsito das mensagens pelos nós intermediários com algum intervalo de tempo diretamente

mensurável pelo usuário do Programa Escalador, de forma a se conseguir estabelecer o tempo de trânsito das mensagens como função apenas do número de nós intermediários pelos quais a mensagem deve passar e do intervalo fornecido pelo usuário. A definição dessa função é importante porque, ao efetuar as escalações, o Programa Escalador poderá alocar os processos em qualquer Transputer, de forma que, para cada caso, os tempos de trânsito variarão de acordo com o número de nós intermediários.

Os testes revelaram que, independentemente do tipo de dado constituinte da mensagem, a seguinte equação é aplicável:

$$\tau_{TM} = (m_i + 1) * \tau_{DM} \quad (3.1)$$

onde: τ_{TM} = tempo de trânsito da mensagem na rede;

m_i = número de nós intermediários; e

τ_{DM} = tempo de duração da transmissão da mensagem.

Esta equação será usada pelo Programa Escalador para calcular os tempos de trânsito de mensagens entre Transputers indiretamente interligados por nós intermediários da rede.

A equação (3.1) se aplica somente se todas as vias de comunicação dos Transputers estiverem ajustadas para a mesma velocidade.

b) Tempo de comunicação interna e externa

Constata-se que, independentemente do processo estar na construção SEQ ou PAR dentro ou fora de PRI PAR, a duração das mensagens (τ_{DM}) é função exclusiva delas serem internas ou externas, onde interna e externa significa a utilização de

canais internos ou externos, respectivamente, sendo que a mesma mensagem consome menos tempo na comunicação interna que na externa. Dessa forma, pode-se afirmar que há necessidade de medir apenas dois tempos de duração de mensagens: interno (τ_{DMINT}) e externo (τ_{DMEXT}).

c) Tempo dispendido no chaveamento de contexto

Sempre que um determinado processo deixa a UCP do Transputer e um outro processo a assume, há um certo tempo envolvido nessa operação.

Os testes indicaram que o tempo de chaveamento de contexto é de cerca de $0,9 \mu s$. ao passo que na literatura técnica menciona-se que esse tempo é da ordem de $1 \mu s$ (INMOS, 1988c). Devido a seu baixo valor, esse tempo é desprezado na simulação dos Transputers.

3.5 Roteamento de Mensagens

Tendo em vista que a troca de mensagens entre Transputers não diretamente interligados da rede ocorre através de nós intermediários, constituídos pelos próprios Transputers, surge a necessidade de se criar um processo a ser instalado em cada nó intermediário, que execute o roteamento da mensagem, isto é, receba a mensagem que chega e determine para onde ela deve ser encaminhada.

Para não haver atrasos no roteamento das mensagens, o processo roteador deve ser prioritário, o que assegura que quando uma mensagem chega a um nó intermediário, ela seja imediatamente

recebida e examinada. Se aquele for seu destino, ela é retida; caso contrário, ela é passada adiante através de uma das vias de comunicação e assim prossegue até alcançar seu processador destino.

O Processo Roteador recebe uma mensagem de uma via de comunicação ou de um processo corresidente no Transputer, examina a mensagem, buscando, por exemplo, na primeira palavra, o destino da mesma e verifica, em uma tabela de rotas, se essa mensagem deve ser retida para uso por algum processo residente nesse nó intermediário ou se deve ser retransmitida por alguma das vias de comunicação (INMOS, 1989a).

Embora o Processo Roteador represente uma carga computacional extra para cada nó, uma vez que uma transferência de dados tenha sido iniciada, as vias de comunicação autônomas dos Transputers se encarregam da transferência, sem intervenção adicional do processador. Baseado nesse fato, desprezar-se-ão, neste trabalho, os atrasos no processamento das tarefas causados pela passagem de mensagens em trânsito pelos nós intermediários da rede de Transputers.

3.5.1 Processos Roteador e Distribuidor

Para poder validar o modelamento da rede simulada de Transputers, houve necessidade de se criar um processo roteador, o qual se baseou no esquema proposto no capítulo 4 de (INMOS, 1989a).

Além do Processo Roteador, verificou-se a necessidade de se desenvolver um processo distribuidor, responsável por receber e

armazenar as mensagens destinadas a processos residentes em cada Transputer da rede. Todo Transputer deve receber o Processo Roteador e o Processo Distribuidor, os quais operam no modo PAR em regime prioritário.

Com esses dois processos operando em conjunto, cada processo que deseje enviar uma mensagem é prontamente atendido pelo Processo Roteador. Mesmo que a mensagem seja destinada a um processo residente no próprio Transputer do processo transmissor, a mensagem deve passar pelo Processo Roteador. Com essa facilidade, o processo transmissor não precisa aguardar sincronização com o processo receptor, bastando enviar a mensagem e o Processo Roteador imediatamente a recebe. Dessa forma, pode-se colocar em SEQ, dentro do mesmo processador, tarefas predecessoras e sucessoras, nessa ordem, que o Processo Roteador/Distribuidor se encarregará de prover a comunicação entre elas.

Esta possibilidade propiciou um grande poder à escalação em redes de Transputers, pois pode-se escalar qualquer processo em qualquer regime de operação em qualquer processador, respeitando apenas que as tarefas sucessoras sejam escaladas após suas tarefas predecessoras no regime SEQ.

Um esquema de inserção dos processos Roteador e Distribuidor dentro do programa a ser instalado em cada Transputer da rede é mostrado a seguir:

```
PRI PAR
  PAR
    Processo Roteador
    Processo Distribuidor
  SEQ
    Processos de baixa prioridade
```

A título de ilustração, apresenta-se na figura 3.3 a codificação em Occam-2 dos processos Roteador e Distribuidor.

```

PROC demo_proc_rot_distr ([4]CHAN OF ANY via.ent, via.saida)
-- Declarações ...
[100] INT rota.externa:
-- A declaração acima supõe que a rede é composta de 100 Transputers
SEQ
-- Inicializações ...
-- Tabela de rotas externas (rota.externa[transp]:=rota)
rota.externa[0]:=1
rota.externa[1]:=4
.
.
.
rota.externa[m-1]:=2
-- rota=0, 1, 2 e 3 corresponde a rotas externas
-- rota=4 corresponde a rota interna (Transputer destino da
-- mensagem=Transputer origem da mensagem)
CHAN OF ANY interno.ent:
[100000] CHAN OF ANY canal.interno:
PRI PAR
[100000] CHAN OF ANY interno:
-- As duas declarações acima pressupõem que haja até 1000
-- processos com 100 mensagens recebidas em cada
PAR
-- *****
-- PROCESSO ROTEADOR
-- *****
INT destino, transp, proc.seq, rota:
[100] INT mensagem:
-- A declaração acima indica que as mensagens recebidas serão
-- vetores com dimensão 100 compostos de números inteiros
WHILE ativo
SEQ
ALT
ALT i=0 FOR 4
via.ent[i] ? mensagem
SKIP
interno.ent ? mensagem
SKIP
destino:=mensagem[0]
transp:=destino/100000
rota:=rota.externa[transp]
IF
rota < 4 -- significando rota externa
via.saida[rota] ! mensagem
rota = 4
SEQ
proc.seq:=destino-(transp*100000)
interno[proc.seq] ! mensagem

```

Fig. 3.3 Programa em Occam-2 descrevendo os processos Roteador e Distribuidor

```

-- *****
-- PROCESSO DISTRIBUIDOR DAS MENSAGENS INTERNAS
-- *****
PAR
--   Para processo abc, mensagem número xy
--   [100] INT mens:
--   SEQ
--       interno[abcxy] ? mens
--       canal.interno[abcxy] ! mens
--   Para processo def, mensagem número wz
--   [100] INT mens:
--   SEQ
--       interno[defwz] ? mens
--       canal.interno[defwz] ! mens
--       .
--       .
--       .
--   Até terminar todas as mensagens recebidas por este Transputer
-- *****
-- PROCESSOS DE BAIXA PRIORIDADE
-- *****
-- Declarações ...
-- [100] INT msg, msgpppnn:
-- SEQ
--   Exemplo de transmissão de mensagem p/ Transputer tt, processo
--   ppp, mensagem número nn, dados inteiros
--   msg[0]:=tppppnn
--   msg[1]:=dado1
--       .
--       .
--       .
--   msg[99]:=dado99
--   interno.ent ! msg
--   Exemplo de recepção de mensagem para processo ppp, mensagem
--   número nn, dados inteiros
--   canal.interno[pppnn] ? msgpppnn
--   Continuação dos processos de baixa prioridade
--   :

```

Fig. 3.3 Programa em Occam-2 descrevendo os processos Roteador e Distribuidor (cont.)

3.5.2 Cabeçalho das Mensagens

Cada mensagem deve conter um cabeçalho constituído de um número inteiro, que represente o destino da mesma, identificando o Transputer destino, o processo destino e a mensagem dentro do

processo destino. Esta última informação visa distinguir diferentes mensagens que chegam para o mesmo processo em ordem aleatória.

Para utilizar os processos Roteador e Distribuidor, o formato previsto do cabeçalho é mostrado na figura 3.4.

onde:

t contém o número de identificação do Transputer destino

p contém o número de identificação do processo destino

n contém o número sequencial da mensagem recebida no processo receptor

Fig. 3.4 Formato do cabeçalho das mensagens

3.5.3 Influência dos Processos Roteador/Distribuidor nos Tempos de Processamento

O Programa Escalador pressupõe que todo Transputer da rede tenha um processo Roteador/Distribuidor instalado nele. Portanto, os processos do usuário do Programa Escalador deverão ser processados em baixa prioridade, seguindo a escalação proposta pelo Programa Escalador, de acordo com o seguinte esquema:

```

PRI PAR
  Processo Roteador/Distribuidor
  SEQ
    SEQ/PAR
    Processos escalados
  
```

Os tempos de processamento resultantes dos processos não prioritários em SEQ ou em PAR não são afetados se esses mesmos

processos são colocados em SEQ ou em PAR dentro de SEQ. Assim, a título de exemplo, os tempos de processamento de P1 e P2 não se alteram nos casos mostrados na figura 3.5.

PRI PAR Proc Rot/Dist SEQ P1	PRI PAR Proc Rot/Dist PAR SEQ P2 SEQ Processo comunicante	PRI PAR Proc Rot/Dist SEQ SEQ P1 PAR SEQ P2 SEQ Processo comunicante
---------------------------------------	--	--

Fig. 3.5 Exemplo para comparar tempos de processamento de tarefas

Os dois primeiros arranjos da figura 3.5 correspondem à forma como o usuário deverá coletar os tempos de processamento das tarefas. O terceiro arranjo mostra as tarefas distribuídas em uma escalação qualquer.

O tempo dispendido por P1 e P2 é qualquer um dos arranjos. Isto valida a idéia de medir os tempos em SEQ e PAR de um processo isolado junto com o Processo Roteador/Distribuidor e utilizá-los na simulação do Programa Escalador, onde as tarefas são processadas em conjunto.

3.5.4 Tempos Envolvidos no Roteamento de Mensagens

O tempo gasto no roteamento de mensagens ocorre nas seguintes situações:

- tempo dispendido no roteamento de mensagens que se originam e são destinadas a processos residentes em um mesmo Transputer;

- tempo dispendido no roteamento realizado no Transputer fonte da mensagem, quando a mensagem é destinada a outro Transputer;
- tempo dispendido no roteamento realizado em um Transputer que funciona como passagem de uma mensagem; e
- tempo dispendido no roteamento realizado no Transputer destino de uma mensagem, supondo que a mensagem venha de outro Transputer.

Medições de tempo realizadas para esses quatro casos utilizando o processo Roteador/Distribuidor instalado em Transputers modelo IMS T800-20, indicaram que o tempo médio de roteamento nas quatro condições acima se situa entre 9 e 10 μ s.

Além desse tempo, cada mensagem consome o equivalente a τ_{DMEXT} para transitar de um Transputer a outro e consome τ_{DMINT} para estar disponível no "buffer" do Processo Distribuidor.

3.5.5 Comunicação entre Processo Roteador e Processos de Baixa Prioridade

O Processo Roteador atua da seguinte maneira sobre os processos de baixa prioridade com os quais ele se comunica:

a) Recepção

Caso a mensagem já tenha sido transmitida e esteja disponível no Processo Distribuidor, o processo receptor continua escalado. Caso contrário, o processo receptor é desescalado e fica aguardando a chegada da mensagem transmitida.

b) Transmissão

O processo transmissor é desescalado e colocado no final da fila de espera. Caso os processos receptor e transmissor estejam alocados no mesmo Transputer e se a mensagem já tenha sido anteriormente solicitada pelo processo receptor, este é colocado no final da fila de espera após o processo transmissor.

3.6 Escalação Convencional versus Escalação em Transputers

Devido à presença do Escalador Interno e à conseqüente possibilidade de se paralelizarem tarefas em um único Transputer, verifica-se que as regras convencionalmente usadas nos casos clássicos de escalação não são aplicáveis na escalação em Transputers.

Para comprovar a assertiva feita acima, apresenta-se um exemplo em que se comparam os comprimentos de escalação gerados pelo método tradicional face aos resultados obtidos em Transputers.

Suponham-se três processos interdependentes P1, P2 e P3 interligados pelo grafo de precedência mostrado na figura 3.6 e com a codificação em Occam apresentada na figura 3.7.

Fig. 3.6 Grafo de precedência entre três tarefas interdependentes

Processo P1:	Processo P2:	Processo P3:
SEQ rel ? t11 SEQ i=1 FOR 900 sum11:=sum11+i rel ? t12 can12 ! m12 rel ? t13 SEQ i=1 FOR 900 sum12:=sum12+i rel ? t14 can13 ! m13 rel ? t15 SEQ i=1 FOR 2700 sum13:=sum13+i rel ? t16	SEQ rel ? t21 SEQ i=1 FOR 2700 sum21:=sum21+i rel ? t22 can23 ! m23 rel ? t23 SEQ i=1 FOR 900 sum22:=sum22+i rel ? t24 can12 ? m12 rel ? t25 SEQ i=1 FOR 1800 sum23:=sum23+i rel ? t26	SEQ rel ? t31 SEQ i=1 FOR 1800 sum31:=sum31+i rel ? t32 can23 ? m23 rel ? t33 SEQ i=1 FOR 900 sum32:=sum32+i rel ? t34 can13 ? m13 rel ? t35 SEQ i=1 FOR 3600 sum33:=sum33+i rel ? t36

Fig. 3.7 Codificação dos processos P1, P2 e P3

No enfoque clássico, o comprimento da escalação destas tarefas é de 241 unidades de tempo (ticks não prioritários), independentemente do número de processadores em paralelo, pois P2 só pode começar após P1 terminar e P3, após P1 e P2 terminarem, equivalendo a um sistema sem nenhuma concorrência. Esta abordagem somente prevê a troca de dados entre as tarefas ao término da sucessora e no início da predecessora. Esta situação corresponde à alocação das três tarefas em SEQ em um único Transputer.

Aplica-se este mesmo conjunto de processos a uma rede composta de dois Transputers (T1 e T2), de acordo com as escalações mostradas na figura 3.8.

1º teste:		2º teste:		3º teste:	
T1:	T2:	T1:	T2:	T1:	T2:
PAR	P3	PAR	P2	PAR	P1
P2		P3		P3	
P1		P1		P2	

Fig. 3.8 Exemplo de diferentes escalações de P1, P2 e P3

O comprimento da escalação obtido em cada teste é, respectivamente, 186, 202 e 219 unidades de tempo. Comparando esses valores com o tempo de 241 unidades de tempo, obtido através do enfoque clássico, verifica-se que, no melhor caso, obteve-se uma redução de 55 unidades de tempo, correspondendo a um decréscimo de 22,8% no comprimento da escalação.

Caso se processe este mesmo exemplo em três Transputers, com cada tarefa colocada em SEQ em um Transputer, o comprimento resultante da escalação é 121 unidades de tempo, o que redundaria em um tempo que é a metade do tempo obtido pelos métodos convencionais.

Através dos resultados anteriores, conclui-se que o modo de operação dos Transputers permite que se utilizem técnicas mais eficientes de escalação, que conduzem a reduções em seu comprimento, quando comparado com as técnicas convencionais.

4 - CONCEPÇÃO DO PROGRAMA ESCALADOR

Este capítulo visa apresentar o problema que se deseja resolver e os requisitos que o método de solução proposto, denominado Programa Escalador, deve satisfazer.

A estruturação deste capítulo e dos próximos utiliza, fundamentalmente, conceitos de Engenharia de Software retirados da referência (GENERAL ELECTRIC, 1986).

4.1 Apresentação do Problema a ser Resolvido

Neste trabalho assume-se que o particionamento dos sistemas de tarefas já tenha sido efetuado de acordo com suas fronteiras naturais (JENNY, 1977), buscando explorar o paralelismo inerente e reduzir ao mínimo a comunicação interprocessos (VRANES, 1989).

Assume-se que o código que representa as tarefas permaneça imutável, muito embora modificações nos dados de entrada a serem processados pelo código sejam permitidas.

4.1.1 Especificação do Problema a ser Resolvido

Especifica-se o problema, definindo-se o modelo a ser empregado e a função-objetivo a ser alcançada, conforme exposto na seção 2.2.

a) Modelo a ser empregado

- Recursos:

- o número de processadores disponíveis é constante e igual a m ;
- todos os processadores têm a mesma capacidade (são idênticos);
- não se pode atribuir mais de uma tarefa a um processador em qualquer instante de tempo;
- um processador não gasta tempo se preparando para iniciar uma nova tarefa;
- as vias de comunicação entre os processadores diretamente interligados são idênticas;
- a comunicação entre processadores não diretamente interligados se processa através de nós intermediários constituídos pelos próprios processadores; e
- o tempo de comunicação interprocessadores depende da distância separando ambos os processadores, do tipo do dado a ser transferido (real, inteiro, booleano, etc) e da quantidade de dados a serem transmitidos.

- Sistema de tarefas:

- o conjunto de n tarefas é fixo;
- a relação de precedência entre as tarefas é dada por um grafo dirigido acíclico arbitrário, significando que elas possuem um ordenamento parcial entre si;
- o tempo de execução das tarefas é arbitrário, podendo ser fornecido seus valores médio ou máximo;
- o tempo de execução das tarefas e o volume de comunicação intermódulos (CIM) devem corresponder a variáveis determinísticas;

- todas as tarefas estão sempre disponíveis para serem executadas;
 - as tarefas são indivisíveis, de modo que no máximo um processador pode estar executando uma dada tarefa em qualquer instante; e
 - não há nenhum intervalo durante o processamento das tarefas em que nenhum processador esteja executando nenhuma tarefa, isto é, não se permite ter todos os processadores ociosos quando tarefas incompletas existem.
- Restrições impostas:
- a preempção de tarefas não é permitida (escalação básica); e
 - cada tarefa deve ser sempre executada no mesmo processador ao qual foi designada (escalação do tipo fixa).

b) Função-objetivo

A função-objetivo é executar as n tarefas nos m processadores no menor tempo possível.

4.1.2 Classificação do Problema a ser Resolvido

Conforme se verifica no sub-item 2.4.1.3, o problema proposto neste trabalho de determinar uma escalação não-preemptiva de mínimo comprimento de n tarefas parcialmente ordenadas, com tempos de execução arbitrários, em m processadores idênticos, é NP-completo (COFFMAN, 1976).

4.2 Especificação de Requisitos do Programa Escalador

Esta seção busca classificar a solução proposta, estipular as interfaces do Programa Escalador, enunciar os requisitos que o Programa Escalador deverá cumprir, estabelecer as funções que ele deverá desempenhar, definir sua estrutura e apresentar os critérios e testes de aceitação a serem aplicados.

4.2.1 Classificação do Programa Escalador

Em um sistema de tempo real, um conjunto fixo de processos aplicativos residem permanentemente no sistema (o sistema é dedicado exclusivamente a um dado objetivo) e todos os atributos das tarefas são conhecidos "a priori". É mais eficiente realizar análises extensivas e demoradas antes do processamento em tempo real, porque se a escalção é feita em tempo real, pode-se reduzir o ganho de se usar um sistema multiprocessado em detrimento de um monoprocesso. Portanto, escalção dinâmica, embora genérica, não é aceitável em sistemas de tempo real (VRANES, 1989).

Conforme já foi visto na alínea "b" do sub-item 2.2.1.2, a escalção dinâmica não é adequada para multiprocessadores fracamente acoplados. Além disso, devido à própria limitação do Occam, a configuração dos processos deve ocorrer em tempo de compilação (MURRAY; WELLINGS, 1988). Esses fatores conduzem à escolha da escalção estática para resolver o problema proposto neste trabalho.

Baseado no que foi exposto no parágrafo anterior e no item 4.1.1, pode-se enquadrar o Programa Escalador como empregando um

algoritmo de escalação do tipo estático, determinístico, fixo e básico (não-preemptivo).

4.2.2 Requisitos do Programa Escalador

Os requisitos considerados importantes para o Programa Escalador são comentados a seguir:

- a) a diferença nos comprimentos de escalação simulada (W_S) e real (W_R) deve apresentar um desvio inferior a 5%, ou seja, deve obedecer a seguinte relação:

$$\frac{|W_S - W_R|}{W_R} \leq 0,05 \quad (4.1)$$

- b) o Programa Escalador deve estar apto a manipular até 1000 processos e 100 Transputers, limitação essa imposta pelo protocolo usado no Processo Roteador, em que se prevê três dígitos para identificar um processo e dois dígitos para identificar um Transputer (vide item 3.5.2);
- c) o Programa Escalador deve ser portátil, devendo ser processável, no mínimo, em qualquer tipo de microcomputador PC-compatível;
- d) o tempo de processamento do Programa Escalador, para qualquer número de tarefas (n) e para qualquer número de Transputers (m), deve ser tal que o usuário não descarte seu uso por excesso de tempo de processamento; e
- e) o Programa Escalador, deve ser aplicável a qualquer modelo de Transputer (T212, T414 ou T800), a qualquer frequência de operação e a qualquer taxa de transmissão de dados entre os

mesmos. No entanto, uma vez selecionado qualquer um dos parâmetros listados acima, todos os elementos da rede devem segui-lo, significando que a rede de Transputers deve ser homogênea. Isto porque se assume (vide item 4.1.1, alínea "a") que todos os processadores possuem a mesma capacidade de processamento e que as vias de comunicação são idênticas.

4.2.3 Interfaces do Programa Escalador

Este item descreve o ambiente de hardware e software básico em que o Programa Escalador está previsto para operar e a interface com os usuários que o empregarão.

a) Plataforma de hardware e software básico

Optou-se por utilizar a linguagem de programação C++, a qual é estruturada, de ampla penetração nos meios acadêmicos e científicos e disponível em microcomputadores PC-compatíveis e em computadores de maior porte, notadamente em máquinas que operam no ambiente UNIX. Dois dos motivos básicos para a escolha da linguagem C++ são os seguintes:

- atender o requisito de portabilidade apresentado na alínea "c" do item 4.2.2; e
- facilidade com que se pode tratar, nessa linguagem, de tipos básicos que requerem alocação dinâmica de memória, como listas, vetores, matrizes, etc.

Em particular, caso se utilizem microcomputadores PC-compatíveis, requer-se a seguinte configuração mínima:

- microcomputador IBM PC ou compatível;

- sistema operacional PC-DOS ou MS-DOS versão 2.0 ou mais moderna; e
- 640 kbytes de memória RAM interna.

b) Interface com o usuário

Todo o interfaceamento com o usuário é feito através de arquivos-texto de entrada/saída, de forma similar ao usado no UNIX.

4.2.4 Seleção do Método para Resolver o Problema de Escalação Proposto

Para resolver o problema de escalação proposto neste trabalho, optou-se por métodos heurísticos, devido às seguintes razões:

- para atender o requisito de tempo de processamento mencionado na alínea "d" do item 4.2.2, de acordo com os argumentos apresentados no sub-item 2.4.1.3;
- pela generalidade que o Programa Escalador deve apresentar, devendo ser aplicável a problemas com qualquer comprimento de entrada, o que tornaria uma solução ótima impraticável, devido à NP-completeza do problema; e
- o emprego de métodos enumerativos implica na necessidade de se conhecer, de antemão, o encadeamento entre as tarefas em um processador, sabendo-se que uma tarefa, uma vez iniciada, será processada até o fim. Essa situação não ocorre ao se usar um Transputer, pois a seqüência de utilização da UCP do Transputer por processos alocados na construção PAR depende das escalações

e desescalagens realizadas pelo Escalador Interno, como função das comunicações/temporizações realizadas por cada processo e do tempo de utilização da UCP de cada um, não se sabendo então, "a priori", quando um processo dentro da construção PAR é iniciado ou quando ele termina. Dessa forma, a determinação da sequência de uso da UCP do Transputer só pode ser feita mediante a simulação dinâmica do processamento do conjunto de processos escalados na rede de Transputers, o que inviabiliza o uso de métodos de escalagem que produzam resultados ótimos.

4.2.5 Descrição do Programa Escalador

O Programa Escalador determina a distribuição das tarefas nos Transputers constituintes da rede, estabelece a sequência de processamento das mesmas em cada Transputer, especifica o regime de operação (SEQ ou PAR) de cada uma delas e informa ao usuário o comprimento da escalagem resultante.

Para chegar a uma solução, o Programa Escalador coleta os dados de entrada necessários, gera diversas escalagens, avalia seu comprimento e seleciona o melhor, comunicando então o resultado ao usuário.

As funções que habilitam o Programa Escalador a executar as operações citadas no parágrafo anterior são esquematizadas na figura 4.1. Essas funções são três e foram designadas por **partes** do Programa Escalador.

Fig. 4.1 Diagrama esquemático das partes constituintes do Programa Escalador

A descrição de cada uma das partes constituintes da arquitetura do Programa Escalador é feita a seguir.

a) Gerenciador do Programa Escalador

Esta parte é incumbida de prover a interface com o usuário, selecionar a melhor escalação e gerenciar as atividades das outras duas partes do Programa Escalador.

b) Escalador Externo (ESCEXT)

Trata-se da parte do Programa Escalador responsável por escalar os processos nos Transputers e determinar seus regimes de operação (SEQ ou PAR).

O ESCEXT deve gerar escalações empregando uma biblioteca de algoritmos heurísticos, objetivando testar diferentes arranjos das tarefas nos Transputers.

c) Simulador da Rede (SIMREDE)

Uma vez definida a distribuição dos processos nos Transputers e o regime de operação de cada processo, há necessidade de se verificar qual o comprimento da escalação resultante desse arranjo. Para tanto, as seguintes alternativas seriam possíveis:

- realizar o processamento das tarefas em uma rede real de Transputers; ou
- simular o processamento das tarefas em uma rede modelada de Transputers.

Optou-se pela segunda alternativa, por ser mais rápida e eficiente, reduzindo o trabalho do usuário do Programa Escalador.

O SIMREDE constitui um sistema capaz de simular dinamicamente o processamento de tarefas em redes de Transputers, através do modelamento das mesmas. Deve-se ressaltar que o modelamento efetuado busca apenas reproduzir a seqüência de chamada das tarefas, as suas escalações e desescações e a monitoração de seus términos de processamento. Dessa forma, o modelamento encara cada tarefa como sendo equivalente a uma "caixa preta", cujo tempo de duração e cujos motivos e instantes de desescação são fornecidos pelo usuário do Programa Escalador. O objetivo da simulação não é reproduzir os cálculos efetuados pelo programa a ser instalado na rede de Transputers, mas sim obter uma estimativa precisa de qual seria o tempo de processamento desse programa em uma determinada rede de Transputers.

O SIMREDE é dividido nas duas seguintes partes:

- Gerenciador da Rede (GEREDE):

É encarregado de prover comunicação e sincronismo entre os Transputers simulados da rede, de cuidar dos quanta de tempo destinados a cada processo e de manipular as temporizações dos processos, sendo responsável também por avançar a simulação no tempo; e

- Modelador SEQ/PAR (MSEQ/PAR):

Executa basicamente o papel de Escalador Interno dos Transputers e simula a operação das construções SEQ e PAR do Transputer real.

Em termos gerais, o GEREDE se preocupa com todos os Transputers da rede como um todo, enquanto que o MSEQ/PAR se encarrega de cada Transputer individualmente.

4.2.6 Descrição das Funções Executadas pelo Programa Escalador

Neste item descrevem-se as funções que cada uma das partes constituintes do Programa Escalador irá executar.

4.2.6.1 Funções do Gerenciador do Programa Escalador

O Gerenciador do Programa Escalador é responsável por coordenar as atividades dentro do Programa Escalador, chamando inicialmente a função que converte os arquivos de entrada gerados pelo usuário em estruturas utilizáveis pelo Programa Escalador. Em seguida, ele chama a função que inicia os parâmetros para utilização nos algoritmos heurísticos. O Gerenciador então

promove a chamada ordenada desses algoritmos heurísticos no ESCEXT, calcula seus comprimentos através do SIMREDE e seleciona a melhor escalação sub-ótima. Ao terminar o processamento das escalações, o Gerenciador deve transferir ao usuário os dados de saída, dando por encerrado o processamento do Programa Escalador.

