Primitivas para Programação Paralela no MULTIPLUS

Rafael Peixoto de Azevedo
Gustavo Peixoto de Azevedo
Júlio Tadeu Carvalho da Silveira
Júlio Salek Aude

Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Caixa Postal 2324, Rio de Janeiro - RJ, CEP 20001
Telefone: (021) 598 3160

Resumo

Este artigo apresenta uma definição do modelo básico de programação paralela que será provido pelo sistema operacional MULPLIX, para a família de computadores paralelos de alto desempenho MULTIPLUS.

O modelo representa simultaneamente uma extensão ao conceito de processo suportado pelos sistemas operacionais com filosofia UNIX – considerado um modelo bem sucedido de computação seqüencial – e uma versão mais realista do modelo PRAM, atualmente o principal modelo para estudo de algoritmos paralelos em memória compartilhada.

O texto explica os mecanismos definidos para execução paralela, alocação de memória e sincronização. A utilização destes mecanismos é exemplificada através de um programa paralelo para a resolução de sistemas lineares pelo método de Eliminação Gaussiana.

Abstract

This article presents a definition of the basic model for parallel programming that will be provided by the MULPLIX operating system, for the high performance parallel computer MULTIPLUS.

The model simultaneously represents an extension to the concept of a UNIX-like process – considered a popular model for sequential computing – and a more realistic version of the PRAM, currently the main model for the study of parallel algorithms on shared memory machines.

The text explains the mechanisms defined in order to provide parallel execution, memory allocation and synchronization. The use of these mechanisms is shown through a parallel program for the Gaussian Elimination.

Rafael Peixoto de Azevedo é Engenheiro Eletrônico pela UFRJ, M.Sc. pela COPPE/UFRJ no Programa de Engenharia de Sistemas de Computação e Pesquisador no NCE/UFRJ. Suas áreas de interesse são: Sistemas Operacionais, Arquiteturas Paralelas e Programação Paralela.

Gustavo Peixoto de Azevedo é Bacharel em Informática pela UFRJ, M.Sc. pela COPPE/UFRJ no Programa de Engenharia de Sistemas de Computação e Pesquisador no NCE/UFRJ. Suas áreas de interesse são: Sistemas Distribuídos Corporativos, Sistemas Operacionais, Arquiteturas Paralelas e Escalonamento Dinâmico Distribuído.

Júlio Tadeu Carvalho da Silveira é Bacharel em Informática pela UFRJ, Mestrando na COPPE/UFRJ no Programa de Engenharia de Sistemas de Computação e Pesquisador no NCE/UFRJ, Suas áreas de interesse são: Sistemas Operacionais, Complexidade de Algoritmos e Inteligência Artificial.

Júlio Salek Aude é Engenheiro Eletrônico pela UFRJ, M.Sc. pela COPPE/UFRJ no Programa de Engenharia de Sistemas de Computação, Ph.D. pela University of Manchester e Pesquisador no NCE/UFRJ, onde coordena o Projeto MULTIPLUS. Suas áreas de interesse são: Arquitetura de Computadores, Ferramentas de CAD, Sistemas Operacionais e Projeto de Circuitos VLSI.

Este trabalho conta com o apoio financeiro do **CNPq** e da **FINEP**.

1 Introdução

O projeto MULTIPLUS [7], atualmente em desenvolvimento no NCE/UFRJ, visa a concepção e construção de uma família 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.

A arquitetura MULTIPLUS é uma estrutura hierárquica baseada em *Nós de Processamento* (**NP**). Cada NP consiste de um processador seqüencial, uma memória cache e um módulo de memória compartilhada por todos os NP no sistema. Os NP são agrupados em **clusters** através da sua interligação por um **barramento**. Os clusters são interligados por uma **Rede de Interconexão** multiestágio do tipo n-cubo invertido.

O sistema operacional MULPLIX [8] está sendo projetado para atuar no MULTIPLUS. Ele é uma evolução do PLURIX [16] visando propiciar ao usuário um ambiente para desenvolvimento e execução de aplicações intensamente paralelas.

Este artigo apresenta as primitivas para programação paralela providas pelo MULPLIX. Estas primitivas constituem a interface dos programas aplicativos com o núcleo do sistema operacional, podendo ser utilizadas diretamente pelos usuários, através de bibliotecas de subrotinas, ou através de compiladores para linguagens de programação seqüenciais ou paralelas.

A utilização das primitivas é exemplificada através de um programa paralelo para a resolução de equações lineares simultâneas através do método de Eliminação Gaussiana.

2 O Modelo MULPLIX de Computação Paralela

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[1].

