ALGORÍTMOS PARALELOS
(Aula 5)

Neyval C. Reis Jr.

OUTUBRO/2004

Laboratório de Computação de Alto Desempenho

DI/UFES

UFES UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Programa do Curso

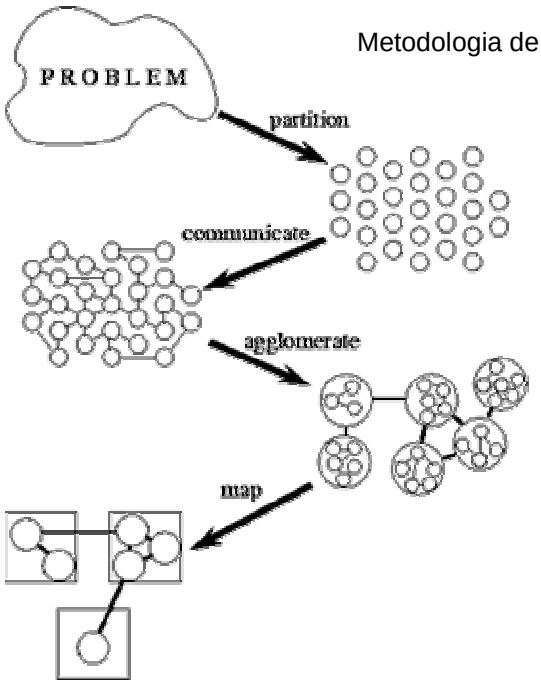
1. Introdução
2. Arquitetura de Computadores
3. Arquiteturas de Sistemas Paralelos
4. Computação de Alto Desempenho
5. Programação Paralela (modelos e paradigmas)
6. Análise de Desempenho e Instrumentação
7. Aplicações

UFES UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Programa do Curso

5. Programação Paralela (modelos e paradigmas)
   a) Começando a pensar em paralelo (exemplo)
   b) Metodologia de design
   c) Paradigmas de Programação
   d) Eficiência
   e) Ferramentas

Metodologia de design



Paralelismo de Dados

Paralelismo de Funcional

Tipos de Ambientes e Paradigmas de Programação

Troca de Mensagens (Message Passing): é o método de comunicação baseada no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre vários processadores que possuam memória própria. O programador é responsável pela sincronização das tarefas.

Paralelismo de Dados (Data Parallel): é a técnica de paralelismo de dados, normalmente automática ou semi-automática, ou seja, é o método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

Memória Distribuída e Compartilhada (Distributed-Shared Memory): Emula uma máquina de memória compartilhada em uma máquina de memória distribuída. O método que se encarrega de efetuar toda a comunicação necessária entre os processos de forma que o programador não necessita entender os métodos de comunicação.

O que é o Modelo Message Passing?

O modelo de Message Passing é um conjunto de processos que possuem acesso à memória local. As informações são enviadas da memória local do processo para a memória local do processo remoto.
A comunicação dos processos é baseada no envio e recebimento de mensagens.

A transferência dos dados entre os processos requer operações de cooperação entre cada processo de forma que operação de envio deve casar com uma operação de recebimento.

O modelo computacional Message Passing não inclui sintaxe de linguagem nem biblioteca e é completamente independente do hardware.

O que é o Modelo Message Passing?

Pontos fortes como:

1. Generalidade: Pode-se construir um mecanismo de passagem de mensagens para qualquer linguagem, como ferramentas de extensão das mesmas (bibliotecas).
2. Adequação à ambientes distribuídos.

Limitações:

1. Primeiro de tudo, o programador é diretamente responsável pela paralelização
2. Custos de comunicação podem tornar extremamente proibitiva a transmissão de mensagens em um dado ambiente.

Exemplos de ambientes de passagem de mensagens

P4

O sistema P4 desenvolvido no *Argonne National Laboratory* foi a primeira tentativa de desenvolvimento de uma plataforma portátil. Seu projeto começou a ser elaborado em 1984 (chamado na época de Monmacs) e a partir da sua terceira geração (em 1989) ele foi reescrito com o objetivo de produzir um sistema mais robusto para as necessidades recentes de passagem de mensagens entre máquinas heterogêneas.

O ambiente P4 é uma biblioteca de macros projetada para expressar uma grande variedade de algoritmos paralelos portáteis, com eficiência e simplicidade. Para atingir a eficiência, cada implementação do P4 é voltada para conseguir o melhor desempenho da plataforma escolhida. A simplicidade é obtida com um número pequeno de conceitos, suficientes entretanto, para permitir o desenvolvimento dos algoritmos paralelos projetados [BUT94].

Dois paradigmas, o SPMD e o mestre-escravo podem ser utilizados no ambiente P4. O SPMD (*Single Program - Multiple Data*) consiste de um único programa sendo executado em todos os processadores de uma máquina MIMD, mas com diferentes dados e com possíveis execuções diferentes em cada processador. O modelo mestre-escravo possui um processo mestre que é chamado pelo usuário e é o responsável por iniciar os outros processos (que são diferentes), denominados escravos.

Exemplos de ambientes de passagem de mensagens

Parmacs

O Parmacs (*PARallel MACroS*) é uma biblioteca de comunicação portátil, a qual tem sido implementada na maioria das máquinas com arquitetura MIMD, em máquinas com MPP e em redes de estações de trabalho. Ele herdou muitas características do P4 e começou a ser desenvolvido em 1987 no *Argonne National Laboratory.*

Seu modelo de programação é baseado em memória local, ou seja, processos paralelos podem ter acesso somente ao seu espaço de endereçamento. Não existem variáveis globais compartilhadas [CAL94].

