SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 29.11.1999

Assinatura:

Um Ambiente de Desenvolvimento para Aplicações Paralelas *Análise de Requisitos e Definição*¹

Mario Augusto Andreta Carvalho

Orientador: *Prof. Dr. Onofre Trindade Junior*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área: Ciências de Computação e Matemática Computacional.

São Carlos
Novembro/1999

¹Trabalho realizado com apoio financeiro da CAPES

Resumo

Neste trabalho é apresentado o estudo de desenvolvimento de um ambiente integrado para programação paralela. O ambiente contém as ferramentas apropriadas para o desenvolvimento de novos programas paralelos, para a paralelização de programas sequenciais, para a análise e otimização de desempenho e a depuração de programas paralelos. O trabalho cobre as fases de análise de requisitos, especificação e projeto do ambiente proposto.

Através de uma pesquisa extensa sobre as ferramentas para programação paralela e da análise de algumas formas de classificação existentes, foi proposta uma nova classificação que possibilita enquadrar um grande número de ferramentas que fizeram parte da pesquisa. Com base nessa classificação foi desenvolvido um modelo gráfico do sistema, determinando-se os módulos que se fazem necessários para cobrir todas as fases de desenvolvimento de programas paralelos. Também é apresentado neste trabalho três opções de implementação do sistema baseadas em ferramentas existentes que podem fazer parte do projeto proposto com ou sem alteração do seu código fonte.

Abstract

This work presents a development study of an integrated environment for parallel programming. The environment has appropriate tools for the development of new parallel programs, the parallelization of sequential programs, analysis of performance, tuning and debugging of parallel programs. This work covers the requirement analysis, specification and design of the environment.

Through an extensive research on parallel programming and the analysis of the classification schemes for parallel programming tools found in the literature, is presented a new classification scheme encompassing a bigger number of tools. Based on this classification, a graphical model of the system is introduced with the specification of the modules needed to cover all development phases of parallel programs. Three options for the implementation of the proposed system are also presented. Some tools were selected to be part of the proposed system, with or without modification of their source code.

Aos meus pais, e a minha avó.

Agradecimentos

À DEUS.

Sumário

1. INTRODUÇÃO

- 1.1 - Aspectos Gerais, 1
- 1.2 - A Estrutura do Sistema Proposto, 2
- 1.2 - Estrutura do Texto, 2

2. REVISÃO DA LITERATURA

- 2.1 - Considerações Iniciais, 4
- 2.2 - Processamento Paralelo e Distribuído, 4
 - 2.2.1 - Arquiteturas para Processamento Paralelo, 4
 - 2.2.2 - Formas de Paralelização de Programas, 9
 - 2.2.3 - Linguagens de Programação, 10
 - 2.2.4 - Programação Baseada em Variáveis Compartilhadas, 12
 - 2.2.5 - Programação Baseada em Passagem de Mensagens, 13
 - 2.2.6 - Programação Baseada no Paralelismo de Dados, 14
 - 2.2.7 - Métricas e Medidas de Desempenho, 14
- 2.3 - Ferramentas para Programação Paralela, 15
- 2.4 - Considerações Finais, 47

3. CLASSIFICAÇÃO DAS FERRAMENTAS

- 3.1 Considerações Iniciais, 48
- 3.2 Proposta de Classificação para as Ferramentas, 48
 - A - Ferramentas de Desenvolvimento de Programas Paralelos, 49
 - A.1 – Linguagens, 49
 - A.2 – Bibliotecas, 50
 - A.3 - Ambientes de Desenvolvimento, 50
 - B - Ferramentas de Paralelização, 51
 - B.1 – Automáticas, 51
 - B.1.1 - Compiladores, 51
 - B.1.2 - Pré-processadores, 52
 - B.2 – Interativas, 52
 - C - Ferramentas de Análise de Desempenho e Depuração, 52
 - C.1 – Depuração, 53
 - C.2 - Análise de Desempenho, 53
 - D - Ambientes Integrados, 54
- 2.3 Considerações Finais, 59

4. ESPECIFICAÇÃO E PROJETO DO SISTEMA PROPOSTO

- 4.1 - Considerações Iniciais, 60
- 4.2 - Objetivos Gerais do Sistema, 60
- 4.3 - Modelo de Software do Sistema, 61
- 4.4 – Especificação do Sistema, 62
 - 4.4.1 - Requisitos Funcionais do Sistema, 62
 - 4.4.2 - Requisitos não Funcionais do Sistema, 63
 - 4.4.3 - Requisitos de Integração do Sistema, 64
- 4.5 - Projeto do Sistema, 65
 - 4.5.1 – Diagrama de Contexto do Sistema, 65
 - 4.5.2 - Descrição dos Módulos do Sistema, 69
- 4.6 - Considerações Finais, 79

5. AVALIAÇÃO E UTILIZAÇÃO DAS FERRAMENTAS EXISTENTES

- 5.1 - Considerações Iniciais, 80
- 5.2 - Descrição das Ferramentas Seleccionadas, 80
 - 5.2.1 - PVM, 82
 - 5.2.2 - MPI, 83
 - 5.2.3 - P2D, 85
 - 5.2.4 - PANORAMA, 88
 - 5.2.5 - PABLO, 91
 - 5.2.6 - HENCE, 94
- 5.3 - Opções de Integração, 98
- 5.4 - Considerações Finais, 101

6. CONCLUSÕES

- 6.1. Considerações Iniciais, 102
- 6.2. Contribuições Deste Trabalho, 103
- 6.3. Trabalhos Futuros, 103
- 6.4. Considerações Finais, 104

REFERÊNCIAS E BIBLIOGRAFIA, 105

Lista de Figuras

- Figura 2.1: Arquitetura de um Computador SIMD, 6
- Figura 2.2: Arquitetura de um Computador MIMD, 7
- Figura 2.3: Máquina MIMD com Memória Compartilhada, 7
- Figura 2.4: Máquina MIMD com Memória Distribuída, 8
- Figura 4.1: Diagrama de Contexto do Ambiente Integrado, 61
- Figura 4.2: Primeiro esboço do sistema proposto, 65
- Figura 4.3: Segundo esboço do sistema proposto, 66
- Figura 4.4: Projeto Final do Sistema Proposto, 68
- Figura 4.5: Modelador de Elementos, 69
- Figura 4.6: Diagramador de Comportamento, 70
- Figura 4.7: Particionador de Código/Objeto, 71
- Figura 4.8: Codificador, 72
- Figura 4.9: Reformatador de Código Sequencial, 72
- Figura 4.10: Analisador de Código Sequencial, 73
- Figura 4.11: Instrumentador de Paralelização, 74
- Figura 4.12: Instrumentador Interativo, 74
- Figura 4.13: Paralelizador Automático, 75
- Figura 4.14: Compilador, 76
- Figura 4.15: Instrumentador de Desempenho, 76
- Figura 4.16: Executor, 77
- Figura 4.17: Analisador em Tempo de Execução, 77
- Figura 4.18: Analisador *Post-Mortem*, 78
- Figura 4.19: Depurador, 79
- Figura 5.1: P2D - Diagrama de Fluxo de Dados do Sistema – Nível 1, 86
- Figura 5.2: Área de atuação da ferramenta P2D do sistema proposto, 87
- Figura 5.3: PANORAMA - Diagrama de Fluxo de Dados, 89
- Figura 5.4: Área de atuação da ferramenta PANORAMA no sistema proposto, 90
- Figura 5.5: PABLO - Diagrama de Fluxo de Dados do Sistema, 92
- Figura 5.6: Área de atuação da ferramenta PABLO no sistema proposto, 93
- Figura 5.7: HeNCE – Diagrama de Fluxo de Dados do Sistema, 95
- Figura 5.8: Área de atuação da ferramenta HeNCE no sistema proposto, 96
- Figura 5.9: Áreas cobertas pelas ferramentas, 97
- Figura 5.10: Integração por acesso a ferramentas comuns, 98
- Figura 5.11: Integração de Dados, 99
- Figura 5.12: Integração Plena, 100

Lista de Tabelas

Tabela 2.1: Características das Ferramentas para Programação Paralela, 38

Tabela 3.1: Ferramentas para Desenvolvimento de Programas Paralelos, 54

Tabela 3.2: Ferramentas para Paralelização de Programas Sequenciais, 57

Tabela 3.3: Ferramentas para Depuração e Análise de Desempenho, 57

Tabela 3.4: Ambientes Integrados, 58

Tabela 3.5: Outras Ferramentas para Processamento Paralelo, 59

Tabela 5.1: Comparativo entre os principais módulos das ferramentas, 101

Capítulo 1

1 - INTRODUÇÃO

1.1 - Aspectos Gerais

A demanda crescente por maior poder de processamento tem motivado a evolução dos computadores, viabilizando aplicações que envolvem processamento intenso e grande volume de dados.

O aumento de desempenho dos sistemas computacionais pode ser obtido através do aumento da velocidade do relógio. Esta alternativa envolve o desenvolvimento da tecnologia de confecção de circuitos integrados, trazendo entretanto problemas, tal como, o aumento de temperatura. O aumento de desempenho também pode ser obtido com melhorias na arquitetura. Essa alternativa motivou o surgimento dos processadores RISC, vetoriais e superescalares. Melhoramentos no acesso à memória, por exemplo, através da exploração da hierarquia de memória, utilizando registradores, memória *cache* e memória principal, podem resultar em um aumento de desempenho significativo. Uma última alternativa para o aumento de desempenho é a utilização de vários elementos processadores, distribuindo entre esses, o processamento do programa. Essa alternativa é conhecida por processamento paralelo.

A implementação de algoritmos em um computador paralelo exige recursos de programação para expressar o paralelismo e incluir mecanismos para a sincronização e comunicação entre processos. Isso ocasiona um aumento significativo na complexidade de desenvolvimento dos programas paralelos, tornando-se cada vez mais necessária a utilização de ferramentas que auxiliem esse tipo de programação. Fica claro, a partir da análise das ferramentas existentes para a programação paralela, a necessidade de uma ferramenta que atenda os requisitos de programação de uma maneira completa e integrada.

1.2 - O Ambiente do Sistema Proposto

O sistema proposto neste trabalho procura prover o usuário com um ambiente integrado completo com ferramentas de auxílio no desenvolvimento de aplicações paralelas.

O projeto é baseado em uma extensa pesquisa das diversas ferramentas para programação paralela já existentes, proporcionando uma visão clara e ampla das principais funções dos diversos tipos de ferramentas. A diversidade dessas ferramentas e suas finalidades culminaram na proposição de uma nova proposta de classificação para as ferramentas.

O ambiente do sistema proposto é caracterizado pela existência de 3 grandes módulos que por sua vez abrangem uma série de funções. O primeiro módulo é responsável pelo desenvolvimento de novas aplicações paralelas e contém ferramentas de análise, projeto, codificação e ferramentas para o particionamento de código. O módulo de paralelização é responsável por conter ferramentas automáticas e interativas para paralelização de programas sequenciais já existentes. O terceiro módulo do sistema é responsável pela análise de desempenho, otimização e detecção de erros nos programas paralelos gerados.

A realização do sistema proposto pode ser feita através da utilização de algumas ferramentas para programação paralela disponíveis, reduzindo bastante o seu tempo de desenvolvimento. A partir da especificação do modelo lógico do sistema proposto, foram selecionadas algumas ferramentas existentes que podem ser utilizadas no projeto com ou sem modificações do seu código fonte.

1.3 - Estrutura do Texto

O presente texto é composto por 6 capítulos. O capítulo 2 fornece uma revisão da bibliografia que norteia o desenvolvimento do trabalho, apresentando os temas centrais do projeto, como o processamento paralelo e distribuído, os modelos e as linguagens de programação paralela. Nesse capítulo também é dada uma breve descrição das ferramentas que fizeram parte da pesquisa realizada.

O capítulo 3 apresenta uma nova proposta de classificação para as ferramentas de programação paralela. Essa classificação procura ser mais abrangente e mais precisa que as classificações encontradas até aqui nas referências consultadas. O capítulo 4 apresenta a especificação e projeto do sistema proposto. Uma especificação inicial descreve os objetivos gerais bem como os requisitos funcionais e não funcionais do sistema. O projeto apresenta um modelo gráfico contendo os principais módulos que se fazem necessários para o desenvolvimento de um ambiente realmente integrado para o desenvolvimento, paralelização, análise de desempenho e depuração de aplicações paralelas.

O capítulo 5 apresenta algumas ferramentas selecionadas que podem fazer parte do projeto com ou sem alteração no seu código fonte. Três propostas de implementação do sistema com níveis crescentes de funcionalidade e dificuldade de realização são apresentadas nesse capítulo. Finalmente o capítulo 6 apresenta as conclusões e contribuições desse trabalho, fazendo-se também sugestões para trabalhos futuros que podem dar continuidade ao trabalho até aqui realizado.

Capítulo 2

2 - REVISÃO DA LITERATURA

2.1 - Considerações Iniciais

Para atingir as metas propostas neste trabalho, é importante uma análise dos conceitos e das ferramentas para programação paralela atualmente existentes. Neste sentido, são apresentados na Seção 2.2 os conceitos envolvidos no processamento paralelo e distribuído como as arquiteturas paralelas, as formas de paralelização de programas e os modelos e linguagens para programação paralela. Na Seção 2.3 é apresentada uma breve descrição das ferramentas que fizeram parte de uma extensa pesquisa sobre as ferramentas existentes para programação paralela. Ficando na Seção 2.4 as considerações finais sobre o presente Capítulo.

2.2 - Processamento Paralelo e Distribuído

A busca de alto desempenho, visando atender a demanda crescente de processamento, motivou o surgimento de vários modelos computacionais e arquiteturas paralelas. Segundo Duncan [133] arquiteturas paralelas são caracterizadas pela presença de múltiplos processadores que cooperam entre si.

2.2.1 - Arquiteturas para Processamento Paralelo

Qualquer sistema computacional incorpora um certo paralelismo; dados podem ser escritos em um *buffer* em paralelo com outro processamento qualquer, ou uma operação de ponto-flutuante pode ser executada em paralelo com um ciclo de busca de dados.

Para efeito de comparação das diferentes arquiteturas para processamento paralelo é importante que se possa agrupá-las em classes. Uma das classificações mais conhecidas, devida a Flynn [100], não consegue descrever algumas arquiteturas mais atuais. Baseada na classificação de Flynn, a taxonomia de Duncan segue as seguintes diretrizes:

- Excluir máquinas de um único processador que apresentam baixo nível de paralelismo como aquelas que possuem somente *pipelining* de instruções, múltiplas unidades funcionais na CPU ou processadores gráficos e de ponto-flutuante.
- Manter a essência da taxonomia de Flynn baseada na quantidade de fluxos de dados e instruções.
- Estender a taxonomia de Flynn para incorporar arquiteturas que são claramente paralelas, mas que não se enquadram no esquema original.

Seguindo essas diretrizes, qualquer máquina é aceita como sendo uma arquitetura paralela desde que se encaixe em uma das categorias apresentadas a seguir.

2.2.1.1 - Arquiteturas Síncronas

Arquiteturas paralelas síncronas são aquelas que se caracterizam por apresentar um único relógio global e unidades de controle central que podem executar qualquer uma das *threads* correntes. A maioria dessas arquiteturas são específicas para determinadas aplicações, não podendo servir como base para um computador paralelo de propósito geral.

Elas podem ser:

- Processadores Pipeline Vetoriais

São máquinas projetadas para explorar o paralelismo temporal envolvido na execução de operações sobre vetores e matrizes.

- SIMD (*Single Instruction Multiple Data*)

Uma unidade de controle central envia instruções para muitas unidades de processamento que agem sobre conjuntos paralelos de dados. Como exemplo tem-se a

Connection Machine [163] com 65536 processadores de um bit, todos operando em *lockstep*. O diagrama básico de uma máquina SIMD pode ser visto na figura 2.1.

Figura 2.1: Arquitetura de um computador SIMD

- Arranjos Sistólicos

Em máquinas do tipo Arranjo Sistólico [164] dados são bombeados ritmicamente através de uma matriz de processadores que executam as transformações desejadas e depois passam os resultados aos próximos processadores.

2.2.1.2 - Arquiteturas MIMD (Multiple Instruction Multiple Data)

Máquinas MIMD são compostas por elementos de processamento assíncronos que atuam de forma autônoma (Figura 2.2).

Figura 2.2: Arquitetura de um computador MIMD

A forma com que as informações são comunicadas entre os processadores dá origem às subclasses MIMD, que são:

- Memória Compartilhada

Essas máquinas são caracterizadas pelo fato de que os elementos de processamento têm acesso a uma região de memória comum (Figura 2.3), e são normalmente conhecidas como multiprocessadores. Tornar a memória global visível a todos os elementos de processamento é a forma mais natural de estender a arquitetura de von Neumann para processamento paralelo. Contudo, a taxa de comunicação de dados com a memória é restrita, o que torna impraticável a construção de máquinas maciçamente paralelas do tipo memória compartilhada. Atualmente muitas pesquisas investigam o uso de memória compartilhada através do mecanismo de troca de mensagens em arquiteturas de memória distribuída.

Figura 2.3: Máquina MIMD com Memória Compartilhada

- Memória Distribuída

Arquiteturas MIMD do tipo memória distribuída (*distributed memory*), compartilham informações através da troca explícita de mensagens entre os processadores de uma rede (Figura 2.4), e são normalmente conhecidas como multicomputadores.

Estas máquinas são a plataforma mais prática para processamento paralelo de propósito geral, diante das facilidades de implementação, baixo custo e elevada escalabilidade. Contudo, é essencial tornar transparente ao usuário os aspectos de hardware da arquitetura, devido às constantes mudanças das tecnologias utilizadas [165].

Um multicomputador pode ter os seus nós contidos em uma mesma placa, como os multicomputadores compostos por uma rede de *transputers*, ou em placas ligadas por barramentos/chaveadores contidas em um único gabinete. Uma terceira forma, bastante interessante sob vários aspectos, é a implementação de multicomputadores como redes locais de estações de trabalho, comumente chamadas de aglomerados. Esta última é a forma mais barata e comum de implementar um multicomputador e diversas linguagens e bibliotecas de programação distribuída têm sido desenvolvidas para tais sistemas.

Figura 2.4: Máquina MIMD com Memória Distribuída

2.2.1.3 - Paradigmas Baseados na Arquitetura MIMD

Nessa classe se enquadram outras arquiteturas que seguem o paradigma MIMD mas que apresentam diferenças de arquitetura fundamentais, como é o caso das máquinas híbridas MIMD/SIMD [166], máquinas de fluxo de dados [167] e arquiteturas *wavefront-vetor*.

2.2.2 - Formas de Paralelização de Programas

Dada a diversidade de arquiteturas paralelas, foram propostos novos compiladores e linguagens, além de bibliotecas de programação especiais, visando gerenciar o paralelismo.

O paralelismo em um programa pode ser explorado implicitamente através de compiladores paralelizadores, como o CRAY/fpp [117, 118] que paraleliza automaticamente programas escritos em Fortran.

A experiência tem mostrado que a programação paralela é significativamente mais difícil que a programação seqüencial, principalmente porque envolve a necessidade de sincronização entre tarefas e a análise de dependência de dados. A utilização de um sistema paralelizador minimiza essas dificuldades e permite também o reaproveitamento de programas seqüenciais já implementados.

Por outro lado, a programação paralela, expressando explicitamente o paralelismo, permite que fontes de paralelismo não passíveis de detecção por um sistema paralelizador possam ser exploradas.

O paralelismo pode ser expresso pelo usuário em um programa através de:

- comandos específicos de uma linguagem de programação paralela;
- chamadas a rotinas de uma biblioteca que gerencia os recursos de paralelismo da máquina;
- diretivas do compilador.

Entre essas formas, a utilização de uma linguagem de programação paralela é a que oferece um ambiente mais adequado de programação.

Bibliotecas de comunicação têm sido comumente utilizadas na implementação de programas paralelos. Como exemplo, pode-se citar as bibliotecas oferecidas pelo sistema P4 [48, 49] e outras voltadas para várias plataformas, como a biblioteca do sistema PVM (*Parallel Virtual Machine*) [31, 32].

2.2.3 - Linguagens de Programação

As linguagens podem ser classificadas em 2 tipos básicos: as linguagens imperativas e as linguagens declarativas.

As linguagens imperativas são caracterizadas por modificar um estado implícito do programa através de comandos. Estão incluídas nessa classe: as linguagens procedurais e orientadas a objetos.

As linguagens declarativas são caracterizadas por uma programação realizada através de expressões. Dentro dessa classe estão incluídas as linguagens funcionais e lógicas.

2.2.3.1 - Linguagens procedurais

As linguagens procedurais são caracterizadas por subdividir o programa em subprogramas também referidos na literatura como rotinas, sub-rotinas ou procedimentos. As linguagens C e Pascal são exemplos dessa classe.