Para realizar o que foi exposto, o Gerenciador do Programa Escalador executa as seguintes funções:

- formatação dos dados de entrada;
- iniciação dos parâmetros para geração de escalações (realizada pelo ESCEXT);
- geração de escalações heurísticas (realizada pelo ESCEXT);
- cálculo do comprimento da escalação gerada (realizado pelo SIMREDE);
- seleção da melhor escalação obtida; e
- formatação dos dados de saída.

Deve-se observar que as funções de iniciação e geração de escalações e de cálculo do comprimento da escalação gerada, devido à sua complexidade e independência, foram colocadas dentro das outras partes do Programa Escalador.

Apresenta-se, a seguir, uma descrição das operações realizadas pelas funções do Gerenciador do Programa Escalador.

a) Formatação dos dados de entrada

Esta função busca, nos arquivos-texto gerados pelo usuário, os dados de entrada necessários para caracterizar o comportamento do conjunto de tarefas a ser escalado e para definir a

topologia da rede de Transputers. O usuário deve gerar um arquivo com os dados das tarefas e outro com os dados da rede de Transputers. Esta função realiza uma série de verificações nos dados de entrada, visando eliminar erros na inserção dos mesmos no Programa Escalador.

Os dados de entrada são transferidos para as estruturas mostradas a seguir.

- Dados de entrada relativos às tarefas, contendo as seguintes informações:

- n ;
- MPREC;
- $V_{\tau\text{SEQ}}$ e $V_{\tau\text{PAR}}$; e
- VLEV, conforme descrito na figura 4.2.

OBS.: O significado das siglas utilizadas neste trabalho encontra-se na Lista de Abreviaturas.

Fig. 4.2 Estrutura do vetor de listas de eventos VLEV

Cada elemento de VLEV corresponde a uma lista ligada com registros do tipo EVENTO, cuja estrutura é mostrada na figura 4.3.

τ_{EV} (SEQ)	τ_{EV} (PAR)	Comunic. (Recepção)	P_T		NSM	
		Comun. (Transmissão)	P_R	NSM	τ_{DMINT}	τ_{DMEXT}
		Temporização	τ_{ATR}			
		Fim processamento	---			

Fig. 4.3 Estrutura do registro EVENTO

Esta estrutura apresenta nos dois primeiros campos a duração dos eventos τ_{EV} em SEQ e PAR. O terceiro campo é uma chave, a qual pode assumir quatro posições, correspondentes a cada uma das possibilidades do evento desescalador. A cada uma das quatro posições corresponde os dados que lhe são necessários.

- Dados de entrada relativos à rede de Transputers, contendo as seguintes informações:
 - m; e
 - MNNI.

b) Seleção da melhor escalação sub-ótima

Esta função compara o valor da última escalação gerada pelo ESCEXT e testada pelo SIMREDE com o valor da melhor escalação até então obtida, armazenando a melhor dentre as duas.

c) Formatação dos dados de saída

Esta função deve formatar os dados da escalação gerados pelo ESCEXT de um modo compreensível ao usuário e transferí-los a um arquivo-texto de saída. Os dados de saída a serem apresentados são os seguintes:

- alocação das tarefas em cada Transputer da rede;
- definição de quais tarefas serão processadas em SEQ ou PAR;
- seqüência de processamento das tarefas em cada Transputer;
- comprimento da melhor escalação obtida (WESCOBT); e
- comprimento da rota crítica WCR (vide sub-item 2.2.2.2),
calculado com os tempos de processamento das tarefas em SEQ
(τ_{SEQ}).

A forma com que essas informações são colocadas no arquivo de saída é exemplificada na figura 4.4.

```

Transputer T1:
Pa
Pb Pc Pd
Pe
Pf

Transputer T2:
Pg Ph Pi Pj
Pk
Pl
Pm

.
.
.

Transputer Tm:
Pr
Ps
Pt

WESCOBT = XXXXX microsegundos

WCR = YYYYY microsegundos

```

Fig. 4.4 Forma de apresentação dos dados de saída ao usuário

A escalação fornecida ao usuário apresenta a identificação do Transputer (T_i) e, após, a seqüência em que as tarefas escaladas em T_i devem ser chamadas, sendo que se elas estiverem em SEQ aparece apenas uma tarefa na linha e se

estiverem em PAR aparecem duas ou mais tarefas. Em seguida são apresentados WESCOBT e WCR, para o usuário poder compará-los.

4.2.6.2 Funções do ESCEXT

A principal atribuição do ESCEXT é executar a escalação das tarefas, empregando algoritmos heurísticos, os quais levam em conta o balanceamento da carga computacional nos processadores. O volume de comunicação intermódulos (CIM, vide item 2.3.1), não é considerado nos algoritmos do ESCEXT, mas na simulação da rede de Transputers, conforme é comentado no sub-item 4.2.6.3.

Os algoritmos heurísticos empregados correspondem a escalações em lista (vide item 2.4.2), muito embora com uma pequena alteração: enquanto na escalação em lista se admite a ociosidade dos processadores, nos algoritmos a serem aqui desenvolvidos, devido à característica de processamento concorrente dos Transputers, não se prevê o uso de processos fictícios para preencher intervalos ociosos dos processadores. Durante a escalação, todo o tempo disponível dos Transputers será preenchido com processos e as eventuais ociosidades ocorrerão apenas na prática, quando o modelo da rede for simulado.

Mesmo a escalação de um processo filho após seu pai em SEQ em um mesmo Transputer será viável, devido à atuação do Processo Escalador/Distribuidor. O único cuidado que se tomará será em não escalar em SEQ um processo filho antes de seu pai no mesmo Transputer, para evitar o travamento do processamento.

A idéia aqui é criar o vetor de prioridade das tarefas (VPRIOR), ordenado segundo critérios que incluem o comprimento

absoluto (CP) e normalizado (CPn) do caminho crítico, a camada ou nível (N), o tempo de processamento das tarefas (τ_{SEQ}/τ_{PAR}) e o número de sucessoras imediatas de cada tarefa (NSI).

Tendo-se gerado o vetor VPRIOR, procede-se à escalação das tarefas, resultando o vetor de listas VLLESC com o formato mostrado na figura 4.5, composto de elementos denominados REGIME, com a composição mostrada na figura 4.6.

Fig. 4.5 Estrutura do vetor de listas VLLESC

Fig. 4.6 Estrutura do elemento REGIME

A estrutura de REGIME é composta de um número inteiro que indica a quantidade de processos nessa construção: se for SEQ, o número é 1 e se for PAR, o número é maior que 1. A lista de processos contém a identificação dos processos em SEQ ou PAR.

Para implementar os algoritmos heurísticos de escalação, o ESCEXT emprega as seguintes funções:

a) Iniciação dos parâmetros para geração de escalões

O comprimento absoluto (CP) e o comprimento normalizado do caminho crítico (CPn) e o nível (N) de cada tarefa são definidos através dos seguintes algoritmos:

- Cálculo do comprimento da rota crítica de cada tarefa

Com base na matriz de precedência, esta função deve calcular o comprimento da rota crítica de cada tarefa (CP), usando os tempos de processamento medidos em SEQ, sendo que o comprimento da rota crítica da tarefa fictícia P_0 , raiz do grafo de precedência, corresponde a WCR.

Toma-se, como exemplo, o grafo de precedência apresentado na figura 4.7, em que a duração de cada tarefa corresponde a sua própria identificação, exceto P_9 , que é uma tarefa fictícia de duração nula. Calcula-se o comprimento da rota crítica de cada tarefa, o qual é colocado ao lado de cada nó da figura 4.7, cujo valor de WCR é 22.

Fig. 4.7 Grafo de precedência mostrando o comprimento da rota crítica de cada tarefa

- Cálculo do comprimento normalizado da rota crítica de cada tarefa

Esta função calcula o tempo médio de duração das tarefas na construção SEQ ($\tau_{SEQ,MÉDIO}$) e, em função desse valor, extrai um fator de correção FC, conforme mostrado na tabela 4.1, o qual multiplica o valor do comprimento da rota crítica de cada tarefa, obtendo um comprimento normalizado.

Faixa de $\tau_{SEQ,MÉDIO}$ (em μs)	FC
0.0 a 9.99	1
10.0 a 99.99	0.1
100.0 a 999.99	0.01
1000.0 a 9999.99	0.001
10000.0 a 99999.99	0.0001
100000.0 a 999999.99	0.00001

Tab. 4.1 Tabela do fator de correção FC para cálculo dos comprimentos normalizados das rotas críticas das tarefas

Para ilustrar a aplicação do fator de correção FC, toma-se o grafo de precedência mostrado na figura 4.7 e calcula-se o comprimento normalizado da rota crítica de cada tarefa. Neste caso, $\sum \tau_{SEQ}(P_i) = 36$, $n = 8$, $\tau_{SEQ,MÉDIO} = 4.5$ e $FC = 1$, significando que o comprimento normalizado e não normalizado da rota crítica possuem o mesmo valor.

- Cálculo da camada ou nível de cada tarefa

Uma tarefa é dita madura se ela está pronta para ser alocada em um processador, isto é, quando ela não tem predecessoras

imediatas (pais) ou quando todos seus pais já foram alocados. Por sua vez, cada conjunto de tarefas que fica maduro quando as tarefas pertencentes ao nível anterior tenham sido todas alocadas pertencem a um mesmo nível. Assume-se que a tarefa fictícia P_0 tenha nível 0.

Deve-se enfatizar que a definição de nível sendo empregada aqui difere daquela apresentada na alínea "a" do sub-item 2.2.2.2, onde nível e comprimento do caminho crítico são sinônimos.

Esta função atribui a cada tarefa um nível. O nível das tarefas do grafo de precedência dado como exemplo na figura 4.7, é mostrado na figura 4.8.

Fig. 4.8 Grafo de precedência mostrando o nível das tarefas

b) Geração de escalação pelo algoritmo CP/NSI

Esta função ordena as n tarefas em ordem decrescente de seu comprimento da rota crítica CP e as coloca no vetor de

prioridade VPRIOR. Em caso de empate, a preferência é dada à tarefa com maior número de sucessoras imediatas (NSI). Caso ainda assim perdure o empate, a colocação em VPRIOR é aleatória.

Em seguida, executa-se a escalção das tarefas no regime SEQ, buscando alocar a próxima tarefa de VPRIOR no Transputer com menor carga computacional acumulada. Caso haja dois ou mais Transputers na mesma situação, dar preferência ao Transputer com menor número de identificação.

Este algoritmo foi criado baseado no algoritmo CP/MISF (vide sub-item 2.4.2.5).

c) Geração de escalção pelo algoritmo CP/NSI/PAR

Esta função provê uma opção para a escalção gerada pelo algoritmo CP/NSI, através do seguinte artifício: as tarefas consecutivas dentro do mesmo nível N e alocadas no mesmo Transputer são colocadas em PAR. Tendo feito isso, verifica-se se restaram regimes PAR consecutivos no mesmo Transputer. Em caso afirmativo, as tarefas que se enquadrem nesses casos são todas agrupadas em um mesmo PAR.

d) Geração de escalção pelo algoritmo $N/\tau_{SEQ}/CP$

Esta função ordena as n tarefas em ordem crescente de seu nível N, colocando no vetor de prioridade VPRIOR primeiro as tarefas de maior duração τ_{SEQ} dentro do mesmo nível. Caso surjam empates, dá-se preferência às tarefas com maior comprimento da rota crítica CP; caso o empate persista, a colocação no vetor é aleatória.

Em seguida, executa-se a escalaco das tarefas da mesma forma que foi feita para a funo "Gerao de escalaco pelo algoritmo CP/NSI".

Este algoritmo se baseia no algoritmo HNF (sub-item 2.4.2.7).

e) Gerao de escalaco pelo algoritmo $N/\tau_{SEQ}/CP/PAR$

Esta funo tem um escopo similar ao da funo "Gerao de escalaco pelo algoritmo CP/NSI/PAR", com a diferena que ela se aplica como opo para o algoritmo $N/\tau_{SEQ}/CP$.

f) Gerao de escalaco pelo algoritmo N/CP

Esta funo gera o vetor VPRIOR, colocando primeiro no vetor as tarefas de menor nvel N e, dentro de um mesmo nvel, as tarefas com maior comprimento de caminho crtico (CP). Em caso de empate a colocao em VPRIOR  aleatria. Aps isso, ela gera a escalaco.

Este algoritmo foi concebido com base no algoritmo ETF/CP (sub-item 2.4.2.7).

g) Gerao de escalaco pelo algoritmo $N/CP/PAR$

Esta funo tem um escopo similar ao da funo "Gerao de escalaco pelo algoritmo CP/NSI/PAR", com a diferena que ela se aplica como opo para o algoritmo N/CP .

h) Gerao de escalaco pelo algoritmo $N/CPn/NSI$

Esta funo gera o vetor VPRIOR, colocando primeiro no vetor as tarefas de menor nvel e, dentro do mesmo nvel, a tarefa

de maior comprimento de caminho crítico normalizado (CPn) e, em caso de empate, a tarefa com maior número de sucessoras imediatas (NSI).

Utiliza-se a normalização porque pode ser eventualmente vantajoso para o bom desempenho deste algoritmo, que valores próximos de CP sejam assumidos como iguais. Assim, por exemplo, ao se aplicar a normalização, pode-se chegar ao caso dos valores CP 10110 e 10145 serem ambos aproximados por 101.

Este algoritmo corresponde à fusão dos algoritmos N/CP e CP/NSI, acrescido da normalização de CP.

i) Geração de escalação pelo algoritmo N/CPn/NSI/PAR

Esta função tem um escopo similar ao da função "Geração de escalação pelo algoritmo CP/NSI/PAR", com a diferença que ela se aplica como opção para o algoritmo N/CPn/NSI.

j) Geração de escalação pelo algoritmo LPT

Esta função gera o vetor VPRIOR, ordenado em ordem decrescente dos tempos de execução das tarefas em PAR (τ_{PAR}). Caso haja duas ou mais tarefas com o mesmo tempo de processamento, sua colocação em VPRIOR é aleatória.

Em seguida, realiza-se a escalação das tarefas no regime PAR, seguindo a mesma regra de distribuição da carga computacional nos Transputers empregada na função "Geração de escalação pelo algoritmo CP/NSI".

Este algoritmo foi criado com base no algoritmo LPT (vide subitem 2.4.2.3).

k) Geração de escalação pelo algoritmo MTP

O método da média dos tempos de processamento (MTP) consiste, basicamente, em gerar o vetor VPRIOR, seguindo o mesmo critério de ordenação do algoritmo LPT.

Em seguida, executa-se a escalação das tarefas no regime PAR, buscando carregar os Transputers em ordem crescente de seu número de identificação, distribuindo as tarefas pelos processadores de forma que o tempo de processamento em cada Transputer se aproxime da média do tempo global de processamento ($\tau_{PAR, MÉDIO}$), calculada pela divisão da somatória dos tempos de processamento em PAR (τ_{PAR}) de todas as tarefas pelo número de Transputers da rede.

Visto que este algoritmo apresenta um algoritmo de escalação mais complexo que os demais, apresenta-se um exemplo para ilustrar seu funcionamento.

Suponha que oito tarefas P_1 a P_8 , com tempos de processamento τ_{PAR} coincidentes com sua identificação (1, 2, ... 8), devam ser escaladas em dois processadores T_1 e T_2 . A média dos tempos de processamento para escalação em dois processadores é $\tau_{PAR, MÉDIO}=18$.

A ordenação de VPRIOR é feita de acordo com os valores decrescentes de τ_{PAR} , resultando o seguinte vetor:

$VPRIOR=(P_8, P_7, P_6, P_5, P_4, P_3, P_2, P_1)$

A escalação resultante é mostrada na figura 4.9, onde se constata que o tempo total de processamento em cada processador coincide com $\tau_{PAR, MÉDIO}$, inexistindo, portanto, o

resíduo, que corresponde às tarefas escaladas em um processador além de $\tau_{PAR,MÉDIO}$.

Fig. 4.9 Exemplo de escalagem pelo método MTP sem resíduo

Criando-se um novo exemplo para o método MTP com resíduo, em que a duração de cada tarefa corresponda à sua própria identificação, resulta em um vetor VPRIOR dado por:

$$VPRIOR=(P_{15},P_{15},P_{15},P_{15},P_{15},P_7,P_4,P_1)$$

Para escalar em três processadores T_1 , T_2 e T_3 , a média resulta em 29. Executando este algoritmo até o ponto em que não se ultrapasse a média, resulta a escalagem mostrada na figura 4.10.

Fig. 4.10 Exemplo de escalagem parcial pelo método MTP com resíduo

Nota-se, pela figura 4.10, que sobram duas tarefas com duração de 15 unidades de tempo, as quais excedem a média de 29 quando somadas ao tempo acumulado de processamento em T_2 ou T_3 . Procura-se, nesse caso, distribuir as tarefas que excedem a média, de forma a balancear a carga computacional nos Transputers, resultando na escalação da figura 4.11.

Fig. 4.11 Exemplo de escalação pelo método MTP com resíduo

4.2.6.3 Funções do GEREDE

O principal motivo de se modelar a operação de uma rede de Transputers é obter, de forma rápida e precisa, o tempo consumido no processamento de um conjunto de tarefas (WESC). Dentro de cada Transputer simulado, cabe ao GEREDE a atribuição de totalizar esse tempo.

Visto que a transferência de mensagens entre processos ocorre de forma sincronizada, é necessário prover uma base comum de tempo para todos os Transputers simulados, para poder comparar o instante de tempo da transmissão e da recepção de mensagens entre processos residentes em Transputers distintos. Essa base comum de

tempo foi designada TRP, o qual é continuamente atualizado e corresponde a um relógio, que marca o instante de tempo em que se encontra a simulação de toda a rede de Transputers.

Com relação aos efeitos da comunicação intermódulos, atrasos causados pela comunicação são considerados apenas nos Transputers transmissor e receptor da mensagem, mas não nos nós intermediários da rede, onde se consome tempo para rotear as mensagens (τ_{ROT}). A razão dessa simplificação foi motivada pela dificuldade que seria gerada no SIMREDE, caso se decidisse levar em conta os atrasos gerados pela passagem das mensagens em trânsito por cada um dos nós intermediários da rede. Ademais, justifica-se essa decisão devido ao fato de τ_{ROT} , que é da ordem de 9 a 10 μs , conforme se mostrou no item 3.5.4, ser muito pequeno perante os tempos de computação das tarefas a serem considerados neste trabalho.

A descrição de cada uma das funções desempenhadas pelo GEREDE é feita a seguir.

a) Coordenação da simulação dinâmica da rede de Transputers

Visto que o Programa Escalador foi construído para ser processado em computadores seqüenciais, há necessidade de se converter ações simultâneas, desempenhadas pela rede de Transputers, em ações seqüenciais. Esta função é encarregada dessa seqüencialização, operando como coordenadora geral da simulação, através do estabelecimento da seqüência de chamada das funções do SIMREDE.

A evolução do tempo na simulação dinâmica da rede de Transputers leva em conta todos os eventos desescaladores e ativadores de processos gerados por cada tarefa, a saber:

- retorno à fila de espera de processos desescalados por temporização (evento ativador de processo monitorado pelo IRFE);
- instante de chegada a seu destino de mensagens transmitidas (evento ativador de processo monitorado pelo ICMT);
- término dos quanta de tempo (evento desescalador monitorado pelo ITQT); e
- comunicações, temporizações e término de processamento das tarefas (eventos desescaladores monitorados pelo IPPEV).

A idéia é fazer com que o tempo consumido nos Transputers simulados da rede avance de forma homogênea a partir de um instante inicial nulo, comum a todos. Cada avanço de tempo vai do instante atual até o próximo evento previsto de ocorrer em qualquer um dos Transputers, cujo instante de ocorrência corresponde ao menor valor dentre os quatro parâmetros listados acima. Ao se chamar o Transputer responsável pelo próximo evento, tem-se a certeza que nenhum outro evento anterior no tempo é deixado para trás na simulação.

O primeiro passo executado por esta função é chamar a função "Inicialização da simulação dinâmica da rede de Transputers". Em seguida, ela passa a controlar a simulação da rede de Transputers, verificando qual o menor valor entre IPPEV, IRFE, ICMT e ITQT, igualando o valor escolhido a TRP e chamando a função que manipula o evento selecionado.

Sempre que se retorna a esta função, verifica-se o número de Transputers ativos. Quando esse valor chega a 0, significando que todos os Transputers encerraram seu processamento, ela

terá calculado o valor do comprimento da escalação WESC e encerrado seu processamento.

b) Iniciação da simulação dinâmica da rede de Transputers

Esta função tem por objetivo preparar os dados para o início da simulação.

Ela inicia algumas variáveis e, depois disso, monta o vetor VALOCA. Para tanto, percorre-se cada uma das listas VLLESC(T_i), correspondentes aos m Transputers da rede e, para cada processo P_j , coloca-se em VALOCA(P_j) a identificação do Transputer T_i que o aloja. Por fim, ela chama os m Transputers simulados da rede, representados pela função controladora do MSEQ/PAR (Controle da ordem de chamada e do regime de operação dos processos), para que cada um deles possa avançar até o primeiro evento desescalador do primeiro processo escalado.

Após ter feito isso, ela retorna à função "Coordenação da simulação dinâmica da rede de Transputers".

c) Controle de IRFE

Esta função deve eliminar da lista LRIRFE o processo que encerrou sua temporização, chamar a função "Manipulação da reativação de processos" e passar-lhe a identificação do processo, para que ele possa ser reativado.

d) Controle de ITQT

Esta função reconhece o instante de ocorrência do término do quantum atual de tempo (ITQT), que se assume ocorrer simultaneamente para todos os Transputers da rede a cada

1024 μ s. A cada vez que isso ocorre, para cada um dos Transputers ativos da rede em regime de operação PAR, com pelo menos um processo ativo, ela chama a função "Manipulação de término de quantum de tempo". Ela então incrementa ITQT de 1024, determinando o instante de ocorrência do próximo término de quantum de tempo.

e) Controle de ICMT

Visto que a comunicação entre processos concorrentes é sincronizada, há necessidade de se criar um artifício para simular a sincronização entre eles. Para tanto, utilizam-se dois registros sinalizadores: o processo receptor da mensagem gera o RIRM, enquanto que o processo transmissor gera o RICMT, conforme mostrado na figura 4.12. Quando ambos registros sinalizadores ficam disponíveis, ocorre a sincronização entre os processos para recepção da mensagem.

Fig. 4.12 Composição dos registros RIRM e RICMT

Ao serem transmitidas, as mensagens dispendem um certo tempo para chegar a seu destino, correspondente aos atrasos que elas sofrem em sua rota. Durante esse tempo, isto é, enquanto elas não chegam a seu destino, são chamadas mensagens em trânsito. Os atrasos são somados a TRP para gerar o instante de chegada da mensagem em trânsito ao Transputer receptor (ICMT).

Como na simulação da rede de Transputers se lida apenas com eventos presentes, há necessidade de se criar uma lista que armazene as mensagens em trânsito (lista LRICMT), para que quando TRP alcance o instante de tempo em que elas chegam, possa se verificar a sincronização. A lista LRICMT é composta dos RICMT de mensagens em trânsito e é mantida em ordem crescente do valor dos ICMT.

Quando TRP alcança o valor de algum ICMT, esta função é chamada e se encarrega de retirar da lista dos LRICMT seu registro RICMT e verificar se seu respectivo RIRM está ou não disponível junto ao RPACM do processo receptor (P_R), indicando que houve sincronização. Cada processo possui um registro RPACM que pode estar vazio ou não, dependendo do fato desse processo estar ou não aguardando a chegada de uma mensagem. Isto acontece porque cada processo aguarda, no máximo, a chegada de uma mensagem por vez.

Quando não ocorre a sincronização, esta função deve gerar o registro RMTAS, conforme mostrado na figura 4.13, a partir do registro RICMT e inseri-lo no final da lista LRMTAS do processo receptor da mensagem, a qual contém os registros RMTAS de mensagens transmitidas que estão aguardando sincronização. Somente são convertidos para RMTAS os registros RICMT que contêm $ICMT \leq TRP$, significando que nas listas LRMTAS não há mensagens em trânsito. Cada processo possui sua própria lista LRMTAS, as quais são iniciadas vazias.

Fig. 4.13 Composição do registro RMTAS

Quando ocorre a sincronização, esta função deve comunicar à função "Manipulação da reativação de processos" a identificação do processo recém-sincronizado para sua reativação. O processo transmissor não precisa ser avisado, porque ele não fica aguardando sincronização, pois assim que ele transmite a mensagem, o Processo Roteador imediatamente a recebe e despacha, e o processo transmissor segue seu processamento (quando em SEQ) ou é colocado no final da fila de espera (quando em PAR).

4.2.6.4 Funções do MSEQ/PAR

Define-se aqui o MSEQ/PAR, que modela o processamento de tarefas dentro das construções SEQ e PAR e cujas funções desempenhadas são descritas a seguir:

- a) Controle da ordem de chamada e do regime de operação dos processos

A seqüência de execução dos processos e seu regime de operação para cada Transputer da rede devem ser controlados por esta função, a qual deve escalar os processos seguindo a ordem estabelecida pelo ESCEXT.

Ao iniciar seu processamento, o GEREREDE chama esta função m vezes, para iniciar o processamento dos m Transputers simulados da rede. Ao concluir o processamento de algum regime em algum Transputer simulado, esta função também é chamada. A cada chamada, ela recebe a identificação do Transputer simulado (NTS) e um ponteiro (PLREGESC), o qual sempre aponta para o próximo regime a ser chamado no Transputer NTS na lista

LLESC. Ela então inicia o processamento das tarefas do regime apontado por PLREGESC, verificando o tipo de regime, identificando o processo escalado (NPE), calculando o instante de ocorrência (IPPEV) e o tipo (TPEV) do primeiro evento de NPE e avançando o ponteiro PLREGESC para o próximo regime de LLESC.

b) Manipulação da recepção de mensagens

Esta função é chamada pelo GEREDDE quando há alguma recepção de mensagem prevista no processo escalado. Ela avança o ponteiro PEVA para o próximo evento do processo NPE e cuida de verificar, através do registro RIRM, se há ou não sincronização.

Se houver sincronização, ela calcula o valor de IPPEV e verifica o TPEV do próximo evento do processo receptor P_R . Se não houver sincronização, a presente função deve colocar o registro RIRM no RPACM de P_R e verificar em que regime está o processo P_R .

Se estiver em SEQ, esta função iguala IPPEV ao maior valor representável no computador e faz $NPE = -1$, significando que o Transputer está ocioso, aguardando reativação de processo. Se o processo estiver em PAR, desescala o processo receptor da mensagem, podendo então ocorrer duas situações: se a fila de espera estiver vazia, significando que o Transputer está ocioso, por ter seus processos aguardando reativação (sincronização ou temporização), esta função iguala IPPEV ao maior valor representável no computador e faz $NPE = -1$. Se a fila de espera não estiver vazia, esta função calcula IPPEV e TPEV do próximo evento do processo no topo da fila de espera.

c) Manipulação da transmissão de mensagens

No caso de transmissão de alguma mensagem, esta função deve calcular ICMT, incrementando TRP com os atrasos na comunicação, incluindo os tempos dispendidos para transmitir a mensagem entre os nós intermediários da rede (τ_{DMEXT}) e para roteá-la em cada nó por que passar (τ_{ROT}). Ela cria o registro RICMT, o qual deve ser armazenado em ordem crescente do valor de ICMT na lista de mensagens em trânsito (lista LRICMT), avança o ponteiro PEVA para o próximo evento do processo transmissor e soma a SARD o tempo dispendido por cada mensagem para transitar nos Transputers origem e destino da mensagem (τ_{ROT} e τ_{DMINT}), visando acumular os tempos que o Transputer fica indisponível para processamento devido à intervenção do Processo Roteador/Distribuidor.

Por fim, esta função deve verificar o regime em que se encontra o Transputer NTS. Se estiver em SEQ, ela mantém o processo transmissor escalado, calculando IPPEV e TPEV relativo ao próximo evento do processo transmissor. Se o regime for PAR, ela deve verificar a situação da fila de espera. Se estiver vazia mantém o processo transmissor escalado, calculando IPPEV e TPEV de seu próximo evento. Se não estiver vazia, coloca o processo transmissor no final da fila de espera e escala o processo no topo da mesma.

d) Manipulação da reativação de processos

O GEREDDE informa a identificação do Transputer a ter processo reativado (NTS) e a identificação do processo a ser reativado (NPR). A reativação significa que um processo inativo teve uma

mensagem sincronizada ou seu período de temporização concluído.

Se o regime for SEQ, reativa o processo que estava inativo, calculando IPPEV e TPEV de seu próximo evento. Se o regime for PAR, verifica se o Transputer NTS tem algum processo ativo. Se tiver, esta função mantém os últimos valores de IPPEV e TPEV, significando que não houve alteração do processo que ocupa a UCP do Transputer e coloca o processo reativado no final de FE. Se o Transputer NTS não tiver nenhum processo ativo, esta função escala o processo recém-reativado e calcula IPPEV e TPEV de seu próximo evento.

e) Manipulação de fim de processamento de uma tarefa

Ao ser chamada pelo GEREDE, esta função é informada da identificação do processo que encerrou seu processamento.

Se o regime for SEQ, ela verifica se o processamento no Transputer terminou. Em caso afirmativo, ela desabilita esse Transputer e compara com WESC o tempo de processamento consumido nesse Transputer (TRP) acrescido de SARD, atribuindo a WESC o maior valor, de forma que WESC sempre contenha o maior comprimento de escalação até então encontrado. Em caso negativo, significa que há outros regimes a serem simulados, de forma que ela chama a função "Controle da ordem de chamada e do regime de operação dos processos" e lhe passa a identificação do Transputer (NTS).

Se o regime for PAR, esta função verifica se o processamento em PAR acabou. Em caso afirmativo, esta função verifica se acabou o processamento no Transputer. Caso isso tenha

ocorrido, ela desabilita esse Transputer e compara WESC com TRP+SARD, atribuindo a WESC o maior valor. Caso tenha terminado apenas o processamento no PAR, mas ainda haja outros processos a serem simulados, ela chama a função "Controle da ordem de chamada e do regime de operação dos processos" e lhe passa a identificação do Transputer (NTS). Pode ocorrer ainda que haja outras tarefas em andamento no PAR. Nesse caso, podem ocorrer duas situações: se a FE estiver vazia, significando que o Transputer está ocioso, esta função faz IPPEV igual ao maior valor representável no computador e NPE= -1. Se FE não estiver vazia, ela chama o próximo processo da FE e calcula o IPPEV e TPEV de seu próximo evento.

f) Manipulação de temporização

Esta função recebe do GEREDE a identificação do processo que vai ser desescalado por temporização, avança seu ponteiro PEVA e verifica o regime do processo desescalado. Se ele estiver em PAR, deve-se calcular seu instante de retorno à fila de espera (IRFE) e escalar o próximo processo da fila, podendo ocorrer duas situações: se FE estiver vazia, faz IPPEV igual ao maior valor representável no computador e NPE= -1. Se FE não estiver vazia, calcula IPPEV e TPEV do próximo evento do processo recém-escalado. Se o processo desescalado estiver em SEQ, passa direto por esse evento, indo para o próximo e atualizando seu IPPEV e TPEV, levando em conta τ_{ATR} e τ_{EV} .

Esta função é responsável por colocar na lista LRIRFE, em ordem crescente do valor de IRFE, o registro RIRFE de cada processo em PAR desescalado por temporização, cuja composição

é mostrada na figura 4.14. A lista LRIRFE pode conter 0, 1 ou mais de um valor de IRFE por Transputer.

IRFE	P _{TEMP}
------	-------------------

Fig. 4.14 Composição do registro RIRFE

g) Manipulação de término de quantum de tempo

Ao ser notificada pelo GEREDE do término do quantum de tempo, esta função verifica se o processo utilizando a UCP do Transputer já utilizou sua cota de tempo. Em caso negativo, mantém o mesmo processo escalado e incrementa o contador de quanta de tempo VCQUAN. Em caso afirmativo, verifica a situação da fila de espera FE. Se estiver vazia, mantém o mesmo processo escalado. Se FE não estiver vazia, desescala o processo que ocupava a UCP do Transputer, subtrai de seu tempo de processamento o tempo já consumido, coloca-o no final de FE, escala o processo do topo de FE e calcula IPPEV e TPEV de seu próximo evento.

4.2.7 Especificação dos Testes de Aceitação do Programa Escalador

O plano de testes de aceitação foi dividido nas quatro etapas a seguir, objetivando analisar a conformidade do software gerado com os requisitos enunciados no item 4.2.2.

a) Testes de aceitação dos módulos do Programa Escalador

Estes testes se concentrarão, basicamente, na avaliação individual de cada um dos módulos do Programa Escalador,

devendo-se verificar o correto desempenho das operações a seguir.

- Gerenciador do Programa Escalador

- transferência dos arquivos-texto de entrada gerados pelo usuário para as estruturas de dados do Programa Escalador;
- seleção da melhor escalação sub-ótima; e
- transferência das informações contidas nas estruturas de dados do Programa Escalador para o arquivo-texto de saída.

- ESCEXT

- iniciação dos vetores VCP, VCPn e VN;
- funções de comparação utilizadas no algoritmo "quicksort" de ordenação de vetores; e
- algoritmos de distribuição das tarefas nos processadores incluindo a opção com compactação PAR.

- SIMREDE

- desempenho individual de cada uma das funções do GERERE e do MSEQ/PAR.

b) Testes de aceitação dos algoritmos de escalação do ESCEXT

Cada um dos dez algoritmos heurísticos de escalação deverá ser individualmente testado, de forma a verificar se a escalação gerada por cada um deles corresponde à sua especificação.

c) Testes de aceitação do modelo dinâmico da rede de Transputers

A aceitação do modelo dinâmico da rede de Transputers, representado pelo SIMREDE, consistirá na operação integrada

das funções do GEREDE e do MSEQ/PAR sob a direção da função "Coordenação da simulação dinâmica da rede de Transputers". Deve-se criar conjuntos de processos parcialmente ordenados e distribuí-los em diversos arranjos em SEQ e PAR em um ou mais Transputers. Medir-se-á, então, o tempo para executá-los em cada Transputer e se comparará o maior desses valores com o tempo WESCOBT obtido através da simulação do processamento das tarefas na rede de Transputers.

A expectativa é que os comprimentos de escalação determinados através da simulação (W_S) apresentem um desvio em relação ao real (W_R) que não ultrapasse 5%, conforme preconizado na alínea "a" do item 4.2.2.

d) Testes de aceitação do Programa Escalador integrado

Deve ser realizado um teste integrado de todos os níveis do Programa Escalador, visando verificar o funcionamento global do mesmo e comparar o comprimento das escalações obtidas (WESCOBT) com o comprimento da rota crítica (WCR), tecendo os devidos comentários a respeito da eficiência do Programa Escalador.

4.3 Diferenças entre o Programa Escalador e os Métodos Convencionais de Escalação Estática

O que distingue o Programa Escalador de outros métodos já publicados de escalação de tarefas em multiprocessadores é o modelamento dinâmico da rede de processadores, pois nos métodos convencionais, nem a rede de processadores nem os processadores

individuais são modelados. Tal fato acontece porque a comunicação entre os processadores, quer constituam sistemas fortemente ou fracamente acoplados, não desescala um processo. No entanto, ao se trabalhar com Transputers, devido à filosofia de comunicação sincronizada, há a desescalação dos processos comunicantes, dificultando a escalação por métodos sem modelamento.

Outra simplificação, corriqueiramente assumida, é que todas as entradas de uma tarefa ocorrem no início de seu processamento, ao passo que todas as saídas ocorrem no seu final. Isto também não acontece aqui, pois a comunicação sincronizada escala e desescala processos, de forma que a hipótese de entrada e saída de dados concentradas, respectivamente, no início e no final do processamento da tarefa é inaplicável aos Transputers.

Por fim, outra peculiaridade dos Transputers são os diferentes regimes de operação disponíveis. No caso deste trabalho, o Programa Escalador tem à sua disposição as construções SEQ e PAR.

5 - PROJETO DO PROGRAMA ESCALADOR

Este capítulo descreve o projeto do Programa Escalador, sendo dividido nas seguintes seções:

- projeto básico do Programa Escalador; e
- projeto detalhado do Programa Escalador.

5.1 Projeto Básico do Programa Escalador

Nesta seção é detalhada cada uma das funções apresentadas no item 4.2.6, é estabelecido o diagrama de fluxo de dados e é definida a base de dados do Programa Escalador.

5.1.1 Detalhamento das Funções do Gerenciador do Programa Escalador

O Gerenciador do Programa Escalador apresenta o seguinte encadeamento de chamada de funções:

- formata_entrada;
- inicia_escalador;
- para cada algoritmo ALGOR de escalação:
 - escala(ALGOR);
 - simula; e
 - seleciona_escalação; e
- formata_saída.

Conforme foi visto no sub-item 4.2.6.1, as funções "inicia_escalador", "escala" e "simula" não estão incluídas no Gerenciador do Programa Escalador.

5.1.1.1 Formatação dos Dados de Entrada (formata_entrada)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Esta função deve primeiramente ler o arquivo de entrada de dados dos processos, cujo formato está descrito no sub-item 6.2.1.1 e convertê-lo em estruturas de dados acessíveis ao Programa Escalador.

O primeiro dado coletado deve ser atribuído a n .

A seguir, esta função deve montar a matriz de precedência MPREC, analisando as linhas seguintes do arquivo de entrada. Em cada linha devem ser colocados os processos sucessores imediatos (filhos) de cada processo pai, obedecendo a ordem crescente de seu número de identificação. Assim, deve-se inicialmente colocar a identificação dos filhos de P_0 , na linha seguinte a identificação dos filhos de P_1 e assim por diante. A identificação do pai corresponde à linha i da matriz MPREC e a identificação de seus filhos à coluna j . A matriz MPREC deve ser iniciada com 0 e as posições (i,j) em que houver relações de precedência devem ser preenchidas com 1.

Após ter feito isso, a função deve gerar os vetores $V\tau_{SEQ}$ e $V\tau_{PAR}$, a partir dos pares de informações contidas nas linhas seguintes do arquivo de entrada.

O próximo passo a ser realizado pela função é montar o vetor de listas de eventos VLEV. Para a formação de cada registro EVENTO, o arquivo de entrada é consultado e cada linha gera um novo evento. Isso ocorre até encontrar o evento "fim de processamento", significando que acabaram os eventos daquele processo. Esse procedimento é repetido n vezes, correspondendo aos n processos a serem escalados. Os valores coletados de τ_{SEQ} , τ_{PAR} , $\tau_{EV,SEQ}$, $\tau_{EV,PAR}$ e τ_{ATR} devem ser multiplicados por 64 para convertê-los de ticks não prioritários para microsegundos, que é a unidade de tempo utilizada internamente no Programa Escalador.

Após ter montado VLEV, esta função deve ler o arquivo de dados de entrada da rede de Transputers, cujo formato está descrito no item 6.2.2, e atribuir o primeiro dado a m e, em seguida, montar a matriz MNNI, colocando em cada linha de MNNI os dados contidos em cada linha do arquivo de entrada.

Esta função realiza uma análise junto aos dados contidos nos arquivos de entrada, visando buscar erros ou inconsistências, sendo que as seguintes situações são verificadas:

- em uma linha só deve haver os parâmetros previstos para ela, verificando-se se há excesso ou falta dos mesmos;
- o número de processos n deve ser menor ou igual a 999 (o processo P_0 é fictício, usado como raiz do grafo de precedência) e o número de Transputers m deve ser menor ou igual a 100;

- identificação dos processos deve estar entre 0 e $n+1$ (P_{n+1} é um processo fictício que encerra o grafo de precedência);
- o número de identificação do processo filho nas relações de precedência deve estar entre 1 e $n+1$;
- o número sequencial da mensagem (NSM) deve estar entre 1 e 99;
- tipo do evento deve estar entre 1 e 4; e
- arquivo de dados de entrada de processos ou de Transputers não pôde ser aberto.

c) Parâmetros de saída

Nenhum.

5.1.1.2 Seleção da Melhor Escalação Sub-ótima (seleciona_escalção)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Comparar o valor de WESC, obtido na simulação, com o valor da melhor escalação até então obtida (WESCOBT), visando selecionar a melhor escalação entre VLLESC e VLLESCOB. Se $WESC \geq WESCOBT$ não faz nada. Caso contrário, faz $WESCOBT = WESC$ e $VLLESCOB = VLLESC$.

c) Parâmetros de saída

Nenhum.

5.1.1.3 Formatação dos Dados de Saída (formata_saída)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Os dados de saída são apresentados ao usuário em um arquivo-texto, cujo formato é mostrado no item 6.2.3. Esta função se encarrega de converter a forma de representação do vetor de listas VLLESCOBT em um modo utilizável pelo usuário. Para tanto, a função percorre VLLESCOBT e, para cada lista ligada correspondente a um Transputer da rede, executa as seguintes operações:

- busca a identificação do Transputer (T_i) e a coloca em uma nova linha do arquivo de saída;
- percorre a lista ligada de regimes, relativa a T_i , desde seu início até seu fim e verifica o tipo de cada regime. Se for SEQ, coloca em uma nova linha do arquivo de saída a identificação do processo em SEQ. Se for PAR, coloca em uma nova linha do arquivo de saída a identificação dos processos em PAR; e

- tendo terminado de percorrer o vetor VLLESCOB_T para os m Transputers, esta função coloca nas linhas seguintes do arquivo de saída os valores de WESCOB_T e WCR.

c) Parâmetros de saída

Nenhum.

5.1.2 Detalhamento das Funções do ESCEXT

Analisando-se os algoritmos de escalação heurísticos propostos, percebe-se que os seguintes procedimentos são comuns a todos eles:

a) Procedimento de ordenação

Ordenar as tarefas, segundo algum critério, empregando algoritmos de ordenação do tipo Quicksort, para gerar o vetor de prioridade VPRIOR.

b) Procedimento de escalação das tarefas

Uma vez se dispondo de um vetor ordenado, executa-se a distribuição das tarefas pelos Transputers no regime SEQ ou PAR, dependendo do algoritmo, buscando um balanceamento de carga computacional nos mesmos.

Adicionalmente, as seguintes funções, cujas tarefas foram escaladas em SEQ, exigem que, uma vez que se tenha uma escalação pronta, se coloque em PAR as tarefas de mesmo nível N alocadas no mesmo Transputer T_i e que as tarefas em regimes PAR consecutivos sejam compactadas em um único PAR:

- Geração de escalação pelo algoritmo CP/NSI/PAR;

- Geração de escalação pelo algoritmo $N/\tau_{SEQ}/CP/PAR$;
- Geração de escalação pelo algoritmo $N/CP/PAR$; e
- Geração de escalação pelo algoritmo $N/CPn/NSI/PAR$.

A chamada das funções do ESCEXT é feita de acordo com o valor de ALGOR, como mostrado na tabela 5.1.

ALGOR	Função chamada
CP/NSI	Geração de escalação pelo algoritmo CP/NSI
CP/NSI/PAR	Geração de escalação pelo algoritmo CP/NSI/PAR
$N/\tau_{SEQ}/CP$	Geração de escalação pelo algoritmo $N/\tau_{SEQ}/CP$
$N/\tau_{SEQ}/CP/PAR$	Geração de escalação pelo algoritmo $N/\tau_{SEQ}/CP/PAR$
N/CP	Geração de escalação pelo algoritmo N/CP
$N/CP/PAR$	Geração de escalação pelo algoritmo $N/CP/PAR$
$N/CPn/NSI$	Geração de escalação pelo algoritmo $N/CPn/NSI$
$N/CPn/NSI/PAR$	Geração de escalação pelo algoritmo $N/CPn/NSI/PAR$
LPT	Geração de escalação pelo algoritmo LPT
MTP	Geração de escalação pelo algoritmo MTP

Tab. 5.1 Tabela de escolha de algoritmos heurísticos de acordo com valor de ALGOR

5.1.2.1 Iniciação dos Parâmetros para Geração de Escalações (inicia_escalador)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Esta função tem um escopo quádruplo: calcular VCP, VCP_n e VN e iniciar VPRIOR, de acordo com o exposto a seguir.

b1) Cálculo de VCP

O algoritmo que calcula o comprimento da rota crítica CP de cada tarefa é mostrado a seguir:

- fazer $VCP(P_i) = -1$, para $0 \leq i \leq n+1$;
- calcular $VCP(P_0)$; e
- fazer $WCR = VCP(P_0)$.

Trata-se de um algoritmo recursivo. Para calcular $VCP(P_i)$, nota-se que se $VCP(P_i) = -1$, significa que o valor de CP de P_i ainda não foi calculado. Neste caso, esta função deve calcular o valor de CP dos filhos de P_i e assim, recursivamente, até chegar ao cálculo de CP do processo P_{n+1} que é conhecido, pois P_{n+1} não tem filhos e, por definição, $VCP(P_i) = V\tau_{SEQ}(P_i)$ se P_i não tem filhos. Enfatiza-se que o único processo sem filhos no grafo de precedência é P_{n+1} , o qual foi concebido justamente com essa finalidade. Por outro lado, se P_i tem filho(s), o valor de CP de P_i é calculado por $VCP(P_i) = V\tau_{SEQ}(P_i) + \max(VCP(P_j))$, onde P_j é filho de P_i .

O seguinte algoritmo implementa o cálculo de $VCP(P_i)$ quando $VCP(P_i) = -1$:

- fazer $CP = 0$;
- para cada P_j , filho de P_i :
 - calcular $VCP(P_j)$;
 - se $VCP(P_j) > CP$, fazer $CP = VCP(P_j)$; e
- fazer $VCP(P_i) = CP + V\tau_{SEQ}(P_i)$.

b2) Cálculo de VCPn

Para normalizar o comprimento da rota crítica de cada tarefa, utiliza-se o seguinte roteiro:

- calcular a média dos τ_{SEQ} da seguinte forma:

$$\tau_{SEQ,MÉDIO} = \frac{\sum_{i=1}^n V\tau_{SEQ}(P_i)}{n} \quad (5.1)$$

- determinar o valor do fator de correção FC apresentado na tabela 4.1, o qual pode ser diretamente obtido através do seguinte procedimento:

- fazer $FC=10$;
- enquanto $FC < \tau_{SEQ,MÉDIO}$, fazer $FC=FC*10$; e
- fazer $FC=FC/10$;

- calcular o vetor VCPn através da seguinte equação:

$$VCPn(P_i) = VCP(P_i) / FC + 0.5 \quad (\text{para } 1 \leq i \leq n) \quad (5.2)$$

$VCPn(P_i)$ deve ser arredondado para o inteiro mais próximo, o que é obtido através da soma de 0.5.

b3) Cálculo de VN

Para calcular o nível de cada tarefa, utiliza-se o seguinte algoritmo recursivo:

- fazer $VN(P_i) = -1$ para $0 \leq i \leq n+1$; e
- calcular $VN(P_{n+1})$;

Para calcular $VN(P_i)$, deve-se notar que se $VN(P_i) = -1$, significa que o valor do nível N de P_i ainda não foi calculado. Neste caso, esta função deve calcular o valor

do nível das tarefas predecessoras imediatas de P_i e assim recursivamente, até chegar ao nível de P_0 , cujo nível é conhecido, pois P_0 não tem pai e, por definição, $VN(P_i)=0$ se P_i não tem pais. Por outro lado, se P_i tem pai(s), o valor de N para P_i é calculado por $VN(P_i)=\max(VN(P_j))$, onde P_j é pai de P_i . O seguinte algoritmo implementa essa idéia quando $VN(P_i)=-1$:

- fazer $NA=-1$;
- para cada P_j , pai de P_i :
 - calcular $VN(P_j)$;
 - se $VN(P_j)>NA$, fazer $NA=VN(P_j)$; e
- fazer $VN(P_i)=NA+1$.

b4) Iniciação de VPRIOR

Para iniciar o vetor VPRIOR faz-se $VPRIOR(i)=i$ para $1 \leq i \leq n$.

c) Parâmetros de saída

Nenhum.

5.1.2.2 Geração de Escalação pelo Algoritmo CP/NSI (algoritmo_ CP/NSI)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função gera o vetor VPRIOR, ordenado segundo o critério CP/NSI, de acordo com a alínea "b" do sub-item 4.2.6.2. O

algoritmo de ordenação utilizado neste trabalho é do tipo "Quicksort" (AHO et al., 1983) e (WIRTH, 1986).

Em seguida, a função gera a escalação das tarefas no regime SEQ, cumprindo-se o seguinte roteiro:

- iniciação da escalação, executada através dos seguintes passos:
 - fazer $VTAPT(T_i)=0$ e $VLLESC(T_i)=vazia$ para $1 \leq i \leq m$; e
 - alocar as m primeiras tarefas de $VPRIOR$ aos m processadores disponíveis, seguindo a ordem crescente da identificação dos Transputers. Para $VPRIOR(i)$, com i de 1 a m , a cada tarefa P_j a ser alocada em cada Transputer T_i , os seguintes passos devem ser executados:
 - fazer $VTAPT(T_i)=V\tau_{SEQ}(P_j)$; e
 - colocar no final da lista $VLLESC(T_i)$ o elemento REGIME montado com P_j ;
- atribuição da próxima tarefa do vetor $VPRIOR$ ao processador com menor carga computacional acumulada $TAPT$. Caso dois ou mais processadores se enquadrem nessa condição, a tarefa é alocada ao processador com menor número de identificação. Repete-se este procedimento até se percorrer todo o vetor $VPRIOR$. Portanto, para $VPRIOR(i)$, $m+1 \leq i \leq n$, a cada tarefa P_j a ser alocada, os seguintes passos devem ser realizados:
 - somar $V\tau_{SEQ}(P_j)$ a $VTAPT(T_k)$, onde T_k corresponde ao Transputer com menor valor de $TAPT$; e
 - montar o registro REGIME com P_j e colocá-lo como último elemento da lista $VLLESC(T_k)$.

c) Parâmetros de saída

Nenhum.

5.1.2.3 Geração de Escalação pelo Algoritmo CP/NSI/PAR (algoritmo_CP/NSI/PAR)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função utiliza o mesmo vetor VPRIOR e o mesmo procedimento de iniciação da escalação usados pela função "Geração de escalação pelo algoritmo CP/NSI".

A escalação das $(n-m)$ tarefas restantes é realizada, repetindo o seguinte procedimento até percorrer todo o vetor VPRIOR, isto é, para $VPRIOR(i)$, $m+1 \leq i \leq n$, atribui-se a próxima tarefa de $VPRIOR(i)$, no caso P_k , ao processador com menor carga computacional acumulada $TAPT$, no caso T_1 . Compara-se então o nível N do último processo alocado em $VLLESC(T_1)$, no caso P_j , com o nível N do processo P_k , resultando:

- se ambos os níveis forem os mesmos, deve-se modificar o último registro REGIME colocado em $VLLESC(T_1)$, inserindo P_k no fim de REGIME e incrementando o contador do número de tarefas em PAR contidas em REGIME. Deve-se então verificar o regime em que a última tarefa em $VLLESC$ foi alocado, que pode resultar em uma das duas seguintes opções:
 - o último processo foi alocado em SEQ. Neste caso, deve-se passá-lo, juntamente com P_j , para o regime PAR, e calcular $TAPT$, considerando a mudança de SEQ para PAR de P_j e o tempo de P_k em PAR, ou seja, deve-se fazer:

$$VTAPT(T_1) = VTAPT(T_1) + V\tau_{PAR}(P_j) - V\tau_{SEQ}(P_j) + V\tau_{PAR}(P_k); \text{ e}$$

- o último processo foi alocado em PAR. Neste caso, deve-se alocar o processo P_k em PAR junto com P_j e calcular TAPT, considerando apenas $V\tau_{PAR}(P_k)$, ou seja, deve-se fazer:

$$VTAPT(T_1) = VTAPT(T_1) + V\tau_{PAR}(P_k); \text{ e}$$

- se os níveis forem diferentes, deve-se alocar o processo P_k em SEQ, montando o registro REGIME com P_k , colocando-o como último elemento da lista $VLLESC(T_1)$ e calcular TAPT da forma indicada a seguir:

$$VTAPT(T_1) = VTAPT(T_1) + V\tau_{SEQ}(P_k).$$

c) Parâmetros de saída

Nenhum.

5.1.2.4 Geração de Escalação pelo Algoritmo $N/\tau_{SEQ}/CP$ (algoritmo_ $N/\tau_{SEQ}/CP$)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função gera o vetor VPRIOR, ordenado segundo o critério apresentado na alínea "d" do sub-item 4.2.6.2. Com o vetor VPRIOR pronto, gera-se a escalação das tarefas em SEQ, usando o mesmo método empregado na função "Geração de escalação pelo algoritmo CP/NSI".

c) Parâmetros de saída

Nenhum.

**5.1.2.5 Geração de Escalação pelo Algoritmo $N/\tau_{SEQ}/CP/PAR$
(algoritmo_ $N/\tau_{SEQ}/CP/PAR$)**

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função compacta, no modo PAR, as tarefas colocadas no mesmo Transputer e pertencentes ao mesmo nível, geradas pelo algoritmo descrito na função "Geração de escalação pelo algoritmo $N/\tau_{SEQ}/CP$ ". Seu escopo é similar ao da função "Geração da escalação pelo algoritmo $CP/NSI/PAR$ "

c) Parâmetros de saída

Nenhum.

5.1.2.6 Geração de Escalação pelo Algoritmo N/CP (algoritmo_ N/CP)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função ordena o vetor VPRIOR, usando o critério exposto na alínea "f" do sub-item 4.2.6.2. Em seguida, gera a escalação das tarefas no regime SEQ, usando o mesmo método empregado na função "Geração de escalação pelo algoritmo CP/NSI".

c) Parâmetros de saída

Nenhum.

5.1.2.7 Geração de Escalação pelo Algoritmo N/CP/PAR (algoritmo_
N/CP/PAR

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função compacta no modo PAR as tarefas colocadas no mesmo Transputer e pertencentes ao mesmo nível, de acordo com a escalação gerada pela função "Geração de escalação pelo algoritmo N/CP". Seu escopo é similar ao da função "Geração da escalação pelo algoritmo CP/NSI/PAR".

c) Parâmetros de saída

Nenhum.

5.1.2.8 Geração de Escalação pelo Algoritmo N/CPn/NSI (algoritmo_N/CPn/NSI)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função gera o vetor VPRIOR, ordenado conforme o critério apresentado na alínea "h" do sub-item 4.2.6.2. Em seguida, ela gera a escalação das tarefas no regime SEQ, usando o mesmo método empregado na função "Geração de escalação pelo algoritmo CP/NSI".

c) Parâmetros de saída

Nenhum.

5.1.2.9 Geração de Escalação pelo Algoritmo N/CPn/NSI/PAR (algoritmo_N/CPn/NSI/PAR)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função compacta no modo PAR as tarefas colocadas no mesmo Transputer e pertencentes ao mesmo nível, de acordo com a escalação gerada pela função "Geração de escalação pelo

algoritmo N/CPn/NSI". Seu escopo é similar ao da função "Geração da escalação pelo algoritmo CP/NSI/PAR".

c) Parâmetros de saída

Nenhum.

5.1.2.10 Geração de Escalação pelo Algoritmo LPT (algoritmo_LPT)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Esta função cria o vetor VPRIOR, segundo o critério constante na alínea "j" do item 4.2.6.2. Em seguida, ela promove a escalação das tarefas em PAR nos processadores, através dos seguintes passos:

- iniciação da escalação, que é similar à iniciação realizada no sub-item 5.1.2.2, exceto que $VTAPT(T_i) = V\tau_{PAR}(P_j)$;
- atribuição da próxima tarefa do vetor VPRIOR ao processador com menor carga computacional acumulada TAPT. Caso dois ou mais processadores se enquadrem nessa condição, a tarefa é alocada ao processador com menor número de identificação. Repete-se este procedimento até se percorrer todo o vetor VPRIOR. Portanto, para $VPRIOR(i)$, $m+1 \leq i \leq n$, a cada tarefa P_j a ser alocada, os seguintes passos devem ser realizados:
 - somar $V\tau_{PAR}(P_j)$ a $VTAPT(T_k)$, onde T_k corresponde ao Transputer com menor valor de TAPT; e

- colocar P_j no final do registro REGIME, o qual será o único elemento da lista VLLESC(T_k), visto que todas as tarefas estão sendo alocadas em PAR.

c) Parâmetros de saída

Nenhum.

5.1.2.11 Geração de Escalação pelo Algoritmo MTP (algoritmo_MTP)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

ALGOR.

b) Escopo da função

Este algoritmo consiste basicamente nos seguintes passos:

- iniciação da escalação, que corresponde a:
 - criar o vetor VPRIOR, colocando as tarefas em ordem decrescente dos tempos de processamento τ_{PAR} ;
 - calcular a média dos tempos de processamento, através da soma dos tempos τ_{PAR} e da divisão desse resultado pelo número de processadores, arredondando essa média sempre para o inteiro imediatamente superior. Esse cálculo é realizado da seguinte forma:

$$\tau_{PAR, MÉDIO} = \frac{\sum_{j=1}^n V\tau_{PAR}(P_j)}{m} \quad (5.3)$$

- fazer VTAPT(T_i)=0 para $1 \leq i \leq m$;

- alocar a primeira tarefa do vetor VPRIOR ao primeiro processador livre (T_i) e, em seguida, alocar outras tarefas em PAR a esse mesmo processador, procurando carregá-lo com as tarefas mais longas, totalizando o tempo acumulado de processamento $VTAPT(T_i)$ a cada nova tarefa escalada, procurando igualar, mas nunca ultrapassar a média $\tau_{PAR,MÉDIO}$, marcando em um vetor booleano auxiliar as tarefas de VPRIOR já escaladas e passando para o próximo elemento de VPRIOR, caso a próxima tarefa leve a média a ser ultrapassada.

Este passo deve ser repetido para cada Transputer T_i da rede da seguinte forma: para cada processo P_j não alocado, pertencente a VPRIOR, tomado na sequência, verificar se $VTAPT(T_i) + V\tau_{PAR}(P_j) \leq \tau_{PAR,MÉDIO}$. Caso a relação seja verdadeira, deve-se alocar P_j em PAR no final do registro REGIME, somar $V\tau_{PAR}(P_j)$ a $VTAPT(T_i)$ e marcar P_j no vetor booleano auxiliar como escalado. Caso a relação não seja satisfeita, deve-se tomar o próximo processo de VPRIOR, sendo que se VPRIOR tiver chegado ao fim, deve-se partir para o próximo Transputer.

Visto que a escalação das tarefas é feita em PAR, resultará um único registro REGIME em cada lista LLESC; e

- distribuir as tarefas que eventualmente sobraram em VPRIOR (resíduo) dentre os processadores com menor carga computacional, repetindo o seguinte procedimento até escalar todas as tarefas de VPRIOR:
 - buscar o primeiro elemento não escalado de VPRIOR (P_j) e alocá-lo em PAR no Transputer com menor valor de $VTAPT(T_i)$, colocando esse processo no único REGIME disponível de VLLESC(T_i);

- somar $V_{\tau PAR}(P_j)$ a $VTAPT(T_i)$; e
- marcar P_j como escalado no vetor booleano auxiliar.

c) Parâmetros de saída

Nenhum.

5.1.3 Detalhamento das Funções do GEREDE

Não se consegue, dentro do nível do SIMREDE, estabelecer, de antemão, uma sequência de chamada de funções. Quem realiza a chamada das funções, definindo sua ordem de chamada, é a função "Coordenação da simulação dinâmica da rede de Transputers", que toma as decisões em tempo de processamento, como fruto dos eventos existentes nos processos alocados aos Transputers.

Pode-se, no entanto, apresentar um diagrama que demonstre quais funções do SIMREDE podem ser chamadas diretamente da função coordenadora da simulação e quais devem ser chamadas através de outras funções, caracterizando assim as interrelações entre funções no SIMREDE, conforme mostrado na figura 5.1.

Fig. 5.1 Diagrama da interrelação entre as funções do SIMREDE

5.1.3.1 Coordenação da Simulação Dinâmica da Rede de Transputers (coordena_simulação)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Para executar a simulação da rede de Transputers pelo SIMREDE chama-se esta função, a qual, ao ser chamada, aciona a função "Iniciação da simulação dinâmica da rede de Transputers". Em seguida, ela entra em uma malha controlada pelo valor do contador CTA.

Enquanto $CTA > 0$, esta função deve executar os seguintes passos:

- buscar o menor valor de tempo dentre as quatro seguintes opções, igualando o valor escolhido a TRP:
 - ITQT;
 - menor valor de IPPEV retirado do vetor VIPPEV, dentre os Transputers ativos (VTA(i) verdadeiro);
 - menor valor de IRFE retirado da lista LRIRFE, a qual somente deve ser consultada se não estiver vazia; e
 - menor valor de ICMT retirado da lista LRICMT, a qual somente deve ser consultada se não estiver vazia.

Caso haja dois ou mais valores dentre os quatro parâmetros acima que satisfaçam a condição de mínimo valor, deve-se selecioná-los obedecendo a seguinte fila de prioridade: $IPPEV > ITQT > ICMT > IRFE$. Por outro lado, caso haja dois

ou mais valores coincidentes dentro das estruturas LRICMT, LRIRFE ou VIPPEV, deve-se tomar cada um deles por vez, na ordem em que aparecem em suas respectivas estruturas.

Caso a escolha recaia sobre IPPEV, deve-se armazenar em NTS a identificação do Transputer selecionado e verificar qual evento lhe corresponde: recepção ou transmissão de mensagem, temporização ou fim de processamento.

- chamar uma das seguintes funções, de acordo com a escolha feita no parágrafo anterior:

Escolha feita	Função a ser chamada
IRFE	Controle de IRFE (GEREDE)
ICMT	Controle de ICMT (GEREDE)
ITQT	Controle de ITQT (GEREDE)
IPPEV (com próximo evento recepção)	Manipulação da recepção de mensagens (MSEQ/PAR)
IPPEV (com próximo evento transmissão)	Manipulação da transmissão de mensagens (MSEQ/PAR)
IPPEV (com próximo evento temporiz.)	Manipulação de temporização (MSEQ/PAR)
IPPEV (com próximo evento fim proces.)	Manipulação de fim processamento de uma tarefa (MSEQ/PAR)

Tab. 5.2 Tabela de escolha de chamada de funções de acordo com o próximo evento previsto em cada Transputer da rede

Quando CTA chega a 0, deve-se retornar ao Gerenciador do Programa Escalador.

c) Parâmetros de saída

Os parâmetros de saída são determinados através da escolha da função feita na tabela 5.2, conforme indicado a seguir:

- se a função chamada é do MSEQ/PAR, tem-se NTS e NPE como parâmetros de saída, exceto para a função "Manipulação de fim de processamento de uma tarefa", a qual recebe apenas NTS; e
- se a função chamada é do GEREDE não há nenhum parâmetro de saída.

Ao retornar para o Gerenciador do Programa Escalador não há nenhum parâmetro de saída.

5.1.3.2 Iniciação da Simulação Dinâmica da Rede de Transputers (inicia_simulação)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Executa a iniciação das seguintes estruturas de dados:

- LRICMT: vazia;
- LRIRFE: vazia;
- TRP=0;
- WESC=0;
- CTA=m, onde m é o número de Transputers na rede;
- ITQT=512 (equivale a 8 ticks não prioritários, conforme preconizado na alínea "c" do item 3.4.1);
- VLRMTAS(P_j): vazio para $1 \leq j \leq n$;
- VRPACM(P_j): vazio ($P_T=-1$) para $1 \leq j \leq n$;

- $VPEVA(P_j)$: aponta para o primeiro evento da lista $VLEV(P_j)$ para $1 \leq j \leq n$;
- $V_{\tau_{PEV,PAR}}(P_j)$ = valor do campo $TEVPAR$ da posição apontada por $VPEVA(P_j)$ em $VLEV(P_j)$ para $1 \leq j \leq n$;
- $VPLREGESC(T_i)$: aponta para o primeiro regime de operação da lista $VLLESC(T_i)$ para $1 \leq i \leq m$;
- $VT_A(T_i)$ = verdadeiro para $1 \leq i \leq m$; e
- $VSARD(T_i) = 0$ para $1 \leq i \leq m$.

Em seguida, esta função monta o vetor $VALOCA$, percorrendo as listas $VLLESC(T_i)$ para os m Transputers ($1 \leq i \leq m$) e, dentro de cada lista $VLLESC$, buscam-se todos os processos nela alocados e, para cada processo P_j de $VLLESC(T_i)$, faz $VALOCA(P_j) = T_i$.

Ela então chama m vezes a função "Controle da ordem de chamada e do regime de operação dos processos", sendo que antes de cada chamada incrementa NTS (o qual é iniciado com 0).

Por fim, ela retorna para a função que a chamou (Coordenação da simulação dinâmica da rede de Transputers).

c) Parâmetros de saída

Em cada uma das m chamadas da função "Controle da ordem de chamada e do regime de operação dos processos", esta função passa a identificação NTS de um dos Transputers da rede. Ao retornar à função "Coordenação da simulação dinâmica da rede de Transputers" não passa parâmetro nenhum.

5.1.3.3 Controle de IRFE (controla_IRFE)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Esta função retira o registro RIRFE do topo da lista LRIRFE, o qual se refere ao processo que encerrou a temporização e faz NPR igual à identificação do processo que encerrou a temporização (P_{TEMP}). Ela então chama a função "Manipulação da reativação de processos".

c) Parâmetros de saída

NPR.

5.1.3.4 Controle de ITQT (controla_ITQT)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Atualiza o valor de ITQT através da seguinte atribuição:

$$ITQT = ITQT + 1024 \quad (5.4)$$

Ela então chama a função "Manipulação de término de quantum de tempo" para todos os Transputers simulados ativos ($VTA(T_i)$ verdadeiro), que estejam operando em regime PAR ($VREG(T_i) = PAR$) e que não estejam com processo inválido, isto é, com $VNPE(T_i) \geq -1$. Após ter feito isso, retorna à função "Coordenação da simulação dinâmica da rede de Transputers".

c) Parâmetros de saída

Quando esta rotina chama a função "Manipulação de término de quantum de tempo" ela passa $NTS=T_i$ como parâmetro de saída. Quando retorna à função "Coordenação da simulação dinâmica da rede de Transputers" não passa parâmetro nenhum.

5.1.3.5 Controle de ICMT (controla_ICMT)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

Nenhum.

b) Escopo da função

Quando esta função é chamada, ela retira do topo da lista LRICMT o RICMT, verifica no RICMT qual o processo destino da mensagem (P_R) e busca em RPACM de P_R se seu respectivo RIRM chegou, para verificar se P_R aguarda esta mensagem. Esta busca consiste em verificar se a identificação de P_T e de NSM em ambos os registros coincide, podendo redundar nas duas seguintes respostas:

- se o RIRM está disponível (há sincronização), esta função cancela o RPACM de P_R , fazendo $P_T=-1$, faz $NPR=P_R$ e chama a função "Manipulação da reativação de processos"; e
- se o RIRM não está disponível (não há sincronização), esta função cria o registro RMTAS a partir do RICMT, aproveitando apenas os campos P_T e NSM e o insere no final

de VLRMTAS(P_R), retornando então à função "Coordenação da simulação dinâmica da rede de Transputers".

c) Parâmetros de saída

Se há sincronização, NPR é o parâmetro de saída; se não há sincronização não há nenhum parâmetro de saída.

5.1.4 Detalhamento das Funções do MSEQ/PAR

Descreve-se detalhadamente, neste item, as funções constituintes do MSEQ/PAR.

5.1.4.1 Controle da Ordem de Chamada e do Regime de Operação dos Processos (controla_chamada)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NTS.

b) Escopo da função

Verifica qual é o próximo regime da lista VLLESC(NTS) apontado por VPLREGESC(NTS), podendo recair em um dos seguintes casos:

- se o próximo regime é SEQ, esta função verifica a identificação do processo a ser escalado (NPE) no Transputer identificado por NTS e faz VNPE(NTS)=NPE e VREG(NTS)=SEQ. Após isso é chamada a função auxiliar "Atualização de VIPPEV e VTPEV" descrita na alínea "a" do sub-item 5.1.4.8;

- se o próximo regime é PAR, esta função verifica quantos são os processos a serem escalados em PAR na lista VLLESC(NTS) e inicia VCPAPAR(NTS) com esse número, gera uma cópia em VLFE(NTS) da lista de processos em PAR (fila de espera), faz VREG(NTS)=PAR e executa a escalação do processo VNPE(NTS), através dos seguintes passos:
 - iguala VNPE(NTS) ao último processo da fila de espera FE;
 - retira de VLFE(NTS) o processo NPE(NTS);
 - faz VCQUAN(NTS)=0; e
 - chama a função auxiliar "Atualização de VIPPEV e VTPEV".

Esta função avança então a posição VPLREGESC(NTS) para o próximo regime da lista VLLESC(NTS).

Por fim, ela retorna à função que a chamou (Iniciação da simulação dinâmica da rede de Transputers ou Manipulação de fim de processamento de uma tarefa).

c) Parâmetros de saída

Nenhum.

5.1.4.2 Manipulação da Recepção de Mensagens (recebe_mensagem)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NTS e NPE.

b) Escopo da função

Esta função monta o registro RIRM, retirando P_T e NSM da posição apontada por VPEVA(NPE) em VLEV(NPE). Em seguida, ela

avança o ponteiro VPEVA(NPE) para o próximo evento de VLEV(NPE), o qual é suposto sempre apontar para o próximo evento a ocorrer no processo NPE, e copia para $V\tau_{PEV,PAR}(NPE)$ o campo TEVPAR da posição apontada por VPEVA(NPE) em VLEV(NPE). Esta cópia é necessária para evitar que a função "Manipulação de término de quantum de tempo" eventualmente altere o valor de $\tau_{EV,PAR}$ na lista VLEV(NPE) original.

A função então verifica na lista VLRMTAS($P_R=NPE$) se o RMTAS relativo a essa mensagem já chegou, comparando se há coincidência nos valores de P_T e de NSM contidos no RIRM e nos RMTAS. Podem ocorrer as duas seguintes respostas:

- RMTAS já chegou (há sincronização). Neste caso, a função retira RMTAS de sua lista, elimina-o juntamente com RIRM e chama a função auxiliar "Atualização de VIPPEV e VTPEV"; e
- RMTAS ainda não chegou (não há sincronização). Neste caso, a função coloca o RIRM em VRPACM(P_R) e verifica o valor de VREG(NTS), que pode assumir as seguintes condições:
 - se VREG(NTS)=SEQ, significa que o Transputer NTS ficará ocioso, aguardando reativação de processo. Faz então $VIPPEV(NTS)=$ maior valor representável no computador; e
 - se VREG(NTS)=PAR, escala o topo da fila de espera empregando a função auxiliar "Escalação de processo" descrita na alínea "b" do sub-item 5.1.4.8.

Por fim, retorna à função que a chamou (Coordenação da simulação dinâmica da rede de Transputers).

c) Parâmetros de saída

Nenhum.

5.1.4.3 Manipulação da Transmissão de Mensagens (transmite_mensagem)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NTS e NPE.

b) Escopo da função

Esta função calcula o valor de ICMT, agregando a TRP os atrasos ocorridos na comunicação, de acordo com as seguintes equações (vide item 3.4.3):

- se T_T for diferente de T_R , então:

$$ICMT = TRP + (NNI+1) * \tau_{DMEXT} + NNI * \tau_{ROT} \quad (5.5)$$

$$VSARD(T_T) = VSARD(T_T) + \tau_{ROT} + \tau_{DMINT} \quad (5.6)$$

$$VSARD(T_R) = VSARD(T_R) + \tau_{ROT} + 2 * \tau_{DMINT} \quad (5.7)$$

- se T_T for igual a T_R , então:

$$ICMT = TRP \quad (5.8)$$

$$VSARD(NTS) = VSARD(NTS) + \tau_{ROT} + 3 * \tau_{DMINT} \quad (5.9)$$

Os parâmetros para o cálculo dessas equações são obtidos da seguinte forma:

- $T_T = NTS$;
- τ_{DMINT} e τ_{DMEXT} : retirados da posição apontada por VPEVA(NPE) em VLEV(NPE);
- $T_R = VALOCA(P_R)$;
- NNI: retirado de MNNI(T_T, T_R); e
- τ_{ROT} : seu valor é assumido igual a $10 \mu s$, conforme foi exposto no item 3.5.4.

Esta função então monta o registro RICMT da seguinte forma:

- $P_T = NPE$;
- P_R , NSM: retirados da posição apontada por $VPEVA(NPE)$ em $VLEV(NPE)$; e
- ICMT: calculado anteriormente na equação (5.5) ou (5.8).

A função chama, então, a função auxiliar "Inserção de registro em lista em ordem crescente", descrita na alínea "c" do subitem 5.1.4.8, para colocar RICMT na lista LRICMT em ordem crescente do valor de ICMT e, em seguida, verifica o valor de $VREG(NTS)$, podendo ocorrer as seguintes situações:

- se $VREG(NTS)=SEQ$, chama a função auxiliar "Atualização de VIPPEV e VTPEV"; e
- se $VREG(NTS)=PAR$, verifica a situação de $VLFE(NTS)$, surgindo as seguintes possibilidades:
 - se FE está vazia, significa que o processo transmissor P_T continua escalado. Nesse caso, chama a função auxiliar "Atualização de VIPPEV e VTPEV"; e
 - se FE não está vazia, as seguintes ações são tomadas:
 - desescala o processo transmissor T_T , inserindo-o no final de FE;
 - iguala $VNPE(NTS)$ ao topo de $VLFE(NTS)$;
 - retira de $VLFE(NTS)$ o topo da fila;
 - faz $VCQUAN(NTS)=0$; e
 - chama a função auxiliar "Atualização de VIPPEV e VTPEV".

Em seguida, ela avança o ponteiro $VPEVA(NPE)$ para o próximo evento de $VLEV(NPE)$ e copia para $V\tau_{PEV,PAR}(NPE)$ o campo $TEVPAR$ da posição apontada por $VPEVA(NPE)$ em $VLEV(NPE)$.

Por fim, ela retorna à função que a chamou (Coordenação da simulação dinâmica da rede de Transputers).

c) Parâmetros de saída

Nenhum.

5.1.4.4 Manipulação da Reativação de Processos (reativa_ processo)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NPR.

b) Escopo da função

Determina $NTS=VALOCA(NPR)$ e verifica $VREG(NTS)$, que pode ter os dois seguintes valores:

- se $VREG(NTS)=SEQ$, a função auxiliar "Atualização de VIPPEV e VTPEV" é chamada; e
- se $VREG(NTS)=PAR$, a função verifica se há algum processo ativo no Transputer NTS , podendo ocorrer os casos a seguir:
 - se não há processo ativo ($VNPE(NTS)=-1$), faz $VNPE(NTS)=NPR$, escala o processo recém-reativado e promove a atualização de $VIPPEV(NTS)$ e $VTPEV(NTS)$. Para tanto, faz $VCQUAN(NTS)=0$ e chama a função auxiliar "Atualização de VIPPEV e VTPEV"; e
 - se há processos ativos ($VNPE(NTS) \geq 0$), a função coloca o processo NPR no final de $VLFE(NTS)$.

Por fim, retorna à função que a chamou (Coordenação da simulação dinâmica da rede de Transputers).

c) Parâmetros de saída

Nenhum.

5.1.4.5 Manipulação de Fim de Processamento de uma Tarefa (finaliza_processamento)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NTS.

b) Escopo da função

Verifica VREG(NTS), o qual pode assumir os seguintes valores:

- se VREG(NTS)=SEQ, significa que terminou o processamento do SEQ. Esta função deve então verificar se a lista dos regimes de operação de VLLESC(NTS) foi toda percorrida, surgindo as seguintes possibilidades:
 - se a lista dos regimes foi toda percorrida, significa que acabou o processamento no Transputer NTS. A função deve decrementar CTA, fazer VTA(NTS)=falso e comparar o valor de WESC com TRP+VSARD(NTS), ocorrendo os seguintes casos:
 - se $WESC < TRP + VSARD(NTS)$, faz $WESC = TRP + VSARD(NTS)$; e
 - se $WESC \geq TRP + VSARD(NTS)$, não faz nada.
 - se a lista dos regimes não foi toda percorrida, significa que acabou o processamento no SEQ, mas ainda há outros regimes a serem escalados. Esta função chama então a

função "Controle da ordem de chamada e do regime de operação dos processos".

- se $VREG(NTS)=PAR$, esta função decrementa $VCPAPAR(NTS)$ e verifica seu valor, podendo ocorrer as seguintes situações:
 - se $VCPAPAR(NTS)=0$ implica que acabou processamento do PAR, devendo se verificar então a lista dos regimes de operação em $VLLESC(NTS)$, surgindo as seguintes opções:
 - se a lista dos regimes foi toda percorrida, significa que acabou o processamento no Transputer NTS. Esta função decrementa então CTA, faz $VTA(NTS)=falso$ e compara o valor de WESC com $TRP+VSARD(NTS)$, ocorrendo as seguintes possibilidades:
 - se $WESC < TRP+VSARD(NTS)$, faz $WESC=TRP+VSARD(NTS)$; e
 - se $WESC \geq TRP+VSARD(NTS)$, não faz nada;
 - se a lista dos regimes não foi toda percorrida, significa que acabou o processamento no PAR mas ainda há outros regimes a serem escalados. Esta função chama então a função "Controle da Ordem de Chamada e do Regime de Operação dos Processos".
 - se $VCPAPAR(NTS) > 0$, ela escala o processo no topo de FE, chamando a função auxiliar "Escalação de processo".

Por fim, retorna-se à função que a chamou (Coordenação da simulação dinâmica da rede de Transputers).

c) Parâmetros de saída

Se a lista dos regimes de operação não foi toda percorrida, coloca NTS como parâmetro de saída. Nos demais casos não há parâmetros de saída.

5.1.4.6 Manipulação de Temporização (temporiza)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NTS e NPE.

b) Escopo da função

Atribui a τ_{ATR} o valor contido na posição apontada por VPEVA(NPE) em VLEV(NPE). Avança então o ponteiro VPEVA(NPE) para o próximo evento de VLEV(NPE) e copia em $V\tau_{PEV,PAR}(NPE)$ o campo TEVPAR da posição apontada por VPEVA(NPE) em VLEV(NPE).

Esta função então verifica VREG(NTS), que pode assumir os seguintes valores:

- se VREG(NTS)=SEQ, ela calcula $VIPPEV(NTS)=TRP+\tau_{ATR}+\tau_{EV,SEQ}$, onde $\tau_{EV,SEQ}$ é retirado da posição apontada por VPEVA(NPE) em VLEV(NPE) e transfere para VTPEV(NTS) o tipo do próximo evento retirado do campo TIPO da posição apontada por VPEVA(NPE) em VLEV(NPE); e
- se VREG(NTS)=PAR, esta função monta o registro RIRFE fazendo $IRFE=TRP+\tau_{ATR}$ e $PTEMP=NPE$. Ela então chama a função auxiliar "Inserção de registro em lista em ordem crescente" para colocar o registro RIRFE na lista LRIRFE em ordem crescente do valor de IRFE e escala o processo no topo da FE, chamando a função auxiliar "Escalação de processo".

Por fim, retorna-se à função que a chamou (Coordenação da simulação dinâmica da rede de Transputers).

c) Parâmetros de saída

Nenhum.

5.1.4.7 Manipulação de Término de Quantum de Tempo (manipula_quantum)

Apresentam-se, a seguir, as características básicas da função.

a) Parâmetros de entrada

NTS.

b) Escopo da função

Esta função verifica o valor de VCQUAN(NTS), que pode assumir os seguintes valores:

- se VCQUAN(NTS)=0, significa que os quanta de tempo disponíveis à tarefa escalada no Transputer identificado por NTS ainda não terminaram. Ela faz VCQUAN(NTS)=1; e
- se VCQUAN(NTS)=1, significa que acabou tempo disponível ao processo escalado no Transputer identificado por NTS. Esta função verifica então a situação de VLFE(NTS), podendo ocorrer as seguintes possibilidades:
 - se FE está vazia, não faz nada, pois o processo continua escalado; e
 - se FE não está vazia, executa as seguintes operações:
 - desescala o processo NPE=VNPE(NTS), colocando-o no final de VLFE(NTS);
 - desconta de $\tau_{EV,PAR}$ o tempo já consumido por NPE ($\tau_{EV,PAR}=VIPPEV(NTS)-TRP$) e o insere em $V\tau_{PEV,PAR}(NPE)$;

- faz $VNPE(NTS)$ =identificação do processo no topo de $VLFE(NTS)$ e o escala, eliminando-o de FE;
- faz $VCQUAN(NTS)=0$; e
- chama a função auxiliar "Atualização de VIPPEV e VTPEV".

Retorna à função que a chamou (Controle de ITQT).

c) Parâmetros de saída

Nenhum.

5.1.4.8 Funções Auxiliares do MSEQ/PAR

Os procedimentos apresentados aqui são utilizados por diversas funções do MSEQ/PAR, justificando-se então a criação das funções auxiliares mostradas a seguir.

a) Atualização de VIPPEV e VTPEV (atualiza_vippev_ vtpev)

Esta função verifica $VREG(NTS)$, que pode assumir os seguintes valores:

- se $VREG(NTS)=SEQ$, calcula $VIPPEV(NTS)=TRP+\tau_{EV,SEQ}$, com $\tau_{EV,SEQ}$ retirado da posição apontada por $VPEVA(NPE)$ em $VLEV(NPE)$; e
- se $VREG(NTS)=PAR$, calcula $VIPPEV(NTS)=TRP+V\tau_{PEV,PAR}(NPE)$.

Transfere então para $VTPEV(NTS)$ o tipo do próximo evento retirado do campo TIPO da posição apontada por $VPEVA(NPE)$ em $VLEV(NPE)$.

b) Escalação de processo (escala_processo)

Esta função verifica a situação de VLFE(NTS), podendo ocorrer as seguintes alternativas:

- se FE está vazia, significa que o Transputer NTS ficará ocioso, aguardando reativação de processo. Realiza então as seguintes operações:
 - $VIPPEV(NTS) = \text{maior valor representável no computador}$; e
 - $VNPE(NTS) = -1$.
- se FE não está vazia, executa os seguintes passos:
 - iguala $VNPE(NTS)$ ao topo de $VLFE(NTS)$;
 - retira de $VLFE(NTS)$ o topo da fila;
 - faz $VCQUAN(NTS) = 0$; e
 - chama a função "Atualização de VIPPEV e VTPEV".

c) Inserção de registro em lista em ordem crescente (insere_ordem_crescente)

Esta função percorre a lista LRIRFE ou LRICMT, dependendo do caso, e insere, respectivamente, o registro RIRFE ou RICMT na mesma, em ordem crescente do valor de IRFE ou ICMT.

5.1.5 Diagramas de Fluxo de Dados

Entende-se aqui como fluxo de dados dentro do Programa Escalador, as informações que são transferidas através de parâmetros. Os demais dados necessários às funções estão disponíveis na base de dados do Programa Escalador.

Este item apresenta os fluxos de dados do Programa Escalador, destacando os seguintes pontos:

- dados de entrada/saída trocados com o usuário, mostrado na figura 5.2;
- troca de dados entre as partes do Programa Escalador, apresentada na figura 5.3; e
- troca de dados entre funções dentro das partes do Programa Escalador. Visto que somente há passagem de parâmetros dentre as funções constituintes do SIMREDE, a figura 5.4 mostra a troca de dados dentro dessa parte do Programa Escalador.

Fig. 5.2 Fluxo de dados com o usuário

Fig. 5.3 Fluxo de dados entre as partes do Programa Escalador

Fig. 5.4 Fluxo de dados do SIMREDE

5.1.6 Definição da Base de Dados

Definem-se, a seguir, as variáveis globais e restritas da base de dados declaradas em cada uma das três partes do Programa Escalador. Entende-se aqui como variável global, aquela que é disponível a qualquer função em qualquer parte do Programa Escalador e, variável restrita, aquela que é disponível apenas às funções contidas em uma das partes do Programa Escalador.

5.1.6.1 Variáveis Globais do Programa Escalador

As variáveis globais do Programa Escalador estão relacionadas na tabela 5.3.

Tipo da variável	Relação das variáveis
Número inteiro	n, m, WESC e WCR
Matriz de variáveis booleanas	MPREC
Matriz de variáveis inteiras	MNNI
Vetor de variáveis inteiras	$V_{\tau PAR}$ e $V_{\tau SEQ}$
Vetor de listas ligadas	VLEV e VLLESC

Tab. 5.3 Variáveis globais do Programa Escalador

5.1.6.2 Variáveis Utilizadas pelo Gerenciador do Programa Escalador

As variáveis restritas ao Gerenciador do Programa Escalador são apresentadas na tabela 5.4.

Tipo da variável	Relação das variáveis
Número inteiro	WESCOBT
Vetor de listas ligadas	VLLESCOB

Tab. 5.4 Variáveis restritas ao Gerenciador do Programa Escalador

Além das variáveis apresentadas na tabela 5.4, as variáveis globais relacionadas na tabela 5.5 são também usadas nesta parte do Programa Escalador.

Tipo da variável	Relação das variáveis
Número inteiro	n, WESC e WCR
Matriz de variáveis booleanas	MPREC
Vetor de variáveis inteiras	$V_{\tau PAR}$ e $V_{\tau SEQ}$
Vetor de listas ligadas	VLLESC

Tab. 5.5 Variáveis globais utilizadas pelo Gerenciador do Programa Escalador

5.1.6.3 Variáveis Utilizadas pelo ESCEXT

As variáveis restritas ao ESCEXT são listadas na tabela 5.6.

Tipo da variável	Relação das variáveis
Vetor de variáveis inteiras	VCP, VCPn, VN e VPRIOR

Tab. 5.6 Variáveis restritas ao ESCEXT

As variáveis globais utilizadas pelo ESCEXT são listadas na tabela 5.7.

Tipo da variável	Relação das variáveis
Número inteiro	n, m e WCR
Matriz de variáveis booleanas	MPREC
Vetor de variáveis inteiras	$V_{\tau PAR}$ e $V_{\tau SEQ}$
Vetor de listas ligadas	VLLESC

Tab. 5.7 Variáveis globais utilizadas pelo ESCEXT

5.1.6.4 Variáveis Utilizadas pelo SIMREDE

As variáveis restritas ao SIMREDE são mostradas na tabela 5.8.

Tipo da variável	Relação das variáveis
Número inteiro	ITQT, TRP e CTA
Matriz de variáveis booleanas	MPREC
Matriz de variáveis inteiras	MNNI
Vetor de variáveis booleanas	VT A e VREG
Vetor de variáveis inteiras	VIPPEV, VNPE, VALOCA, VTPEV, VSARD $V_{\tau PEV, PAR}$, VCPAPAR e VCQUAN
Vetor de registros	VRPACM
Vetor de ponteiros	VPEVA e VPLREGESC
Lista ligada de registros	LRIRFE e LRICMT
Vetor de listas ligadas	VLRMTAS e VLFE

Tab. 5.8 Variáveis restritas ao SIMREDE

As variáveis globais usadas pelo SIMREDE são listadas na tabela 5.9.

Tipo da variável	Relação das variáveis
Número inteiro	n, m e WESC
Matriz de variáveis inteiras	MNNI
Vetor de listas ligadas	VLEV e VLLESC

Tab. 5.9 Variáveis globais utilizadas pelo SIMREDE

5.2 Projeto Detalhado do Programa Escalador

Esta seção procura detalhar as funções apresentadas na seção anterior, descendo a um nível de detalhes que permita a codificação do programa que implementa o Programa Escalador. Apresentam-se os diagramas de Nassi-Schneiderman dos módulos correspondentes às funções descritas na seção 5.1, empregando-se a simbologia adotada em (ARAKAKI et al., 1990).

Utiliza-se a seguinte convenção nos diagramas apresentados:

- * nome_do_módulo *

Corresponde a módulos de manipulação em estruturas de dados ou de algoritmos de ordenação do tipo Quicksort, desenvolvidos como apoio para o Programa Escalador; e

- ** nome_do_módulo **

Corresponde aos módulos apresentados na seção 5.1 e cujos diagramas de blocos se encontram presentes nesta seção.

O diagrama de blocos do Programa Escalador é mostrado na figura 5.5.

Fig. 5.5 Diagrama de blocos do Programa Escalador

5.2.1 Módulos do Gerenciador do Programa Escalador (programa_escalador)

A tabela 5.10 apresenta a relação entre os módulos e submódulos do Programa Escalador e as figuras que descrevem os seus diagramas de blocos.

Módulo	Submódulo	Figura
formata_entrada	---	5.6
--	pega_número	5.7
formata_saída	---	5.8

Tab. 5.10 Relação dos módulos componentes do Gerenciador do Programa Escalador

Fig. 5.6 Diagrama de blocos do módulo formata_entrada

OBS.: As buscas de erros nos arquivos de entrada não estão incluídas neste diagrama de blocos.

Fig. 5.6 Diagrama de blocos do módulo formata_entrada (cont.)

Lê um número do arquivo ARQ, ignorando caracteres inválidos
Retorna número lido

Fig. 5.7 Diagrama de blocos do módulo pega_número(ARQ)

Fig. 5.8 Diagrama de blocos do módulo formata_saída

5.2.2 Módulos do ESCEXT

Ao se descrever as funções do ESCEXT no item 5.1.2, mostrou-se quais são os passos que definem cada algoritmo heurístico, sem, no entanto, desmembrar esses algoritmos em procedimentos. Neste item, no entanto, os diagramas de blocos que são apresentados descrevem os módulos que compõem cada algoritmo, conforme indicado na tabela 5.11.

Módulo	Submódulo	Sub-submódulo	Figura
inicia_escalador	---	---	5.9
--	inicia_VCP	---	5.10
--	---	calcula_VCP	5.11
--	inicia_VCPn	---	5.12
--	inicia_VN	---	5.13
--	---	calcula_VN	5.14
escala	---	---	5.15
--	ordena	---	5.16
--	---	cp_nsi_cmp	5.17
--	---	n_tseq_cp_cmp	5.18
--	---	n_cp_cmp	5.19
--	---	n_cpn_nsi_cmp	5.20
--	---	lpt_cmp	5.21
--	distribui_SEQ	---	5.22
--	---	seleciona_transputer	5.23
--	---	aloca_SEQ	5.24
--	distribui_compactado	---	5.25
--	---	aloca_PAR	5.26
--	distribui_PAR	---	5.27
--	distribui_MTP	---	5.28

Tab. 5.11 Relação dos módulos componentes do ESCEXT

Fig. 5.9 Diagrama de blocos do módulo inicia_escalador

Fig. 5.10 Diagrama de blocos do módulo inicia_VCP

Fig. 5.11 Diagrama de blocos do módulo calcula_VCP(i)

Fig. 5.12 Diagrama de blocos do módulo inicia_VCPn

Fig. 5.13 Diagrama de blocos do módulo inicia_VN

Fig. 5.14 Diagrama de blocos do módulo calcula_VN(i)

Fig. 5.15 Diagrama de blocos do módulo escala(ALGOR)

* Usa algoritmo Quicksort para ordenar VPRIOR usando
função de comparação fcomp determinada por ALGOR *

Fig. 5.16 Diagrama de blocos do módulo ordena(VPRIOR,ALGOR)

Fig. 5.17 Diagrama de blocos da função de comparação $cp_nsi_cmp(i,j)$

Fig. 5.18 Diagrama de blocos da função de comparação $n_tseq_cp_cmp(i,j)$

Fig. 5.19 Diagrama de blocos da função de comparação
n_cp_cmp(i,j)

Fig. 5.20 Diagrama de blocos da função de comparação
n_cpn_nsi_cmp(i,j)

Fig. 5.21 Diagrama de blocos da função de comparação `lpt_cmp(i,j)`

Fig. 5.22 Diagrama de blocos do módulo `distribui_SEQ(VPRIOR)`

Fig. 5.23 Diagrama de blocos do módulo `seleciona_transputer(VTAPT)`

Fig. 5.24 Diagrama de blocos do módulo aloca_SEQ(i,t,VLLESC)

Fig. 5.25 Diagrama de blocos do módulo distribui_compactado (VPRIOR)

Fig. 5.26 Diagrama de blocos do módulo aloca_PAR(i,t,VLLESC)

Fig. 5.27 Diagrama de blocos do módulo distribui_PAR(VPRIOR)

Fig. 5.28 Diagrama de blocos do módulo distribui_MTP(VPRIOR)

5.2.3 Módulos do GEREDE

A relação dos módulos que compõem o GEREDE é apresentada na tabela 5.12.

Módulo	Figura
coordena_simulação	5.29
inicia_simulação	5.30
controla_IRFE	5.31
controla_ITQT	5.32
controla_ICMT	5.33

Tab. 5.12 Relação dos módulos que compõem o GEREDE

Fig. 5.29 Diagrama de blocos do módulo coordena_simulação

Fig. 5.30 Diagrama de blocos do módulo inicia_simulação

Fig. 5.31 Diagrama de blocos do módulo controla_IRFE

Fig. 5.32 Diagrama de blocos do módulo controla_ITQT

Fig. 5.33 Diagrama de blocos do módulo controla_ICMT

5.2.4 Módulos do MSEQ/PAR

A tabela 5.13 contém a relação dos módulos que constituem o MSEQ/PAR.

Módulo	Figura
controla_chamada	5.34
recebe_mensagem	5.35
transmite_mensagem	5.36
reativa_processo	5.37
finaliza_processamento	5.38
temporiza	5.39
manipula_quantum	5.40

Tab. 5.13 Relação dos módulos que constituem o MSEQ/PAR

Os módulos citados na tabela 5.11 chamam três submódulos auxiliares pertencentes ao MSEQ/PAR e que estão relacionados na tabela 5.14.

Submódulo auxiliar	Figura
atualiza_VIPPEV/VTPEV	5.41
escala_processo	5.42
insere_ordem_crescente	5.43

Tab. 5.14 Relação dos submódulos auxiliares do MSEQ/PAR

Verifica o número de processos contidos no regime RGM apontado por VPLREGESC(NTS) em VLLESC(NTS). Se for 1 significa SEQ, se for maior que 1 significa PAR	
Qual o regime RGM?	
SEQ	PAR
VCPAPAR(NTS)=0	VCPAPAR(NTS)=número de processos em RGM
NPE=identificação do processo em RGM	VLFE(NTS)=lista de processos em RGM
	VREG(NTS)=PAR
VREG(NTS)=SEQ	* Remove fim de VLFE(NTS) e coloca em NPE *
	VCQUAN(NTS)=0
VNPE(NTS)=NPE	
Avança ponteiro VPLREGESC(NTS) para próximo regime (RGM) de VLLESC(NTS)	
** atualiza_VIPPEV/VTPEV(NTS,NPE) **	

Fig. 5.34 Diagrama de blocos do módulo controla_chamada(NTS)

Fig. 5.35 Diagrama de blocos do módulo recebe_mensagem (NTS,NPE)

Fig. 5.36 Diagrama de blocos do módulo transmite_mensagem
(NTS,NPE)

Fig. 5.37 Diagrama de blocos do módulo reativa_processo(NPR)

Fig. 5.38 Diagrama de blocos do módulo finaliza_processamento(NTS)

Fig. 5.39 Diagrama de blocos do módulo temporiza(NTS,NPE)

Fig. 5.40 Diagrama de blocos do módulo manipula_quantum(NTS)

Qual é o regime em VREG(NTS)?	
SEQ	PAR
Coloca em TEVSEQ seu valor retirado da posição apontada por VPEVA(NPE) em VLEV(NPE)	VIPPEV(NTS)=TRP+VTPEVPAR(NPE)
VIPPEV(NTS)=TRP+TEVSEQ	
Coloca em VTPEV(NTS) o valor do campo TIPO retirado da posição apontada por VPEVA(NPE) em VLEV(NPE)	

Fig. 5.41 Diagrama de blocos do módulo atualiza_VIPPEV/VTPEV(NTS,NPE)

* VLFE(NTS) está vazia? *	
S	N
VIPPEV(NTS)=maior valor representável no computador	* Remove topo de VLFE(NTS) e coloca em NPE *
VNPE(NTS)=-1	VNPE(NTS)=NPE
	VCQUAN(NTS)=0
	** atualiza_VIPPEV/VTPEV(NTS,NPE) **

Fig. 5.42 Diagrama de blocos do módulo escala_processo(NTS)

Este módulo é chamado dos módulos transmite_mensagem e temporiza. No caso do módulo transmite_mensagem, o registro RGT a ser inserido é o RICMT através da ordenação de ICMT. No caso de temporiza, o registro a ser inserido é o RIRFE através da ordenação de IRFE. Portanto, a função insere_ordem_crescente terá duas formas, uma para cada função que a chama, sendo que a diferença entre ambas está nos campos RGT.CMP e REGISTRO.CMP, que será ICMT no caso de ser chamada de transmite_mensagem e IRFE no caso de ser chamada de temporiza.

Fig. 5.43 Diagrama de blocos do módulo insere_ordem_crescente
(RGT,LST)

6 - IMPLEMENTAÇÃO E TESTES DO PROGRAMA ESCALADOR

Este capítulo é composto das três seguintes seções:

- a primeira contém comentários a respeito da implementação do Programa Escalador;
- a segunda ilustra a forma de entrada e saída dos dados no Programa Escalador; e
- a terceira apresenta os testes de aceitação a que se submeteu o Programa Escalador.

6.1 Comentários sobre a Implementação do Programa Escalador

O programa foi desenvolvido no Borland C++, versão 2.0, podendo ser usado em qualquer computador que disponha de um compilador dessa linguagem. Sua versão executável ocupa cerca de 121 kbytes de memória.

Apresentam-se, a seguir, comentários a respeito da implementação das estruturas de dados do Programa Escalador. Utilizando-se os recursos do C++, as listas ligadas, vetores e matrizes foram implementadas na forma de classes.

A classe Lista permite que se manipule listas ligadas como se fossem vetores, isto é, a posição na lista passa a ser tratada como o índice de um vetor. Dessa forma, a consulta e a remoção de elementos da lista é facilitada ao usuário. A analogia com a manipulação de vetores só não é completa, porque para inserir na lista é necessário chamar uma função de inserção em lista, não sendo possível a atribuição direta feita em vetores (`vetor[i]=x`).

As classes Lista, Vetor e Matriz permitem prover alocação dinâmica em memória, possibilitando a economia da mesma, pela alocação dos espaços como função dos valores de m e n introduzidos pelo usuário. No caso de listas, seria preciso alocar e posteriormente desalocar o espaço necessário em memória. No caso de vetores e matrizes, com a utilização de alocação estática de memória, seria necessário dimensioná-los em função dos valores máximos que m e n podem atingir, ocasionando uma reserva de espaço que nem sempre seria totalmente utilizada. A alocação dinâmica provê comodidade ao programador, que não precisa se preocupar em alocar/desalocar o espaço necessário em memória para listas e em não desalocar o espaço requerido para vetores e matrizes. Isto se torna viável graças ao emprego dos construtores e destrutores providos pelo C++.

Outra possibilidade gerada pelo uso de classes foi a aplicação de tipos genéricos (parametrizados) de dados, através do emprego de macros, conforme descrito em (STROUSTRUP, 1987).

6.2 Entrada e Saída dos Dados no Programa Escalador

A seqüência de passos que o usuário do Programa Escalador deve seguir para poder utilizá-lo é a seguinte:

- geração do arquivo de entrada de dados dos processos;
- geração do arquivo de entrada de dados dos Transputers;
- acionamento do Programa Escalador; e
- leitura dos resultados apresentados.

A explanação de como inserir os dados de entrada e como interpretar os dados de saída é dada nos itens que se seguem. Um

exemplo completo de utilização do Programa Escalador é mostrado no item 6.3.4.

6.2.1 Geração do Arquivo de Entrada de Dados dos Processos

O usuário deve gerar um arquivo-texto com os dados dos processos, podendo-se utilizar qualquer editor de texto que gere arquivos não formatados e os armazene no formato ASCII.

Com relação à digitação dos dados de entrada nos arquivos, enfatizam-se os seguintes pontos:

- o usuário pode inserir nos arquivos de entrada qualquer tipo de comentário que desejar, exceto os dígitos 0 a 9, visto que a função "formata_entrada" busca como dado de entrada apenas números. Entretanto, se for necessário introduzir informações numéricas como comentários, deve-se colocá-las entre aspas (");
- a separação entre dois dados consecutivos pode ser feita através de espaços em branco, tabulação, vírgula, ponto e vírgula, barra ou qualquer outro caractere não-numérico que se desejar, exceto aspas; e
- pode-se pular quantas linhas se desejar entre linhas que contêm dados.

6.2.1.1 Inserção dos Dados de Entrada dos Processos

Os dados relativos aos processos devem ser introduzidos no arquivo de entrada de dados dos processos na ordem em que são apresentados a seguir:

a) Número de processos a serem escalados (n)

Este dado é colocado sozinho em uma linha do arquivo. O Programa Escalador está previsto para operar com $n \geq m$, pois se $n \leq m$, basta colocar os n processos em SEQ em cada um dos m Transputers disponíveis e já se terá a melhor escalação possível.

b) Relações de precedência

Correspondem à definição do relacionamento pai/filho entre as tarefas. Os dados devem ser introduzidos colocando-se, em cada linha, a identificação dos filhos para o pai correspondente àquela linha, seguindo a ordem crescente de identificação dos pais, conforme exemplo da tabela 6.1, que se refere ao grafo de precedência mostrado na figura 4.7.

Pai	Filhos
"P0"	P1 P2 P3
"P1"	P4 P5
"P2"	P5
"P3"	P5
"P4"	P6
"P5"	P6
"P6"	P7 P8
"P7"	P9
"P8"	P9

Tab. 6.1 Relações de precedência das tarefas da figura 4.7

Este é o único lugar na entrada de dados onde o usuário utiliza os processos fictícios P_0 e P_{n+1} , criados por conveniência do Programa Escalador. Nos demais pontos, a entrada de dados se refere apenas aos processos reais, os quais devem ser numerados consecutivamente de 1 a n .

c) Tempo de processamento das tarefas em SEQ e PAR (τ_{SEQ}/τ_{PAR})

Cada linha do arquivo de dados de entrada dos processos deve receber um par de valores $\tau_{SEQ}(P_i)$ e $\tau_{PAR}(P_i)$, em ordem crescente do número de identificação dos processos, correspondente ao tempo de processamento em SEQ e PAR de cada processo a ser escalado. Um exemplo de inserção dos dados é mostrado na tabela 6.2.

Processo	τ_{SEQ}	τ_{PAR}
"P1"	$\tau_{SEQ}(P_1)$	$\tau_{PAR}(P_1)$
"P2"	$\tau_{SEQ}(P_2)$	$\tau_{PAR}(P_2)$
⋮	⋮	⋮
"Pn"	$\tau_{SEQ}(P_n)$	$\tau_{PAR}(P_n)$

Tab. 6.2 Exemplo de entrada de dados dos tempos de processamento das tarefas

d) Dados dos eventos dos processos

Cada linha do arquivo de entrada dos dados dos processos deve receber os dados relativos a um determinado evento de cada processo, seguindo a ordem crescente do número de identificação dos processos, conforme indicado no exemplo a

seguir. Enfatiza-se que as molduras das tabelas ou outros dados não-numéricos aqui apresentados são utilizados apenas para facilitar a leitura das tabelas, podendo ser descartados.

- Dados dos eventos do processo P1:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
10	12	2	--	P2	1	10	60	--
100	115	3	--	--	-	--	--	3
5	6	4	--	--	-	--	--	--

- Dados dos eventos do processo P2:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
22	25	1	P1	--	1	--	--	--
15	18	2	--	P3	2	10	60	--
0	0	4	--	--	-	--	--	--

- Dados dos eventos do processo Pn:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
122	133	1	P3	--	1	--	--	--
100	115	4	--	--	-	--	--	--

O significado e a forma de coletar e calcular os dados que envolvem tempo, isto é, $\tau_{EV,SEQ}$, $\tau_{EV,PAR}$, τ_{DMINT} , τ_{DMEXT} e τ_{ATR} são explicados no sub-item 6.2.1.2. O significado dos demais dados dos eventos, independentes do tempo, é explicado a seguir.

Há quatro tipos de evento, sendo que a forma de representação e a relação de dados que caracterizam cada tipo são diferentes, conforme mostrado na tabela 6.3.

Tipo do evento	Representação	Dados do evento
RECEPÇÃO	1	P_T e NSM
TRANSMISSÃO	2	P_R , NSM, τ_{DMINT} e τ_{DMEXT}
TEMPORIZAÇÃO	3	τ_{ATR}
FIM_PROCESSAMENTO	4	----

Tab. 6.3 Forma de representação e dados descritores de cada tipo de evento

P_T e P_R correspondem, respectivamente, ao número de identificação do processo transmissor e receptor da mensagem e NSM é um número seqüencial crescente que identifica, junto ao processo receptor, qual a ordem de chegada das mensagens recebidas. Assim, se um dado processo recebe duas mensagens, a primeira terá NSM=1 e a segunda NSM=2.

A ordem de inserção dos eventos deve ser a mesma em que eles ocorrem no processo.

6.2.1.2 Coleta dos Dados de Entrada dos Processos que Envolvem Tempo

Os dados que envolvem tempo requerem que o usuário execute certas operações com os processos nos Transputers.

Visto que se supõe que o Processo Roteador/Distribuidor está instalado em todos os Transputers da rede, operando em modo prioritário, há necessidade de se fornecer os intervalos de tempo entre os eventos de cada processo medidos dentro do contexto do Processo Roteador/Distribuidor para os regimes SEQ e PAR.

Ao se coletar os tempos de um processo não-prioritário, o temporizador disponível no Transputer é o de baixa prioridade, que fornece tempos com precisão de $64 \mu s$. O ideal seria coletarem-se os tempos com o processo em teste colocado como de alta prioridade em uma construção PRI PAR, resultando então tempos em microsegundos. No entanto, tal procedimento conduz a intervalos de tempo menores que aqueles obtidos quando se coloca esse mesmo processo como de baixa prioridade conjuntamente com o Processo Roteador/Distribuidor.

Isto se justifica pela alocação em memória dos processos ser diferente, pois ao alocar em memória o Processo Roteador/Distribuidor e o processo em teste, o compilador Occam utiliza a memória embutida de forma diferente que utilizaria se tivesse que alocar apenas o processo em teste, tornando o processamento das tarefas mais lento. Os dados assim obtidos conduziriam a um comprimento simulado da escalação menor do comprimento real. Deve-se ressaltar que essa variação não ocorre nos tempos de comunicação internos ou externos, que dependem apenas do tempo médio de acesso da memória externa do Transputer (para comunicação interna) ou da taxa de transmissão de dados ajustada nas vias de comunicação do Transputer (para comunicação externa).

Para a coleta dos tempos, recomenda-se o seguinte procedimento:

- a) colocar instruções de tomada de tempo no início da tarefa, antes e depois de cada evento desescalador (comunicação ou temporização) e no fim da mesma;

b) colocar a tarefa como não-prioritária juntamente com o Processo Roteador/Distribuidor e processá-la em SEQ e em PAR, obedecendo ao seguinte arranjo, visando a obtenção dos instantes de ocorrência dos eventos de cada processo:

- para regime SEQ:

```
PRI PAR
  Processo Roteador/Distribuidor
  Declaração de variáveis
  SEQ
    mensagens transmitidas para tarefa em teste
    tarefa em teste
    mensagens recebidas da tarefa em teste
```

- para regime PAR:

```
PRI PAR
  Processo Roteador/Distribuidor
  PAR
    Declaração de variáveis
    SEQ
      tarefa em teste
    Declaração de variáveis
    SEQ
      tarefa comunicante
```

Convém observar que a tarefa comunicante precisa apenas receber e transmitir as mensagens geradas pelo processo em teste.

A título de exemplo, mostra-se nas figuras 6.1 e 6.2 um processo em teste codificado em Occam-2, configurado, respectivamente, para coleta dos tempos nos regimes SEQ e PAR.

```

[10]INT msg:
TIMER rel:
INT atraso, ti, t1, t2, t3, t4, t5, t6, tf:
SEQ
-- Mensagens transmitidas para o processo em teste
-- Assume-se que o Transputer utilizado é o de número 1
msg[0]:=0100201 -- (msg p/ Transputer 1, tarefa 2, NSM=1)
interno.ent ! msg

-- Processo em teste (processo 2)
rel ? ti
.
.
rel ? t1
msg[0]:=0100403 -- (msg p/ Transputer 1, tarefa 4, NSM=3)
interno.ent ! msg
rel ? t2
.
.
rel ? t3
canal.interno[201] ? msg
rel ? t4
.
.
atraso:=10
rel ? t5
rel ? AFTER t5 PLUS atraso
rel ? t6
.
.
rel ? tf

-- Mensagens recebidas do processo em teste
canal.interno[403] ? msg

```

Fig. 6.1 Exemplo de arranjo para a coleta de tempos de processo em SEQ

```

PAR
-- Processo em teste (processo 2)
[10]INT msg:
TIMER rel:
INT atraso, ti, t1, t2, t3, t4, t5, t6, tf:
SEQ
    rel ? ti
    .
    .
    rel ? t1
    msg[0]:=0100403 -- (msg p/ Transp. 1, tarefa 4, NSM=3)
    interno.ent ! msg
    rel ? t2
    .
    .
    rel ? t3
    canal.interno[201] ? msg
    rel ? t4
    .
    .
    atraso:=10
    rel ? t5
    rel ? AFTER t5 PLUS atraso
    rel ? t6
    .
    .
    rel ? tf

-- Processo comunicante
[10]INT msg:
SEQ
    canal.interno[403] ? msg
    msg[0]:=0100201 -- (msg p/ Transp. 1, tarefa 2, NSM=1)
    interno.ent ! msg

```

Fig. 6.2 Exemplo de arranjo para a coleta de tempos de processo em PAR

- c) para a obtenção de τ_{DMEXT} , coloca-se o processo em teste em SEQ como prioritário em uma construção PRI PAR, sem a presença do Processo Roteador, e coloca-se o processo comunicante em um Transputer distinto mas diretamente interligado ao Transputer que contém a tarefa em teste. Esta situação gera comunicações externas, podendo-se então medir o valor de τ_{DMEXT} em ticks

prioritários, relativo à duração da transferência de mensagens entre Transputers diretamente interligados.

É necessário utilizar o processo em teste como prioritário para poder coletar os tempos com o relógio de alta prioridade, visto que os tempos externos de comunicação são, via de regra, da ordem de dezenas de microsegundos. Este é o único ensaio para coleta de dados que exige o emprego de dois Transputers.

O arranjo a ser feito é exemplificado na figura 6.3 através da mesma tarefa utilizada nas figuras 6.1 e 6.2.

Transputer T1:	Transputer T2:
<pre> PRI PAR -- Processo em teste [10]INT msg: TIMER rel: INT atraso,t1,t2,t3,t4,t5,t6: CHAN OF ANY can12, can21: SEQ . . . rel ? t1 can12 ! msg rel ? t2 . . . rel ? t3 can21 ? msg rel ? t4 . . . atraso:=10 rel ? t5 rel ? AFTER t5 PLUS atraso rel ? t6 . . . SKIP </pre>	<pre> -- Processo comunicante [10]INT msg: CHAN OF ANY can12,can21: SEQ can12 ? msg can21 ! msg </pre>

Fig. 6.3 Exemplo de arranjo para a coleta de τ_{DMEXT}

- d) para a obtenção de τ_{DMINT} , o processo em teste e o processo comunicante são colocados em PAR dentro da construção PRI PAR no mesmo Transputer. Esta situação gera comunicações internas, podendo-se então medir o valor de τ_{DMINT} em μs relativo à duração da transferência de mensagens dentro de um Transputer. O arranjo a ser utilizado é exemplificado na figura 6.4 através da mesma tarefa utilizada na figura 6.3.

```

PRI PAR
  PAR
    -- Processo em teste
    Declaração de variáveis
    SEQ
      .
      .
      .
      rel ? t1
      can12 ! msg
      rel ? t2
      .
      .
      .
      rel ? t3
      can21 ? msg
      rel ? t4
      .
      .
      .
      atraso:=10
      rel ? t5
      rel ? AFTER t5 PLUS atraso
      rel ? t6
      .
      .
      .
    -- Processo comunicante
    Declaração de variáveis
    SEQ
      can12 ? msg
      can21 ! msg
    SKIP
  
```

Fig. 6.4 Exemplo de arranjo para a coleta de τ_{DMINT}

Tendo-se coletado os tempos, há necessidade de se calcular os seguintes valores:

a) τ_{SEQ}/τ_{PAR}

No caso das figuras 6.1 e 6.2 eles são calculados por:

$$\tau_{SEQ} = t_f - t_i \text{ (da figura 6.1)}$$

$$\tau_{PAR} = t_f - t_i \text{ (da figura 6.2)}$$

b) $\tau_{EV}(i)$

Corresponde ao intervalo entre dois eventos consecutivos (i e $i-1$). Conforme se depreende do que foi exposto no Anexo, τ_{EV} deve ser calculado para as construções SEQ e PAR, resultando, respectivamente, $\tau_{EV,SEQ}$ e $\tau_{EV,PAR}$. No caso apresentado nas figuras 6.1 e 6.2 há quatro intervalos de tempo a serem medidos, conforme mostrado na figura 6.5.

Da figura 6.1:	Da figura 6.2:
$\tau_{EV,SEQ}(1) = t_1 - t_i$	$\tau_{EV,PAR}(1) = t_1 - t_i$
$\tau_{EV,SEQ}(2) = t_3 - t_2$	$\tau_{EV,PAR}(2) = t_3 - t_2$
$\tau_{EV,SEQ}(3) = t_5 - t_4$	$\tau_{EV,PAR}(3) = t_5 - t_4$
$\tau_{EV,SEQ}(4) = t_f - t_6$	$\tau_{EV,PAR}(4) = t_f - t_6$

Fig. 6.5 Exemplo de cálculo do intervalo de duração de cada evento

c) $\tau_{DMINT}/\tau_{DMEXT}$

Deve-se informar a duração da transferência de cada mensagem a ser trocada com outro processo, supondo comunicação externa (τ_{DMEXT}) e interna (τ_{DMINT}). Para evitar repetições, deve-se

informar os valores de τ_{DMEXT} e τ_{DMINT} apenas junto ao processo transmissor da mensagem.

No caso da figura 6.3, τ_{DMEXT} é calculado da seguinte forma:

$$\tau_{DMEXT}(1^a \text{ comunicação}) = t_2 - t_1$$

$$\tau_{DMEXT}(2^a \text{ comunicação}) = t_4 - t_3$$

τ_{DMINT} é calculado da mesma forma que τ_{DMEXT} , mas tomando os dados da figura 6.4, como mostrado abaixo:

$$\tau_{DMINT}(1^a \text{ comunicação}) = t_2 - t_1$$

$$\tau_{DMINT}(2^a \text{ comunicação}) = t_4 - t_3$$

d) τ_{ATR}

É calculado medindo-se o tempo que uma temporização consome. Este valor é o mesmo tanto para SEQ quanto para PAR, ou seja:

$$\tau_{ATR} = t_6 - t_5$$

Na realidade, este valor não precisaria nem ser medido, pois na própria instrução de atraso já consta o valor do atraso imposto pela temporização.

Nos tempos fornecidos pelo usuário, relativos aos eventos de cada processo, caso haja mais de um intervalo de tempo possível, devido à presença de malhas, "IF" ou outras condições variáveis, pode-se fornecer o tempo médio ou máximo, seguindo a filosofia de tempo médio ou máximo mencionada no sub-item 2.2.1.2, alínea "c".

Os tempos de processamento das tarefas τ_{SEQ}/τ_{PAR} , as durações dos eventos $\tau_{EV,SEQ}/\tau_{EV,PAR}$ e os intervalos de temporizações τ_{ATR} , devem ser fornecidos pelo usuário em ticks não

prioritários, por ser essa a unidade de tempo obtida nas operações de coleta de tempo desses valores. No entanto, o Programa Escalador converte esses intervalos de tempo para microsegundos, pois é a unidade de tempo usada internamente pelo programa. Os únicos valores de tempo que devem ser fornecidos pelo usuário diretamente em microsegundos são as durações das comunicações interna (τ_{DMINT}) e externa (τ_{DMEXT}), devido a seus pequenos valores e também por serem coletados nessa unidade de tempo.

Recomenda-se que a declaração do tipo das variáveis nos processos, seja feita o mais próximo possível de seu lugar de uso, de maneira a reduzir seu escopo à área em que a dita variável é efetivamente usada. Assim, por exemplo, a variável "atraso" utilizada nas figuras 6.1 e 6.2, deve ser definida apenas junto à construção em que ela vai ser empregada.

Deve-se evitar a utilização de variáveis globais, definidas no início do processo. Essa precaução se justifica pela grande influência do local em que uma variável é declarada no tempo de processamento de uma tarefa. Assim, no caso das tarefas utilizadas para testar o SIMREDE, apresentadas na seção 6.3, surgem diferenças enormes no tempo de processamento das tarefas se as variáveis "sum" são declaradas local ou globalmente. Essa diferença de tempo pode chegar a 20 ou 25% do tempo de processamento da tarefa, sendo que no caso de se utilizar a variável como global, o processamento fica mais lento. Isso acontece devido à alocação que o compilador Occam faz das variáveis na memória embutida ou externa do Transputer.

6.2.2 Geração do Arquivo de Entrada de Dados dos Transputers

O usuário deve gerar um arquivo-texto com qualquer nome, empregando um editor de texto que gere arquivos não formatados.

Os seguintes dados devem ser introduzidos no arquivo de entrada dos dados dos Transputers, na ordem em que são apresentados:

a) Número de Transputers (m)

O número de Transputers na rede deve ser inserido sozinho em uma linha do arquivo.

b) Matriz MNNI

Na posição (i,j) e (j,i) da matriz MNNI deve-se colocar o número de nós intermediários entre os Transputers T_i e T_j . Se $T_i=T_j$ ou ambos são diretamente interligados, então $MNNI(i,j)=MNNI(j,i)=0$.

Cada linha do arquivo de entrada deve conter os m dados relativos a cada linha da matriz. A tabela 6.4 mostra um exemplo de inserção de dados da matriz MNNI.

	"T1"	"T2"	...	"Tm"
"T1"	0	0	...	3
"T2"	0	0	...	2
.	.	.		.
.	.	.		.
.	.	.		.
"Tm"	3	2	...	0

Tab. 6.4 Exemplo de inserção dos dados da matriz MNNI

6.2.3 Formato do Arquivo de Saída de Dados

O arquivo de saída de dados apresenta a escalação de cada Transputer no seguinte formato:

- m vezes a seguinte seqüência de dados:
 - identificação do Transputer T_i ;
 - identificação dos processos escalados em T_i em SEQ e PAR; e
- os valores de WESCOBT e WCR em microsegundos.

Um exemplo de um arquivo de saída é mostrado na figura 6.6.

```

Transputer T1:
P1
P4
P7 Pn P9

Transputer T2:
P2 P3 P6
P5

.
.
.

Transputer Tm:
P10
P13

WESCOBT = 53425 microsegundos

WCR = 58650 microsegundos

```

Fig. 6.6 Exemplo de arquivo de saída mostrando a escalação gerada

6.3 Testes de Aceitação do Programa Escalador

Esta seção relaciona os testes efetuados em módulos individuais e nas três partes do Programa Escalador e termina

apresentando os testes aplicados ao Programa Escalador integrado, seguindo o roteiro proposto no item 4.2.7.

6.3.1 Testes de Aceitação dos Módulos do Programa Escalador

Os testes de aceitação a que cada módulo do Programa Escalador foi submetido obedeceram o seguinte roteiro:

- criou-se uma base de dados fictícia, acessível ao módulo em teste, incluindo especificamente os dados de entrada relacionados com o dito módulo;
- chamou-se o módulo, transferindo-lhe seus parâmetros de entrada. Os dados e os parâmetros de entrada foram modificados, de sorte que todos os caminhos dentro do módulo foram percorridos, com todas as condições testadas;
- após o módulo ter sido processado, verificaram-se os seguintes aspectos:
 - se todas as variáveis de saída modificadas pelo módulo em questão foram corretamente atualizadas na base de dados;
 - caso o módulo em teste acionasse outros módulos, se a chamada desses módulos estava sendo efetuada e, caso aplicável, se os parâmetros de saída estavam corretos; e
 - caso o módulo em teste retornasse um valor, a correção desse retorno.

6.3.2 Testes de Aceitação dos Algoritmos Heurísticos de Escalação

Para verificar a correção no funcionamento dos algoritmos de escalação, foram geradas escalações manualmente e através dos

algoritmos do ESCEXT, para alguns grafos de precedência, visando comparar seus resultados. Os grafos de precedência utilizados nesses testes são mostrados nas figuras 2.1, 4.7, 6.7 e 6.8.

Fig. 6.7 Grafo de precedência utilizado para testar os algoritmos heurísticos

Fig. 6.8 Grafo de precedência utilizado para testar os algoritmos heurísticos

Os resultados obtidos nos testes através dos algoritmos do ESCEXT coincidiram com as escalações realizadas manualmente, o que comprovou a correta operação dos mesmos.

6.3.3 Testes de Validação do Modelo Dinâmico da Rede de Transputers

Os algoritmos heurísticos do ESCEXT sofreram um processo de aceitação, enquanto o modelo da rede de Transputers sofreu um

processo de validação. A diferença básica consiste em que, enquanto para a aceitação basta verificar se o produto gerado segue as especificações técnicas fornecidas, para a validação é preciso confrontar os resultados obtidos pelo modelo com resultados coletados experimentalmente. Na aceitação, pressupõe-se que haja uma perfeita conformidade entre o especificado e o produzido, ao passo que na validação opera-se com margens aceitáveis de desvio entre os resultados reais e simulados.

Tendo em vista que o fluxo de dados dentro do SIMREDE é complexo, é recomendável prover uma integração gradual de seus módulos, de modo a testar as estruturas lógicas que se formam à medida em que se interligam os módulos.

6.3.3.1 Avaliação de Desempenho do SIMREDE

Uma vez integrado, o SIMREDE foi avaliado para verificar se cumpria o proposto na alínea "a" do item 4.2.2 em termos de desempenho. Para tanto, gerou-se uma série de escalões com complexidade crescente, em diferentes redes de Transputers, a fim de se comparar os comprimentos real e simulado de cada escalação.

Em todos os testes realizados, mostrados a seguir, os processos são compostos de eventos desescaladores (comunicações e temporizações) e de malhas de soma, que simulam o processamento que normalmente seria efetuado pelos Transputers.

a) Primeiro teste

Neste teste foi dada ênfase para a construção SEQ e para o evento TEMPORIZAÇÃO, tendo sido utilizados três processos em um único Transputer interligado com um PC-hospedeiro.

A escalação empregada foi a seguinte:

```
SEQ
  PRI PAR
    Processo Roteador/Distribuidor
    SEQ
      rel ? ti
      P1
      P2
      P3
      rel ? tf
```

onde a codificação em Occam-2 dos processos utilizados é mostrada a seguir:

- processo P1:

```
SEQ
  rel ? t10
  sum1:=0
  SEQ i=1 FOR 1100
    sum1:=sum1+i
  atraso:=10
  rel ? t11
  rel ? AFTER t11 PLUS atraso
  rel ? t12
  sum1:=0
  SEQ i=1 FOR 1200
    sum1:=sum1+i
  rel ? t13
```

- processo P2:

```
SEQ
  rel ? t20
  sum2:=0
  SEQ i=1 FOR 2100
    sum2:=sum2+i
  atraso:=20
  rel ? t21
  rel ? AFTER t21 PLUS atraso
  rel ? t22
  sum2:=0
  SEQ i=1 FOR 2200
    sum2:=sum2+i
  rel ? t23
```

- processo P3:

```
SEQ
  rel ? t30
  sum3:=0
  SEQ i=1 FOR 3100
    sum3:=sum3+i
  atraso:=30
```



```

rel ? t31
rel ? AFTER t31 PLUS atraso
rel ? t32
sum3:=0
SEQ i=1 FOR 3200
    sum3:=sum3+i
rel ? t33

```

Convém observar que os processos P1, P2 e P3 são similares, exceto pelo atraso imposto e pelo número de repetições das malhas de soma.

Deve-se observar que nos processos acima foram deixadas as instruções de coleta de tempo (rel ? tij). Essas instruções somente devem ser colocadas nos processos na hora de coletar os tempos nas construções SEQ e PAR e devem ser retiradas na hora de colocar os processos escalados nos Transputers. A razão dessas instruções terem sido deixadas é para ilustrar o leitor em que pontos elas devem ser colocadas para efetuar as coletas de tempo.

Para a execução dos testes, os tempos relativos a cada tarefa foram coletados seguindo o proposto no sub-item 6.2.1.2 e cujos valores obtidos são mostrados na tabela 6.5.

Processo P1		Processo P2		Processo P3	
t10	6623	t20	6623	t30	6623
t11	6648	t21	6670	t31	6692
t12	6659	t22	6691	t32	6723
t13	6685	t23	6739	t33	6793

Tempos em ticks não prioritários=64 μ s

Tab. 6.5 Valores de tempo coletados no primeiro teste

Os valores de tempo obtidos são transferidos para as tabelas padronizadas no sub-item 6.2.1.1 e mostradas a seguir. O cálculo da duração de um evento é feito através da consulta das tabelas dos tempos coletados. Por exemplo, para determinar a duração do primeiro evento de P1, executa-se o seguinte cálculo:

$$\tau_{1^{\circ}\text{EV},\text{SEQ}}(\text{P1}) = t_{11} - t_{10} = 25 \text{ ticks não prioritários}$$

- dados do processo P1:

$\tau_{\text{EV},\text{SEQ}}$	$\tau_{\text{EV},\text{PAR}}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
25	---	3	--	--	---	--	--	11
26	---	4	--	--	---	--	--	--

- dados do processo P2:

$\tau_{\text{EV},\text{SEQ}}$	$\tau_{\text{EV},\text{PAR}}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
47	---	3	--	--	---	--	--	21
48	---	4	--	--	---	--	--	--

- dados do processo P3:

$\tau_{\text{EV},\text{SEQ}}$	$\tau_{\text{EV},\text{PAR}}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
69	---	3	--	--	---	--	--	31
70	---	4	--	--	---	--	--	--

Executando-se as tarefas em um Transputer, resulta o seguinte tempo de processamento:

$t_i = 6623$ ticks não prioritários

$t_f = 6971$ ticks não prioritários

$W_R = t_f - t_i = 348$ ticks não prioritários

Convertendo W_R para μs , resulta:

$$W_R = 348 * 64 = 22.272 \mu s$$

Acionando o SIMREDE, resulta o seguinte comprimento simulado da escalação:

$$W_S = 22.272 \mu s$$

Comparando-se os resultados obtidos, resulta um desvio=0%.

b) Segundo teste

Neste teste foi dada ênfase para a construção PAR e para os eventos RECEPÇÃO e TRANSMISSÃO, tendo sido utilizados três processos em um único Transputer interligado com um PC-hospedeiro.

A escalação empregada foi a seguinte:

```
SEQ
  PRI PAR
    Processo Roteador/Distribuidor
  SEQ
    rel ? ti
  PAR
    P1
    P2
    P3
  rel ? tf
```

onde os processos utilizados são descritos a seguir:

- processo P1:

```
SEQ
  rel ? t10
  sum1:=0
  SEQ i=1 FOR 1100
    sum1:=sum1+i
  rel ? t11
  canal.interno[00101] ? msg
  rel ? t12
  sum1:=0
  SEQ i=1 FOR 1200
    sum1:=sum1+i
  rel ? t13
  canal.interno[00102] ? msg
  rel ? t14
```

```

sum1:=0
SEQ i=1 FOR 1300
  sum1:=sum1+i
rel ? t15
msg[0]:=0100301
interno.ent ! msg
rel ? t16
sum1:=0
SEQ i=1 FOR 1400
  sum1:=sum1+i
rel ? t17

```

- processo P2:

```

SEQ
rel ? t20
sum2:=0
SEQ i=1 FOR 2100
  sum2:=sum2+i
rel ? t21
msg[0]:=0100101
interno.ent ! msg
rel ? t22
sum2:=0
SEQ i=1 FOR 2200
  sum2:=sum2+i
rel ? t23
canal.interno[00201] ? msg
rel ? t24
sum2:=0
SEQ i=1 FOR 2300
  sum2:=sum2+i
rel ? t25

```

- processo P3:

```

SEQ
rel ? t30
sum3:=0
SEQ i=1 FOR 3100
  sum3:=sum3+i
rel ? t31
msg[0]:=0100102
interno.ent ! msg
rel ? t32
sum3:=0
SEQ i=1 FOR 3200
  sum3:=sum3+i
rel ? t33
msg[0]:=0100201
interno.ent ! msg
rel ? t34
sum3:=0
SEQ i=1 FOR 3300
  sum3:=sum3+i
rel ? t35
canal.interno[00301] ? msg
rel ? t36
sum3:=0

```

```

SEQ i=1 FOR 3400
  sum3:=sum3+i
rel ? t37

```

O grafo de precedência gerado por este conjunto de tarefas não é acíclico. No entanto, como não se está testando o ESCEXT, mas apenas o SIMREDE, esta opção é aceitável, pois a única situação que impossibilitaria a execução do SIMREDE seria a alocação em SEQ, dentro do mesmo Transputer, de uma tarefa pai após sua tarefa filha.

Os tempos coletados de cada tarefa são apresentados na tabela 6.6.

Processo P1		Processo P2		Processo P3	
t10	6426	t20	6777	t30	6508
t11	6451	t21	6823	t31	6576
t12	6451	t22	6824	t32	6577
t13	6478	t23	6873	t33	6648
t14	6478	t24	6873	t34	6648
t15	6507	t25	6924	t35	6722
t16	6508			t36	6722
t17	6539			t37	6803

Tab. 6.6 Valores de tempo coletados no segundo teste

Todas as mensagens utilizadas neste teste e nos seguintes são compostas de dez números inteiros ([10]INT msg). Utilizando-se o procedimento recomendado no sub-item 6.2.1.2, obtiveram-se os dados mostrados a seguir:

- dados do processo P1:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	25	1	2	--	1	--	--	--
--	27	1	3	--	2	--	--	--
--	29	2	--	3	1	6	47	--
--	31	4	--	--	---	--	--	--

- dados do processo P2:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	46	2	--	1	1	6	47	--
--	49	1	3	--	1	--	--	--
--	51	4	--	--	---	--	--	--

- dados do processo P3:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	68	2	--	1	2	6	47	--
--	71	2	--	2	1	6	47	--
--	74	1	1	--	1	--	--	--
--	81	4	--	--	---	--	--	--

Os comprimentos real W_R e simulado W_S da escalação foram calculados conforme indicado a seguir, tendo-se medido $t_i=6506$ e $t_f=7062$:

$$W_R = 556 \cdot 64 = 35.584 \text{ } \mu s$$

$$W_S = 35.440 \text{ } \mu s$$

Comparando-se os resultados obtidos, obtém-se um desvio dado por:

$$\text{desvio} = (35.584 - 35.440) / 35.584 = 0,40\%$$

c) Terceiro teste

Neste teste foi dada ênfase para a construção SEQ e para todos os tipos de eventos, tendo sido utilizados quatro processos e dois Transputers T1 e T2, sendo que T1 se interligava com T2 e com um PC-hospedeiro.

A escalação empregada foi a seguinte:

- para o Transputer T1:

```
SEQ
  PRI PAR
    Processo Roteador/Distribuidor
  SEQ
    -- Mensagem sincronizadora
    msg[0]:=0205501
    interno.ent ! msg
    rel ? ti1
    P11
    P12
    rel ? tf1
```

- para o Transputer T2:

```
SEQ
  PRI PAR
    Processo Roteador/Distribuidor
  SEQ
    -- Mensagem sincronizadora
    canal.interno[05501] ? msg
    rel ? ti2
    P21
    P22
    rel ? tf2
```

A razão para se utilizar a mensagem sincronizadora de T1 para T2 é que o instante inicial de processamento das tarefas em cada Transputer é diferente. Ao iniciar o processamento na rede de Transputers, os relógios dos Transputers da rede vão sendo disparados à medida em que o programa carregador do Occam transfere para cada processador, através dos nós intermediários da rede, o código dos processos que lhe foram atribuídos e seus respectivos dados.

O tempo para carregar o Transputer com seus processos e liberá-lo para processamento varia conforme o tamanho dos processos e com a rota que o carregador toma para acessar os Transputers.

Visto que é através do Transputer que se comunica com o PC-hospedeiro, denominado Transputer raiz, que se efetua a transferência dos programas armazenados na memória de massa do hospedeiro para a memória local de cada Transputer, constata-se que o Transputer raiz é o último a ser liberado para início de processamento. No caso em apreço, T1 é o Transputer raiz.

Conforme foi mencionado na alínea "a" do sub-item 4.2.6.3, o modelo da rede pressupõe que todos os Transputers iniciam simultaneamente o processamento de suas tarefas. Para permitir que o processamento das tarefas nos Transputers da rede comece em instantes mais próximos, defasados do tempo de trânsito de uma mensagem, utiliza-se a mensagem sincronizadora.

Visto que os processos a serem utilizados neste e nos próximos testes possuem uma estrutura similar aos processos usados no primeiro e no segundo testes, emprega-se, para sua descrição, apenas o número de repetições das malhas de soma e a sequência dos eventos que os compõem, conforme indicado a seguir:

- processo P11:

- malhas de soma: 1110, 1120 e 1130 repetições;
- eventos: recepção e transmissão para 0202201;

- processo P12:

- malhas de soma: 1210, 1220 e 1230 repetições;
- eventos: transmissão para 0202202 e recepção;

- processo P21:

- malhas de soma: 2110, 3220, 2130 e 2140 repetições;
- eventos: transmissões para 0101201 e 0101101 e temporização (atraso:=20);

- processo P22:

- malhas de soma: 2210, 2220 e 2230 repetições; e
- eventos: dupla recepção.

Os tempos coletados de cada tarefa são mostrados na tabela 6.7.

Processo P11		Processo P12		Processo P21		Processo P22	
t110	6542	t120	6542	t210	6541	t220	6696
t111	6567	t121	6569	t211	6588	t221	6746
t112	6567	t122	6569	t212	6588	t222	6746
t113	6592	t123	6599	t213	6664	t223	6795
t114	6592	t124	6599	t214	6665	t224	6795
t115	6619	t125	6628	t215	6712	t225	6844
				t216	6733		
				t217	6780		

Tab. 6.7 Valores de tempo coletados no terceiro teste

Os dados obtidos acerca de cada tarefa são mostrados a seguir:

- dados do processo P11:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
25	---	1	21	--	1	--	--	--
25	---	2	--	22	1	6	47	--
27	---	4	--	--	---	--	--	--

- dados do processo P12:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
27	---	2	--	22	2	6	47	--
30	---	1	21	--	1	--	--	--
29	---	4	--	--	---	--	--	--

- dados do processo P21:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
47	---	2	--	12	1	6	47	--
76	---	2	--	11	1	6	47	--
47	---	3	--	--	-	--	--	21
47	---	4	--	--	---	--	--	--

- dados do processo P22:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
50	---	1	11	--	1	--	--	--
49	---	1	12	--	2	--	--	--
49	---	4	--	--	---	--	--	--

Os comprimentos real W_R e simulado W_S da escalação foram obtidos e chegou-se ao seguinte:

- para o Transputer T1, com $ti1=6696$ e $tf1=6958$:

$$W_{R,T1}=262*64=16.768 \mu s$$

$$W_{S,T1}=16.827 \mu s$$

- Para o Transputer T2, com $ti2=6698$ e $tf2=7086$:

$$W_{R,T2}=388*64=24.832 \mu s$$

$$W_{S,T2}=24.780 \mu s$$

Comparando-se os resultados obtidos, obteve-se um desvio de 0,35% para o Transputer T1 e de -0,21% para o Transputer T2.

d) Quarto teste

Neste teste foi dada ênfase para as construções SEQ e PAR e para todos os tipos de eventos, tendo sido utilizados seis processos e dois Transputers T1 e T2, sendo que T1 se interligava com T2 e com um PC-hospedeiro.

A escalação empregada foi a seguinte:

- para o Transputer T1:

```
SEQ
  PRI PAR
    Processo Roteador/Distribuidor
  SEQ
    transmissão msg sincronizadora
    rel ? ti1
  P11
  PAR
    P12
    P13
  rel ? tf1
```

- para o Transputer T2:

```
SEQ
  PRI PAR
    Processo Roteador/Distribuidor
  SEQ
    recepção msg sincronizadora
    rel ? ti2
  PAR
    P21
    P22
  P23
  rel ? tf2
```

As características dos processos utilizados neste teste são as seguintes:

- processo P11:

- malhas de soma: 1110, 1120, 1130 e 1140 repetições;
- eventos: transmissões para 0202101, 0202102 e 0101301;

- processo P12:
 - malhas de soma: 1210, 1220, 1230 e 1240 repetições;
 - eventos: tripla recepção;
- processo P13:
 - malhas de soma: 1310, 1320 e 1330 repetições;
 - eventos: recepção e transmissão para 0101202;
- processo P21:
 - malhas de soma: 2110, 2120, 2130 e 2140 repetições;
 - eventos: dupla recepção e transmissão para 0101203;
- processo P22:
 - malhas de soma: 2210, 2220 e 2230 repetições;
 - eventos: transmissão para 0202301 e 0101201.
- processo P23:
 - malhas de soma: 2310 e 2320 repetições; e
 - eventos: recepção.

Os tempos coletados das tarefas são mostrados na tabela 6.8.

Proc. P11		Proc. P12		Proc. P13		Proc. P21		Proc. P22		Proc. P23	
t110	6542	t120	6427	t130	6697	t210	6427	t220	6696	t230	6541
t111	6567	t121	6454	t131	6726	t211	6474	t221	6745	t231	6593
t112	6567	t122	6454	t132	6726	t212	6474	t222	6746	t232	6593
t113	6594	t123	6484	t133	6755	t213	6524	t223	6795	t233	6644
t114	6594	t124	6484	t134	6756	t214	6524	t224	6796		
t115	6620	t125	6511	t135	6785	t215	6572	t225	6845		
t116	6620	t126	6511			t216	6572				
t117	6645	t127	6539			t217	6620				

Tab. 6.8 Valores de tempo coletados no quarto teste

Os dados obtidos acerca de cada tarefa são mostrados a seguir:

- dados do processo P11:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
25	---	2	--	21	1	6	47	--
27	---	2	--	21	2	6	47	--
26	---	2	--	13	1	6	47	--
25	---	4	--	--	---	--	--	--

- dados do processo P12:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	27	1	22	--	1	--	--	--
--	30	1	13	--	2	--	--	--
--	27	1	21	--	3	--	--	--
--	28	4	--	--	---	--	--	--

- dados do processo P13:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	29	1	11	--	1	--	--	--
--	29	2	--	12	2	6	47	--
--	29	4	--	--	---	--	--	--

- dados do processo P21:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	47	1	11	--	1	--	--	--
--	50	1	11	--	2	--	--	--
--	48	2	--	12	3	6	47	--
--	48	4	--	--	---	--	--	--

- dados do processo P22:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	49	2	--	23	1	6	47	--
--	49	2	--	12	1	6	47	--
--	49	4	--	--	---	--	--	--

- dados do processo P23:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
52	---	1	22	--	1	--	--	--
51	---	4	--	--	---	--	--	--

Os comprimentos real W_R e simulado W_S da escalação foram obtidos e chegou-se ao seguinte:

- para o Transputer T1, com $ti1=6779$ e $tf1=7087$:

$$W_{R,T1}=308*64=19.712 \mu s$$

$$W_{S,T1}=19.571 \mu s$$

- Para o Transputer T2, com $ti2=6782$ e $tf2=7226$:

$$W_{R,T2}=444*64=28.416 \mu s$$

$$W_{S,T2}=28.456 \mu s$$

Comparando-se os resultados obtidos, obteve-se para o Transputer T1 um desvio de -0,72% e para o Transputer T2 um desvio de 0,14%.

e) Quinto teste

Neste teste foi dada ênfase para as construções SEQ e PAR e para todos os tipos de eventos, tendo sido utilizados dez

processos e quatro Transputers, interligados entre si e com o PC-hospedeiro, de acordo com a figura 6.9.

Fig. 6.9 Topologia da rede de Transputers utilizada para o quinto teste

A escalação empregada foi a seguinte:

- para o Transputer T1:

```
SEQ
  transmissão mensagem sincronizadora para T2
  transmissão mensagem sincronizadora para T3
PRI PAR
  Processo Roteador/Distribuidor
  SEQ
    rel ? ti1
    P11
    P12
    rel ? tf1
```

- para o Transputer T2:

```
SEQ
  recepção mensagem sincronizadora de T1
  transmissão mensagem sincronizadora para T4
PRI PAR
  Processo Roteador/Distribuidor
  SEQ
    rel ? ti2
    P21
  PAR
    P22
    P23
  rel ? tf2
```

- para o Transputer T3:

```
SEQ
  recepção mensagem sincronizadora de T1
```

```

PRI PAR
  Processo Roteador/Distribuidor
SEQ
  rel ? ti3
  PAR
    P31
    P32
  P33
  rel ? tf3

```

- para o Transputer T4:

```

SEQ
  recepção mensagem sincronizadora de T2
PRI PAR
  Processo Roteador/Distribuidor
SEQ
  rel ? ti4
  PAR
    P41
    P42
  rel ? tf4

```

As características dos processos utilizados neste teste são as seguintes:

- processo P11:

- malhas de soma: 1110 e 1120 repetições;
- eventos: transmissão para 0404101;

- processo P12:

- malhas de soma: 1210, 1220 e 1230 repetições;
- eventos: recepção e transmissão para 0202101;

- processo P21:

- malhas de soma: 2110, 2120 e 2130 repetições;
- eventos: transmissão para 0303301 e recepção;

- processo P22:

- malhas de soma: 2210 e 2220 repetições;
- eventos: transmissão para 0303201;

- processo P23:
 - malhas de soma: 2310 e 2320 repetições;
 - eventos: transmissão para 0303101;
- processo P31:
 - malhas de soma: 3110, 3120 e 3130 repetições;
 - eventos: recepção e transmissão para 0303202;
- processo P32:
 - malhas de soma: 3210, 3220 e 3230 repetições;
 - eventos: dupla recepção;
- processo P33:
 - malhas de soma: 3310, 3320 e 3330 repetições;
 - eventos: recepção e temporização (atraso:=30);
- processo P41:
 - malhas de soma: 4110, 4120 e 4130 repetições;
 - eventos: recepção e transmissão para 0404201;
- processo P42:
 - malhas de soma: 4210, 4220 e 4230 repetições; e
 - eventos: transmissão para 0101201 e recepção.

Os tempos coletados de cada tarefa são mostrados na tabela 6.9.

Proc. P11		Proc. P12		Proc. P21		Proc. P22		Proc. P23	
t110	6540	t120	6542	t210	6542	t220	6541	t230	6541
t111	6565	t121	6569	t211	6588	t221	6593	t231	6595
t112	6565	t122	6569	t212	6589	t222	6594	t232	6596
t113	6592	t123	6596	t213	6640	t223	6646	t233	6651
		t124	6596	t214	6640				
		t125	6626	t215	6690				

Proc. P31		Proc. P32		Proc. P33		Proc. P41		Proc. P42	
t310	6696	t320	6695	t330	6541	t410	6695	t420	6424
t311	6765	t321	6767	t331	6614	t411	6786	t421	6524
t312	6765	t322	6767	t332	6614	t412	6786	t422	6524
t313	6834	t323	6838	t333	6688	t413	6878	t423	6618
t314	6835	t324	6838	t334	6719	t414	6878	t424	6619
t315	6904	t325	6910	t335	6792	t415	6976	t425	6712

Tab. 6.9 Valores de tempo coletados no quinto teste

Os dados obtidos acerca de cada tarefa são mostrados a seguir:

- dados do processo P11:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
25	---	2	--	41	1	6	47	--
27	---	4	--	--	---	--	--	--

- dados do processo P12:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
27	---	1	42	--	1	--	--	--
27	---	2	--	21	1	6	47	--
30	---	4	--	--	---	--	--	--

- dados do processo P21:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
46	---	2	--	33	1	6	47	--
51	---	1	12	--	1	--	--	--
50	---	4	--	--	---	--	--	--

- dados do processo P22:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	52	2	--	32	1	6	47	--
--	52	4	--	--	---	--	--	--

- dados do processo P23:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	54	2	--	31	1	6	47	--
--	55	4	--	--	---	--	--	--

- dados do processo P31:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	69	1	23	--	1	--	--	--
--	69	2	--	32	2	6	47	--
--	69	4	--	--	---	--	--	--

- dados do processo P32:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	72	1	22	--	1	--	--	--
--	71	1	31	--	2	--	--	--
--	72	4	--	--	---	--	--	--

- dados do processo P33:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
73	---	1	21	--	1	--	--	--
74	---	3	--	--	-	--	--	31
73	---	4	--	--	---	--	--	--

- dados do processo P41:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	91	1	11	--	1	--	--	--
--	92	2	--	42	1	6	47	--
--	98	4	--	--	---	--	--	--

- dados do processo P42:

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
--	100	2	--	12	1	6	47	--
--	94	1	41	--	1	--	--	--
--	93	4	--	--	---	--	--	--

Os comprimentos real W_R e simulado W_S da escalação foram obtidos e chegou-se ao seguinte:

- para o Transputer T1, com $ti1=6592$ e $tf1=6844$:

$$W_{R,T1}=252*64=16.128 \mu s$$

$$W_{S,T1}=16.293 \mu s$$

- para o Transputer T2, com $ti2=6595$ e $tf2=7086$:

$$W_{R,T2}=491*64=31.424 \mu s$$

$$W_{S,T2}=31.268 \mu s$$

- para o Transputer T3, com $ti3=6602$ e $tf3=7515$:

$$W_{R,T3}=913*64=58.432 \mu s$$

$$W_{S,T3}=58.509 \mu s$$

- para o Transputer T4, com $ti4=6608$ e $tf4=7171$:

$$W_{R,T4}=563*64=36.032 \mu s$$

$$W_{S,T4}=36.418 \mu s$$

Comparando-se os resultados obtidos, obtiveram-se os seguintes desvios:

- para o Transputer T1: 1,02%;
- para o Transputer T2: -0,50%;
- para o Transputer T3: 0,13%; e
- para o Transputer T4: 1,07%.

6.3.3.2 Conclusões a Respeito do Desempenho do SIMREDE

Compilam-se, na tabela 6.10, os resultados dos cinco testes realizados no sub-item 6.3.3.1, os quais geraram dez comprimentos de escalação.

Nº do teste	Nº do Transputer	Desvios obtidos (%)
1	T1	0,0
2	T1	0,40
3	T1 T2	0,35 -0,21
4	T1 T2	-0,72 0,14
5	T1 T2 T3 T4	1,02 -0,50 0,13 1,07

Tab. 6.10 Compilação dos resultados obtidos nos testes de avaliação do SIMREDE

A análise da tabela 6.11 indica os dois seguintes fatos:

- o SIMREDE apresentou um desempenho bastante superior ao especificado na alínea "a" do item 4.2.2, caracterizando um modelo confiável da rede de Transputers; e
- a forma proposta no sub-item 6.2.1.2 para coleta dos dados de entrada dos processos que envolvem tempo, mostrou-se adequada, pois, caso contrário, ainda que o modelo fosse perfeito, os resultados obtidos com dados incorretos seriam ruins.

Podem-se levantar as seguintes razões para as pequenas discrepâncias existentes entre os resultados real e simulado:

- na coleta dos tempos dos eventos de cada processo, os tempos são medidos em ticks não-prioritários, o que torna baixa a precisão desses valores;

- mesmo com a utilização das mensagens sincronizadoras, o início do processamento das tarefas nos Transputers não é simultâneo, contrariando uma premissa assumida no modelo;
- o primeiro quantum de tempo alocado ao primeiro processo escalado é predefinido no modelo (24 ticks), mas na prática é um número entre 16 e 32; e
- há pausas ocasionais dos processos não-prioritários, interrompidos temporariamente pela passagem de mensagens em trânsito ou pela chegada de mensagens ao Processo Roteador/Distribuidor. O primeiro efeito é desprezado no modelo, devido ao baixo valor dos atrasos (da ordem de $\tau_{ROT}=10\mu s$) e o segundo é parcialmente corrigido, ao se adicionar o somatório desses atrasos ao comprimento da escalação, no final da simulação do SIMREDE, através do vetor VSARD. Na prática, essas interrupções acarretam a parada temporária do processo escalado, com a conseqüente perda de tempo no processamento das tarefas alocadas naquele Transputer.

6.3.4 Teste de Aceitação do Programa Escalador

Cumpra agora testar o Programa Escalador totalmente integrado. Para tanto, seguem-se os passos recomendados na seção 6.2.

6.3.4.1 Apresentação do Sistema de Tarefas e da Rede de Transputers

Apresentam-se, neste sub-item, as informações que caracterizam o sistema de tarefas e a rede de Transputers.

a) Apresentação do sistema de tarefas

Propõe-se empregar, neste teste, vinte processos, relacionados pelo grafo de precedência mostrado na figura 6.10.

Fig. 6.10 Grafo de precedência usado para testar o Programa Escalador

Apresentam-se, a seguir, as características dos vinte processos empregados neste teste:

- processo P1 dado pela seguinte codificação em Occam-2:

```
SEQ
  rel ? t10
  sum1:=0
  SEQ i=0 FOR 1000
    sum1:=sum1+i
  rel ? t11
  msg[0]:=0400401
  interno.ent ! msg
  rel ? t12
```

- processo P2 dado pela seguinte codificação em Occam-2:

```
SEQ
  rel ? t20
  sum2:=0
  SEQ i=0 FOR 1500
    sum2:=sum2+i
  rel ? t21
  msg[0]:=0400402
  interno.ent ! msg
  rel ? t22
```

- processo P3:

- malhas de soma: 1600, 1200 e 2000 repetições;
- eventos: transmissões para 0301303, 0201401 e 0101501;

- processo P4:

- malhas de soma: 2600, 3200 e 3000 repetições;
- eventos: dupla recepção e transmissão para 0300501;

- processo P5:

- malhas de soma: 1100, 4500 e 1300 repetições;
- eventos: recepção e transmissões para 0100701 e 0400601;

- processo P6:

- malhas de soma: 6000 e 2300 repetições;
- eventos: recepção e transmissão para 0200801;

- processo P7:

- malhas de soma: 3000 e 3000 repetições;
- eventos: recepção e transmissão para 0300901;

- processo P8:
 - malhas de soma: 1400 e 5000 repetições;
 - eventos: recepção e transmissão para 0401001;
- processo P9:
 - malhas de soma: 3800 e 3400 repetições;
 - eventos: recepção e transmissão para 0101101;
- processo P10:
 - malhas de soma: 3300, 2200 e 4400 repetições;
 - eventos: recepção e transmissões para 0201202 e 0301301;
- processo P11:
 - malhas de soma: 1600, 5500 e 2900 repetições;
 - eventos: recepção e transmissões para 0301302 e 0201201;
- processo P12:
 - malhas de soma: 2500, 4300, 1800 e 6700 repetições;
 - eventos: recepção, transmissão para 0301601, recepção e transmissão para 0401701;
- processo P13:
 - malhas de soma: 3100, 4200, 1500, 5400 e 2800 repetições;
 - eventos: tripla recepção e transmissões para 0101801 e 0201901;
- processo P14:
 - malhas de soma: 7000 e 5000 repetições;
 - eventos: recepção e transmissão para 0402001;
- processo P15:
 - malhas de soma: 1700 e 7300 repetições;
 - eventos: recepção e transmissão para 0402002;

- processo P16:
 - malhas de soma: 1900 e 1200 repetições;
 - eventos: recepção e transmissão para 0402003;
- processo P17:
 - malhas de soma: 8000 e 3000 repetições;
 - eventos: recepção e transmissão para 0402004;
- processo P18:
 - malhas de soma: 6500 e 5500 repetições;
 - eventos: recepção e transmissão para 0402005;
- processo P19:
 - malhas de soma: 2500 e 4500 repetições;
 - eventos: recepção e transmissão para 0402006;
- processo P20:
 - malhas de soma: 6 malhas de 1000 e 1 malha de 9000 repetições; e
 - eventos: sêxtupla recepção.

Deve-se notar que nos processos deste teste, diferentemente do que ocorre nos testes do sub-item 6.3.3.1, não há simulação de processamento (malha de soma) no final do processo, sendo que em seu encerramento sempre ocorre uma comunicação. O único processo deste teste que possui uma malha de soma no final é P20, simbolizando o tempo dispendido por esse processo para transferir os dados para o PC-hospedeiro.

b) Apresentação da rede de Transputers

A topologia da rede de transputers empregada neste teste é a mesma da figura 6.9.

6.3.4.2 Montagem do Arquivo de Dados de Entrada dos Processos

Efetuem-se os procedimentos indicados no sub-item 6.2.1.2 para coletar os dados de entrada dos processos que envolvem tempo. De posse desses dados, monta-se o arquivo de dados dos processos, pronto para ser inserido como entrada no Programa Escalador.

O primeiro dado a figurar no arquivo de entrada é o número n de processos, que neste caso é 20.

Em seguida, deve-se entrar com as relações de precedência, conforme ilustrado pela tabela 6.11.

Pai	Filhos
"P0"	P1 P2 P3
"P1"	P4
"P2"	P4
"P3"	P13 P14 P15
"P4"	P5
"P5"	P6 P7
"P6"	P8
"P7"	P9
"P8"	P10
"P9"	P11
"P10"	P12 P13
"P11"	P12 P13
"P12"	P16 P17
"P13"	P18 P19
"P14"	P20
"P15"	P20
"P16"	P20
"P17"	P20
"P18"	P20
"P19"	P20
"P20"	P21

Tab. 6.11 Relações de precedência dos vinte processos usados no teste de aceitação do Programa Escalador

O tempo de processamento das tarefas, conforme mostrado na tabela 6.12, deve ser o próximo conjunto de dados a ser inserido no arquivo de entrada dos processos.

Identificação do processo	TSEQ	TPAR
"P1"	22	24
"P2"	33	36
"P3"	107	110
"P4"	201	196
"P5"	155	154
"P6"	184	194
"P7"	133	133
"P8"	142	142
"P9"	160	160
"P10"	220	220
"P11"	222	223
"P12"	355	341
"P13"	383	386
"P14"	274	277
"P15"	200	200
"P16"	69	69
"P17"	244	256
"P18"	275	276
"P19"	156	155
"P20"	339	349

Tab. 6.12 Tempo de processamento das vinte tarefas usadas no teste de aceitação do Programa Escalador

O próximo conjunto de dados a entrar no arquivo de entrada dos processos são os dados a respeito dos eventos, conforme indicado a seguir:

- processo "P1":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
22	23	2	--	P4	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P2":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
33	36	2	--	P4	2	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P3":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
35	36	2	--	P13	3	4	16	--
27	29	2	--	P14	1	4	16	--
45	45	2	--	P15	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P4":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
63	58	1	P1	--	1	--	--	--
71	71	1	P2	--	2	--	--	--
66	66	2	--	P5	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P5":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
25	24	1	P4	--	1	--	--	--
100	100	2	--	P7	1	4	16	--
29	28	2	--	P6	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P6":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
133	142	1	P5	--	1	--	--	--
51	51	2	--	P8	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P7":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
66	66	1	P5	--	1	--	--	--
66	67	2	--	P9	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P8":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
31	31	1	P6	--	1	--	--	--
111	111	2	--	P10	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P9":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
84	84	1	P7	--	1	--	--	--
84	75	2	--	P11	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P10":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
73	73	1	P8	--	1	--	--	--
49	49	2	--	P12	2	4	16	--
98	98	2	--	P13	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P11":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
35	35	1	P9	--	1	--	--	--
122	122	2	--	P13	2	4	16	--
65	64	2	--	P12	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P12":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
59	55	1	P11	--	1	--	--	--
96	96	2	--	P16	1	4	16	--
41	41	1	P10	--	2	--	--	--
158	148	2	--	P17	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P13":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
74	70	1	P10	--	1	--	--	--
93	93	1	P11	--	2	--	--	--
33	36	1	P3	--	3	--	--	--
119	119	2	--	P18	1	4	16	--
62	66	2	--	P19	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P14":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
155	166	1	P3	--	1	--	--	--
119	110	2	--	P20	1	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P15":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
38	37	1	P3	--	1	--	--	--
161	161	2	--	P20	2	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P16":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
42	42	1	P12	--	1	--	--	--
27	26	2	--	P20	3	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P17":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
177	189	1	P12	--	1	--	--	--
67	67	2	--	P20	4	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P18":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
144	154	1	P13	--	1	--	--	--
130	122	2	--	P20	5	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P19":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
55	55	1	P13	--	1	--	--	--
99	100	2	--	P20	6	4	16	--
0	0	4	--	--	-	--	--	--

- processo "P20":

$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	TIPO	P _T	P _R	NSM	τ_{DMINT}	τ_{DMEXT}	τ_{ATR}
24	24	1	P14	--	1	--	--	--
25	25	1	P15	--	2	--	--	--
22	22	1	P16	--	3	--	--	--
22	22	1	P17	--	4	--	--	--
24	24	1	P18	--	5	--	--	--
22	22	1	P19	--	6	--	--	--
200	200	4	--	--	-	--	--	--

Convém observar que, em todos os processos, a mensagem utilizada consistia em um vetor de três números inteiros.

6.3.4.3 Montagem do Arquivo de Dados de Entrada dos Transputers

O primeiro dado a ser inserido no arquivo de entrada de dados dos Transputers é o numero m de Transputers, que neste caso é 4.

Em seguida, monta-se a matriz MNNI, mostrada na tabela 6.13.

	"T1"	"T2"	"T3"	"T4"
"T1"	0	0	0	0
"T2"	0	0	0	0
"T3"	0	0	0	0
"T4"	0	0	0	0

Tab. 6.13 Matriz MNNI relativa ao teste de aceitação do Programa Escalador

6.3.4.4 Apresentação dos Resultados

O arquivo de saída do Programa Escalador fornece as informações mostradas na figura 6.11.

```
Transputer T1:
P2
P15
P7
P11
P18

Transputer T2:
P1
P14
P8
P12
P19

Transputer T3:
P3
P5
P9
P13
P16

Transputer T4:
P4
P6
P10
P17
P20

WESCOBT = 88450 microsegundos

WCR = 123648 microsegundos
```

Fig. 6.11 Apresentação do resultado da escalação gerada no teste de aceitação do Programa Escalador

6.3.4.5 Avaliação dos Resultados do Teste de Aceitação

Ao se processar esse conjunto de tarefas em quatro Transputers, obedecendo-se a escalação proposta na figura 6.11, obtiveram-se os resultados apresentados na tabela 6.14.

Número do Transputer (T _i)	Instante inicial de processamento (t _i)	Instante final de processam. (t _f)	Tempo total de processamento (W _R (T _i))
T1	6738	7771	1033
T2	6743	7879	1136
T3	6749	7790	1041
T4	6756	8140	1384

Tab. 6.14 Tempo real de processamento em quatro Transputers da escalação proposta na figura 6.11

Visto que o comprimento da escalação é definido como o maior dentre os tempos de processamento, resulta que $W_R = 1384 \times 64 = 88.576 \mu s$. Comparando W_R com W_S e lembrando que $W_S = 88.450 \mu s$ (vide figura 6.11), resulta o seguinte desvio:

$$\text{Desvio} = \frac{W_S - W_R}{W_R} = -0,14\%$$

Este resultado mostra uma conformidade quase perfeita entre W_R e W_S , mostrando a operacionalidade do SIMREDE e aprovando-o como um modelo válido para representar redes de Transputers.

Há um outro ponto interessante a ser ressaltado como resultado deste teste: em uma escalação convencional, o melhor resultado possível de se obter é WCR (vide alínea "c" do sub-item 2.2.2.2). Verifica-se que, neste caso, o comprimento obtido da escalação WESCOBT corresponde a 71,53% de WCR. Isto comprova a eficiência deste método de escalação utilizando Transputers, quando comparado com métodos convencionais.

6.3.4.6 Observações Importantes a Respeito do Teste de Aceitação

A título de informação, apresentam-se, nas figuras 6.12, 6.13 e 6.14 as escalações geradas pelos dez algoritmos heurísticos disponíveis no ESCEXT.

CP/NSI	CP/NSI/PAR	$N/\tau_{SEQ}/CP$	$N/\tau_{SEQ}/CP/PAR$
Transputer T1: P2 P7 P8 P3 P12 P16 P20	Transputer T1: P2 P7 P8 P3 P12 P16 P20	Transputer T1: P3 P5 P9 P12 P19	Transputer T1: P3 P5 P9 P12 P19
Transputer T2: P1 P6 P10 P14 P19	Transputer T2: P1 P6 P10 P14 P19	Transputer T2: P2 P15 P7 P11 P18 P20	Transputer T2: P2 P15 P7 P11 P18 P20
Transputer T3: P4 P11 P18 P17	Transputer T3: P4 P11 P18 P17	Transputer T3: P1 P4 P6 P10 P17	Transputer T3: P1 P4 P6 P10 P17
Transputer T4: P5 P9 P13 P15	Transputer T4: P5 P9 P13 P15	Transputer T4: P14 P8 P13 P16	Transputer T4: P14 P8 P13 P16
WESCOBT=97680	WESCOBT=93248	WESCOBT=88500	WESCOBT=88500

Fig. 6.12 Escalações geradas pelos algoritmos CP/NSI, CP/NSI/PAR, $N/\tau_{SEQ}/CP$ e $N/\tau_{SEQ}/CP/PAR$

N/CP	N/CP/PAR	N/CPn/NSI	N/CPn/NSI/PAR
Transputer T1: P2 P15 P7 P11 P18	Transputer T1: P2 P15 P7 P11 P18	Transputer T1: P1 P14 P9 P12 P19	Transputer T1: P1 P14 P9 P12 P19
Transputer T2: P1 P14 P8 P12 P19	Transputer T2: P1 P14 P8 P12 P19	Transputer T2: P2 P15 P6 P13 P16	Transputer T2: P2 P15 P6 P13 P16
Transputer T3: P3 P5 P9 P13 P16	Transputer T3: P3 P5 P9 P13 P16	Transputer T3: P3 P5 P8 P11 P17	Transputer T3: P3 P5 P8 P11 P17
Transputer T4: P4 P6 P10 P17 P20	Transputer T4: P4 P6 P10 P17 P20	Transputer T4: P4 P7 P10 P18 P20	Transputer T4: P4 P7 P10 P18 P20
WESCOBT= 88450	WESCOBT=88450	WESCOBT=92098	WESCOBT=92098

Fig. 6.13 Escalações geradas pelos algoritmos N/CP, N/CP/PAR, N/CPn/NSI e N/CPn/NSI/PAR

Algoritmo LPT	Algoritmo MTP
Transputer T1: P13 P10 P9 P5 P2	Transputer T1: P13 P20 P11
Transputer T2: P20 P11 P4 P8 P16	Transputer T2: P12 P14 P18 P16
Transputer T3: P12 P17 P6 P7 P1	Transputer T3: P17 P10 P15 P4 P2 P1 P3
Transputer T4: P14 P18 P15 P19 P3	Transputer T4: P6 P9 P19 P5 P8 P7
WESCOBT=99330 μ s	WESCOBT=123314 μ s

Fig. 6.14 Escalações geradas pelos algoritmos LPT e MTP

A análise das escalações mostradas nas figuras 6.12, 6.13 e 6.14 indica que a melhor escalção para esse caso foi obtida pelos algoritmos N/CP e N/CP/PAR. Percebe-se que a compactação PAR no caso do algoritmo CP/NSI/PAR redundou em um resultado melhor que no caso do algoritmo CP/NSI, tornando válida essa opção nos algoritmos propostos. Nota-se ainda que o pior resultado foi obtido pelo algoritmo MTP.

Foi feito um novo teste, tomando-se o mesmo grafo de precedência e a mesma rede de Transputers. As tarefas foram tomadas com durações diferentes dos eventos e dos tempos de comunicação. Os novos tempos de processamento utilizados para o teste são mostrados na tabela 6.15.

Identificação do processo	TSEQ	TPAR
"P1"	31	34
"P2"	47	52
"P3"	62	69
"P4"	78	85
"P5"	63	68
"P6"	110	121
"P7"	156	172
"P8"	79	86
"P9"	16	17
"P10"	32	35
"P11"	126	138
"P12"	94	102
"P13"	140	153
"P14"	78	86
"P15"	156	172
"P16"	16	17
"P17"	47	51
"P18"	78	86
"P19"	126	138
"P20"	143	154

Tab. 6.15 Tempo de processamento das tarefas para execução de um novo teste

Os novos intervalos de duração dos eventos de cada tarefa são mostrados na figura 6.15.

Processo "P1":		Processo "P2":		Processo "P3":		Processo "P4":	
$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$
31	34	47	52	23	26	16	17
0	0	0	0	23	26	31	34
				16	17	31	34
				0	0	0	0

Processo "P5":		Processo "P6":		Processo "P7":		Processo "P8":	
$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$
16	17	63	69	78	86	16	17
16	17	47	52	78	86	63	69
31	34	0	0	0	0	0	0
0	0						

Processo "P9":		Processo "P10":		Processo "P11":		Processo "P12":	
$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$
13	14	8	9	16	17	31	34
3	3	16	17	47	52	16	17
0	0	8	9	63	69	16	17
		0	0	0	0	31	34
						0	0

Fig. 6.15 Duração dos eventos das tarefas na execução de um novo teste

Processo "P13":		Processo "P14":		Processo "P15":		Processo "P16":	
$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$
16	17	47	52	109	120	3	3
31	34	31	34	47	52	13	14
31	34	0	0	0	0	0	0
31	34						
31	34						
0	0						

Processo "P17":		Processo "P18":		Processo "P19":		Processo "P20":	
$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$	$\tau_{EV,SEQ}$	$\tau_{EV,PAR}$
16	17	31	34	63	69	16	17
31	34	47	52	63	69	16	17
0	0	0	0	0	0	16	17
						16	17
						16	17
						16	17
						47	52

Fig. 6.15 Duração dos eventos das tarefas na execução de um novo teste (cont.)

Os tempos de comunicação das tarefas são alterados para $\tau_{DMINT}=3 \mu s$ e $\tau_{DMEXT}=24 \mu s$.

Os comprimentos das escalas obtidas são mostrados na tabela 6.16.

Algoritmo	Comprimento simulado da escalação (W_S)
CP/NSI	40.730 μs
CP/NSI/PAR	40.090 μs
N/ τ_{SEQ} /CP	41.507 μs
N/ τ_{SEQ} /CP/PAR	41.507 μs
N/CP	39.680 μs
N/CP/PAR	40.592 μs
N/CPn/NSI	39.680 μs
N/CPn/NSI/PAR	40.592 μs
LPT	51.085 μs
MTP	46.384 μs

Tab. 6.16 Comprimentos das escalações obtidas no novo teste para os dez algoritmos

A melhor escalação (39.680 μs) foi obtida pelos algoritmos N/CP e N/CPn/NSI e a pior pelo algoritmo LPT. Neste caso, a compactação PAR foi melhor no caso do algoritmo CP/NSI e pior no caso dos algoritmos N/CP e N/CPn/NSI.

6.3.5 Comentários Finais a Respeito do Programa Escalador

Os seguintes comentários são relevantes a respeito do desenvolvimento/aplicação do Programa Escalador:

- houve um número muito pequeno de erros na depuração dos módulos do Programa Escalador. Na depuração e integração dos módulos do SIMREDE, detetaram-se apenas dois erros; no caso do ESCEXT um único erro foi detetado; no caso do Gerenciador do Programa

Escalador houve apenas um erro e ao se realizar a integração das três partes do Programa Escalador não se detetou erro nenhum. Esses resultados ressaltam a importância de se seguir um método de Engenharia de Software para desenvolver programas;

- as comparações de W_R com W_S nos testes efetuados com o SIMREDE revelaram que o modelamento da rede de Transputers apresentou resultados melhores que os solicitados na especificação de requisitos (vide alínea "a" do item 4.2.2);
- o Programa Escalador manipula o número de processos e de Transputers solicitado na alínea "b" do item 4.2.2;
- o Programa Escalador é portátil, conforme requerido na alínea "c" do item 4.2.2;
- ao se executar o Programa Escalador com o conjunto de vinte tarefas apresentadas na figura 6.10 em um microcomputador PC-compatível, munido de um microprocessador 80386 de 25 MHz, resultou em um tempo de processamento de apenas 1,3 segundos. Esse tempo está dentro das expectativas apresentadas na especificação de requisitos (vide alínea "d" do item 4.2.2); e
- por fim, a exigência listada na alínea "e" do item 4.2.2, a respeito da aplicação do Programa Escalador a qualquer modelo, frequência de operação ou taxa de transmissão das vias de comunicação dos Transputers, desde que a rede seja homogênea, é atendida, na medida em que os tempos envolvidos no modelamento ($\tau_{SEQ/PAR}$, $\tau_{EV,SEQ/PAR}$, $\tau_{DMINT/DMEXT}$ e τ_{ATR}) são informados ao Programa Escalador pelo usuário, baseado em uma coleta de dados feita na rede de Transputers que lhe é disponível e visto que o modo de operação do Escalador Interno dos Transputers independe de seu modelo.

7 - CONSIDERAÇÕES FINAIS

Este capítulo apresenta um balanço global do presente trabalho, em que se faz uma análise crítica a respeito do que foi feito, mostra-se sua colaboração dentro do campo de escalação estática e se analisam as perspectivas de continuação/otimização do mesmo.

7.1 Análise Crítica

A proposta de se criar um novo conceito em termos de escalação estática de tarefas, através da simulação da rede de processadores utilizada, foi plenamente atingida. Logrou-se êxito no modelamento da rede de Transputers e em sua aplicação na avaliação de algoritmos heurísticos de escalação. Criou-se um programa que coleta dados dos processos e da rede de processadores e devolve uma escalação sub-ótima, pronta para ser implantada na rede de Transputers.

Há, no entanto, alguns melhoramentos que poderiam ser implementados no Programa Escalador, em particular no SIMREDE. O modelo da rede de Transputers poderia ser melhorado, incluindo, por exemplo, o modelamento da construção ALT, os atrasos no processamento das tarefas causados pela passagem de mensagens em trânsito através de nós intermediários da rede de Transputers e os atrasos no processamento das tarefas residentes nos Transputers origem e destino de mensagens, considerados no instante em que a mensagem é transmitida ou recebida. Este último efeito é considerado no SIMREDE, mas de forma simplificada

através do VSARD, totalizando os atrasos causados pela transmissão/recepção de mensagens e somando-os ao tempo de processamento do Transputer no final do processamento de suas tarefas.

A construção ALT poderia ser encarada pelo SIMREDE como uma construção SEQ composta de uma múltipla recepção de mensagens, a qual seria liberada assim que a primeira mensagem chegasse. Para implementar os efeitos da passagem de mensagens em trânsito, seria necessário coletar a identificação dos nós intermediários que interligassem todos os pares de Transputers da rede e, de posse dessas informações, calcular os instantes de chegada da mensagem em trânsito em cada nó por que ela passasse. Por fim, para considerar os efeitos dos atrasos causados pelo envio ou recebimento de mensagens nos Transputers transmissor e receptor das mesmas, seria necessário incluir no IPPEV de cada Transputer os instantes em que esses eventos terminassem, de forma que enquanto o Processo Roteador/Distribuidor estivesse ativo, os processos de baixa prioridade ficassem inativos.

7.2 Colaboração deste Trabalho no Campo de Escalação Estática

A principal colaboração deste trabalho reside no fato de se utilizar, pela primeira vez, um modelo da rede de processadores em que a escalação será implantada para avaliá-la. Esse ineditismo, tornado possível devido aos recursos computacionais cada vez mais disseminados, permite que se gere uma escalação heurística e imediatamente se analise seu comprimento, abrindo novas perspectivas para a evolução na área de escalação estática sub-ótima em multiprocessadores. Através dessa ferramenta,

dezenas de diferentes algoritmos podem ser testados em um curto espaço de tempo, possibilitando a seleção do melhor, de forma rápida e eficiente.

7.3 Continuidade do Trabalho

Há certos aprimoramentos, descritos a seguir, que poderiam ser implementados no Programa Escalador, a fim de torná-lo uma ferramenta mais eficiente de escalação de tarefas em redes de Transputers.

Uma primeira sugestão seria a coleta automática de dados que envolvessem tempo, o que implicaria em medir os intervalos relativos a cada evento de forma automatizada. Nesse caso, o usuário geraria arquivos que contivessem cada processo na forma SEQ e PAR para coleta de dados, inseriria as instruções de coleta de tempos nesses processos e batizaria cada arquivo de uma forma padronizada, de forma que o sistema automático de coleta de dados pudesse reconhecer. Essa opção se estenderia, inclusive, à coleta dos intervalos de tempo dispendidos em cada comunicação (τ_{DMINT} e τ_{DMEXT}).

Ao ter completado essa tarefa, o usuário acionaria o sistema de coleta de dados, o qual se encarregaria de compilar e ligar cada módulo, mandaria executar os módulos, direcionaria a saída dos módulos com os valores de tempo para arquivos de saída, leria esses arquivos e efetuarias as subtrações necessárias sobre os valores de tempo coletados, de forma a transformar instantes de tempo em intervalos. Dessa forma, seriam gerados automaticamente os valores de τ_{SEQ}/τ_{PAR} , $\tau_{EV,SEQ}/\tau_{EV,PAR}$, τ_{ATR} e $\tau_{DMINT}/\tau_{DMEXT}$.

Ainda dentro desse conceito de otimizar a entrada dos dados, as relações de precedência para gerar a matriz de precedência MPREC poderiam deixar de ser fornecidas, pois através das tabelas de eventos dos processos, poder-se-ia gerar MPREC, analisando as comunicações que cada processo efetua. Assim, em termos de processos, o usuário deveria fornecer apenas o número de processos a serem escalados e a sequência de eventos que caracteriza cada processo, sendo que no caso de recepção ou transmissão, deveria se informar também, respectivamente, os processos transmissor e receptor e o valor de NSM de cada mensagem. Poder-se-ia inclusive implementar a validação completa dos dados de entrada, através da verificação da existência de eventuais malhas (loops), que tornem o grafo de precedência cíclico.

Outra possibilidade de aprimoramento seria a manipulação automática dos dados de saída, de forma a implantar as escalões geradas diretamente na rede de Transputers. A idéia aqui é que o usuário colocasse cada processo em um arquivo com um nome que tivesse relação com a identificação do processo. Uma vez que se soubesse qual é a distribuição dos processos nos Transputers, o próprio Programa Escalador poderia montar os m arquivos que contivessem os processos que fossem alocados a cada Transputer, colocando os processos na sequência e no regime de operação definidos pela escalção. Além de fazer isso, o Programa Escalador montaria o programa configurador, que define qual conjunto de tarefas é processado em qual Transputer da rede e que estipula como os canais de comunicação dos Transputers devem ser interligados.

Outra alternativa para melhorar o Programa Escalador seria expandir a biblioteca de algoritmos heurísticos do ESCEXT com novos algoritmos. Já estariam à disposição do usuário a matriz de precedência (MPREC), os valores de nível (VN) e de caminho crítico (VCP) dos processos, as durações dos processos (τ_{SEQ} e τ_{PAR}) e um algoritmo que calcula o número de sucessores imediatos de cada processo.

Por fim, alargando os horizontes e levando-se em conta a evolução do processamento paralelo e as perspectivas de expansão futura do multiprocessamento, seria interessante pensar em se desenvolver o modelo de outras redes de processadores, ainda que baseadas na filosofia de memória compartilhada, a fim de se obterem algoritmos confiáveis de escalação estática, que considerassem as características específicas dos processadores e a topologia da rede. Isso seria um passo adiante na metodologia convencional de escalação, que supõe processadores ideais e tarefas que se comunicam apenas no início e no fim de seu processamento.

ANEXO - FORMA DE TRATAMENTO DOS TEMPOS ENVOLVIDOS NO MODELAMENTO DE SEQ E PAR

Para modelar o processamento das tarefas nas construções SEQ e PAR, devem-se medir valores de tempo, que variam segundo a distribuição das tarefas nos processadores, o regime de operação das mesmas em cada um deles e a evolução da tarefa em termos de comunicações e temporizações que ela tenha realizado.

1) Valores de Tempo Necessários para Caracterizar as Construções SEQ e PAR

Relacionam-se, neste item, os instantes e os intervalos de tempo necessários para caracterizar as construções SEQ e PAR. Os instantes de tempo se relacionam com eventos que provocam a desescalação dos processos, conforme mencionado no item 3.3.3 ou então, que provocam interrupções no processamento da tarefa por tempo variável, de acordo com a situação; ao passo que os intervalos de tempo se referem ao tempo dispendido em comunicações ou em atrasos causados pelo temporizador.

Os eventos desescaladores ou interrompedores de tarefas, passíveis de terem seus instantes de ocorrência medidos pelo usuário são os seguintes:

- comunicação (recepção ou transmissão de mensagens);
- atraso gerado pelo temporizador; e
- término de processamento da tarefa.

Cabe lembrar que um mesmo evento tem efeito desescalador em uma construção PAR em que haja dois ou mais processos ativos e tem efeito interrompedor em uma construção SEQ ou PAR com um único processo ativo.

Tendo em vista que se assume que todo Transputer deverá ter um Processo Roteador/Distribuidor (conforme seção 3.5), isso implica que não haverá comunicação interna ou externa entre os processos sendo alocados, mas apenas comunicação interna entre eles e o Processo Roteador/Distribuidor. Portanto, deste ponto em diante, quando se mencionar comunicação, subentender-se-á comunicação interna com o processo Roteador/Distribuidor.

Os tempos necessários para caracterizar cada tarefa nas construções SEQ e PAR são os seguintes:

- instantes de ocorrência das comunicações (recepção ou transmissão) ($t_{MSG,SEQ}$ e $t_{MSG,PAR}$);
- instantes de ocorrência de instruções de atraso do temporizador ($t_{ATR,SEQ}$ e $t_{ATR,PAR}$);
- tempo de processamento da tarefa ($\tau_{PROC,SEQ}$ e $\tau_{PROC,PAR}$);
- intervalo de tempo consumido em cada comunicação (τ_{DMINT} e τ_{DMEXT}); e
- intervalo de tempo consumido em cada instrução de atraso do temporizador (τ_{ATR}).

Visto que há diferença na velocidade de processamento das tarefas nas construções SEQ e PAR (vide item 3.4.2), resulta que os instantes de ocorrência dos eventos diferem nesses dois casos, sendo então necessário se dispor de ambos os valores.

2) Tratamento dos Valores de Tempo no Programa Escalador

O item 1) deste anexo apresentou os dados necessários para se caracterizarem as construções SEQ e PAR. Entretanto, para poder operar com o Programa Escalador, esses dados devem ser convertidos pelo usuário em uma forma adequada ao Programa Escalador, que lida apenas com intervalos de tempo entre eventos e com durações de eventos, conforme mostrado na figura A.1.

onde:

$t_{EV(i-1)}$ e $t_{EV(i)}$ correspondem, respectivamente, aos instantes de ocorrência dos eventos (i-1) e i

$\tau_{EV(i)}$ corresponde à duração do evento i

$\tau_{CONSUM(i-1)}$ corresponde ao tempo consumido pelo evento (i-1)

Fig. A.1 Valores de tempo manipulados pelo Programa Escalador

Para o Programa Escalador interessam apenas dois intervalos de tempo: $\tau_{EV(i)}$ e $\tau_{CONSUM(i-1)}$, onde $\tau_{EV(i)}$ corresponde ao intervalo entre os eventos i e (i-1), descontado o tempo consumido pelo evento (i-1), e $\tau_{CONSUM(i-1)}$ equivale ao intervalo dispendido na transferência de uma mensagem ou a um atraso causado pelo temporizador.

$\tau_{EV(i)}$ é calculado pela seguinte equação:

$$\tau_{EV(i)} = t_{EV(i)} - t_{EV(i-1)} - \tau_{CONSUM(i-1)} \quad (A.1)$$

Verifica-se que τ_{EV} mede apenas os intervalos de tempo de utilização da UCP do Transputer, desprezando os intervalos dispendidos com comunicação ou temporização.

REFERÊNCIAS BIBLIOGRÁFICAS

- ADAM, T.L.; CHANDY, K.M.; DICKSON, J.R. A comparison of list schedules for parallel processing systems. **Communications of the ACM**, v.17, n.12, p.685-90, Dec. 1974.
- AHO, A.V.; HOPCROFT, J.E.; ULLMAN, J.D. **The design and analysis of computer algorithms**. Reading, Addison-Wesley, 1975.
- AHO, A.V.; HOPCROFT, J.E.; ULLMAN, J.D. **Data Structures and Algorithms**. Reading, Addison-Wesley, 1983.
- ARAKAKI, R. et al. **Fundamentos de programação C**. 2.ed. São Paulo, Livros Técnicos e Científicos, 1990.
- ATHAS, W.C.; SEITZ, C.L. Multicomputers: message-passing concurrent computers. **Computer**, v.21, n.8, p9-24, Aug. 1988.
- BALAS, E. Branch and bound/implicit enumeration. **Annals of Discrete Mathematics**, v.5, p.185-91, 1979.
- BARROSO, A.C.O.; GARCIA, C. A utilização de simuladores de tempo real e o projeto de um simulador-estimulador modular. In: Congresso Nacional de Automação Industrial - CONAI, 3., São Paulo, 1988. **Anais**. São Paulo, SUCEsu/ABCPAI, 1988. p.330-7.
- BELLMAN, R. **Dynamic programming**. Princeton, Princeton University Press, 1957.
- BELLMAN, R.E.; DREYFUS, S.E. **Applied Dynamic Programming**. Princeton, Princeton University Press, 1962.
- BERTSEKAS, D.P.; TSITSIKLIS, J.N. **Parallel and Distributed Computation**. Englewood Cliffs, Prentice-Hall, 1989. p.1-2.
- BRUNO, J.L.; DOWNEY, P.J. Probabilistic bounds on the performance of list scheduling. **SIAM Journal on Computing**, v.15, n.2, p.409-17, May 1986.
- BUSSELL, B.; FERNANDEZ, E.; LEVY, O. Optimal scheduling for homogeneous multiprocessors. In: IFIP CONGRESS 74, Stockholm, 1974. Information Processing 74. **Proceedings**. Amsterdam, North-Holland, 1974. p.286-90.
- CHEN, N.F.; LIU, C.L. On a class of scheduling algorithms for multiprocessors computing systems. **Lecture Notes in Computer Science**, v.24, p.5-20, 1975. /Apresentado a Sagamore Computer Conference, Sagamore, 1974/.
- CHU, W.W. et al. Task allocation in distributed data processing. **Computer**, v.13, n.11, p.57-69, Nov. 1980.
- COFFMAN, E.G. **Computer and job-shop scheduling theory**. New York, Wiley, 1976.
- COFFMAN, E.G.; DENNING, P.J. **Operating systems theory**. Englewood Cliffs, Prentice-Hall, 1973.

- COFFMAN, E.G.; FREDERICKSON, G.N.; LUEKER, G.S. A note on expected makespans for largest-first sequences of independent tasks on two processors. **Mathematics of Operations Research**, v.9, n.2, p.260-66, May 1984.
- COFFMAN, E.G.; GILBERT, E.N. On the expected relative performance of list scheduling. **Operations Research**, v.33, n.3, p.548-61, May/June 1985.
- COFFMAN, E.G.; GRAHAM, R.L. Optimal scheduling for two-processor systems. **Acta Informatica**, v.1, p.173-89, 1972.
- CONWAY, R.W.; MAXWELL, W.L.; MILLER, L.W. **Theory of scheduling**. Reading, Addison-Wesley, 1967.
- COOK, S.A. The complexity of theorem-proving procedures. In: ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, 3., Shaker Heights, USA, 1971. **Proceedings**. New York, ACM, 1971. p.151-8.
- DAS, P.K.; FAY, D.Q.M. Dinamically reconfigurable multi-Transputer systems. **Microprocessing and Microprogramming**, v.23, n.1-5, p.247-52, Mar. 1988.
- FERNÁNDEZ, E.B.; BUSSELL, B. Bounds on the number of processors and time for multiprocessor optimal schedules. **IEEE Transactions on Computers**, v.22, n.8, p.745-51, Aug. 1973.
- FERNÁNDEZ, E.B.; LANG, T. Computation of lower bounds for multiprocessor schedules. **IBM Journal of Research and Development**, v.19, n.2, p.435-44, Sept. 1975.
- FUJII, M.; KASAMI, T.; NINOMIYA, K. Optimal sequencing of two equivalent processors. **SIAM Journal on Applied Mathematics**, v.17, n.4, p.784-9, July 1969.
- GABOW, H.N. An almost-linear algorithm for two-processor scheduling. **Journal of the ACM**, v.29, n.3, p.766-80, July 1982.
- GAJSKI, D.D.; PEIR, J. Essential issues in multiprocessor systems. **Computer**, v.18, n.6, p.9-27, June 1985.
- GARCIA, C.; MARTUCCI, M. Proposta de arquitetura paralela para simulador de processos industriais em tempo real. In: Congresso Nacional de Automação Industrial - CONAI, 4., São Paulo, 1990. **Anais**. São Paulo, SUCEsu-SP/ABCPAI/SBI, 1990. p.278-88.
- GAREY, M.R.; GRAHAM, R.L.; JOHNSON, D.S. Performance guarantees for scheduling algorithms. **Operations Research**, v.26, n.1, p.3-21, Mar./Apr. 1978.
- GAREY, M.R.; JOHNSON, D.S. Approximation algorithms for combinatorial problems: an annotated bibliography. In: TRAUB, J.F. **Algorithms and complexity: new directions and recent results**. New York, Academic Press, 1976. p.41-52.
- GAREY, M.R.; JOHNSON, D.S. **Computers and intractability: a guide to the theory of NP-completeness**. San Francisco, W. H. Freeman, 1979.

- GENERAL ELECTRIC COMPANY. **Software Engineering handbook.** New York, McGraw-Hill, 1986.
- GONZALEZ, M.J. Deterministic processor scheduling. **ACM Computing Surveys**, v.9, n.3, p.173-204, Sept. 1977.
- GRAHAM, R.L. Bounds for certain multiprocessing anomalies. **The Bell System Technical Journal**, v.45, n.9, p.1563-81, Nov. 1966.
- GRAHAM, R.L. Bounds on multiprocessing anomalies and related packing algorithms. In: Spring Joint Computer Conference, 40, Atlantic City, NJ, 1972. Montvale, NJ, AFIPS Press, 1972, **AFIPS Conference Proceedings**. p.205-17.
- GRAHAM, R.L. Bounds on multiprocessing timing anomalies. **SIAM Journal on Applied Mathematics**, v.17, n.2, p.416-29, Mar. 1969.
- GRAHAM, R.L. The combinatorial mathematics of scheduling. **Scientific American**, v.238, n.3, Mar. 1978.
- GRAHAM, R.L. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. **Annals of Discrete Mathematics**, v.5, p.287-326, 1979.
- HAYES, J.P. **Computer architecture and organization.** 2.ed. New York, McGraw-Hill, 1988. p.572-3.
- HOROWITZ, E.; SAHNI, S. Exact and approximate algorithms for scheduling nonidentical processors. **Journal of the ACM**, v.23, n.2, p.317-27, Apr. 1976.
- HOROWITZ, E.; SAHNI, S. **Fundamentals of computer algorithms.** Rockville, Computer Science Press, 1984.
- HORVATH, E.C.; LAM, S.; RAVI, S. A level algorithm for preemptive scheduling. **Journal of the ACM**, v.24, n.1, p32-43, Jan. 1977.
- HOWARD, R.A. Dynamic programming. **Management Science**, v.13, n.4, p.B176-85, Dec. 1966.
- HU, T.C. Parallel sequencing and assembly line problems. **Operations Research**, v.9, n.6, p.841-48, Nov./Dec. 1961.
- HU, T.C. **Combinatorial algorithms.** Reading, Addison-Wesley, 1982.
- HWANG, J. et al. Scheduling precedence graphs in systems with interprocessor communications times. **SIAM Journal on Computing**, v.18, n.2, p.244-57, Apr. 1989.
- HWANG, K.; BRIGGS, F.A. **Computer architecture and parallel processing.** Singapore, McGraw-Hill, 1988.
- IBARAKI, T. Theoretical comparisons of search strategies in branch-and-bound algorithms. **International Journal of Computer and Information Sciences**, v.5, n.4, p.315-44, Dec. 1976.

- INDURCHYA, B.; STONE, H.S.; XI-CHENG, L. Optimal partitioning of randomly generated distributed programs. **IEEE Transactions on Software Engineering**, v.SE-12, n.3, p.483-95, Mar. 1986.
- INMOS LIMITED. **Occam 2 reference manual**. Hertfordshire, UK, Prentice Hall International, 1988.
- INMOS LIMITED. **Transputer instruction set - a compiler writer's guide**. Hertfordshire, UK, Prentice Hall International, 1988.
- INMOS LIMITED. **Transputer reference manual**. Hertfordshire, UK, Prentice Hall International, 1988.
- INMOS LIMITED. **The Transputer applications notebook - architecture and software**. Almondsbury, UK, Inmos Limited, 1989.
- INMOS LIMITED. **The Transputer applications notebook - systems and performance**. Almondsbury, UK, Inmos Limited, 1989.
- INMOS LIMITED. **The Transputer databook**. 2.ed. Almondsbury, UK, Inmos Limited, 1989.
- JENNY, C.J. Process partitioning in distributed systems. In: NTC'77 CONFERENCE RECORD, Los Angeles, 1977. **Proceedings**. New York, IEEE, 1977. v.2, p.31:1-1 a 31:1-10.
- JOHNSON, D.S. The NP-completeness column: an ongoing guide. **Journal of Algorithms**, v.4, n.2, p.189-203, June 1983.
- JOHNSON, S.M. Optimal two and three-stage production schedules with set-up time included. **Naval Research Logistics Quarterly**, New York, v.1, n.1, p.61-8, Mar. 1954. Reimpresso em: MUTH, J.F.; THOMPSON, G.L. **Industrial Scheduling**. Englewood Cliffs, Prentice-Hall, 1963. cap.2.
- KARP, R.M. Reducibility among combinatorial problems. In: MILLER, R.E.; THATCHER, J.W. **Complexity of Computer Computations**. New York, Plenum Press, 1972. p.85-103.
- KARP, R.M. On the computational complexity of combinatorial problems. **Networks**, v.5, n.1, p.45-68, Jan. 1975.
- KARP, R.M. Combinatorics, complexity and randomness. **Communications of the ACM**, v.29, n.2, p.98-111, Feb. 1986.
- KARP, R.M.; HELD, M. Finite-space processes and dynamic programming. **SIAM Journal on Applied Mathematics**, v.15, n.3, p.693-718, May 1967.
- KASAHARA, H.; NARITA, S. Parallel processing for real-time control and simulation of DCCS. In: IFAC WORKSHOP ON DISTRIBUTED COMPUTER CONTROL SYSTEMS, 4, Tallinn, U.S.S.R., 1982. **Proceedings**. Oxford, England, Pergamon Press, 1983. p.103-15.
- KASAHARA, H.; NARITA, S. Load distribution among real-time control computers connected via communication media. In: IFAC WORLD CONGRESS, 9, Budapest, 1984. **Proceedings**. Oxford, Pergamon Press, 1985. v.5, p.2695-700.

- KASAHARA, H.; NARITA, S. Practical multiprocessor scheduling algorithms for efficient parallel processing. **IEEE Transactions on Computers**, v.33, n.11, p.1023-9, Nov. 1984.
- KEMAL, E. Heuristic models of task assignment scheduling in distributed systems. **Computer**, v.16, n.6, p.50-6, June 1982.
- KOHLER, W.H. A preliminary evaluation on the critical path method for scheduling tasks on multiprocessor systems. **IEEE Transactions on Computers**, v.24, n.12, p.1235-8, Dec. 1975.
- KOHLER, W.H.; STEIGLITZ, K. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. **Journal of the ACM**, v.21, n.1, p.140-56, Jan. 1974.
- KOHLER, W.H.; STEIGLITZ, K. Exact, approximate and guaranteed accuracy algorithms for the flow-shop problem $n/2/F/=F$. **Journal of the ACM**, v.22, n.1, p.106-14, Jan. 1975.
- LAM, S.; SETHI, R. Worst case analysis of two scheduling algorithms. **SIAM Journal on Computing**, v.6, n.3, p.518-36, Sept. 1977.
- LAWLER, E.L.; LENSTRA, J.K. Machine scheduling with precedence constraints. NATO Advanced Study Institutes series, series C - Mathematical and Physical Sciences, v.83 - Ordered Sets, p.655-75, 1981. Editado por RIVAL, I., D. Reidel Publishing Company. /Apresentado a NATO Advanced Study Institute, Banff, Canada, Sept. 1981/.
- LENSTRA, J.K.; RINNOOY KAN, A.H.G. Complexity of scheduling under precedence constraints. **Operations Research**, v.26, n.1, p.22-35, Jan./Feb. 1978.
- LENSTRA, J.K.; RINNOOY KAN, A.H.G. Sequencing and scheduling. In: O'HEIGEARTAIGH, M.; LENSTRA, J.K.; RINNOOY KAN, A.H.G. **Combinatorial optimization - annotated bibliographies**. Chicester, Hohn Wiley & Sons, 1985. p.164-89.
- LENSTRA, J.K.; RINNOOY KAN, A.H.G.; BRUCKER, P. Complexity of machine scheduling problems. **Annals of Discrete Mathematics**, v.5, p.343-62, 1977.
- LEPPALA, K. Utilization of parallelism in Transputer-based real-time control systems. **Microprocessing and Microprogramming**, v.21, n.1-5, p.629-36, Aug. 1987.
- LEWIS, H.R.; PAPADIMITRIOU, C.H. The efficiency of algorithms. **Scientific American**, v.238, n.1, p.96-109, Jan. 1978.
- MA, P.R.; LEE, E.Y.S.; TSUCHIYA, M. A task allocation model for distributed computing systems. **IEEE Transactions on Computers**, v.C-31, n.1, p.41-7, Jan. 1982.
- MANACHER, G.K. Production and stabilization of real-time task schedules. **Journal of the ACM**, v.14, n.3, p.439-65, July 1967.
- MARTIN, D.F.; ESTRIN, G. Experiments on models of computations and systems. **IEEE Transactions on Electronic Computers**, v.EC-16, n.1, p.59-69, Feb. 1967.

- MCCREARY, C.; GILL, H. Automatic determination of grains size for efficient parallel processing. **Communications of the ACM**, v.32, n.9, p.1073-8, Sept. 1989.
- McNAUGHTON, R. Scheduling with deadlines and loss functions. **Management Science**, v.6, n.1, p.1-12, Oct. 1959.
- MITTEN, L.G. Branch-and-bound methods: general formulation and properties. **Operations Research**, v.18, n.1, p.24-34, Jan./Feb. 1970.
- MUNTZ, R.R.; COFFMAN, E.G. Optimal preemptive scheduling on two-processor systems. **IEEE Transactions on Computers**, v.C-18, n.11, p.1014-20, Nov. 1969.
- MUNTZ, R.R.; COFFMAN, E.G. Preemptive scheduling of real-time tasks on multiprocessor systems. **Journal of the ACM**, v.17, n.2, p.324-338, Apr. 1970.
- MURRAY, K.A.; WELLINGS, A.J. Issues in the design and implementation of a distributed operating system for a network of Transputers. **Microprocessing and Microprogramming**, v.24, n.1-5, p.169-77, Aug. 1988.
- NI, L.M.; WU, C.E. Design tradeoffs for process scheduling in shared memory multiprocessor systems. **IEEE Transactions on Software Engineering**, v.15, n.3, p.327-34, Mar. 1989.
- PAPADIMITRIOU, C.H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. Englewood Cliffs, Prentice-Hall, 1982.
- POLYCHRONOPOULOS, C.D. Toward auto-scheduling compilers. **The Journal of Supercomputing**, v.2, n.3, p.297-329, Nov. 1988.
- POLYCHRONOPOULOS, C.D.; KUCK, D.J. Guided self-scheduling: a practical scheduling scheme for parallel supercomputers. **IEEE Transactions on Computers**, v.C-36, n.12, p.1425-39, Dec. 1987.
- POUNTAIN, D. Configuring parallel programs - part 1. **BYTE**, v.14, n.12, p.349-52, Dec. 1989.
- POUNTAIN, D. Configuring parallel programs - part 2. **BYTE**, v.15, n.1, p.327-34, Jan. 1990.
- RAMAMOORTHY, C.V.; CHANDY, K.M.; GONZALEZ, M.J. Optimal scheduling strategies in a multiprocessor system. **IEEE Transactions on Computers**, v.21, n.2, p.137-46, Feb. 1972.
- RINNOOY KAN, A.H.G. Scheduling. **Annals of Discrete Mathematics**, v.5, p.423-53, 1979.
- ROTHKOPF, M.H. Scheduling independent tasks on parallel processors. **Management Science**, v.12, n.5, p.437-47, Jan. 1966.
- SAHNI, S. Algorithms for scheduling independent tasks. **Journal of the ACM**, v.23, n.1, p.116-27, Jan. 1976.
- SAHNI, S. General techniques for combinatorial approximation. **Operations Research**, v.25, n.6, p.920-36, Nov./Dec. 1977.

- SAHNI, S.; HOROWITZ, E. Combinatorial problems. **Operations Research**, v.26, n.5, p.718-59, Sept./Oct. 1978.
- SCHWARTZ, E.S. An automatic sequencing procedure with application to parallel programming. **Journal of the ACM**, v.8, n.4, p.513-37, Oct. 1961.
- SCHWARTZ, E.S. A heuristic procedure for parallel sequencing with choice of machines. **Management Science**, v.10, n.4, p.767-77, July 1964.
- SHIRAZI, B.; WANG, M. Heuristic functions for static task allocation. **Microprocessing and Microprogramming**, v.26, n.3, p.187-94, Oct. 1989.
- STONE, H.S. Multiprocessor scheduling with the aid of network flow algorithms. **IEEE Transactions on Software Engineering**, v.3, n.1, p.85-93, Jan. 1977.
- STONE, H.S.; BOKHARI, S.H. Control of distributed processes. **Computer**, v.11, n.7, p.97-106, July 1978.
- STROUSTRUP, B. **The C++ Programming Language**. Reading, Addison-Wesley, 1987.
- TAYLOR, R.; WILSON, P. Process-oriented language meets demands of distributed processing. **Electronics**, v.55, n.24, p.89-95, Nov. 1982.
- TERRANO, A.E.; DUNN, S.M.; PETER, J.E. Using an architectural knowledge base to generate code for parallel computers. **Communications of the ACM**, v.32, n. 9, p.1065-72, Sept. 1989.
- ULLMAN, J.D. Polynomial complete scheduling problems. **ACM Operating Systems Review**, v.7, n.4, p.96-101, Oct. 1973. /Apresentado a 4. Symposium on Operating Systems Principles, Yorktown Heights, NY, 1973/.
- ULLMAN, J.D. NP-complete scheduling problems. **Journal of Computer and System Sciences**, v.10, n.3, p.384-393, June 1975.
- VRANES, S. A heuristic algorithm for real-time application allocation to multimicrocomputer. **Microprocessing and Microprogramming**, v.25, n.1-5, p.347-52, Jan. 1989.
- WIRTH, N. **Algoritmos e Estruturas de Dados**. Trad. de Cheng Mei Lee. Rio de Janeiro, Prentice-Hall do Brasil, 1989.
- YANG, C.C. Fast algorithms for bounding the performance of multiprocessor systems. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, Waldonwoods, MI, 1976. **Proceedings**. New York, IEEE, 1976. p.73-82.