A comunicação entre os processos é feita pela troca de mensagens as quais são seções de memória contínuas caracterizadas pelo seu endereço inicial e comprimento. Uma mensagem pode ser enviada para outro processo especificando o processo destino e a mensagem que se deseja enviar, sendo que a recepção pode também obter o identificador do processo que enviou.

A comunicação pode ser síncrona ou assíncrona. No primeiro modo, o processo que enviou e o que recebeu a mensagem realizam um *rendezvous*. No segundo caso, o processo que enviou a mensagem continua a sua execução tão logo a mensagem tenha sido copiada para o *buffer* do usuário (origem) e esteja apta para a transmissão.

Exemplos de ambientes de passagem de mensagens

Express

O sistema Express é um produto da empresa Parasoft, desenvolvido para ser um ambiente de passagem de mensagens atuando sobre várias arquiteturas MIMD com memória distribuída [ALM94], visando obter o máximo de desempenho de cada plataforma utilizada [FLO94].

Com a sua evolução, o Express passou a ser considerado também como um conjunto de ferramentas para desenvolvimento de software paralelo [FLO94]. A sua interface com o usuário está sendo melhorada, procurando-se esconder a maioria de detalhes internos. Isso levou ao desenvolvimento de mapeamentos e bibliotecas de comunicação para os diferentes tipos de topologias empregadas, para se obter o melhor desempenho possível em cada uma.

Uma das características principais do Express é abordar problemas como o balanceamento dinâmico de carga e de desempenho em I/O paralelos. Esses dois fatores são quase totalmente ignorados pela maioria dos ambientes de passagem de mensagens.

Exemplos de ambientes de passagem de mensagens

PVM

O PVM (*Parallel Virtual Machine*) é um conjunto integrado de bibliotecas e de ferramentas de software, cuja finalidade é emular um sistema computacional concorrente heterogêneo, flexível e de propósito geral [BEG94].

Diferente de outros ambientes portáteis desenvolvidos inicialmente para máquinas com multiprocessadores (como o P4, Express e outros), o PVM nasceu com o objetivo de permitir que um grupo de computadores interconectados, possivelmente com diferentes arquiteturas, possa trabalhar cooperativamente formando uma máquina paralela virtual [GEI94].

O projeto PVM teve início em 1989 no *Oak Ridge National Laboratory* - ORNL. A versão 1.0 (protótipo), foi implementada por *Vaidy Sunderam* e *Al Geist* (ORNL) sendo direcionada ao uso em laboratório. Depois de várias revisões (PVM 2.1 - 2.4), o PVM foi completamente reescrito, gerando a versão 3.0 (em fevereiro de 1993).

Exemplos de ambientes de passagem de mensagens

PVM

Várias mudanças foram feitas na versão 3.0, com objetivo de retirar erros de programação e ajustar pequenos detalhes como oferecer interface com o usuário melhor e aumentar o desempenho de certas comunicações (como em multiprocessadores). A versão disponível mais recente é o PVM 3.3, sendo a versão discutida neste trabalho. Já foram realizadas várias revisões nessa versão tendo como objetivo a retirada de erros e a inclusão de novas arquiteturas.

O sistema PVM permite que sejam escritas aplicações nas linguagens Fortran, C e C++. A escolha por esse conjunto de linguagens deve-se ao fato de que a maioria das aplicações passíveis de paralelização estão escritas nessas linguagens.

Atualmente o PVM está disponível para uma grande variedade de plataformas. A tabela abaixo mostra alguns equipamentos que podem ser utilizados pela versão 3.3.

Exemplos de ambientes de passagem de mensagens

MPI

Diversas plataformas portáteis atuais apresentam, ainda, alguns dos problemas identificados nos ambientes de passagem de mensagens propostos inicialmente, desenvolvidos por fabricantes para execução apenas nas suas máquinas. A maioria das plataformas portáteis existentes possui apenas um subconjunto das características necessárias para os mais diversos equipamentos fabricados, dificultando a tarefa de implementação em diferentes máquinas.

Devido a esses problemas foi criado o Comitê MPI (em 1992), para definir um padrão para os ambientes de passagem de mensagens denominado MPI (*Message Passing* Interface). Os principais objetivos são a padronização para a maioria dos fabricantes de hardware e plataformas portáteis, e eficiência.

O comitê MPI reúne membros de aproximadamente 40 instituições e inclui quase todos os fabricantes de máquinas com MPP, universidades e laboratórios governamentais pertencentes à comunidade envolvida na computação paralela mundial [MCB94] [WAL94].

Exemplos de ambientes de passagem de mensagens

MPI

O MPI é um padrão de interface de passagem de mensagens para aplicações que utilizam computadores MIMD com memória distribuída. Ele não oferece nenhum suporte para tolerância a falhas e assume a existência de comunicações confiáveis.

O MPI não é um ambiente completo para programação concorrente, visto que ele não implementa: I/O paralelos, depuração de programas concorrentes, canais virtuais para comunicação e outras características próprias de tais ambientes [WAL94].

Modelo Message Passing - Implementações

As duas principais bibliotecas de Message Passing da atualidade são:

PVM: (Parallel Virtual Machine) - Possui como característica, o conceito de "máquina virtual paralela", dentro da qual processadores recebem e enviam mensagens, com finalidades de obter um processamento global.

MPI - " Message Passing Interface", que está se tornando um padrão na indústria.

As principais implementações MPI são:

IBM MPI: Implementação IBM para SP e clusters

MPICH: Argonne National Laboratory/Mississipi State University

LAM: Ohio Supercomputer Center

PVM versus MPI

- **Portabilidade versus Interoperabilidade:**

PVM	MPI
1. Nível de Interoperabilidade: 2. Além da portabilidade (como o MPI), os programas permitem nível de interoperabilidade, tornando possível execuções em arquiteturas distintas quaisquer modificações. 3. PVM não somente suporta o nível de portabilidade, mas o expande para o nível de interoperabilidade: um mesmo programa pode estar interagindo com outro gerado para uma arquitetura completamente diferente.	1. Nível de Portabilidade: 2. Programas escritos para uma arquitetura podem ser compilados para uma outra arquitetura e executados sem quaisquer modificações.

