**"XGRAPH - UMA FERRAMENTA DE
MONITORAÇÃO PARA SISTEMAS
PARALELOS"**

Oswaldo Lázaro Mendes

Dissertação apresentada ao Instituto de Física de São Carlos, da Universidade de São Paulo, para obtenção do Título de Mestre em Ciências: Física Aplicada opção Física Computacional.

Orientador: Prof. Dr. Álvaro Garcia Neto 28

São Carlos
1996

Mendes, Oswaldo Lázaro

XGRAPH - Uma Ferramenta de Monitoração para
Sistemas Paralelos / Oswaldo Lázaro Mendes
- São Carlos, 1996.
137 p.

Dissertação de (Mestrado) - Instituto de
Física de São Carlos, 1996

Orientador: Prof. Dr. Álvaro Garcia Neto.

1.Arquitetura de Computadores. 2. Computação
Gráfica. I. Título

IFSC UNIVERSIDADE
DE SÃO PAULO
Instituto de Física de São Carlos

Av. Dr. Carlos Botelho, 1465
CEP 13560-250 · São Carlos · SP
Brasil

Fone (016) 272-6222
Fax (016) 272-2218

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE OSWALDO
LAZARO MENDES APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, DA
UNIVERSIDADE DE SÃO PAULO, EM 26 DE MARÇO DE 1996.

COMISSÃO JULGADORA:

Prof. Dr. Alvaro Garcia Neto

Prof. Dr. Eduardo Whitaker Bergamini

Profa. Dra. Agma Juci Machado Traina

Dedico este trabalho, com muito amor, à minha esposa **Osmari** que com enorme paciência, me ajudou, dando-me compreensão e carinho; meus pais **Otagino** e **Shirley**, que criaram condições para o meu progresso e acreditaram em mim e, acima de todos, meu **DEUS** que sempre guiou meus caminhos e, através destas pessoas, está concretamente presente em minha vida.

AGRADECIMENTOS

Ao Professor Dr. Álvaro Garcia Neto, pela orientação e paciência dispensadas na realização deste trabalho, bem como os ensinamentos transmitidos que muito contribuíram para a minha formação profissional.

Ao Professor Dr. Odelar Leite Linhares pela grande amizade, paciência, dedicação, competência e principalmente lição de vida nos ensinamentos durante a minha graduação.

Aos Professor Dr. Jan Frans Willem Slaets pelos ensinamentos que ajudaram no meu aprendizado na pós-graduação.

Ao CNPQ pelo apoio financeiro.

À Bene e ao Godoy pela dedicação e competência na confecção dos desenhos contidos nesta dissertação.

Aos velhos amigos da graduação, Marcus Vinícius, Pinda, Hasegawa, Luis Paulo, Marcelo, que muito me apoiaram nos estudos.

Aos novos amigos da sala 8, Regina, Márcio, Patrícia, João, Marcos, João Ângelo e tantos outros, pelas conversas nas noites de estudo.

Aos meus amigos Mauro, João de Lucca e Nelson Roberto pela ajuda, tanto de caráter profissional quanto pessoal.

A todos os meus amigos professores da UNAERP em especial (Toninho), funcionários do CDCC - USP e os seguintes funcionários da Física-USP Lírrio, Cláudio, Cláudia, Otávio, Wladerez pelo apoio e estímulo durante este trabalho.

Aos meus avós, tios e irmãos Carla e Paina, que me apoiaram com carinho e amor em todas as fases da minha vida.

A todos os meus amigos e colegas, que direta ou indiretamente contribuíram com amizade, apoio e auxílio.

SUMÁRIO

LISTA DE FIGURAS	i
LISTA DE TABELAS	iii
RESUMO	iv
ABSTRACT	v
INTRODUÇÃO	1
 1 - MONITORAÇÃO DE SISTEMAS	 4
1.1 DEFINIÇÃO DE DADOS	4
1.2 ÍNDICES DE PERFORMANCE	5
1.3 TIPOS DE ÍNDICE DE DESEMPENHO	7
1.4 TÉCNICAS PARA ANÁLISE DE DESEMPENHO	13
1.4.1 Técnicas de Medidas	14
1.4.1.1 Detecção de Eventos	15
1.4.1.2 Amostragem	15
1.4.2 Técnicas de Aferição	16
1.4.2.1 Benchmarking	16
1.4.2.2 Coleta de Dados	17
1.4.2.3 Construção de Protótipos	17

1.4.3	Técnicas de Modelagem	18
1.4.3.1	Modelagem Analítica	18
1.4.3.2	Modelagem Simulativa	19
1.5	FERRAMENTAS DE COLETA DE DADOS	20
1.5.1	Observação Humana	21
1.5.2	Sistema de Contagem	21
1.5.3	Monitores	22
1.5.3.1	Monitores de Hardware	23
1.5.3.2	Monitores de Software	24
1.5.3.3	Monitores Híbridos	26
1.5.4	Simuladores	27
1.6	REPRESENTAÇÃO E ANÁLISE DE DADOS	30
1.7	FAZENDO UMA ESCOLHA	34
1.8	SUMÁRIO	35
2	- ARQUITETURA A FLUXO DE DADOS	37
2.1	INTRODUÇÃO	37
2.2	MODELO DE COMPUTAÇÃO	37
2.3	PROGRAMANDO EM DATAFLOW	40
2.4	ARQUITETURA DA MÁQUINA DATAFLOW DE MANCHESTER	41
2.5	CONTROLE DO PARALELISMO	44
2.6	MR - O SIMULADOR MULTI-RING	46
2.6.1	A Chave	47
2.6.2	O Processador de Elementos	48
2.6.3	Estrutura de Armazenamento	48
2.6.4	Alocador Global	49

2.6.5	Monitoramento	50
2.7	OUTROS SIMULADORES	51
2.7.1	SIM	51
2.7.2	SAW	54
2.8	RESUMO DAS CARACTERÍSTICAS DOS SIMULADORES	56
3	- A INTERFACE GRÁFICA	58
3.1	HISTÓRICO DO SISTEMA XWINDOW	58
3.2	O SISTEMA XWINDOW	59
3.2.1	Modelo Cliente-Servidor	62
3.2.2	XPROTOCOL: A Linguagem de Máquina do X	63
3.2.3	XLIB: A Linguagem montadora do X	64
3.2.4	XTOOLKITS: A Linguagem de alto nível do X	65
3.2.5	Dependência do Sistema Operacional	66
3.2.6	Interface Gráfica do Usuário e o X	67
3.3	XVIEW	69
3.3.1	XView e o Sistema Xwindow	69
3.3.2	Programação Orientada a Objeto	70
3.3.2.1	Classe Hierárquica do Objeto	71
3.3.2.2	Chave de Objeto	72
3.3.3	Funções Baseadas em Atributos	73
3.3.3.1	Criando e Manipulando Objetos	74

3.3.3.2	Tipos de Atributos	74
3.3.4	Tipos de Objetos	75
3.3.4.1	Objetos Genéricos	76
3.3.4.2	Objetos Windows	77
3.3.4.3	Frames e Subframes	78
3.3.4.4	Subwindows	81
3.3.4.5	Objetos não Visuais	82
3.3.5	O Modelo Notifier	83
3.3.5.1	Estilo de Programação Callback	83
3.3.5.2	Porque um Sistema Baseado em Notificação?	86
3.3.5.3	Relacionamento entre o Notificador,Objetos e uma Aplicação	86
3.4	SUMÁRIO	87
4	- XGRAPH - A FERRAMENTA DE MONITORAÇÃO AO NÍVEL DE MÁQUINA	88
4.1	INTRODUÇÃO	88
4.2	MATRIZ DE MEDIDAS	89
4.3	DESCRIÇÃO DA ENTRADA DOS DADOS	92
4.4	VISÃO DO USUÁRIO COM RELAÇÃO A FERRAMENTA XGRAPH	95
4.4.1	Fase de Seleção	95
4.4.2	Fase de Processamento	99
4.5	UM EXEMPLO DE USO	103
4.6	MEDIDAS E GRÁFICO HISTORY	109
4.7	ESTRUTURA DE DADOS	112

4.8	FUNÇÕES DAS ESTRUTURAS DO PROGRAMA	113
4.9	FAZENDO UMA OBSERVAÇÃO	116
4.10	SUMÁRIO	117
5	- DIFERENÇAS ENTRE AS FERRAMENTAS GRAPH E XGRAPH E CONCLUSÕES	119
5.1	INTRODUÇÃO	119
5.2	ARQUITETURA DOS DOIS SISTEMAS	119
5.3	OBJETOS	120
5.3.1	Canvas	120
5.3.2	Panels e Menus	122
5.3.3	Text	122
5.3.4	Scrollbar	123
5.3.5	Imagens	123
5.4	MELHORAMENTOS DO XGRAPH EM RELAÇÃO AO GRAPH	124
5.5	CONVERSOR AUTOMÁTICO	126
5.6	VANTAGENS E DESVANTAGENS	127
6	- CONCLUSÕES E FUTUROS TRABALHOS	129
	BIBLIOGRAFIA	131

LISTA DE FIGURAS

Figura 1 - Classificação de monitoração	5
Figura 2 - Uma utilização gráfica	32
Figura 3 - Figura Kiviat de um sistema bem balanceado	33
Figura 4 - Gráfico do Dataflow	38
Figura 5 - Estrutura da Máquina Dataflow	44
Figura 6 - Estrutura do Simulador Multi-Ring	46
Figura 7 - Arquivo gerado pelo Simulador SIM	52
Figura 8 - Estrutura do Simulador SAW	56
Figura 9 - Uma estação com o Sistema Xwindows	61
Figura 10 - Modelo Cliente/Servidor	63
Figura 11 - Estrutura geral de uma Aplicação X	66
Figura 12 - Área de trabalho OPEN LOOK	68
Figura 13 - Classes hierárquicas do Xview	71
Figura 14 - Criação de objetos é top down; Estabelecer atributos é bottom up	76
Figura 15 - Frame Base	79
Figura 16 - Command Frame	80
Figura 17 - Help Frame	80
Figura 18 - Notice	81
Figura 19 - Fluxo de controle de um programa convencional	84

Figura 20 - Fluxo de controle de um programa baseado em Notificador	85
Figura 21 - Interpretação gráfica de uma máquina paralela	91
Figura 22 - Gráfico de saída da simulação do XGraph	91
Figura 23 - Especificação sintática da linguagem de entrada do Xgraph	92
Figura 24 - Painel para seleção das matrizes e colunas a serem exibidas	96
Figura 25 - Painel de seleção das medidas que estão para ser Dump	98
Figura 26 - Painel pronto para iniciar a fase de processamento	99
Figura 27 - Painel fazendo a simulação passo a passo	100
Figura 28 - Painel com a simulação congelada	101
Figura 29 - Painel fazendo a simulação com intervalo de tempo definido pelo usuário	101
Figura 30 - Painel mostrando a opção de fazer o retorno a uma parte já mostrada pela simulação	102
Figura 31 - Painel mostrando a simulação retornando ao início	102
Figura 32 - Painel mostrando o final da simulação	104
Figura 33 - Janela History Graph causada pela seleção do botão Show History	105
Figura 34 - Painel mostrando o History Graph	106
Figura 35 - Janela History transformada em ícone	107
Figura 36 - Janela File causada pela seleção do botão Dump History	107
Figura 37 - Saída do arquivo de formato raster gerado pela opção Dump History	108
Figura 38 - Janela de saída do programa	109
Figura 39 - Reescalonamento de uma medida	110
Figura 40 - Canvas	121

LISTA DE TABELAS

Tabela I	- Distribuição em porcentagem das sentenças	11
Tabela II	- Distribuição em porcentagem dos operadores	12
Tabela III	- Distribuição em porcentagem dos tipos de operandos	12
Tabela IV	- Distribuição em porcentagem da localidade dos operandos	13
Tabela V	- Principais características dos simuladores MR, SIM e SAW	57
Tabela VI	- Funções de manipulação dos objetos	74
Tabela VII	- Subjanelas e suas respectivas funções	82
Tabela VIII	- Objetos não visuais	83
Tabela IX	- Objetos, Procedimentos e Ações dos eventos na Fase de Seleção	114
Tabela X	- Objetos, Procedimentos e Ações dos eventos na Fase de Processamento	115

RESUMO

O presente trabalho estuda técnicas para monitoração de sistemas paralelos. Baseado nesses estudos, uma ferramenta para monitoração de sistemas paralelos (Xgraph) foi construída. Essa ferramenta, baseada na Graph desenvolvida em Manchester, permite uma compreensão mais profunda das características de arquiteturas a fluxo de dados (dataFlow) sob o ambiente XWindows.

ABSTRACT

This work comprises studies on techniques that can be applied on the monitoring of parallel systems. Based on these studies, the XGRAPH tool for use in the monitoring of parallel systems has been constructed and is also presented in this work. The XGRAPH tool is based on the GRAPH tool developed in Manchester, and runs on a Xwindows environment, permitting a better understanding of architectures dataflow.

INTRODUÇÃO

Esta dissertação estuda técnicas de monitoramento de sistemas paralelos e descreve uma ferramenta de monitoração desses sistemas.

Uma das características do processamento paralelo é a geração de uma grande quantidade de dados, e ferramentas para avaliar o desempenho são essenciais para identificar os dados relevantes e fazer uma avaliação de um sistema paralelo em aplicações realísticas.

Uma ferramenta, chamada **GRAPH** foi desenvolvida pelo Grupo de Pesquisa em Fluxo de Dados de Manchester, por Tang **Tang**¹, utilizando a ferramenta SunView, sob SunWindows, portanto só disponível para Hardware SUN. Neste mestrado a ferramenta **GRAPH** foi portada para a ferramenta Xview, dentro do gerenciador de janela Xwindows para poder trabalhar em um ambiente

mais moderno baseado no conceito Cliente/Servidor, usando protocolo independente do hardware e do sistema operacional e foi chamada de **XGRAPH**. Além disso, a ferramenta foi expandida, possibilitando maior interação com o usuário na fase de processamento.

Essa dissertação está dividida em seis capítulos:

O capítulo 1 descreve conceitos básicos de monitoração de sistemas, as diferentes técnicas e quais os tipos e as capacidades das ferramentas que podem ser utilizadas. Discute também os prós e contras das diversas características de uma e outra ferramentas.

O capítulo 2 descreve a arquitetura de Fluxo de Dados (Dataflow) que explora o paralelismo no nível de instrução, mostrando sua arquitetura, seu controle de paralelismo, e descrevendo o simulador, que gera dados de entrada que a ferramenta **XGRAPH** monitora e apresenta em forma de gráficos.

O capítulo 3 descreve o sistema Xwindow, o mecanismo Cliente/Servidor, o Xprotocolo, o Xlib e a ferramenta orientada a objetos Xview, explicando como são representados e manipulados cada um dos seus

objetos e como é a interação entre os objetos e o notificador que é responsável pela detecção e despachos dos eventos.

O capítulo 4 descreve a ferramenta de monitoração ao nível de máquina, **XGRAPH**, as matrizes, as estruturas de dados, os dados de entrada, passando por todas as fases do processo de execução da ferramenta em cima de um exemplo mostrando o funcionamento desta ferramenta. Este capítulo pode ser usado como um manual para utilização da ferramenta.

O capítulo 5 descreve as principais diferenças entre as ferramentas GRAPH e XGRAPH, para isso discute as diferenças entre SunView e Xview, em relação a arquitetura dos sistemas SunWindows e Xwindows, os objetos e as imagens e as novas opções que a ferramenta XGRAPH possibilita para o usuário.

O capítulo 6 sumariza os principais resultados obtidos e propõe trabalhos subseqüentes.

CAPÍTULO 1 - MONITORAÇÃO DE SISTEMAS

1.1 DEFINIÇÃO DE DADOS

Monitoração de desempenho de computador é o processo de coleta de dados referente a vários subsistemas de computador. Duas atividades estão envolvidas no processo de monitoração: A coleta efetiva de estatística durante a operação do sistema, e a interpretação e análise da coleta de dados.