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:

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

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 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 Primitivas para Programação Paralela

Esta seção 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 conceito mais abrangente 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 [30].

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 facil'ta enormemente a comunicação entre as linhas de controle. Por exemplo: o acesso compart.lhado 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.

Esta seção obedece à seguinte organização: a ativação e o término de linhas de controle são enfocados na subseção 3.1; a alocação de memória é o tema da subseção 3.2; os mecanismos explícitos de sincronização e as atividades de sincronização são abordadas na subseção 3.3.

3.1 Criação de Linhas de Controle

Uma linha de controle, de acordo com o modelo MULPLIX de computação paralela (seção 2) é um conjunto de atividades de processamento programadas para execução seqüencial.

Do mesmo modo que os sistemas UNIX tradicionais, no MULPLIX um processo sempre inicia com apenas uma linha de controle, denominada **linha de controle original**. Em contraposição, no MULPLIX, uma linha de controle (por exemplo, a linha de controle original) pode pedir ao núcleo a criação de novas linhas de controle. O processo somente termina após o término de todas as suas linhas de controle.

Para o núcleo do sistema operacional MULPLIX, uma linha de controle está associada a um procedimento[2], que delimita a sua execução. No caso da linha de controle original, em programas escritos na linguagem C este procedimento é a função `main`. Considerando que a execução das diversas linhas de controle em um processo evolui independentemente, com a única exceção para as 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, denominada `th_spawn`, em que são especificadas, o número de linhas de controle a serem iniciadas (parâmetro `nthreads`), o procedimento associado às linhas de controle (parâmetro `func`) e um argumento comum para envio às linhas de controle (parâmetro `arg`). As linhas de controle iniciam através da chamada ao seu procedimento associado, (função `func`) que recebe dois argumentos: o argumento comum (argumento `arg`) e a ordem desta linha de controle no grupo (argumento `order`). 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 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** (parâmetro `mapping`). O estabelecimento de um NP preferencial para uma linha de controle indica para o núcleo do MULPLIX um processador

[2]O termo *"procedimento"* assume neste texto o seu significado usual: é uma função, subrotina ou subprograma que não define um valor de retorno.

```
#include        <threads.h>

int
th_spawn (
int     nthreads,       /* No. de Linhas de Controle */
void    (*func) (),     /* Procedimento a ser executado */
int     arg,            /* Argumento comum para o grupo */
int     *mapping)       /* Mapeamento nos Processadores */

void
th_term (void)

void
func (
int     arg,            /* Argumento comum para o grupo */
int     order)          /* Ordem no grupo */
```

Figura 1: Criação e Término de Linhas de Controle

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 subseção).

Uma linha de controle termina ao final da execução de seu procedimento associado ou através da chamada `th_term`.

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:

- 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.

3.2 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 das chamadas ao núcleo do MULPLIX denominadas `me_salloc` e `me_palloc`. Estas chamadas permitem a alocação explícita de incrementos para os segmentos de dados.

```
#include        <threads.h>

char *
me_salloc (
int     size,           /* Tamanho do incremento (seg. compartilhado) */
int     conc)           /* Alocação concentrada ? */

char *
me_palloc
int     size)           /* Tamanho do incremento (seg. privado) */
```

Figura 2: Alocação de Memória

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.

3.3 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.

3.3.1 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.

```
#include        <threads.h>

MUTEX
mx_create (
int     n)              /* No. de Alocações simultâneas */

void
mx_free (
MUTEX   mx)             /* Mutex a ser liberado */

void
mx_lock (
MUTEX   mx)             /* Mutex a ser alocado */

void
mx_delete (
MUTEX   mx)             /* Mutex a ser destruído */
```

Figura 3: Semáforos para Exclusão Mútua

3.3.2 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**.

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

```
#include        <threads.h>

MUTEX
ev_create (
int     nsignals,       /* No. de Sinalizações */
int     nwaits)         /* No. de Reconhecimentos */

void
ev_signal (
MUTEX   ev)             /* Event a ser sinalizado */

void
ev_wait (
MUTEX   ev)             /* Event a ser reconhecido */

void
ev_swait (
MUTEX   ev)             /* Event a ser sinalizado e reconhecido */

void
ev_delete (
MUTEX   ev)             /* Event a ser destruído */
```

Figura 4: Semáforos para Ordem Parcial

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.

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.

4 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 [31]. 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 [26]. Em função destas características, este problema tem servido extensamente como veículo para o estudo de programação paralela.