MPI

MPI Forum

O padrão para Message Passing, denominado MPI (Message Passing Interface), foi projetado em um forum aberto constituído de pesquisadores, acadêmicos, programadores, usuários e vendedores, representando 40 organizações ao todo.

Itens discutidos e definidos no MPI Forum:

- Sintaxe
- Semântica
- Conjunto de rotinas padronizadas para Message Passing.

MPI Forum

A aceitação e utilização do padrão MPI deve-se a colaboração dos seus membros:

- Representantes da Convex, Cray, IBM, Intel, Meiko, nCUBE, NEC e Thinking Machines;
- Membros de grupos de desenvolvedores de softwares, tais como, PVM, P4, Zipcode, Chamaleon, PARMACS, TCGMSG e Express
- Especialistas da área de Processamento Paralelo.

A documentação do MPI foi apresentada em Maio de 1994 (versão 1.0) e atualizada em Junho de 1995 (versão 1.1). O documento que define o padrão "MPI : A Message-Passing Standard" foi publicado pela Universidade de Tennesee e encontra-se disponível via WWW na home-page do Laboratório Nacional de Argonne.

http://www.mcs.anl.gov/mpi/

Padrão MPI - Características

Eficiência: Foi cuidadosamente projetado para executar eficientemente em máquinas diferentes. Especifica somente o funcionamento lógico das operações. Deixa em aberto a implementação. Os desenvolvedores otimizam o código usando características específicas de cada máquina.

Facilidade: Define uma interface não muito diferente dos padrões PVM, NX, Express, P4, etc. e acrescenta algumas extensões que permitem maior flexibilidade.

Portabilidade: É compatível para sistemas de memória distribuída, NOWs (network of workstations) e uma combinação deles.

Padrão MPI - Características

Transparência: Permite que um programa seja executado em sistemas heterogêneos sem mudanças significativas.

Segurança: Provê uma interface de comunicação confiável. O usuário não precisa preocupar-se com falhas na comunicação.

Escalabilidade: O MPI suporta escalabilidade sob diversas formas, por exemplo: uma aplicação pode criar subgrupos de processos que permitem operações de comunicação coletiva para melhorar o alcance dos processos.

Rotinas de Message Passing

As bibliotecas de Message Passing possuem rotinas com finalidades bem específicas, como:

- Rotinas de gerência de processos :
  - Inicializar e finalizar processos
  - Determinar número de processos
  - Identificar processos

Rotinas de comunicação Ponto a Ponto onde a comunicação é feita entre dois processos:

- Comunicação síncrona ou assíncrona
- Bloqueante ou não-bloqueante
- Empacotamento de dados

Rotinas de comunicação de grupos:

- Broadcast
- Sincronização de processos (barreiras)

Biblioteca MPI

MPI é uma biblioteca de Message Passing desenvolvida para ambientes de memória distribuída, máquinas paralelas massivas, NOWs (network of workstations) e redes heterogêneas.

MPI define um conjunto de rotinas para facilitar a comunicação (troca de dados e sincronização) entre processos paralelos.

A biblioteca MPI é portável para qualquer arquitetura, tem aproximadamente 125 funções para programação e ferramentas para se analisar a performance.

A biblioteca MPI possui rotinas para programas em Fortran 77 e ANSI C, portanto pode ser usada também para Fortran 90 e C++. Os programas são compilados e linkados a biblioteca MPI.

Todo paralelismo é explícito, ou seja, o programador é responsável por identificar o paralelismo e implementar o algoritmo utilizando chamadas aos comandos da biblioteca MPI.

MPI

MPI-Rotinas Básicas: O MPI básico contém 6 funções básicas indispensáveis para o uso do programador, permitindo escrever um vasto número de programas em MPI.

MPI-Rotinas Avançadas: O MPI avançado contém cerca de 125 funções adicionais que acrescentam às funções básicas a flexibilidade (permitindo tipos diferentes de dados), robustez (modo de comunicação non-blocking), eficiência (modo ready), modularidade através de grupos e comunicadores (group e communicator) e conveniência (comunicações coletivas, topologias).

As funções avançadas fornecem uma funcionalidade que pode ser ignorada até serem necessárias.

Conceitos Básicos MPI

Processo: Por definição, cada programa em execução constitui um processo. Considerando um ambiente multiprocessado, podemos ter processos em inúmeros processadores.

Mensagem: É o conteúdo de uma comunicação, formado de duas partes:

Envelope: Endereço (origem ou destino) e rota dos dados. O envelope é composto de três parâmetros:

- Identificação dos processos (transmissor e receptor);
- Rótulo da mensagem;
- Comunicator.

Dado: Informação que se deseja enviar ou receber. É representado por três argumentos:

- Endereço onde o dado se localiza;
- Número de elementos do dado na mensagem;
- Tipo do dado.

Tipos do dados MPI

Tipos de Dados Básicos no C		Tipos de Dados Básicos no Fortran	
Definição no MPI	Definição no C	Definição no MPI	Definição no Fortran
MPI_CHAR	signed char	MPI_CHARACTER	CHARACTER(1)
MPI_INT	signed int	MPI_INTEGER	INTEGER
MPI_FLOAT	float	MPI_REAL	REAL
MPI_DOUBLE	Double	MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_UNSIGNED_SHORT	Unsigned short int	MPI_LOGICAL	LOGICAL
-	-	MPI_COMPLEX	COMPLEX
MPI_BYTE	-	MPI_BYTE	-
MPI_PACKED	-	MPI_PACKED	-

Conceitos Básicos MPI

Rank: Todo o processo tem uma identificação única atribuída pelo sistema quando o processo é inicializado. Essa identificação é contínua representada por um número inteiro, começando de zero até N-1, onde N é o número de processos.
Rank é o identificador único do processo, utilizado para identificar o processo no envio (send) ou recebimento (receive) de uma mensagem.