Após a escolha de uma linha de avaliação, é necessário decidir qual característica avaliar, como avaliá-la, e quais ferramentas precisam ser usadas para isolar o dado desejado. O dado coletado e identificado pode, então, ser exibido para análise. As diversas etapas do processo de monitoração pode ser classificado na figura 1.

Figura 1 - Classificação de Monitoração

1.2 ÍNDICES DE PERFORMANCE

O sistema de software inclui o sistema operacional e programas de sistemas tais como assemblers, compiladores, bibliotecas de subrotinas e utilitários de gerenciamento de recursos. Programas de Aplicação também fazem parte do sistema de computador, e sua performance é uma componente do desempenho do sistema total. O desempenho de um sistema de software inclui características tais como níveis de multiprocessamento e multiprogramação, tempo de retorno, tempo de resposta e *throughput* (taxa de desempenho).

Não é fácil generalizar quais índices de performance são necessários. A escolha depende dos objetivos da avaliação, e de outras situações específicas. Por exemplo, um estudo que procure aumentar a capacidade do sistema pode ter os seguintes objetivos:

- Reduzir o *overhead* do sistema e de outras atividades não produtivas.
- Reduzir a existência da Carga de Trabalho (*WorkLoad*).

Embora ambos objetivos pareçam similares, requerem acessos completamente diferentes **Ferrari**². Redução no sistema de *overhead* pode envolver modificações operacionais no Sistema Operacional ou reconfiguração do hardware. Reduzir a existência da *WorkLoad*, envolve sintonia na existência de programas. Medidas tais como tempo de CPU, entrada/saída (E/S) e atividades de paginação são necessárias, e a sintonia do processo pode incluir modificação estrutural das funções do programa, otimização de código e redução das atividades de E/S.

1.3 TIPOS DE ÍNDICE DE DESEMPENHO

Para obter avaliação comparativa, é necessário que os experimentos sejam reprodutíveis, e que, para uma mesma carga de trabalho os mesmos resultados sejam alcançáveis. Por exemplo, comparando-se o desempenho de diferentes sistemas de configuração, a diferença na performance será causada pela mudança em ambos a carga de trabalho e a configuração. É então difícil separar a respectiva contribuição desses dois fatores.

Um *benchmark* é um conjunto de programas existentes que são codificados em uma linguagem específica e executado no sistema que está sendo avaliado, e um conjunto amplo de *benchmark* pode demonstrar vários aspectos do sistema.

Os principais *benchmarks* segundo Weiker **Weiker**³ são:

Whetstone

O Whetstone foi um dos primeiros programas na literatura projetado especificamente para ser um *benchmark*. Publicado em 1976, sendo Algol 60 a

linguagem original de publicação, atualmente é usado quase que exclusivamente nas versões Fortran e C.

O programa consiste em diversos módulos, cada um contendo comandos de algum tipo particular (aritmética inteira, aritmética em ponto flutuante, comandos condicionais, chamadas a procedimentos e outras), terminando com comandos para impressão dos resultados. Pesos são atribuídos aos diferentes módulos tal que a distribuição das "instruções Whetstone" para o programa *benchmark* corresponda à distribuição observada no programa amostral. Esses pesos são escolhidos de tal forma que o programa execute um múltiplo de um milhão dessas instruções Whetstone. Assim os resultados são fornecidos como KWIPS (*Kilo Whetstone Instructions per Second*). Dessa forma o termo instruções por segundo é mantido, mas impõe um sentido independente de máquina.

O programa possui alta porcentagem de dados e operações em ponto flutuante. Isso ocorre porque o objetivo do *benchmark* é justamente representar programas numéricos. Uma alta porcentagem do tempo de execução é gasto com funções de bibliotecas matemáticas. Essa propriedade é derivada dos dados estatísticos que constituem a base do Whetstone.

Como pode ser observado na tabela IV o Whetstone usa poucas variáveis locais e muitas variáveis globais. Um compilador no qual as variáveis globais mais usadas são alocadas em registradores, gerará código com melhores índices.

Linpack

O programa Linpack foi publicado em 1976 e correspondia inicialmente somente a uma coleção de subrotinas de álgebra linear freqüentemente usadas em programas Fortran. O autor considerou o que era parte da "vida real" de um programa e o transformou em *benchmark*.

O programa opera sobre uma grande matriz, onde as subrotinas internas manipulam-na como um vetor unidimensional (uma otimização comum na programação em Fortran). Os resultados são fornecidos em milhões de operações em ponto flutuante por segundo (MFLOPS); o número de operações em ponto flutuante que o programa executa pode ser derivado a partir do comprimento do vetor.

Essa terminologia implica que as operações inteiras são negligenciadas, ou ainda, que seu tempo

de execução está incluído nas operações em ponto flutuante.

O programa Linpack possui alta porcentagem de operações em ponto flutuante. Outra característica importante do Linpack é a utilização de variáveis locais opostamente ao Whetstone (tabela IV).

Drystone

O Drystone foi publicado em 1984 e codificado em ADA. Atualmente é usado principalmente na linguagem C.

A base para Drystone é a distribuição das características da linguagem fonte na programação não numérica (software básico) como por exemplo sistema operacional, compiladores, editores, etc. Além da diferença nos tipos de dados, os programas numéricos e os programas tipo-sistema tem duas diferenças: os programas tipo-sistemas contêm laços menores, sentenças simplificadas de cálculos e mais comandos condicionais e chamadas a procedimentos. O Drystone consiste em doze procedimentos incluídos em um laço de medidas com 94 comandos. Um laço corresponde a um Drystone, os resultados são usualmente dados em Drystone por segundo.

Entre suas principais características têm-se: o Drystone não contém operações em ponto flutuante em seu laço de medida; uma porcentagem considerável do tempo de execução é gasto em funções de cadeias de caracteres. Somente uma pequena quantidade de dados globais é manipulada.

Tabela I - Distribuição em porcentagem das sentenças

Comando	Dhrystone	Whetstone	Linpak
Comando For	6.8	17.3	49.3
Comando Goto	-	0.5	-
Comando While	4.9	-	-
Comando Switch	1.0	-	-
Comando Break	1.0	-	-
Comando Return	4.9	-	-
Comando Call (procedimento do usuário)	9.7	11.9	-
Comando Call (função do usuário)	4.9	-	-
Comando Call (procedimento do sistema)	1.0	-	-
Comando Call (função do sistema)	1.0	4.7	-
Comando IF	13.6	8.6	2.2
Associações	51.6	56.7	48.5
Total	100.0	100.0	100.0

Tabela II - Distribuição em porcentagem dos operadores

Operador	Dhrystone	Whetstone	Linpack
+ (int/char)	21.0	11.9	14.1
- (int)	5.0	6.0	-
* (int)	2.5	6.0	-
/ (int)	0.8	-	-
+(float/double)	-	14.9	14.1
-(float/double)	-	2.1	-
*(float/double)	-	9.3	14.1
/(float/double)	-	4.6	-
<, ?, <=, ??	10.1	10.7	14.5
>, >=	11.7	2.8	0.6
Operadores Lógicos	3.3	-	0.2
Indexação (unidimensional)	5.9	24.5	-
Indexação (bidimensional)	3.4	-	0.2
Seleção de registro	7.6	-	42.3
Operadores Específico da linguagem C	13.4	7.2	-
Total	100.0	100.0	100.0

Tabela III - Distribuição em porcentagem dos tipos de operandos

Tipo de Operando	Dhrystone	Whetstone	Linpack
Inteiro	57.0	55.7	67.2
Caracter	19.6	-	-
Float/Double	-	44.3	32.8
Enumerado	10.9	-	-
Lógico	4.2	-	-
Vetor	0.8	-	-
Cadeia de caracteres	2.3	-	-
Ponteiro	5.3	-	-
Total	100.0	100.0	100.0

**Tabela IV - Distribuição em porcentagem da
localidade dos operandos**

Localidade de Operando	Drystone	Whetstone	Linpack
Local	48.7	0.4	49.5
Global	8.3	56.3	-
Parâmetro (valor)	10.6	18.6	17.0
Parâmetro(referência)	6.8	1.9	24.6
Resultado da Função	2.3	1.3	-
Constante	23.4	21.6	8.8
Total	100.0	100.0	100.0

1.4 TÉCNICAS PARA ANÁLISE DE DESEMPENHO

Durante a avaliação de desempenho de um sistema o avaliador deve colher informações associadas aos parâmetros significativos à análise para uma dada carga de trabalho **Fernandes**⁴. Técnicas de avaliação são os métodos pelos quais tais informações podem ser obtidas a partir do próprio sistema (técnicas de aferição) ou então através de um modelo representativo do sistema (técnicas de modelagem).

As técnicas de modelagem são, em geral, adotadas uma vez que os sistemas computacionais modernos são complexos e as técnicas de aferição constituem tarefas que não são fáceis de serem conduzidas **Santana**⁵. Além

disso, essas técnicas devem ser empregadas quando se deseja analisar o desempenho de um sistema já implementado ou em fase final de desenvolvimento.

Técnicas de avaliação baseadas em modelos requerem que esses modelos sejam resolvidos. A solução pode ser obtido através de métodos analíticos ou de simulações **Orlandi**⁶. A confiabilidade de um modelo pode ser assegurada com o conhecimento prévio de que sua base conceitual esteja correta. Sem essa certeza pode-se apenas concluir que os resultados provenientes do modelo refletem as tendências do comportamento do sistema real.

1.4.1 TÉCNICAS DE MEDIDAS

Medidas diretas do sistema requerem a atual existência do sistema. Características são tratadas quantitativamente, permitindo significado cheio de comparações de dados. Detecção de Eventos e Amostragem são as mais comuns técnicas, e cada uma é descrita a seguir.

1.4.1.1 DETECÇÃO DE EVENTOS

Detecção de Eventos captura todos os eventos associados e grava eles em ordem nas quais eles ocorreram. Um evento de software é associado com uma função de programa, por exemplo quando um programa pesquisa um certo estágio de execução. Um evento de hardware consiste do aparecimento ou mudança de um ou mais sinais nos circuitos do sistema componente, e é independente do conteúdo lógico do programa sendo executado naquele momento. Essa técnica é baseada na interceptação e gravação de todos os eventos de um dado tipo e este é usado por esta razão quando é necessário conhecer a seqüência de eventos ou o exato número dessas ocorrências num dado intervalo de tempo. Isto pode resultar numa soma excessiva de dados, e o *overhead* é diretamente proporcional ao número de tipos de eventos para ser interceptado e para as suas freqüências.

1.4.1.2 AMOSTRAGEM

Amostragem pode ser usado para medir os valores dos diferentes sistemas componentes acima de um período de tempo. No intervalo de tempo específico o sistema é interrompido para detectar o estado de

alguns destes componentes. Os dados coletados podem determinar o que aconteceu durante o intervalo de tempo e, como diferentes atividades estão relacionadas umas com as outras.

1.4.2 TÉCNICAS DE AFERIÇÃO

Técnicas de aferição são ferramentas utilizadas para analisar o desempenho de um sistema através de sua experimentação. Essa experimentação pode ser efetuada utilizando software (como por exemplo os *benchmarks*) ou hardware (como por exemplo monitores de hardware). As técnicas de aferição podem ser divididas em Benchmarking, Coleta de Dados e Construção de Protótipos **Fernandes⁴**.

1.4.2.1 BENCHMARKING

Em algumas situações, como na aquisição de novos equipamentos, é interessante que o desempenho de diferentes sistemas, sejam comparados. Para isso, é necessário que o desempenho seja obtido com todas as máquinas operando sobre as mesmas condições, ou seja, a mesma carga de trabalho deve ser executada em cada uma delas. A carga de trabalho utilizada nesse

procedimento (que pode ser uma aplicação qualquer do usuário ou em um procedimento específico para este objetivo), é chamada de *benchmarking* **Fernandes**⁴.

1.4.2.2 COLETA DE DADOS

Essa técnica deve ser conduzida sobre sistemas reais, operando em condições normais, através da inserção de instruções de algum hardware ou ao acréscimo de algum software específico para este fim. Essa técnica é a que fornece resultados mais precisos, entretanto sua aplicação requer que o sistema em questão esteja fisicamente disponível **Santana, R**⁷.

1.4.2.3 CONSTRUÇÃO DE PROTÓTIPOS

É uma técnica empregada quando se deseja obter dados de um sistema ainda não disponível. O protótipo nada mais é que uma aplicação simplificada do sistema real **Orlandi**⁶. É um método que oferece boa precisão nos resultados, porém a construção do protótipo torna-se algumas vezes inviável devido aos custos envolvidos com a construção, testes e modificações do mesmo.

1.4.3 TÉCNICAS DE MODELAGEM

Técnicas de modelagem podem ser apenas usadas em protótipos, ou seja em sistemas que estão no estágio de construção. Em situações onde o sistema não existe (durante a fase de projeto, por exemplo), ou não está disponível (quando os efeitos das modificações do sistema tem de ser determinadas antes da mudança ser feita).

Técnicas de modelagem para análise de desempenho de sistemas computacionais consistem na construção e análise de modelos representativos do sistema em estudo. Análise visa a obtenção de informações que auxiliem na resolução de algumas questões sobre o sistema real. Existem dois tipos de técnicas de modelagem: Modelagem Analítica e Modelagem Simulativa.

1.4.3.1 MODELAGEM ANALÍTICA

A Modelagem Analítica é uma representação matemática do sistema de modelos analíticos que foram construídos, ou seja, permite escrever uma relação funcional entre os parâmetros do sistema e os critérios de desempenho escolhido, em termos de equações que podem ser resolvidas analiticamente

Soares⁸. Os sistemas que usam teoria de listas são os mais comuns.

A Modelagem Analítica oferece vantagens por causa de sua aplicabilidade geral, baixo custo relativo, facilidade e flexibilidade de uso. Contudo, a necessidade para o modelo ser matematicamente solúvel, freqüentemente obriga suposições que influenciam o grau de detalhe do modelo, limitando sua precisão. Entretanto muitos sistemas reais e muitas medidas da performance da máquina podem ser manipulados por modelos matemáticos.

Para arquiteturas paralelas, modelagem analítica não é sempre possível mesmo para um sistema com um nível de complexidade moderado, devido as características de complexidade do sistema e as interações entre elas.

1.4.3.2 MODELAGEM SIMULATIVA

A dificuldade de ocorrência descrevendo o sistema pela modelagem analítica tem favorecido técnicas de Modelagem Simulativa. Simulação coleta informações observando o modelo de um sistema que pode predizer qual o procedimento do sistema real será exibido.

Para a simulação é necessário ter um modelo. O projeto do modelo pode ser executado fora, sem considerar explicitamente alguma obrigação matemática. Modelos detalhados e sofisticados podem ser construídos da descrição cuidadosa das funções do sistema. O modelo deve reproduzir todos os aspectos do sistema para ser estudado. Este deveria não incluir detalhes do sistema que são irrelevantes para a performance. Uma vez que um modelo simulativo esteja construído, este pode ser tratado como um sistema real.

1.5 FERRAMENTAS DE COLETA DE DADOS

Ferramentas de coleta de dados são dispositivos que observam a operação de um sistema do computador em um período de tempo e registra, tanto quantitativamente quanto qualitativamente, valores de certas variáveis ou componentes do sistema. Idealmente, as ferramentas deveriam consumir tão poucos recursos do sistema avaliado quanto possível, de maneira que o *overhead* causado pela coleta de dados não altere a performance do sistema.

As duas técnicas de medição mencionadas na seção prévia não são reciprocamente exclusiva.

Freqüentemente, elas coexistem dentro de uma só ferramenta de coleta de dados.

1.5.1 OBSERVAÇÃO HUMANA

A mais simples forma de coleta de dados é a observação humana de uma atividade de máquina. Algumas indicações auditivas e audio-visuais podem ajudar a entender onde ineficiências possam existir. Por exemplo, um led de interrupção que fica permanentemente aceso pode indicar sobrecarga de interrupção no sistema. O operador pode identificar informações tais como, dispositivos periféricos que tem a mais alta média de faltas em relação aos outros, trabalhos que causam várias contenções de recursos e *bottlenecks*. Embora a observação humana seja um caminho grosseiro para coletar dados de performance, esta pode dar uma média estimada das atividades do sistema, e prover conveniente chave para as características operacionais do sistema.