Esta seção apresenta uma solução 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 4.1 define o problema de resolução de um sistema de equações lineares simultâneas e faz uma revisão do Método da Eliminação Gaussiana. A seção 4.2 analisa as possibilidades de exploração de paralelismo existentes no método avaliando a eficiência destas possibilidades na arquitetura MULTIPLUS. A seção 4.3 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.

4.1 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:

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

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 (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 [15]. Este é o caso de interesse para esta seção.

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

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

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

$$Ax = b \tag{3}$$

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 nesta seção 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 [10].

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, $a^n_{ij} = 0 \ \forall i > j$. A equação (4) mostra o sistema linear triangular equivalente.

$$\begin{array}{rcl} a^n_{11}x_1 + a^n_{12}x_2 + \cdots + a^n_{1n}x_n &=& b^n_1 \\ a^n_{22}x_2 + \cdots + a^n_{2n}x_n &=& b^n_2 \\ & \vdots & \\ a^n_{nn}x_n &=& b^n_n \end{array} \tag{4}$$

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 (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).

$$m_{ik} = -\frac{a_{ik}^k}{a_{kk}^k}, \quad k < i < n \tag{5}$$

A equação (6) resume o passo k da Triangularização.

$$a_{ij}^{k+1} = a_{ij}^k + a_{kj}^k \times m_{ik}, \quad 1 \leq k < n, \; k < i < n, \; k \leq j < n \tag{6}$$

4.2 Análise das Possibilidades de Exploração de Paralelismo

Diversas possibilidades de exploração de paralelismo no método de Eliminação Gaussiana são analisadas a seguir. Inicialmente, uma análise matemática identifica os cálculos que podem ser realizados independentemente e assim caracteriza as possibilidades máximas de exploração de paralelismo. A seguir, são definidos critérios para a avaliação dos custos envolvidos em cada alternativa para programação paralela. Finalmente, estes critérios são utilizados para comparação entre diversas opções para a distribuição de dados e processamento entre linhas de controle no MULTIPLUS.

4.2.1 Cálculos Independentes

Uma análise da equações (5) e (6) revela que, em um passo k, o cálculo dos coeficientes a_{ij}^{k+1} 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.

4.2.2 Custos Envolvidos na Exploração de Paralelismo

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:

Gerência de Linhas de Controle — Quantidade de tempo e memória gastos para a criação e escalação das linhas de controle. Numa primeira aproximação, os custos de memória são proporcionais ao número de linhas de controle definidas, enquanto os custos relacionados à escalação aumentam com o desbalanceamento das necessidades de processamento entre as linhas de controle. Assim, nem sempre a exploração máxima de paralelismo através da definição de um grande número de linhas de controle é a forma mais eficiente de aproveitamento da arquitetura MULTIPLUS. A situação ideal seria a definição de uma linha de controle para cada NP disponível, com um perfeito balanceamento de carga entre estas linhas de controle.

Compartilhamento de Dados — Quantidade de dados cujo uso é compartilhado por linhas de controle distintas. A situação ideal é a distribuição dos dados pelas linhas de controle de modo a todos os cálculos realizados por cada linha de controle dependam apenas de dados no segmento de memória privada da linha de controle. No caso da Eliminação Gaussiana, o interesse é direcionado para a quantidade de coeficientes, oriundos de outras linhas de controle, 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. No caso da etapa de Triangularização da Eliminação Gaussiana, o mínimo de sincronização entre linhas de controle significa que as linhas de controle somente podem eliminar coeficientes em mais uma coluna após os coeficientes correspondentes nas colunas anteriores haverem sido eliminados.

4.2.3 Distribuição de Dados e de Processamento

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

Sem Agrupamento — Cada linha de controle é responsável por um único coeficiente; isto significa a criação de $n \times (n+1)$ linhas de controle. Esta opção representa a tentativa de explorar o máximo de paralelismo de processamento, baseada na constatação de que, a cada passo, os cálculos dos coeficientes dependem apenas de valores calculados no passo anterior.

O custo associado à gerência de linhas de controle e ao compartilhamento de dados torna esta opção proibitiva.

Agrupamento por Linhas ou Colunas — Cada linha de controle é responsável pelos coeficientes em uma linha ou coluna na matriz A. Isto significa a criação de n (linhas) ou $n+1$ (colunas) linhas de controle.

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.

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

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.

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 por submatrizes espalhadas alia um baixo custo de compartilhamento de dados com um bom balanceamento de carga e, por isso, é a solução adotada.

4.3 Uma Solução Adequada ao MULTIPLUS

As figuras a seguir mostram a programação da linha de controle original (figura 5) que comanda a execução paralela da Triangularização (figura 6) e da Substituição Regressiva (figura 7).

```
    Linha de Controle Original

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

spawn (P, triangularização, 0)

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

ev_wait (triângulo)

spawn (P, substituição, 0)

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

ev_wait (resolução[0])

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

Figura 5: Linha de Controle Original

A linha de controle original é responsável pela criação das linhas de controle para realizar a Triangularização (1ª etapa) e, a seguir, das linhas de controle para a Substituição Regressiva (2ª etapa). Além disto, a linha de controle cria todos os mecanismos de sincronização.

A constante P indica o número de processadores no sistema, que para simplificação é um quadrado perfeito da constante L, ou seja, $P = L^2$.

Cada linha de controle na Triangularização é responsável pelos coeficientes presentes em um conjunto de segmentos de linhas da matriz A. Estes segmentos são definidos de acordo com as seguintes características:

- Cada segmento contém exatamente o número de coeficientes (C) que podem ser armazenados em uma linha na memória cache dos NP.
- Os segmentos são dispostos verticalmente a distâncias periódicas de N/L linhas da matriz A e horizontalmente a distâncias periódicas de L × C colunas da matriz A.

A sincronização da Triangularização 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. Os semáforos `eliminação` e `triangular` indicam respectivamente o término de cada passo e o término total da Triangularização.

A distribuição de dados e processamento adotada para as linhas de controle correspondentes à Substituição Regressiva é exatamente a mesma adotada para a triangularização. O cálculo de uma raiz depende dos valores já obtidos para as raizes referentes às linhas de ordem mais lata na matriz.

```
            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)
        }
        ev_signal (triangular)
}
```

Figura 6: 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 7: Substituição Regressiva

5 Estágio Atual e Perspectivas Futuras

O sistema operacional MULPLIX, incluindo as primitivas para programação paralela descritas neste artigo, já se encontra operacional em um computador comercial monoprocessado.

O primeiro protótipo do MULTIPLUS já se encontra em fase final de montagem. O transporte do MULPLIX para este novo ambiente de hardware possibilitará através da experimentação avaliar e aperfeiçoar não somente as primitivas para programação paralela, como também alternativas de programação paralela de algoritmos como a Eliminação Gaussiana.

Referências

[1] A. Aggarwal, A. K. Chandra, M. Snir; *"Communication Complexity of PRAMs"*; IBM Research Report RC 14998, fevereiro de 1989.

[2] A. V. Aho, J. E. Hopcroft, J. D. Ullman; *"The Design and Analysis of Computer Algorithms"*; Addison-Wesley, Reading, Mass., 1974.

[3] A. V. Aho, R. Sethi, J. D. Ullman; *"Compilers, Principles, Techniques, and Tools"*; Addison-Wesley, Reading, Mass., 1986.

[4] B. Alpern, L. Carter, E. Feig; *"Uniform Memory Hierarchies"*; IBM Research Report RC 16069, agosto de 1990.

[5] T. E. Anderson; *"The Performance of Spin Lock Alternatives for Shared Memory Multiprocessors"*; IEEE Trans. on Parallel and Distributed Systems, January 1990, Vol 1, Number 1, pp. 6-16.

[6] J. S. Aude; *"Uma Proposta de Implementação do Algoritmo de Lee no Multiprocessador MULTIPLUS"*; pp. 555-568.

[7] J. S. Aude et al; *"MULTIPLUS: A Modular High-Performance Multiprocessor"*; Proceedings of the Euromicro 91, Microprocessing and Microprogramming Vol. 32 (1991).

[8] G. P. de Azevedo, R. P. de Azevedo, N. R. Figueira, J. S. Aude; "MULPLIX: *Um Sistema Operacional tipo* UNIX *para o Multiprocessador* MULTIPLUS"; Relatório Técnico RT 91-1, NCE/UFRJ, janeiro de 1991. *Este texto é uma versão revista e ampliada do trabalho homônimo apresentado no III SBAC-PP em novembro de 1990.*

[9] M. Bach; *"The Design of the UNIX Operating System"*; New Jersey, Prentice-Hall, 1986.

[10] J. L. Boldrini, S. I. Costa, V. L. Ribeiro, H. G. Wetzler; *"Álgebra Linear"*; Harper & Row do Brasil, São Paulo, 1980.

[11] P. Brinch Hansen; *"Concurrent Programming Concepts"*; ACM Computing Surveys, Vol. 5, No. 4, dezembro de 1973; pp. 223-245.

[12] G. Bronstein, A. J. O. Cruz, O. C. M. B. Duarte; *"Análise de Desempenho de Redes de Interconexão para Máquinas Paralelas"*; Anais do III SBAC-PP, Rio de Janeiro, novembro de 1990, pp. 345-360.

[13] D. R. Cheriton, H. A. Goosen, P. D. Boyle; *"ParaDiGM: A Highly Scalable Shared-Memory Multicomputer Architecture"*; IEEE Computer, Vol. 24, No. 2, fevereiro de 1991; pp. 33-46.

[14] R. Cole, O. Zajicek; *"The* APRAM*: Incorporating Asynchrony into the* PRAM *Model"*; Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, Santa Fe, NM, junho de 1989, pp. 169-178.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest; *"Introduction to Algorithms"*; MIT Press, Cambridge, Massachusetts, 1990.

[16] N. Faller et al; *"O Projeto Pegasus-32X/Plurix"*; Anais do XVII Congresso Nacional de Informática, 1984.

[17] N. Faller, P. Salenbauch; *"PLURIX, O Sistema Operacional Multiprocessador do NCE-UFRJ: (1) Sincronização de Processos"*; Data News, 24 de setembro de 1985.

[18] S. Fortune, J. Wyllie; *"Parallelism in Random Access Machines"*; Proceedings of the 10th ACM Symposium on Theory of Computing, 1978, pp.114-118.

[19] A. Gibbons, W. Rytter; *"Efficient Parallel Algorithms"*; Cambridge University Press, 1988.

[20] P. B. Gibbons; *"A More Practical* PRAM *Model"*; Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, Santa Fe, NM, junho de 1989, pp. 158-168.

[21] J. R. Goodman, M. K. Vernon, P. J. Woest; *"Efficient Synchronization Primitives for Large Scale Cache-Coherent Multiprocessors"*; Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 3 Boston, Mass., 3-6 de abril de 1989; pp. 64-75.

[22] R. M. Karp, V. Ramachandran; *"A Survey of Parallel Algorithms for Shared Memory Machines"*; In Handbook of Theoretical Computer Science, ed. J. van Leeuwen, North Holland, Amsterdam, 1989.

[23] T. J. LeBlanc, J. M. Mellor-Crummey, N. M. Gafter; *"The Elmwood Multiprocessor Operating System"*; Software – Practice & Experience V.19, No.110, novembro de 1989, pp. 1029-1055.

[24] T. J. LeBlanc, M. L. Scott, C. M. Brown; *"Large Scale Parallel Programming: Experience with the BBN Butterfly Parallel Processor"*; Proceedings of the ACM SIGPLAN PPEALS 1988 – Parallel Programming: Experience with Applications, Languages and Systems, julho de 1988, pp. 161-172.

[25] U. Manber; *"Introduction to Algorithms: A Creative Approach"*; Addison-Wesley, 1989.

[26] E. P. Markatos, T. J. LeBlanc; *"Shared-Memory Multiprocessor Trends and the Implications for Parallel Program Performance"*; Relatório Técnico 420, Computer Science Departament, The University of Rochester, maio de 1992.

[27] A. M. Meslim, A. Pacheco; *"Sistema de Memória Multicache para uma Máquina Paralela MIMD – Projeto MULTIPLUS"*; Anais do III SBAC-PP, Rio de Janeiro, novembro de 1990, pp. 179-193.

[28] S. C. Oliveira, J. S. Aude; *"O Sistema de Memória de Massa do Multiprocessador MULTIPLUS"*; Anais do III SBAC-PP, Rio de Janeiro, novembro 1990, pp. 298-313.

[29] D. A. Patterson, J. L. Hennessy; *"Computer Architecture: A Quantitative Approach"*; Morgan Kaufmann Publishers, Palo Alto, California, 1990.

[30] R. Rashid, A. Tevanian Jr., M. Young, D. Golub, R. Baron, D. Black, W. Bolosky, J. Chew; *"Machine Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures"*; IEEE Transactions on Computers, Vol. 37, No. 8, agosto de 1988; pp. 896-908.

[31] H. S. Stone; *"High-Performance Computer Architecture"*; Addison-Wesley, Reading, Massachusetts, 1987.

[32] H.-M. Su, P.-C. Yew; *"On Data Synchronization for Multiprocessors"*; ACM Computer Architecture News, Vol. 17, No. 3, junho de 1989; pp. 416-423.

[33] Sun Microsystems Inc.; *"The SPARC Architecture Manual"*; outubro de 1987.

[34] L. G. Valiant; *"Bulk-Synchronous Parallel Computers"*; Technical Report TR-08-89, Harvard University, Cambridge Mass., 1989.

[35] L. G. Valiant; *"General Purpose Parallel Architectures"*; In Handbook of Theoretical Computer Science, ed. J. van Leeuwen, North Holland, Amsterdam, 1989.