Group: Group é um conjunto ordenado de N processos. Todo e qualquer group é associado a um communicator muitas vezes já predefinido como "MPI_COMM_WORLD". Inicialmente, todos os processos são membros de um group com um communicator.

Communicator: O communicator é um objeto local que representa o domínio (contexto) de uma comunicação (conjunto de processos que podem ser contactados). O MPI utiliza esta combinação de grupo e contexto para garantir segurança na comunicação e evitar problemas no envio de mensagens entre os processos.
O MPI_COMM_WORLD é o comunicador predefinido que inclui todos os processos definidos pelo usuário numa aplicação MPI.

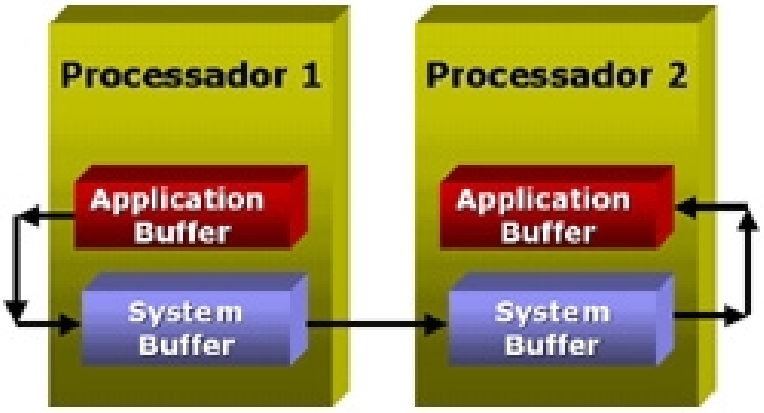
Conceitos Básicos MPI

Application Buffer: É o endereço de memória, gerenciado pela aplicação, que armazena um dado que o processo necessita enviar ou receber.

System Buffer: É um endereço de memória reservado pelo sistema para armazenar mensagens.

Dependendo do tipo de operação de envio/recebimento (send/receive), o dado no buffer da aplicação (application buffer) pode necessitar ser copiado de/para o buffer do sistema (system buffer) através das rotinas send buffer/receive buffer. Neste caso a comunicação é assíncrona.

Conceitos Básicos MPI



Conceitos Básicos MPI

Blocking Comunnication: Em uma rotina de comunicação blocking, a finalização da chamada depende de certos eventos.
Em uma rotina de envio de mensagem, o dado tem que ter sido enviado com sucesso, ou ter sido salvo no buffer do sistema (system buffer), indicando que o endereço do buffer da aplicação (application buffer) pode ser reutilizado.
Em uma rotina de recebimento de mensagem, o dado tem que ser armazenado no buffer do sistema (system buffer), indicando que o dado pode ser utilizado.

Non-Blocking Comunnication: Em uma rotina de comunicação non-blocking, a finalização da chamada não espera qualquer evento que indique o fim ou sucesso da rotina.
As rotinas non-blocking comunnication não esperam pela cópia de mensagens do buffer da aplicação (application buffer) para o buffer do sistema (system buffer), ou a indicação do recebimento de uma mensagem.

Conceitos Básicos MPI

Overhead: Overhead é um desperdício de tempo que ocorre durante a execução dos processos. Os fatores que levam ao overhead são: comunicação, tempo em que a máquina fica ociosa (tempo idle) e computação extra.
Existem os seguintes tipos de overhead:

System Overhead: É o tempo gasto pelo sistema na transferência dos dados para o processo destino.

Synchronization Overhead: É o tempo gasto para a sincronização dos processos.

Observação:

O MPI sempre apresenta um overhead de sicronização inicial, quando espera até que todas as máquinas estejam disponíveis para iniciar o processamento e inicializar os processos. O mesmo ocorre quando da conclusão: existe um delay até que todos os processos possam encerrar adequadamente e terminarem a execução.

Tipos de Comunicação MPI

Existem os seguintes tipos de comunicação entre os processos:

Ponto a Ponto

Dentre as rotinas básicas do MPI estão as rotinas de comunicação ponto a ponto que executam a transferência de dados entre dois processos.

Coletiva

É a comunicação padrão que invoca todos os processos em um grupo (group) - coleção de processos que podem se comunicar entre si. Normalmente a comunicação coletiva envolve mais de dois processos. As rotinas de comunicação coletiva são voltadas para a comunicação / coordenação de grupos de processos.

Tipos de Comunicação MPI

O MPI define quatro formas de se enviar uma mensagem ponto a ponto. Quem deve decidir de qual de forma a comunicação será feita é o programador. As formas são:

Send: Pode ser implementado como Ssend ou o Bsend, cabendo ao MPI decidir como será feita a comunicação;

Ssend: Forma síncrona, ao enviar a mensagem o processo remetente permanece bloqueado até que o receptor confirme que recebeu a mensagem, assim existe a garantia de que o processo recebeu a mensagem. O seu uso pode diminuir consideravelmente a performance do programa;

Tipos de Comunicação MPI

Bsend: Forma bufferizada, antes de enviar a mensagem o processo remetente deve alocar um espaço de memória do tamanho da mensagem vezes o número de mensagem que podem ser armazenadas no buffer para armazenar as mensagens (ver exemplos). As mensagens permanecerão em uma fila FIFO (fist in first out) até que o processo receptor remova as mensagens;

Rsend: Forma de envio com maior complexidade pois assume que o processo que irá receber a mensagem já está esperando. A mensagem é enviada logo que a instrução é executada.