Existe uma dependência de dados entre dois comandos quando um deles altera um dado que é lido ou escrito pelo outro. Nesse caso os dois comandos não podem ser executados simultaneamente. Devido os efeitos colaterais que podem existir, como a alteração de dados por um procedimento que são lidos por outros procedimentos, a análise de dependência de dados torna-se uma tarefa complexa.

Na programação através de uma linguagem que permite o usuário expressar paralelismo, este deve efetuar uma análise de dependência de dados. No caso de existir uma dependência de dados, mecanismos de exclusão mútua para acesso devem ser utilizados.

Algumas linguagens procedurais são largamente utilizadas e é importante que haja extensões para permitir a programação paralela nas mais conhecidas, como as linguagens C e Fortran. Foram propostas diversas extensões da linguagem C, entre as quais podem ser citadas: Jade [58, 59] e CPAR [30], todas descritas neste trabalho. Extensões da linguagem Fortran incluem Fortran M [39] e HPF (High Performance Fortran) [29].

2.2.3.2 - Linguagens orientadas a objetos

O paradigma de programação com orientação a objetos, onde cada objeto encapsula um conjunto de informações e métodos, e se comunica com outros objetos através de passagem de mensagens, torna esse paradigma atraente para a programação paralela.

Não existe dependência de dados entre objetos e eles podem ser executados em paralelo sem restrição.

A execução paralela de métodos do mesmo objeto pode ser explorada. Entretanto, neste caso acessos a um mesmo dado podem ocorrer e deve ser efetuada uma análise de dependência de dados entre os métodos.

Várias linguagens para arquiteturas paralelas com memória compartilhada ou distribuída foram propostas. Podem ser citadas COOL [66, 67] e Mentat [64, 65].

2.2.3.3 - Linguagens funcionais

Nos programas escritos em linguagens funcionais as funções comportam-se como funções matemáticas: o resultado obtido depende apenas de seus argumentos. Não são gerados efeitos colaterais, pois não é permitido que uma função interaja na execução de outra função através de variáveis globais ou do uso de ponteiros.

Esta ausência de efeitos colaterais é uma característica relevante para a programação paralela, pois não existe dependência entre as funções, possibilitando sua execução em paralelo. Entretanto, quando uma função utiliza o resultado retornado por

uma outra função a primeira é obrigada a esperá-lo, prejudicando o desempenho do programa.

Embora possa existir uma grande quantidade de possibilidades para exploração do paralelismo, algumas apresentam granulosidades muito finas. Por causa do custo da implementação do paralelismo, o desempenho do programa normalmente cresce com a granulosidade. Para não permitir a ocorrência de paralelismo com granulosidades muito finas, é necessário dispor de uma forma para indicar as expressões que serão processadas em paralelo.

2.2.3.4 - Linguagens lógicas

As linguagens lógicas seguem um paradigma de programação onde os programas são expressos como regras lógicas. Uma propriedade das linguagens de programação lógicas, presente na sua semântica, é a independência da ordem em que são processadas as diferentes operações durante a execução do programa. Esta independência permite que essas operações sejam executadas paralelamente.

Algumas propostas de linguagens lógicas para processamento paralelo foram apresentadas, explorando implicitamente o paralelismo tipo AND e o paralelismo tipo OR .

2.2.4 - Programação Baseada em Variáveis Compartilhadas

Nos multiprocessadores, todos os processadores podem ler e escrever dados sobre uma área de memória comum. Nesses computadores, a memória é utilizada para armazenar variáveis que são compartilhadas entre os processos de um programa paralelo. Diversos mecanismos são utilizados para resolver conflitos no acesso simultâneo de posições da memória compartilhada. Entre esses mecanismos pode-se citar semáforos, monitores, eventos, barreiras, etc [169].

Em um programa implementado em uma linguagem procedural, podem ser identificadas as seguintes fontes de paralelismo:

- Sub-rotinas: módulos independentes de computação que podem ser executados paralelamente. Tais módulos podem ser especificados como macrotarefas. Tem-se aqui um paralelismo de granulosidade grossa. Iterações de laços independentes podem ser executadas paralelamente. Dependendo do processamento envolvido em cada iteração tem-se um paralelismo de granulosidade grossa ou média. Blocos podem ser executados paralelamente se não existem dependências entre blocos. Um bloco básico é uma sequência de comandos que não contém desvios. Conforme o processamento envolvido tem-se um paralelismo de granulosidade grossa ou média.
- Comandos ou operações: comandos ou operações podem ser processadas simultaneamente, resultando em um paralelismo de granulosidade fina.
- Instruções: a execução paralela de múltiplas instruções é usualmente feita dentro de cada processador através de múltiplas unidades funcionais *pipelining*. Tem-se aqui um paralelismo de granulosidade fina.

2.2.5 - Programação Baseada em Passagem de Mensagens

Para que uma aplicação (programa) possa ser executada por um computador paralelo com memória distribuída é necessário distribuir trechos de seu código entre os processadores. No modelo chamado SPMD (*Single Program Multiple Data*), um único programa é utilizado em todos os processadores, embora cada processador possa executar uma parte diferente do código desse programa. Algumas bibliotecas de programação utilizam o modelo SPMD, enquanto que outras permitem que trechos heterogêneos de código sejam distribuídos pelos processadores.

As subunidades de um programa comumente recebem denominações como tarefas ou processos. Ambos os termos serão utilizados indistintamente neste trabalho. Tais subunidades fazem parte de um mesmo programa, e portanto devem trocar dados, resultados e informações de sincronização.

Nos multicomputadores, os processadores encontram-se distribuídos fisicamente e somente podem ter acesso a sua própria memória local. O modelo mais elementar de programação em multicomputadores é aquele no qual os processos que compõem um programa paralelo são distribuídos pelos nós da rede, e trocam mensagens entre si.

2.2.6 - Programação Baseada no Paralelismo de Dados

O paradigma de programação paralela baseada no paralelismo de dados representa uma forma de exploração de operações simultâneas sobre grandes conjuntos de dados, ao invés da especificação de várias tarefas de controle em paralelo. Este estilo de programação é adotado em máquinas contendo dezenas a milhares de processadores.

A concorrência não é especificada na forma de diferentes operações simultâneas, mas na aplicação de uma mesma operação sobre múltiplos elementos de uma estrutura de dados. Esse estilo de programação é adotado em máquinas paralelas do tipo SIMD, ou MIMD programadas utilizando o modelo SPMD.

Uma das primeiras máquinas comerciais a adotar esse paradigma de programação foi a Connection Machine Modelo CM-1, introduzida pela Thinking Machines Corporation em 1986. A programação na CM-1 utiliza a linguagem de programação C*. Outros exemplos de linguagens de programação que dão suporte ao paralelismo de dados são a CPAR [30], PC++ [63] e, mais recentemente, a HPF [29].

2.2.7 - Métricas e Medidas de Desempenho

Medidas de desempenho permitem a análise do ganho obtido com a utilização de processamento paralelo. Duas medidas usuais são: o *speedup* (ganho de desempenho) e a eficiência.

□ Speedup

O *speedup* (S) obtido por um algoritmo paralelo executado em p processadores é a razão entre o tempo de execução da versão serial mais rápida (t_s) e o tempo de execução da versão paralela (t_p) [168].

□ Eficiência

A eficiência (E) é a razão entre o *speedup* obtido com a execução de um algoritmo em p processadores e p. Esta medida mostra o quanto o paralelismo foi explorado no algoritmo. Quanto maior a fração inerentemente sequencial do programa, menor será a eficiência [168].

2.3 - Ferramentas para Programação Paralela

Com o crescente interesse na área da programação paralela, verificou-se a necessidade de ferramentas que auxiliassem os profissionais e pesquisadores na produção de programas paralelos. Como na programação sequencial, são necessárias diversas ferramentas para cobrir todo o ciclo de desenvolvimento dos programas, tais como compiladores, Visualizadores, depuradores, monitores, etc.

Com os esforços efetuados nessa área surgiram uma grande gama de ferramentas para programação paralela. Nas seções abaixo é dada uma pequena descrição sobre as ferramentas que fizeram parte de uma extensa pesquisa sobre as ferramentas para programação paralela existentes.

□ ADAPTOR (Automatic DATA Parallelism TranslatOR)

A ADAPTOR [1, 2] é uma ferramenta que transforma programas com paralelismo de dados, escritos em Fortran, com extensões de vetores, *loops* paralelos e diretivas de layout/distribuição de dados, para programas paralelos com passagem de mensagens explícitas.

□ AIMS (The InstruMentation System)

A AIMS [108] é uma ferramenta para análise de desempenho e instrumentação de sistemas paralelos, capaz de fazer a análise de um programa e as alterações automáticas necessárias para a melhoria do seu desempenho.

□ ATEXPERT

A ATEXPERT [102] é uma ferramenta para análise de desempenho de programas paralelos complexos, possibilitando a visualização e o monitoramento do desempenho com apresentação gráfica dos dados.

□ **BBN PERFORMANCE TOOLS**

A BBN PERFORMANCE TOOLS [107] é uma ferramenta para análise de desempenho em programas paralelos. Ela é constituída basicamente por dois módulos: o GIST responsável pela análise de eventos e o ProfView que gera um perfil estatístico do programa.

□ **BLOBS**

A BLOBS [135] é uma ferramenta de visualização para avaliação interativa de estratégias alternativas para o balanceamento de carga através de simulação, bem como o partição do código para ser executado em vários processadores. Possui facilidades para o controle do número de processadores e métodos de partição.

□ **C****

A linguagem C** [152] é uma linguagem baseada na exploração do modelo de paralelismo de dados de granulosidade grossa. É uma linguagem orientada a objetos que herda as abstrações de dados do C++. Dá Suporte para operações paralelas com *threads* independentes.

□ **CALYPSO**

Calypso [159] é um sistema de software capaz de auxiliar no desenvolvimento e execução de programas paralelos em plataformas, sistemas operacionais e compiladores comerciais. Calypso provê uma utilização transparente de recursos compartilhados e possui uma arquitetura moderna garantindo uma forte tolerância a falhas.

❑ **CANOPY**

CANOPY [85, 86] é uma biblioteca em tempo de execução para o desenvolvimento eficiente de algoritmos orientados a malhas em sistemas maciçamente paralelos da classe MIMD, utilizando o modelo SPMD.

❑ **CAPER (Concurrent Application Programming Environment)**

CAPER [80] é um ambiente integrado para programação paralela que consiste em uma ferramenta de programação visual para auxiliar o programador no desenvolvimento da aplicação, um pré processador, um depurador para examinar os estados dos processos e da comunicação e uma ferramenta básica para monitoração de desempenho. ENTERPRISE Enterprise [13, 14] é um ambiente integrado composto por três componentes: a interface gráfica com o usuário, o pré-compilador e o sistema de tempo de execução. O Enterprise permite aos usuários criar, executar e depurar programas paralelos em uma rede de estações de trabalho.

❑ **CC++**

A linguagem CC++ [127] provê algumas extensões para a linguagem seqüencial C++, de modo a fornecer um sistema que seja adequado para uma ampla variedade de modos de programação paralela, tanto em ambientes com memória compartilhada como distribuída. A CC++ dá suporte para diferentes níveis de granulosidade na exploração do paralelismo.

❑ **CHARM**

CHARM [60, 61] é uma extensão da linguagem C para programação paralela em sistemas com memória compartilhada e passagem de mensagens. Para a geração do código na linguagem C, utiliza-se construções paralelas específicas da máquina.

❑ **CHARM++**

CHARM++ [153, 154] é uma linguagem para programação paralela orientada a objetos, baseada em C++, utilizando paralelismo explícito. A CHARM++ fornece uma

clara separação entre objetos seqüenciais e paralelos. O modelo de execução e a estratégia de escalonamento utilizam passagem de mensagens.

□ **CHIMP (Common High-level Interface to Message Passing)**

A CHIMP [144] é uma biblioteca de passagem de mensagens que fornece uma interface portátil de alto nível para uma grande variedade de computadores paralelos e sistemas distribuídos. O CHIMP pode ser utilizado para a implementação de programas em C ou Fortran.

□ **CID**

CID [155] é um super conjunto da linguagem C que implementa um sistema de memória compartilhada para máquinas de memória distribuída. O CID utiliza um pré-processador que gera os *handlers* de requisições e recebimentos.

□ **CM (Communications Manager)**

CM [93, 94] é uma biblioteca de rotinas para execução paralela ao nível de tarefas em uma rede de máquinas heterogêneas, provendo serviços de comunicação, de diretório, gerenciamento de aplicação, como também o gerenciamento de tarefas. Possui facilidades para especificar as aplicações a serem executadas e seus argumentos de entrada e saída.

□ **CODE 2.0**

A CODE 2.0 [40] é uma linguagem de fluxo de dados para o desenvolvimento de programas paralelos de granulosidade grossa. Um programa CODE 2.0 consiste em nós e arcos, onde os nós representam as computações seqüenciais expressadas por uma linguagem seqüencial, e os arcos representam as dependências de fluxo de dados entre os nós.

□ **COOL (Concurrent Object Oriented Language)**

COOL [66, 67] é uma extensão da linguagem C++ projetada para expressar paralelismo ao nível de tarefas em multiprocessadores. Fornece também um conjunto de ferramentas para a identificação da memória e gargalos do sistema, bem como a análise do desempenho do programa sobre diferentes hierarquias de memória.

□ **CPAR (C PARalelo)**

A linguagem CPAR [30] é uma extensão da linguagem C que foi projetada visando oferecer construções simples para a exploração do paralelismo em múltiplos níveis, além de permitir a otimização do uso da localidade de memória.

□ **CPS (Cooperative Processes Software)**

A CPS [127] é uma biblioteca de rotinas que podem ser utilizadas para programação paralela nas Linguagens C e Fortran, provendo suporte à distribuição de tarefas através de uma rede heterogênea de máquinas Unix. Também possui uma ferramenta para depuração (JMDB), um gerenciador de tarefas e duas ferramentas de monitoramento de desempenho (CPS_XPSMON e CPS_PSMON).

□ **CRAY/fpp**

A CRAY/fpp [117, 118] é uma ferramenta para paralelização de programas Fortran onde o código paralelo é gerado de forma automática na sua compilação. Provê uma interface com o usuário orientada a lotes e uma opção de transformação do código fonte para arquiteturas CRAY.

□ **DINO (Distributed Numerically Oriented Language)**

A linguagem DINO [8, 9] é uma extensão da linguagem C para o desenvolvimento de aplicações paralelas, explorando paralelismo de dados em multicomputadores. Utiliza o paradigma SPMD.EPPP O EPPP [145] é um ambiente para programação paralela portátil destinado para gerações atuais e futuras de computadores paralelos. Ele

consiste nos seguintes módulos: um compilador High Performance C (HPC), um simulador e um visualizador/otimizador de desempenho.

□ EXECDIFF

A EXECDIFF [127] é uma ferramenta para depuração paralela que consiste em uma biblioteca de funções para especificar quais serão os objetos que serão monitorados durante a execução do programa.

□ EXPRESS

EXPRESS [50, 51] é um conjunto de ferramentas que atuam em todas as fases de desenvolvimento de uma aplicação paralela. Ele consiste em um paralelizador automático, um visualizador algorítmico, uma biblioteca de comunicação de alto nível, um depurador paralelo e um conjunto de ferramentas para análise de desempenho.

□ FALCON

A FALCON [113] é uma ferramenta para monitoração *on-line* e visualização gráfica de programas paralelos, fazendo uma instrumentação interativa dos programas em máquinas de memória compartilhada.

□ FORCE

A linguagem FORCE [127] é uma extensão da linguagem Fortran para multiprocessadores. Ele transforma um programa FORCE para um programa Fortran 77 com construções paralelas dependentes do sistema, sendo portanto um tipo de pré-compilador.

□ FORGE

O FORGE [12] é um ambiente integrado para análise e instrumentação de programas Fortran 77. Dispõe de ferramentas de auxílio à paralelização semi-

automática para arquiteturas do tipo vetorial, memória compartilhada ou memória distribuída.

□ **FORTTRAN 90**

A Fortran90 [19] é uma extensão do Fortran 77 para a programação paralela, possuindo algumas características adicionais como operações com vetores, novas funções para operação com matrizes e maior facilidade para organizar dados e definir procedimentos.

□ **FORTTRAN M**

A Fortran M [39] é uma extensão da linguagem Fortran 77 para programação paralela, que suporta uma aproximação modular para o desenvolvimento de programas paralelos com passagem de mensagens e paralelismo ao nível de tarefas.

□ **FTS - Fort Translation System**

O Fort Translation System [156] é um tradutor de código-fonte para código-fonte completamente automático, que aceita um programa escrito em um subconjunto da linguagem Fortran 90 e gera um código com todas as primitivas de comunicação e distribuição dos vetores para um sistema de passagem de mensagens. É formado por 6 módulos: Code Handler, Preference Determination Module, Preference Combination Module, Distribution Scheme Generator, Code Generator e Performance Estimator;

□ **GENMP**

A GENMP [96] é uma biblioteca de funções em tempo de execução para arquiteturas de computadores MIMD. Provê um balanceamento de carga dinâmico e o gerenciamento da comunicação entre os processadores.

□ **GPMS (General Parallel Monitoring System)**

O GPMS [127] é uma ferramenta para análise e monitoração de sistemas desenvolvidos nas linguagens C e Fortran77, fazendo uma verificação das possibilidades de paralelização do código fonte desses programas.

□ **GRIDS**

A GRIDS [41] possui um ambiente para computação numericamente intensiva baseada em malhas. Pode-se definir, através de uma linguagem declarativa, todo o controle do método da solução. A descrição da topologia é separada da descrição do algoritmo computacional, e as construções de GRIDS são extensões da linguagem Fortran. O GRIDS dá suporte para malhas regulares e irregulares.

□ **HENCE (Heterogeneous Network Computing Environment)**

O HeNCE [15, 16] é ambiente integrado para ajudar no desenvolvimento de programas paralelos que são executados sobre uma rede de computadores. HENCE é composto por ferramentas gráficas integradas para criar, compilar, executar, e analisar os programas fontes.

□ **HPC++ (High Performance C++)**

A HPC++ [147] é uma extensão da linguagem C++ formada por uma biblioteca de classes em C++, diretivas de compilador e umas poucas extensões de linguagem para alcançar a meta da programação paralela portátil. É uma tentativa de criar um modelo padrão para programação paralela em C++.

□ **HPF (High Performance Fortran)**

A HPF [29] é uma extensão da linguagem Fortran 90 para programação paralela de alto desempenho, utilizando o modelo de programação de paralelismo de dados. O paralelismo é proveniente de operações independentes sobre elementos de grandes estruturas de dados.

□ **HYPERTOOL**

A HYPERTOOL [62] é uma extensão da linguagem C para programação paralela, cujo pré-compilador transforma o código escrito em Hypertool para código C com passagem de mensagens entre os procedimentos. O HYPERTOOL dispõe de uma ferramenta que permite estimar o desempenho dos programas paralelos gerados.

□ **IMPROV (Integrated Manipulation of PROgram Visualization)**

O IMPROV [27, 28] é uma meta-ferramenta para rápida prototipação de visualizações complexas do comportamento de programas paralelos, fornecendo suporte para uma generalização formal e completa da visualização do programa, através de conceitos básicos como eventos e comportamentos.

□ **INTEL PERFORMANCE ANALYSIS TOOLS**

INTEL PERFORMANCE ANALYSIS TOOLS [103] é um conjunto de ferramentas gráficas para análise de desempenho baseada no Paragraph. Faz a animação da execução do programa paralelo utilizando as informações de rastreamento geradas durante a execução da aplicação.

□ **IPS-2**

A IPS-2 [110, 111, 112] é uma ferramenta para análise de desempenho de programas paralelos nas linguagens C e Fortran. Ela provê o monitoramento, análise e visualização do desempenho do programa.

□ **ISIS**

ISIS [52, 53, 54] é uma biblioteca que foi desenvolvida para fornecer ferramentas de construção de programas distribuídos tolerantes a falhas. Os processos que compõem a computação distribuída são organizados em grupos. Para um processo participar da computação, ele tem que se associar a um grupo já existente ou criar um grupo novo.

□ JADE

JADE [58, 59] é uma linguagem declarativa, orientada a dados para programação paralela com granulosidade grossa. O JADE dá suporte para máquinas baseadas em passagem de mensagens, memória compartilhada e redes de estações heterogêneas.

□ KAP/CRAY

A KAP/CRAY [119] é uma ferramenta para paralelização de sistemas desenvolvidos na linguagem Fortran. Tal paralelização é feita de forma automática para arquiteturas da CRAY.

□ KSR PERFORMANCE TOOLS

O KSR PERFORMANCE TOOLS [106] é um conjunto de ferramentas para análise de desempenho em programas paralelos. É composto por uma biblioteca de monitoramento de desempenho e um visualizador dos dados gerados pela sub-rotina de monitoramento.

