MULPLIX: Um Sistema Operacional tipo UNIX para Programação Paralela

Rafael Peixoto de Azevedo

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Júlio Salek Aude, Ph.D.
(Presidente)

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Carlo Emmanoel Tolla de Oliveira, Ph.D.

Prof. Adriano Joaquim de Oliveira Cruz, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1993

AZEVEDO, RAFAEL PEIXOTO DE

MULPLIX: Um Sistema Operacional tipo UNIX para Programação Paralela
[Rio de Janeiro], 1993
vi, 81 p., 29,7cm. (COPPE/UFRJ, M.Sc.,
Engenharia de Sistemas e Computação, 1993)
Tese - Universidade Federal do Rio de Janeiro
1. Sistemas Operacionais 2. Programação Paralela
3. Multiprocessadores
I. COPPE/UFRJ II. Título (série).

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

MULPLIX: Um Sistema Operacional tipo UNIX para Programação Paralela

Rafael Peixoto de Azevedo
março de 1993

Orientador: Júlio Salek Aude
Programa: Engenharia de Sistemas e Computação

O poder de processamento e a capacidade de comunicação são os recursos que fundamentalmente delimitam o potencial de desempenho de uma arquitetura de computadores. A arquitetura MULTIPLUS oferece alta capacidade de processamento – através de um número elevado de processadores RISC – e alta capacidade de comunicação – através de uma estrutura hierárquica de memória compartilhada. A efetivação do alto desempenho da arquitetura MULTIPLUS depende, entretanto, da exploração intensa de paralelismo e localidade.

A tese desta dissertação afirma a viabilidade da construção de um sistema operacional tipo UNIX para a execução com alto desempenho de aplicações científicas e de engenharia em arquiteturas multiprocessadas com estrutura hierárquica de memória, como a arquitetura MULTIPLUS. Um sistema operacional com estas características deve estender o conceito de processo UNIX, possibilitando a definição e escalação de múltiplas linhas de controle e provendo novos mecanismos para a alocação de memória e para a sincronização entre as linhas de controle.

Esta tese é sustentada pela experiência de definição e construção do sistema operacional MULPLIX como uma evolução do sistema operacional PLURIX. A definição do sistema abrangeu: (1) a derivação a partir do modelo PRAM de um modelo para programação paralela adequado à arquitetura MULTIPLUS e (2) a definição das primitivas providas pelo núcleo do sistema para dar suporte ao novo conceito de processo correspondente a este modelo. A construção do sistema incluiu o desenvolvimento de novos algoritmos e estruturas de dados mais adequados e a realização efetiva das modificações necessárias no núcleo do sistema.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

MULPLIX: A UNIX-like Operating System for Paralell Programming

Rafael Peixoto de Azevedo
March, 1993

Thesis Supervisor: Júlio Salek Aude
Department : Systems Engineering and Computing

The processing power and the communication capacity are the resources that fundamentally delimit the potential performance of a computer architecture. The MULTIPLUS architecture offers high processing power – through a high number of RISC processing elements – and high communication capacity – through a hierarchical shared memory structure. The architecture's effective high performance depends, however, on the intense exploitation of parallelism and locality.

The thesis of this dissertation affirms the viability of a UNIX-like operating system capable of running high performance scientific and engineering applications in a hierarchical shared memory multiprocessor architecture, like MULTIPLUS. An operating system with this characteristics extends the UNIX concept of a process, allowing the definition and scheduling of multiple threads of control and providing new mechanisms for memory allocation and synchronization.

This thesis is supported by the experience of defining and building the MULPLIX operating system as an evolution of the PLURIX operating system. The system's definition included: (1) the definition of a parallel programming model derived from the PRAM model and suitable to the MULTIPLUS architecture and (2) the definition of the new system primitives necessary to support the new process concept. The building of the system involved designing new, more suitable, algorithms and data structures and the effective execution of the corresponding modifications in the system's kernel.

Índice

I Introdução 1

1 Apresentação 2

1.1 Motivação e Proposta da Tese 3

1.2 Organização do Texto 4

2 O Projeto MULTIPLUS 6

2.1 A Arquitetura MULTIPLUS 6

2.2 MULPLIX: Uma Evolução do Sistema Operacional PLURIX . . . 10

2.3 Princípios Fundamentais 13

II Programação Paralela no MULPLIX 15

3 Definição de um Modelo Básico de Computação Paralela 16

3.1 Importância de um Modelo de Computação Paralela 17

3.2 Análise do Modelo PRAM 18

3.3 Definição e Análise do Modelo MULPLIX 21

4 Primitivas para Programação Paralela 25

4.1 Criação de Linhas de Controle 26

4.2 Alocação de Memória 27

4.3 Sincronização entre Linhas de Controle 28

5 Exemplo: Eliminação Gaussiana **33**

5.1 Definição do Problema e Revisão do Método de Solução 34

5.2 Análise das Possibilidades de Exploração de Paralelismo 36

5.3 Uma Solução Adequada ao MULTIPLUS 37

III A Implementação do MULPLIX **40**

6 Escalação de Linhas de Controle **41**

6.1 Análise Geral do Problema de Escalação 42

6.2 Objetivos para a Escalação no MULPLIX 43

6.3 Escalação no MULPLIX 44

6.4 Comparação com a Escalação no PLURIX 46

7 Gerência de Memória **47**

7.1 Definição do Espaço Virtual 48

7.2 Alocação de Memória Física 50

7.3 Avaliação 52

8 Implementação dos Mecanismos de Sincronização **53**

8.1 Parâmetros e Critérios para Avaliação 55

8.2 Exclusão Mútua Simples 55

8.3 Exclusão Mútua Múltipla 58

8.4 Ordem Parcial 60

IV Conclusões **65**

9 Conclusões 66

V Anexos **69**

A Manual de Referência: Primitivas para Programação Paralela 70

B Implementação da Instrução F&I no MULTIPLUS 77

Bibliografia 78

Parte I

Introdução

Capítulo 1

Apresentação

A utilização de computadores de alto desempenho, ou supercomputadores, está se tornando um fator decisivo de competitividade para um número crescente de atividades científicas, de engenharia e industriais. A importância estratégica da supercomputação em termos nacionais reflete-se não somente nos vultosos investimentos realizados na área pelos países mais industrializados, mas também nas dificuldades de natureza técnica e política impostas por estes países à aquisição de equipamentos e tecnologia de supercomputação pelo Brasil. Neste contexto a criação de tecnologia própria de supercomputação torna-se requisito necessário para a continuidade do desenvolvimento econômico brasileiro.

O projeto **MULTIPLUS** [7] representa um esforço importante do Núcleo de Computação Eletrônica da UFRJ para o desenvolvimento de tecnologia nacional no campo de supercomputação. O projeto visa a concepção e construção de uma família modular de computadores paralelos de alto desempenho para aplicações científicas e de engenharia. A sua realização demanda pesquisa e desenvolvimento nas áreas de arquitetura de computadores, sistemas operacionais, microeletrônica e algoritmos paralelos.

O sistema operacional **MULPLIX** [10] está sendo projetado para atuar no MULTIPLUS. Ele é um sistema tipo UNIX resultante de uma evolução do **PLURIX** [19] visando propiciar ao usuário um ambiente para desenvolvimento e execução de aplicações intensamente paralelas. A construção do MULPLIX a partir do PLURIX, não somente permite a concentração do esforço de desenvolvimento nos aspectos relacionados à programação paralela e à arquitetura MULTIPLUS, como também garante o acesso, por compatibilidade, a todo o conjunto de utilitários e aplicativos já desenvolvidos e/ou transportados para o PLURIX.

A construção do sistema MULPLIX está sendo realizada por etapas, de acordo com o seguinte planejamento:

1. Alterações restritas ao núcleo do PLURIX e realizadas em um computador EBC32010, de produção industrial e para o qual o PLURIX já havia sido transportado, com o objetivo de construir um ambiente, a nível de núcleo de sistema operacional, para programação paralela. O sistema no estágio correspondente ao final desta etapa é identificado como a versão 1.0 do MULPLIX.
2. Transporte do sistema para o primeiro protótipo do MULTIPLUS. O sistema ao final desta etapa é identificado como a versão 2.0 do MULPLIX.
3. Evolução contínua, a partir da construção de ferramentas para programação de aplicações paralelas e da avaliação do desempenho destas aplicações e do núcleo do sistema.

Esta dissertação descreve, analisa e discute o trabalho de definição e implementação da versão 1.0 e a definição da versão 2.0 do MULPLIX. A apresentação prossegue nas seções a seguir: a seção 1.1 explica a motivação e a proposta da tese e a seção 1.2 resume o conteúdo e a organização do texto.

1.1 Motivação e Proposta da Tese

Um grande número de computadores paralelos desenvolvidos no passado teve um escopo de utilização limitado por causa da dificuldade para programá-los. Esta dificuldade decorre das limitações ou das particularidades dos modelos de programação paralela estabelecidos para estas máquinas. Assim, normalmente, as aplicações são programadas especificamente para um computador particular, com uma configuração de hardware pré-estabelecida.

Em relação à interação com usuários, o projeto MULTIPLUS tem por objetivo prover um ambiente de programação mais flexível e confortável. Este objetivo acarretou as seguintes decisões de projeto:

- definição de um modelo genérico, porém adequado à arquitetura MULTIPLUS, de programação paralela;
- utilização como ambiente para desenvolvimento de software de um sistema operacional tipo UNIX, ao qual a maioria dos programadores sofisticados está habituado;

- utilização do mesmo sistema operacional para desenvolvimento e execução de aplicações paralelas.

A tese desta dissertação afirma a viabilidade da construção de um sistema operacional tipo UNIX para a execução, com alto desempenho, de aplicações científicas e de engenharia em arquiteturas multiprocessadas com estrutura hierárquica de memória, como a arquitetura MULTIPLUS. Um sistema operacional com estas características deve estender o conceito de processo UNIX, possibilitando a definição e escalação de múltiplas linhas de controle[1] e provendo novos mecanismos para a alocação de memória e para a sincronização entre as linhas de controle.

Esta tese é sustentada pela experiência de definição e construção do sistema operacional MULPLIX como uma evolução do sistema operacional PLURIX. A definição do sistema abrangeu: (1) a derivação a partir do modelo teórico para programação paralela PRAM de um outro modelo adequado à arquitetura MULTIPLUS e (2) a definição das primitivas providas pelo núcleo do sistema para dar suporte ao novo conceito de processo correspondente a este modelo. A construção do sistema incluiu o desenvolvimento de novos algoritmos e estruturas de dados e a realização efetiva das modificações necessárias no núcleo do sistema para a preparação da versão 1.0.

1.2 Organização do Texto

O texto está dividido em cinco partes. A primeira parte introduz a dissertação. Este capítulo apresenta a motivação e a proposta da tese e resume a organização do texto. O capítulo 2 estabelece os pontos de partida para este trabalho, apresentando a arquitetura MULTIPLUS, avaliando as necessidades de alteração do sistema operacional PLURIX para atender aos objetivos do projeto MULTIPLUS e revelando os princípios fundamentais que orientaram este trabalho.

A parte II apresenta o MULPLIX como um instrumento para a programação de aplicações paralelas para o MULTIPLUS. O capítulo 3 define um modelo de computação paralela adequado à arquitetura MULTIPLUS. O capítulo 4 explica a implementação deste modelo, a partir da redefinição do conceito de processo e da definição de novas primitivas oferecidas pelo MULPLIX para suportar o modelo. O capítulo 5 exemplifica a programação paralela no MULPLIX com a apresentação de uma solução para uma

[1]Nos sistemas operacionais tipo UNIX, como o PLURIX, o processo define uma única unidade de execução, ou linha de controle.

importante aplicação de computadores de alto desempenho: a resolução paralela de equações lineares através do Método de Eliminação Gaussiana.

A parte III aborda a construção interna do MULPLIX, enfocando a implementação a nível do núcleo do sistema dos principais aspectos em que o MULPLIX difere do PLURIX. O capítulo 6 explica a escalação de linhas de controle no MULPLIX. O capítulo 7 descreve a gerência de memória adotada pelo MULPLIX. O capítulo 8 contém a definição para a implementação dos mecanismos de sincronização adotados pela versão 2.0 do MULPLIX.

A parte IV contém as conclusões desta dissertação. O capítulo 9 avalia o trabalho realizado e apresenta sugestões para a continuação do trabalho.

Os anexos a seguir complementam a dissertação. O anexo A contém as páginas do manual de referência do MULPLIX relativas às primitivas para programação paralela. O anexo B descreve a implementação no MULTIPLUS da instrução F&I (*"Fetch and Increment"*) utilizada na implementação dos semáforos para sincronização.

Capítulo 2

O Projeto MULTIPLUS

Este capítulo enfoca os principais aspectos técnicos do projeto MULTIPLUS que definem o ponto de partida para o trabalho de tese. A seção 2.1 descreve a arquitetura MULTIPLUS, cobrindo os Nós de Processamento e a estrutura hierárquica de memória. A seção 2.2 avalia o sistema operacional PLURIX, segundo o ponto de vista da exploração de paralelismo e da adequação à arquitetura MULTIPLUS, com o objetivo de identificar os aspectos do sistema PLURIX que devem ser alterados para transformá-lo no sistema MULPLIX. A seção 2.3 estabelece os princípios fundamentais para a orientação deste trabalho.

2.1 A Arquitetura MULTIPLUS

A arquitetura MULTIPLUS é uma arquitetura multiprocessada modular, capaz de suportar até 2048 *Nós de Processamento* (**NP**), baseada em uma estrutura hierárquica de memória, constituindo um espaço de 32 Gbytes de memória global compartilhada. A arquitetura MULTIPLUS é capaz de originar configurações abrangendo um amplo espectro em termos de capacidade de desempenho, indo desde estações de trabalho de alto desempenho contando com alguns NP até computadores com desempenho equivalente aos chamados supercomputadores atuais.

As subseções a seguir detalham a arquitetura MULTIPLUS, descrevendo os seus componentes de maior interesse para este trabalho. Assim elas apresentam a composição de cada NP e a estrutura hierárquica de memória utilizada na comunicação entre os NP, incluindo o esquema adotado para a manutenção de coerência entre as unidades de memória cache. A última subseção avalia, suscintamente e em termos globais, o potencial de desempenho da arquitetura MULTIPLUS.

2.1.1 Nós de Processamento

Cada NP consiste dos seguintes elementos:

- Um microprocessador RISC baseado na definição da arquitetura SPARC [41], com capacidade de processamento seqüencial equivalente a 25 MIPS VAX a 40 MHz.
- Um Módulo de Memória compartilhada contendo até 32 Mbytes, que representam uma região dentro do espaço global de 32 Gbytes.
- Unidades de Memória Cache para instruções e para dados, contendo em cada caso 64 Kbytes.
- Unidades de Gerência de Memória Paginada para instruções e dados, incluindo um cache para mapeamentos de endereçamento virtual/físico (*"Translation Look Aside Buffer"*).
- Co-processador de Ponto Flutuante, com capacidade de processamento equivalente a 6 MFLOPS a 40MHz.

2.1.2 Comunicação entre Nós de Processamento

Os NP são agrupados em dois níveis: no nível mais baixo, grupos com até 8 NP interligados por um **sistema de barramentos** formam **clusters**; no nível mais alto, os clusters são interligados por uma *Rede de Interconexão* (**RI**) multiestágio do tipo n-cubo invertido.

A arquitetura MULTIPLUS define quatro formas de acesso à memória, de acordo com a localização do NP e da palavra de memória envolvidos:

Acesso à Cache — Acesso direto à Memória Cache local (pertencente ao NP realizando o acesso). Esta é a forma mais eficiente de acesso, pois não utiliza nenhum recurso compartilhado com outros NP.

Acesso Local — Acesso ao Módulo de Memória local. Esta forma de acesso, assim como a anterior, não requer a utilização do sistema de barramentos.

Acesso Intracluster — Acesso a um Módulo de Memória não local pertencente a um NP no mesmo cluster. Esta forma de acesso utiliza o sistema de barramentos interligando os dois NP envolvidos.

Acesso Intercluster — Acesso a um Módulo de Memória pertencente a um NP em outro cluster. Esta forma exige a utilização do sistema de barramentos do cluster origem, da RI e do sistema de barramentos do cluster destino.

Os sistemas de barramentos são duplos, ou seja, eles incluem um barramento para tráfego de instruções e o outro para dados, cada um com largura de 64 bits.

A RI multiestágio [14] é implementada com módulos de chaves *"cross-bar"* 2×2 com FIFOs nas saídas e se liga aos barramentos através de circuitos de interface inteligentes. Esta estrutura de rede apresenta duas propriedades muito interessantes para a arquitetura MULTIPLUS:

- **Modularidade.** Esta propriedade permite o crescimento da RI através da simples adição de novos módulos de chaves. A modularidade da RI é importante em termos de praticidade para a construção física do sistema.
- **Partibilidade.** Esta propriedade permite a formação de subredes independentes compostas de um número de clusters igual a uma potência de 2, de modo tal que o tráfego gerado por uma subrede independente não interfere com o tráfego gerado por outra subrede independente.

  A condição para se obter tal particionamento é que o conjunto de bits que define a faixa de endereços de memória associada a cada subrede independente contenha um subconjunto com configuração constante para todos os clusters de cada subrede independente.

  A partibilidade é importante em termos do desempenho global da arquitetura MULTIPLUS porque amplia a capacidade de comunicação da RI através da **exploração de paralelismo** nas transferências e porque possibilita baixa interferência na comunicação entre tarefas independentes ou fracamente acopladas.

2.1.3 Esquema de Coerência para Memória Cache

A multiplicidade de Unidades de Memória Cache em conjunto com a complexidade da organização de memória da arquitetura MULTIPLUS, caracterizariam um problema de Manutenção de Coerência de Cache extremamente difícil se todo o espaço de memória fosse *cacheável*[1]. A arquitetura MULTIPLUS define um esquema híbrido de coerência de cache [33], baseado nas seguintes decisões de projeto:

Especialização dos Barramentos – O sistema de barramentos interligando cada cluster possui dois barramentos: um dedicado à leitura de instruções e o outro

[1] O termo *"cacheável"* é utilizado neste trabalho para significar *"passível de ser armazenado em memória cache"*.

dedicado à leitura e escrita de dados. As Unidades de Memória Cache de Instruções e de Dados estão ligadas respectivamente a estes barramentos.

A especialização dos barramentos garante a coerência das informações nas Unidades de Memória Cache de Instruções, reduzindo o escopo do problema apenas às Unidades de Memória Cache de Dados.

Limitação do Espaço de Memória Cacheável — Uma Unidade de Memória Cache de Dados normalmente armazena apenas dados oriundos de posições de memória correspondentes ao cluster de seu NP.

O software, entretanto, tem a possibilidade de especificar para cada Unidade de Cache de Dados outras regiões cacheáveis no espaço de memória. Neste caso, é responsabilidade do software garantir que estas regiões contêm unicamente dados apenas para leitura.

As decisões de projeto descritas acima resultam na adoção de um esquema que pode ser implementado em hardware de modo simples. As Unidades de Memória Cache de Instruções não necessitam de nenhum mecanismo para manutenção da coerência. As Unidades de Memória Cache de Dados podem utilizar um método para manutenção de coerência por hardware baseado na monitoração do barramento de dados ao qual estão ligadas, com as opções dos métodos *"write through"* ou *"write back"* [39] para a atualização dos Módulos de Memória.

2.1.4 Entrada e Saída

A arquitetura de Entrada e Saída (E/S) do MULTIPLUS é distribuída [34]. A cada cluster estão associados dois processadores de E/S. Um deles é orientado a caracteres (PESC), processando operações de E/S com terminais e impressoras e interfaceando o MULTIPLUS com redes "ethernet". O outro processador de E/S é orientado a blocos (PESB) e processa operações de E/S com unidades de disco e fita. Os PESB situados em clusters distintos se interligam através de uma rede de alta velocidade.

2.1.5 Resumo

O poder de processamento e a capacidade de comunicação são os recursos que fundamentalmente delimitam o potencial de desempenho de uma arquitetura de computadores [8]. A arquitetura MULTIPLUS oferece alta capacidade de processamento – através de um número elevado de processadores RISC – e alta capacidade de

comunicação – através de uma estrutura hierárquica de memória compartilhada. A efetivação do alto desempenho da arquitetura MULTIPLUS depende, entretanto, da exploração intensa de paralelismo tanto em termos de processamento como em termos de comunicação.

2.2 MULPLIX: Uma Evolução do Sistema Operacional PLURIX

O PLURIX é um sistema operacional tipo UNIX, inteiramente projetado e construído no NCE/UFRJ a partir de 1982, em conjunto com o multimicroprocessador PEGASUS [19]. A capacidade do PLURIX de controlar simetricamente computadores multiprocessados exige que o núcleo do PLURIX seja um programa paralelo e que a sua construção seja baseada em técnicas mais sofisticadas, normalmente não empregadas no sistema operacional UNIX em suas versões iniciais monoprocessadas.

A disponibilidade do código fonte do núcleo do PLURIX, a sua compatibilidade com o sistema UNIX e a sua capacidade de multiprocessamento foram os principais argumentos para que ele fosse o ponto de partida para a construção do MULPLIX. As subseções a seguir avaliam o PLURIX quanto à capacidade de exploração de paralelismo por suas aplicações e quanto à sua adequação à arquitetura MULTIPLUS.

2.2.1 Exploração de Paralelismo

A capacidade de multiprocessamento do PLURIX é quase invisível para os usuários, ou seja, o PLURIX oferece ao programador um ambiente multiprogramado essencialmente igual ao UNIX. Entretanto, é possível acelerar até certo ponto determinadas aplicações, desde que elas possam ser decompostas em programas seqüenciais de tamanho razoável e de interação bastante simples, que possam ser executados como processos UNIX.

O utilitário make do PLURIX [9] representa um dos seus melhores exemplos de exploração de paralelismo. Capaz de identificar os programas que podem ser executados concorrentemente, o make do PLURIX pode gerenciar a execução paralela destes programas em computadores multiprocessados. No caso da utilização típica do PLURIX em ambiente universitário - desenvolvimento de software - o exemplo mais claro de uso desta facilidade refere-se à aceleração da compilação de programas com

fontes organizados em vários módulos. Em um nível mais alto, esta tarefa envolve a compilação (concorrente) dos módulos, cujos objetos são a seguir ligados. Em um nível mais baixo, a compilação de cada módulo pode, por sua vez, ser composta por três fases (concorrentes) interligadas por um *"pipe"*: pré-processamento, compilação propriamente dita e montagem.

O exemplo dado no parágrafo anterior sugere uma possibilidade de grande aumento da velocidade de processamento a partir da utilização da capacidade de multiprocessamento do PLURIX. Uma análise mais aprofundada, entretanto, revela que no caso mais otimista (compilação de muitos módulos e disponibilidade de muitos processadores) o tempo total de compilação tem um limite inferior dado pelo tempo de ligação dos módulos objetos, porque este procedimento é realizado seqüencialmente por um processo UNIX.

Muitas aplicações não podem ser grandemente aceleradas a partir da utilização da capacidade de multiprocessamento do PLURIX. Este é o caso, por exemplo, de uma aplicação típica em ambientes de projeto de engenharia: a resolução de sistemas de equações lineares. O aumento do desempenho desta aplicação através da sua programação na forma de um conjunto numeroso de processos paralelos exige uma capacidade de compartilhamento de dados e sincronização que o modelo de processo do PLURIX não permite implementar com eficiência.

O grau de paralelismo que pode ser explorado pelo PLURIX é adequado à arquitetura multiprocessadora de pequena escala do PEGASUS. Em casos específicos, ele proporciona ganhos de velocidade compatíveis com um pequeno número de processadores (até cerca de uma dezena). No caso mais geral, entretanto, os processadores são vistos no PLURIX como mais um recurso disponível para compartilhamento entre aplicações. Assim, a vantagem principal obtida através do uso de um número maior de processadores não é acelerar o processamento de uma aplicação, mas sim atender satisfatoriamente a um número maior de aplicações ou usuários.

Em termos de desempenho, o objetivo fundamental do MULTIPLUS e, conseqüentemente, do MULPLIX é viabilizar, através da exploração intensa de paralelismo, a execução de aplicações científicas e de engenharia que demandem uma enorme quantidade de processamento. Isto significa acelerar estas aplicações por um fator de dezenas ou centenas de vezes. A capacidade de exploração de paralelismo que o PLURIX oferece às aplicações não é suficiente para este objetivo.

O MULPLIX pode oferecer uma maior capacidade de exploração de paralelismo por parte das aplicações através da redefinição do conceito de processo, de modo a

que este incorpore a possibilidade de paralelismo. O suporte por parte do núcleo do MULPLIX a esta redefinição implica em alterações na escalação de processos e gerência de memória e na implementação de novos mecanismos para sincronização a nível das aplicações.

2.2.2 Adequação à Arquitetura MULTIPLUS

O bom desempenho da arquitetura MULTIPLUS em configurações incluindo um grande número de Nós de Processamento depende de uma utilização adequada da estrutura hierárquica de memória. Isto significa aproximar processamentos e os seus dados correspondentes, se possível alocando-os nos mesmos Nós de Processamento.

A implementação do PLURIX não contempla este aspecto porque ele influencia muito pouco o desempenho do PEGASUS. No caso do PEGASUS, que apresenta a estrutura de compartilhamento de memória baseada em um barramento único ligando os processadores e os módulos de memória, o tempo de acesso à memória principal independe do processador e do módulo de memória envolvidos. O único cuidado que se justifica em relação a este aspecto é o de conservar um processo executando num mesmo processador, qualquer que este seja, de modo a aproveitar ao máximo a informação já presente na unidade de memória cache do processador.

No caso do MULPLIX, a obtenção de um bom desempenho na arquitetura MULTIPLUS exige alterações a nível do núcleo do sistema, em relação ao PLURIX, principalmente na escalação de processos e na gerência de memória.

2.2.3 Conclusão

Considerando os objetivos particulares de cada projeto e as diferenças de arquitetura entre as máquinas base para desenvolvimento (MULTIPLUS e PEGASUS), o MULPLIX representa em alguns aspectos uma evolução do PLURIX. Deste modo, o MULPLIX é um ambiente mais completo para o desenvolvimento e execução de programas paralelos e utiliza técnicas, para gerência de memória, gerência de processos e sincronização a nível das aplicações, mais adequadas à programação paralela e à arquitetura MULTIPLUS.

2.3 Princípios Fundamentais

O desenvolvimento integral do trabalho de definição e construção do sistema operacional MULPLIX foi orientado por três princípios fundamentais:

Simplicidade — Este princípio se reflete principalmente na definição das seguintes técnicas de projeto. (1) **Transparência** dos aspectos fundamentais para os níveis mais altos de abstração. A estruturação do sistema em camadas com níveis crescentes de abstração deve ocultar de modo gradativo os aspectos menos importantes das camadas inferiores, transferindo, entretanto, a responsabilidade pelas questões fundamentais para as camadas superiores. (2) Concentração do esforço de desenvolvimento no atendimento aos objetivos especificados. (3) Adoção de estruturas de programação simples; somente a avaliação do sistema baseada na experiência de uso pode justificar estruturas mais complexas.

Prioridade para os Casos Mais Freqüentes — A melhoria no desempenho do sistema nos casos de uso mais freqüente contribui de modo mais significativo para a melhoria no desempenho do sistema como um todo. Este fato é expresso pela **Lei de Amdahl**, que afirma [35]:

> *A melhoria no desempenho a ser ganha utilizando um modo de execução mais rápido é limitada pela fração do tempo em que o modo mais rápido pode ser usado.*

Exploração de Paralelismo — Conforme explicado na seção 2.1, o desempenho efetivo total da arquitetura MULTIPLUS depende da exploração intensa de paralelismo tanto em termos de processamento como de comunicação.

Os princípios fundamentais de simplicidade e de prioridade para os casos mais freqüentes são genericamente válidos para o projeto de qualquer sistema. O princípio de exploração de paralelismo, entretanto, tem validade especial para este trabalho e, por isso, merece algumas considerações adicionais.

A maior utilização do poder de processamento através da exploração de paralelismo normalmente aumenta a necessidade de comunicação (em algum nível) entre os processadores. A tentativa de aumento puro e simples da taxa de utilização de processadores em um computador paralelo pode ocasionar uma necessidade de comunicação além da capacidade do computador; neste caso o funcionamento dos processadores é suspenso por um ou mais ciclos de modo a ajustar a necessidade de comunicação à capacidade da arquitetura.

A não consideração da comunicação entre processadores como uma questão fundamental explica a dificuldade encontrada na atual geração de sistemas multiprocessadores, baseada em barramento único e coerência automática e integral de cache, para escalar o desempenho de aplicações paralelas com o número de processadores, quando este ultrapassa uma ou duas dezenas [32]. Considerando-se a tendência corrente na evolução tecnológica de aumento do poder de processamento em uma proporção superior ao aumento de capacidade de comunicação, este fator tende a tornar-se cada vez mais crítico.

A comunicação entre processadores pode ser classificada em **compartilhamento de dados** e **sincronização**. As seguintes técnicas podem ser adotadas para aumentar o grau de paralelismo tanto em termos de processamento como em termos de comunicação:

Isolamento — Esta técnica tem como objetivo reduzir a necessidade de comunicação entre processadores. O **isolamento espacial** consiste em organizar os processamentos de modo que cada processador opere preponderantemente sobre dados de seu uso exclusivo, a fim de evitar a necessidade de compartilhamento de dados. O **isolamento temporal** consiste em organizar os processamentos de modo que cada processador possa progredir num ritmo o mais independente possível do ritmo de progresso dos outros processadores, a fim de evitar a necessidade de sincronização.

Localidade — Esta técnica tem o objetivo de aumentar a eficiência na comunicação entre processadores. A **localidade espacial** consiste em localizar fisicamente próximos os processadores que se comunicam. A **localidade temporal** consiste em realizar simultaneamente os processamentos que precisam se sincronizar.

A exploração de paralelismo, tanto em termos de processamento como em termos de comunicação, é uma questão fundamental porque determina, em última instância, a escalabilidade do desempenho de arquiteturas multiprocessadoras (como a arquitetura MULTIPLUS) em relação ao número de Nós de Processamento. Conseqüentemente, de acordo com a primeira técnica derivada do princípio da simplicidade, a exploração de paralelismo em ambos os aspectos deve ser transparente aos níveis de abstração mais altos. Isto significa que as aplicações paralelas devem definir explicitamente para o núcleo do MULPLIX não somente as unidades de processamento paralelo como também a comunicação entre estas unidades.

Parte II

Programação Paralela no MULPLIX

Capítulo 3

Definição de um Modelo Básico de Computação Paralela

A definição de um modelo teórico universal para um computador paralelo é objeto de intensas pesquisas recentes [1][4][17][24][42]. O modelo PRAM é atualmente o modelo mais importante para o estudo de algoritmos paralelos. Existem, entretanto, algumas dificuldades e/ou inconveniências para a implementação na arquitetura MULTIPLUS das abstrações em que o modelo PRAM se baseia.

O conceito de processo tradicionalmente suportado por sistemas operacionais tipo UNIX representa um exemplo bem sucedido do ponto de vista técnico e industrial do chamado modelo *"Von Neumann"* de computador seqüencial. A execução de um programa paralelo a partir de um conjunto de processos UNIX é, entretanto, ineficiente em razão, principalmente, do alto custo envolvido nos mecanismos providos pelo sistema para a comunicação entre processos.

O modelo básico para programação paralela adotado no MULPLIX representa simultaneamente uma derivação do modelo PRAM e uma adaptação do conceito de processo UNIX, com o objetivo de torná-lo capaz de executar eficientemente aplicações paralelas na arquitetura MULTIPLUS.

Este capítulo define conceitualmente o modelo básico adotado pelo MULPLIX para programação paralela. O capítulo 4 descreve uma implementação do modelo a nível de primitivas providas pelo sistema operacional MULPLIX.

O capítulo está dividido em três seções. A seção 3.1 defende a importância da definição de modelos teóricos adequados para a programação paralela. A seção 3.2 avalia a viabilidade de implementação do modelo PRAM na arquitetura MULTIPLUS. Finalmente, a seção 3.3 define o modelo MULPLIX para programação paralela.

3.1 Importância de um Modelo de Computação Paralela

O modelo de Von Neumann tem uma importância fundamental para o espetacular desenvolvimento dos sistemas de computação eletrônica seqüencial [43]. Isto se deve a razões de natureza técnica/teórica e de natureza industrial.

Do ponto de vista técnico/teórico:

- *Universalidade da máquina de Turing.* A máquina de Turing é um formalismo matemático correspondente ao modelo de Von Neumann. A Teoria da Computação [26] demonstra que a máquina de Turing é capaz não apenas de executar qualquer função computável como também de simular qualquer computador.
- *Técnicas de compilação eficientes.* Na década de 1950, a compilação das chamadas linguagens de alto nível para a máquina de Von Neumann era notoriamente considerada um problema difícil. O esforço desenvolvido desde então resultou em técnicas sistemáticas e eficientes para a grande maioria das tarefas envolvidas no processo de compilação [3].
- *Implementação eficiente por hardware.* Os avanços na tecnologia eletrônica, especialmente na área da microeletrônica, possibilitaram uma evolução dos computadores Von Neumann, em termos de desempenho e eficiência[1], sem precedentes em outras áreas da engenharia.

Do ponto de vista industrial:

- *Referência para Programação de Computadores.* O modelo de Von Neumann serviu como base universal para a definição de inúmeras **linguagens de programação de alto nível**, algumas de uso extremamente difundido, tais como FORTRAN, Cobol, Pascal e C. O desenvolvimento dos produtos da indústria de software a partir de linguagens de alto nível amplia consideravelmente o espectro de computadores em que estes produtos são aplicáveis e, conseqüentemente, estimula maiores investimentos tanto por parte de produtores como de consumidores de software.
- *Referência para o Projeto de Computadores.* O projeto de computadores tem evoluído muito rapidamente como conseqüência do desenvolvimento de novas tecnologias eletrônicas e da exploração de novas idéias de arquitetura. Esta

[1] Consumo de energia, tamanho físico, custos fixos na fabricação em massa, etc...

evolução, a nível industrial, tem normalmente sido direcionada para a construção de máquinas Von Neumann cada vez mais eficientes. Deste modo os projetistas de hardware para computadores têm a garantia da existência de software aplicativo que pode usufruir dos benefícios decorrentes destas inovações.

Analogamente ao ambiente seqüencial, o estabelecimento de um modelo adequado para um ambiente paralelo poderá impulsionar vigorosamente o desenvolvimento dos computadores paralelos. Mas, para que este modelo efetivamente sirva como referência tanto para a programação como para o projeto de computadores paralelos, é necessário que ele seja teoricamente universal, possa ser implementado eficientemente por hardware e disponha de linguagens de alto nível adequadas e compiláveis eficientemente.

3.2 Análise do Modelo PRAM

O modelo *"Random Access Machine"* (RAM) [2] é uma formalização elegante do modelo de Von Neumann que, em virtude de sua grande simplicidade e de proporcionar uma razoável estimativa de custo, tem servido extensamente de base para o projeto e a análise de algoritmos em computadores seqüenciais.

3.2.1 Definição

O modelo *"Parallel Random Access Machines"* (PRAM) [22] é uma extensão para computadores paralelos do modelo RAM, baseada no seguinte conjunto de abstrações:

- *Disponibilidade de um número ilimitado de processadores.* Cada processador equivale a uma RAM de custo uniforme (todos os tipos de instruções são executados em um mesmo período de tempo).
- *Sincronização total, automática e com granularidade mínima.* A cada passo, todos os processadores executam exatamente uma instrução.
- *Disponibilidade de um número ilimitado de células de memória compartilhada.* Todos os processadores podem realizar acessos tanto de leitura como de escrita à memória.

- *Capacidade ilimitada de acesso simultâneo à memória.* A cada passo todos os processadores podem acessar a memória. A capacidade de acesso simultâneo por um número plural de processadores a uma mesma célula de memória classifica o modelo PRAM nos seguintes submodelos: EREW PRAM ("Exclusive Read, Exclusive Write"), CREW PRAM ("Concurrent Read, Exclusive Write") e CRCW PRAM ("Concurrent Read, Concurrent Write").

3.2.2 Dificuldades para Implementação no MULTIPLUS

Um modelo deve ser definido ocultando informações irrelevantes do objeto real, de modo a possibilitar a concentração nos seus aspectos fundamentais. O modelo PRAM oculta ao programador todos os custos relativos à comunicação entre processadores. A análise desenvolvida nesta subseção demonstra que estes custos têm importância fundamental para o desempenho de aplicações paralelas na arquitetura MULTIPLUS. Conseqüentemente, neste aspecto, a definição do modelo PRAM é inadequada para os objetivos deste trabalho.

A comunicação entre processadores no modelo PRAM é realizada através de acessos à memória compartilhada e da sincronização entre os processadores. Estes temas orientam a análise a seguir.

3.2.2.1 Acesso à Memória Compartilhada

O modelo PRAM supõe que a cada passo cada processador pode realizar um acesso à memória compartilhada. A implementação desta característica exige uma enorme capacidade de comunicação, que no momento é inviável por causa dos seguintes tipos de contenção característicos da implementação física dos computadores com memória compartilhada:

- *Contenção nos Módulos de Memória.* As células de memória compartilhada normalmente são agrupadas em módulos de memória, cada um suportando um número máximo de acessos por ciclo.
- *Contenção na Rede de Interconexão.* Estes computadores normalmente são estruturados como uma rede interligando processadores e módulos de memória. Quando o número de processadores e de módulos de memória é muito grande, torna-se inviável dispor de um caminho independente ligando cada processador a todos os módulos de memória. Neste caso, os acessos que compartilham recursos de comunicação têm que ser serializados.

As limitações na capacidade de acesso paralelo à memória tornam-se mais importantes à medida em que o número de processadores aumenta. Uma solução de arquitetura para reduzir este problema, que é adotada no projeto MULTIPLUS, é organizar a memória segundo uma estrutura hierárquica, em que acessos a módulos de memória mais próximos possam ser realizados em tempos mais curtos e com maior grau de paralelismo. É importante ressaltar que esta solução somente é efetiva se o modelo de computação paralela adotado permite a utilização da técnica de isolamento espacial (seção 2.3) na programação de aplicações paralelas.

3.2.2.2 Sincronização

O modelo PRAM supõe que a execução do algoritmo pelos processadores é sincronizada a cada instrução. O tempo de execução das instruções em computadores como o MULTIPLUS está sujeito a grandes variações por causa, entre outros, dos seguintes fatores:

- *Variação no Tempo de Acesso à Memória.* De acordo com a análise no item anterior, o tempo de acesso à memória pode variar em função da contenção dos dispositivos de armazenamento e de comunicação e da distância entre o processador e a célula de memória envolvidos no acesso.
- *Variação no Tempo de Execução de Instruções Distintas*[2]. De acordo com a sua funcionalidade, a implementação em hardware de uma instrução do processador pode apresentar uma grande variação em custo, que é expressa em termos de tamanho do circuito e tempo de execução. Por exemplo: uma instrução de soma em ponto flutuante certamente é mais demorada do que uma instrução de comparação entre inteiros.
- *Interferência do Sistema Operacional.* A aplicação paralela normalmente compartilha o uso do computador não somente com outras aplicações, mas também com o próprio sistema operacional. De acordo com o funcionamento do sistema operacional, um processador pode estar indisponível enquanto os outros processadores estão executando a aplicação. Este fator tem o potencial de causar uma variação na velocidade de processamento muito superior aos fatores listados anteriormente.

O suporte à sincronização total e automática conforme previsto no modelo PRAM é extremamente oneroso porque pode haver uma discrepância enorme na velocidade

[2]Este fator não é válido para uma especialização do modelo PRAM, denominada SIMD PRAM, em que todos os processadores a cada passo executam uma mesma instrução.

de execução dos programas em processadores distintos e porque exige a realização de alguma forma de comunicação global em uma freqüência indesejavelmente grande, considerando-se a capacidade limitada de comunicação na arquitetura MULTIPLUS.

Um modelo de computação paralela que permita a construção de programas mais eficientes deve se deslocar do ponto extremo onde se encontra o modelo PRAM, em termos de sincronização, e ser capaz de suportar a utilização da técnica de isolamento temporal (seção 2.3).

3.3 Definição e Análise do Modelo MULPLIX

Considerando as dificuldades para a implementação eficiente do modelo PRAM nas arquiteturas multiprocessadoras e na arquitetura MULTIPLUS em particular, faz-se necessária a definição de um outro modelo de computação paralela mais adequado à tecnologia disponível atualmente.

As subseções a seguir definem o modelo MULPLIX e demonstram a sua derivação a partir do modelo PRAM como resultado da aplicação dos princípios fundamentais estabelecidos na seção 2.3.

3.3.1 Definição

Uma **aplicação paralela** para o MULPLIX é um conjunto de *atividades de processamento* que operam sobre dados armazenados em *variáveis* e cooperam entre si para a resolução de um problema.

Uma **atividade de processamento** é uma seqüência de instruções que, independentemente de qualquer condição externa, pode ser executada continuamente pelo processador. O **tempo de execução** de uma atividade de processamento compreende o período desde o início da execução de sua primeira instrução até o final da última instrução. Uma **variável** é a menor unidade de armazenamento de dados.

O funcionamento correto de uma aplicação paralela torna necessária uma coordenação entre as atividades de processamento. Os aspectos temporais desta coordenação representam restrições que devem ser obedecidas quanto ao tempo de execução das atividades de processamento. Estas restrições são expressas através do estabelecimento de duas **relações de sincronização** sobre as atividades de processamento: a relação de *ordem parcial* e a relação de *exclusão mútua*.

A relação de **ordem parcial** é uma relação transitiva, que define pares ordenados de atividades de processamento na forma (a, b), significando que a atividade b somente pode ser executada em um tempo posterior ao da atividade a. O adjetivo *"parcial"* indica que a relação não abrange necessariamente todos os possíveis pares de atividades[3].

A relação de **exclusão mútua** define conjuntos de atividades de processamento sujeitas a restrições quanto à sobreposição no tempo de execução. O modelo MULPLIX distingue dois casos de relação de exclusão mútua:

exclusão mútua simples — Este caso proibe completamente a sobreposição no tempo de execução das atividade de processamento pertencentes ao conjunto definido.

exclusão mútua múltipla — Este caso permite a execução, com sobreposição no tempo, de no máximo um número definido de atividades de processamento pertencentes ao conjunto definido.

A relação de exclusão mútua está diretamente associada ao controle de acesso a recursos compartilhados por atividades de processamento. Assim, o controle de acesso a um recurso único pode ser modelado por uma exclusão mútua simples; enquanto o controle de acesso a um recurso existente em número plural pode ser modelado por uma exclusão mútua múltipla.

O conjunto de atividades de processamento de uma aplicação paralela é particionado em subconjuntos denominados **linhas de controle**. Uma linha de controle estabelece uma seqüência temporal para a execução de suas atividades de processamento. Isto implica que as relações de sincronização envolvendo as atividades de processamento pertencentes a uma mesma linha de controle são automaticamente cumpridas. O cumprimento das relações de sincronização envolvendo atividades de processamento pertencentes a linhas de controle distintas é realizado através da inclusão nas linhas de controle de **atividades de sincronização**, que utilizam mecanismos explícitos de sincronização. Conseqüentemente, uma linha de controle é uma seqüência de atividades de processamento intercaladas por atividades de sincronização.

O conjunto de variáveis definidas para uma aplicação paralela é particionado em subconjuntos denominados *segmentos*. Um segmento estabelece uma seqüência

[3]Esta *"parcialidade"* é exatamente a característica que possibilita a execução paralela de atividades de processamento.

espacial para o endereçamento de suas variáveis. Cada linha de controle está associada a um **segmento privado**, que contém as variáveis usadas exclusivamente pelas atividades de processamento contidas na linha de controle. Todas as linhas de controle de uma aplicação paralela estão associadas a um (único) **segmento compartilhado**, que engloba todas as variáveis que podem ser usadas por mais de uma linha de controle da aplicação paralela.

Em resumo, uma aplicação paralela consiste de um conjunto de linhas de controle paralelas, seus segmentos privados associados e um segmento compartilhado. As linhas de controle são constituídas por uma seqüência de atividades de processamento e de atividades de sincronização. Os segmentos são constituídos por variáveis que são objeto de operações realizadas pelas atividades de processamento. As linhas de controle se comunicam através de atividades de processamento que operam sobre variáveis no segmento compartilhado e de atividades de sincronização que garantem o cumprimento das relações de ordem parcial, exclusão mútua simples e de exclusão mútua múltipla entre as atividades de processamento.

3.3.2 Análise

O modelo MULPLIX é uma derivação do modelo PRAM obtida através da aplicação de técnicas desenvolvidas com base nos princípios fundamentais estabelecidos na seção 2.3. Estas técnicas são aplicadas sobre os pontos identificados como os mais difíceis para a implementação eficiente do modelo PRAM na arquitetura MULTIPLUS: sincronização e acesso à memória.

Os ítens a seguir listam as técnicas aplicadas e as modificações resultantes realizadas no modelo PRAM para transformá-lo no modelo MULPLIX:

Transparência — A sincronização automática foi substituída pela sincronização definida explicitamente na programação dos algoritmos.

Isolamento Temporal — A sincronização total foi substituída pela sincronização parcial definida pelas relações de sincronização.

Isolamento Temporal — A granularidade da sincronização aumentou de uma instrução para uma seqüência de instruções.

Localidade Espacial — As variáveis são agrupadas em segmentos.

Isolamento Espacial — Definição de segmentos privados para as variáveis operadas exclusivamente por cada uma das linhas de controle.

Transparência — Alocação explícita das variáveis aos segmentos.

O modelo PRAM representa uma posição extrema: granularidade mínima de sincronização, sincronização total, memória totalmente compartilhada. O modelo MULPLIX representa um deslocamento a partir desta posição extrema.

A programação de uma aplicação paralela no modelo MULPLIX engloba não somente a definição dos algoritmos e das estruturas de dados, como ocorre no modelo PRAM, mas também da sincronização e da alocação das estruturas de dados nos segmentos.

O capítulo seguinte mostra que o modelo MULPLIX pode ser implementado a nível de sistema operacional através de uma extensão do conceito de processo UNIX.

Capítulo 4

Primitivas para Programação Paralela

Este capítulo aborda a implementação do modelo MULPLIX de computação paralela através do conjunto de primitivas para programação paralela providas pelo sistema operacional MULPLIX. Estas primitivas podem ser utilizadas diretamente pelas aplicações paralelas ou podem servir de base para a implementação de bibliotecas de subrotinas ou para a compilação de linguagens de alto nível paralelas ou seqüenciais, no caso de compiladores paralelizantes.

As primitivas definem para o sistema operacional MULPLIX um novo conceito de processo, que representa uma extensão ao conceito de processo normalmente suportado por sistemas operacionais tipo UNIX. O ponto principal desta redefinição de processo refere-se à incorporação da capacidade de expressão de paralelismo através da separação entre as unidades para execução e alocação de recursos. Em sistemas tipo UNIX um processo é tanto a unidade para alocação de recursos como a unidade de execução. No caso do MULPLIX, um processo é igualmente a unidade para alocação de recursos, mas a unidade de execução é a linha de controle. Assim um processo é um ambiente para a execução de uma ou mais linhas de controle. Este conceito de processo é semelhante ao conceito de *"task"* suportado pelo sistema operacional Mach [36].

Um processo aloca, através do núcleo do MULPLIX, um conjunto de recursos que são, em sua maioria, compartilhados pelas linhas de controle associadas ao processo. Este compartilhamento de recursos facilita enormemente a comunicação entre as linhas de controle. Por exemplo: o acesso compartilhado à memória, através do segmento compartilhado, permite que o compartilhamento de dados pelas linhas de

controle seja muito mais eficiente. Outro exemplo: o compartilhamento de mecanismos de sincronização permite que as atividades de sincronização sejam muito mais eficientes.

Este capítulo obedece à seguinte organização: a ativação e o término de linhas de controle são enfocados na seção 4.1; a alocação de memória é o tema da seção 4.2; os mecanismos explícitos de sincronização e as atividades de sincronização são abordadas na seção 4.3.

4.1 Criação de Linhas de Controle

Uma linha de controle, de acordo com o modelo MULPLIX de computação paralela (seção 3.3) é um conjunto de atividades de processamento programadas para execução seqüencial.

Para o núcleo do sistema operacional MULPLIX, uma linha de controle está associada a um procedimento[1], que delimita a sua execução. Considerando que a execução das diversas linhas de controle em um processo evolui independentemente, com a única exceção das atividades de sincronização, cada linha de controle necessita de registros de ativação próprios. O uso de procedimentos para delimitar a execução das linhas de controle estabelece um modo bem definido para a manipulação das pilhas de registros de ativação.

As linhas de controle são criadas sempre em **grupos** através de uma chamada ao núcleo do MULPLIX, em que são especificadas, entre outras informações, o número de linhas de controle a serem iniciadas, o procedimento associado às linhas de controle e um argumento comum para envio às linhas de controle. As linhas de controle iniciam através da chamada ao seu procedimento associado, que recebe dois argumentos: o argumento comum e a ordem desta linha de controle no grupo. Estes argumentos são suficientes para que a linha de controle seja capaz de se identificar e, entre outras coisas, distinguir os dados sobre os quais atuará.

A criação de um grupo de linhas de controle através de uma única chamada ao núcleo do MULPLIX apresenta as seguintes vantagens em comparação com a repetição sucessiva de ativações individuais:

[1] Um procedimento é uma função, subrotina ou subprograma que não define um valor de retorno.

- Possibilita ao núcleo do MULPLIX funcionar paralelamente e assim reduzir o custo do processamento associado à ativação das linhas de controle.
- O conhecimento prévio do número de linhas de controle que serão ativadas representa uma informação importante para o algoritmo de escalonamento de linhas de controle do MULPLIX.
- No caso de utilização direta por um programador, a ativação de linhas de controle em grupos associados a um procedimento facilita o entendimento da estrutura de funcionamento da aplicação paralela.

A ativação das linhas de controle pode ser **síncrona** ou **assíncrona**. A ativação síncrona causa o bloqueio da linha de controle até que todas as linhas de controle no grupo terminem a sua execução. A ativação assíncrona permite a execução paralela da linha de controle com as linhas de controle por ela ativadas. Normalmente, o término da execução de um grupo de linhas de controle é um fato importante para a sincronização da aplicação paralela. A ativação síncrona evita o uso de mecanismos explícitos de sincronização para o reconhecimento deste fato.

A ativação das linhas de controle proporciona ao programador com maior conhecimento da arquitetura MULTIPLUS uma oportunidade para estabelecer um mapeamento de cada uma das linhas de controle a um **NP preferencial**. O estabelecimento de um NP preferencial para uma linha de controle indica para o núcleo do MULPLIX um processador para a execução da linha de controle e um módulo de memória para a alocação da memória associada à linha de controle (veja a próxima seção).

4.2 Alocação de Memória

No MULPLIX as linhas de controle dispõem para alocação de dados das seguintes áreas contínuas de memória, denominadas **segmentos de dados**:

Dados Compartilhados — Este segmento contém os dados que são objeto de leitura e/ou escrita por mais de uma linha de controle do processo. O segmento de dados compartilhados pode ser acessado por todas as linhas de controle do processo. (Pode-se incluir dados com valores iniciais estabelecidos.)

Dados Privados — Este segmento contém os dados de uso exclusivo pela linha de controle. Naturalmente o segmento de dados privados de uma linha de controle não pode ser acessado por outras linhas de controle.

O tamanho inicial dos segmentos de dados é definido estaticamente no programa executado pelas linhas de controle. A alocação de memória para os segmentos de dados em seu estado inicial é realizada implicitamente pelo núcleo do MULPLIX, no momento da carga do programa a ser executado pelas linhas de controle do processo.

O tamanho dos segmentos de dados pode ser estendido dinamicamente pelas linhas de controle através da execução de chamadas ao núcleo do MULPLIX. Estas chamadas permitem a alocação explícita de incrementos para os segmentos de dados.

A alocação de um incremento para o segmento de dados compartilhados pode ser uma alocação **concentrada** ou uma alocação **distribuída**. A alocação concentrada indica ao núcleo do MULPLIX que este incremento será acessado em maior freqüência pela linha de controle que está requerendo a alocação. A alocação distribuída indica uma previsão de acesso uniformemente distribuído por todas as linhas de controle do processo.

O núcleo do MULPLIX normalmente aloca os segmentos de dados privados e as porções do segmento de dados compartilhados com alocação concentrada no NP preferencial associado a uma linha de controle. Essa associação é estabelecida opcionalmente pela aplicação paralela na ativação da linha de controle. Esta característica permite uma execução mais eficiente de aplicações programadas a partir de um conhecimento mais detalhado da arquitetura MULTIPLUS.

4.3 Sincronização entre Linhas de Controle

O sistema operacional MULPLIX oferece dois mecanismos explícitos de sincronização: os semáforos **mutex**, para garantir o cumprimento da relação de exclusão mútua, e os semáforos **event**, para garantir o cumprimento da relação de ordem parcial.

Esta seção está organizada como se segue. A subseção 4.3.1 caracteriza as necessidades específicas de sincronização em computadores paralelos. As subseções 4.3.2 e 4.3.3 descrevem o funcionamento dos mecanismos de sincronização providos pelo MULPLIX.

4.3.1 Sincronização em Computadores Paralelos

O desenvolvimento de computadores multiprogramados teve como objetivo principal tornar economicamente viável o uso de computadores, através de seu compar-

tilhamento por uma comunidade de usuários. As atividades de sincronização eram utilizadas neste contexto para a coordenação entre operações de entrada e saída e atividades de processamento, realizadas concorrentemente por diversos usuários.

O desenvolvimento de computadores paralelos, como o MULTIPLUS, tem como objetivo principal acelerar significativamente o processamento de aplicações científicas e de engenharia através da utilização de um número elevado de processadores. Conseqüentemente, as atividades de sincronização em computadores paralelos apresentam algumas características que as distinguem das atividades de sincronização normalmente associadas a sistemas multiprogramados:

- *Operação iterativa.* Aplicações científicas e de engenharia normalmente operam iterativamente sobre conjuntos de dados. Esta característica é refletida pelas atividades de sincronização.
- *Operação por grupos de linhas de controle e não apenas por linhas de controle individuais.* Esta característica é conseqüência da exploração estruturada de um grau maior de paralelismo.
- *Maior freqüência de uso.* À medida em que a exploração de paralelismo é mais intensa, a freqüência com a qual as atividades de sincronização são executadas torna-se maior.

Estas características trazem duas conseqüências em relação às atividades de sincronização: (1) o tempo de processamento a elas associado torna-se proporcionalmente maior em comparação com o tempo dedicado às atividades de processamento, e (2) a contenção de recursos de uso compartilhado pelos processadores executando atividades de sincronização torna-se um risco eminente. Assim, o custo de implementação das atividades de sincronização tem importância fundamental.

Do ponto de vista da definição de primitivas de sincronização, é importante que tipos de operações freqüentes possam ser executados preferencialmente através de uma única chamada, evitando-se a execução de mais do que uma primitiva em seqüência. Naturalmente, isto deve representar um compromisso com o objetivo conflitante de generalidade para as primitivas.

4.3.2 Semáforos para Exclusão Mútua

Os semáforos mutex são mecanismos para sincronização entre linhas de controle que têm como objetivo fazer cumprir a relação de exclusão mútua, assim cada mutex está associado a um conjunto de atividades de processamento cuja sobreposição no

tempo é proibida. Um mutex pode estar DISPONÍVEL ou OCUPADO, indicando respectivamente a permissão ou não para a execução de uma destas atividades.

As seguintes operações são definidas para um mutex:

Criação Esta operação é uma preparação para o uso do mutex. Um mutex é sempre criado no estado DISPONÍVEL.

Alocação No caso de exclusão mútua simples, a alocação de um mutex o torna (ou o mantém) OCUPADO. No caso de exclusão mútua múltipla, o mutex somente se torna OCUPADO através da alocação simultânea por um número de linhas de controle definido na criação do mutex.

Liberação A liberação de um mutex o torna DISPONÍVEL.

Extinção Esta operação indica que o mutex não será mais utilizado. Normalmente, um mutex não é extinto enquanto estiver OCUPADO.

A alocação de um mutex pode ser realizada através de uma **espera** ou de uma **verificação**. A espera por um mutex bloqueia a linha de controle até que o mutex esteja DISPONÍVEL. A verificação de um mutex não bloqueia a linha de controle, mas, como anteriormente, o mutex somente é alocado se estiver DISPONÍVEL no momento da verificação.

A possibilidade de alocação de um mutex através de verificação é necessária em determinados casos para evitar a ocorrência de situações de bloqueio fatal (*"deadlock"*). Nestes casos, em que uma linha de controle depende de mais de um semáforo para progredir, a linha de controle tem a oportunidade de ao verificar a não disponibilidade imediata de um determinado semáforo, liberar os mutex já alocados e reiniciar a alocação destes mutex, de modo a permitir que outras linhas de controle tenham a oportunidade de alocá-los.

4.3.3 Semáforos para Ordem Parcial

Um semáforo event é um mecanismo para sincronização entre linhas de controle que tem como objetivo fazer cumprir a relação de ordem parcial entre atividades de processamento. Um event pode estar ATIVO ou INATIVO, indicando respectivamente a confirmação ou não de que determinadas atividades de processamento já foram concluídas.

As seguintes operações são definidas para um event:

Criação Esta operação é uma preparação para o uso do event. Um event é sempre criado no estado INATIVO.

Sinalização A sinalização de um event o torna ATIVO. Normalmente, a sinalização de um event exige a participação de um número predeterminado de linhas de controle (distintas). Enquanto este número não é atingido, o event permanece INATIVO e dizemos que a sinalização está **pendente**.

Reconhecimento O estado de ativação de um event é percebido pelas linhas de controle através de uma operação de reconhecimento. Normalmente, um event é programado para ser reconhecido por um número predeterminado de linhas de controle; quando este número é atingido o event torna-se automaticamente INATIVO; enquanto isto não ocorre dizemos que o reconhecimento está **incompleto**.

Ativação Esta operação torna o event ATIVO imediata e incondicionalmente.

Desativação Esta operação torna o event INATIVO imediata e incondicionalmente.

Extinção Esta operação indica que o event não será mais utilizado. Normalmente, um event não é extinto enquanto o seu reconhecimento estiver incompleto.

Durante a criação de um event, as suas características são definidas: número de linhas de controle para uma sinalização completa e número de linhas de controle para um reconhecimento completo.

Analogamente à alocação de um mutex, o reconhecimento de um event pode ser realizado através de uma **espera** ou de uma **verificação**. A espera bloqueia a linha de controle até que o event esteja ATIVO. A verificação normalmente não bloqueia a linha de controle, mas somente resulta no reconhecimento se o event está ATIVO no momento da verificação. Conforme explicado em relação aos semáforos mutex, a possibilidade de reconhecimento através de verificação é necessária para evitar *"deadlocks"*.

A sinalização de um event pode ser **síncrona** ou **assíncrona**. A sinalização síncrona significa que a linha de controle sinaliza o event e a seguir o reconhece por espera. A sinalização assíncrona normalmente não bloqueia a linha de controle.

As operações de ativação e desativação são necessárias em situações de erro ou de impossibilidade de sinalização ou reconhecimento normal de um event, que poderiam causar o bloqueio indesejado de linhas de controle.

Capítulo 5

Exemplo: Eliminação Gaussiana

A resolução de um sistema de equações lineares simultâneas é um problema fundamental que aparece em diversas áreas científicas e de engenharia, sendo, por isso, considerada um exemplo representativo de aplicação para computadores de alto desempenho, servindo, conseqüentemente, de base para diversos testes para avaliação do desempenho destes computadores [39]. Por outro lado, a interdependência entre os seus dados exige que um programa paralelo eficiente para o problema contemple cuidadosamente não somente os aspectos relacionados ao processamento como também aqueles relacionados à comunicação [32]. Em função destas características, este problema tem servido extensamente como veículo para o estudo de programação paralela.

Este capítulo apresenta uma solução [38] para o problema através de um programa paralelo baseado no modelo MULPLIX e na utilização direta das primitivas providas pelo sistema operacional MULPLIX. A seção 5.1 define o problema de resolução de um sistema de equações lineares simultâneas e faz uma revisão do Método da Eliminação Gaussiana. A seção 5.2 analisa as possibilidades de exploração de paralelismo existentes no método avaliando a eficiência destas possibilidades na arquitetura MULTIPLUS. A seção 5.3 apresenta uma solução adequada à arquitetura MULTIPLUS, incluindo a sincronização e compartilhamento de dados entre as linhas de controle e mostrando a utilização de primitivas para programação paralela definidas no sistema operacional MULPLIX.

5.1 Definição do Problema e Revisão do Método de Solução

Um sistema de equações lineares com n incógnitas e n equações simultâneas, ou, abreviadamente, um **sistema linear**, pode ser escrito na forma:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{5.1}$$

Os **índices** i e j estão compreendidos nas faixas definidas pelas relações $1 \leq i \leq n$ e $1 \leq j \leq n$. As variáveis x_j são as **incógnitas** do sistema. As constantes a_{ij} são chamadas **coeficientes**. As constantes b_i são chamadas **termos independentes.**

Uma **solução** para um sistema linear é um conjunto de valores para $x_1, x_2, \cdots, x_n$ que satisfaz simultaneamente todas as equações 5.1. Dois sistemas lineares são ditos **equivalentes** quando admitem as mesmas soluções. Se as equações de um sistema linear forem linearmente independentes e consistentes entre si, então existe uma única solução para o sistema [18]. Este é o caso de interesse para este capítulo.

Com o objetivo de facilitar a sua manipulação em computador, um sistema linear pode ser mais convenientemente representado pela equação matricial

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{5.2}$$

ou, equivalentemente, fazendo $A = (a_{ij})$, $x = (x_j)$, $b = (b_i)$, como:

$$Ax = b \tag{5.3}$$

onde A é a matriz de coeficientes, x é o vetor de incógnitas e b é o vetor de termos independentes.

A idéia principal do método de resolução de sistemas lineares adotado neste capítulo consiste na transformação do sistema linear dado em um outro sistema linear equivalente e de solução mais fácil, como por exemplo o caso trivial em que a matriz de coeficientes do sistema é a matriz identidade.

A transformação do sistema linear ocorre pela substituição de cada equação pela sua soma com uma combinação linear das demais equações no sistema. Estas substituições mantêm o sistema equivalente ao sistema original e as equações continuam sendo linearmente independentes e consistentes entre si [12].

A estratégia adotada para a resolução do sistema consiste de duas etapas:

1. **Triangularização** — Transformação do sistema em um sistema equivalente no qual a matriz de coeficientes A^n seja uma matriz triangular superior. Isto significa que todos os elementos abaixo da diagonal principal são nulos, ou seja, $a_{ij}^n = 0 \ \forall i > j$. A equação 5.4 mostra o sistema linear triangular equivalente.

$$\begin{aligned} a_{11}^n x_1 + \ a_{12}^n x_2 + \cdots + \ a_{1n}^n x_n &= b_1^n \\ a_{22}^n x_2 + \cdots + \ a_{2n}^n x_n &= b_2^n \\ &\vdots \\ a_{nn}^n x_n &= b_n^n \end{aligned} \tag{5.4}$$

2. **Substituição Regressiva** — Um sistema linear triangular tem resolução mais fácil do que o caso geral de sistema linear. O valor da incógnita x_n pode ser imediatamente calculado na última equação. De modo geral, o valor da incógnita x_k pode ser calculado a partir do valor das incógnitas de x_n até x_{k+1}. A substituição regressiva consiste em calcular iterativamente os valores das incógnitas desde x_n até x_1 através da substituição das demais incógnitas nas equações 5.4 pelos seus valores calculados nas iterações anteriores.

O método mais freqüentemente utilizado em computadores para a triangularização da matriz de coeficientes A é o método de Eliminação Gaussiana. Este método consiste da anulação sucessiva, para cada coluna A_k, $1 \leq k < n$, dos coeficientes a_{ik}, $k < i \leq n$. O passo k anula os coeficientes a_{ik} somando cada linha i com a linha k multiplicada pelo coeficiente de anulação m_{ik} definido pela equação 5.5.

$$m_{ik} = -\frac{a_{ik}^{k-1}}{a_{kk}^{k-1}}, \quad k < i < n \tag{5.5}$$

A equação 5.6 resume o passo k da triangularização.

$$a_{ij}^k = a_{ij}^{k-1} \times m_{ik}, \ \ 1 \leq k < n, \ k < i < n, \ k \leq j < n \tag{5.6}$$

5.2 Análise das Possibilidades de Exploração de Paralelismo

5.2.1 Cálculos Independentes

Uma análise da equações 5.5 e 5.6 revela que, em um passo k, o cálculo dos coeficientes a_{ij}^{k} depende apenas de valores calculados no passo $k-1$. Isto significa que a cada passo k todos os cálculos podem ser realizados independentemente e, conseqüentemente, em paralelo.

5.2.2 Distribuição de Dados e de Processamento

Os seguintes critérios são utilizados para a comparação do custo envolvido nas diversas possibilidades de distribuição de dados e de processamento entre as linhas de controle:

Compartilhamento de Informação — Quantidade de coeficientes, cujos valores são necessários para o cálculo dos novos valores dos coeficientes correspondentes a cada linha de controle.

Sincronização — Quantidade de tempo em que os processadores dispendem sinalizando e aguardando semáforos.

Gerência de Linhas de Controle — Quantidade de tempo e memória gastos para a criação e escalação das linhas de controle.

As seguintes opções de distribuição foram avaliadas para a etapa de Triangularização:

Sem Agrupamento — Cada linha de controle é responsável por um único coeficiente.

Agrupamento por Linhas ou Colunas — Cada linha de controle é responsável pelos coeficientes em uma linha ou coluna.

Agrupamentos por Submatrizes Contínuas — Cada linha de controle é responsável pelos coeficientes dentro de uma submatriz da matriz A.

Agrupamentos por Submatrizes Espalhadas — Semelhante à opção anterior, porém as submatrizes são particionadas e as partições resultantes são espalhadas uniformemente pela matriz A.

A distribuição sem agrupamento representa a tentativa de explorar o máximo de paralelismo de processamento, baseada na constatação de que, a cada passo k, os cálculos dos coeficientes são independentes uns dos outros. O custo associado à gerência de linhas de controle torna esta opção proibitiva.

A distribuição por linhas ou colunas apresenta um custo de compartilhamento de dados muito grande. O cálculo de m coeficientes por uma linha de controle demanda o compartilhamento de $m + 1$ coeficientes com outras linhas de controle.

A distribuição por submatrizes contínuas apresenta um custo menor de compartilhamento de dados. O cálculo de m coeficientes por uma linha de controle demanda o compartilhamento de $\sqrt{2m}$ coeficientes com outras linhas de controle. O custo de gerência de linhas de controle é entretanto alto, porque, à medida em que o processamento avança, linhas de controle vão se tornando inúteis, em razão da anulação dos coeficientes em suas submatrizes contínuas.

A distribuição por submatrizes espalhadas alia um baixo custo de compartilhamento de informações com um bom balanceamento de carga e é a solução adotada.

5.3 Uma Solução Adequada ao MULTIPLUS

As figuras a seguir mostram a programação da linha de controle original (figura 5.3) que comanda a execução paralela da Eliminação Gaussiana (figura 5.3) e da Substituição Regressiva (figura 5.3). A constante `P` indica o número de processadores no sistema, que para simplificação é um quadrado perfeito da constante `L`.

A sincronização da Eliminação Gaussiana é baseada na programação de dois semáforos `event`. O semáforo `transporte` indica, a cada passo k, que a linha base (a linha que é multiplicada e somada às demais linhas) e os coeficientes de anulação já se encontram totalmente disponíveis. O semáforo `eliminação` indica o término do passo k da Eliminação Gaussiana.

```
    Linha de Controle Original

/*  ---- 1a Etapa ----
 */
transporte  =  ev_create (2L, P)
eliminação  =  ev_create (P, P)

spawn (P, triangularização, 0)

/*  ---- 2a Etapa ----
 */
para l = 0 até N-1 faça
    resolução[l] = ev_create (1, l)

spawn (P, substituição, 0)

/*  --- Finalização ---
 */
ev_delete (transporte)
ev_delete (eliminação)

ev_wait (resolução[0])

para l = 0 até N-1 faça
    ev_delete (resolução[l])
```

Figura 5.1: Linha de Controle Original

```
            Eliminação Gaussiana

triangularização (arg, p)
{
    para passo = 0 até N-1 faça
    {
        se processador p está na linha de atuação
        {
            copia_linhabase
            ev_signal (transporte)
        }

        se processador p está na coluna de atuação
        {
            calcula_coeficientes
            ev_signal (transporte)
        }

        ev_wait (transporte)

        processa_eliminação
        ev_swait (eliminação)
    }
}
```

Figura 5.2: Eliminação Gaussiana

```
            Substituição Regressiva

substituição (arg, p)
{
    para cada linha l alocada ao processador p
    {
        para cada coeficiente i na linha l
        {
            ev_wait (resolução[i])
            RAIZ[i] = RAIZ[i] - Raizes[i] * A[l][i]
        }
        Raizes[l] = RAIZ[i] / A[l][l]
        ev_signal (resolução[l])
    }
}
```

Figura 5.3: Substituição Regressiva

Parte III

A Implementação do MULPLIX

Capítulo 6

Escalação de Linhas de Controle

O conceito de processo normalmente suportado por sistemas operacionais tipo UNIX representa não somente o ambiente para a execução de um programa, como também a unidade de execução do programa. Conforme explicado nos capítulos 3 e 4, a fim de facilitar a programação de aplicações paralelas, o MULPLIX estende este conceito de processo, possibilitando a definição de múltiplas unidades de execução para um mesmo processo, denominadas linhas de controle do processo.

A escalação de linhas de controle determina a alocação ao longo do tempo de processadores para a execução das linhas de controle dos processos. A escalação de linhas de controle no MULPLIX é um problema análogo à escalação de processos no caso do PLURIX ou de outros sistemas tipo UNIX. As políticas adotadas entretanto, diferem em função do objetivos de exploração de paralelismo e prioridade para a obtenção de maior desempenho para uma única aplicação, em contraposição aos objetivos estabelecidos para um sistema de compartilhamento de tempo voltado para um ambiente multiusuário interativo.

Este capítulo enfoca a escalação de linhas de controle no MULPLIX. A seção 6.1 analisa de modo geral o problema de escalação em sistemas de computação paralelos. A seção 6.2 determina os objetivos para a escalação de linhas de controle no MULPLIX. A seção 6.3 descreve as políticas e a implementação da escalação no MULPLIX. A seção 6.4 realiza uma comparação entre a escalação nos sistemas MULPLIX e PLURIX.

6.1 Análise Geral do Problema de Escalação

Esta seção analisa as diversas alternativas de funcionamento para um escalador de um computador paralelo. Um escalador pode ser classificado pelas opções de funcionamento que adota [15]. Assim a sua classificação facilita a identificação suscinta do seu modo de funcionamento.

Quanto ao seu escopo, um escalador pode ser local ou global. Ele é **local** se atua apenas na escalação de tarefas a um processador. Um escalador **global** decide em que processador cada processo deve executar, deixando a decisão de escalação das tarefas em um processador para o escalador local.

Um escalador pode ser classificado segundo o momento de aquisição das informações relativas à utilização de recursos pelas tarefas. Se as informações utilizadas pelo escalador precisam estar disponíveis antes da execução das tarefas, o escalador é **estático**. Se elas são obtidas durante a execução, o escalador é **dinâmico**. Normalmente os escaladores estáticos realizam a escalação *antes* da execução das tarefas, durante a fase de geração delas. Os escaladores dinâmicos sempre realizam a escalação *durante* a execução das tarefas, com base no comportamento apresentado pelo sistema.

As classificações a seguir são pertinentes aos escaladores globais e dinâmicos.

A responsabilidade pela realização de uma escalação, classifica os escaladores em distribuídos e não distribuídos. São **distribuídos** os escaladores onde todos os processadores participam das atividades de obtenção de informações sobre o sistema e de execução das decisões da escalação.

A autoridade para a tomada de decisões pelos escaladores distribuídos, os divide em descentralizados e centralizados. Nos escaladores **centralizados** as decisões da escalação são tomadas por apenas um processador. Nos escaladores **descentralizados** todos processadores estão autorizados a tomar decisões de escalação.

Os escaladores distribuídos podem ser classificados de acordo com o grau de autonomia na determinação de como seus recursos devem ser utilizados. Eles podem ser cooperativos e não cooperativos. Se cada processador realiza ações independentemente das ações realizadas pelos demais, se preocupando apenas com as conseqüências locais destas ações, o escalador é **não cooperativo**. Nos escaladores **cooperativos** cada processador realiza as ações que lhe competem quanto à escalação, mas todos os processadores trabalham de acordo com objetivos comuns

quanto à escalação global e eventualmente sacrificam o seu desempenho individual em benefício do total.

Quanto à distribuição da carga de processamento entre os diversos processadores, um escalador pode ter uma estratégia de balanceamento de carga ou de divisão de carga. A estratégia de **balanceamento de carga** visa distribuir igualmente a carga pelos processadores. Seu princípio fundamental é que a minimização do tempo total de execução de uma aplicação, pode ser obtida através da minimização da diferença entre os tempos de término de cada processo da aplicação. Já a estratégia de **divisão de carga** visa apenas manter todos os processadores com carga. Seu princípio fundamental é que a minimização do tempo total de execução de uma aplicação pode ser obtida pela minimização do custo adicional da escalação e pela manutenção de todos os processadores ocupados durante a execução da aplicação, sem que necessariamente a carga se mantenha igualmente distribuída pelos processadores.

Os escaladores podem ainda ser classificados quanto à mobilidade das tarefas nos processadores. Um escalador pode realizar uma atribuição estática ou dinâmica de tarefas aos processadores. Na atribuição **estática** os tarefas permanecem nos processadores de sua primeira atribuição. Na atribuição **dinâmica** os tarefas podem ter sua atribuição alterada ao longo do tempo.

6.2 Objetivos para a Escalação no MULPLIX

Os escaladores dos sistemas multiprogramados interativos têm por objetivos prover a ilusão da existência de um processador para cada unidade de execução e maximizar a utilização dos processadores. Como forma de atingir estes objetivos, eles são dinâmicos e realizam um compartilhamento do tempo dos processadores entre as unidades de execução. As políticas adotadas para este compartilhamento visam distribuir com justiça o tempo de processador entre as unidades de execução, minimizar os seus tempos de reação aos comandos dos usuários e evitar esperas indefinidas.

O objetivo fundamental do MULTIPLUS é tornar viável a execução de aplicações científicas ou de engenharia que demandem muito processamento, através da exploração intensiva de paralelismo. A viabilidade está relacionada ao tempo real necessário à finalização das aplicações paralelas em execução, conseqüentemente o objetivo central da escalação no MULPLIX é minimizar este tempo.

Uma qualidade desejável do escalador é a **generalidade**. Deve ser possível a escalação de qualquer tipo de aplicação paralela. O escalador deve prover um ambiente de execução que independa da configuração disponível de hardware, mas que permita às aplicações paralelas utilizar de modo transparente e eficiente as características e os recursos do hardware disponível. O próprio escalador deve ser capaz de automaticamente se adaptar a qualquer configuração do MULTIPLUS. A generalidade do escalador deve prover ainda a existência de um mesmo ambiente para a execução de aplicações paralelas e o seu desenvolvimento.

6.3 Escalação no MULPLIX

Esta subseção apresenta as alternativas de funcionamento adotadas pelo escalador do MULPLIX e algumas heurísticas empregadas.

6.3.1 Características Gerais

O MULTIPLUS pode se apresentar sob a forma de diversas configurações diferentes. Ele pode ter desde alguns NP até milhares deles. Como forma de simplificar a adaptação do escalador a esta diversidade, o escalador é *simétrico*, ou seja o mesmo código do escalador é copiado em cada NP.

Apesar de dispor de uma pluralidade de processadores, a arquitetura MULTIPLUS refere-se a computadores paralelos unitários, portanto a sua escalação é *global*.

Os escaladores estáticos podem produzir uma escalação ótima e podem apresentar um mínimo de custo extra durante a execução, porque utilizam o conhecimento a priori das características de execução das aplicações. Entretanto, eles são inviáveis para muitas aplicações paralelas, para as quais a obtenção deste conhecimento é impossível ou demasiadamente onerosa. Os escaladores dinâmicos, por outro lado, podem se adaptar a qualquer carga de processamento, suportam eficientemente aplicações interativas e podem apresentar um desempenho próximo do ótimo quase sempre, mesmo utilizando heurísticas simples, que implicam num custo adicional desprezível. Assim a escalação *dinâmica* é mais adequada aos objetivos do MULPLIX e é a adotada.

Uma característica importante da arquitetura do MULTIPLUS é a sua organização com até milhares de NP separados em grupos. A utilização eficiente desta

organização é fundamental para o desempenho geral do sistema. Os dois tipos de operações a seguir são ineficientes: as que envolvam simultaneamente muitos NP e aquelas que só possam ser realizadas por um pequeno subconjunto dos NP. Como forma de evitar as operações do primeiro tipo, o escalador do MULPLIX é *pouco cooperativo*. Para evitar as do segundo tipo, ele é *distribuído* e *descentralizado*.

Outra característica da arquitetura do MULTIPLUS que influencia o escalador é o compartilhamento de memória por todos os NP. A possibilidade de acesso a toda memória por qualquer NP, permite uma grande mobilidade das tarefas e logo a sua atribuição dinâmica aos processadores. A atribuição dinâmica dá uma maior liberdade ao escalador, aumentando a sua adaptabilidade às condições de execução. Considerando as suas vantagens e a possibilidade de seu uso, a *atribuição dinâmica* de tarefas a processadores é adotada no MULPLIX.

6.3.2 Política de Particionamento do Tempo

A **política de particionamento do tempo** estabelece para cada processador em que momentos há oportunidade para a substituição da linha de controle ativa.

O desempenho das aplicações científicas paralelas é especialmente contemplado pelo escalador através do privilegiamento das linhas de controle destas aplicações em relação às linhas de controle de aplicações interativas.

O privilégio das linhas de controle de aplicações paralelas científicas se manifesta através do tratamento diferenciado à evolução das suas prioridades e da existência de um modo dedicado de funcionamento dos processadores. Esta diferenciação refere-se à não alteração das suas prioridades. As vantagens advindas desta diferenciação são a manutenção da prioridade inicial (*alta*) das linhas de controle privilegiadas, não considerando a utilização de processadores por elas, e a economia do processamento que seria necessário para o cálculo de sua evolução.

Quando um processador está funcionando em modo dedicado, a linha de controle que o utiliza sofre o mínimo de interferências externas. Ela executa sem interrupções, até que termine ou necessite de um recurso não disponível imediatamente.

A monopolização do MULTIPLUS pelas linhas de controle privilegiadas ocorreria se todos os processadores estivessem em modo dedicado ao mesmo tempo. Esta monopolização é inconveniente se há em execução processos interativos, ou de monitoração das aplicações paralelas em execução. Ela é impedida pela determinação

de um número máximo de processadores simultaneamente em modo dedicado, que é controlado por um semáforo. Este número é definido durante a iniciação do MULPLIX e pode ser alterado posteriormente.

6.3.3 Política de Seleção

A **política de seleção** determina qual linha de controle é escalada quando um processador torna-se disponível.

A utilização eficiente da hierarquia de memória do MULTIPLUS depende da exploração da localidade de memória. A localidade de memória cresce à medida que a posição de memória a ser acessada por um NP está na memória local de um NP em outro cluster, na memória local de outro NP no mesmo cluster, na memória local do próprio NP, ou no cache do próprio NP. Assim a próxima linha de controle a ser escalada em um processador, pertence ao processo de maior prioridade e maximiza a localidade de memória. Do mesmo modo um processo é criado no cluster que contém o PESB que gerencia o disco de seu diretório preferencial de trabalho.

6.3.4 Implementação

Como forma de maximizar a localidade de memória, há uma fila de linhas de controle prontas por cluster. Cada uma destas filas pode ser acessada por todos os processadores, mas um processador disponível só procura uma linha de controle pronta para executar na fila de outro cluster se a fila do seu cluster estiver vazia.

6.4 Comparação com a Escalação no PLURIX

A escalação no PLURIX [37], apesar de mais flexível que a do UNIX, não é adequada à exploração de paralelismo e à arquitetura MULTIPLUS. Sua inadequação à exploração de paralelismo decorre do não tratamento diferenciado à execução de aplicações paralelas. A inadequação à arquitetura MULTIPLUS advém da existência de um fila única de processos prontos e da não consideração da localidade de memória.

Capítulo 7

Gerência de Memória

Em sistemas operacionais que provêem um ambiente de multiprogramação, a gerência de memória é responsável pelo controle da utilização e do compartilhamento de memória entre os diversos processos. Isto significa que cada processo não manipula diretamente a memória, mas recebe da gerência de memória um **espaço de endereçamento virtual**. Assim a programação de aplicações é baseada em *endereços virtuais* que são **mapeados** em tempo de execução (normalmente por hardware) nos *endereços físicos*, de acordo com a **alocação** dos espaços de endereçamento virtual **na memória física**, realizada pelo sistema operacional. Além disto, qualquer tentativa por parte de um processo de leitura ou escrita em um endereço que não pertença ao seu espaço virtual de endereçamento pode ser detectada (normalmente por hardware) e impedida, sem que haja interferência no funcionamento dos outros processos.

No caso de sistemas operacionais para programação paralela como o MULPLIX, a gerência de memória deve adicionalmente dar suporte ao conceito de processo com múltiplas linhas de controle de modo a ser um instrumento para a exploração de paralelismo e aplicação das técnicas de isolamento e localidade espacial.

Este capítulo apresenta e discute a gerência de memória da versão 1.0 do MULPLIX. A seção 7.1 define o espaço de endereçamento virtual proporcionado aos processos. A seção 7.2 descreve a política adotada pelo núcleo do sistema para a alocação de memória física.

7.1 Definição do Espaço Virtual

O espaço virtual de endereçamento provido pelos sistemas tipo UNIX consiste de um conjunto de áreas contínuas denominadas **segmentos**. Normalmente, como é o caso do PLURIX, há segmentos para instruções, dados e pilha, definidos tanto para o modo usuário como para o modo supervisor.

O MULPLIX estende o espaço de endereçamento virtual provido pelo PLURIX através do particionamento dos segmentos de dados em segmentos de dados locais ou privados e segmentos de dados globais ou compartilhados. As subseções a seguir descrevem detalhadamente os segmentos de memória definidos no MULPLIX para o modo usuário e para o modo supervisor.

7.1.1 Segmentos de Memória em Modo Usuário

Os segmentos definidos abaixo estão associados em modo usuário no MULPLIX a cada linha de controle em um processo:

Texto — Este segmento contém as instruções do programa preparado pelo usuário. De acordo com a definição de processo adotada pelo MULPLIX, o segmento de texto de usuário é compartilhado por todas as linhas de controle do processo. (O segmento de texto também pode ser compartilhado com outros processos executando o mesmo programa).

O segmento de texto é alocado pelo núcleo do MULPLIX durante a preparação para a execução de um novo programa em um processo.

Biblioteca Compartilhada — Este segmento contém as instruções dos procedimentos e funções das bibliotecas mais usadas no sistema. O segmento de biblioteca compartilhada é compartilhado por todas as linhas de controle de todos os processos e tem tamanho constante definido pelo núcleo do sistema.

Dados Compartilhados — Este segmento contém as variáveis para uso por mais de uma linha de controle do processo, sendo compartilhado por todas as linhas de controle no processo.

O tamanho inicial do segmento é determinado pela quantidade de variáveis definidas estaticamente pelo programa. O segmento pode crescer dinamicamente durante a execução do programa através de chamadas ao núcleo do MULPLIX, que estabelecem o tamanho para um incremento do segmento e a sua forma de alocação como concentrada ou distribuída.

Dados Privados — Este segmento contém as variáveis para uso exclusivo por sua linha de controle. Os segmentos de dados privados ocupam uma mesma região do espaço virtual, mas naturalmente referem-se a regiões distintas na memória física. O tamanho deste segmento é definido do mesmo modo que o segmento de dados compartilhados.

Pilha — Este segmento é utilizado para a alocação dos registros de ativação dos procedimentos.[1] Este segmento é de uso exclusivo por sua linha de controle.

O tamanho inicial deste segmento é definido pelo núcleo do MULPLIX. O aumento do número de chamadas ativas a procedimentos faz com que o núcleo do MULPLIX efetive transparentemente o crescimento do segmento, de modo a que o tamanho do segmento seja sempre suficiente para armazenar os registros de ativação correpondentes aos procedimentos ativos.

7.1.2 Segmentos de Memória em Modo Supervisor

Os segmentos definidos no MULPLIX para o modo supervisor são relacionados a seguir:

Texto — Instruções para o núcleo do sistema operacional. O segmento de texto de supervisor é compartilhado por todas as linhas de controle de todos os processos.

Dados Globais — Este segmento contém variáveis para uso global por parte do sistema operacional. O segmento de dados globais de supervisor é compartilhado por todas as linhas de controle de todos os processos.

Dados Locais — Este segmento contém as variáveis utilizadas exclusivamente pelo núcleo do sistema operacional em execução neste NP. O segmento de dados locais de supervisor é compartilhado por todas as linhas de controle ativas neste NP. Este segmento não está implementado na versão 1.0 do MULPLIX

[1] Quase todas as linguagens de programação modernas incluem procedimentos recursivos, que requerem alocação de memória em pilha. Cada chamada recursiva requer a alocação de uma nova cópia das variáveis locais definidas pelo procedimento; assim a quantidade de variáveis requeridas durante a execução do programa não é conhecida em tempo de compilação. Os **Registros de Ativação** são utilizados para a implementação das chamadas recursivas: eles englobam todo o espaço de memória necessário para a execução de um procedimento. Cada vez que um procedimento é chamado (ou ativado) um registro de ativação é empilhado no segmento de pilha. Quando o procedimento retorna, o registro de ativação é desempilhado, liberando o espaço de memória correspondente.

porque o computador base para desenvolvimento EBC32010 é monoprocessado e tem uma estrutura centralizada de memória.

Pilha — Este segmento contém os registros de ativação para o modo supervisor, que incluem não apenas aqueles correspondentes aos procedimentos chamados pelo núcleo, mas também aos procedimentos ativados por interrupções. Cada linha de controle aloca um segmento de pilha de supervisor.

TCB — O bloco de controle para a linha de controle (*"Thread Control Block"* – TCB) contém informações para a gerência da linha de controle, como por exemplo o contexto de execução para a linha de controle. O TCB é de uso exclusivo por sua linha de controle.

PCB — O bloco de controle para o processo (*"Process Control Block"* – PCB) contém informações sobre o processo adicionais às informações mantidas nas estruturas de dados internas ao núcleo do sistema operacional. Estas informações podem estar fora do núcleo porque são necessárias apenas no contexto de execução das linhas de controle deste processo.

O PCB é de uso compartilhado pelas linhas de controle do processo, enquanto estas executam em modo supervisor.

7.2 Alocação de Memória Física

7.2.1 Segmentação × Paginação

O PLURIX adota um esquema de **alocação segmentada** para a memória física. O mapeamento de endereços virtuais em endereços físicos é (simplificadamente) realizado através de uma operação de soma do endereço virtual com um **endereço base** correspondente ao segmento virtual em questão [21].

A alocação segmentada exige que os segmentos virtuais sejam alocados em áreas contínuas de memória física. Esta exigência traz como conseqüência a necessidade de movimentação pela memória física de segmentos virtuais já alocados, quando não há um espaço livre contínuo para a alocação de um segmento que está sendo criado ou expandido.

O MULPLIX adota um esquema de **alocação paginada** para a memória física. Este esquema particiona a memória em um conjunto de unidades, denominadas **páginas**, com tamanhos iguais e potência de dois. Desta forma, um endereço virtual admite uma representação binária composta de duas partes:

1. o número da página — dado pelos bits mais significativos — e
2. um deslocamento dentro da página — dado pelos bits restantes.

O mapeamento de endereços virtuais em endereços físicos consiste em substituir os bits relativos ao número da página, mantendo-se o deslocamento dentro da página.

A alocação paginada é mais flexível do que a alocação segmentada porque não há restrições quanto às posições relativas das páginas de memória física alocadas para um segmento. Esta maior flexibilidade resulta nas seguintes vantagens:

- A alocação de memória correspondente à criação ou à expansão de um segmento virtual nunca exige a alteração na alocação de outros segmentos, ou da parte já alocada do segmento em expansão.
- Possibilita o uso de técnicas mais sofisticadas, como por exemplo, a cópia na escrita (*"copy on write"*) para a realização de cópias virtuais. (Esta técnica torna a criação de processos muito mais eficiente.)
- A alocação paginada funciona nos casos em que há uma *"lacuna"* no espaço de endereçamento físico. Isto pode ocorrer, por exemplo, se algum NP estiver defeituoso (a área de memória correspondente ao seu módulo de memória não pode ser utilizada).

7.2.2 Política de Alocação

A política adotada para alocação de memória física correspondente aos segmentos virtuais tem como objetivo privilegiar a velocidade de processamento da aplicação, através da observação das seguintes regras:

- O segmento de texto definido para o modo supervisor é replicado para cada NP.
- O segmento de biblioteca compartilhada é replicado para cada NP.
- O segmento de texto para o modo usuário é replicado para cada cluster em que há NP preferenciais para as linhas de controle do processo.
- Os segmentos de pilha, PCB e TCB, para o modo supervisor, e de pilha e de dados privados, para o modo usuário, são alocados preferencialmente no Módulo de Memória do NP preferencial da linha de controle.
- As porções concentradas do segmento de dados compartilhados são alocadas no Módulo de Memória do NP executando a expansão do segmento.

- As porções distribuídas do segmento de dados compartilhados são alocadas nos diversos NP preferenciais das linhas de controle do processo.
- Os segmentos de texto, dados globais e dados locais definidos para o modo supervisor são pré-alocados, ocupando as páginas de endereço mais baixo em cada NP.
- O segmento de biblioteca compartilhada é alocado a seguir, durante a iniciação do sistema.

7.3 Avaliação

A simplicidade é a característica principal da organização do espaço virtual de memória adotada no MULPLIX. Esta simplicidade possibilita grande eficiência tanto em termos da exploração de localidade no processamento pela aplicação como em termos do desempenho do núcleo do sistema operacional:

- O gerenciamento da migração e replicação de dados, com o objetivo de melhor explorar localidade no processamento, é responsabilidade da aplicação paralela, que dispõe de melhores informações a respeito da natureza dos acessos à memória. Este gerenciamento não significa um incômodo muito grave, porque pode ser realizado não apenas diretamente pelo usuário programador, mas também através de bibliotecas ou compiladores. De qualquer modo, esta característica é coerente com a intenção de não ocultar inteiramente a estrutura hierárquica de memória (subseção 3.2.2 e seção 3.3).
- O compartilhamento de um segmento único de texto por todas as linhas de controle em um processo reduz significativamente o custo de migração de linhas de controle entre processadores e, conseqüentemente, viabiliza uma maior flexibilidade para o algoritmo de escalação de linhas de controle.
- A definição das linhas de controle que podem acessar a memória compartilhada é extremamente simples, abrangendo todas as linhas de controle no processo e nenhuma linha de controle em outro processo. Esta simplificação possibilita uma implementação mais eficiente para a gerência de memória em comparação com sistemas mais complexos como, por exemplo, o sistema operacional Mach [36].
- A semelhança com o PLURIX facilita a implementação.

Capítulo 8

Implementação dos Mecanismos de Sincronização

Os mecanismos de sincronização são instrumentos para comunicação entre linhas de controle que assumem papel de fundamental importância no caso de computadores paralelos. A eficiência da implementação destes mecanismos é um dos fatores determinantes da escalabilidade de sistemas com arquiteturas multiprocessadoras de larga escala com memória compartilhada como a arquitetura MULTIPLUS.

Os mecanismos de sincronização implementados no MULPLIX podem ser classificados de acordo com o método empregado para a sua implementação em: (1) semáforos baseados em monitoração contínua e (2) semáforos baseados no escalonamento de linhas de controle.

A principal distinção entre as duas classes de semáforos refere-se à política adotada para a utilização do processador no caso do semáforo não se encontrar imediatamente no estado desejado. Os semáforos baseados na monitoração contínua mantêm o processador ocupado pela linha de controle até a mudança de estado do semáforo. Os semáforos baseados no escalonamento de linhas de controle permitem a utilização do processador por outras linhas de controle.

De modo geral, o uso de semáforos baseados em monitoração contínua resulta em um progresso mais rápido para a execução das linhas de controle quando o período de espera pela mudança de estado dos semáforos é muito curto. Por outro lado, os semáforos baseados no escalonamento de linhas de controle possibilitam um maior aproveitamento do processador no caso de um período mais longo de espera[1].

[1]Um período de tempo bem maior do que o necessário para a substituição da linha de controle executando no processador.

A versão 1.0 do MULPLIX implementa apenas um tipo de semáforo baseado em monitoração contínua: o semáforo binário para o caso de exclusão mútua simples. Este semáforo é utilizado apenas em modo supervisor na sincronização interna do núcleo do sistema e na implementação dos semáforos baseados no escalonamento de linhas de controle. A implementação de semáforos para sincronização em modo usuário baseada em monitoração contínua não tem sentido em um computador monoprocessado, como é o caso do EBC32010 utilizado como máquina para desenvolvimento da versão 1.0 do MULPLIX.

A versão 1.0 do MULPLIX essencialmente conserva a implementação dos semáforos baseados no escalonamento de linhas de controle já existentes no PLURIX [20] para sincronização interna do núcleo do sistema e utiliza a mesma estrutura para a implementação dos novos tipos de semáforos definidos para a programação paralela. O único ponto alterado refere-se à adaptação necessária para a utilização do semáforo por linhas de controle ao invés de processos. Esta estrutura de implementação é construída a partir das seguintes idéias básicas:

- As linhas de controle aguardam a mudança de estado de um semáforo em uma fila de espera definida para o semáforo através de uma função de espalhamento aplicada ao endereço do semáforo.
- A linha de controle que altera o estado do semáforo é responsável por retirar da fila de espera uma ou todas (de acordo com o tipo do semáforo) as linhas de controle que aguardavam este semáforo na fila de espera e transferi-las para as filas de linhas de controle prontas, de modo a que possam ser escaladas para execução.

A parte restante deste capítulo enfoca a implementação definida para a versão 2.0 do MULPLIX dos semáforos baseados em monitoração contínua. A seção 8.1 estabelece parâmetros e critérios para avaliação do desempenho de uma implementação destes semáforos na arquitetura MULTIPLUS. A seção 8.2 descreve a implementação dos semáforos para exclusão mútua simples. Esta implementação é estendida nas seções 8.3 e 8.4 para o caso dos semáforos exclusão mútua múltipla e ordem parcial.

A implementação de todos os semáforos baseados em monitoração contínua utiliza a instrução atômica F&I (*"Fetch and Increment"*), que retorna o conteúdo de uma variável e (imediatamente a seguir e sem interrupções) o incrementa. A definição para a implementação desta instrução na arquitetura MULTIPLUS se encontra no anexo B.

8.1 Parâmetros e Critérios para Avaliação

Os parâmetros e critérios relacionados a seguir são fundamentais para a avaliação do desempenho de um algoritmo para a implementação de mecanismos de sincronização baseados em monitoração contínua:

- Latência mínima. Este parâmetro estabelece o tempo mínimo necessário para a alocação de um semáforo, considerando-se que não há contenção.
- Taxa de *"throughput"*. Esta taxa fornece o número máximo de vezes que o semáforo pode ser alocado durante um dado intervalo de tempo, supondo-se que vários processadores estão tentando alocar o semáforo.
- Interferência no desempenho dos outros processadores. Isto pode ocorrer por exemplo através da contenção das vias de acesso à memória. As vias de acesso à memória constituem um recurso de uso extremamente crítico no caso de configurações com um número elevado de processadores.

8.2 Exclusão Mútua Simples

O algoritmo *"Queueing in Shared Memory"* apresentado por Anderson em [5] foi utilizado como base para desenvolvimento de um novo algoritmo mais adequado ao MULTIPLUS. As idéias básicas deste novo algoritmo são:

- Os processadores esperando pelo semáforo binário são enfileirados em um buffer circular, de dimensão superior ao número máximo de processadores no sistema.
- A liberação do semáforo é realizada pelo processador que o detém e inclui a transferência do semáforo para o processador seguinte na Fila (se houver).
- Um processador esperando um semáforo percebe a transferência do semáforo para si através da monitoração contínua (*"polling"*) de uma variável local no cache.

O enfileiramento dos processadores resolve a disputa pelo semáforo no momento da decisão da posição de cada um deles na Fila (esta decisão é implementada por hardware através de uma instrução atômica, conforme será visto adiante). Este procedimento resulta em um esquema de *"round robin"*, em que o tempo máximo de espera é limitado (não há *"starvation"*).

A transferência direta do semáforo do processador que o detém para o processador seguinte na Fila evita a interferência de/nos outros processadores.

A percepção da recepção do semáforo através da monitoração de uma variável local presente na Memória Cache mantém o barramento, a Rede de Interconexão e o Módulo de Memória local livres para acesso por outros processadores.

8.2.1 Definições Básicas e Estruturas de Dados

Os algoritmos operam com as seguintes variáveis e constantes:

Fila — Vetor para a implementação de uma fila circular. Cada posição da Fila pode ser ocupada pela identificação de um processador ou por uma das constantes OK e VAZIO. Há uma instância desta variável para cada semáforo.

local[p] — Área local para monitoração pelo processador p.

f_atual — Posição ocupada pelo último processador a entrar na Fila. Há uma instância desta variável para cada semáforo.

f — Posição ocupada por este processador na Fila. Esta variável conserva o seu valor entre a obtenção e a liberação do semáforo. Há uma instância desta variável para cada processador.

id — Identificação de um outro processador na Fila.

ID — Constante identificando este processador.

OK — Constante que indica a liberação do semáforo.

VAZIO — Em uma posição da Fila, esta constante indica que não há processador na posição aguardando a liberação do semáforo. Na área local de um processador, esta constante indica que o semáforo ainda não foi liberado para o processador.

As posições na Fila são representadas por inteiros de 32 bits, que admitem uma variação muito mais abrangente do que a faixa de posições disponíveis na Fila. A operação POS(x) realiza a conversão necessária, retornando a posição válida na Fila correspondente ao seu argumento x. As operações ANT(x) e SUC(x) são similares à operação POS(x), com a exceção de retornarem respectivamente as posições anterior e posterior na Fila em relação à posição retornada por POS(x). Estas operações admitem implementação direta através de máscaras de bits, se o tamanho da Fila é uma potência de 2.

8.2.2 Algoritmos

Os algoritmos para a iniciação, obtenção e liberação de um semáforo binário são explicados a seguir.

```
Iniciação de um Semáforo Binário

  para cada posição i de Fila
          Fila[i] := VAZIO
  f_atual := 0
  Fila[ANT (f_atual)] := OK
```

Figura 8.1: Iniciação do Semáforo Binário

O algoritmo para a **iniciação** prepara a Fila de processadores, indicando a disponibilidade do semáforo para o primeiro processador que executar o algoritmo para obtenção.

```
    Obtenção de um Semáforo Binário

local[ID] := VAZIO                          (1)
f := POS (F&I (atual))                      (2)
Fila[f] := ID                               (3)
se Fila[ANT (f)] <> OK                      (4)
        espera (local[ID] = OK)
```

Figura 8.2: Obtenção do Semáforo Binário

O algoritmo para **obtenção** consiste dos seguintes passos: (1) preparação da área local para a monitoração futura, (2) alocação de uma posição na Fila, (3) posicionamento do processador na Fila, (4) se o processador que estava posicionado na Fila imediatamente à frente já liberou o semáforo, então o semáforo já está obtido; em caso contrário, a liberação do semáforo é aguardada através da monitoração contínua da área local.

O algoritmo para a **liberação** consiste dos seguintes passos: (1) avisar na Fila que o semáforo está liberado, (2) se já há algum outro processador imediatamente atrás na Fila, avisar também em sua área local, (3) reiniciar a posição imediatamente à sua frente (a sua própria posição é usada para indicar a liberação do semáforo).

```
  Liberação de um Semáforo Binário

Fila[f] := OK                          (1)
se Fila[SUC(f)] = id                   (2)
       local[id] := OK
Fila[ANT(f)] := VAZIO                  (3)
```

Figura 8.3: Liberação do Semáforo Binário

8.3 Exclusão Mútua Múltipla

A exclusão mútua múltipla representa uma extensão ao caso de exclusão mútua simples porque pode envolver um número plural de processadores com o semáforo e, conseqüentemente, a liberação concorrente do semáforo por mais de um processador.

A solução adotada para a implementação da exclusão mútua múltipla estende a solução para o caso simples através da definição de uma fila adicional para os processadores liberando o semáforo.

Quando há transferências concorrentes do semáforo, o enfileiramento tanto dos processadores aguardando o semáforo como dos processadores liberando o semáforo decide os pares de processadores envolvidos nestas transferências.

8.3.1 Definições Básicas e Estruturas de Dados

Os algoritmos operam com as seguintes variáveis e constantes:

N — Número de alocações simultâneas do semáforo. Este valor é definido para cada semáforo pelo programa de usuário durante a criação do semáforo.

Doação — Fila de Liberação. Vetor para a implementação de uma fila circular para a doação do semáforo. Cada posição da fila pode ser ocupada pela identificação de um processador ou por uma das constantes OK e VAZIO. Há uma instância desta variável para cada semáforo.

Espera — Fila de Obtenção. Vetor para a implementação de uma fila circular para a espera pelo semáforo. Cada posição da fila pode ser ocupada pela identificação de um processador ou por uma das constantes OK e VAZIO. Há uma instância desta variável para cada semáforo.

d_atual — Posição ocupada pelo último processador a entrar na fila de liberação. Há uma instância desta variável para cada semáforo.

e_atual — Posição ocupada pelo último processador a entrar na fila de obtenção. Há uma instância desta variável para cada semáforo.

d — Posição ocupada por este processador na fila de liberação. Há uma instância desta variável para cada processador.

e — Posição ocupada por este processador na fila de obtenção. Há uma instância desta variável para cada processador.

local[p] — Área local para monitoração pelo processador p.

id — Identificação de um outro processador na fila.

ID — Constante identificando este processador.

OK — Constante que indica a liberação do semáforo.

VAZIO — Em uma posição da Fila de Obtenção esta constante indica que não há processador na posição aguardando a liberação do semáforo. Em uma posição da Fila de Liberação, esta constante indica que não há processador na posição liberando o semáforo. Na área local de um processador, esta constante indica que o semáforo ainda não foi liberado para o processador.

As operações POS(x), ANT(x) e SUC(x), assim como a instrução F&I, têm definição e implementação idênticas ao caso de exclusão mútua simples.

8.3.2 Algoritmos

Os algoritmos para a iniciação, obtenção e liberação de um semáforo múltiplo são explicados a seguir.

O algoritmo para a **iniciação** de um semáforo múltiplo prepara as filas de processadores, indicando a disponibilidade do semáforo para os primeiros N processadores que executarem o algoritmo para obtenção.

Um processador executando o algoritmo para **obtenção** de um semáforo múltiplo realiza as seguintes operações: (1) indica em sua área local que está aguardando a liberação do semáforo; (2) aloca uma posição na fila de obtenção; (3) se coloca nesta fila; (4) se o processador que estava na mesma posição da fila de liberação já liberou o semáforo, então o semáforo já é seu; em caso contrário, aguarda a liberação do semáforo monitorando continuamente a sua área local; (5) indica a obtenção do semáforo nas filas de obtenção e liberação.

```
    Iniciação de um Semáforo Múltiplo

para cada posição i de Espera          (1)
        Espera[i] := VAZIO
e_atual := 0

para cada posição i de Doação          (2)
        Doação[i] := VAZIO
d_atual := 0

para cada i  tal que  0 <= i < N       (3)
        Doação[i] := OK
```

Figura 8.4: Iniciação do Semáforo Múltiplo

```
    Obtenção de um Semáforo Múltiplo

local[ID] := VAZIO                          (1)

e := POS (F&I (e_atual))                    (2)
Espera[e] := ID                             (3)

se Doação[e] <> OK                          (4)
        espera (local[ID] = OK)

Espera[e] := VAZIO                          (5)
Doação[e] := VAZIO
```

Figura 8.5: Obtenção do Semáforo Múltiplo

O algoritmo para a liberação de um semáforo múltiplo realiza as seguintes operações: (1) aloca uma posição na fila de liberação, (2) avisa na fila de liberação que o semáforo está liberado, (3) se já há algum outro processador na mesma posição da fila de obtenção avisa também em sua área local.

8.4 Ordem Parcial

Os semáforos de ordem parcial são objeto de duas operações principais: (1) a operação de sinalização e (2) a operação de reconhecimento.

A operação de sinalização é implementada simplesmente através de uma contagem utilizando a instrução F&I até que seja alcançado o número de processadores definido para a sinalização do semáforo.

```
    Liberação de um Semáforo Múltiplo

d := POS (F&I (d_atual))                    (1)
Doação[d] := OK                             (2)
se Espera[d] = id                           (3)
        local[id] := OK
```

Figura 8.6: Liberação do Semáforo Múltiplo

A operação de reconhecimento é baseada no posicionamento dos processadores em uma árvore binária, de acordo com a ordem em que eles se tornam prontos para participar do reconhecimento. A sua implementação constitui uma extensão da concatenação das algoritmos de obtenção e liberação de semáforos de exclusão mútua simples.

8.4.1 Definições Básicas e Estruturas de Dados

Os algoritmos operam com as seguintes variáveis e constantes:

NS — Número de processadores que devem participar da sinalização para que esta seja considerada completa. Este valor é definido para cada semáforo pelo programa de usuário durante a criação do semáforo.

NR — Número de processadores que devem participar do reconhecimento para que este seja considerado completo. Este valor é definido para cada semáforo pelo programa de usuário durante a criação do semáforo.

Contagem — Contador de processadores que já participaram desta sinalização. Há uma instância desta variável para cada semáforo.

Sinalização — Variável que pode assumir dois valores: VAZIO, indicando que a sinalização ainda não está completa, e OK, indicando a sinalização completa. Há uma instância desta variável para cada semáforo.

Reconhecimento — Fila de Reconhecimento. Vetor para a implementação de uma fila circular para o reconhecimento do semáforo. Cada posição da fila pode ser ocupada pela identificação de um processador ou por uma das constantes OK e VAZIO. Há uma instância desta variável para cada semáforo.

r_atual — Posição ocupada pelo último processador a entrar na Fila de Reconhecimento. Há uma instância desta variável para cada semáforo.

r — Posição ocupada por este processador na Fila de Reconhecimento. Há uma instância desta variável para cada processador.

Primeiro — Identidade do primeiro processador pronto para a operação de reconhecimento.

local[p] — Área local para monitoração pelo processador p.

id — Identificação de um outro processador na fila.

ID — Constante identificando este processador.

OK — Constante que indica a sinalização completa do semáforo.

VAZIO — Em uma posição da Fila de Reconhecimento esta constante indica que não há processador na posição aguardando o reconhecimento do semáforo. Na área local de um processador, esta constante indica que o semáforo ainda não foi liberado para o processador.

A operação POS(x) e a instrução F&I têm definição e implementação idênticas ao caso de exclusão mútua simples.

As operações BIN_ANT(x), BIN_SUCP(x) e BIN_SUCI(x) retornam respectivamente as posições anterior e posteriores par e ímpar em uma árvore binária linearizada na Fila de Reconhecimento.

8.4.2 Algoritmos

Os algoritmos para a iniciação, sinalização e reconhecimento de um semáforo de ordem parcial são descritos a seguir.

```
            Iniciação do Semáforo
Contagem := 1                                    (1)
Sinalização := VAZIO                             (2)

para cada posição i de Reconhecimento            (3)
          Reconhecimento[i] := VAZIO
r_atual := 0
```

Figura 8.7: Iniciação do Semáforo

O algoritmo para a **iniciação** consiste dos seguintes passos: (1) iniciação do contador de processadores participando da sinalização, (2) indicação de que a sinalização ainda não está completa e (3) preparação da Fila de Reconhecimento.

O algoritmo para **sinalização** consiste dos seguintes passos: (1) incremento na contagem de processadores; se a contagem alcançou NS então (2) indicação da

```
     Sinalização do Semáforo
s := F&I (Contagem)                (1)

se s = NS
    Sinalização := OK              (2)
    se Primeiro = id               (3)
        local[id] := OK
```

Figura 8.8: Sinalização do Semáforo

sinalização completa e (3) se já existe um processador pronto para o reconhecimento, transmissão desta informação a este processador.

O algoritmo para o **reconhecimento** consiste de três etapas: (1) recepção da informação de sinalização completa, (2) transmissão desta informação através de uma árvore binária e (3) finalização que prepara as estruturas de dados para o próximo reconhecimento.

```
       Reconhecimento do Semáforo: Recepção
local[ID] := VAZIO                                    (1.1)
r := POS (F&I(r_atual))                               (1.2)
Reconhecimento[r] := ID
se  r = 0                                             (1.3)
       Primeiro := ID
se  r > 0 e Reconhecimento[BIN_ANT(r)] <> OK          (1.4)
 ou r = 0 e Sinalização <> OK
       espera (local[ID] = OK)
```

Figura 8.9: Reconhecimento do Semáforo: Recepção

A etapa de **recepção** consiste dos seguintes passos: (1.1) preparação para monitoração da área local; (1.2) posicionamento na Fila de Reconhecimento; (1.3) se este é o primeiro processador a participar deste reconhecimento, coloca a sua identidade vísivel para o algoritmo de sinalização; (1.4) se a sinalização ainda não está completa, aguarda esta informação monitorando continuamente a área local.

A etapa de **transmissão** consiste dos seguintes passos: (2.1) indicação de sinalização completa na posição deste processador na Fila de Reconhecimento; (2.2) indicação na área local do processador sucessor par; (2.3) indicação na área local do processador sucessor ímpar.

A etapa de finalização consiste dos seguintes passos: (3.1) se o processador foi o

```
      Reconhecimento do Semáforo: Transmissão
Reconhecimento[r] := OK                              (2.1)
se BIN_SUCP(r) < NR                                  (2.2)
    se Reconhecimento[BIN_SUCP(r)] = id
        local[id] := OK
se BIN_SUCI(r) < NR
    se Reconhecimento[BIN_SUCI(r)] = id              (2.3)
        local[id] := OK
```

Figura 8.10: Reconhecimento do Semáforo: Transmissão

```
      Reconhecimento do Semáforo: Finalização
se  r = 0                                            (3.1)
        Primeiro := VAZIO
se  r é ímpar                                        (3.2)
        Reconhecimento[BIN_ANT(r)] := VAZIO
se  BIN_SUCP(r) > NR                                 (3.3)
        Reconhecimento[r] := VAZIO
```

Figura 8.11: Reconhecimento do Semáforo: Finalização

primeiro a participar deste reconhecimento torna a sua identidade invísivel ao algoritmo de sinalização; (3.2) se o processador é um sucessor ímpar anula a transmissão de seu antecessor; (3.3) se o processador não tem sucessores anula a sua transmissão.

Parte IV

Conclusões

Capítulo 9

Conclusões

A motivação principal para este trabalho foi a criação de um ambiente mínimo para a programação paralela no projeto MULTIPLUS. Assim, esta tese define um modelo de computação paralela adequado à arquitetura MULTIPLUS e mostra como implementar este modelo em um sistema operacional tipo UNIX, como o sistema operacional PLURIX desenvolvido no NCE/UFRJ.

O modelo de computação paralela PRAM é atualmente o modelo mais importante para o estudo de algoritmos paralelos e, por isso, serviu de base para este trabalho. A constatação da inadequação do modelo PRAM à arquitetura MULTIPLUS, em razão dele tornar invisível para o programador o aspecto fundamental do custo da comunicação entre processadores, justificou a definição do modelo MULPLIX.

O modelo MULPLIX é uma derivação do modelo PRAM, que torna a comunicação entre processadores mais transparente para o programador. Um programa paralelo no modelo MULPLIX define explicitamente a sincronização e o compartilhamento de dados entre processadores, permitindo, assim, uma exploração mais efetiva de paralelismo, através da aplicação das técnicas de isolamento e localidade.

O sistema operacional MULPLIX é o resultado da evolução do sistema PLURIX com o objetivo de facilitar a programação paralela e de extrair o máximo de desempenho da arquitetura MULTIPLUS. A programação paralela é facilitada através da redefinição do conceito de processo e do provimento de primitivas que implementam o modelo MULPLIX a nível de sistema operacional. A implementação do sistema MULPLIX a partir do PLURIX significou alterações no núcleo do sistema em relação à escalação de processos, gerência de memória e sincronização resultantes tanto da implementação do modelo MULPLIX como da melhor adequação à arquitetura MULTIPLUS.

Apesar deste trabalho focalizar primordialmente a arquitetura MULTIPLUS, os seus resultados são aplicáveis a outras arquiteturas MIMD (*"Multiple Instruction, Multiple Data"*) com estrutura NUMA (*"Non-Uniform Memory Access"*). Esta classe de arquiteturas tem um papel muito importante para o desenvolvimento atual da computação paralela por causa dos seguintes fatores:

- O compartilhamento de memória permite uma maior flexibilidade para a implementação de uma extensa gama de modelos de programação paralela com alto nível de abstração [29].
- Uma estrutura hierárquica NUMA possibilita a expansão eficiente do sistema pelo menos até uma centena de processadores [16].
- O uso de processadores projetados essencialmente para execução seqüencial possibilita o aproveitamento de processadores com alto desempenho já disponíveis no mercado.
- A arquitetura privilegia o processamento local paralelo. Esta característica aparece freqüentemente em problemas científicos e de engenharia e pode ser eficientemente explorada na tecnologia de microeletrônica [31][39]. Assim, a pesquisa em processamento local paralelo, que é estimulada por estas arquiteturas, pode contribuir significativamente para o desenvolvimento de uma nova geração de computadores de alto desempenho, baseada na exploração intensa de paralelismo nos níveis mais baixos do hardware.

O projeto MULTIPLUS está em pleno desenvolvimento. O primeiro protótipo já se encontra em fase de construção física. A versão 1.0 do MULPLIX está sendo testada com a programação de alguns algoritmos paralelos. A preparação para o transporte do sistema para o protótipo do MULTIPLUS já se iniciou.

Em termos do desenvolvimento específico de software, as seguintes direções devem ser atacadas como continuação a este trabalho:

- Desenvolvimento de um sistema paralelo de arquivos para a melhor exploração pelo MULPLIX da arquitetura distribuída e paralela de Entrada e Saída do MULTIPLUS.
- Aperfeiçoamento do modelo MULPLIX, dotando-lhe, por exemplo, de uma metodologia para o cálculo da complexidade de algoritmos em termos de custo de processamento e de comunicação.
- Definição de linguagens de alto nível adequadas para a programação paralela eficiente na arquitetura MULTIPLUS.

- Desenvolvimento de técnicas de otimização para a compilação eficiente de linguagens seqüenciais ou paralelas no MULTIPLUS.
- Desenvolvimento de ferramentas para depuração, análise e avaliação do funcionamento de programas paralelos no MULTIPLUS.

Parte V

Anexos

Anexo A

Manual de Referência: Primitivas para Programação Paralela

Este anexo contém uma cópia do manual de referência das primitivas disponíveis no MULPLIX para a programação de aplicações paralelas.

```
THREAD (sys)        MULPLIX: Manual de Referência      THREAD (sys)


NOME
     Criação e término de threads.

     th_spawn - criação e execução de threads
     th_term  - término da execução da thread

SINTAXE
     #include <threads.h>

     void
     th_spawn (nthreads, func, arg, mapping, sync)
     int         nthreads;
     void        (*func) ();
     int         arg;
     int         *mapping;
     int         sync;

     void
     th_term ()

     void
     func (arg, order)
     int         arg;
     int         order;

DESCRIÇÃO
     A primitiva "th_spawn" comanda a execução concorrente de um
     grupo de threads. O argumento "nthreads" especifica o número
     de threads no grupo. O argumento "func" especifica uma função
     do programa por onde cada uma das threads do grupo iniciará a
     sua execução. O argumento "arg" é repassado para as threads do
     grupo. O argumento "mapping" é o endereço de um vetor com
     dimensão "nthreads" que estabelece para cada uma das threads
     no grupo o processador em que ela será preferencialmente
     executada. Se este argumento é um endereço nulo, então o
     processador preferencial para cada thread é estabelecido pelo
     núcleo. O argumento "sync" especifica se a chamada é síncrona
     (valor unitário) ou assíncrona (valor nulo).

     Cada thread inicia a sua execução por uma função declarada da
     mesma forma que "func". O primeiro argumento recebido é uma
     cópia do argumento enviado a todas as threads no grupo e por
     isto pode ser utilizado para identificar o grupo de threads. O
     segundo argumento indica a ordem desta thread dentro de seu
     grupo e assim pode identificar esta thread dentro de seu
     grupo.

     Quando a chamada "th_spawn" é síncrona, a thread que a executa
     é bloqueada até que todas as threads criadas na chamada


Atualizado em em 09.08.92    Versão 1.0                  Pág. 1
```