Tipos de Comunicação MPI

Cada uma das quatro formas de envio de mensagem ponto a ponto possui uma versão bloqueante e outra não-bloqueante. Uma instrução que possua a característica bloqueante, significa que ela tem a capacidade de suspender a execução do processo enquanto ela não for concluída. Assim a principal diferença entre as formas bloqueantes e as não-bloqueantes (a instrução inicia-se com a letra maiúscula “i”), para a criação de programas utilizando MPI, é o momento em que a instrução é completada.

Tipos de Comunicação MPI

As instruções são completadas quando:

Bsend e **Ibsend**: A instrução é completada só depois que a mensagem é colocada no buffer. Caso não exista espaço no buffer acontece um erro;

Ssend: A instrução é completada somente depois que o destinatário avisar que a mensagem foi recebida;

Issend: A instrução é completada quando o destinatário começar a receber a mensagem;

Send: Depende se foi utilizado o Bsend ou Ssend;

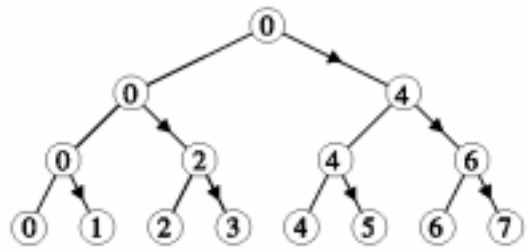
Isend: Uma thread auxiliar é criada para o envio da mensagem. A instrução é completada logo que a mensagem começa a ser enviada pela thread auxiliar;

Rsend e **Irsend**: A instrução é completada logo depois do envio da mensagem.

Tipos de Comunicação MPI

Comunicação Coletiva

A comunicação coletiva acontece quando um processo deseja enviar um conjunto de informações para um conjunto de processos que pode ou não incluir o processo que está enviando. Esta instrução não deve ser avaliada como tendo o tempo de execução *O(1)*, mas como tendo o tempo de execução *O(lg(p))*, onde lg é logaritmo na base dois e p é o número de processos, pois a forma como os processos se comunicam descreve uma árvore binária.



Tipos de Comunicação MPI

Existem os seguintes tipos de comunicação coletiva:

Broadcast (Difusão)

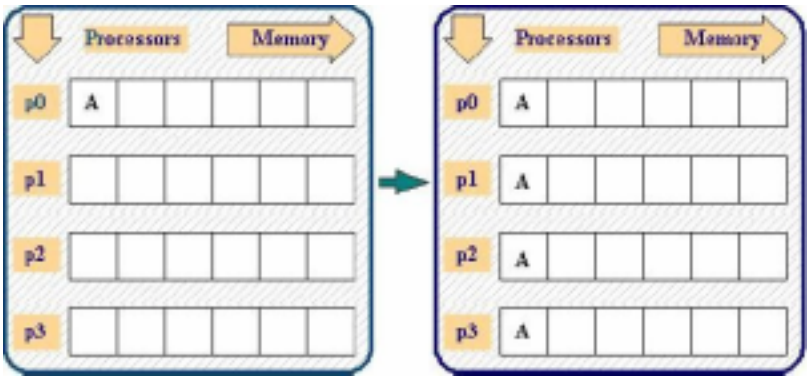
Reduction (Redução)

Gather (Coleta)

Scatter (Espalhamento)

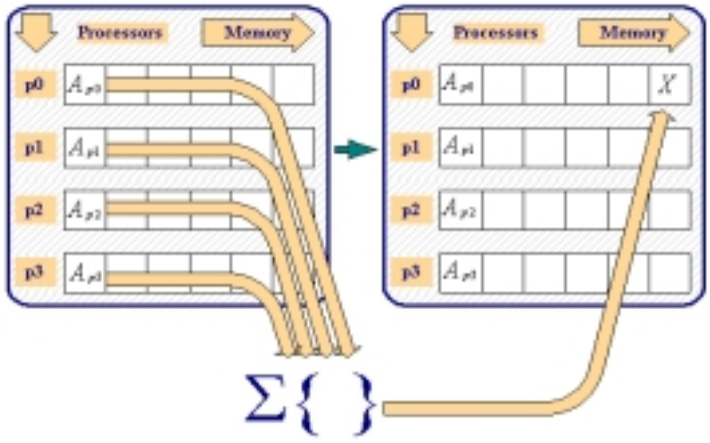
Tipos de Comunicação MPI

Broadcast (Difusão): É a comunicação coletiva em que um único processo envia (send)os mesmos dados para todos os processos com o mesmo communicator.



Tipos de Comunicação MPI

Reduction (Redução): É a comunicação coletiva onde cada processo no communicator contém um operador, e todos eles são combinados usando um operador binário que será aplicado sucessivamente.



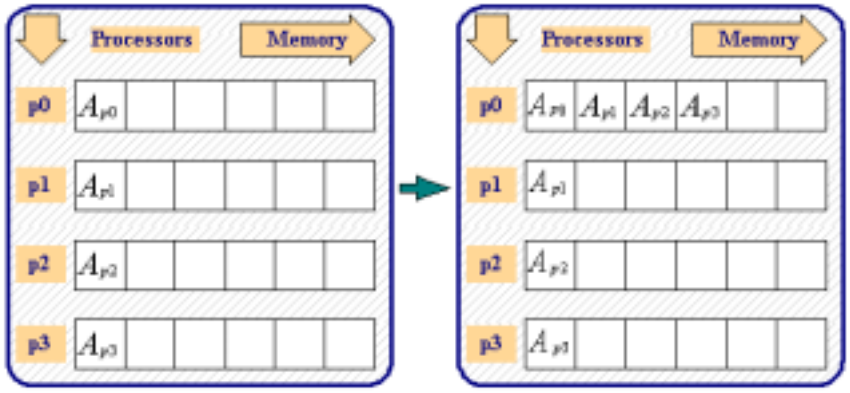
Tipos de Comunicação MPI

As operações possíveis são listadas na tabela abaixo:

Função	Significado	C	Fortran
MPI_MAX	Valor máximo	Int, float	integer, real, complex
MPI_MIN	Valor míximo	Int, float	integer, real, complex
MPI_SUM	Somatório dos valores	Int, float	integer, real, complex
MPI_PROD	Produtório dos valores	Int, float	integer, real, complex
MPI_LAND	E (and) lógico	Int	Boolean
MPI_BAND	E (and) lógico a nível de BIT	Int	Boolean
MPI_LOR	OU (or) lógico	Int	Boolean
MPI_BOR	OU (or) lógico a nível de BIT	Int	Boolean
MPI_LXOR	OU-EXCLUSIVO (xor) lógico	Int	Boolean
MPI_BXOR	OU-EXCLUSIVO (xor) lógico a nível de BIT	Int	Boolean
MPI_MAXLOC	Valor máximo de maior índice	Int, float	integer, real, complex
MPI_MINLOC	Valor mínimo de menor índice	Int, float	integer, real, complex

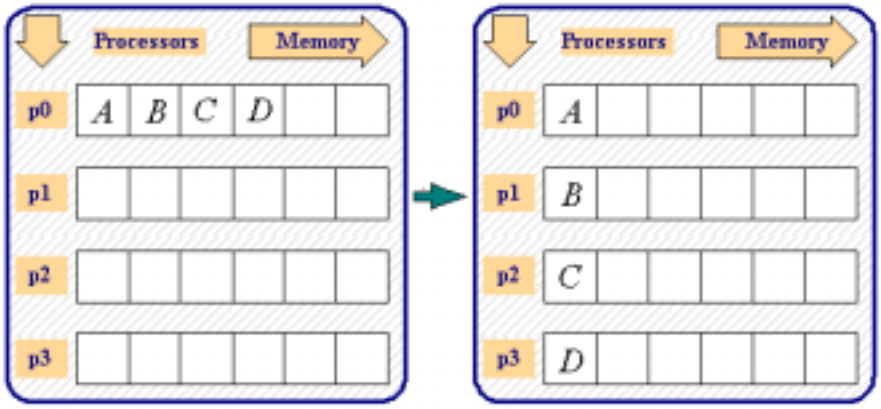
Tipos de Comunicação MPI

Gather (Coleta): A estrutura dos dados distribuídos é coletada por um único processo.



Tipos de Comunicação MPI

Scatter (Espalhamento):A estrutura dos dados que está armazenado em um único processo é distribuído a todos os processos.



UTILIZANDO A BIBLIOTECA MPI

Preparação do ambiente

Antes de rodar qualquer programa MPI, é necessário que o ambiente paralelo esteja preparado para tal. Segue abaixo os dois comandos de preparação do ambiente mais utilizados no LAM/MPI:

lamboot: esse comando monta a topologia da rede de processamento paralelo, inicializando os nós.

lamhalt: esse comando desfaz a topologia da rede, finalizando o daemon do LAM/MPI e liberando recursos da máquina.

É importante lembrar que esses comandos não fazem parte da especificação MPI e são apenas utilitários fornecidos pela biblioteca LAM/MPI. No próximo tópico, será mostrado um exemplo de utilização desses comandos.

UTILIZANDO A BIBLIOTECA MPI

Um programa simples

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char **argv)
{
    int rank, nproc;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    printf("Msg do processo de rank %d (total de %d)\n", rank, nproc);
    MPI_Finalize();

    return 0;
}
```

UTILIZANDO A BIBLIOTECA MPI (LAM-MPI)

Compile-o utilizando a seguinte linha de comando:

```
$ mpicc -o exemplo exemplo.c
```

Agora será necessário inicializar o ambiente com o comando lamboot:

```
$ lamboot -v [arquivo_hosts]
```

O parâmetro *-v* serve apenas para que o resultado do comando seja sempre exibido. Embora não tenha sido utilizado no exemplo acima, o comando lamboot aceita também como parâmetro um arquivo que contém o nome das máquinas que serão utilizadas para o processamento. Caso esse parâmetro não seja informado, o MPI simulará a rede de processamento criando processos na máquina onde o lamboot for executado.

Agora que a rede foi inicializada, podemos executar o nosso programa com a seguinte linha de comando:

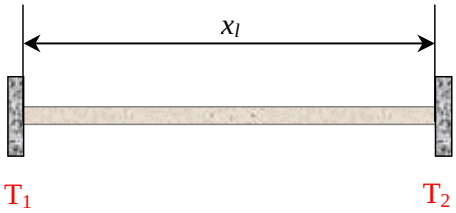
```
$ mpirun -np 4 exemplo
```

UTILIZANDO A BIBLIOTECA MPI

O resultado do comando acima deverá ser algo como:

```
Msg do processo de rank 0 (total de 4)
Msg do processo de rank 2 (total de 4)
Msg do processo de rank 1 (total de 4)
Msg do processo de rank 3 (total de 4)
```

Exemplo MPI



O problema selecionado é uma caso de condução de calor unidimensional em uma barra de comprimento de 1 metro, com propriedades homogêneas, e temperaturas constantes em cada extremidade (T_1=100 ºC e T_2=10º C), conforme apresentado na Fig. acima. É possível escrever a equação governante deste problema como:

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+S=0$$

onde k é a o condutividade térmica, T é a temperatura e S é a taxa de geração de calor por unidade de volume.

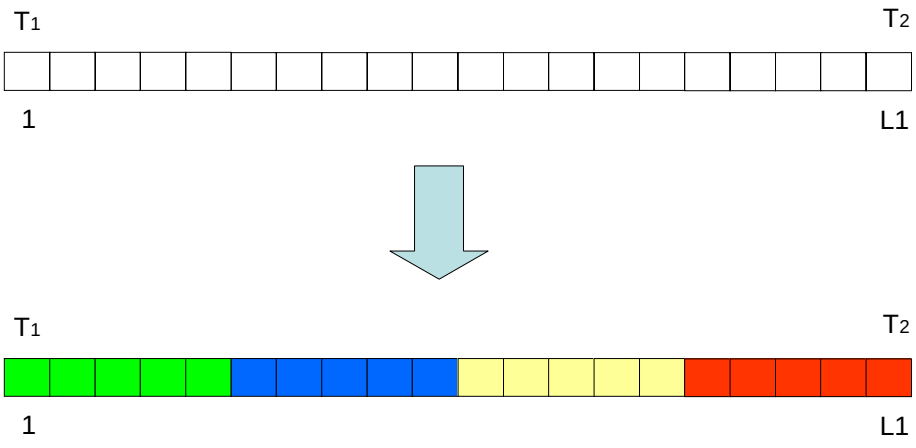
O algoritmo para solução do problema acima foi baseado no método de diferenças finitas e implementado utilizando Fortran 77.

```
! Leitura dos dados de entrada - condutividade térmica (k), condições de contorno (T1 e T2),
! termo de fonte (S), comprimento do domínio computacional (XL), número de pontos do grid (n);
! Calcula coeficientes independentes AE, AW, AP, b e inicializa matriz de temperatura.
Do while (not stop)
      Do i =2, (n-1)
```

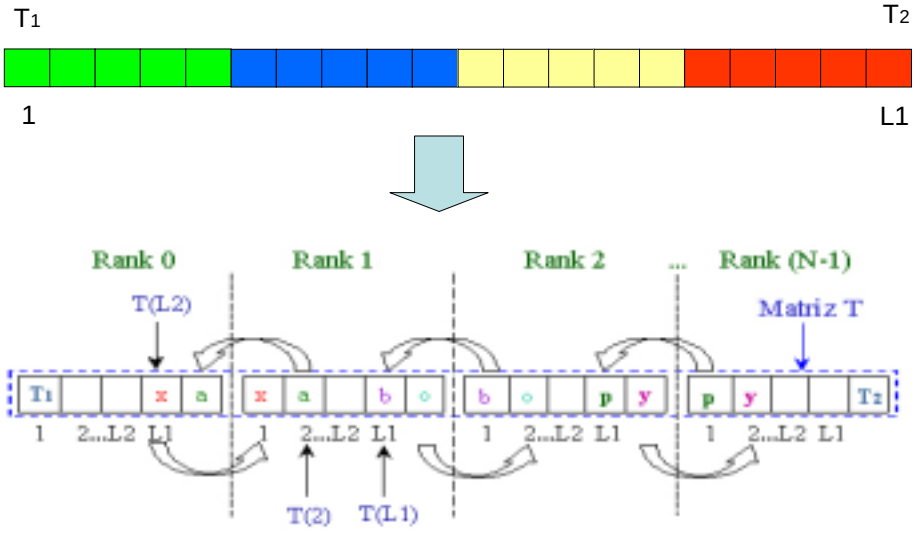
$$T(i) = \text{Relax}\ \frac{(A_E(i) * T(i+1) + A_W(i) * T(i-1) + b(i))}{A_P(i)} + (1 - \text{Relax})\ * T(i)$$