1.5.2 SISTEMA DE CONTAGEM

Muitos sistemas operacionais disponibilizam rotinas de contagem que dão informações rotineiras das atividades do sistema. Eles gravam recursos requeridos

usados por cada trabalho, tais como, o tempo que um trabalho torna-se ativo ou inativo, a soma do tempo de processamento, tempo de I/O e o equipamento requerido pelo trabalho.

A maior vantagem dos sistemas de contagem é que eles causam muito pouca disjunção para sistemas de atividades normal, pois a manutenção da contagem diária é geralmente uma função padrão do sistema. Por exemplo, o computador IBM 370 tem o SMF (Sistema Gerenciamento de Facilidades) o qual é um pacote de contagem para o sistema operacional MVS **IBM**⁹. Ele registra dados relativos aos programas de aplicação, tais como tempo usado da CPU, número de falta de página e número de operações I/O. Para o sistema operacional UNIX temos o AUDIT **SUN**¹⁰.

1.5.3 MONITORES

Monitores são dispositivos que medem a performance do sistema. Existem basicamente dois tipos de monitores, diferindo das tecnologias para suas implementações.

1.5.3.1 MONITORES DE HARDWARE

Monitores de hardware correspondem ao emprego de um hardware específico para coletar e analisar alguns dados pertinentes ao objeto em estudo. Todas as informações sobre a execução do programa podem, a princípio, ser coletadas monitorando todos os sinais do processador (por exemplo utilizando um analisador lógico) **Trindade**¹¹. Um problema que pode ocorrer é quando se considera um sistema com um grande número de processadores amostrando todos os sinais em uma tarefa massiva. Em contraposição, limitando o número de sinais amostrados diminui o nível de informações a serem obtidas sobre o comportamento do sistema.

Alguns monitores de hardware coletam os sinais presentes nos terminais dos processadores e os transferem para um computador pessoal onde os dados são analisados e apresentados em tempo real na forma de gráficos de barras, histogramas ou símbolos coloridos. Alternativamente, dados numéricos podem ser armazenados em um arquivo para posterior análise.

Os monitores de hardware não devem ser intrusivos **Trindade**¹¹, ou seja, devem basicamente ler os sinais e

não alterá-los, pois caso contrário pode-se alterar o comportamento dos sistemas sob investigação.

Comercialmente disponíveis os analisadores lógicos são exemplos de monitores de hardware. Eles geralmente contém canais de alta velocidade para a coleta de dados, um módulo de memória para gravar medidas, e uma tela para mostrar os resultados. Usuários podem configurar várias amostras e estímulos médios e definir eventos complexos para serem traçados. Modelos mais sofisticados são capazes na análise de software, e verificam os conteúdos das localizações de memória Tang¹.

1.5.3.2 MONITORES DE SOFTWARE

Um monitor de software é um conjunto de rotinas de medidas inseridas dentro de um sistema de software que grava o estado prescrito e eventos do sistema. O código é encaixado como um programa separado, com especiais modificações para o software medido, ou para o sistema operacional dele mesmo. Além do espaço de memória ocupado, este requer um sistema de recursos para execução e performance de operação de I/O, deste

modo ocupando de 5% a 10% do overhead do sistema **Rose**¹².

Monitores de software são flexíveis na determinação e realização de mudanças dos parâmetros a serem monitorados. Dados monitorados são extraídos pelo acesso a localização de memória, tais como, sistema de tabela e estrutura de dados, e em ambos, dados descritivos e quantitativos podem ser obtidos.

Como um monitor de software é uma parte integral do sistema medido, este projeto geralmente envolve características do sistema operacional e arquitetura do sistema hospedeiro. Contudo monitores de software são geralmente orientados além de um típico específico de computador e sistema operacional. Eles usam outros sistemas que geralmente envolvem modificações substanciais.

Por exemplo, a série IBM 370 tem uma larga variedade de monitores para o uso com o sistema operacional MVS. O RMF (*Resource Measurement Facility*) é um monitor de software do qual reúne os dados pela detecção de eventos e amostras. Isto mede a utilização de recursos tais como, processadores, dispositivos físicos e canais lógicos. O GTF (*Generalized Trace*

Facility) é um evento que o monitor traça para eventos do Sistema Operacional, tais como, chamadas do supervisor, memórias de alocação e procedimento de paginação.

1.5.3.3 MONITORES HÍBRIDOS

Monitores de software são capazes de extrair dados da memória, mas geralmente causam um certo aumento do *overhead*. Monitores de hardware, por outro lado, causam muito pouca interferência na aquisição de dados, mas podem somente registrar eventos que tenham representação de hardware externamente acessível. Um caminho atrativo é combinar ambos tipos de monitores para produzir um monitor híbrido.

Um monitor híbrido é basicamente uma ferramenta de hardware com uma interface programável. Como um puro monitor de hardware que é invisível para o sistema medido, a porta de hardware de um monitor híbrido é tratado como um "dispositivo periférico inteligente" que pode ser usado pela porta de software do monitor. Em alguns monitores híbridos, um microcomputador é usado para controlar uma função da porta de hardware do monitor Nutt¹³. Em outros casos, o sistema

monitorado tem para a performance uma parte da coleta de dados como uma tarefa dele mesmo, tal como gerar dados na forma que é aceitável pelo monitor de hardware Svodobova¹⁴.

Eventos de software na máquina hospedeira, que são difíceis de detectar por um monitor de hardware puro, pode ser detectado por uma porta de software do monitor, causando uma interrupção. A porta de hardware pode então coletar dados relevantes. A porta de hardware pode iniciar uma interrupção na máquina hospedeira para causar informação de status de software para ser passado sobre a porta de software.

1.5.4 SIMULADORES

Um simulador modela um procedimento dinâmico de um sistema no tempo simulado, o qual procede no intervalo de tempo discreto, possibilitando diferentes comprimentos. O intervalo de tempo simulado no qual nenhum evento ocorre pode ser ignorado quando não existe nada acontecendo no modelo. O *overhead* da coleta dos dados executados pode ser minimizado, assim o *clock* simulado pode estar parado.

Simuladores comercialmente disponíveis são adaptáveis para vários tipos de sistemas, configurações e aplicações. Eles usualmente tem uma biblioteca de algoritmos, e um sistema rígido de entrada. Como máquinas paralelas são fundamentalmente diferentes de máquinas convencionais, simuladores são construídos especificamente para o estudo de um determinado tipo de máquina paralela.

Simuladores podem ser escritos geralmente por linguagens de programação específica tais como Pascal, C, ou Fortran. Essas linguagens não disponibilizam alguma estrutura de dados ou funções para os simuladores propostos, tal como escalonamento de eventos. Linguagens de simulação de propósito especial tem sido desenvolvidas para facilitar a construção do modelo. Eles provêm um trabalho conceitual que o sistema deve adotar, como por exemplo um processo baseado na linguagem SIMULA **Dahl**¹⁵ que visa o sistema como uma coleção de processo interagindo, o qual comporta-se como uma unidade do programa independente. SIMSCRIPT **Markowitz**¹⁶, uma linguagem baseada em evento, define um modelo em termos de eventos, cada tipo de eventos é associada com um procedimento que é disparado quando um evento daquele tipo ocorre. Em uma linguagem baseada em transação, GPSS **Schriber**¹⁷,

elementos ativos são transações que circulam através de uma série de blocos passivos de diferentes funções.

Linguagens de simulação facilitam a descrição do sistema, mas alguns aspectos não podem ser convenientemente encaixado dentro de um esquema conceitual de linguagem de simulação. Para sistemas grandes e complexos, modelos certos são ainda difíceis de construir e os simuladores produzidos dessas linguagens são freqüentemente menos eficientes. Apesar de que, linguagens de simulação oferecerem vantagens de clareza e facilidade de utilização, simplificando a tarefa de construção do modelo.

A maior desvantagem do desenvolvimento em um simulador é o alto custo. Contudo, no caso de desenvolvimento de um sistema de computador, o custo de simulação é geralmente razoável, dado a extensão do projeto. Uma outra desvantagem do simulador é que normalmente eles são ordens de magnitude mais lentos do que as máquinas que eles simulam. No caso de sistemas paralelos complexos, este problema é ainda pior, isto é, em consequência do alto tempo consumido e do custo para a execução da simulação. Por exemplo simulações simples de arquitetura de Fluxo de Dados demoram mais de 60 horas em um Pentium 100 Mhz.

Embora simuladores possam apenas predizer, mas não diretamente medir a atual performance do sistema, o modelo simulativo pode reproduzir no tempo certos aspectos do procedimento dinâmico do sistema. Desde que, o simulador modele o sistema real razoavelmente bem, isto é possível para executar a medida real no modelo, e portanto os simuladores podem ser considerados como uma ferramenta de medição.

1.6 REPRESENTAÇÃO E ANÁLISE DE DADOS

Como a soma dos dados gravados é geralmente considerado, análise manual é difícil e algumas vezes quase impossível. Fica mais complexa devido ao grande número de dados que são geralmente medidos. Contudo, a detenção e análise dos relacionamentos lógicos entre os dados coletados é apenas possível se os dados estão propriamente apresentados.

Representação de dados é algumas vezes diretamente feita pela ferramenta de coleta de dados. Contudo em outros casos, as ferramentas de coleta de dados não são suficientemente sofisticadas para interpretar estes dados e melhorar a informação da performance do sistema, portanto uma ferramenta separada é necessária. Esta ferramenta de apresentação de dados

deve então ser independente da ferramenta de coleta de dados, de maneira que esta possa ser usada com outras ferramentas de coleta de dados, ou outros sistemas de computador.

Representações tabular e textual são simplesmente geradas. Eles que dão uma visão compreensiva e exata dos componentes do sistema, mas ela é freqüentemente dificultada para identificar ou localizar problema de ponto. A Representação gráfica pode dar uma clara indicação de estados relativos do sistema ou na avaliação acima do programa. A representação é geralmente aproximada, mas pode ser melhorada se mostrada junto com alguns dados numéricos.

Entre as mais populares formas estão o histograma e a utilização de gráficos. A utilização gráfica é apresentada na figura 2, que pode mostrar uma combinação lógica de atividades. O comprimento de cada segmento lógico horizontal corresponde a percentagem do tempo que a atividade é verdadeira.

Um aumento na representação popular é a figura Kiviat **kiviat**¹⁸ que é conveniente na descrição de um sistema modelar. Uma figura Kiviat é um círculo com vários cortes saindo do centro do círculo da

Figura 2 - Uma utilização gráfica

Figura 3 - Figura Kiviat de um sistema bem balanceado

Representação textual é inerentemente mais seqüencial e além disto é difícil interpretar muitos elementos concorrentes em máquinas paralelas. A representação gráfica é geralmente mais apropriada na apresentação dos dados monitorados. Contudo, como existe um grande número de concorrência, atividades assíncronas, este resulta em um amplo número de tabelas ou gráficos. O problema real não é determinar o problema ou tipo de gráfico desenhado, mas como agrupar as várias quantidades para dar uma visão integral e mostrar o significado da informação.

Muitas ferramentas foram desenvolvidas para mostrar a monitoração de dados na forma gráfica. Essas incluem a ferramenta **MONA** descrita em **Joyce**²⁰ para monitorar um sistema distribuído, a ferramenta **CUI Cohen**²¹ e um traçado da análise lógica discreta **Disz**²².

A apresentação de dados é conveniente para identificar áreas que necessitam de atenção, mas não faz geralmente sugestão alguma à ação feita, ou à melhoramentos que podem ser feitos.

1.7 FAZENDO UMA ESCOLHA

Simulação é a mais potente e flexível técnica para projetar a performance. Na construção de um novo sistema, a simulação oferece maior vantagem na avaliação do sistema proposto. Contudo, esforço, tempo e custo de construção de um simulador são geralmente considerados. Isto é improvável que um simulador deveria apenas ser construído para propósito de monitoração do sistema, aonde os monitores de software e hardware podem ser usados favoravelmente e facilmente.

Em alguns sistemas paralelos, pode ser atrativo construir um protótipo como uma base para projetar a performance. O protótipo pode ser estendido pela adição de mais hardware sem a necessidade de reconfigurar o sistema.

Para um sistema existente, observações humanas e sistema de contagem podem prover um ponto inicial na avaliação, com experimentos mais exatos de medidas executados mais tarde. A escolha da técnica e da ferramenta irá depender da natureza do índice de performance, e nas características específicas para um particular sistema avaliado. Cada tipo de ferramenta tem vantagens e limitações. Por exemplo ferramentas de hardware podem medir exatamente a utilização da CPU pelo Sistema Operacional. Contudo ferramentas de software são necessárias para medir a utilização de cada módulo individual.

1.8 SUMÁRIO

A avaliação da performance do computador é essencialmente uma decisão do processo, e tal decisão feita é baseada no número de fatores. Monitoração de um sistema de computador requer uma cuidadosa seleção de qual índice de performance avaliar, qual trabalho

de carregamento deriva do sistema, e qual técnica e ferramenta usar para avaliar os índices. Assim a maioria das máquinas paralelas são correntemente pesquisadas e conseqüentemente não disponíveis para a medição direta, simulação é a mais favorável técnica, e simuladores são geralmente propositalmente construídos de forma proposital para um particular sistema paralelo.

A segurança nos resultados fornecidos pelos programas de simulação depende da confiabilidade do simulador e da precisão do modelo utilizado para representar o sistema real. A escolha do simulador adequado, com o modelo correto é a chave para se evitar problemas de simulação **Donlin**²³.

Dados coletados e mostrados devem ser vistos como duas atividade separadas, permitindo que ferramentas de apresentação de dados sejam implementadas sem a necessidade de considerar os detalhes de como os dados monitorados são coletados, e independente do sistema avaliado.

CAPÍTULO 2 - ARQUITETURA A FLUXO DE DADOS

2.1 INTRODUÇÃO

Este capítulo descreve arquitetura baseada em fluxo de dados como a Máquina de Fluxo de Dados de Manchester (MFDM). Faz também a apresentação do simulador MR, pois seus dados de saída são usados pela ferramenta de monitoração de dados que será apresentada no capítulo 5. Faz também uma comparação com outros dois simuladores o SIM (também da MFDM) e o simulador da arquitetura WOLF (SAW).

2.2 MODELO DE COMPUTAÇÃO

Arquiteturas a Fluxo de Dados (Dataflow) exploram o paralelismo ao nível de instrução **Arvind**²⁴. Um programa é descrito como um grafo de dados direto no qual os nós representam instruções e os arcos representam caminhos ao longo do fluxo dos valores

dados. Os valores dados são encapsulados em "fichas" que transversam o grafo. Nós são escalados para execução através de "disparos", sendo executados quando as entradas estão disponíveis, e os resultados são enviados ao longo dos arcos de saída. Um grafo Dataflow simples é mostrado na figura 4.

Figura 4 - Gráfico do Dataflow

O grafo calcula a expressão $(b^2 - 4ac)$. O nó DUP gera duas cópias dos valores de entrada. A execução dos nós é inteiramente assíncrona e depende apenas da disponibilidade dos dados e, no caso de Dataflow estático, da saída de cada nó. Um grande número de nós podem deste modo ser calculados concorrentemente e o paralelismo implícito ao programa é explorado automaticamente.

Grafos representam construções de alto nível, tais como laços condicionais e funções que podem ser

cíclicas. Isto causa problemas de implementação como por exemplo, fichas múltiplas de sinais no mesmo arco. Isto leva a três tipos de modelos abstratos de computação dataflow, que diferem do modo de tratamento de re-entrada de código:

O Esquema *estático* limita o número de instruções imediatas para um **Dennis**²⁵. Isto previne recursão geral, e apenas laços podem ser implementados. O Esquema *dinâmico* usa cada código copiado para criar código apropriado para cada momento **Caluwaerts**²⁶. O Esquema *etiquetado* serve para distinguir entre diferentes momentos, deste modo, uma só cópia do código é compartilhada **Gurd**²⁷, sendo assim permitindo múltiplos momentos.

Em uma arquitetura etiquetada, cada sinal carrega um identificador (etiqueta), e cada laço ou função de invocação gera uma etiqueta diferente para suas fichas. A regra de disparo somente opera para fichas que estão indo para o mesmo nó e têm a mesma etiqueta.

2.3 PROGRAMANDO DATAFLOW

Embora programas dataflow possam ser escritos no nível de máquina, geralmente é mais conveniente escrever em linguagem de alto nível, e usar um compilador para gerar o código dataflow. Uma Linguagem imperativa convencional poderia ser usada, mas um algoritmo de análise para formar gráfico em dataflow de cada linguagem é complexo, como o valor da variável pode mudar durante a computação. A tecnologia atual de computadores ainda tem dificuldades para tratar esse tipo de problema. Uma solução é restringir a definição da variável para apenas uma vez. A ordem de cálculos é então determinada pela definição dos dados.

Linguagem de atribuição única (LAU) (*single-assignment-language*) não tem conceito de execução sequencial e comandos de controle direto, e não há o conceito de variáveis residindo na memória. Ao invés, identificadores são considerados como entidades constante cujos valores não mudam durante a execução do programa. Nesse aspecto, são muito similares às linguagens funcionais. Ao contrário da maioria das linguagens funcionais, as linguagens de atribuição única tendem a usar estruturas de dados tais como matrizes e canais e freqüentemente tem escopo rigoroso

e regras tipadas. Construções tal como laços, que permitem controlar redesignação, são também providas.

LAU são particularmente adaptadas para o dataflow, como os dados podem ser usados tão breve quanto estes são definidos. Exemplos de LAUs são: ID **Arvind**²⁸, VAL **Ackerman**²⁹ e SISAL **McGraw**³⁰.

2.4 ARQUITETURA DA MÁQUINA DATAFLOW DE MANCHESTER

A arquitetura multi-anel de Manchester (MMDM) tem uma arquitetura etiquetada. Uma etiqueta contém um nome de ativação para identificar diferentes invocações de funções ou loops, e um índice para distinguir elementos de uma estrutura de dados.

A estrutura da máquina dataflow é mostrada na figura 5. É composta de um anel de quatro unidades especializadas, que incluem Fila de Fichas, Unidade de Emparelhamento, Unidade de Armazenamento de Grafos e Unidade de Processamento. Cada unidade tem um relógio interno e opera assincronamente na maneira pipeline. Uma estrutura separada armazena a unidade, que é responsável pelo armazenamento das estruturas de dados, e é conectada independentemente para interconecção em rede. Isto consiste de uma unidade de

alocação, uma unidade de liberação de espaço livre, uma estrutura de memória, e uma lista de acesso de transferência. Mais detalhes foram publicados por Gurd Gurd²⁷, e cada unidade é descrita sucintamente abaixo :

Fila de Fichas -> É um buffer com a disciplina *first-in-first-out* para igualar médias desiguais de geração e de consumo de fichas. Este armazena temporariamente as fichas até a unidade de emparelhamento estar disponível para processá-los.

Unidade de Emparelhamento-> Tem a tarefa de emparelhar fichas consecutivamente. Para evitar o uso de memória puramente associativa usa-se uma estratégia *hashing*. Uma função de *hash* é aplicada para a etiqueta e destinação de campos de fichas que estão para chegar, e o resultado de um endereço é pesquisado nos vários bancos de memória em paralelo. Se nenhuma combinação de fichas for encontrada, a ficha de entrada é armazenada. Por outro lado, a ficha de combinação é extraída da armazenagem e o par de ficha é formado dentro de um pacote para ser enviado ao nó de armazenamento. Fichas destinadas a operadores monádicos simplesmente contornam a unidade de emparelhamento.

Nó de Armazenamento -> Contém o programa código codificado como Grafo de Dependência de Dados. Um par de fichas chega na unidade de emparelhamento buscando o código de instrução, e o pacote executável resultante é então enviado a unidade de processamento.

Unidade de Processamento -> Contém funções de unidades de propósito geral (FUs). Um pacote executável é enviado para qualquer FU pelo distribuidor. A instrução é então executada, e o resultado coletado por um árbitro para uma entrada na lista de sinais via interconecção de rede.

Unidade de Alocação -> É responsável pelo gerenciamento e armazenagem do espaço para estrutura de dados.

Unidade de Liberação -> Executa coleta de lixo quando uma estrutura de dados não é mais usada.

Estrutura de Memória -> Armazena uma estrutura de dados e permite acesso assíncrono para ela. Estruturas de dados são armazenadas sem etiqueta, permitindo mais eficiência no uso da armazenagem.

Lista de Acesso à transferência -> Armazena a leitura requerida se os dados requeridos não estão ainda

escritos. Esse requerimento será liberado quando o correspondente dado estiver escrito.

Figura 5 - Estrutura da máquina dataflow

2.5 CONTROLE DE PARALELISMO

Tem sido observado que muitos programas tem muito mais paralelismo do que a máquina pode explorar, pelo menos durante um período de sua execução Ruggiero³¹. Esta união imperfeita do programa e paralelismo da máquina tem conduzido ao uso excessivo de armazenagem de emparelhamento e lista de sinais,

como muitos resultados parciais são gerados antes deles serem usados.

Uma técnica de controle de paralelismo proposta é o *throttling* **Ruggiero**³², um mecanismo de software *throttling* que controla o paralelismo restringindo a quantidade de instâncias diferentes que podem coexistir num determinado momento da computação.

Um processo corresponde para uma chamada de função ou laço invocação, é iniciado pela geração de um novo nome de ativação. *Throttling* trabalha pelo controle do número de processos ativo no sistema, o qual é um retorno ao controle pela alocação dos nomes de ativação.

Um requerimento para iniciar um novo processo é efetuado pela execução da instrução GAN (*Generate Ativation Name*), a máquina então decide se inicia o processo imediatamente pela alocação em um nome de ativação ou suspende ele até que os recursos estejam disponíveis para processá-los, quando um processo termina, este nome de ativação pode ser reusado, e um novo processo pode iniciar neste lugar.

2.6 MR - O SIMULADOR MULTI-RING

O trabalho na Universidade de Manchester envolveu a investigação de uma versão de multiprocessador na máquina dataflow. Um software simulador, MR, foi implementado em Linguagem Pascal. A máquina multi-ring consiste de alguns anéis conectados por uma chave. Há três tipos de anéis, como mostra a figura 6. NR Processadores de elementos (PEs), NS Estruturas de Armazenamento (SSs) e 1 Alocador Global (GA) que podem ser simulados.

Figura 6 - Estrutura do simulador multi-ring

O intervalo de monitoramento é dado em ciclos medidos usando um relógio interno global ao simulador. Esse intervalo pode ser determinado pelo usuário através da linha de comando. Dois arquivos são criados com a simulação: um contém o resultado do programa e outro as mensagens referentes ao desempenho do

simulador como ocupação da memória de dados, ocupação da fila de fichas, utilização da memória de estruturas e informações referentes aos processadores

Para facilitar a experimentação, diversas características da máquina simulada foram parametrizadas. Pela especificação alternada ao conjunto de valores de parâmetro, várias configurações do sistema podem ser simuladas. Os parâmetros incluem tipos e atraso de chave, tempo de acesso na estrutura de armazenamento, tempo de alocação para a estrutura de dados, NS ou NR, que assume um valor padrão pré-definido se não for especificado.

O Simulador MR é resumidamente descrito a seguir, baseado na descrição de Teo **Teo**³³.

2.6.1 A CHAVE

A chave usa uma função aleatória para distribuir trabalho igualmente entre as PEs. Esta função é aplicada para o pacote de sinais chegando na chave, produzindo o número de PEs para o qual o sinal é direcionado. A chave de execução da MMDM usa topologia Banyan. O MR simplifica a chave para consistir de NR+NS+1 listas. Pacotes de sinais ficam na chave por

pelo menos o passo de tempo SwDelay, que pode ser alterado pelo usuário.

2.6.2 O PROCESSADOR DE ELEMENTOS

O processador de elementos faz o processamento dos modelos de hardware, com muitas simplificações. O nó armazenado é simulado por um buffer. A velocidade das FUs no simulador MR é maior do que no protótipo MFDM. Apenas metade dos buffers do protótipo da máquina é simulada para diminuir a soma do paralelismo necessária para saturar um PE. Essas simplificações permitem executar o MR com programas de menor tamanho.

2.6.3 ESTRUTURA DE ARMAZENAMENTO

Os pedidos de estruturas requeridas são enviadas pelas PEs para estrutura de armazenamento, e respostas são retornadas via uma chave. A estrutura de armazenamento simulada consiste de uma lista de fichas, que temporariamente armazenam pacotes de fichas excessivos antes enviados a eles para o distribuidor. O distribuidor manda os pacotes para cada alocador, um módulo de controle de armazenagem ou módulo de liberação, e os sinais produzidos por esses módulos são coletados pelo arbitrador.

O alocador é responsável pelo espaço de armazenagem, usando um método de página livre proposto por Sargent **Sargent**³⁴. O módulo de liberação usa um esquema contínuo de referência para recuperar o espaço de armazenagem quando uma estrutura de dado não é usada. O módulo de controle de armazenagem divide os acessos a memória tais como leitura e escrita.

2.6.4 ALOCADOR GLOBAL

O alocador global é responsável pelo gerenciamento de estruturas maiores, que são alocados com entrelaçamento. O tempo de acesso dos elementos dessas matrizes é crítico, e assim é desejável que muitos acessos possam ser executados em paralelo.

O alocador global é também responsável pelo controle do processo. Uma instrução GAN (executada quando se dispara um novo processo) causa uma PE para enviar um requerimento para o alocador global para iniciar um novo processo. O requerimento é pendurado numa lista apropriada. O alocador global usa uma PE média no comprimento de listas de sinais para medir a atividade da máquina. No intervalo, a atividade da máquina é inspecionada e uma decisão é feita como se um processo pendurado deveria ser tornado ativo.

2.6.5 MONITORAMENTO

O MR modela o tempo e reúne ambos programa e máquina estatística. Máquinas características são amostradas em todo tempo referido, o passo de tempo referido é um parâmetro específico. Cada unidade tem um número de características que são de interesse. Para PEs, estas incluem: comprimento de lista de fichas, armazenagem de emparelhamento ocupada e FUs usadas. Para SSs, as características tais como: TQ comprimento da fila de fichas, Armazenagem ocupada, número de acesso à armazenagem e alocações necessitam ser monitoradas. No caso de alocador global, dados de interesse incluem números de atividades e processos pendurados, e um número grande de estruturas de dados.

Estas estatísticas são mostradas *on-line*, e também são armazenadas no arquivo monitor para subsequente análise. O mostrador de fichas *on-line* detecta um ponto quente ou frio. Por exemplo, se uma PE tem uma característica que mostra grande discrepância com outras PEs, aquela característica é realçada.

Informações de monitoração relativo aos procedimentos do programa são coletadas em um evento dirigido ao gerente. Quando um processo é pendurado, ativado ou terminado, a informação relevante é escrita

dentro de um monitor de arquivo separado em forma textual.

Além da simplificação já mencionada o MR não pode modelar a natureza não determinística do protótipo da máquina, e a máquina simulada trabalha de maneira assíncrona. Contudo, estudo de comparação mostra que o MR é um modelo razoável **Teo**³², e o MR tem sido usado como a principal ferramenta na performance de avaliação da máquina de fluxo de dados multi-ring.

2.7 OUTROS SIMULADORES

Existem dois outros simuladores, um usado pelo grupo de Manchester (SIM) e o outro é o simulador da arquitetura WOLF (SAW) desenvolvido numa colaboração entre o Grupo de Instrumentação e Informática (GII) da Universidade de São Paulo (USP) e o Grupo de Fluxo de Dados da Universidade de Campinas (UNICAMP).

2.7.1 SIM

É um simulador funcional simplificado da MFDM que não possui conceito de tempo, e simula um único anel. Seus módulos são ativados pela presença de dados de

entrada de cada módulo, e o paralelismo não é muito bem definido. O simulador roda sob o sistema operacional VMS e UNIX. SIM usa como entrada três arquivos: o arquivo de código (.COD), o arquivo de dados (.DAT) e o arquivo de segmento (.SEG). Esses arquivos são gerados durante a compilação do programa em SISAL e esse processo está ilustrado na figura 7. Durante a sua execução, o SIM gera dois arquivos: um é usado para armazenar as fichas resultantes da simulação e o outro é usado para armazenar os dados sobre monitoramento da simulação.

```
$ SIM/RS=100 PROG1
  Initialising
  Reading program and data

S1  Sinf  Pi   MS   TQ   Def  Res   SS acc  occ  all  DAQ us len
-----
100  92   1.1  119   1   0   0
200  192  1.0   19   1   0   0

Natural termination of simulation
Total instructions executed (S1)           221
Critical path length (Sinf)                213
Average parallelism (Pi)                   1.0
Peak parallelism                           3
Number of result tokens                     1
Proportion of bypass actions (Pby)         0.081
Total token queue traffic                   424
Maximum token queue length                 103
Maximum matching store occupancy           198

Matching store empty
Token queue empty
Structure store not used
$
```

Figura 7 - Arquivo gerado pelo Simulador SIM

Ao término da simulação, o SIM produz uma lista de dados que contêm várias estatísticas sobre simulação. Algumas dessas estatísticas são:

1. O número total de instruções executadas;
2. O tempo necessário para executar o programa em uma máquina ideal com um número infinito de processadores;
3. O paralelismo médio que é dado pelo quociente das duas grandezas acima. Essa grandeza fornece uma medida do paralelismo do programa;
4. O número de fichas produzidas;
5. A razão entre o número de operações em ponto flutuante por segundo e o número total de instruções executadas;
6. O tráfego na *token queue*. Essa grandeza mede o número de fichas que passaram pelo anel;
7. O comprimento *token queue* e a ocupação máxima da *matching store*. Essas duas grandezas medem a quantidade de memória utilizada durante a simulação.

O arquivo de monitoramento contém informações mais detalhadas sobre a execução do programa. Por exemplo, erros encontrados em tempo de execução (divisão por zero, etc) são armazenados nesse arquivo.

O intervalo de monitoramento pode ser controlado através da passagem de um parâmetro através da linha de comando. Como o simulador não possui o conceito de tempo, o conceito de ciclos é dado pelo número de instruções executadas.

Esse simulador foi implementado usando a linguagem PASCAL. Maiores informações podem ser encontradas em Sargeant **Sargent**³⁴.

2.7.2 SAW

O SAW simula uma versão simplificada da arquitetura WOLF. O simulador SAW foi implementado em estação de trabalho SUN, usando a linguagem orientada a objetos C++ versão 2.0 da AT&T feita por Marcus **Cavenaghi**³⁵.

A estrutura do simulador, é constituída pelas unidades interconectadas mostradas na figura 8. Essa

versão não explora características da arquitetura original tais como granularidade variável e tratamento de matrizes e vetores através de uma memória de estruturas. A arquitetura explora granularidade ao nível de instrução (fina), e não identifica instruções que fazem parte de uma cadeia, como a arquitetura WOLF original se propõe a fazer.

Os dados são agrupados em estruturas de dados globais ao programa simulador. Essas estruturas são passadas de objeto a objeto e através delas que os objetos comunicam-se entre si. A chegada de uma estrutura de dados ao objeto simula a chegada de uma ficha em uma parte do WOLF. Com a chegada, um objeto é ativado e então começa a operar sobre a estrutura. Dessa forma a operação pode ser classificada como sendo dirigida a evento onde cada evento pode ser entendido como sendo a chegada de uma estrutura de um objeto. Os resultados da simulação são apresentados no monitor de vídeo da estação de trabalho hospedeira e através de arquivos de dados que são criados durante a simulação.

Figura 8 - Estrutura do Simulador SAW

O simulador roda sob o sistema UNIX. SAW usa como entrada três arquivos: o arquivo de código (.COD), o arquivo de dados (.DAT) e o arquivo de segmento (.SEG) e gera dois ou três arquivos de saída, dependendo da opção usada durante a execução do SAW.

2.8 RESUMO DAS CARACTERÍSTICAS DOS SIMULADORES

A tabela V mostra as principais características dos simuladores MR, SIM e SAW.

Tabela V - Principais características dos simuladores MR, SIM e SAW

Simulador	Linguagem Fonte	Escalonamento	Ambiente	Intervalo de monitoramento
MR	Pascal	Dirigido a Tempo	VMS e UNIX	Número de Ciclos
SIM	Pascal	Dirigido a Eventos	VMS e UNIX	Número de Instruções Necessárias
SAW	C++	Dirigido a Eventos	UNIX	Número de Ciclos

CAPÍTULO 3 - A INTERFACE GRÁFICA

3.1 HISTÓRICO DO SISTEMA XWINDOW

O desenvolvimento do sistema XWindow iniciou em 1984 no *M.I.T.* (*Massachusetts Institute of Technology*) sob o patrocínio do Laboratório para Ciência de Computação e do Projeto Atenas, sendo ambos do *M.I.T.*. Desde o início, o X teve suporte industrial da DEC e IBM por causa do projeto Athenas. Por volta de 1986 a DEC introduziu a primeira implementação comercial do X executando em um *VAXstation-11/GPX* em cima de um sistema operacional *Ultrix*. Esta era a versão X10 release 3 (*X10R3*). Breve X atraiu a atenção de alguns outros proeminentes vendedores de estações de trabalho, tais como *Hewlett-Packard*, *Apollo Computer*, *Sun Microsystem*, e *Tektronix*.

O *Feedback* dos usuários do X10 apressaram membros do projeto para iniciar um maior redesenho do protocolo X. Enquanto o projeto daquele que se

tornaria X versão 11 (X11) estaria em andamento. X10R4 foi lançado em dezembro de 1986, esta era a última revisão da versão X10.

Em janeiro de 1987, durante a primeira conferência técnica X, os onze maiores vendedores de computadores anunciaram um esforço conjunto para padronizar a versão X11. A primeira revisão do X11, X11R1, tornou-se disponível em setembro de 1987. Para garantir uma evolução do X acima de uma organização de controle aberto o *M.I.T. Consortium* foi formado em janeiro de 1988, sendo o chefe de projeto Robert W. Scheifler **Scheifler**³⁶, um dos principais arquitetos do X, sendo assim o consórcio teria uma razão maior para o sucesso do X.

Em março de 1988, X11R2 tornou-se disponível. X11R3 foi lançado no final de outubro de 1988 em janeiro de 1990 o consórcio lançou X11R4, em 1991 o X11R5 e em 1993 o X11R6.

3.2 O SISTEMA XWINDOW

A equipe do MIT desenvolveu o XWindow para ser um gerenciador de janelas distribuído independente de dispositivo, capaz de fazer o processamento

multitarefa e preemptivo com recursos gráficos **Jonhson**³⁷.

O sistema XWindow é um sistema baseado em um modelo cliente-servidor. O servidor X processa, executando na estação de trabalho com uma tela gráfica de mapa de bits (Bitmapped), gerenciando regiões da tela conhecida como janela, onde a saída de uma aplicação do cliente X aparece. Um cliente X, de qualquer forma, executando localmente ou em um computador remoto, envia requerimento para o servidor usando um canal de comunicação. Os bytes transferidos entre um cliente e o servidor obedece um protocolo X. X versão 11, X11, consiste de um servidor X, um protocolo X, e Xlib (uma biblioteca de rotinas em C que programadores usam para acessar o servidor. Os projetistas do X visaram prover apenas primitivas com capacidade de construções necessárias para suportar interação com o usuário. Eles impuseram como um usuário deveria olhar e sentir. Deste modo, X pode ser usado para construir uma grande variedade de interfaces com o usuário.

Pelo oferecimento de um sistema de janelas padrão, o X permite a clientes e servidores interoperacionar: um cliente X em um sistema pode mostrar sua saída em

algum outro servidor X, sem respeitar a tela da máquina ou o sistema operacional. Uma estação de trabalho pode tornar-se uma parte integral de um ambiente tão longo quanto este possa suportar X e pode estar em rede com o resto do sistema de servidores X. Servidores X estão disponíveis mesmo para diversas plataformas e diversas versões do UNIX **Barkakati**³⁸.

Figura 9 - Uma estação com o sistema XWindow

A janela 1 da figura 9 mostra um terminal para a estação. A janela 2 mostra a saída de uma aplicação X que também está executando na estação de trabalho. A janela 3 é uma outra janela terminal onde pode-se estar interagindo com um computador A, enquanto a saída de outra aplicação X executando em um computador B aparece na janela 4.

3.2.1 MODELO CLIENTE-SERVIDOR

O servidor disponibiliza um serviço requerido pelo cliente. Geralmente, o cliente se comunica com o servidor através de uma rede, e o cliente e servidor trocam dados usando um protocolo que ambos entendem nas estações. Pode-se ver este modelo em ação. Por exemplo um *servidor de arquivos* armazena arquivos e permite ao cliente acessar e manipular estes arquivos. A figura 10 mostra o modelo cliente-servidor.

Fig 10 - Modelo Cliente/Servidor

3.2.2 XPROTOCOL: A LINGUAGEM DE MÁQUINA DO X

O protocolo é um acordo entre o servidor e o cliente sobre como eles irão trocar informações e como as informações serão interpretadas. Desenhando uma analogia com microprocessadores, pode-se dizer que o protocolo X é uma *linguagem de máquina* do sistema XWindow. Apenas como um circuito lógico em um microprocessador interpreta os padrões de bits nos bytes de instruções e executam tarefas simples, o servidor X interpreta o fluxo de bytes do protocolo X e gera saídas gráficas.

3.2.3 XLIB: A LINGUAGEM MONTADORA DO X

Pode-se escrever uma aplicação que usa o servidor X pelo envio diretamente de bytes conforme o protocolo X, mas fazer isto é muito tedioso (igualmente a programar um microprocessador usando linguagem de máquina). Contudo você não precisa fazer isto. O Sistema XWindow vem com uma biblioteca de rotinas em C comumente referenciada como Xlib. Xlib dá à você acesso ao protocolo X através de mais de 300 rotinas utilitárias. Se o protocolo X é a linguagem de máquina do X, então Xlib é a linguagem montadora. Programar em linguagem montadora é complexo, mas é mais fácil do que em linguagem de máquina.

Desenvolvedores de aplicações não trabalham diretamente com o protocolo X. Eles usam rotinas Xlib ou uma das muitas ferramentas (*toolkits*) X, que provêm objetos de alto nível *widgets** para construir uma interface de usuário. Mesmo quando se usa uma ferramenta, operações básicas de desenvolvimento usualmente requer que se use rotinas Xlib **Barkakati**³⁸.

**Widgets* - são os diversos objetos que formam a interface, por exemplo : janelas, botões, ícones, etc.

3.2.4 XTOOLKITS: A LINGUAGEM DE ALTO-NÍVEL DO X

O Xlib não tem funções que irão mostrar na tela um menu com uma lista de entrada para seleção. Pode-se criar um menu chamando um número de rotinas Xlib, mas isto é uma tarefa árdua. Para resolver este problema, você necessita de um outro conjunto de rotinas para implementar objetos tais como botões, listas e menus que podem ser usados para construir uma interface gráfica do usuário. Esta idéia foi proposta por vários grupos. O sistema XWindow vem com *X Toolkits Intrinsics* (também conhecida como *Xt Intrinsics*), o qual usa um acesso orientado à objeto para implementar construções de blocos básicos chamados *widgets*. Algumas outras ferramentas, tais como *Motif toolkit* usam um alto nível de abstração. A ferramenta *Motif* foi construída em cima do *X Toolkit Intrinsics*. A ferramenta *XView* é equivalente a *Xt Intrinsics*. Como *Xt Intrinsics*, *XView* foi construída sobre Xlib. Continuando com a analogia para programação de microcomputadores, estas são linguagens de alto nível do sistema XWindow.

A figura 11 mostra uma estrutura geral de uma aplicação X.

Figura 11 - Estrutura geral de uma aplicação X

3.2.5 DEPENDÊNCIA NO SISTEMA OPERACIONAL

O projeto básico do sistema X é independente de um sistema operacional específico. Todo X necessita de um mecanismo confiável dos dados entre o cliente e o servidor X. Até agora, implementações X têm usado protocolos de redes TCP/IP, DECNET e STREAMS para transferência de dados entre o servidor e cliente. Por exemplo, no sistema UNIX Berkeley (também conhecido

como BSD UNIX), O sistema X comunica-se usando TCP/IP, por outro lado nas máquinas Digital, os bytes do protocolo X são enviados usando protocolo de rede DECNET.

Implementações AT&T do X usam o mecanismo STREAMS nativo para o sistema UNIX V. Contudo, tão longe quanto existe um protocolo de rede comum disponível para transferir dados, o servidor X pode mostrar a saída de algum cliente sem respeitar o sistema operacional acima do qual o cliente executa.

3.2.6 INTERFACE GRÁFICA DO USUÁRIO E O X

O termo *graphical user interface* (GUI) descreve uma interface de usuário que faz uso de janelas, menus, e outros objetos gráficos e que, para uma grande extensão, permite usuários interagirem pelo click dos botões do mouse. Do ponto de vista de um desenvolvedor de aplicações, uma GUI é uma combinação de um gerenciador de janela, um condutor de estilo e uma biblioteca de rotinas (toolkit) que pode ser usada para construir uma interface de usuário.

O sistema XWindow contém funções básicas que podem ser usadas para a construção de uma interface gráfica

com o usuário (GUI), mas não requer que o programador siga algum estilo específico. Por causa disto, o X mesmo não é uma GUI, mas muitas GUIs são construídas baseadas no X. OSF/MOTIF e OPEN LOOK são exemplos de tais GUIs. Cada uma dessas GUIs tem seus próprios estilos e terminologias únicos, mas nenhum oferece substanciais benefícios acima do outro. Ambos OSF/MOTIF e OPEN LOOK produzem adequadas facilidades para construções de interface de usuários para suas aplicações. A escolha de um ou outro é uma decisão mais pessoal do que técnica.

Figura 12 - Área de trabalho OPEN LOOK

3.3 XVIEW

Xview (*XWindow System based Visual Integrated Enviroment for Workstation*) é uma ferramenta de interface com o usuário para suportar gráficos interativos baseados em aplicações executando acima do sistema XWindow. Esta ferramenta foi desenvolvida pela Sun Microsystem Inc **Heller**³⁹.

Como qualquer ferramenta X, o Xview provê um conjunto de objetos pré-construídos de interface para o usuário, tais como menus, janelas, subjanelas, botões de controle e barras de rolagem. A aparência e funcionalidade desses objetos seguem a Open Look GUI. Juntamente desenvolvida pela Sun Microsystem e AT&T como uma interface-usuário gráfica padrão para o Sistema V versão 4, Open Look provê aos usuários uma simples, consistente e eficiente interface para executar tarefas dentro de uma aplicação.

3.3.1 XVIEW E O SISTEMA XWINDOW

A ferramenta XView permite que programadores construam a interface para uma aplicação sem ter que aprender muitos detalhes fundamentais do sistema

XWindow. Contudo, é importante ter algum conhecimento do sistema X antes de tentar construir aplicações em cima do XView.

3.3.2 PROGRAMAÇÃO ORIENTADA À OBJETO

Para o programador, o XView é classificada como uma ferramenta orientada à objetos. Objetos Xview podem ser considerados blocos construídos para o qual a interface usuário de uma aplicação é montada. Cada pedaço pode ser considerado como objeto de um particular pacote. Cada pacote provê uma lista de propriedades para o qual você pode escolher a configuração de um objeto. Pela seleção de um objeto de um pacote você pode construir a interface com o usuário para uma aplicação.

O XView é baseado nos vários princípios fundamentais de programação orientada a objeto:

- * Objetos são representados em classes hierárquicas.
- * Objetos são tipos de dados opacos.
- * Objetos tem atributos que podem ser estabelecidos via funções passando mensagens.
- * Objetos podem ter retorno de procedimentos que são despachados por eventos.

3.3.2.1 CLASSE HIERÁRQUICA DO OBJETO

O XView define classes de objetos em árvores hierárquicas. Por exemplo, *frame* é uma subclasse da classe mais geral *window*, o qual é uma subclasse de *drawable*. A Figura 13 mostra a classe hierárquica do XView e os relacionamentos entre as classes. Cada classe tem formas idênticas que fazem dela única em relação às outras classes ou pacotes. Em XView uma classe é freqüentemente chamada de pacote, contudo, existem pacotes que não são membros da classe hierárquica dos objetos, tal como o pacote *Notifier*.

Figura 13 - Classes Hierárquicas do XView

Alguns objetos são visuais e outros não são. Objetos visuais incluem janelas, barra de rolagem, moldura, botões e itens de botões entre outros. Objetos não visuais são objetos que não tem aparência, mas têm informações que permitem mostrar um objeto visual. Exemplos de objetos não visuais incluem o servidor, a tela e os objetos fontes. Todos os objetos visuais e não visuais fazem parte deste sistema de classe objeto. O sistema é extensível, e novas classes podem ser criadas, baseadas ou não na existência de classes. Todos os objetos de uma particular classe herdam as propriedades da classe pai.

O XView usa subclasses para que cada pacote possa herdar as propriedades da sua superclasse. O pacote *Panel* é subclasse do pacote *Window*, que tem propriedades específicas para todas as janelas, tais como dimensão da janela, localização da tela, espessura da borda, profundidade, informações visuais e mapa de cores.

3.3.2.2 CHAVE DE OBJETO (OBJECT HANDLES)

Quando um objeto é criado a função do XView retorna uma chave para este objeto. Depois, quando deseja-se manipular o objeto ou inquirir sobre seu

estado, esta chave é passada para uma função apropriada. Esta segurança no objeto chave é uma forma de esconder a informação. As chaves são opacas no sentido que não pode-se ver dela para uma atual estrutura de dados que representa o objeto.

Há dados cujos tipos são indexados. Para esses tipos de dados opacos, existem vários **typedefs** que se referem não para ponteiros, mas para estruturas: **event**, **rect**. Geralmente ponteiros para essas estruturas são passadas para funções XView, assim elas são declaradas como **Event***, **Rect***, etc... Os asteriscos não estão incluídos no **typedef** porque as estruturas estão publicamente disponíveis.

3.3.3 FUNÇÕES BASEADAS EM ATRIBUTOS

A idéia básica através de uma interface XView é prover um pequeno número de funções, que tenham como argumentos um grande conjunto de atributos. Para uma dada chamada para criação ou modificação de um objeto, apenas um subconjunto do conjunto de todos os atributos aplicáveis serão de interesse.

3.3.3.1 CRIANDO E MANIPULANDO OBJETOS

Existe um conjunto comum de funções que permitem ao programador manipular alguns objetos pela referência a objetos chave.

Tabela VI - Funções de manipulação dos objetos

Funções	Descrição
xv_init	Estabelece a conexão para o servidor, inicializa o notificador, carrega o servidor e lê o arquivo ~/Xdefaults e alguns atributos passados
xv_create	Cria um objeto
xv_destroy	Destrói um objeto
xv_find	Procura um objeto através de certos critérios, ou se o objeto não existe cria-o
xv_get	Consegue o valor de um atributo
xv_set	Altera o valor de um atributo

Usando, estas seis rotinas, objetos podem ser criados e manipulados de todos os pacotes disponíveis em Xview.

3.3.3.2 TIPOS DE ATRIBUTOS

Atributos podem ser divididos em três categorias. Aqueles que se aplicam para todos os objetos XView são chamados de atributos genéricos; Atributos que são

suportados por vários, mas não todos os objetos são chamados de atributos comuns; Atributos que são associados a um particular pacote ou classe de objeto são chamados de atributos específicos.

O XView usa nomeações convencionais para simplificar a identificação de uma tarefa no atributo. Aqueles atributos que aplicam-se para um específico pacote, tem seu nome prefixado pelo nome do pacote. Estes atributos tem prefixos que indicam o tipo de objeto em que ele se aplicam, CANVAS_* , CURSOR_*, FRAME_*, ICON_*, PANEL_*, SCROLLBAR_* ,etc...

Atributos genéricos e comuns aplicam-se sobre vários tipos de objetos diferentes e são prefixados por XV_* . Por exemplo, o atributo XV_HEIGHT aplica-se para todos os objetos, assim todos os objetos devem ter altura. Em contraste atributos que aplicam-se apenas para janelas são prefixadas por WIN_*.

3.3.4 TIPOS DE OBJETOS

A seguinte seção descreve um nível conceitual de diferentes tipos de objetos que XView oferece. Em muitos casos, figuras trazidas do guia de especificação OPEN LOOK GUI são usados para mostrar a

aparência do objeto, detalhes sobre o objeto, como criar eles, suas propriedades e seus valores *default*.

A lista de objetos que podem ser criados incluem:

Objetos Genéricos,Windows, Frames, Openwins, Canvases, Text Windows, Menus, Scrollbars.

3.3.4.1 OBJETOS GENÉRICOS

O objeto genérico é um objeto raiz de uma classe hierárquica. A figura 14 mostra o caminho trilhado quando um objeto é criado.

Figura 14 - Criação de objetos é top down;
Estabelecer atributos é bottom up

O processo de criação é o seguinte:

Primeiro o objeto genérico é criado; então a subclasse daquele objeto é criada por todo o caminho abaixo até a classe de um tipo de objeto desejado ser criado. Naquele ponto, uma completa instância do objeto que foi criado com todas as propriedades default do conjunto classe. Se existe algum par específico valor-atributo em uma chamada `xv_create()`, aqueles atributos são definidos em ordem reversa (e os atributos específicos para uma classe de uma instância de um objeto são definidos primeiro, seguidos por seus atributos de classe pais e assim por diante, até o atributo genérico ser definido).

3.3.4.2 OBJETOS WINDOWS

Muitos objetos XView contêm XWindow em disposição para mostrar na tela eles mesmo e receber eventos. Exemplos incluem : frames, scrollbars e ícones.

A classe `window` do XView, como a classe de Objetos Genéricos, é uma classe oculta; um objeto `window` nunca é criado explicitamente. Particularmente, um objeto que está em uma subclasse da classe `window` é criado.

Isto inclui a maioria dos objetos visuais com exceção dos itens de painel.

Objetos não visuais são então nomeados, porque eles não contém, ou não são subclasses da classe *window*. Por exemplo, Fontes são mostrados na janela, ou na memória da imagem ou em algum lugar que contém um bitmap, mas fontes não contém ou requerem janelas para serem usados.

3.3.4.3 FRAMES E SUBFRAMES

Existem dois tipos de frames:

Base Frame

Pop-up Frame

Os *base frame* residem na raiz da janela. Subframes são quadros cujo dono é um *frame*; Eles são controlados por um *base frame* na aplicação. A figura 15 mostra um exemplo de um *base frame* no OPEN LOOK GUI.

Figura 15 - Base Frame

Pop-up frames são tipicamente utilizados para executar uma ou mais funções transientes. Eles não são intencionados a permanecer depois de um conjunto de funções terem sido executadas, embora eles possam permanecer se o usuário ou aplicação assim escolher. Existem vários tipos de *pop-up frames* :

Command frames - Dá ao operando um conjunto de parâmetros necessários para a execução de um comando.

Help frames - Mostra um texto de ajuda sobre um objeto apontado.

Notices - São janelas pop-up especiais que são usadas para confirmar uma solicitação.

A figura 16 mostra um exemplo de *command frame*; a figura 17 mostra um exemplo de um *help frame* e a figura 18 mostra um *simples notice*.

Figura 16 - Command frame

Figura 17 - Help frame

Figura 18 - Notice

3.3.4.4 SUBWINDOWS

As *Subwindows* diferem do *frame* em vários aspectos básicos. Eles nunca existem independentemente e estão sempre sob ordens e são mantidos por um *frame* ou outra *window*. Enquanto *frames* podem se mover livremente ao redor da tela, *subwindows* são obrigados a caber dentro da borda de um *frame* para o qual ele pertence.

Tabela VII - Subwindows e suas respectivas funções

Canvas Subwindow	Produz uma superfície de desenho, um lugar no qual o resultado de uma chamada gráfica Xlib possa ser mostrada.	
Text Subwindow	Produz capacidades básicas de edição de texto usando o modelo de edição de texto OPEN LOOK.	
Panel	Controle de área, faz o gerenciamento de uma variedade de itens de panel: <i>Botões, Lista de escolha exclusiva e não exclusivaSliders, Texto e campo numérico</i>	
Menus	São subclasses do objeto genérico e em XView temos três tipos de menus :	
	Pop_up menus	São aqueles mostrados quando o usuário pressiona o botão de menu na janela.
	Pull_right menus	São aqueles mostrados como um menu à direita do menu.
	Pull_down menus	São aqueles mostrados abaixo de um botão de menu ou panel.
Scrollbars	Um scrollbar é um objeto que pode existir independentemente ou ligado	
Icones	Um ícone é uma pequena imagem representando a aplicação, quando a aplicação do frame é fechada.	

3.3.4.5 OBJETOS NÃO VISUAIS

Existem vários tipos de objetos não visuais que podem ser representados na tela, mas são subclasses do objeto genérico:

Tabela VIII - Objetos não visuais

Server	Este pacote interage com o XServer. A janela server é um programa que faz o desenho na tela e recebe entrada do usuário.
Screen	Este objeto descreve o visual e outras características da tela física.
Font	Este pacote permite ao programador requerer fontes de vários atributos, tais como (Font, Family, Style). Fontes podem ser acessados por nome tamanho e escala.
Cursor	O cursor ou ponteiro é um objeto visual que indica a localização do mouse na tela.
Cms	Segmento de mapa de cores são objetos que são associados com janelas, o qual provem suas especificações de cores. Objetos cms podem ser compartilhados por múltiplas janelas.

3.3.5 O MODELO NOTIFIER

XView é um sistema baseado em notificações. O *Notifier* age controlando entidades dentro de um processo de usuário, lendo entradas do Sistema Operacional e formatando estas dentro de **eventos** de alto nível, o qual este distribui para diferentes objetos XView.

3.3.5.1 ESTILO DE PROGRAMAÇÃO CALLBACK

Em um estilo de programação interativo, o principal loop de controle reside em uma aplicação. Um editor por exemplo, irá ler um caracter, fazer alguma

ação baseada no caracter, então ler o próximo caracter e assim por diante. Quando um caracter que é recebido representa um requerimento do usuário para sair, o programa termina. A figura 19 representa em um fluxograma este estilo.

Fig 19 - Fluxo de controle em um programa convencional

Sistemas baseados em notificações invertem esta posição de linha da estrutura de controle. O principal loop de controle reside no notificador, não em uma aplicação. O Notificador lê eventos e notifica, ou

chama para fora, vários procedimentos o qual uma aplicação tem prévios registros com o Notificador. Esses procedimentos são chamados *notify* ou *callback* procs. Esta estrutura de controle é mostrada na figura 20.

Figura 20 - Fluxo de controle em um programa baseado em Notificador

3.3.5.2 PORQUE UM SISTEMA BASEADO EM NOTIFICAÇÃO?

Em XView, uma aplicação típica tem muitos objetos. Na falta de um notificador centralizado, cada aplicação deve ser responsável pela detecção e despachos dos eventos para todos os objetos no processo. Com um Notificador centralizado, cada componente de uma aplicação recebe apenas os eventos do usuário que são direcionados para ele mesmo.

3.3.5.3 RELACIONAMENTO ENTRE O NOTIFICADOR, OBJETOS E UMA APLICAÇÃO

XView tem dois esquemas em série no qual os pacotes que suportam vários objetos: *panels*, *canvases*, *scrollbars*, etc, interagem com o Notificador diretamente, registrando suas próprias chamadas de retorno dos procedimentos. Uma aplicação, em volta, registra sua próprias chamadas de retorno de procedimentos com o objeto.

Tipicamente quando se escreve uma aplicação XView, cria-se várias janelas e outros objetos que necessitam-se para sua interface e registram suas próprias chamadas de retorno de procedimentos com o

objeto. Então passa-se o controle para o Notificador. O trabalho é feito em várias chamadas de retorno de procedimentos.

3.4 SUMÁRIO

A ferramenta, Xview, provê um rico conjunto de ferramentas e operações, capacitando a criação e execução de aplicações dentro de um ambiente multijanelas nas *workstations* (estações de trabalho). O Xview pode ser considerado como uma potente e flexível ferramenta de interface para o usuário.

CAPÍTULO 4 - UMA FERRAMENTA DE MONITORAÇÃO AO NÍVEL DE MÁQUINA

4.1 INTRODUÇÃO

Uma das características da máquina multiprocessador é que ela é tipicamente construída com um grande número de unidades de pequenas variedades. Por ex: para uma máquina dataflow multiring, as unidades são processadores de elementos (PEs), (Program), (SW). Para cada tipo de unidade, um número de características são de interesse e necessitam ser monitoradas. No caso das PEs, dados tais como comprimento da lista de sinais (TQL), matching store ocupance e todas as unidades de funções usadas aparecem na base per-PE.

Uma integração e interpretação da potencialidade do grande número de características monitoradas poderia ser dificultada se a apresentação usada fosse

puramente textual, sendo assim gráficos são usados para representar o estado da máquina.

4.2 MATRIZ DE MEDIDAS

Um pacote gráfico, *XGraph*, foi projetado e implementado na estação de trabalho SUN para observar o procedimento da máquina multiprocessador.

Uma matriz de medidas gráficas é apropriada para mostrar as características da máquina. Cada tipo de unidade ocupa uma matriz separada, com cada linha da matriz ocupando uma unidade separada e cada coluna ocupando uma característica. Cada medida é essencialmente um modelo de atividade das características em uma unidade.

XGraph reconhece um arquivo de dados que contém descrições da máquina, junto com medidas das características para serem mostradas. O *XGraph* transforma esta informação dentro de matrizes separadas com medidas e números gráficos (chamados headers no programa). Cada matriz é implementada como uma janela separada, assim a janela de operação pode

ser aplicada para uma matriz independente das outras matrizes.

XGraph tem o conceito de *clock* global. As medidas das características são separadas por *ticks* de *clock*; em cada *tick* todas as medidas e cabeçalhos são atualizados. O *XGraph* apenas reconhece o *tick* de *clock*, mas não tem noção do intervalo de tempo entre os *ticks*. A imposição de um *clock* global é necessário, e isto capacita características de diferentes unidades sejam relacionadas e comparadas no tempo. Em sistemas com grande número de unidades, cada operação acima do seu *clock*, relatando as diferentes características será dificultada. O *clock* global provê tempo como uma base comum, tal que, características das diferentes unidades podem ser relacionadas e interpretadas. A Figura 21 mostra a visão de uma máquina paralela adaptada pelo *XGraph*. Cada unidade fornece medidas para as características que são coletadas juntas, adiante com *tick* de *clock* gerado pelo *clock* global sobre um arquivo de dados. O arquivo é então interpretado pelo *XGraph* para dar um *display* gráfico. O usuário pode interagir com o *XGraph* para influenciar o *display*.

Figura 21 - Interpretação Gráfica de uma Máquina Paralela

XGraph é usado para estudar simulação da máquina dataflow. A Figura 22 mostra um exemplo de saída da execução simulada do MR para o programa BR, com dois PEs, um Program e dois SWs.

Figura 22 - Gráfico de Saída da Simulação do XGraph

4.3 DESCRIÇÃO DA ENTRADA DOS DADOS

A especificação sintática da linguagem de entrada é mostrada na Figura 23. A especificação usa uma BNF estendida, com a notação $[x]^n$ denotando n ocorrências de x .

```

<Program>      ::= <Init> { <Update> } EN
<Init>         ::= { <Descriptor> } FS
<Descriptor>   ::= FI <MatxNo> <MaxRow> <MaxCol> <MaxHeader> <Label>
                (FT [<Label> ^ [<MaxCol>]])
                (FH [<Label> ^ [<MaxHeader>]])
<Update>       ::= TS { <NewData> }
<NewData>      ::= GC <MatxNo> <ColumnNo> [<PosReal>] ^ [<MaxRow>] |
                GR <MatxNo> <RowNo> [<PosReal>] ^ [<MaxCol>] |
                GS <MatxNo> <RowNo> <ColumnNo> [<PosReal>] |
                HB <MatxNo> [<Real>] ^ [<MaxHeader>] |
                HS <MatxNo> <HeaderNo> <Real>
<MatxNo>       ::= 0 | 1 | .. | 6 | 7
<HeaderNo>     ::= 0 | 1 | .. | 6 | 7
<ColumnNo>     ::= 0 | 1 | .. | 6 | 7
<RowNo>        ::= 0 | 1 | .. | 62 | 63
<MaxHeader>    ::= 0 | 1 | .. | 7 | 8
<MaxCol>       ::= 0 | 1 | .. | 7 | 8
<MaxRow>       ::= 0 | 1 | .. | 63 | 64
<Label>        ::= Uma string de até oito caracteres
<PosReal>      ::= Números reais positivos

```

Figura 23 - Especificação Sintática da Linguagem de Entrada do XGraph

O arquivo de entrada consiste de duas partes essenciais :

* Matriz de Descrição -> Consiste da declaração e inicialização de comandos o qual descreve cada

matriz que caracteriza o *display*. Isto corresponde a uma seção <Init>

* Processamento dos dados -> Consiste dos dados gravados. Esses são as estatísticas para uma medida, separados pelo *tick* de *clock*.

A seção <Init> inicializa todas as matrizes presentes no arquivo de dados, cada matriz é inicializada por uma cláusula <Descriptor>. *XGraph* permite o máximo de oito matrizes, e cada matriz é identificada pelo *MatxNo*, que varia de 0 a 7. A matriz é definida por uma chave FI com seu *MatxNo*, e é seguida pelo *MaxRow*, *MaxCol*, *MaxHeader*, denotando o número total de linhas, colunas e cabeçalho naquela matriz. *XGraph* impõe um limite superior de 64 linhas, 8 colunas e 8 cabeçalhos em cada matriz. O rótulo é uma string de caracteres que aparece no topo de uma janela de uma matriz para identificar a janela na tela. A expressão FI é seguida por uma expressão opcional FT, que dá o título para cada coluna das características. Ao cabeçalho também pode ser dado um título opcional pelo uso de uma expressão FH.

Cada <Update> consiste de um *tick* de *clock*, distinguido por uma chave TS, e uma sequência de dados gravados definindo os valores das medidas e cabeçalhos

que são fornecidos por aquele *tick* de *clock*. O usuário pode atualizar a coluna de medidas completa na matriz usando GC, linhas completa usando GR, ou apenas uma só medida usando GS. Similarmente, todos os cabeçalhos na matriz pode ser atualizados usando HB, ou um só cabeçalho usando HS.

Algumas medidas ou cabeçalhos que não dão um novo valor no *tick* de *clock* é assumido ter o mesmo valor do *tick* de *clock* anterior.

Os dados abaixo são relativos a uma pequena parcela do arquivo br.gra que foi usado para gerar os gráficos das figuras 22,24,25,...,38.

```

FI 0 3 2 0 PE
FT TQL %UTIL
FI 1 3 1 1 Program
FT #INSTR
FH S1
FI 2 3 2 0 SW
FT BackCnt BackMag
FS
TS
GR 0 0 1 20
GR 1 0 1
GR 2 0 2 13
GR 0 1 1 20
GR 1 1 1
GR 2 1 1 17
GR 0 2 1 20
GR 1 2 1
GR 2 2 1 20
HS 1 0 3
TS
```

4.4 VISÃO DO USUÁRIO COM RELAÇÃO A FERRAMENTA

XGRAPH

A operação do *XGraph* pode ser dividida em duas fases: a fase de seleção e a fase de processamento.

4.4.1) FASE DE SELEÇÃO

O usuário pode projetar um *display* e selecionar certas características para *dump*. Isto pode ser realizado através da interação com um Painel Inicial e vários Ítems de painéis.

Display de Seleção -> Como um arquivo de dados tipicamente contém muitas características, um *display* habitual capacita o usuário a enfocar uma parte da máquina que é de maior interesse dele. O usuário pode selecionar quais matrizes ou colunas irão aparecer no *display*. O mesmo arquivo de dados pode conseqüentemente ser usado para dar vários *display* que podem ser de interesse para diferentes usuários.

A Figura 24 mostra o painel para seleção de quais matrizes e colunas serão exibidas. O *default* é exibir todas as colunas em todas as matrizes inicializadas. No painel de botões, cada pequeno quadrado representa uma coluna de medidas, e um *tick* marcado indica que aquela coluna é para ser mostrada. Na Figura 24 o usuário indica que a coluna 0 na matriz PE não é para aparecer no *display*. O rótulo da matriz, junto com o título de cada coluna são mostrados para ajudar na seleção.

Figura 24 - Painel para seleção das matrizes e colunas a serem exibidas

History Dump -> Um mecanismo *history* capacita a mudança das características acima de um tempo total de execução para ser estudado. Este também é requerido se diferentes características são para serem comparadas. O histórico de uma medida pode ser obtido por uma facilidade dump no *XGraph*. O usuário pode selecionar qual medida para dump, e o histórico de qual medida é escrita sobre uma separada pintura de *history*. O usuário pode inspecionar a pintura através de uma janela *history*, ou dump este conteúdo sobre um arquivo raster no final do display. Se a dimensão da pintura *history* excede o da tela *scrollbars* são ligados na janela *history* para acessar a pintura total. A dimensão da pintura *history* depende do número de característica para *dump*, e o comprimento de seus históricos. O arquivo raster pode ser plotado no papel por uma impressora.

A Figura 25 mostra o painel de seleção das medidas que estão para ser dump. A mudança de modo é afetado pelo *Panel_Choice_String*. Cada quadrado agora só representa uma só medida. Nota-se que um scrollbar é ligado pelo painel de botão, de modo que para uma matriz contendo muitas linhas características a linha de interesse pode ser quebrada dentro da visão. Na

Figura 25 o usuário tem indicado os históricos do comprimento da lista de sinais (TQL), para as PEs 0 e 1 e os históricos da função de utilização da unidade (%UTIL) para as PEs 0 e 2 estão para ser dump.

Uma matriz que aparece no display permite algumas de suas medidas ser *dump*, mesmo se aquelas medidas não estão mostradas. Contudo se uma matriz não está mostrada toda nenhuma de suas medidas pode ser dump.

Initialize Panel

Finish

Save

Mode : ☒ Dump

Matrix : ☒ PE ☐ Program ☐ SW

PE : TQL %UTIL

0 : ☒ 0 ☒ 1

1 : ☒ 0 ☐ 1

2 : ☐ 0 ☒ 1

Figura 25 - Painei de seleção das medidas que estão para serem Dump

4.4.2) FASE DE PROCESSAMENTO

A fase de processamento lê o dado gravado de um arquivo de dados e atualiza as medidas e cabeçalhos. A velocidade de atualização é controlada pelo intervalo de tempo como mostra a Figura 26.

Figura 26 - Painel pronto para iniciar a fase de processamento

Um painel de controle provê a interação do usuário, o usuário tem três formas de iniciar a execução da simulação: a 1ª, é passo a passo através do botão single , ou seja, a simulação ocorre cada vez

que o usuário pressiona o botão de step (Figura 27); a 2ª é pelo intervalo de tempo através do botão timed e sendo possível selecionar entre 1 e 2000 o intervalo de tempo, podendo fazer uma simulação continua mais devagar ou mais rápida, podendo também congelar a simulação em determinado *tick* de *clock* e depois continuar (Figura 28); e a 3ª é através do botão contínuos, fazendo a simulação continuamente, podendo congelar e depois continuar igual a anterior (Figura 29). Nas três formas de iniciar a execução, o usuário, se desejar pode fazer o play_back da simulação ,ou seja, fazer o retorno a uma parte já mostrada da simulação que não está aparecendo mais na tela (Figura 30 e Figura 31).

Figura 27 - Paineil com simulação passo a passo

Figura 28 - Painei com a simulaão parada

Figura 29 - Painei fazendo a simulaão com intervalo de tempo definido pelo usurio.

Figura 30 - Pannel mostrando a opção de retorno a uma parte já mostrada pela simulação.

Figura 31 - Pannel mostrando a simulação retornando ao início.

Para economizar espaço da tela e evitar confusão do usuário, cada janela apenas acomoda 7 linhas. Para cada matriz, linhas de 0 a 7 são implementadas como um *Base Frame* e mais linhas são colocadas dentro de *Subframes*. *Subframes* podem ser quebrados pela seleção de um menu de *Base Frame*.

Todas as medidas na mesma coluna e na mesma matriz tem o mesmo limite, capacitando valores relativos de medidas nas diferentes linhas para serem facilmente comparados. O limite é ajustado dinamicamente e é mostrado no canto inferior esquerdo embaixo da medida.

4.5 UM EXEMPLO DE USO

A Figura 32 mostra a execução do programa BR, com 2 PEs, 2 SWs e 1 Program, o intervalo de tempo de cada *tick* de *clock* é 2000 passos de tempo e na execução da figura foi executado 30 passos de *clock*.

No fim do *display*, se pelo menos uma medida foi *dump*, os botões de ShowHistory e DumpHistory são mostrados, como mostra a Figura 32. Selecionando o botão ShowHistory causa uma janela *history*, assim que um gráfico *history* possa ser inspecionado, como na

Figura 33 e Figura 34. O usuário pode transformar as janelas em ícones, como mostra a Figura 35. Selecionando o botão DumpHistory, por outro lado, causa um gráfico *history* para ser *dump* para um arquivo de formato raster, que pode ser impresso na impressora, como na Figura 36 e Figura 37. Ao término do programa o usuário deverá selecionar o botão Quit, causando assim um janela de confirmação, como mostra a Figura 38.

Um gráfico *history* não apenas capacita um histórico inteiro de uma característica para ser estudada, mas também permite que diferentes características possam ser comparadas e relatadas.

Figura 32 - Painel mostrando o final da simulação.

Figura 33 - Janela History Graph causada pela seleção do botão Show History.

Figura 34 - Painei mostrando o History Graph

Figura 35 - Janela History transformada em ícone.

Figura 36 - Janela File causada pela seleção do botão Dump History.

Figura 37 - Arquivo de formato Raster gerado pela opção Dump History.

UNSCUJEP ARQUIVO DE BIBLIOTECAS

Figura 38 Janela de Saída do programa.

4.6 MEDIDAS E GRÁFICO HISTORY

Uma medida pode ser considerada como uma janela de visão dentro de um *history* total das características, como mostra a Figura 39. Com a facilidade do play_back as medidas podem escorregar para frente e para traz ao longo do histórico.

Figura 39 - Reescalamento de uma medida.

Reescalamento de uma medida pode ser necessária freqüentemente completa. Por exemplo na posição a , com um limite menor consegue-se mais precisão, assim também mais detalhes podem ser vistos. Como os valores medidos aumentam através das posições b e c, o limite é aumentado para acomodar os novos pontos dados. Na posição d, contudo a extensão foi reduzida novamente.

Assim todas as medidas nas colunas tem a mesma extensão, reescalamento é aplicado para uma coluna total. Um limite corrente, *range*, é associado com cada coluna de medidas. Se um valor dado na medida cai fora do limite, *range* é aumentado e a coluna total de medida é reescalada.

Um valor *ucnt* é usado para contar o número de dados consecutivos na coluna total de medida que estão

abaixo de $1/3$ do limite corrente. Se todos os valores dados nas medidas estão abaixo daquele valor, *range* é dividido em partes iguais de tal maneira que mais detalhes são vistos.

A medida é pintada pelo rolamento dela para à esquerda, e o novo ponto dado é adicionado para à direita. Este é revertido durante o play_back: a medida é rolada para à direita, e o ponto velho adicionado para à esquerda. A medida total é repintada apenas quando *range* muda. As colunas de medidas são então repintadas com todos os pontos dados medidos para um novo *range*

Para capacitar o play_back, valores de medidas e cabeçalhos são armazenados no buffer. Play_back envolve leitura de valores prévios do buffer. Em *XGraph*, o tamanho do buffer de 100 é usado.

Para um gráfico *history*, reescalonamento significa parar um gráfico que não é desejável. Um gráfico *history* é desenhado quando um buffer para um valor medido está cheio. Um valor, *maxrg*, é usado para gravar o limite máximo de coluna de características acima de um período de 100 ticks de clock. Isto é uma

escala para um gráfico history acima de 100 valores dados, de modo que nenhum reescalonamento é necessário. Para o próximo conjunto de valores dados, reescalonamento pode ser necessário apenas se o novo *maxrg* é mais extenso que o prévio.

4.7 ESTRUTURA DE DADOS

A principal estrutura de dados é um array de *frame_info*, onde *frame_info* é uma estrutura contendo informações para cada matriz, incluindo o número de linhas, colunas, cabeçalho, coordenadas das medidas da janela, ponteiros para medidas e cabeçalho, *range*, *maxrg* e *ucnt* para cada coluna na matriz.

Uma medida ou cabeçalho é representado por um array, que atua como um buffer circular armazenando os valores das medidas ou cabeçalho.

Cada gráfico na janela *history* é representado por uma estrutura *hist*, que armazena o limite do gráfico e sua posição na janela *history*.

Um ponteiro, *svptr* aponta para um indexador posição corrente no array contendo os valores medidos.

Um outro ponteiro *bkptr*, aponta para indexador posição no array durante o play_back. O valor cabeçalho mostrado correntemente é indexado pelo ponteiro *bkptr* dentro de um array contendo os valores do cabeçalho.

4.8 FUNÇÕES DAS ESTRUTURAS DO PROGRAMA

A estrutura de chamada Xview tem facilidade no gerenciamento de eventos e faz a estrutura do programa principal muito simples. O programa principal pega o nome de um arquivo de dados, e chama *init_panel*, que cria um painel usado para especificar informações dump e display. O procedimento *Inicialize* é então chamado para ler a seção de inicialização no arquivo de dados, e o procedimento *main_proc* é chamado. *Main_proc* contém uma chamada Xview para *xv_main_loop*, e o controle é passado para o *notifier* para iniciar o processamento do evento.

Existem muitos tipos de eventos que são de interesse. Isto inclui interação através do mouse, painéis e sistema de janela. Tempo expirado também dispara eventos no intervalo de tempo. A Tabela IX sumariza os eventos recebidos na fase específica. A 1ª coluna da tabela apresenta o objeto que causa o

evento, a 2ª coluna apresenta os procedimentos que são associados com cada evento, e a 3ª coluna rapidamente descreve suas ações.

Tabela IX - Objetos, Procedimentos e ações dos eventos na fase de seleção.

Objeto que causa evento	Procedimento associado ao evento	Descreve as ações dos eventos
Frame_choice	Choice_change	Muda a corrente matriz
Mode_choice	Choice_change	Muda modo corrente Display/Dump
Save_button	Save_frame	Salva as informações da corrente matriz
Finish_button	Start_Draw	Vai fase processamento
Quit_window	Destroy_func	Termina Programa

O fechamento da janela de eventos é causado pelo sistema de janela através da seleção da opção QUIT no menu frame do painel, que termina o programa.

Start_draw destrói o painel de inicialização e inicia a fase de processamento, o controle do painel e a matriz de janelas são repartidos e diferentes eventos são agora de interesse, e são sumarizados na

Tabela X. O topo da seção da tabela indica itens de painel.

Tabela X - Objetos, Procedimentos e ações dos eventos na fase de processamento.

Objeto que causa evento	Procedimento associado ao evento	Descreve as ações dos eventos
t_button	quit_proc	Termina programa
Freeze_button	freeze_pict	congela e reinicia display
Time_slide	change_intv	muda intervalo de tempo
Rewind_cycle	Set_flag	Estabelece flags para rewind liga/desliga
ShowHist_button	Show_hist	Mostra janela com histórico
DumpHist_button	Dump_file	Mostra janela para gravar histórico no arquivo
Resize_window	Frame_resize	redesenha medida para caber no novo tamanho da janela
Quit_window	Destroy_func	Destrói janela
Interval_time	Timer_tick	Lê dados do arquivo para um tick de clock
Right_mouse_button	Click_hander	Mostra menu para trazer subframe
Single_button	Set_step	Inicia execução passo a passo
Continuos_button	Set_step	Inicia execução continuamente
Timed_button	Set_step	Inicia execução com intervalo de tempo definido

Os eventos *quit* e *resize* são gerados pelo sistema de janela pela seleção através do menu *frame* de janela. Terminando, uma matriz de janela apenas

destrói a matriz; mas terminando o painel de controle, termina o programa. Redefinição do painel de controle não tem efeito e não causa um evento, mas redefinição de um base frame causa uma medida para ser redesenhada para caber no novo formato da janela.

Valores dados para uma medida são lidos do arquivo de dados ou do buffer no mesmo *tick* de *clock* disparado pelo intervalo de tempo.

4.9 FAZENDO UMA OBSERVAÇÃO

O *XGraph* foi achado ser muito proveitoso na MFDM. A facilidade *history dump* é particularmente proveitosa, pois permite o acompanhamento da evolução das características acima de um tempo para ser estudada, e características diferentes para serem comparadas e relatadas no tempo.

Embora não seja geralmente possível extrair o exato valor das características, *XGraph* dá seus valores de pico que são suficientes em muitos casos, sendo este geralmente o estado relativo das características que são de maior interesse.

A especificação de entrada do *XGraph* permite um total de 8 matrizes, cada uma podendo ocupar 8 janelas.

As medidas das características tem que estar em um formato específico. Isto não é considerado um problema, pois a maioria dos simuladores, ou seus computadores hospedeiros de um sistema paralelo que executa as medidas características podem gerar o formato íntegro requerido facilmente. Alguma flexibilidade é provido na especificação de medidas. Em muitos sistemas, as várias unidades tem seu próprio *clock* e operam assincronamente. Estes sistemas não tem noção do tempo global, e o relato das características das diferentes unidades pode conseqüentemente ser dificultada.

4.10 SUMÁRIO

Uma máquina paralela tipicamente consiste de muitas unidades de variedades específicas, e máquinas ao nível de monitoração envolve as características e procedimentos dessas unidades. Este capítulo descrevem uma ferramenta de monitoração que apresenta estatísticas ao nível da máquina como um *display* gráfico. Embora a ferramenta tenham apenas sido usado

na MFDM, o método foi proposto para ser geral e deveria ter um extenso limite de aplicabilidade.

CAPÍTULO 5 - DIFERENÇAS ENTRE AS FERRAMENTAS GRAPH E XGRAPH

5.1 INTRODUÇÃO

Este capítulo descreve as diferenças entre as ferramentas GRAPH e XGRAPH, para isso, discute: as diferenças entre as ferramentas SunView e Xview, em relação à arquitetura dos sistemas SunWindows e XWindows e as novas opções que o programa XGRAPH possibilita para o usuário.

5.2 ARQUITETURA DOS DOIS SISTEMAS

Embora tenham filosofias e origens afins, as arquiteturas do sistema Xview e SunView diferem em diversos aspectos. O Xview implementa um sistema de janelas baseado no conceito do servidor X11, enquanto SunView é um ferramenta evolucionária baseada no Sistema Operacional SunWindows.

A diferença básica entre esses dois sistemas é que sistemas baseados em *Kernel* são geralmente dependentes de hardware e Sistema Operacional específicos, enquanto que sistemas baseados na filosofia Cliente/Servidor usam um protocolo independente do hardware e, até certo ponto do sistema operacional. Sistemas de janelas baseados em servidor podem executar em uma máquina qualquer da rede, independente da arquitetura da máquina, sistema operacional, resolução de tela e capacidade de cores e outras características.

5.3 OBJETOS

A seguir será comparado as principais diferenças do ponto de vista do usuário. Não entrando no mérito do código implicado. As principais diferenças estão nos objetos descritos abaixo:

5.3.1 CANVAS

Há uma maior clareza de implementação do *Xview* em relação ao *SunView* tanto conceitualmente como na programação, pois o *Xview* permite ao objeto *canvas* o desenho de aplicações em cima de uma área maior que a

janela na qual o desenho é exibido, utilizando duas janelas separadas a *paint window* e a *view window*, sendo o canvas a união das duas como mostra a Figura 40, enquanto que em SunView desenho de superfície é implementado como uma região Pixwin.

Figura 40 - Canvas

5.3.2 PANELS E MENUS

Em Xview o controle de *panels* e *menus* são visualmente idênticos, muitos dos objetos disponíveis como ítems de panel, estão também disponíveis como ítems de menu, enquanto que em SunView, panels e menus oferecem pouca sobreposição funcional ou consistência visual.

Em Xview, panels são implementados como janelas, enquanto menus e ítems individuais de panels e menus não são, enquanto que em SunView panels e menus são considerados como janelas, sendo que cada janela é representada por um arquivo descritor do UNIX. Eles devem ser usados com parcimônia, pois existe um limite definido pelo Sistema Operacional quanto ao número de arquivos descritores que podem ser criados por processo.

5.3.3 TEXT

Uma área de edição de texto é outra componente importante em muitas aplicações. Uma subjanela *Text* do Xview retém a maioria das características de uma subjanela *Text* do Sunview, exceto que a subjanela

Xview usa o modelo de seleção OPEN LOOK U.I. Em Xview, o conteúdo de uma janela Text é armazenado em um processo Cliente, o servidor apenas mostra o texto, mas não armazena ou manipula o texto.

5.3.4 SCROLLBAR

As barras de rolagem (Scrollbar) no Xview são diferentes em relação ao SunView porque: em Xview barras de rolagem são janelas que implementam metaforicamente scrollbar OPEN LOOK U.I de um cabo em cima do elevador, enquanto que em SunView barras de rolagem são implementadas no Pixwin, em vez de na janela, por causa do número limitado de janelas para uma aplicação.

5.3.5 IMAGENS

Em SunView, a interface PixWin é responsável pelas imagens. Pixwin é uma camada lógica acima da biblioteca Pixrect. Esta implementa noções básicas de *clipping* de janela e coexistência com outras janelas.

Em Xview, a imagem é carregada fora do servidor de janela X11.

Em SunView, Pixrect provê Pixwin com uma biblioteca de hardware abstrato contendo as operações gráficas necessárias para mostrar as imagens. Pixrect suporta conceito de coordenadas de tela, pixels, bitmaps, operações raster, vetores e desenhos de texto Sun⁴⁰.

Em Xview, Xlib provê uma biblioteca de hardware abstrato para operações gráficas. Por causa da atual implementação de rotinas gráficas estar no servidor, os gráficos podem ser implementados por hardware ou Sistema Operacional. A biblioteca X11 provê o mesmo conceito de coordenadas de tela, pixels, bitmap, e classifica operações de desenho como Pixwin. Para prover uma transição fácil para programas SunView, a interface Pixwin do SunView foi retida, mas a implementação foi descartada e repassada pelas operações gráficas X11.

5.4 MELHORAMENTOS DO XGRAPH EM RELAÇÃO AO GRAPH

Na fase de processamento foram criadas três possibilidades de execução da simulação: passo a

passo, contínua com intervalo de tempo definido pelo usuário através da seleção do intervalo de tempo, podendo haver o congelamento da simulação e depois continuar, e contínua sem intervalo de tempo definido pelo usuário. Este último modo executa tão rápido quanto este pode e é tipicamente usado quando lê entrada de um *pipe*, também podendo haver o congelamento. O botão de congelamento passou a ter três opções *start*, *continue* e *congela*, sendo que no Xgraph a simulação só tinha uma opção a contínua com intervalo de tempo definido pelo usuário e o botão de congelamento só tinha duas opções: *continue* e *congela*, sendo que a simulação já iniciava quando era passada para a fase de processamento.

Durante a execução da simulação foi criada a possibilidade para o usuário poder retornar uma parte já mostrada na simulação que não está mais aparecendo na tela, através da opção *Forward/Backward*, com isso o usuário pode ir e voltar a simulação quantas vezes quiser.

Mudança na forma como o *history* gráfico era produzido, ao invés de desenhar durante o processo de simulação através da entrada do arquivo todos os

valores são salvos e o gráfico é desenhado em um passo no final.

Foi removida a limitação de oito caracteres no comprimento das medidas, mudança no painel de inicialização para usar o nome das matrizes ao invés de seus números, no history graph foi colocado rótulo na abscissa horizontal com o número de tick.

O programa Xgraph tem 3432 linhas de código fonte escrito na linguagem C e foi subdividido em 48 módulos, sendo esses chamados como funções de biblioteca no programa principal, enquanto que o programa Graph é composto por um único programa.

5.5 CONVERSOR AUTOMÁTICO

Existe um conversor automático (programa), chamado `convert_to_xview` **Sun**⁴¹, que recebe como entrada de dados um arquivo contendo um programa em SunView e gera como saída de dados um arquivo contendo o programa convertido para Xview.

A funcionalidade desse programa para aplicações simples é boa, pois com pequenas modificações no arquivo de saída feitas pelo usuário a conversão total

se concretiza, mas para aplicações envolvendo gráficos, a conversão gerada pelo programa é mínima. Para chegar à conversão completa o usuário deverá conhecer e utilizar profundamente os Manuais: Converting SunView Application **Sun**⁴¹; Xview Programming Manual **Sun**⁴⁰, Xlib Manual Vol.1 **Nye**⁴² e Xlib Manual Vol.2 **Nye**⁴³.

5.6 VANTAGENS E DESVANTAGENS

As vantagens do Xgraph em relação ao Graph são:

- 1) Possibilidade de executar o programa em uma máquina da rede e mostrar a saída em outra qualquer da rede, independente da arquitetura e sistema operacional, sendo que a ferramenta Graph só executa em hardware Sun.

- 2) Facilidade de poder usar mais de 8 janelas, pois este é o número máximo de arquivos descritores permitido pelo UNIX, sendo que no SunView cada janela corresponde a um arquivo descritor.

3) Em relação ao suporte do programa a ferramenta XGraph está toda modulada facilitando as alterações necessárias.

4) Melhora na qualidade da imagem, pois o ambiente da ferramenta é mais moderno e de melhor resolução.

5) Maior interação do usuário com o programa, pois este tem a possibilidade de fazer a simulação de três formas já descritas e existe a opção de *playblack* que permite o usuário voltar a simulação a partes que não estão mais aparecendo na tela.

As desvantagens da ferramenta Xgraph em relação a ferramenta Graph:

1) Código executável dobrou de tamanho.

2) O sistema ficou um pouco mais lento.

Capítulo 6 - CONCLUSÕES E FUTUROS TRABALHOS

Na construção de computadores paralelos, monitoração é essencial para tornar o projeto viável. O uso de diferentes níveis de abstração conduz a uma variedade de monitoração vista e conseqüentemente diferentes interpretações e ferramentas são necessárias.

Com a conversão e as modificações da ferramenta Graph, para a ferramenta Xgraph a interação com o usuário foi facilitada e melhorada, tanto ao nível de opções, quanto ao nível de poder trabalhar num ambiente mais moderno baseado no conceito Cliente/Servidor, usando protocolo independente do hardware e do Sistema Operacional.

A ferramenta Xgraph mostra o estado da máquina paralela, mas ela é apenas uma ferramenta de apresentação de dados, ou seja, o usuário não pode influenciar como e quais dados serão coletados através

da ferramenta Xgraph, mas pode influenciar utilizando o simulador MR. Embora Xgraph foi apenas usado na MFDM, os métodos são gerais e deve ter uma extensa aplicabilidade. Como trabalhos futuros o programa Xgraph poderia ser modificado para monitorar o simulador SIM (também da MFDM) e o simulador da arquitetura WOLF (SAW).

Sistemas paralelos de propósito geral não são ainda uma realidade. Contudo, eles podem vir a ser um computador num futuro próximo, e ferramentas de monitoração sofisticadas são essenciais para o entendimento dos seus procedimentos.

BIBLIOGRAFIA

1. **TANG, H. W. H.** *Monitoring Tools for Parallel Systems.* Technical Report Series, University of Manchester, Dec 1987.
2. **FERRARI, D.; SERAZZI, G.; ZEIGNER, A.** *Measurement and Tuning of Computer Systems.* Prentice-Hall, Englewood Cliffs, N. J., 1983.
3. **WEICKER, R.** *An Overview of Common Benchmarks.* IEEE Computer, 65-75, December 1990.
4. **FERNANDES, M.** *Modelagem Analítica de Desempenho de Sistemas Multiprocessadores: Aplicação ao Multiprocessador CPER.* Dissertação de Mestrado, Universidade Federal de São Carlos, dezembro de 1992.

5. **SANTANA, M.** *An Advanced Filestore Architecture for a Multiple - Lan Distributed Computing System.* PHD Thesis, University of Southampton, January 1990.

6. **ORLANDI, Regina C. G. de Souza.** *Ferramentas para Análise de Desempenho de Sistemas Computacionais Distribuídos.* Dissertação de Mestrado, São Carlos, ICMSC-USP, 1995.

7. **SANTANA, R.** *Performance Evaluation of Lan-Based File Servers.* PHD Thesis, University of Southampton, January 1990.

8. **SOARES, L.** *Modelagem e Simulação Discreta de Sistemas.* Editora Campus 1992.

9. **IBM.** *OS/VS2 System Management Facility.* IBM Corporation, July 1977.

10. **SUN MANUAL.** *SunOS Reference Manual - Maintenance Commands.* Sun Microsystem, Vol. 3, March 1990.

11. **TRINDADE, JR, O.; Zaluska, E.** *Real Time Monitoring of Concurrent Computation.* Anais do SEMISH 94, Caxambu, MG, Agosto de 1994.

12. **ROSE, C. A.** *A Measurement Procedure for Queueing Network Models of Computer System.* Computer Survey, Vol 3, No. 3, pp 263-280, September 1978.

13. **NUTT, G. J.** *Tutorial: Computer System Monitors.* IEEE Computer, Vol 8, No. 11, pp 51-61, November 1975.

14. **SVODOBOVA, L.** *Online System Performance Measurement with Software and Hybrid Monitors.* Proceedings of the 4th ACM/SIGOPS-Symposium on Operating System Principles, pp 45-53, October 1973.

15. **DAHL, O. J.; NYGAARD, K.** *SIMULA: A Language for Programming and Description of Discrete Event Systems.* 5th Edition, Norwegian Computing Centre, Oslo, 1967.

16. **MARKOWITZ, H.; KIVIAT, P. J.; VILLANEVA, R.** *The SIMSCRIPT II Programming Language.* Prentice Hall, Englewood Cliffs, N. J. 1969.

17. **SCHRIEBER, T. J.** *Simulation using GPSS.* John Wiley & Sons Inc. , 1974.

18. **KIVIAT, P. J.; KOLENCE, K. W.** *Software Unit Profiles and Kiviat Figures.* SIGMETRICS Performance Evaluation Review, vol. 2 no. 3 pp 2-12, September 1973.

19. **MORRIS, M.** *Kiviat Graphs - Conventions and Figures of Merit* SIGMETRICS Performance Evaluation Review, Vol 3, No. 12, pp 2-8, October 1974.

20. JOYCE, J.; LOMOW, G.; SLIND, K.; UNGER, B. *Monitoring Distributed Systems*. ACM transaction on Computer Systems, Vol. 5, no. 2, pp. 121-150, May 1987.

21. COHEN S.; DAVIS, A.; ROBINSON, S. *THE FAIM-1 User interface: Human Engineering for the Fifth Generation*. Woods J. V. (ed) ; *Fifth Generation Computer Architectures*, North Holland, Amsterdam, pp. 257-273, 1986.

22. DISZ, T. et al. *Experiments With Or Parallel Programs*. Fourth International Conference on Logic Programming, Camberra, 1987.

23. DONLIN, M. *System Designers Face Hard Choices When Forging Simulation Strategie*. Computer Design, 62-72, April 1993.

24. ARVIND & GOSTELOW, K. P. *The U-interpreter*. IEEE Computer. 15(2): 42-49, February 1982.

25. DENNIS, J. B.; BOUGHTON, G. A.; LEUNG, C. K. C. *Building Blocks For Data Flow Prototypes*. Proceedings of the 7th Annual Symposium on Computer Architecture, pp 1-8, May 1980.

26. CALUWAERTS, L. J.; DEBACKER, J.; PEPERSTRAETE, J. A. *A Data Flow Architecture with a Paged Memory System*. Proceedings of

the 9th Annual Symposium on Computer Architecture pp 120-127, April 1982.

27. GURD, J. R.; KIRKHAM, C. C.; BONHM, A. P. W. *The Manchester Prototype Dataflow Computer*. Communication of the ACM, vol 28, no 1, pp 34-52, January 1985.

28. ARVIND, K. P.; GOSTELOW, K. P.; PLOUFFE, P. *An Asynchronous Programming Language and Computing Machine*. Technical Report TR 114a, Department of Information and Computer Science, University of California, Irvine, December 1978.

29. ACKERMAN, W. B.; DENNIS, J. B. *Val - A Value-oriented algorithmic Language: preliminary Referency Manual*. Technical Report TR-218, Computation Structures Group, Laboratory for Computer Science, MIT, Cambridge, Mass., June 1979.

30. MCGRAW J. R. et al. *SISAL - Streams and Iteration in a Single-Assignment Language. Language Reference Manual*. Lawrence Livermore National Laboratory, Livermore CA, 1983.

31. RUGGIERO, C. A. *Trotter Mechanism for Manchester Dataflow Machine*. PHD. Thesis, Department of Computer Science, University of Manchester, July 1987.

32. RUGGIERO, C. A.; SARGENT, J. *Hardware and Software Mechanism for Control of Parallelism*. Dataflow Group Internal

Report, Department of Computer Science, University of Manchester, 1985.

33. TEO Y. M.; BONHM, A. P. W. *The Multi Ring Dataflow Machine Simulator*. Internal Report, Department of Computer Science, University of Manchester, 1987.

34. SARGENT, J. *Efficient Stored Data Structures for Dataflow Computing*. P.H.D. Thesis, University of Manchester, 1985.

35. CAVENAGHI, M. A.; GARCIA NETO, A. *Desempenho do Simulador SAW em problemas de métodos numéricos*. XVI-CILAMCE Congresso Ibero Americano sobre métodos computacionais para engenharia, novembro 1995.

36. SCHEIFLER, R. W.; GETTYS, J. *The X Window System Software-Practice and Experience*. vol 20(S2), S2/5-S2/34, 1990.

37. JOHNSON, E. F.; REICHARD, K. *Advanced Xwindow Applications Programming*. MIS - Management Information Source, Inc. Press, 1990

38. BARKAKATI, N. *X Window System Programming*. SAMS - Macmillan Computer Publishing, Carmel Indiana, 1991.

39. HELLER, D. *Xview Programming Manual - An OPEN LOOK Toolkit for X11*. O'Reilly & Associates, Inc, 1990.

40. **SUN MANUAL.** *Sunview System Programmer's Guide - Pixrect Reference.* Sun Microsystem, March, 1990.

41. **SUN MANUAL.** *Xview Version 2 Reference Manual: Converting SunView Applications.* Sun Microsystem, June 1990.

42. **NYE, A.** *Xlib Programming Manual.* O'Reilly & Associates, Inc, 1990.

43. **NYE, A.** *Xlib Reference Manual.* O'Reilly & Associates, Inc, 1990.