□ LINDA

LINDA [17, 18] é um modelo para a comunicação e coordenação de processos paralelos. O meio básico de comunicação é o *tuple space*. Um *tuple space* é uma área de *tuples* compartilhada, que é acessada de modo associativo através de um mecanismo de busca.

□ LMPS (Livermore Message Passing System)

A Livermore Message Passing System [127] é uma biblioteca de rotinas para programação paralela que implementa um sistema eficiente de passagem de mensagens no BBN TC2000, dando suporte para comunicação síncrona e assíncrona, bloqueante e não bloqueante.

□ **LPB (Linda Program Builder)**

O Linda Program Builder [162] é um ambiente de programação de alto-nível que apoia o projeto e desenvolvimento de software paralelo. Ele reduz em muito o esforço na construção de programas paralelos pela manutenção de um banco de dados com os modelos de paralelização disponíveis.

□ **MARITXU**

MARITXU [115, 116] é uma ferramenta para análise de desempenho de programas paralelos onde o usuário determina suas preferências, ícones, mapeamento de dados, topologia, estatísticas e define a interface entre uma ferramenta de monitoração e o Maritxu.

□ **MEANDER**

O MEANDER [161] é um conjunto de ferramentas visuais que trabalham seguindo o paradigma de processos sequenciais que executam paralelamente e interagem através da passagem de mensagens. Esse conjunto de ferramentas inclui: editores gráficos para a especificação, codificação do programa e configuração do hardware, uma ferramenta de mapeamento, uma ferramenta para visualização de lógica e visualização de desempenho.

□ **MELDC**

MELDC [68] é uma linguagem de programação concorrente orientada a objetos baseada em C. Em MELDC as classes e meta-classes são objetos, que são escritos na linguagem MELDC. O comportamento dos objetos pode ser computado e estendido em tempo de execução e podem ser construídas novas propriedades e características.

□ **MENTAT**

A MENTAT [64, 65] é uma extensão da linguagem C++ para programação paralela que segue o modelo de programação dirigida a dados, utilizando especificações do usuário para compor o paralelismo. Possui um compilador que detecta automaticamente

a dependência de dados e controles entre as instâncias de classes do MENTAT envolvidas na sua invocação, comunicação e sincronização.

□ MIPRAC

O MIPRAC [137] é um compilador paralelizador interprocedural para computações simbólicas e numéricas, que opera sobre uma forma intermediária orientada a semântica, para uma grande variedade de linguagens como Scheme, Common Lisp, C e Fortran.

□ MODULA-2*

Modula-2* [73, 74] é uma extensão da linguagem Modula-2 para programação paralela que utiliza todos os mecanismos de abstração de dados da sua linguagem de origem. O compilador faz a distribuição automática dos dados e dos processos em computações paralelas síncronas e assíncronas.

□ MPI

MPI [55] não é propriamente a implementação de um sistema, mas sim uma proposta de padronização para a interface de troca de mensagens em máquinas paralelas com memória distribuída. A idéia de se padronizar essa interface é a de possibilitar a portabilidade dos programas entre máquinas diferentes e facilitar o seu uso.

□ MPPE (MasPar Programming Environment)

MPPE [98] é um ambiente integrado para programação paralela composto por um depurador gráfico, um profiler de desempenho e um visualizador, com arquitetura cliente-servidor para depuração remota. O SISTEMA Modula-2* O Sistema Modula-2* [73, 74] é um ambiente para programação paralela portátil para Modula-2*, uma extensão a linguagem Modula-2. Este sistema consiste em um compilador otimizador e reestruturador, um depurador paralelo interativo, um utilitário de *make* automático, um painel de controle gráfico X-Windows, sistemas de tempo de execução para diferentes máquinas e conjuntos de bibliotecas paralelas escaláveis.

□ **MTASK**

A MTASK [127] é uma linguagem para programação paralela que pode ser executada no sistema operacional Windows NT e fornece facilidades para o desenvolvimento de programas em multiprocessadores. Possui os seguintes componentes: manipulador de tarefas, *lock* paralelo, evento paralelo e *Data Plate*.

□ **OBJECT-ORIENTED FORTRAN**

Object-oriented Fortran [37, 38] é uma extensão da linguagem Fortran para o desenvolvimento de aplicações paralelas que dá suporte à declaração, criação e gerenciamento de objetos. Possui um pré-processador para transformar essas construções de objetos para o Fortran 77 padrão. Utiliza o paradigma de programação de passagem de mensagens.

□ **OCCAM**

A OCCAM [137] é uma linguagem de programação paralela que possibilita a descrição de um programa como uma coleção de processos que operam concorrentemente e se comunicam através de canais.

□ **OOMPI (Object-Oriented Message Passing Interface)**

OOMPI [148] é uma biblioteca de passagem de mensagens orientada a objetos para programação paralela. É uma versão orientada a objetos da MPI (Message Passing Interface). A OOMPI consiste em uma biblioteca de classes que encapsulam a funcionalidade da MPI em uma hierarquia funcional de classes, fornecendo uma interface simples, flexível e intuitiva.

□ **OPUS**

A OPUS [142] é um conjunto de extensões da linguagem Fortran que fornece suporte integrado ao paralelismo de dados e de tarefas. Ele possui um sistema de tempo de execução dividido em duas partes, uma dependente e a outra independente da linguagem.

□ **P2D**

O P2D [132] é um ambiente integrado para o auxílio à paralelização de aplicações escritas em Fortran. O sistema é composto por um reformatador, um editor, um analisador de código, um instrumentador, um paralelizador, um executor, um simulador e um visualizador. Diversas técnicas de Engenharia de Software foram utilizadas na sua implementação.

□ **P4**

A P4 [48, 49] é uma biblioteca de macros e subrotinas para programação em uma variedade de máquinas paralelas, com diferentes modelos de acesso à memória. A biblioteca p4 pode ser utilizada com as linguagens C ou Fortran, provendo funções para o envio e recebimento de mensagens, para comunicação global e para sincronização.

□ **PABLO (Performace Analysis Enviroment)**

O PABLO [109] é um ambiente para análise de desempenho de programas paralelos desenvolvidos na linguagem C que possui uma interface gráfica e uma biblioteca de documentação que pode ser utilizada para descrever processos e tipos de dados.

□ **PADE**

O PADE [79] é um ambiente flexível e customizado para o desenvolvimento de aplicações paralelas que usam a biblioteca de passagem de mensagens . O PADE consiste em uma interface gráfica e intuitiva com o usuário e uma série de utilitários PVM.

□ **PARA++**

A PARA++ [149] é uma extensão da linguagem C++ . É composta por uma biblioteca de construções em C++ para programação paralela utilizando o paradigma de passagem de mensagens para a comunicação.

□ **PARAGRAPH**

A PARAGRAPH [104, 105] é uma ferramenta para análise gráfica de desempenho de aplicações paralelas através da visualização dos dados obtidos em rastreios gerados por programas paralelos com passagem de mensagens. Outras informações sobre o desempenho e gargalos do programa paralelo sob análise também são apresentadas na forma de gráficos.

□ **PARALEX**

Paralex [158] é um ambiente de programação completo que faz uso extensivo de gráficos para definir, editar, executar e depurar aplicações científicas paralelas. Todo o código necessário para distribuir a computação e replicá-la, como também o balanceamento dinâmico de carga são gerados automaticamente pelo sistema. O Paralex consiste em quatro componentes lógicos: um editor gráfico para o desenvolvimento dos programas, um compilador, um executor e um ambiente de suporte em tempo de execução.

□ **PARALLAX**

PARALLAX [82, 83] é uma linguagem para especificar um programa paralelo como um diagrama hierárquico de fluxo de dados de granulosidade grossa. Possui ferramentas para especificar a máquina destino como um grafo de processadores e conexões de rede.

□ PARALLAXIS

A PARALLAXIS [71, 72] é uma extensão de Modula-2 para programação paralela com paralelismo de dados, utilizando o modelo SIMD. Para descrever a máquina paralela virtual são utilizados o número de processadores e quantidade de memória, o nome dos portos de comunicação e a topologia da rede.

□ PARAPET (Parallel Programming Environment Toolkit)

O PARAPET [150] é um conjunto de ferramentas e um ambiente de programação paralela desenvolvido para prototipação e para facilitar a portabilidade dos programas paralelos. Trabalha sobre o modelo SPMD com passagem de mensagens.

□ PARASCOPE

O PARASCOPE [10, 11] é um ambiente para programação paralela desenvolvido para suportar programação científica em multiprocessadores. Possui um conjunto de ferramentas que utilizam a análise global do programa para ajudar os usuários a desenvolver e depurar programas paralelos.

□ PARASPHERE

O PARASPHERE [127] é um ambiente integrado para programação paralela com interface gráfica semelhante ao OSF/Motif, trabalha com o High Performance Fortran com interface para C e Fortran 77. O ambiente é formado por um depurador paralelo interativo ao nível de código, um analisador estatístico, um browser gráfico para visualizar o relacionamento entre os procedimentos e um tradutor de Fortran 77 para DECmmp High Performance Fortran.

□ PARCEL

O PARCEL [136] é uma ferramenta para programação paralela que consiste em um compilador e um sistema de tempo de execução que automaticamente paraleliza um programa Scheme para a execução em multiprocessadores.

□ **PARMACS**

A PARMACS [87, 88] é uma interface de programação de passagem de mensagens para multiprocessadores e multicomputadores. Consiste de uma biblioteca de funções e macros e também de uma ferramenta para análise de desempenho, onde podem ser visualizados o estado dos processos e o estado da comunicação entre processos (*post-mortem*).

□ **PARTI (Parallel Automated Runtime Toolkit at ICASE)**

A PARTI [91, 92] é um conjunto de procedimentos para pré processamento em tempo de execução, realizando a paralelização automática de aplicações seqüenciais e coordenando o movimento de dados entre processadores.

□ **PAT (Parallelization AssistanT)**

O PAT [20, 21, 22] é um conjunto de ferramentas para paralelização interativa de loops em programas Fortran. Inclui ferramentas para análise, transformação, paralelização interativa e instrumentação de programas.

□ **PC++ (Parallel C++)**

PC++ [63] é uma extensão da linguagem C++ projetada para permitir a construção de uma coleção de classes agregadas que podem ser distribuídas sobre a hierarquia de memória de uma máquina paralela na forma definida pelo High Performance Fortran Forum (HPFF).

□ **PCN (Program Composition Notation)**

A PCN [77] é uma linguagem semelhante à linguagem C que pode ser utilizada para escrever programas paralelos. Possui um compilador altamente portátil, um depurador ao nível de código integrado e ferramentas para análise de desempenho.

□ PDDP (The Parallel Data Distribution Preprocessor)

A PDDP [56, 57] é uma extensão da linguagem Fortran para a programação paralela de dados em memória compartilhada, possuindo operações com vetores e diretivas para a distribuição de dados.

□ PEMPI

O PEMPI [157] é um ambiente de programação paralela que permite o programador trabalhar dentro do contexto de funções de alto-nível para resolver problemas regulares, como também no nível de chamadas MPI para resolver problemas irregulares ou de desempenho crítico. A arquitetura de PEMPI possui duas camadas: *Kernel* PEMPI (KPEMPI) e um conjunto de ferramentas chamado Outer PEMPI (OPEMPI).

□ PFP - (Fortran Preprocessor)

PFP [42, 43] é uma extensão da linguagem Fortran para expressar paralelismo através do modelo de programação paralela *fork-join* em multiprocessadores, apresentando construções de controle com baixo custo e sem necessidade de sincronização.

□ PHOSPHORUS

A Phosphorus [151] é uma biblioteca de funções que permite o compartilhamento de dados entre programas paralelos para sistemas com memória distribuída e passagem de mensagens. Utiliza PVM como ferramenta de comunicação entre processos.

□ PICL (Portable Instrumented Communications Library)

A PICL [89, 90] é uma biblioteca de subrotinas utilizada para o desenvolvimento de programas paralelos que são portáteis em arquiteturas de memória distribuída que utilizam comunicação por passagem de mensagens. Provê um rastreamento da execução de programas para ser utilizado no monitoramento de desempenho.

□ **P-LANGUAGES (PC e PFortran)**

PC e PFortran [36] são extensões das linguagens C e Fortran para programação paralela, utilizando o modelo de programação de memória compartilhada para operações binárias associativas e comutativas em máquinas com passagem de mensagens. Utiliza o modelo SPMD. X3H5 A X3H5 [33, 34, 35] é uma extensão da linguagem Fortran e C para programação paralela que utiliza o modelo SPMD, baseado no paradigma *fork-join*, em ambientes de programação com memória compartilhada.

□ **POLARIS**

O POLARIS [140] é um compilador paralelizador Fortran para aplicações industriais, que utiliza processadores maciçamente paralelos com um espaço de endereçamento global. Esse compilador tem como entrada um programa escrito em Fortran 77 gerando um código paralelo executável como saída.

□ **PRISM**

O PRISM [97] é um ambiente integrado para programação paralela formado por um depurador gráfico, uma ferramenta para análise de desempenho e um visualizador de dados.

□ **PTOPP (Practical Toolset for the Optimization of Paralle Programs)**

O PTOPP [138, 139] é um conjunto de ferramentas que auxiliam o programador na otimização de programas científicos para computadores paralelos. É orientado para duas fases principais do desenvolvimento de um programa: a criação de um conjunto consistente de variantes de um programa experimental e a interpretação dos resultados da compilação e análise de desempenho.

□ **PUL (Parallel Utilities Library)**

A PUL [145] é uma biblioteca de utilitários para programação paralela que auxilia os programadores através de um conjunto de componentes reutilizáveis para a implementação de aplicações paralelas. Está dividida em quatro grupos de utilitários:

SYSTX, EVENT, GRID e MESH. Fornece um serviço complementar de passagem de mensagens para a biblioteca CHIMP e serviços para que um grupo de processos realizem I/O através de um arquivo comum.

❑ **PVM - Parallel Virtual Machine**

A biblioteca PVM - Parallel Virtual Machine [31, 32] foi projetada para sistemas heterogêneos, compostos de estações de trabalho e PC's. O ambiente da PVM é constituído de uma biblioteca de rotinas com interface para C e FORTRAN, cuja principal função é a troca de mensagens entre processos.

❑ **PYRROS**

A PYRROS [26] é uma ferramenta para paralelização de programas com o paralelismo sendo especificado pelo usuário. A linguagem é gráfica e textual, onde pode-se definir graficamente um grafo de tarefas que serão executadas pelo programa e as dependências de dados entre essas tarefas.

❑ **SCHEDULE**

SCHEDULE [24, 25] é uma ferramenta para paralelização de programas. O paralelismo é especificado pelo usuário através de uma linguagem de fluxo de dados com granulosidade grossa para especificar as dependências entre blocos de código escritos em Fortran.

❑ **SISAL**

A SISAL [76] é uma linguagem funcional de propósito geral para computação paralela numérica. A SISAL explora o paralelismo automático além de possuir um compilador otimizador e um depurador simbólico.

❑ **SPPL (Stuttgart Parallel Processing Library)**

A SPPL [95] é uma biblioteca de passagem de mensagens para sistemas heterogêneos com memória distribuída. Essa biblioteca foi projetada para trabalhar com a linguagem C e não possui suporte para a linguagem Fortran. Podem ser utilizados diversos tipos de dados abstratos nas mensagens, tais como: tipos de dados simples, vetores, registros e estruturas de ponteiros arbitrárias.

□ SR

A SR [46, 47] é uma linguagem para escrever programas paralelos e distribuídos, dando suporte para diferentes ambientes de programação. Consiste em um compilador que traduz SR para C, um *link editor* que gera saídas para o *link editor* ld e algumas bibliotecas da linguagem C.

□ STRAND88

STRAND88 [75] é uma linguagem de desenvolvimento para processamento paralelo semelhante ao Prolog. Possui também um ambiente contendo ferramentas para o monitoramento da carga de comunicação e de processamento, para visualização dos dados e um depurador.

□ TCGMSG (Theoretical Chemistry Group Message Passing System)

A TCGMSG [84] é um conjunto de ferramentas para escrever programas paralelos portáteis utilizando o modelo de passagem de mensagens. É formado por uma biblioteca de passagem de mensagens, (um pequeno subconjunto do PARMACS), para multicomputadores ou multiprocessadores.

□ TINY

TINY [23] é uma ferramenta educacional para paralelização interativa, direcionada para experiências de testes com dependência de dados. A TINY provê uma análise de dependência e fornece as transformações necessárias nos programas fontes, demonstrando os resultados através de uma interface gráfica com o usuário.

❑ TOPSYS (TOols for Parallel SYStems)

TOPSYS [78, 79] é uma coleção de ferramentas para programação paralela que consiste em: objetos autônomos (semáforos, caixa-postal e tarefas), biblioteca de passagem de mensagens, depurador paralelo (DETOP), monitor de desempenho (PATOP) e uma ferramenta para visualização do programa (VISTOP).

❑ TOTALVIEW

O TOTALVIEW [127] é um depurador multi-processos para programas paralelos, com interface gráfica para controle do programa e definição dos *breakpoints*. Pode ser utilizado para programas em C, C++ ou Fortran.

❑ UBD (KSR symbolic debugger)

O UDB [99] é um depurador simbólico para programas paralelos que trabalham sobre a plataforma KSR. Possui uma interface gráfica baseada em janelas para a visualização do *status* e para controle do programa que está sendo depurado.

❑ UPSHOT

A UPSHOT [114] é uma ferramenta gráfica para análise de desempenho de programas paralelos desenvolvidos nas linguagens C, Fortran, Aurora Parallel Prolog e PCN. A análise é feita usando rastreios de execução produzidos pelos programas paralelos.

❑ VIENNA FORTRAN COMPILATION SYSTEM

VIENNA [69, 70] é uma ferramenta para programação paralela que transforma programas fonte em Vienna Fortran ou Fortran 77 para código fonte Fortran com passagem de mensagens explícita. A geração do código pode ser feita para várias máquinas diferentes. A paralelização e vetorização do código é feita de forma automática.

□ VISAGE (VISual Attributed Graph Environment)

O VISAGE [81] é um ambiente para programação paralela baseada em grafos para decomposição funcional em multicomputadores. Ele é composto por um ambiente de programação visual, ferramentas para previsão de desempenho e simulação do comportamento da execução e um analisador *post-mortem*.

□ VOYEUR

A VOYEUR [127] é uma ferramenta para análise de desempenho de programas paralelos desenvolvidos nas linguagens Fortran e C. Possui uma biblioteca de rotinas para geração e inspeção de rastreios.

□ VPE

O VPE [160] é um ambiente de programação paralela visual para paralelismo em ambientes com comunicação por passagem de mensagens. A estrutura de um programa é descrita através de um grafo no qual os nós representam processos e o fluxo de mensagens os arcos.

□ XPDB

XPDB [101] é um depurador gráfico para programas paralelos que utilizam a biblioteca de passagem de mensagens SPPL (Stuttgart Parallel Processing Library), com a possibilidade de impressão das mensagens no formato apropriado, dependendo do tipo de dado.

2.4 - Comparação das Principais Características das Ferramentas Pesquisadas

A tabela 2.1, apresenta uma tabela comparativa das ferramentas descritas segundo algumas características importantes para este trabalho.

Tabela 2.1: Características das Ferramentas para Programação Paralela

Ferramenta	Código-fonte	Sistema Operacional	Linguagem de programação	Linguagem de programação	Interface gráfica
ACTIVE-MESSAGES	Disponível	Vários		Fortran	Não
ADAPTOR	Disponível	UNIX	C	Fortran	X Windows
AIMS	Disponível	UNIX		Fortran	Não
ATEXPERT	Comercial	UNIX	C	Fortran, C	X Windows
BBN	Comercial	BBN-10, SunOS, RSX-11, OS		Fortran	DIRC
BLOBS	Disponível	UNIX	C		Xview.e OpenWindows
BSP-Lib	Disponível	Unix/Linux		Fortran	Não
C REGION LIBRARY	Disponível	SunOS4.1/TCP	C	C	Não
C++	Disponível	Vários			Não
CALYPSO	Comercial	SunOS	C++	C++	
CANOPY	Disponível	POSIX-like system calls	C	Fortran	Não
CAPER		UNIX, VORX	C, C++ Concurrent C	C, C++ Concurrent C	X11R3-R5
CARNIVAL		SUN.com PVM	C		PerlTk

CC++	Disponível	UNIX	C, C++	CC++	Não
CDS	Comercial	Solaris, IRIX			Não
CHARM	Disponível	Vários	C	C, C++	X Motif
CHARM++	Disponível	Sun/OS Solaris, IRIX HP	C++	C++ Client	Não
CHIMP	Disponível	Vários	C	Fortran, C	Não
CHIRON	Comercial				
GID		Digital Alpha/OSF1	C	GID, C	Não
CM	Disponível	Sun/OS DEC/Unix SGI/IREX/OS	C, TCP/IP UNIX/OS	C	Não
CODE 2.0	Comercial	Vários	C++	Code 2.0	X11R4 ou X11R5
COOL	Disponível	UNIX	C, C++	COOL	Não
CPAR	Disponível	Vários	C	CPAR	Não
CPS	Disponível	UNIX	C	C, Fortran	Não
CRAY/fpp	Comercial	UNICOS, COS		Fortran	Não
DIMENAS	Disponível	IRIX, Solaris			X-Windows
DINO	Disponível	Sun/OS	C, Pascal	C	Não
DPV				C	X-Windows
ENTERPRISE		UNIX	Smalltalk, C, C++	C	X Windows

EXPP	Disponível	UNIX	C	HP	Não
EXECDIFF	Disponível	UNIX	C	C, Fortran	Não
EXPRESS	Comercial	NX on i386/860 Paragon OSE/1	Fortran 77 Fortran 90 C	C Fortran	X-Window Sunview
FALCON			C		X-Window
FALCON	Disponível	SUN OS Mach 1000	C, C++	C	Não
FORCE	Disponível	Vários	Fortran 77, E, C	Force	Não
FORGE	Comercial	UNIX	C	Fortran 77 Fortran 90 HP	Sunview X-Window
FORTAN 90	Disponível	Vários	Dependente	Fortran 90	Não
FORTAN M	Disponível	Vários	C e Perl	Fortran M	Não
FTS		VX9000/VMS	Modula-2	Fort	Não
GENMP	Disponível	Vários	Fortran 77	Fortran 77	Não
GPMS	Disponível	Trollius	C	C, Fortran 77	X11R4
GRIDS	Comercial	ADK	C	FORTAN 77	X11R5
GUARD		Sun, Silicon Graphics, IBM RS6000			
HENCE	Disponível	UNIX	C	C, Fortran 77	X11R4
HPC++	Disponível	Vários	C++	HPC++	Não
HPE	Disponível	Vários	Dependente	HPE	Não

HYPERTOOL	Disponível	Vários	C	C	Não
ICC	Disponível	Paran OS Sun OS	C	C Fortran	Não
IMPROV	Disponível	UNIX	C	Ascii e binário	X11R4
INTEL P A 1	Comercial	IPS 860 Paran OS Com 486	C C++	C Fortran	Motif
IPS-2	Comercial	UNIX	C	Fortran C	X11
ISIS	Disponível	UNIX	C	C C++ Fortran Lisp Ada	
JADE	Disponível	Vários	C	C	Não
KAP/GRAY	Comercial	UNIX OS COS UNIX Ibm		Fortran	Não
KSR	Comercial	KSR OS	C	Fortran 77 C	Não
LAM	Disponível	Unix /Linux	C	C	Não
LEGION	Disponível	Linux, Solaris, Aix, Irix, Dec Unix	C	Mentat, C	Não
Lightweight Browser	Corefile Disponível	Solaris SunOS HP UX AIX	C++		EC/TK
LINDA	Disponível	Vários	Vários	Vários	
LMPS	Disponível	UNIX	C PCP	Fortran 77 C	Não

LPB	Comercial		C	C-Binda	X.Windows
MAINTINS		Thinking Machines Corp. e Mas	C	Splice	Tcl/Tk
MARITXU	Comercial	UNIX	C	Transputers e CM2	X11R4
MEANDER		Embarcos e航海 e PVM	C++	Meander	X11R5
MELDC	Disponível	SunOS, Ultrix	C e assembler	MeldC	Não
MENTAT	Disponível	Varios	C++	MPI e extensão (C++)	Não
MIDWAY	Disponível	DECstations com Mach 3.0 e ATM	C	C	Não
MIPRAC			C	Scheme Common Lisp e Fortran	
MODULA 2*	Disponível	UNIX	Modula-2, C	Modula-2*	Não
MPI	Disponível	Varios	C	C, C++ e Fortran	Não
MPICH	Disponível	Unix, Linux e outros	C		Não
MPPE	Comercial	DecStation, Ultrix, SPARC e SunOS	Smalltalk, C++	C, Fortran 77	DecStation, Motif, SPARC e OpenWindow

MTASK	Disponível	Concentrix (Alliant Unix)	C	C	Não
NBLV	Comercial	Star			Windows
NUPSHOT	Disponível				Tcl/Tk
OBJECT ORIENTED FORTRAN	Disponível	Vários	C	Fortran	Não
Object Oriented MPI	Disponível	SunOS e Linux	C++		Não
OGC-AM		Dependente			Não
OOMPI	Disponível	Vários	C++	C++	Não
OPUS		UNIX	C	FORTRAN	Não
P-LANGUAGES (PC)	Comercial	Vários	C, Lex, Yacc	PC, Fortran	Não
P2D	Disponível	UNIX	C++	Fortran	X-Windows
P2D2		NAS SP-2		Utilizando PVM ou MPI	X-Windows
P4L	Disponível	Vários	C	C, Fortran/77	Não
PABLO	Disponível	Linux-PC, Solaris, HP-UX AIX	C	C	X-Windows
PADE	Disponível	UNIX	C	C, Fortran	X-Windows
PANORAMA	Disponível	Paragon, IPSC/860, nCUBE/2 e CM-5	C	C, Fortran	Tcl/tk
PARA++		SunOS e UNICOS	C	C, C++, Fortran, Pascal	X-Windows

PARADISE	Disponível	UNIX	C	Ascii	Sunview or X11R4
PARAGRAPH	Disponível	IB		Fortran	Windows
PARALEX					
PARALLAX	Disponível	Macintosh ou	Pascal	Paralla	Macintosh
PARALLAXIS	Disponível	Vários	C	Parallaxis	X11R5
PARAPET		No			Não
PARASCOPE	Comercial	SunOS, AIX	C, C++, Yacc, Lex	Fortran77	X11R4
PARASPHERE	Comercial	UNIX	C, C++	C, Fortran, DPL, EPE	Motif
PARCEL				Fortran	
PARDI	Comercial	Linux, Net e outros			Não
PARMACS	Comercial	Vários	C	Fortran77, Parmarcs, C	X11R3
PARTI	Disponível	Vários	C	C, Fortran	Não
PAT	Disponível	UNIX	C e C++	Fortran77	X11R4
PC++	Disponível	Vários	C++ e C	PC++	Não
PCN	Disponível	Vários	C	RCN, C, Fortran	X11R5
PDDP	Disponível	UNIX	C, LEX e	Fortran77	Não

PEMPI		SP1, Fujitsu AP1000	C	C	Não
PEE					
PHOSPHORUS		SunOS, Solaris	C	C	Não
PICL	Disponível		C	Fortran 77, C	Não
P-LANGUAGES (Pfortran)	Comercial	Vários	C, Lex, Yacc	PC, Pfortran	Não
POILARIS		CRAY, IBM T3E, T3E90	C	Fortran 77, C, C++ HIP	Não
PRISM	Comercial	CMOS, CMOST	C	C, Fortran	X Motif
PTOPP		UNIX	Lisp	Fortran	X-Windows GIMACS
PUL		Sun, HP Workstation	C	C, Fortran	Não
PVM	Disponível	UNIX	C	C, Fortran 77	Não
PYRROS	Disponível	UNIX	C	C	X Windows
SCHEDULE	Disponível	UNIX	C	Fortran	XMR4
SISAL	Disponível	UNIX	C	Sisal	Não
SPLHEC	Disponível	Meiko	C	Split-C	Não
SPPL	Comercial	AIX, SunOS, Ulrix	C	C	Não
SR	Disponível	UNIX	C, Yacc, Lex	SR	X-Window

STRAND88	Comercial	Vários	C, Assembly	Strand88, Fortran, C	X11top
TCGMSG	Disponível	Vários	C	Fortran	Não
TINY	Disponível	UNIX	C	Tiny	Xterm
TOPSYS	Comercial	UNIX, Hain, CHORUS	C	Fortran	Windows
TOTALVIEW	Comercial	UNIX	C++	C, C++, Fortran	X11R4
TRAPPER		Transputer	C	Fortran	
TREADMARKS	Comercial	FreeDSB e Linux	C, C++ e Fortran	C, C++ e Fortran	Não
UDE	Comercial	KSR OS	C	Fortran, C	URS
UNH C*	Disponível	Intel Delta, SUN, SGI, DEC	C	C*	Não
UNIFY	Disponível		C		Não
UPSHOT	Disponível	Vários	C	C, Fortran, Prolog, PCN	X11R4
VIENNA-FORTRAN	Disponível	Vários	GNU C	Vienna Fortran, Fortran 77	X11R5, OSE/Molt
VISAGE	Disponível	Meiko's CS tools, Mach	C, gl	C	GL
VOYEUR	Disponível	Convexos	Fortran	Fortran, C	Não
VPE		UNIX	C, Fortran	C, Fortran	X Windows

XPDB	Dependence	Vendor	Dependencies	Language	Platform
XPDB	Commercial	AIX, SUN, Ulrix	C	C	X Windows

2.5 - Considerações Finais

Dentre as várias alternativas para aumentar o desempenho de um sistema de software, revistas no Capítulo 1, o uso de processamento paralelo apoiado pela utilização de ferramentas de auxílio à paralelização mostra-se a mais adequada, principalmente quanto à facilidade de utilização.

Existem diversas ferramentas e ambientes para auxílio no desenvolvimento de aplicações paralelas. Muitas dessas ferramentas foram sumariamente descritas neste Capítulo. No próximo Capítulo é apresentada uma proposta de classificação para as ferramentas de auxílio à programação paralela.

Capítulo 3

3 - CLASSIFICAÇÃO DAS FERRAMENTAS

3.1 - Considerações Iniciais

O Capítulo 2 apresenta uma visão geral de uma grande gama de ferramentas para processamento paralelo. A diversidade dessas ferramentas e suas finalidades exigiram o estabelecimento de classes de ferramentas, permitindo a organização do esforço de desenvolvimento deste projeto e facilitando a comparação de ferramentas dentro de uma mesma classe. Neste sentido, é apresentada na Seção 3.2 uma nova proposta de classificação para as ferramentas de programação paralela. Na Seção 3.3 são apresentadas as considerações finais deste Capítulo e uma tabela onde são classificadas as ferramentas revistas no Capítulo anterior dentro do esquema proposto.

3.2 - Uma Proposta de Classificação para as Ferramentas

Através da análise de algumas formas de classificação existentes [127, 131, 132], foi proposta uma nova classificação que possibilita enquadrar um número maior de ferramentas pesquisadas de uma forma mais adequada que as formas até aqui existentes.

A nova proposta de classificação divide as ferramentas para programação paralela em 4 grandes classes. Essas classes são divididas em níveis, cada um definindo um conjunto de funcionalidades que permite enquadrar cada ferramenta para programação paralela de uma forma mais precisa.

A classificação proposta é:

A - Ferramentas de Desenvolvimento

A.1 - Linguagens

A.2 - Bibliotecas

A.3 - Ambientes de Desenvolvimento

B - Ferramentas de Paralelização

B.1 - Automáticas

B.1.1 - Pré-processadores

B.1.2 - Compiladores

B.2 - Interativas

C - Ferramentas de Análise de Desempenho e Depuração

C.1 - Depuração

C.2 - Análise de Desempenho

D - Ambientes Integrados

Nas seções abaixo é definida cada uma das classes.

A - Ferramentas de Desenvolvimento de Programas Paralelos

Nessa classe estão todas as ferramentas que fazem parte do processo de desenvolvimento de novos programas paralelos. Neste grupo de ferramentas estão as linguagens para programação paralela, as bibliotecas para computação paralela e os ambientes de desenvolvimento de aplicações paralelas.

A.1 - Linguagens

Nessa categoria, encontram-se todas as linguagens de programação que podem expressar o paralelismo através de primitivas, novos comandos ou diretivas para o compilador.

A representação do paralelismo nas linguagens paralelas pode ser feita de forma explícita, por exemplo quando o usuário utiliza primitivas de comunicação e

sincronização dentro do programa, ou de forma implícita, quando a linguagem possui comandos como DOALL e PARALLEL LOOP que são interpretados pelo compilador e transformados em código de máquina paralelo, de maneira automática.

OCCAM [137] é uma linguagem de programação paralela que possibilita a descrição de um programa como uma coleção de processos que operam concorrentemente e se comunicam através de canais.

A.2 - Bibliotecas

A implementação de um ambiente de programação paralela através de uma biblioteca de sub-rotinas é uma tarefa simples, tornando essa abordagem altamente atrativa aos desenvolvedores de sistemas. Extensões do sistema ou modificações podem ser introduzidas facilmente através de bibliotecas em tempo de execução que constituem um meio efetivo e flexível para experimentos com novos ambientes de programação. Em uma etapa posterior, essas características podem ser migradas em definitivo para as novas linguagens de programação paralela ou extensões de linguagens sequenciais existentes.

A biblioteca PVM - Parallel Virtual Machine [31,32] foi projetada para sistemas heterogêneos, compostos de estações de trabalho e PC's. O ambiente da PVM é constituído de uma biblioteca de rotinas com interface para C e FORTRAN, cuja principal função é a troca de mensagens entre processos.

A.3 - Ambientes de Desenvolvimento

É aceito na comunidade de processamento paralelo que a paralelização automática de programas é um processo complexo. Uma abordagem mais apropriada para a paralelização de programas envolve uma combinação de intuição humana, análise automática e ferramentas de transformação de código. Nessa categoria situam-se as ferramentas que oferecem recursos para facilitar o desenvolvimento dos programas paralelos.

O HeNCE [15,16] é ambiente integrado para ajudar no desenvolvimento de programas paralelos que são executados sobre uma rede de computadores. HENCE é composto por ferramentas gráficas integradas para criar, compilar, executar, e analisar os programas fontes.

B - Ferramentas para Paralelização de Programas Seqüenciais

Nesta classe estão as ferramentas que fazem parte do processo de paralelização de programas seqüenciais. Tal paralelização consiste na transformação de um código seqüencial já existente em um código de mesma função, mas que possa ser executado em paralelo. Esse processo de paralelização pode ser automático ou interativo.

B.1 - Ferramenta para Paralelização Automática

As ferramentas de paralelização automática buscam de forma automática, ou seja, sem a intervenção do usuário, paralelizar o código fonte. Para que tal ação possa ser efetuada a ferramenta recebe o código fonte formatado, as dependências existentes e através do uso de heurísticas de paralelização gera a versão paralela correspondente do código seqüencial.

B.1.1 - Compiladores Paralelizadores

Atualmente um dos meios mais atrativos de usar paralelismo é escondendo-o do programador. Isto se torna possível através do uso de compiladores paralelizadores, esses buscam paralelizar o código automaticamente recebendo como entrada o código fonte seqüencial e gerando diretamente um código paralelo executável. Somente fica a cargo do programador escrever ou re-usar um código seqüencial.

O POLARIS [140] é um compilador paralelizador Fortran para aplicações industriais, que utiliza processadores maciçamente paralelos com um espaço de endereçamento global. Esse compilador tem como entrada um programa escrito em Fortran 77 gerando um código executável paralelo como saída.

B.1.2 - Pré-processadores Paralelizadores

Os Pré-processadores paralelizadores fazem parte de uma outra categoria de ferramentas de paralelização automática. Muito parecida com os compiladores paralelizadores, buscam paralelizar automaticamente o código, sem a intervenção do usuário. No entanto, esta categoria difere dos compiladores paralelizadores por receber como entrada um código fonte sequencial e gerar um código fonte paralelo, podendo ter seu código ainda alterado pelo programador antes da geração do código executável final.

A ADAPTOR [1,2] é uma ferramenta que transforma programas com paralelismo de dados, escritos em Fortran, com extensões de vetores, *loops* paralelos e diretivas de layout/distribuição de dados, para programas paralelos com passagem de mensagens explícitas.

B.2 - Ferramentas para Paralelização Interativa

Uma abordagem mais apropriada para a paralelização de programas envolve uma combinação de intuição humana, análise automática e ferramentas de transformação de código. Nessa categoria estão incluídas as ferramentas para paralelização de programas sequenciais que trabalham de forma interativa, baseando-se nas opções de paralelização definidas pelo usuário.

O FORGE [12] é um ambiente integrado para análise e instrumentação de programas Fortran 77. Dispõe de ferramentas de auxílio a paralelização semi-automática para arquiteturas do tipo vetorial, memória compartilhada ou memória distribuída.

C - Ferramentas de Análise de Desempenho e Depuração

Esta categoria de ferramentas está ligada com a análise dos programas paralelos gerados, criando recursos de depuração dos códigos fontes paralelos e informações

sobre a execução e o funcionamento dos programas paralelos visando melhorar o seu desempenho.

C.1 - Ferramentas de Depuração

Intimamente ligadas às ferramentas de desenvolvimento estão as de depuração, que são ferramentas para a determinação e correção de erros. Essas ferramentas podem ser divididas em duas categorias: as ferramentas interativas e as *post-mortem*. Os depuradores interativos são mais poderosos e comuns no desenvolvimento de aplicações sequenciais, permitindo que a execução se processe passo a passo, isto é, seja suspensa periodicamente para que as variáveis possam ser inspecionadas ou mesmo modificadas. Depuradores *post-mortem* operam depois do término da execução da aplicação, normalmente após uma condição de erro ter sido detectada.

O TOTALVIEW [127] é um depurador multi-processos para programas paralelos, com interface gráfica para controle do programa e definição dos *breakpoints*. Pode ser utilizado para programas em C, C++ ou Fortran.

C.2 - Ferramentas de Análise de Desempenho

Essas ferramentas fornecem mecanismos para o monitoramento da execução de programas paralelos, gerando uma grande variedade de informações que podem ser analisadas através de utilitários para a sua visualização. O principal objetivo consiste na extração de informações a respeito dos programas que permitam ao usuário alterar o código de forma a obter ganhos desempenho.

O PABLO [109] é um ambiente para análise de desempenho de programas paralelos desenvolvidos na linguagem C que possui uma interface gráfica e uma biblioteca de documentação que pode ser utilizada para descrever processos e tipos de dados.

D - Ambientes Integrados

Essa categoria de ferramentas engloba ambientes que são compostos por um conjunto de ferramentas necessárias para o desenvolvimento, depuração, análise de desempenho, otimização e testes de programas paralelos.

Paralex [158] é um ambiente de programação completo que faz uso intensivo de gráficos para definir, editar, executar e depurar aplicações científicas paralelas. Todo o código necessário para distribuir a computação e replicá-la, como também o balanceamento dinâmico de carga são gerados automaticamente pelo sistema. Apesar do sistema oferecer uma série de recursos para computação paralela o Paralex não dispõe de ferramentas para a paralelização de programas sequenciais

3.3 – Tabela de Classificação das Ferramentas Pesquisadas

A tabela 3.1, apresenta uma relação das ferramentas descritas segundo a classificação proposta no item 3.2.

Tabela 3.1 – Ferramentas para Desenvolvimento de Programas Paralelos

Linguagens	<i>Intrinsecamente paralelas</i>	LINDA OCCAM SISAL SR
	<i>Orientadas a Objeto</i>	C** CC++ CHARM++ COOL - Concurrent Object Oriented Language HPC++ MELDC MENTAT Object Oriented FORTRAN PARA++ Parallel Java PARC++ PC++ - Parallel C++ Pssather

	uC++
<i>Derivadas da Linguagem C</i>	CARPET - Cellular Programming Environment CHARM CID CPAR DINO - Distributed Numerically Oriented Language FORK95 HYPERTOOL IPC - Implicitly Parallel C JADE PCN - Program Composition Notation PCP – Parallel C Preprocessor P-Languages – PC X3H5
<i>Derivadas da Linguagem FORTRAN</i>	Force FORTRAN 90 Fortran M Grids HPF – High Performance Fortran OPUS PDDP – Parallel Data Distribution Preprocessor PFP – Parallel C Preprocessor P-Languages – Pfortran X3H5
<i>Outras Linguagens</i>	ABCL/ST (Statically Typed) Dagger MODULA 2* PARALLAXIS STRAND88
Bibliotecas para computação paralela	ACE - Adaptive Communication Environment ADSMITH AFAPI - Aggregate Function Application Program Interface APPL - Application Portable Parallel Library BlockSolve 95 Brazos C4 – Canonical Classes For Concurrency Control CANOPY Chaos Chaos++ CHIMP – Common High-level Interface to Message Passing

	CM – Communications Manager CPS – Cooperative Processes Software CVM - Coherent Virtual Machine DAPPLE - DAta-Parallel Programming Library for Education DREAM - Distributed REgions And shared Memory EXPRESS GENMP ISIS LMPS – Livermore Message Passing System MPI – Message Passing Interface MTASK Nexus OOMPI – Object-Oriented Message Passing Interface P4 – Portable Programs for Parallel Processors PARMACS PARTI – Parallel Automated Runtime Toolkit at ICASE PHOSPHORUS PICL POM - Parallel Observable Machine PUL – Parallel Utilities Library PVM – Parallel Virtual Machine Regis SCIDDLE SIMPLE SPPL - Stuttgart Parallel Processing Library TCGMSG – Theoretical Chemistry Group Message Passing System Tempest TreadMarks Unify
Ambientes de Desenvolvimento	CODE 2.0 DagTool HeNCE – Heterogeneous Network Computing Environment LPB – Linda Program Builder PYRROS VPE – Visual Parallel Environment

Tabela 3.2 – Ferramentas para Paralelização de Programas Sequenciais

Automáticas	<i>Pré- Processadores Paralelizadores</i>	ADAPTOR – Automatic DATA Parallelism TranlatOR TINPAR – TINY PARallelizer FTS – Forth Translation System MIPRAC PARCEL
	<i>Compiladores Paralelizadores</i>	POLARIS SUIF Compiler – Stanford University Intermediate Format Compiler CRAY/fpp KAP Vienna Fortran Compilation System
Interativas		FORGE P2D PAT – Parallelization Assistant Tool PTOPP – Practical Toolset for the Optimization of Parallel Programs TINY

Tabela 3.3 – Ferramentas para Depuração e Análise de Desempenho

Ferramentas de Depuração	EXECDIFF IPD PANORAMA P2D2 TotalView UBD – KSR Symbolic Debugger XAB XPDB – X-Windows Parallel Debugger
Ferramentas de Análise de Desempenho	AIMS – Automated Instrumentation And Monitoring System ATEXPERT BBN Performance Tools BLOBS FALCON GPMS – General Parallel Monitoring System INTEL Performance Analysis Tools IPS-2 KSR Performance Tools MARITXU

	NTV - NAS Trace Visualizer PABLO Performance Analysis Environment Paradyn PARAGRAPH Projections SVMView TMON- Transputer MONitor UPSHOT VOYER XPVM
--	---

Tabela 3.4 – Ambientes Integrados

Ambientes Integrados	ANNAI CAPER – Concurrent Application Programming Environment ENTERPRISE EPPP – Environment for Portable Parallel Programming MEANDER Modula-2* System MPPE – MasPar Programming Environment PADE – Parallel Application Development Environment PARALEX PARALLAX ParaScope PARASPHERE PEMPI – Programming Environment for the MPI PRISM SCHEDULE TOPSYS – TOols for Parallel Systems TRAPPER VISAGE – VISual Attributed Graph Environment
---------------------------------------	--

Tabela 3.5 - Outras Ferramentas para Processamento Paralelo

Outras Ferramentas	ARJUNA Calypso CCS - Computing Center Software CONDOR CONVERSE DBC - Distributed Batch Controller DJM - Distributed Job Manager DOME - Distributed Object Migration Environment DQS - Distributed Queuing System FUNNEL Hector IMPROV - Integrated Manipulation of PROgram Visualization NetSolve Network Linda PARADISE - PARAllel Animated Debugging and Simulation Environ PARAPET - PARAllel Programming Environment Tools PARIX - PARAllel extensions for uniX POLKA SAGE ++ TOP/DOMDEC - DOMain DEComposition
-------------------------------	--

3.4 - Considerações Finais

Existem diversas ferramentas e ambientes para auxílio no desenvolvimento de aplicações paralelas. Muitas dessas ferramentas foram descritas no Capítulo 2. Neste Capítulo foi proposto um novo esquema de classificação para as ferramentas de processamento paralelo. Essa classificação é mais abrangente que as classificações encontradas nas referências pesquisadas e permite uma classificação mais precisa das ferramentas.

No próximo Capítulo é introduzido o sistema integrado para auxílio à programação paralela proposto neste trabalho.

Capítulo 4

4 - ESPECIFICAÇÃO E PROJETO DO SISTEMA

4.1 - Considerações Iniciais

No Capítulo 2 foram apresentados os conceitos e ferramentas para programação paralela que devem ser considerados na concepção de um ambiente integrado para desenvolvimento de aplicações paralelas. As ferramentas e modelos apresentados forneceram os subsídios necessários para a especificação do sistema proposto, que é o tema alvo deste Capítulo.

A Seção 4.2 apresenta os objetivos gerais do sistema proposto. Na Seção 4.3 está a descrição do modelo de software. A Seção 4.4 aborda a especificação inicial do sistema proposto e na Seção 4.5 é apresentado o projeto final do sistema, ficando na Seção 4.6 as considerações finais deste Capítulo.

4.2 - Objetivos Gerais do Sistema

Fica claro, a partir da análise das ferramentas existentes para programação paralela, a necessidade de uma ferramenta que atenda os requisitos de programação de uma maneira completa e integrada.

Na definição desse ambiente serão consideradas as características mais importantes das ferramentas existentes e características adicionais que tornem o ambiente compatível com o estado da arte no desenvolvimento de programas.

4.3 - Modelo de Software do Sistema

O diagrama de contexto apresentado na figura 3.1 ilustra o modelo adotado para o sistema.

Figura 4.1: Diagrama de Contexto do Ambiente Integrado

Os seguintes aspectos definem sua operação:

- O usuário pode fornecer ao sistema um código fonte sequencial escrito na linguagem C ou Fortran, ou ainda especificações para a modelagem de um projeto baseado em uma metodologia orientada a objetos;
- A ferramenta de análise e projeto recebe do usuário as especificações do projeto, e baseado em uma metodologia de desenvolvimento orientado a objetos, é produzido um código paralelo;

..... de paralelização recebe o código fonte sequencial do usuário, o analisa e modifica, de forma a produzir um código paralelo;

- A ferramenta de depuração e análise de desempenho recebe o código paralelo gerado, o analisa devolvendo ao usuário informações sobre a depuração e o desempenho;

4.4 - Especificação do Sistema

No projeto de um ambiente integrado para desenvolvimento de aplicações paralelas, é importante uma prévia determinação dos requisitos que se fazem necessários para a especificação inicial do projeto. Nas seções seguintes, são descritos os requisitos funcionais, não funcionais e de integração para o sistema proposto.

4.4.1 - Requisitos Funcionais do Sistema

Tendo como base a motivação e os objetivos apresentados na Seção 4.2 e a revisão das ferramentas existentes constante no Capítulo 2, é fundamental que sejam obedecidos os seguintes requisitos funcionais para o sistema proposto:

- Possibilitar a paralelização automática e interativa de programas sequenciais escritos em C ou Fortran, para multicomputadores e multiprocessadores;
- Fornecer ferramentas de especificação e projeto de programas paralelos com uma metodologia orientada a objetos;
- Dispor de ferramentas de codificação em C e Fortran para multicomputadores e multiprocessadores;
- Fornecer ferramentas para visualização e depuração;
- Fornecer ferramentas de monitoramento e análise de desempenho;
- Apresentar um ambiente interativo, com uma interface amigável, de forma a atingir um elevado índice de usabilidade;

- Prover o usuário especialista de todas as informações sobre as dependências de dados e desempenho, de forma que o mesmo possa participar do processo de otimização do código;
- Permitir a geração automática e interativa de código paralelo, tornando sempre que possível, transparente ao usuário o paralelismo envolvido;
- Ser desenvolvido sobre o sistema operacional Linux, aumentando a portabilidade do sistema.

4.4.2 - Requisitos não Funcionais do Sistema

A seguir são apresentados os requisitos não funcionais do sistema.

- O sistema deve permitir a interação do usuário no processo de paralelização, informando-o de dependências de dados que são obstáculos a paralelização e também tornando possível que o próprio usuário fragmente e mapeie os grãos nos processadores, tal qual o proposto no trabalho de Callahan [171] sobre o ParaScope.
- Permitir a utilização integrada dos compiladores f77, g77 e f2c em conjunto com o cc ou gcc;
- Utilizar o modelo de programação paralela promovido pelo PVM, melhorando as características de interoperabilidade, padronização e portabilidade do sistema;
- Apresentar uma interface gráfica baseada no X-Windows [172] e Xview [173], dispondo de um editor de textos para código fonte. Isso com o objetivo de incrementar os atributos de usabilidade;
- Possibilitar a utilização da ferramenta XPVM em conjunto com o sistema;
- Ser um sistema estruturado em módulos com elevado grau de desacoplamento, facilitando a substituição ou a inclusão de recursos no mesmo, provendo-se a característica de extensibilidade.

4.4.3 - Requisitos de Integração do Sistema

O sistema proposto combina uma variedade de diferentes ferramentas de diferentes itens de informação de uma forma que possibilita o fechamento da comunicação entre as ferramentas ao longo do processo de desenvolvimento de aplicações paralelas. As ferramentas são integradas para que as informações do projeto do usuário tornem-se disponíveis para cada ferramenta que delas precise. Para definir “integração” no contexto do sistema proposto, é preciso estabelecer um conjunto de requisitos para que o sistema seja realmente integrado. Um sistema integrado deve:

- Oferecer um mecanismo para compartilhar as informações do desenvolvimento do software entre todas as ferramentas contidas no ambiente;
- Permitir que uma mudança efetuada num item de informação seja rastreada até outros itens de informação relacionados;
- Oferecer gerenciamento de configuração global para todas as informações de engenharia de software.
- Permitir acesso direto, não-sequencial, a qualquer ferramenta contida no ambiente.
- Estabelecer suporte automatizado para um contexto procedimental de trabalho de desenvolvimento de software que integre as ferramentas e os dados numa estrutura de divisão de trabalho padrão.
- Possibilitar que os usuários de cada ferramenta experimentem uma visão e uma percepção consistentes em nível de interface homem/máquina.
- Apoiar a comunicação entre os usuários do sistema.
- Coletar tanto métricas administrativas como técnicas que possam ser usadas para melhorar o processo e o produto.

4.5 - Projeto do Sistema

O projeto do sistema segue a linha da proposta da classificação das ferramentas, projetando um ambiente integrado composto por três grandes classes de ferramentas para programação paralela, visando especificar melhor as funções das ferramentas para programação paralela. A primeira classe categoriza as ferramentas existentes para o processo de desenvolvimento de novos programas paralelos. A segunda classe categoriza-se pelas ferramentas que de alguma forma trabalham com a paralelização de programas sequenciais. Na terceira classe estão as ferramentas de análise, podendo ser esta análise de desempenho, otimização ou de depuração dos programas paralelos gerados.

4.5.1 - Diagrama de Contexto do Sistema

Com base nos objetivos principais do projeto do sistema, pode-se gerar um primeiro esboço de alto nível para o projeto do sistema, conforme figura 4.1.

Figura 4.2: Primeiro esboço do sistema proposto

O primeiro esboço do sistema tem por objetivo principal a especificação das principais entradas e saídas do sistema. O usuário pode fornecer especificações de análise e projeto ao ambiente. O ambiente utiliza estas informações e com base em uma metodologia para desenvolvimento de programas paralelos fornece os recursos necessários para o desenvolvimento do projeto lógico do sistema, em etapas posteriores ferramentas adicionais utilizam o projeto lógico para o desenvolvimento automático ou semi-automático do projeto físico do sistema.

O código sequencial faz parte de uma entrada bastante importante para o sistema, visto que um dos objetivos principais do ambiente é a paralelização de programas sequenciais. O código fonte é fornecido pelo usuário e ferramentas de paralelização específicas podem gerar o código paralelo correspondente de forma interativa ou automática.

O usuário pode ainda fazer uso de ferramentas independentes para análise de desempenho, otimização e depuração dos programas paralelos gerados pelo ambiente proposto.

O código paralelo gerado e as informações de análise de desempenho dos programas paralelos são os principais dados de saída do sistema, porém informações sobre o desenvolvimento do projeto também podem ser consultadas.

Como descrito na Seção 4.2 o projeto do sistema segue a linha da proposta de classificação das ferramentas dividindo as ferramentas para programação paralela em três grandes classes, desta forma podemos especificar um segundo esboço para o diagrama do projeto composto por três grandes módulos, conforme figura 4.2.

Figura 4.3: Segundo esboço do sistema proposto

O primeiro módulo representa um conjunto de ferramentas capaz de fornecer os recursos necessários para a análise e projeto de programas paralelos. O segundo módulo representa um conjunto de ferramentas responsáveis pela paralelização de programas sequenciais. O ultimo módulo como demonstra a figura 4.2 tem o objetivo de analisar o código paralelo gerado e extrair informações de desempenho e depuração

A figura 4.3 representa o detalhamento do projeto final do sistema, contendo os principais módulos que se fazem necessários para o desenvolvimento de um ambiente de programação paralelo completo e integrado.

4.5.2 - Descrição dos Módulos do Sistema

O diagrama de contexto apresentado na Seção anterior define os módulos necessários para o projeto de um ambiente de programação paralela realmente integrado, cobrindo de forma completa as fases de desenvolvimento de novos programas paralelos, a paralelização de programas sequenciais e análise de desempenho e depuração dos programas gerados pela ferramenta. A seguir é dada uma descrição de todos os módulos que fazem parte do sistema proposto.

□ *Modelador de Elementos*

Em UML [174] um elemento é definido como uma estrutura. A definição formal de um elemento é feita com o significado exato do que ele representa sem definições duvidosas ou ambíguas. Sua representação gráfica que é mostrada nos diagramas de comportamento. Um elemento pode existir em diversos tipos de diagramas, mas existem regras que definem quais elementos podem ser mostrados em que tipos de diagramas.

Figura 4.5: Modelador de Elementos

Essa ferramenta deve fornecer os recursos necessários para uma prototipação rápida e eficaz dos elementos envolvidos. Alguns exemplos de modelos de elementos são as classes, objetos, estados, pacotes e componentes. Os relacionamentos também são modelos de elementos, e são usados para conectar outros modelos de elementos entre si.

No contexto do projeto, o modelador de elementos (Figura 4.4) recebe como entrada as definições dos elementos que farão parte do sistema, gerando como saída dados dos elementos criados, como estruturas de dados envolvidas e as dependências existentes entre eles.

❑ *Diagramador de comportamento*

Todos os sistemas possuem uma estrutura estática e um comportamento dinâmico. A UML engloba modelos estáticos (estrutura estática), dinâmicos (comportamento dinâmico) e funcionais. A modelagem estática é expressa pelo diagrama de classes e de objetos, que consiste nas classes e seus relacionamentos. Os relacionamentos podem ser de associações, herança (generalização), dependência ou refinamentos. Os modelamentos dinâmicos são expressos pelos diagramas de estado, sequência, colaboração e atividade. Finalmente, o modelamento funcional é expresso pelos diagramas de componentes e execução.

Figura 4.6: Diagramador de Comportamento

O diagramador de comportamento recebe do modelador de elementos os elementos que foram construídos e modelados para o sistema, recebe ainda, do usuário as opções de diagramação para formar diagramas que representam o comportamento dos elementos do sistema (Figura 4.5). Esse módulo fornece como saída o projeto lógico do sistema a ser desenvolvido, projeto esse representado por diagramas que envolvem os elementos e os eventos de comportamento a eles associados.

❑ *Particionador de código*

O particionador de código recebe do diagramador de comportamento dados referentes ao projeto lógico do sistema e é responsável pelo primeiro passo do desenvolvimento técnico do sistema. O particionador de código analisa os diagramas de comportamento dos elementos procurando dependências para gerar uma estrutura de

vários procedimentos que poderão ser implementados e executados em paralelo (Figura 4.6).

Figura 4.7: Particionador de Código/Objeto

O resultado da operação do Particionador de Código é caracterizado por uma estrutura formada por um esqueleto de programa contendo apenas os cabeçalhos de todos os procedimentos que devem ser implementados para tradução do projeto lógico para o projeto físico.

□ *Codificador*

O Codificador é o módulo responsável pela implementação real do sistema. Esse módulo pode ser dividido em dois tipos de processo. O processo manual de codificação exige um editor de texto para que seja feito o desenvolvimento dos métodos definidos nos módulos anteriores. Uma ferramenta de geração de código automática, também chamada de gerador de programas, pode realizar a geração de código de forma automática com base nas especificações definidas pelos módulos anteriores.

O codificador recebe um esqueleto de programa já particionado como entrada e as opções de geração de código definidas pelo usuário, no caso de geração automática do código (Figura 4.7).

Figura 4.8: Codificador

□ *Reformatador de código seqüencial*

O programa seqüencial do usuário pode ser composto por um ou mais arquivos contendo código produzido pelo usuário que será o alvo do processo de paralelização. No momento da definição dos arquivos que integram o projeto, cada arquivo inserido passa pela função de reformatação. Essa função recebe como entrada o código fonte do usuário e produz como saída o código fonte formatado (Figura 4.8). Nesse processo, o código fonte será indentado por meio da inserção de espaços em branco no texto em função da hierarquia das instruções de controle. Tal ação visa o incremento da legibilidade do código fonte.

Figura 4.9: Reformatador de Código Seqüencial

Numa Segunda etapa o arquivo, ou os arquivos, contendo o código fonte serão fragmentados em vários outros arquivos onde cada um conterá apenas um dos módulos do programa do usuário, ou seja, cada arquivo conterá apenas uma função, sub-rotina ou o programa principal. Tal ação visa acelerar o processo de compilação uma vez que, através do uso de “makefiles”, apenas os módulos modificados serão re-compilados.

□ *Analizador de código seqüencial*

Dentro do sistema uma das funções mais importantes é o analisador de código. Esse atua sobre o código fonte formatado (Figura 4.9) e dele extrai todas as informações pertinentes aos dados e módulos, armazenando essas informações na base de dados do sistema. Essas informações são:

- Módulos: quais são, onde são definidos e onde são chamados todos os módulos presentes no código;
- Variáveis: quais são, as linhas onde são utilizadas, definidas e o tipo de utilização;
- Tabela de referência: para cada linha do código são determinadas todas as variáveis utilizadas na mesma e o tipo de utilização. Também é definido se a linha pertence ou não a uma estrutura de controle;
- Dependências: todas as dependências de dados e de controle existentes entre as linhas do código, gerando um grafo de dependências do programa. Esse grafo de dependências é utilizado na paralelização do código original.

Figura 4.10: Analisador de Código Seqüencial

❑ *Instrumentador de paralelização*

O módulo instrumentador de paralelização recebe como entradas o código fonte formatado e as informações sobre os módulos existentes, gerando uma nova versão do código onde antes e depois de cada chamada a um módulo são inseridas chamadas às funções de temporização (Figura 4.10). Essas funções possibilitam a coleta de dados do tempo gasto com cada chamada e o número de vezes que a mesma foi efetuada.

Figura 4.11: Instrumentador de Paralelização

❑ *Paralelizador interativo*

O paralelizador interativo atua como um pré-processador. Com base no código sequencial instrumentado e nas informações de dependências o paralelizador interage com o usuário oferecendo opções de construção do código paralelo (Figura 4.11). Tais opções são avaliadas pelo usuário que direciona a solução desejada para cada evento de paralelização.

Figura 4.12: Paralelizador Interativo

□ *Paralelizador Automático*

O paralelizador automático busca de forma automática, ou seja, sem a intervenção do usuário, paralelizar o código fonte. Para que tal ação possa ser efetuada o paralelizador recebe o código fonte formatado, as dependências existentes e através do uso de heurísticas de paralelização, é gerada uma versão paralela do código (Figura 4.12). A versão paralela do código é obtida da fragmentação do código original em grãos e pela inserção de primitivas necessárias para sincronização ou comunicação.

Figura 4.13: Paralelizador Automático

□ *Compilador*

O módulo compilador é responsável pela tradução do código fonte dos programas gerados em código de máquina. Este módulo pode ser composto por um ou mais compiladores, ficando a cargo do usuário a configuração para a utilização dos mesmos. Em princípio, devem ser utilizados compiladores para as linguagens C e Fortran.

O módulo compilador pode receber o código fonte paralelo com ou sem instrumentação, como mostra a figura 4.13. O código paralelo é instrumentado com a utilização de funções para análise de desempenho e depuração.

Figura 4.14: Compilador

□ *Instrumentador de Desempenho*

O Instrumentador de desempenho recebe o código fonte paralelo, faz as alterações necessárias para a coleta de dados e passa o código paralelo instrumentado para o compilador como mostra a figura 4.14.

As funções para a coleta de dados inseridas no código visam a determinação do seu desempenho para posterior análise e otimização.

Figura 4.15: Instrumentador de Desempenho

□ *Executor*

O módulo Executor recebe o código paralelo executável instrumentado e ativa uma sequência de operações de simulação com a finalidade de extrair uma grande quantidade de dados sobre a execução do programa paralelo (Figura 4.15). Os dados extraídos nessa operação são chamados de dados de rastreo e são armazenados em uma base de dados para uma posterior avaliação das estatísticas de desempenho do programa.

Figura 4.16: Executor

□ *Analizador de desempenho em tempo de execução*

Esta ferramenta permite a análise do desempenho do programa paralelo em tempo de execução. Ela interage com as funções de instrumentação do código para a obtenção dos dados de desempenho (Figura 4.16) que podem ser transformados em informações importantes e transferidos de forma imediata para o usuário, ou armazenados em uma base de dados contendo todos os dados de rastreo da execução do programa.

Figura 4.17: Analisador em Tempo de Execução

□ *Analizador de desempenho “post-mortem”*

O analisador de desempenho post-mortem tem por objetivo gerar informações de desempenho, após uma ou várias execuções do programa paralelo final.

Figura 4.18: Analisador Post-Mortem

O módulo recebe da base de dados os dados de rastreio (Figura 4.17), contendo todas as informações registradas nas execuções do aplicativo. Estatísticas de execução são geradas representando a média de desempenho do programa.

□ *Depurador*

O depurador é a ferramenta responsável pela análise e interpretação do código fonte gerado ou escrito pelo usuário da ferramenta. Esse módulo deve ser capaz de detectar o maior número possível de erros e incorreções passíveis de ocorrência durante o processo de construção de um programa paralelo. A ferramenta deve ainda, além da capacidade de detecção de erros, fornecer os subsídios necessários para a correção automática ou semi-automática dos problemas detectados.

O módulo Depurador pode receber o código fonte paralelo ou ainda o código executável para o processo de depuração (Figura 4.18), o usuário pode interagir com a ferramenta através das opções de depuração ou correção que a ferramenta pode oferecer.

Figura 4.18: Depurador

4.6 - Considerações Finais

O enfoque principal deste trabalho está centrado na concepção das funções necessárias para o projeto de um ambiente completo e integrado para o desenvolvimento de aplicações paralelas e na avaliação de possíveis ferramentas existentes que podem fazer parte deste ambiente.

Neste Capítulo foram apresentados os requisitos a serem atingidos, a especificação e o projeto básico do sistema proposto, descrevendo-se os principais módulos que se fazem necessários para a construção desse ambiente.

No próximo Capítulo, são analisadas algumas ferramentas disponíveis para programação paralela e como essas ferramentas podem ser utilizadas na implementação do ambiente proposto, permitindo uma redução substancial no seu tempo de desenvolvimento.

Capítulo 5

5 - UTILIZAÇÃO DAS FERRAMENTAS EXISTENTES

5.1 - Considerações Iniciais

No Capítulo 2 foi realizada uma pesquisa extensa sobre as ferramentas para programação paralela disponíveis. As informações levantadas nessa pesquisa fornecem uma idéia geral das principais características das ferramentas para programação paralela existentes que serviram de base para a especificação e o projeto do ambiente proposto, apresentados no Capítulo anterior. Neste Capítulo são revistas algumas dessas ferramentas que apresentam características apropriadas para fazer parte do ambiente de programação proposto, com ou sem alteração do seu código fonte.

A Seção 5.2 apresenta uma descrição detalhada das ferramentas selecionadas que podem, a princípio, fazer parte do projeto do ambiente proposto. A Seção 5.3 apresenta a proposta de integração das ferramentas selecionadas, ficando na Seção 5.4 as considerações finais deste Capítulo.

5.2 - Seleção das Ferramentas

Com base no estudo detalhado feito sobre as ferramentas que fizeram parte da pesquisa, pode-se fazer a seleção de algumas dessas ferramentas que podem ser úteis para o desenvolvimento do projeto proposto.

A utilização de ferramentas já existentes para programação paralela é de grande importância para este projeto no que se refere a otimização do seu tempo de desenvolvimento. A integração de várias ferramentas exige, entretanto, que essas possuam algumas características em comum, tornando a integração possível e adequada.

O Capítulo 2 apresentou uma avaliação das principais características de todas as ferramentas que fizeram parte da pesquisa e um resumo desse estudo foi apresentado na tabela 2.1. Desta tabela, selecionou-se ferramentas que atendem as seguintes características consideradas importantes para o ambiente proposto:

- Disponibilidade de código fonte;
- Possibilidade de execução no sistema operacional UNIX;
- Linguagem de implementação C ou C++;
- Suporte para programação paralela na linguagem C e Fortran;
- Interface gráfica compatível com o UNIX (X-Windows, Tcl/Tk).

As bibliotecas de passagem de mensagens são ferramentas de grande importância para qualquer projeto de programação paralela, fornecendo os mecanismos básicos para a comunicação entre processos paralelos. Nessa classe, as ferramentas que mais se destacam são a PVM e a MPI.

Os paralelizadores de programas sequenciais são ferramentas poderosas e de difícil implementação, o P2D é uma ferramenta dessa classe que se adequa muito bem ao projeto proposto.

O PANORAMA é uma ferramenta de depuração que cobre as necessidades de depuração do sistema proposto e o PABLO é uma ferramenta que pode ser utilizada nos módulos de análise de desempenho.

No contexto da pesquisa realizada, notou-se uma grande carência de ferramentas para o desenvolvimento de aplicações paralelas baseadas em metodologias orientadas a objetos, principalmente de domínio público. O HENCE é um ambiente de desenvolvimento de aplicações paralelas, que embora não possua uma metodologia de desenvolvimento orientada a objetos, cobre de forma adequada os módulos de desenvolvimento do sistema proposto.

A seguir é dada uma descrição mais detalhada dessas ferramentas e como elas podem ser encaixadas no projeto do ambiente proposto.

5.2.1 - PVM (*Parallel Virtual Machine*)

O sistema PVM [31, 32] foi proposto visando o desenvolvimento de programas paralelos e distribuídos para serem executados em um conjunto heterogêneo de computadores interligados por uma rede de comunicação. É composto basicamente por dois componentes: um *daemon* chamado *pvm* e uma biblioteca de rotinas que estabelece a interface entre o programa do usuário e o sistema.

O processamento paralelo é tratado pelo PVM como uma coleção de processos que se comunicam através de mensagens. Essas mensagens são transmitidas utilizando-se o protocolo UDP/TCP, no caso de redes de computadores, ou qualquer outro mecanismo de comunicação disponível. Para o envio e recebimento das mensagens são utilizados *buffers* dinâmicos, de tal forma que o tamanho das mensagens é limitado apenas pela disponibilidade de memória.

O núcleo do modelo utilizado pelo PVM é uma máquina paralela virtual. Essa máquina virtual é composta por um conjunto de máquinas físicas interligadas, sendo utilizada pelo usuário como uma máquina paralela fracamente acoplada onde os programas paralelos são executados. A máquina virtual é composta por computadores interligados e gerenciados por uma coleção de processos *daemons* disparados por um mesmo usuário.

Para que um computador possa fazer parte desta máquina virtual, é necessário que o processo *daemon* esteja sendo executado por ele. O *daemon* pode ser iniciado na console da máquina pelo usuário ou por um programa que já esteja sendo executado na máquina paralela virtual.

Para que um programa possa ser executado nessa máquina virtual, é necessário que ele se registre com o *daemon* *pvm* da estação correspondente, através da chamada de uma função da biblioteca do PVM. Após ser registrado, ele pode utilizar as várias funções que o PVM oferece, tais como iniciar processos em outros computadores, verificar o estado dos processos que estão sendo executados na máquina virtual, enviar e receber mensagens de modo bloqueante ou não-bloqueante, enviar mensagens de difusão múltipla (*broadcast*), executar rotinas do tipo barreira e rotinas globais de comunicação de dados.

Este sistema é de domínio público e pode ser obtido através do endereço <http://www.netlib.org/pvm3/index.html>.

5.2.2 - MPI (*Message-Passing Interface*)

O MPI [55] não é propriamente uma implementação de um sistema, mas sim uma proposta de padronização para a interface de troca de mensagens em máquinas paralelas com memória distribuída. A idéia de se padronizar essa interface é a de possibilitar a portabilidade dos programas entre as diferentes máquinas e facilitar o seu uso.

O padrão MPI não contém nenhuma definição para tolerância a falhas e assume que o ambiente em que será executado é confiável. Ele define uma interface de troca de mensagens e não pretende definir um ambiente completo para programação paralela. Por essa razão, não são tratados tópicos como entrada/saída paralela, estruturação do programa paralelo e depuração.

Também não é definido nenhum mecanismo explícito para *multithreading*, apesar de ser um dos objetivos do projeto do padrão o de garantir que ele possa ser implementado de maneira eficiente em ambientes que possibilitem *multithreading*.

O núcleo do MPI é formado por rotinas para comunicação ponto a ponto entre pares de processos. Essas rotinas podem ser chamadas de forma bloqueante ou não bloqueante. Existem três modos de comunicação: *ready*, no qual a mensagem só pode ser enviada se uma operação de recepção correspondente já tiver sido iniciada; *standard*, no qual a mensagem pode ser enviada independentemente de haver uma operação de recepção para ela; e *synchronous*, que funciona de maneira semelhante ao modo standard, com a diferença de que a operação de envio não será considerada finalizada até que a operação de recepção correspondente seja iniciada no processo de destino.

Outras facilidades definidas são as rotinas para grupos de processos, que visam coordenar a comunicação entre os processos pertencentes a um determinado grupo. Essas rotinas permitem a transmissão coletiva dos dados e oferecem as seguintes funcionalidades:

- Envio de um mesmo dado por um determinado participante do grupo para todos os demais (*broadcast*);
- Envio de dados diferentes por um determinado participante do grupo para todos os demais (*scatter*);
- Envio de dados de todos os participantes do grupo para um determinado participante (*gather*);
- Envio de dados de cada participante para todos os demais (*allgather*);
- Envio de dados diferentes de cada participante para todos os demais (*alltoall*).

Também são definidas operações para a computação de certas funções sobre dados pertencentes aos participantes de um determinado grupo. Essas operações são denominadas redução e varredura. Na operação de redução, uma função pré-definida ou definida pelo usuário é aplicada sobre os dados localizados nos diversos processadores que estão executando os processos pertencentes a um determinado grupo. O resultado dessa função pode ser enviado a todos os participantes do grupo ou a somente para um deles.

Um exemplo dessa operação seria determinar o valor máximo de um conjunto de dados, onde cada elemento desse conjunto está associado a um processo pertencente a um determinado grupo. A operação de varredura executa o mesmo procedimento que a redução, mas sobre subconjuntos dos participantes do grupo.

Uma implementação desse padrão pode ser obtida através de ftp anônimo para o diretório pub/mpi no endereço info.mcs.anl.gov.

5.2.3 - P2D

O P2D [132] é um paralelizador de aplicações seqüenciais Fortran. Seu ambiente procura prover o usuário com ferramentas de auxílio para análise e otimização de programas escritos na linguagem Fortran e ferramentas específicas de auxílio na paralelização desses programas para execução em arquiteturas paralelas.

A realização destas idéias se faz possível através do desenvolvimento de um sistema, que tendo como base o programa original escrito pelo usuário, faz uma análise do código determinando todas as estruturas de dados, fluxos de dados e de execução, e também os tempos envolvidos na execução de cada trecho de código. Com base nesses dados o usuário pode modificar o código buscando um melhor desempenho, ou interagir com um processo de fragmentação semi-automática do código em blocos que possam ser executados em paralelo.

A facilidade de uso do sistema proposto é enfatizada através de uma interface homem-máquina baseada no ambiente X Windows. O sistema foi desenvolvido no ICMC pelo aluno Paulo André como parte do trabalho de mestrado. O código fonte e toda a documentação necessária estão disponíveis no Laboratório de Alto Desempenho do ICMC – USP, de São Carlos. A figura 5.1 apresenta o diagrama de fluxo de dados do sistema.

Para atender completamente a funcionalidade prevista no sistema proposto, é necessário que o P2D seja estendido para incluir ferramentas para o processamento de programas escritos na linguagem C. O módulo de paralelização do P2D se encontra em fase de desenvolvimento e não é adequado para o sistema, exigindo melhorias significativas no seu desempenho e funcionalidade.

Figura 5.1: P2D – Diagrama de Fluxo de Dados do Sistema – Nível 1

No sistema P2D o projeto do usuário é composto por um ou mais arquivos contendo o código fonte produzido pelo usuário. O Reformatador é responsável pela reformatação do código sequencial de entrada. O editor permite ao usuário a facilidade de edição do código. O analisador de código extrai todas as informações pertinentes aos dados e módulos do usuário, armazenando na base de dados do sistema. O Instrumentador gera uma nova versão do código onde são inseridas chamadas à biblioteca de temporização.

O módulo de Paralelização recebe como entrada o código fonte formatado e as informações de dependências existentes gerando um novo código paralelo de forma automática ou interativa. O Executor prepara e ativa a execução do código instrumentado ou do código paralelo, enquanto o Visualizador apresenta ao usuário informações na forma de um gráfico ou relatório.

O sistema se enquadra perfeitamente nos módulos de paralelização do sistema proposto cobrindo uma área de atuação conforme a figura 5.2.

5.2.4 - PANORAMA

Essa ferramenta fornece uma interface de depuração padrão para programas paralelos utilizando um arquivo para a tradução entre os comandos da Panorama e aqueles utilizados pelo depurador nativo da máquina. Para aumentar a portabilidade e extensibilidade do sistema é utilizada a interface Tcl/Tk que se comunica através de um *socket* com o depurador. A Panorama [177,178] teve a linguagem C como linguagem de implementação, podendo ser executada no sistema operacional UNIX.

No Panorama é possível abrir múltiplas janelas para visualizações dos dados e das comunicações. O usuário pode utilizar janelas separadas para visualizar o código fonte referente a cada processador, o controle do programa e a avaliação de expressões. Essas tarefas são realizadas utilizando janelas de texto Tk.

As técnicas de visualização usam primariamente mecanismos gráficos, com botões e manipulação através do *mouse*. As janelas do Panorama operam de forma independente e as informações são atualizadas conforme solicitado pelo usuário.

Além das propriedades de depuração, a Panorama permite o rastreamento da atividade de comunicação para depuração *post-mortem*, fornecendo para cada máquina uma biblioteca de rotinas de rastreamento e macros de pré-processador. A apresentação dos dados acrescenta um aumento entre 5 e 65% no custo de comunicação.

O modelo usado para construir a Panorama resultou em uma plataforma altamente portátil para depuração de programas, com facilidades para incluir novas características na visualização, além de fornecer uma interface padrão com um mínimo de esforço (Figura 5.3).

Figura 5.3: PANORAMA - Diagrama de Fluxo de Dados

O Controlador de Programas é o módulo pelo qual o usuário interage com a ferramenta. Nesse módulo o usuário determina qual ou quais são os programas que serão carregados para o processo de depuração. Após a seleção o código é enviado para o tradutor de comandos que faz uma interpretação do código baseando-se na biblioteca padrão da ferramenta. O código fonte interpretado é dado de entrada no Avaliador de Expressões, responsável por detectar incorreções sintáticas ou semânticas no código avaliado.

O processo de depuração na ferramenta Panorama pode ser acompanhado em tempo de execução, tarefa do Visualizador, módulo que recebe as informações do avaliador de expressões e as apresenta ao usuário. O Rastreador é um componente da ferramenta capaz de realizar uma análise de rastreamento dos programas depurados, apresentando dados de depuração “*post-mortem*”.

A Panorama se enquadra no sistema proposto cobrindo o módulo de depuração conforme demonstrado na figura 5.4

Figura 5.4: Área de atuação da ferramenta PANORAMA no sistema proposto

5.2.5 - PABLO (Performace Analysis Enviroment)

A principal meta no desenvolvimento do Pablo [109] foi a de criar um ambiente portátil de análise de dados de desempenho que pode ser utilizado em uma variedade de sistemas maciçamente paralelos.

O ambiente de análise de desempenho do Pablo consiste em vários componentes independentes que compartilham o formato de dados SDDF (Self-Defining Data Format) como um meio de troca de informações. A SDDF é uma linguagem de descrição dos dados de desempenho.

O componente SvPablo é uma interface gráfica para instrumentação do código fonte e visualização dos dados de desempenho em tempo de execução. Este componente foi desenvolvido para capturar, analisar e apresentar os dados de desempenho das aplicações.

A Trace Library contém uma biblioteca básica para a captura de dados de desempenho e gravação de registros de eventos. Existem extensões construídas sobre essa biblioteca para gravar informações de desempenho sobre chamadas de passagem de mensagens e pedidos de entrada/saída.

O componente I/O Analysis contém programas para produzir relatórios resumidos das atividades de entrada e saída de uma aplicação através dos registros de eventos de entrada e saída de um arquivo de rastreo SDDF gerado pela Trace Library I/O Extension.

O Pablo Performance Analysis Environment possui também um componente chamado Analysis GUI. Esse componente consiste em um sistema de visualização de desempenho que fornece uma ferramenta portátil, escalável e extensível para analisar e mostrar os dados escritos no formato SDDF. A figura 5.5 representa o diagrama de fluxo de dados do sistema.

O Pablo e todos os seus componentes podem ser encontrados no endereço <http://vibes.cs.uiuc.edu/Software/Pablo/pablo.htm>.

Figura 5.5: PABLO - Diagrama de Fluxo de Dados do Sistema

No diagrama de fluxo de dados da ferramenta PABLO são identificados quatro grandes componentes. O Instrumentador gera uma nova versão do código onde são inseridas chamadas à biblioteca de temporização da ferramenta. O código fonte instrumentado é então dado como entrada no Analisador de Comportamento, módulo esse responsável por simular, analisar e capturar os dados de desempenho das aplicações paralelas.

Os dados de desempenho são enviados em tempo de execução para o Visualizador que os organiza e apresenta ao usuário. Esses mesmos dados de desempenho são armazenados em uma base de dados para uma avaliação estatística posterior do desempenho do sistema. O Gerador de Relatórios é o componente responsável por capturar essas informações registradas na base de dados e gerar relatórios comparativos dos dados de desempenho “*post-mortem*”. As áreas cobertas pela ferramenta são mostradas na figura 5.6.

Figura 5.6: Área de atuação da ferramenta PABLO no sistema proposto

5.2.6 - HENCE (HEterogeneous Network Computing Enviromenmt)

O HENCE [15,16] é um ambiente de desenvolvimento de programas paralelos baseado na interface *X-Windows*, projetado para auxiliar cientistas no desenvolvimento de programas paralelos que são executados em uma rede de computadores.

Essa ferramenta foi construída sobre a biblioteca de passagem de mensagens PVM que fornece suporte ao gerenciamento e comunicação entre processos em uma rede heterogênea de estações de trabalho. O HENCE é baseado na descrição de um programa por um grafo, onde os nós do grafo representam as subrotinas e os arcos representam as dependências de dados. Ele é composto de ferramentas gráficas integradas para criar, compilar, executar e analisar programas HENCE.

A filosofia do HENCE impõe ao programador a tarefa de especificar explicitamente o paralelismo da computação e a ferramenta auxilia nas tarefas de desenvolvimento do programa.

Os grafos do programa são especificados pelo programador através de uma interface gráfica *X-Windows* e os procedimentos referentes a cada nó são definidos de uma forma textual, podendo ser escritos na linguagem C ou FORTRAN. Com o grafo e os procedimentos definidos, a ferramenta fornece facilidades para edição e compilação desses procedimentos em várias arquiteturas que foram definidas como a máquina paralela virtual, gerando o código referente a essa arquitetura.

Uma característica importante do HENCE é a possibilidade do usuário especificar múltiplas implementações dos nós HENCE baseadas na arquitetura destino.

A interface gráfica do HENCE, além de ser utilizada para escrever programas, também possibilita a configuração da máquina virtual, visualização de informações de rastreo, além da compilação, execução e análise de programas HENCE. A figura 5.7 representa o diagrama de fluxo de dados em alto nível da ferramenta.

Essa ferramenta foi desenvolvida no Oak Ridge National Laboratory e Universidade do Tennessee e possui distribuição gratuita.

Figura 5.7: HENCE – Diagrama de Fluxo de Dados do Sistema

O construtor é o componente da ferramenta responsável pela descrição do programa por meio de um grafo, onde os nós representam as rotinas que devem ser implementadas. Os arcos do grafo representam as dependências de dados existentes entre os módulos do sistema a ser construído. O Descritor recebe o grafo de dependências do Construtor e interage com o usuário para a desenvolvimento das subrotinas que se fazem necessárias para a implementação dos nós do grafo.

O Modelador é responsável por avaliar o código fonte até então desenvolvido e validar o paralelismo definido pelo usuário, tais informações são enviadas ao usuário que as analisa e define as alterações necessárias no código paralelo.

A ferramenta possui ainda um conjunto de ferramentas a exemplo do P2D para visualização e análise de desempenho dos programas gerados, a área de atuação da ferramenta HENCE é demonstrada na figura 5.8, ficando na figura 5.9 a demonstração de todas as áreas cobertas pelas ferramentas selecionadas.

Figura 5.9: Áreas cobertas pelas ferramentas

5.3 - Opções de Integração

As ferramentas podem ser integradas em um único ambiente de muitas maneiras diferentes [175]. Na realidade, as ferramentas pesquisadas apresentam módulos duplicados, com funcionamento ligeiramente diferente e uma interface com o usuário que pode ser bastante diferente em alguns casos.

As interfaces gráficas não têm aparência nem modos de uso uniformes. Feitas essas considerações, propõe-se 3 modos para a integração das ferramentas selecionadas com níveis crescentes de complexidade, facilidade de uso e uniformidade na interação com o usuário.

□ Acesso Comum a Ferramentas

O primeiro modo de integração proposto é a utilização de uma interface comum para execução das ferramentas (Figura 5.10), mantendo-se a interface gráfica original da ferramenta. Tal opção é atrativa pois as características originais das ferramentas são preservadas e fica reduzido ao mínimo o esforço de desenvolvimento do sistema proposto. Além disso, novas versões das ferramentas utilizadas ou outras ferramentas podem ser introduzidas no ambiente de forma quase imediata.

Neste ambiente, o intercâmbio de dados entre ferramentas fica minimizado e pode ser feito com o desenvolvimento de um procedimento de tradução. A maioria das ferramentas tem, pelo menos, a capacidade de exportar as informações que coletam e criam na forma de arquivos não estruturados. Essa tradução pode ser automática (disparada automaticamente quando necessário) ou pode ser feita quando solicitada pelo usuário.

Figura 5.10: Integração por acesso a ferramentas comuns

❑ *Gerenciamento de Dados Comum*

Em um segundo modo de integração, os dados das ferramentas podem ser mantidos em um único banco de dados lógico (Figura 5.11), que fisicamente pode ser centralizado ou distribuído. Isso simplifica o intercâmbio de informações e melhora a integridade dos dados compartilhados, uma vez que cada ferramenta sempre tem acesso às últimas informações do projeto.

É necessário o desenvolvimento de procedimentos para realizar o acesso aos dados possibilitando que ferramentas compartilhem uma base de dados comum do ambiente. O formato dessa base de dados é específico do ambiente proposto, possibilitando que a interface comum possa dispor de funções para o gerenciamento e monitoramento das informações armazenadas.

Figura 5.11: Integração de Dados

□ *Integração Plena*

No último modo de integração proposto, tem-se a integração completa do ambiente. Nesse nível, o processo de integração requer que as ferramentas sejam integradas no nível de código fonte, de forma que todas as ferramentas do projeto sejam adaptadas para formar uma única ferramenta, como demonstrado na figura 5.12.

O ambiente deve possuir uma interface integrada para todas as ferramentas, eliminando as interfaces particulares de cada ferramenta. As ferramentas devem ser integradas de forma a compartilharem suas principais funcionalidades sem a necessidade de armazenamento temporário ou procedimentos de tradução, conforme descrito nas propostas anteriores.

Figura 5.12: Integração Plena

Para que tal integração seja implementada é necessário uma análise das funcionalidades das ferramentas, de forma que seja aproveitado apenas os principais módulos de cada ferramenta, com o intuito de otimizar o código final e evitar a redundância de módulos com funcionalidades idênticas no projeto. A tabela 5.1 apresenta um quadro comparativo das ferramentas selecionadas e dos seus principais módulos.

Tabela 5.1: Comparativo entre os principais módulos das ferramentas

Módulo	P21	FENCE	PANORAMA	PABLO
Analisador de código	X	X		
Analisador de comportamento				X
Avaliador de expressões			X	
Consultor		X		
Controlador de programas			X	
Descompilador				
Editor	X			
Executor				
Instrumentador	X			X
Modelador				
Paralelizador	X			
Rastreador			X	X
Reformatador	X			
Tradutor de comandos			X	
Visualizador	X	X	X	X

5.4 - Considerações Finais

Neste Capítulo foram apresentadas as ferramentas existentes para programação paralela que podem ser integradas ao ambiente proposto. Foram discutidos os principais aspectos relacionados com a integração de cada ferramenta, particularmente quanto ao papel que as mesmas podem desempenhar dentro ao ambiente.

Também foram apresentadas três propostas para a implementação do sistema, com níveis crescentes de funcionalidade e complexidade. A primeira dessas propostas tem implementação relativamente simples e deve ser realizada em um próximo trabalho futuro. A última proposta exige o desenvolvimento de uma quantidade de código que não é compatível com o tempo disponível para a realização de um único programa de mestrado.

Capítulo 6

6 - CONCLUSÕES

6.1 - Considerações Iniciais

No Capítulo 2 são apresentados os principais conceitos e trabalhos relacionados com processamento paralelo e uma pequena descrição das principais ferramentas para programação paralela que fizeram parte de uma pesquisa extensa. Fundamentado nesse estudo, no Capítulo 3 é apresentada uma nova proposta de classificação para as ferramentas de programação paralela com a finalidade de uma melhor avaliação das suas características e áreas de abrangência.

A especificação e o projeto do ambiente do sistema proposto são apresentados no Capítulo 4. O Capítulo 5 apresenta algumas ferramentas para programação paralela que podem fazer parte do projeto e especifica o projeto de integração dessas ferramentas em três níveis.

Neste Capítulo são apresentadas as conclusões obtidas após a análise do trabalho desenvolvido e são propostos trabalhos futuros complementares ao esforço até aqui empreendido.

Na Seção 6.2 são sumarizados as contribuições deste trabalho, ficando na Seção 6.3 as extensões propostas para continuação do trabalho, conforme as conclusões obtidas. A Seção 6.4 apresenta alguns comentários finais, avaliando os objetivos propostos, os resultados obtidos e as perspectivas para a construção do ambiente integrado.

6.2 - Contribuições Deste Trabalho

O desenvolvimento de um projeto desse porte resulta em algumas contribuições significativas, ressaltando-se:

- Extensa pesquisa sobre as ferramentas para programação paralela, com a geração de um relatório técnico intitulado “Uma Classificação das Ferramentas para Programação Paralela Disponíveis – Revisão e Disponibilidade,” [176];
- Proposta de uma nova classificação para as ferramentas de programação paralela;
- Projeto de um ambiente realmente integrado para desenvolvimento, paralelização, análise e depuração de aplicações paralelas;
- Avaliação da possibilidade de aproveitamento de algumas ferramentas para programação paralela já existentes no desenvolvimento do ambiente proposto. Essa avaliação propõe um novo modelo para o desenvolvimento dessa classe de sistemas, baseado no aproveitamento modular de sistemas afins com interfaces com o usuário não uniformes.

6.3 - Trabalhos Futuros

Dada a complexidade de implementação do sistema proposto e seu vasto campo de aplicação na área de processamento paralelo, são propostos os seguintes trabalhos em continuidade ao até aqui desenvolvido:

- Implementação de uma interface comum para a integração das ferramentas que farão parte do ambiente integrado, conforme proposta de nível I no Capítulo anterior;

- Adaptação de uma base de dados comum para todas as ferramentas, de forma que todas as ferramentas do ambiente possam compartilhar os dados do projeto, conforme proposta de nível 2 no Capítulo anterior;
- Adaptação das ferramentas existentes que podem fazer parte do projeto com as devidas alterações no código fonte, fazendo-se o aproveitamento no nível modular do código disponível, conforme proposta de nível 3 no Capítulo anterior;
- Manutenção do P2D para incluir o processamento de programas na linguagem C e melhoria no subsistema de paralelização automática e interativa;

6.4 - Considerações Finais

Novas máquinas paralelas têm sido lançadas a cada dia. Contudo, explorar o poder de processamento oferecido por essas novas arquiteturas é uma tarefa árdua e difícil para os usuários. Nesse contexto, surgem as ferramentas dedicadas à paralelização de aplicações sequenciais que buscam, de forma transparente ao usuário, explorar o potencial oferecido por essas arquiteturas.

Fundamentado em uma extensa pesquisa sobre as diversas ferramentas para programação paralela, foi proposto neste trabalho um ambiente integrado para programação paralela, capaz de cobrir todas as fases de desenvolvimento de aplicações paralelas. A utilização de ferramentas de domínio público também foi abordada pelo projeto, demonstrando que é possível a integração de ferramentas existentes para o desenvolvimento de um ambiente integrado.

Os objetivos propostos para o trabalho foram atingidos, mostrando sua viabilidade e sugerindo novos e importantes trabalhos na área.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] BRANDES, T., "Efficient Data Parallel Program without Explicit Message Passing for Distributed Memory Multiprocessors," GMD Technical Report, TR92-4, 1992.
- [2] BRANDES, T., "ADAPTOR: Users guide. Internal Report Adaptor 2," German National Research Institute for Computer Science, October 92. Draft.
- [4] QUEALY, A.; GOLE, G. L. and BLECH, R. A., "Portable Programming on Parallel/Networked Computers Using the Application Portable Parallel Library (APPL)," to be published as a NASA TM, 1993.
- [5] LITZKOW, M., "Remote Unix - turning idle workstations into cycle servers," In Proceedings of the Summer 1987 Usenix Conference, Phoenix, Arizona, June 1987.
- [6] LITZKOW, M. J.; LIVNY, M. and MUTKA, M. W., "Condor - a hunter of idle workstations," In Proceedings of the 8th International Conference on Distributed Computing Systems, pages 104--111, San Jose, CA, June 1988.
- [7] LITZKOW, M. and LIVNY, M., "Experience with the Condor distributed batch system," Computer Sciences Department, University of Wisconsin, Madison, WI.
- [8] ROSING, M.; SCHNABEL, R. B. and WEAVER, R. P., "The DINO parallel programming language," Technical Report CU-CS-457-90, Computer Science Department, University of Colorado at Boulder, April 1990.
- [9] DERBY, T. M.; ESKOW, E.; NEVES, R.; ROSING, M., "DINO 1.0 user's manual. Technical Report CU-CS-501-90," Computer Science Department, University of Colorado at Boulder, April 1990.
- [10] CARLE, A; COOPER, K. D.; HOOD, R. T.; KENNEDY, Y. K.; TORCZON and WARREN, S. K., "A Practical Environment for Scientific Programming," IEEE Computer, Nov. 1987.
- [11] CALLAHAN, C. D.; COOPER, K. D.; HOOD, R. T.; KENNEDY, K. and TORCZON, L., "Parascope: A Parallel Programming Environment," International Journal of Supercomputer Applications, vol. 2 No. 4, pp. 84-99, Winter, 1988.
- [12] California: Applied Parallel Research, FORGE 90 Version 8.0 Baseline System User's Guide, April 1992.

- [13] LU, G. P.; MELAX S.; PARSONS, I.; SCHAEFFER, J.; SMITH, C. and SZAFRON, D., "The Enterprise Model for Developing Distributed Applications," TR 92-20, Department of Computing Science, University of Alberta, 1992.
- [14] SZAFRON, D.; SCHAEFFER, J.; WONG, P. S.; CHAN, E.; LU, G. P. and SMITH Carol, "The Enterprise Distributed Programming Model," Programming Environments for Parallel Computing, pp. 67-76, Elsevier Science Publishers, 1992.
- [15] BEGUELIN, A.; DONGARRA, J.; GEIST, A.; MANCHEK, R.; MOORE, K. and WADE, R., "HeNCE: A user's guide (Draft)," Oak Ridge National Laboratory, November 1991.
- [16] BEGUELIN, A.; DONGARRA, J.; GEIST, A.; MANCHEK, and SUNDERAM V. S., "Graphical development tools for network-based concurrent supercomputing," In Proceedings of Supercomputing 1991, pages 435--444, Albuquerque, 1991.
- [17] GELERNTER, D.; CARRIERO, N.; CHANDRAN, S. and CHANG, S., "Parallel Programming in Linda," Proceedings of the 1985 International Conference on Parallel Processing, pp. 255-263, 1985.
- [18] AHUJA, S.; CARRIERO, N. and GELERNTER, D., "Linda and Friends," IEEE Computer, pp. 26-34, Aug. 1986.
- [19] ISO/IDE, "Information technology - Programming languages - Fortran," International Standard, Reference number ISO/IEC 1539: 1991 (E), 1991.
- [20] APPELBE, B. and SMITH, K., "Start/Pat: A Parallel-Programming Toolkit," IEEE Software, pp. 29-38, July 1989.
- [21] SMITH, K.; APPELBE, B. and STIREWALT, K., "Incremental Dependence Analysis for Interactive Parallelization," ICS, pp. 330-341, June 1990.
- [22] APPELBE, B.; SMITH, K. and STIREWALT, K., "PATCH - A New Algorithm for Rapid Incremental Dependence Analysis," ICS, pp. 424-432, June 1991.
- [23] WOLFE, M., "The Tiny Loop Restructuring Research Tool," Proc. of 1991 International Conference on Parallel Processing, pp. II-46 - II-53, 1991.
- [24] DONGARRA, J. J. and SORENSEN, D. C., "SCHEDULE: Tools for Developing and Analyzing Parallel Fortran Program," Tech. Memo 86, Argonne National Laboratory, Nov. 1986.
- [25] DONGARRA, J. J. and SORENSEN, D. C., "SCHEDULE Users Guide," Argonne National Laboratory, June 1987.
- [26] YANG, T. and GERASOULIS, A., "PYRROS: Static scheduling and code generation for message passing multiprocessors," Proc. of 6th ACM Inter. Conf. on Supercomputing, Washington D. C., pp. 428-437, July 1992.

- [27] CASAVANT, T. L. and KOHL, J. A., "The IMPROV Meta-Tool Design Methodology for Visualization of Parallel Programs," Invited Paper, International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), January 1993.
- [28] KOHL, J. A. and CASAVANT, T. L., "Methodologies for Rapid Prototyping of Tools for Visualizing the Performance of Parallel Systems," Presentation at Workshop on Parallel Computer Systems: Software Tools, Santa Fe, New Mexico, October 1991.
- [29] High Performance Fortran Forum, High Performance Fortran Language Specification, Version 1.0 Draft, Jan. 25, 1993.
- [30] SATO, L. M., "Um sistema de programação e processamento para sistemas multiprocessadores," Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho, 4, São Paulo, SP. Anais. p.95-107. 1992.
- [31] BEGUELIN, A.; DONGARRA, J.; GEIST, A.; MANCHEK, R. and SUNDERAM V., "A user's guide to PVM: Parallel virtual machine," Technical Report ORNL/TM-11826, Mathematical Sciences Section, Oak Ridge National Laboratory, September 1991.
- [32] SUNDERAM, V., "Concurrent computing with PVM," In Proceedings of the Workshop on Cluster Computing, Tallahassee, FL, December 1992.
- [33] ANSI Technical Committee X3H5, Parallel Processing Model for High Level Programming Languages, June 1992.
- [34] ANSI Technical Committee X3H5, Parallel Fortran Standard, 1992.
- [35] ANSI Technical Committee X3H5, Fortran Binding - Data Model Section, 1992.
- [36] SCOTT, L. R., "Pfortran: a parallel dialect of Fortran," Fortran Forum 11, vol. No. 3, pp. 20-31, Sept. 1992.
- [37] REESE, D. S. and LUKE, E., "Object oriented Fortran for development of portable parallel programs," In Proceedings of the 3rd IEEE Symposium on Parallel Distributed Processing, pages 608--615, Dallas, TX, December 1991.
- [38] SMITH, G., "Object-oriented Fortran tutorial. Engineering Research Center," Mississippi State University, 1992.
- [39] FOSTER, I. T. and CHANDY, K. M., "Fortran M: A language for modular parallel programming," Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, June 1992.
- [40] NEWTON, P. and BROWNE, J. C., "The CODE 2.0 Parallel Programming Language," Proc. ACM International Conf. On Supercomputing, July 1992.

- [41] REUTER, A.; GEUDER, U.; HAERDTNER, M.; WOERNER, B. and ZINK, R., "The GRIDS Project," Technical Report, University of Stuttgart, 1992.
- [42] BROOKS, E.; GORDA, B. and WARREN, K., "The Parallel C Preprocessor," in Scientific Programming, vol. 1, Number 1, John Wiley & Sons, Inc., New York.
- [43] GORDA, B.; WARREN, K. and BROOKS, E. D., "Programming in PCP," Technical Report, Lawrence Livermore National Laboratory, UCRL-MA-107029, April 1991.
- [44] SMITH, N. P., "Network Linda achieves near-linear speed-ups on multisystem networks," Supercomputing Review, pages 53--54, May 1991.
- [45] SCEARCE, S., "Network Linda builds a virtual supercomputer," on LANs. RISC World, 1(3), August 1991.
- [46] ANDREWS, G. R. and OLSSON, R. A., "The SR programming language," The Benjamin/Cummings Publishing Company, 1993.
- [47] OLSSON, R. A.; ANDREWS, G. R.; COFFIN, M. H. and TOWNSEND, G. M., "SR: A language for parallel and distributed programming," Technical Report TR 92-09, Department of Computer Science, The University of Arizona, Tucson, Arizona, March 1992.
- [48] LUSK, R. and BUTLER, R., "Portable parallel programming with p4," In Proceedings of the Workshop on Cluster Computing, Tallahassee, FL, December 1992.
- [49] BUTLER, R. and LUSK, E., "User's guide to the p4 programming system," Technical Report ANL-92/17, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, October 1992.
- [50] KOLAWA, A., "Parasoft: a comprehensive approach to parallel and distributed computing," In Proceedings of the Workshop on Cluster Computing, Tallahassee, FL, December 1992.
- [51] FLOWER, J.; KOLAWA, A. and BHARADWAJ, S., "The Express way to distributed processing," Supercomputing Review, pages 54-55, May 1991.
- [52] BIRMAN, K. and COOPER, R., "The ISIS project: real experience with a fault tolerant programming system," Operating System Review, pages 103-107, April 1991.
- [53] BIRMAN, K.; COOPER, R.; JOSPEH, T. A.; KANE, K. P.; SCHMUCK, F. and WOOD, M., "ISIS-a distributed programming environment," Cornell University, Ithaca, NY, June 1990.
- [54] FINCH, R. and KAO, Shao-Kong, "Coarse-grain parallel computing using ISIS tool kit," Journal of Computing in Civil Engineering, 6(2):233-244, April 1992.

- [55] GROPP, B. and LUSK, Ewing, "A test implementation of the MPI draft message-passing standard," Mathematics and Computer Science Division, Argonne National Laboratory, 1992.
- [56] GORDA, B., "Data Parallel Programming," Spring Proceedings, 1992 Cray User Group.
- [57] BROOKS, E. D., "The 1992 MPCl Yearly Report: Harnessing the Killer Micros," Lawrence Livermore National Laboratory, UCRL-ID-107022-92, March 1991.
- [58] RINARD, M. C.; SCALES, D. J. and LAM, M. S., "Heterogeneous parallel programming in Jade," In Proceedings of the Workshop on Cluster Computing, Tallahassee, FL, December 1992.
- [59] LAM, M. S., "Jade: a coarse-grain parallel programming language," In Proceedings of the Heterogeneous Network-Based Concurrent Computing Workshop, Tallahassee, FL, October 1991.
- [60] KALE, L.V., "The Chare Kernel Parallel Programming Language and System," in: Proc. of the International Conference on Parallel Processing, Aug. 1990.
- [61] KALE, L.V., "A Tutorial Introduction to Charm," Parallel Programming Laboratory Internal report, 1992.
- [62] WUAND, Min-You, "Hypertool: A Programming Aid for Message-Passing Systems," IEEE Trans. on Parallel and Distributed Systems, vol. 1 No. 3, pp. 330-343, July 1990.
- [63] BODIN, F.; BECKMAN, P.; GANNON, D.; NARAYANA, S. and YANG, S., "Distributed pC++: Basic Ideas for an Object Parallel Language," Proceedings of Supercomputing 91 (Albuquerque, Nov. 1991), IEEE Computer Society and ACM SIGARCH, pp. 273-282.
- [64] GRIMSHAW, A. S., "The Mentat run-time system: support for medium grain parallel computation," In Proceedings of the Fifth Distributed Memory Computing Conference, pages 1064-1073, Charleston, SC, April 9-12, 1990.
- [65] GRIMSHAW, A. S., "An introduction to parallel object-oriented programming with Mentat," Technical Report TR-91-07, Department of Computer Science, University of Virginia, April 1991.
- [66] CHANDRA, R.; GUPTA, A. and HENNESY, J. L., "Integrating Concurrency and Data Abstraction in the COOL Parallel Programming Language," Technical Report CSL-TR-92-511, Computer Systems Lab, Stanford University, February 1992.

- [67] CHANDRA, R.; GUPTA, A. and HENNESY, J. L., "Data Locality and Load Balancing in COOL," To Appear in the Symposium on Principles and Practices of Parallel Programming (PPoPP), May 1993.
- [68] HSEUSH, W.; LEE, J. C. and KAISER, G. E., "MeldC Threads: Supporting Large-Scale Dynamic Parallelism," Technical Report CUCS-010-92, Columbia University, March, 1992.
- [69] CHAPMAN, B.; MEHROTRA, P. and ZIMA H. P., "Vienna FORTRAN - A Fortran Language Extension for Distributed Memory Multiprocessors," in Compilers and Runtime Software for Scalable Multiprocessors, ed. J. Saltz and P.Mehrotra, Elsevier, Amsterdam, 1991.
- [70] CHAPMAN, B.; MEHROTRA, P. and ZIMA H. P., "Programming in Vienna Fortran," Scientific Programming, vol. Vol.1, No.1, 1992.
- [71] BRAUNL, Thomas, "Structured SIMD Programming in Parallaxis," Structured Programming Journal, vol. 10/3, 1993.
- [72] BRAUNL, Thomas, "Parallel Programming," in An Introduction Textbook, Prentice-Hall, summer 1993.
- [73] TICHYAND, W.; CHRISTIAN, F. and HERTER, G., "Modula-2*: An Extension of Modula-2 for Highly Parallel, Portable Programs," Technical Report KA-INF, 1990.
- [74] PHILIPPSEN, M. and TICHY, W. F., "Modula-2* and its Compilation," First International Conference of the Austrian Center for Parallel Computation, pp. 169-183, Springer Verlag, September 30 - October 2, 1991.
- [75] FOSTER, I. and TAYLOR, S., Strand New Concepts in Parallel Programming, Prentice Hall, 1989.
- [76] FEO, J. T.; CANN, D.C. and OLDEHOEFT, R. R., "A Report on the SISAL Language Project," Journal of Parallel and Distributed Computing, Vol. 12 No. 10, pp. 349-366, December 1990.
- [77] FOSTER, I.; OLSON, R. and TUECKE, S., "Productive parallel programming: The PCN approach," Scientific Programming, vol. 1(1), pp. 51-66, 1992.
- [78] BEMMERL, T. and BODE A., "An Integrated Tool Environment for programming distributed memory multiprocessors," in Distributed memory computing, Lecture Notes in Computer Science, ed. A. Bode, vol. Vol. 487, pp. 130 - 142, Springer-Verlag, 1991.
- [79] DEVANEY, J. E.; LIPMEN R.; LO, M. and MITCHELL W.F., "The Parallel Applications Development Environment (PADE)," PADE - Major Release 1.4, November 21, 1995.

- [80] SUGLA, B.; EDMARK, J. and ROBINSON, B., "Introduction to the CAPER Concurrent Application Programming Environment," IEEE Conference on Parallel Processing, 1989.
- [81] KIMELEMAN, D. and ZERNIK, D., "On-the-Fly Topological Sorting for Interactive Debugging and Live Visualization of Parallel Programs," To appear in the Third ACMONR Workshop on Parallel and Distributed Debugging, may, 1993.
- [82] EL-REWINI, H. and LEWIS, T., "Scheduling and Performance Evaluation Tool for Parallel Computing," Proceedings of 4th Annual Symposium on Parallel Processing, Fullerton, CA., p. 60, April 1990.
- [83] EL-REWINI, H. and LEWIS, T., "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," Journal of Parallel and Distributed Computing, Vol. 9, pp. 138-153, June 1990.
- [84] MATTSON, T. G.; DOUGLAS, C. C. and SCHULTZ, M. H., "A Comparison of CPS, LINDA, P4, POSYBL, PVM, and TCGMSG: Two Node Communication Times," Yale University Technical Report YALEU/DCS/TR-975, 1993.
- [85] FISCHLER, M.; HOCKNEY, G. and MACKENZIE, P., "Canopy 5.0 Manual," Technical Report, Fermilab.
- [86] FISCHLER, M., "The ACPMAPS System - A Detailed Overview," Fermilab publication FERMILAB-TM-1780.
- [87] BOMANS, L.; HEMPEL, R. and ROOSE, D., "The Argonne/GMD macros in Fortran for portable parallel programming and their implementation on the Intel iPSC/2," Parallel Computing, North-Holland, Vol. 15, pp. 119-132, 1990.
- [88] HEMPEL, R., "The ANL/GMD Macros (PARMACS) in Fortran for Portable Parallel Programming using the Message Passing Programming Model," User's Guide and Reference Manual, Version 5.1, Nov. 1991.
- [89] GEIST, G. A.; HEATH, M. T.; PEYTON, B. W. and WORLEY, P. H., "PICL: A portable instrumented communications library," Technical Report TM-11130, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1990.
- [90] GEIST, G. A.; HEATH, M. T.; PEYTON, B. W. and WORLEY, P. H., "A user's guide to PICL: a portable instrumented communications library," Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, Tennessee, January 1992.
- [91] CHASE, C.; CROWLEY, K.; SALTZ, J. and REEVES, A. Reeves, "Parallelization of Irregularly Coupled Regular Meshes," Sixth International Conference on Supercomputing, Washington DC, June 1992.
- [92] DAS, R.; MAVRIPLIS, D. J.; SALTZ, J.; GUPTA, S. and PONNUSAMY, R., "The Design and Implementation of a Parallel Unstructured Euler Solver Using

- Software Primitives (AIAA-92-0562)," Proceedings of the 30th Aerospace Sciences Meeting, Reno NV, 1992.
- [93] KANNAN, R., "Software Environment for Network Computing," Workshop on "Heterogeneous Network-Based Concurrent Computing," SCRI/Florida State University, October 16-18, 1991.
 - [94] KANNAN, R.; CHEN, C. L.; PACKER, M. and SINGH, H., "Directory Service for Group Work," CSCW 92 Tools & Technologies Workshop, Toronto Canada.
 - [95] ZINK, R., "The Stuttgart Parallel Processing Library SPPL and the X Windows Parallel Debugger XPDB," To be presented at the Seventh International Parallel Processing Symposium Parallel Systems Fair, Newport Beach CA, April 1993.
 - [96] BADEN, S. B. and KOHN, S., "The Reference Guide to GenMP -- The Generic Multiprocessor," Technical Report, University of California, San Diego, Dept. of Computer Science and Engineering, CS92-243, June, 1992.
 - [97] Thinking Machine Co., Prism User's Guide version 1.1, Dec. 1991.
 - [98] MasPar Computer, MPPE User Guide.
 - [99] Intel Supercomputer Systems Division, iPSC/2 and iPSC/860 Interactive Parallel Debugger Manual, April 1991.
 - [100] FLYNN, M., "Very high speed computing systems," Proc. IEEE, vol. 54, pp. 1901-1909, 1966.
 - [101] ZINK, R., "The Stuttgart Parallel Processing Library SPPL and the X Windows Parallel Debugger XPDB," To be presented at the Seventh International Parallel Processing Symposium Parallel Systems Fair, Newport Beach CA, April 1993.
 - [102] Cray Research Inc., UNICOS Performance Utilities Reference Manual SR-2040 6.0, pp. 199-234, 1991.
 - [103] Intel Supercomputer Systems Division, Performance Analysis Tools Manual.
 - [104] HEATH, M. T. and ETHERIDGE, J. A., "Visualizing the Performance of Parallel Programs," IEEE Software, Vol. 8, No. 5, pp. 29-39, September 1991.
 - [105] FINEMAM, C.; HONTALAS, P.; LISTGARTEN, S. and YEN, Jerry, "A User's Guide to AIMS," Version 1.1, May, 1992.
 - [106] Kendall Square Research, KSR Manual.
 - [107] BBN Systems and Technologies, Inc., Using the Xtra(TM) Programming Environment.
 - [108] FINEMAN, C.; HONTALAS, P.; LISTGARTEN, S. and YEN, J., "A User's Guide to AIMS," Version 1.1, May, 1992.

- [109] REED, D. A.; AYDT, R. A.; MADHYASTHA, T. M.; NOE, R. J.; SHIELD, K. A. and SCHWARTZ, B.W., "The Pablo Performance Analysis Environment," Technical Report, University of Illinois.
- [110] MILLER, B. P.; CLARK, M., HOLLINGSWORTH, J.; KIERSTEAD, S.; LIM, Sek-See and TORZEWSKI, T., "IPS-2: The Second Generation of a Parallel Program Measurement System," IEEE Trans. on Parallel and Distributed Systems, vol. 1 No. 2, April 1990.
- [111] HOLLINGSWORTH, J. K., "Dynamic Control of Performance Monitoring on Large Scale Parallel Systems," Technical Report, Univ. of Wisconsin-Madison Comp. Sci. Dept.
- [112] IRVIN, R. B. and MILLER, B. P., "Multi-Application Support in a Parallel Program Performance Tool," Univ. Of Wisconsin-Madison Comp. Sci. Dept. TechRep., #1135.
- [113] GU, W. and SCHWAN, K., "Falcon: A Monitoring and Visualization System for Parallel and Distributed Systems," Technical Report GIT-CC-93/11, Georgia Institute of Technology, January 1993. Draft.
- [114] HERRARTC, V. and LUSK, E., "Studying Parallel Programming Behavior with Upshot," Argonne National Laboratory Technical Report ANL-91/15.
- [115] ZABALA, E. and TAYLOR, R., "Maritxu: Generic visualization of highly parallel processing," Programming Environments for Parallel Computing, pp. 171-180, North-Holland, 1992.
- [116] ZABALA, E. and TAYLOR, R., "Process and processor interaction: architecture independent visualisation schema," Environments and Tools for Parallel Scientific Computing, 7-8 September, Saint Hilairdu Touvet, France, 1992.
- [117] Cray Research, Inc., CF77 Compiling System Volume 4: Parallel Processing Guide.
- [118] PASE, D. M. and FLETCHER, K. E., "Automatic Parallelization: A Comparison of CRAYfpp and KAI KAP/CRAY," NASA Ames NAS Technical Report, RND-90-010, Nov. 1990.
- [119] PASE, D. M. and FLETCHER, K. E., "Automatic Parallelization: A Comparison of CRAYfpp and KAI KAP/CRAY," NASA Ames NAS Technical Report, RND-90-010, Nov. 1990.
- [120] GANNIN, D.; LEE, J. K.; SHEL, B. S.; SARUKKAI, S.; NARAYANA, S.; SUNDARESAN, N.; ATTAPATU, D. and BODIN, F., "Sigma II: A Tool Kit for Building Parallelizing Compilers and performance Analysis Systems," Proceedings 1992.

- [121] KOHL, J. A. and CASAVANT, T. L., "Use of PARADISE: A Meta-Tool for Visualizing Parallel Systems," Proceedings of the Fifth International Parallel Processing Symposium (IPPS), Anaheim, California, pp. 561-567, May 1991.
- [122] KOHL, J. A. and CASAVANT, T. L., "A Software Engineering, Visualization Methodology for Parallel Processing Systems," Proceedings of the Sixteenth Annual International Computer Software&Applications Conference (COMPSAC), Chicago, Illinois, pp. 51-56, September 1992.
- [123] CASAVANT, T. L.; KOHL, J. A. and PAPELIS, Y. E., "Practical Use of Visualization for Parallel Systems," Invited Keynote Address Text for 1992 European Workshop on Parallel Computers(EWPC), Barcelona, Spain, March 23-24, 1992.
- [124] STASKO, J. T. and KRAEMER, E., "A Methodology for Building Application Specific Visualizations of Parallel Programs," Technical Report Graphics, May 1993.
- [125] GREEN, T. P. and SNYDER, J. S., "DQS, A Distributed Queuing System," DQS Documents, March 1992.
- [126] REVOR, L. S., "DQS Users Guide," DQS Documents, Sept. 1992.
- [127] CHENG, D. Y., "A Survey of Parallel Programming Languages and Tools," NASA Ames Research Center, Report RND-93-005, March 1993.
- [128] KLIETZ, A., "The Distributed Job Manager," User's Guide of DJM, 1992.
- [129] BRICKER, A.; LITZKOW, M. and LIVNY, M., "Condor Technical Summary," Condor Documents, Sept. 1991.
- [130] LITZKOW, M. and LIVNY, M., "Supporting Check pointing and Process Migration Outside the UNIX Kernel," Usenix Winter Conference, San Francisco, CA, 1992.
- [131] TURCOTTE, L. H., "A Survey of Software Environments for Exploiting Networked Computing Resources," Engineering Research Center for Computational Field Simulation, Mississippi State, June 1993.
- [132] PEREZ, P. A. S., "P2D - Um Ambiente de Auxílio à Paralelização de Aplicações Fortran," Universidade de São Paulo, Instituto de Ciências Matemáticas de São Carlos, Setembro 1996.
- [133] DUNCAN, R., "A Survey of parallel computer architectures," IEEE Comput. Mag., vol. 23, pp5-16, February 1990.
- [134] FARHAT, C. and LESOINNE, M., "Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics," International Journal for Numerical Methods in Engineering, Vol. 36, No. 5, pp. 745-764, 1993.

- [135] KOHN, S. R. and BADEN, S. B., "Blobs: Visualization of Particle Methods on Multiprocessors," CSE Technical Report Number CS92-241, May, 1992.
- [136] HARRISON, W. L. and AMMARGUELLAT, Z., "PARCEL and MIPRAC: Parallelizers for Symbolic and Numeric Programs," National Science Foundation, May 7, 1992.
- [137] BARRETT, G., "The development of OCCAM: typec, classes and sharing," In OCCAM Users group: Real-Time Systems with Transputers, 13, pp. 119-147, IOS Press, 1990.
- [138] MACCLAUGHRY, P. E., "PTOPP – An Practical Tool set for the Optimization of Parallel Programs," University of Illinois at Urbana-Champaign, 1992.
- [139] EIGENMANN, R. and MACCLAUGHRY, P., "Practical Tools for Optimizing Parallel Programs," Center for Supercomputing research and Development, University of Illinois at Urbana-Champaign, 1993.
- [140] PADUA, D. A.; EIGENMANN, R.; HOEFINGER, J.; PETERSEN, P.; TU, P.; WEARTERFORD, S. and FAIGIN, K., "Polaris: A New-Generation Parallelizing Compiler for MPPs," University of Illinois, R&D Status Report, June 15, 1993.
- [141] RÖTTGER, P.; SCHROEDER, U. and SIMON, J., "Virtual Topology Library for PARIX," Department of Mathematical and Computer Science, University of Paderborn, Alemanha, 1993.
- [142] MEHROTRA, P. and HAINES, M., "Overview of the Opus Language and Runtime System," NASA Langley Research Center, 1994.
- [143] LIANG, Wen-Yen; KING, Chun-Ta and LAI, F., "ADSMITH: Efficient Object-Based Distributed Shared Memory System on PVM," National Taiwan University, Taipei, Taiwan, 1996.
- [144] BRUCE, R. A. A.; CHAPPLE S.; MACDONALD, N. B. and TREW, A. S., "CHIMP and PUL: Support for Portable Parallel Computing," Edinburgh Parallel Computing Center, University of Edinburgh, March 1993.
- [145] BONELLO, C.; HURTEAU, G.; DONGEN, V. V. and GAO, G., "EPPP – an Integrated Environment for portable Parallel Programming," Rapport Technique CRIM-EPPP-94/07-20, 1995.
- [146] KALÉ, L. V.; BHANDARKAR, M.; JAGATHESAN, N.; KRISHNAN, S. and YELON, J., "Converse: An Interoperable Framework for Parallel Programming," Department of Computer Science, University of Illinois, Urbana, 1996.

- [147] BECKMAN, P.; GANNON, D.; and JOHNSON, E., "Portable Parallel Programming in HPC++," Department of Computer Science, Indiana University, Bloomington, 1996.
- [148] SQUYRES, J. M.; McCANDLESS, B. C. and LUMSDAINE, A., "Object Oriented MPI (OOMPI): A C++ Class Library for MPI Version 1.0.2," Laboratory for Scientific Computing, Department of Computer Science and Engineering, University of Notre Dame, 1996.
- [149] COULAUD, O. and DILLON, E., "PARA++: C++ Bindings for Message Passing Libraries – User Guide Version 2.1," Institute National de Recherche en Informatique et en Automatique – INRIA, France, March 25, 1997.
- [150] DEBBAGE, M. and HILL, M. B., "PARAPET: A Parallel Programming Environment Toolkit," University of Southampton, Parallel Application Center, November 3, 1992.
- [151] DANTART, R. C.; DEMEURE I.; MEUNIER, P. e BARTRO, V., "Phosphorus: a Tool for Shared Memory Management in a Distributed Environment," Ecole Nationale Supérieure des Télécommunications, Département Informatique, France, December 4, 1995.
- [152] LARUS, J. R.; RICHARDS, B. e VISWANATHAN, G., "C**": A Large-Grain, Object-Oriented, Data-Parallel Programming Language," Computer Science Department, University of Wisconsin-Madison, Madison, USA, November 24, 1992.
- [153] KALE, L. V. e KRISHNAN, S., "CHARM++: A Portable Concurrent Object Oriented system Based On C++," Department of Computer Science, University of Illinois, Urbana-Champaign, 1994.
- [154] "The Charm++ (4.5) Programming Language Manual," Department of Computer Science, University of Illinois, Urbana-Champaign, August 18, 1997.
- [155] NIKHIL, R. S., "CID: A Parallel, 'Shared-Memory' C for Distributed-Memory Machine," Cambridge Research Laboratory, Digital Equipment Corporation, September 29, 1994.
- [156] PHILIP, C., "An Automatic Program Translator For Distributed Memory MIMD Machines," Faculty of Science, Queen's University of Belfast, 1994.
- [157] FRANG, N, and BURKHART, H., "PEMPI: An Integrated Environment for Structured MPI Programming," University Basel, Department of Informatics.
- [158] DAVOLI, R.; GIACHINI, L. A.; AMOROSO, A. and ALVISI, L., "Parallel Computing in Networks of Workstations with Paralex," Technical Report UBLCS-92-4, October 1992.

- [159] BARATLOO, A.; DASGUPTA, P. and KEDEM, Z. M., "Calypso: An Environment for Reliable Distributed Parallel Processing," New York University, September 1995.
- [160] NEWTON, P. and DONGARRA, J., "Overview of VPE: A Visual Environment for Message-Passing Parallel Programming," 1995.
- [161] WIRTZ, G., "A Visual Toolset for Message-Passing Parallel Programming," Westfälische Wilhelms Universität, Dep. Of Computer Science.
- [162] AHMED, S.; CARRIERO, N. and GALERNTER, D., "A Program Building Tool for Parallel Applications," Yale University, Department of Computer Science, December 1993.
- [163] HILLIS, W., "The connection machine," Scientific American, vol. 256, pp. 108-115, June 1987.
- [164] KUNG, H., "Why Systolic architectures?," IEEE Computing Mag., vol. 15, pp. 37-46, January 1982.
- [165] HILLIS, W., "Massively Parallel Supercomputing: the Connection Machine," University Communications : Stanford, 1991.
- [166] HOSHINO, T., "An Invitation to the World of Pax," IEEE Comput. Mag., vol. 19, pp. 68-79, May 1986.
- [167] DENNIS, J., "Data Flow Supercomputer," IEEE Comput. Mag., vol. 13, pp. 48-56, November 1980.
- [168] ALMASI, G. S., "Highly Parallel Computing," 2 Ed., Red Wood City : California, 1994.
- [169] TANENBAUM, A. S. "Modern Operating Systems," Prentice-Hall International, Inc., 1992.
- [170] TRINDADE, O.; MARQUES, E.; JEUKENS, I. "A parallel architecture based on personal computers - requirements and definitions," In: Simpósio Nipo-Brasileiro de Ciência e Tecnologia, p. 203-212, Agosto 1995.
- [171] CALLAHAN, D.; COOPER, K.; TOREZON, L. "Parascope, a parallel programming enviroment," J. Supercomput. Appl., vol 5, Winter 1988.
- [172] BARKAKATI, N. "X Window System Programing," SAMS, 1991.
- [173] HELLER, D. "Xview Programing Manual – for Version 11 of X Window System, vol. 7. O' Reilly & Associates Inc., 1990.
- [174] FOWLER, M.; SCOTT, K. "UML Distilled: Applying the Standard Object Modeling Language," Addison-Wesley Pub Co., June 1997.

- [175] PRESSMAN, R. S.; "Engenharia de Software," Makron books do Brasil, 3^a ed.1995.
- [176] CARVALHO, M. A. A.; OLIVETE, A. L. and TRINDADE, O.; "Uma Classificação das Ferramentas para Programação Paralela Disponíveis – Revisão e Disponibilidade," Relatório Técnico em formulação, Universidade de São Paulo, São Carlos, 1998.
- [177] MAY, J. and BERMAN F., "Panorama: a portable, extensible parallel debugger", SIGPLAN Notices, pp. 96-106, December 1993.
- [178] SISTARE, S.; ALLEN, D.; BOWKER, R.; JOUDENAI, K.; SIMONS, J. and TITLE, R., "A scalable debugger for massively parallel Message-Passing programs", IEEE Parallel & Distributed Technology: Systems & Applications, Vol. 2, No. 2, pp. 50-6, Summer 1994.