```
      End-do
      !Calcula resíduo médio
      If (Resíduo < Precisão)
            stop=true;
      End-if
End-do
```

Versão Paralela
Troca de mensagens



Versão Paralela
Troca de mensagens



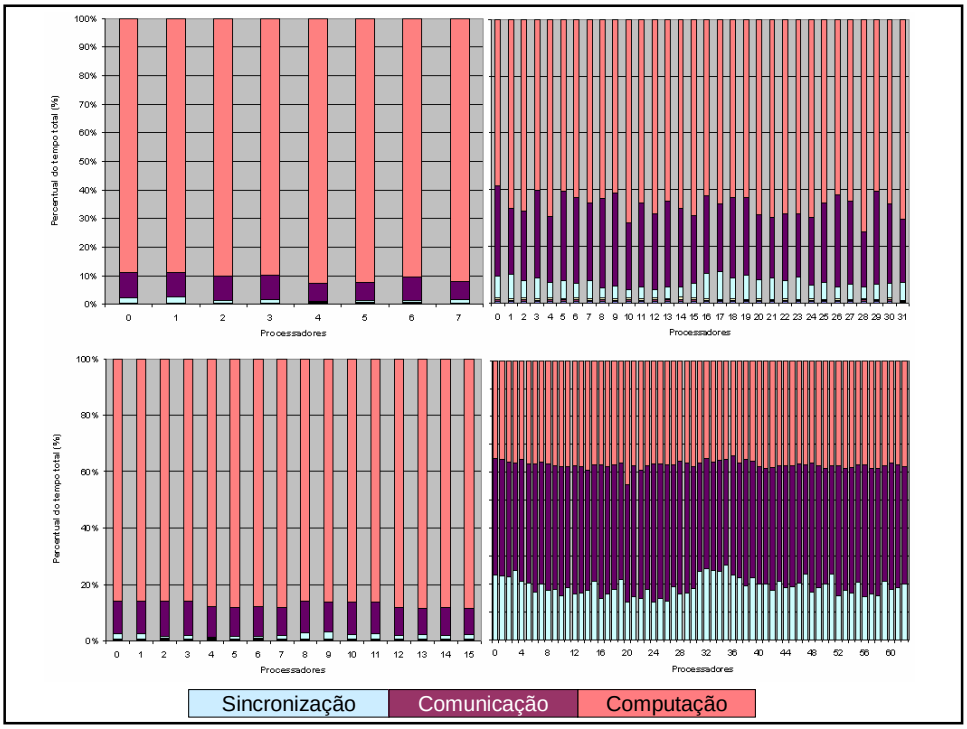
```
! Inicialização do ambiente paralelo
If (Rank = 0)
  ! Leitura dos dados de entrada
  Mpi_send (...)          ! Dados de entrada são enviados para os outros processadores
Else
  Mpi_recv (...)          ! Outros processadores recebem valores de entrada do processador 0 (zero)
End-if
! Definição da malha (uniforme)
! Cada processador calcula os coeficientes independentes A_E, A_W, A_P, b e inicializa a matriz de temperatura
Do while (not Stop)
  Do while ((Residuo > Precisao) and (Iter < Ntimes))
      Do i=2, L2
          T(i) = Relax (A_E(i) * T(i+1) + A_W(i) * T(i-1) + b(i)) / A_P(i) + (1 - Relax) * T(i)
      End-do
      Do i=2, L2
          !Cada processador calcula o seu resíduo médio
      End-do
      Iter = Iter + 1      ! Soma as iterações
  End-do
  Mpi_Send(...), Mpi_Recv(...)   ! Comunicação entre processadores – cada processador envia para os
  ! seus vizinhos e recebe deles os valores armazenados no buffer da matriz de temperatura (ver Fig.(7))
  ! Cada processador recalcula o seu resíduo médio após troca de mensagens
  Mpi_Gather(...)              ! Acumula no processador 0 (zero) todos os resíduos médios recalculados
  If (Rank = 0)                ! Verifica no processador zero (0) se todos os processadores convergiram
      Do i=1, Size             ! Size: Quantidade total de processadores
          If (Resid(i) < Precisao)
              Cont=Cont + 1;
          End-if
      End-do
      Mpi_Send(...)            ! Envia valor de Cont para os outros ranks
  Else
      Mpi_Recv(...)            !Recebe o valor de Cont enviado pelo processador 0 (zero)
      If (Cont = Size)         ! Todos os processadores terminam o cálculo
          Stop=true;
      End-if;
  End-if
End-do;
If (Rank = 0)
  Mpi_Recv(...)           ! Recebe as matrizes de temperatura dos outros processadores para gravação arquivo
Else
  Mpi_Ssend(…)            ! Envia a matriz de temperaturas para o processador 0(zero) em ordem. Isto permite a
                          ! gravação no arquivo de saída em ordem crescente de número de processador
End-if
```


```
! Inicialização do ambiente paralelo
If (Rank = 0)
  ! Leitura dos dados de entrada
  Mpi_send (...)          ! Dados de entrada são enviados para os outros processadores
Else
  Mpi_recv (...)          ! Outros processadores recebem valores de entrada do processador 0 (zero)
End-if
! Definição da malha (uniforme)
! Cada processador calcula os coeficientes independentes A_E, A_W, A_P, b e inicializa a matriz de temperatura
Do while (not Stop)
  Do while ((Residuo > Precisao) and (Iter < Ntimes))
      Do i=2, L2
          T(i) = Relax (A_E(i) * T(i+1) + A_W(i) * T(i-1) + b(i)) / A_P(i) + (1 - Relax) * T(i)
      End-do
      Do i=2, L2
          !Cada processador calcula o seu resíduo médio
      End-do
      Iter = Iter + 1      ! Soma as iterações
  End-do
```


1

1

```
Mpi_Send(...), Mpi_Recv(...)   ! Comunicação entre processadores – cada processador envia para os
! seus vizinhos e recebe deles os valores armazenados no buffer da matriz de temperatura (ver Fig.(7))
! Cada processador recalcula o seu resíduo médio após troca de mensagens
Mpi_Gather(...)                ! Acumula no processador 0 (zero) todos os resíduos médios recalculados
If (Rank = 0)                  ! Verifica no processador zero (0) se todos os processadores convergiram
    Do i=1, Size               ! Size: Quantidade total de processadores
        If (Resid(i) < Precisao)
            Cont=Cont + 1;
        End-if
    End-do
    Mpi_Send(...)              ! Envia valor de Cont para os outros ranks
Else
    Mpi_Recv(...)              !Recebe o valor de Cont enviado pelo processador 0 (zero)
    If  (Cont = Size)          ! Todos os processadores terminam o cálculo
        Stop=true;
    End-if;
End-if
End-do;
If (Rank = 0)
  Mpi_Recv(...)          ! Recebe as matrizes de temperatura dos outros processadores para gravação arquivo
Else
  Mpi_Ssend(…)           ! Envia a matriz de temperaturas para o processador 0(zero) em ordem. Isto permite a
                         ! gravação no arquivo de saída em ordem crescente de número de processador
End-if
```



Algoritmos Paralelos (disciplina regular)
Ementa:

1. Modelos de computação paralela:
   - Memória compartilhada: PRAM síncrona e assíncrona, QSP, QWQR, BSP, LogP, SMP
   - Memória distribuída: SMPD
   - circuitos combinatoriais
2. paradigmas de programação: multi-threaded, BSP, passagem de mensagem, funcional, tuple space, CSP
3. desempenho: profundidade de computação, trabalho, custo, speed-up, eficiência
4. Famílias fundamentais de algoritmos: árvore binária balanceada, divide-and-conquer, jumping pointer, compressão, randomização
5. aplicações
   - ordenação
   - grafos
   - processamento de strings
   - álgebra linear
   - otimização
   - Complexidade paralela: P-completeness

UFES
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO