

Computação Paralela sobre Sistemas Distribuídos

■ Vantagens:

- Relação Custo x Desempenho;
- Melhorar a utilização dos recursos (hardware) já existentes;
- Utilização de “ferramentas” conhecidas;
- Aprendizagem relativamente fácil;

Ambientes para Troca de Mensagens

- Desenvolvimento de “Plataformas Portáteis”;
 - Conjunto de funções independentes de máquina, que executam em diversas plataformas de hardware;
- Oferecem o suporte necessário para o desenvolvimento de aplicações paralelas em SD;
 - Bibliotecas de comunicação: extensão das linguagens seqüenciais;
- Necessidade de tratar a heterogeneidade → Portabilidade;

Ambientes para Troca de Mensagens

- Diversas opções disponíveis:
 - Exemplos: PVM, Express, P4, PCODE, Parmacs, entre outros;
- Mais utilizadas: PVM e MPI

Exemplos de Ambientes para Troca de Mensagens

■ PVM - *Parallel Virtual Machine*

- 1ª versão - 1989;
- ORNL, Univ. of Tennessee e Emory University;
- Conjunto integrado de bibliotecas e ferramentas de software;
- Emula um sistema de computação concorrente, flexível e de propósito geral;
- Padrão “de fato”;
- Paradigmas de Programação adotados:
 - SPMD → Paralelismo de Dados;
 - MPMD → Paralelismo Funcional.

Exemplos de Ambientes para Troca de Mensagens

■ PVM - *Parallel Virtual Machine*

- Possibilita o desenvolvimento de aplicações em C/C++ e Fortran;
- Todos os programas PVM devem ser “linkados” com a biblioteca PVM;
- É um ambiente *freeware* e *open-source*;
- Existem diversas ferramentas para “depurar” programas PVM e também verificar o estado da máquina paralela virtual.

Exemplos de Ambientes para Troca de Mensagens

- PVM - *Parallel Virtual Machine*
 - Dividido em duas partes:
 - PVM *Daemon* (pvmd): Responsável pelo roteamento das mensagens entre aplicações; Deve estar executando em todos os *hosts* da máquina paralela virtual;
 - Biblioteca libpvm: Contém as rotinas que o usuário pode chamar para enviar/receber mensagens, gerar processos e modificar a máquina virtual.

Desenvolvimento de Aplicações PVM

- Passos para o desenvolvimento de aplicações PVM:
 - Funções de controle de processo;
 - Podem aparecer tanto no início como no fim de programas;
 - Funções de informação;
 - Funções para envio e recebimento de mensagens.

Desenvolvimento de Aplicações PVM

- Funções (C) de controle de processo:
 - `pvm_mytid()` : Retorna o Identificador do processo atual. Se for chamada pela 1ª vez, “cadastra” o processo no sistema PVM;
 - `pvm_parent()` : Retorna o Identificador do processo “pai” que gerou o processo atual ou o valor `PvmNoParent` se não foi gerado por `pvm_spawn()` ;

Desenvolvimento de Aplicações PVM

■ Funções (C) de controle de processo:

- `pvm_spawn(char *task, char **argv, int flag, char *where, int ntask, int *tids);`
- Esta função gera `<ntask>` cópias de um arquivo executável `<task>` com o(s) argumento(s) `<argv>` nos computadores que compõem a máquina virtual. O argumento `<flag>` especifica opções para a geração das tarefas, e pode contemplar uma soma dos valores:
 - `PvmTaskDefault`: PVM escolhe (política *round-robin*) onde os processos serão executados;
 - `PvmTaskHost`: Indica que o argumento `<where>` é o *host* onde a tarefa deverá ser executada;
 - `PvmTaskDebug`: Inicia tarefas com um *debugger*;
 - `PvmTaskTrace`: Dados de *trace* são gerados;

Desenvolvimento de Aplicações PVM

■ Funções (C) de controle de processo:

- `pvm_spawn(char *task, char **argv, int flag, char *where, int ntask, int *tids):`
- O argumento `<tids>` é configurado como um vetor contendo o Identificador de cada uma tarefas geradas;
- Esta função retorna o número de tarefas geradas com sucesso ou um código de erro se nenhuma tarefa pode ser gerada.

Desenvolvimento de Aplicações PVM

- Funções (C) para envio de mensagens:
 - Envio de mensagens: 3 etapas;
 - `pvm_initsend(int enconding)` : “Limpa” ou cria um buffer para enviar uma nova mensagem. O argumento `<enconding>` pode conter um dos seguintes valores:
 - `PvmDataDefault`: Indica que a codificação XDR será utilizada;
 - `PvmDataRaw`: Nenhum esquema de codificação é utilizado;
 - `PvmDataInPlace`: Indica que o buffer deve conter apenas tamanhos e ponteiros para os dados que serão enviados, pois, esses dados devem ser copiados diretamente do espaço do usuário.

Desenvolvimento de Aplicações PVM

■ Funções (C) para envio de mensagens:

- `pvm_pk<tipo de dado>(<tipo de dado> *cp, int nitem, int stride):`
- “Empacota” um vetor de um determinado tipo de dado dentro do *buffer* ativo. Tipo de dado pode ser um dos seguintes valores:
 - `byte, double, float, int, long, short, str;`
- O argumento `<cp>` é um ponteiro para o primeiro item a ser empacotado; `<nitem>` indica o número total de itens; `<stride>` indica qual seqüência (2 em 2, 3 em 3) deve ser adotada para empacotar o vetor enviado.

Desenvolvimento de Aplicações PVM

■ Funções (C) para o envio de mensagens:

- `pvm_send (int tid, int msgtag) :`
 - Identifica por `<msgtag>` a mensagem que foi previamente empacotada e envia para o processo com o identificador `<tid>`;
- `pvm_mcast (int *tids, int ntask, int msgtag) :`
 - Identifica por `<msgtag>` a mensagem que foi previamente empacotada e envia para todos os processos especificados no vetor de identificadores `<tids>`;
 - O parâmetro `<ntask>` indica o número de elementos do vetor `<tids>`.

Desenvolvimento de Aplicações PVM

- Funções (C) para recebimento de mensagens:
 - `pvm_recv (int tid, int msgtag)` : Esta função aguarda até que uma mensagem `<msgtag>` do processo `<tid>` tenha chegado;
 - Quando uma mensagem chega, um novo *buffer* de recepção é criado;
 - Se o valor -1 for especificado em `<tid>` ou em `<msgtag>`, qualquer mensagem de qualquer processo será recebida;
 - Retorna o identificador do buffer criado.
 - `pvm_nrecv (int tid, int msgtag)` : Esta função retorna 0 se a mensagem indicada por `<msgtag>` ainda não chegou;
 - Pode ser chamada diversas vezes para a mesma mensagem.

Desenvolvimento de Aplicações PVM

■ Funções (C) para recebimento de mensagens:

- `pvm_upk<tipo de dado> (<tipo de dado> *cp, int nitem, int stride):`
- Desempacota o vetor de <tipo de dado> do buffer de recepção ativo;
- <nitem> especifica o número de itens de <tipo de dado> que deve ser desempacotado;
- <stride> indica a seqüência que deve ser adotada.
 - <tipo de dado> pode assumir um dos seguintes valores: `byte`, `float`, `double`, `int`, `long`, `short`, `char`;

Desenvolvimento de Aplicações PVM

■ Funções (C) de controle de processos:

- `pvm_exit()` : Informa ao pvmd local que o processo está saindo do sistema PVM;
 - Após chamar a função `pvm_exit()`, o processo pode desempenhar tarefas como qualquer outro processo;
 - Comumente, a função `pvm_exit()` é chamada antes de `return` ou `exit`.

Compilando e executando aplicações PVM

- Compilação:
 - Utilitário `aimk`;
- Execução:
 - A máquina paralela virtual deve ser previamente criada;
 - Em caso de aplicações MPMD executa-se o Mestre;

Exemplo de uma aplicação PVM

Aplicação MPMD

Identificação de processos

Recebendo

Msg.

Saída do sistema

```
#include "pvm3.h"
main()
{
    int cc, tid, msgtag;
    char buf[100];
    printf("i'm %x", pvm_mytid());
    cc = pvm_spawn("hello_other",
                  (char **)0,0,"", 1, &tid);
    if (cc==1){
        msgtag=1;
        pvm_rcv (tid, msgtag);
        pvm_upkstr(buf);
        printf("from %x: %s\n", tid, buf);
    }else
        printf("can't start hello_other\n");
    pvm_exit();
}
```

```
#include "pvm3.h"
main()
{
    int ptid, msgtag;
    char buf[100];

    ptid = pvm_parent();
    strcpy(buf, "hello, world from");
    gethostname(buf+strlen(buf), 64);
    msgtag=1;
    pvm_initsend (PvmDataDefault);
    pvm_pkstr (buf);
    pvm_send (ptid, msgtag);

    pvm_exit();
}
```

Exemplos de Ambientes para Troca de Mensagens

- MPI - *Message Passing Interface*
 - Início: 1992;
 - Proposta de um **Padrão** de interface de passagem de mensagens para computadores com memória distribuída e NOWs (*Networks Of Workstations*);
 - Objetivos:
 - Unir portabilidade e facilidade de uso;
 - Fornecer uma especificação precisa para o desenvolvimento de ambientes de passagem de mensagem;
 - Possível crescimento da indústria de software paralelo;
 - Difusão do uso de computadores paralelos.

Exemplos de Ambientes para Troca de Mensagens

- MPI - *Message Passing Interface*
 - Rotinas de comunicação ponto a ponto;
 - Rotinas de comunicação coletiva;
 - Rotinas de gerenciamento de processos;
 - E/S paralelos;
 - Suporte para grupos de processos;
 - Suporte para contextos de comunicação;
 - Suporte para topologias de aplicação.

Exemplos de Ambientes para Troca de Mensagens

- MPI - *Message Passing Interface*
 - Exemplos de implementações (*freeware*) MPI:
 - CHIMP/MPI: Edinburgh Parallel Computing Centre (EPCC);
 - LAM: Ohio Supercomputer Center;
 - MPIAP: Australian National University;
 - MPICH: Argonne National Laboratory & Mississippi State University;
 - MPI-FM: University of Illinois;
 - W32MPI: Universidade de Coimbra - Portugal.

Desenvolvimento de Aplicações MPI

- Mesmos passos utilizados no desenvolvimento de aplicações PVM;
- Implementação LAM (*Local Area Muticomputer*)
 - Mudanças, em relação ao PVM, na maneira como a máquina virtual é gerenciada e na execução das aplicações;

Desenvolvimento de Aplicações MPI

■ Funções (C) para controle de processos:

- `MPI_Init (int *argc, char ***argv) :` Registra a aplicação junto a sessão MPI;
- `MPI_Comm_rank (MPI_Comm comm, int *rank) :`
Retorna em `<rank>` o Identificador do processo atual pertencente ao espaço `<comm>`;
- `MPI_Comm_size (MPI_Comm comm, int *size) :`
Retorna em `<size>` o número atual de processos em execução pertencentes ao espaço `<comm>`.

Desenvolvimento de Aplicações MPI

- Funções (C) para envio de mensagens:
 - 4 modos: padrão, bufferizado, síncrono e pronto;
 - Padrão: Dependendo da implementação pode assumir uma semântica síncrona ou de bufferização;
 - Bufferizado: A operação *Send* é completada quando a mensagem é bufferizada no espaço fornecido pela aplicação;
 - Síncrono: O transmissor recebe uma confirmação de recebimento da mensagem por parte do receptor;
 - Pronto: O transmissor envia a mensagem sem considerar buffers ou confirmações. O *Send* não deve iniciar a menos que um *Receive* tenha iniciado;
 - Possibilidade de utilizar funções bloqueantes ou não-bloqueantes;

Desenvolvimento de Aplicações MPI

- Funções (C) para envio de mensagens:
 - Função bloqueante: Não retorna até que o buffer de dados possa ser reutilizado;
 - Todas as funções dessa categoria possuem a mesma sintaxe;
 - Função Não-bloqueante: Retorna imediatamente; Há possibilidade do buffer de dados ser sobrescrito; Possibilidade de inconsistência!;
 - Para cada função bloqueante, há uma não-bloqueante;

Desenvolvimento de Aplicações MPI

Funções (C) para envio de mensagens:

	Padrão	Bufferizado	Síncrono	Pronto
Bloqueante	MPI_Send	MPI_Bsend	MPI_Ssend	MPI_Rsend
Não Bloqueante	MPI_Isend	MPI_Ibsend	MPI_Issend	MPI_Irsend

Desenvolvimento de Aplicações MPI

■ Funções (C) para envio de mensagens:

- `MPI_Send (void *buf, int count, MPI_DataType dtype, int dest, int tag, MPI_Comm comm):`
- Envia a mensagem `<buf>` com identificador `<tag>` para o destino `<dest>` pertencente ao espaço `<comm>`. O parâmetro `<count>` indica quantos elementos do tipo `<dtype>` existem na mensagem;
- Os tipos possíveis para uma mensagem são: `MPI_CHAR`, `MPI_SHORT`, `MPI_INT`, `MPI_LONG`, `MPI_FLOAT`, `MPI_DOUBLE`, `MPI_BYTE`, `MPI_UNSIGNED_CHAR`, `MPI_UNSIGNED_SHORT`, `MPI_UNSIGNED`, `MPI_UNSIGNED_LONG`, `MPI_LONG_DOUBLE`;

Desenvolvimento de Aplicações MPI

- Funções (C) para recebimento de mensagens:
 - são divididas em bloqueantes e não-bloqueantes;
 - `MPI_Recv`: Recebimento bloqueante; Aguarda até que uma mensagem tenha chegado no buffer de recepção;
 - `MPI_Irecv`: Recebimento Não-bloqueante; Verifica se existe uma mensagem no buffer de recepção e, caso positivo, recebe a mensagem. Em caso negativo, retorna imediatamente;

Desenvolvimento de Aplicações MPI

■ Funções (C) para recebimento de mensagens:

- `MPI_Recv (void *buf, int count, MPI_Datatype dtype, int source, int tag, MPI_Comm comm, MPI_Status *status) :`
- Recebe a mensagem `<buf>` com identificador `<tag>` da fonte `<source>` pertencente ao espaço `<comm>`. O parâmetro `<count>` indica quantos elementos do tipo `<dtype>` existem na mensagem;
- Se o valor `MPI_ANY_SOURCE` ou `MPI_ANY_TAG` for especificado, respectivamente, em `<source>` e `<tag>` serão aceitas todas as mensagens;

Desenvolvimento de Aplicações MPI

- Funções (C) para controle de processos:
 - `MPI_Finalize()` : Informa que o processo está saindo de uma sessão MPI;
 - `MPI_Abort(MPI_Comm comm, int errcode)` : Termina a execução de todos os processos de um comunicador `<comm>`, incluindo o próprio processo que realizou a chamada à função `MPI_Abort()`;

Compilando e executando aplicações LAM

■ Execução:

- Arquivos esquema da aplicação: Arquivos texto que contém onde (em que *host*) cada arquivo executável (mestre e escravo) deverá ser executado; LAM identifica cada *host* como um nó (n) juntamente com um número identificador;
- Exemplo de um arquivo esquema:
 - n0 /home/master
 - n1 /home/slave
 - n2 /home/slave
 - n3 /home/slave

Iniciando e Gerenciando o Multicomputador no LAM

■ Utilitários:

- `lamboot <hostsfile>`: Inicia o Multicomputador nos *hosts* especificados em `<hostsfile>`; `<hostsfile>` é um arquivo texto contendo os nomes das máquinas que formarão o Multicomputador;
 - `%lamboot hosts`
- Exemplo de um arquivo esquema de *hosts*:

```
lasdpc06
lasdpc07
lasdix
```
- `recon <hostsfile>`: Verifica se o Multicomputador está pronto;

Exemplo de uma aplicação LAM

- Objetivo: Transmitir um vetor de inteiros entre dois processos

Aplicação SPMD

```
#include <mpi.h>
#define BUFSIZE 64
int buf[BUFSIZE];
main(argc, argv)
int argc;
char *argv[];
{
    int size, rank;
    MPI_Status status;
    MPI_Init (&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size!=2){
        MPI_finalize();
        return (1);
    }
    MPI_Comm_rank (MPI_COMM_WORLD, &rank)

    if (rank==0)
        MPI_Send(buf, sizeof(buf),MPI_INT,
                  1,11, MPI_COMM_WORLD);

    else
        MPI_Recv(buf,sizeof(buf),MPI_INT,
                  0,11,MPI_COMM_WORLD,&status);

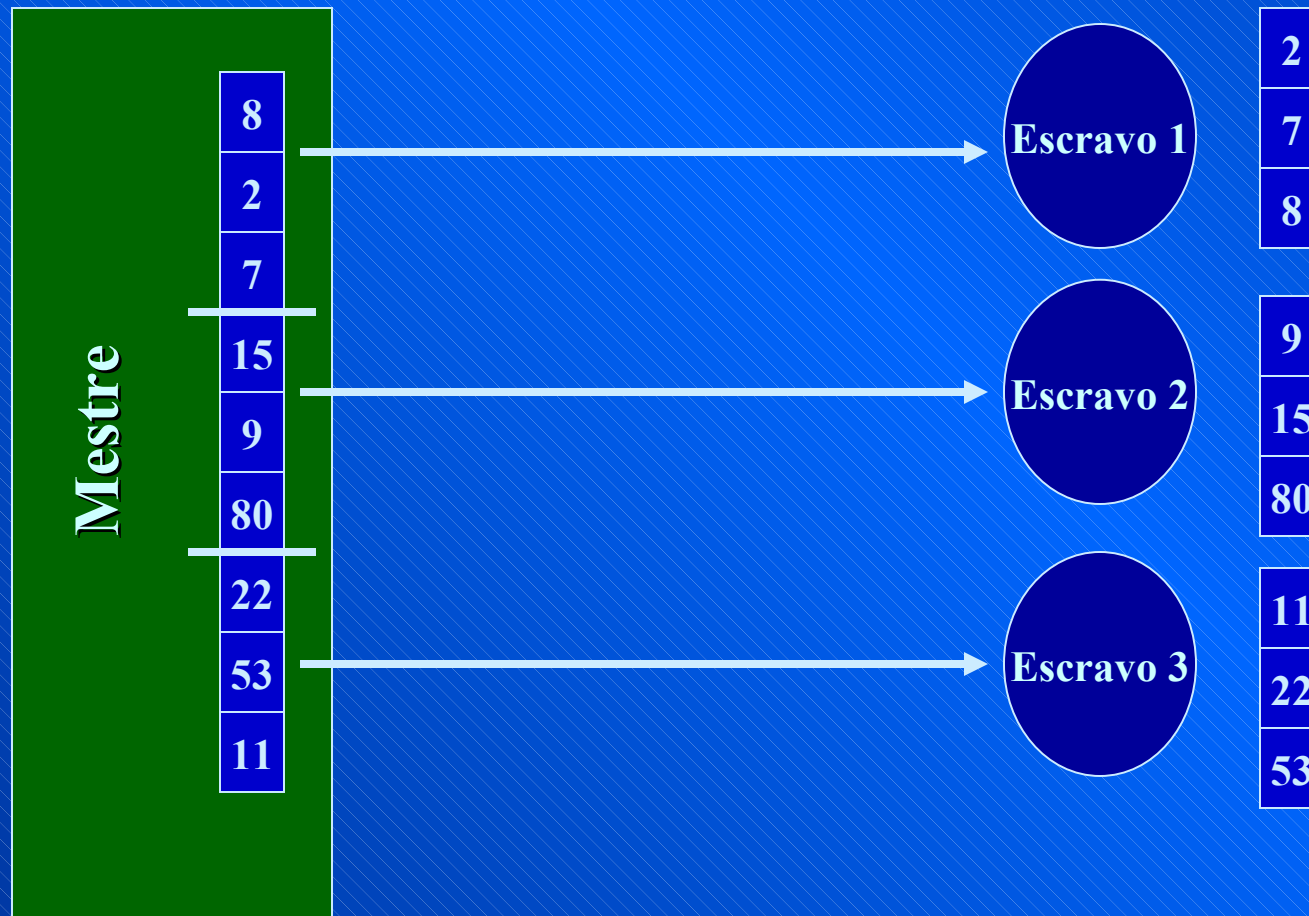
    MPI_Finalize();
    return (0);
}
```

Exemplos de Aplicações Paralelas em PVM e MPI (LAM)

- Ordenação de vetores (algoritmo *QuickSort*);
 - Utilizando o paradigma MPMD
 - Mestre: ordena
 - Escravo(s): quick
- Método do Trapézio Composto;
 - Utilizando o paradigma SPMD
- Multiplicação de Matrizes;
 - Utilizando o paradigma SPMD

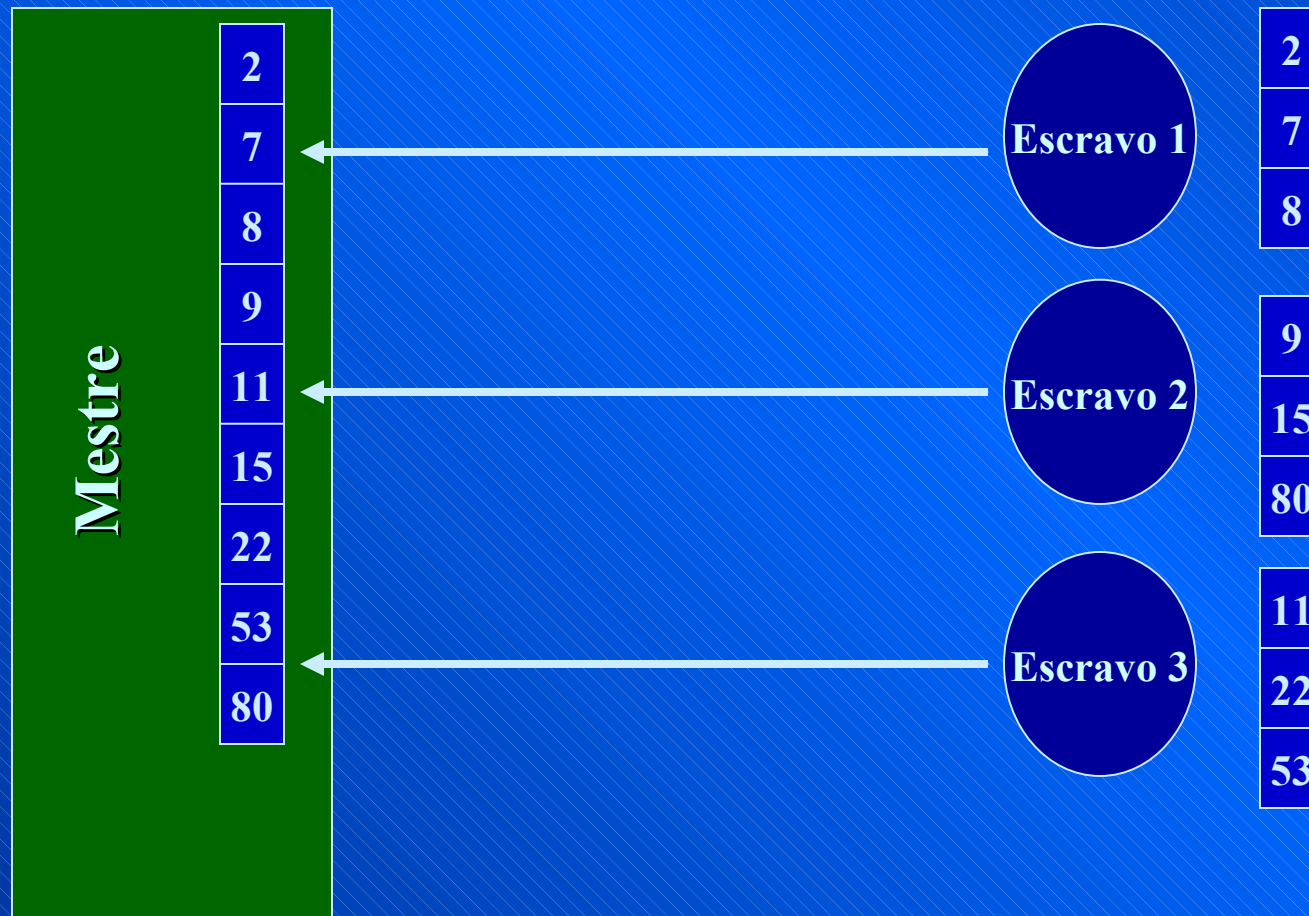
Exemplos de Aplicações em PVM e MPI (LAM)

- Ordenação de vetores (algoritmo QuickSort);



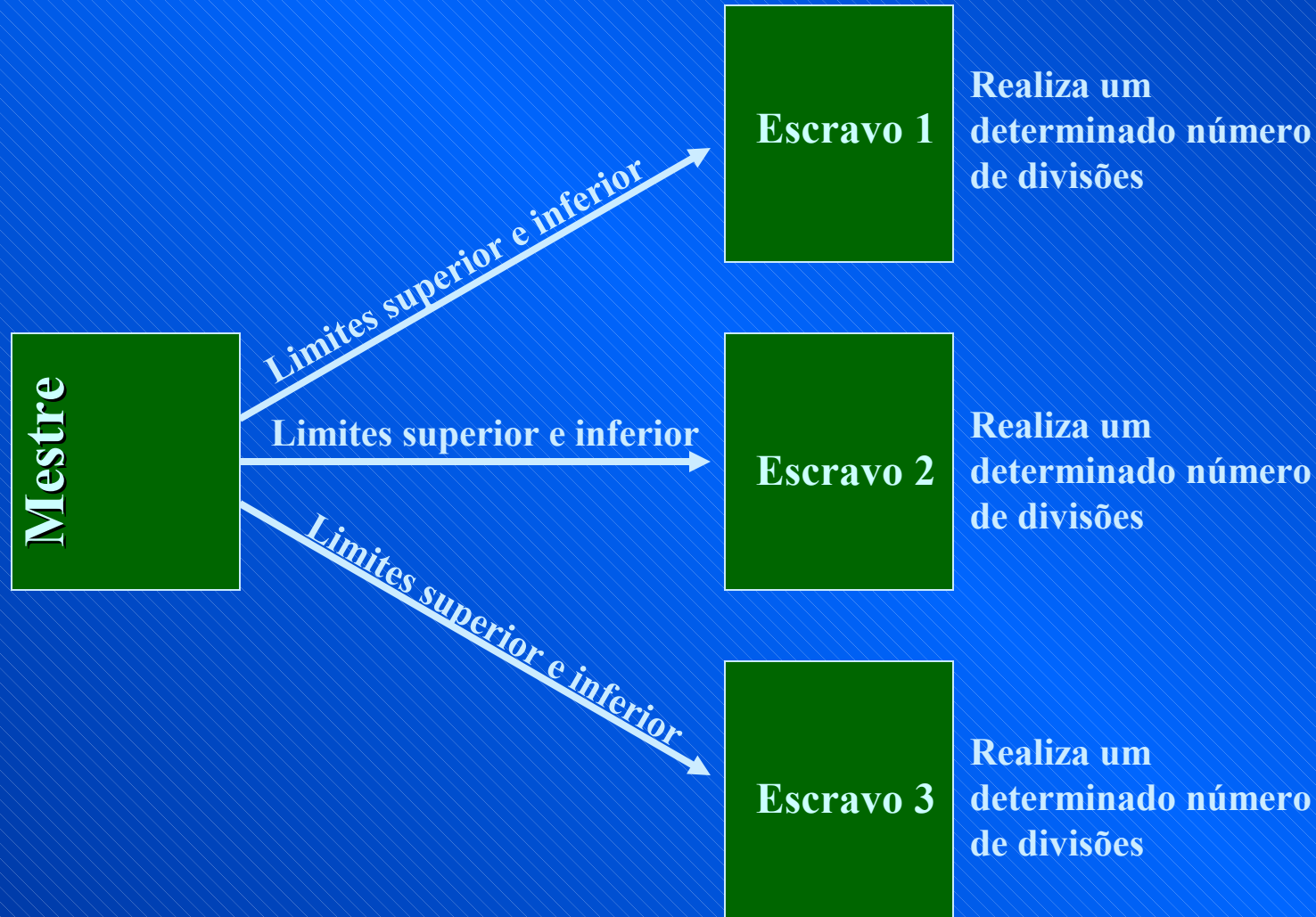
Exemplos de Aplicações em PVM e MPI (LAM)

- Ordenação de vetores (algoritmo QuickSort);



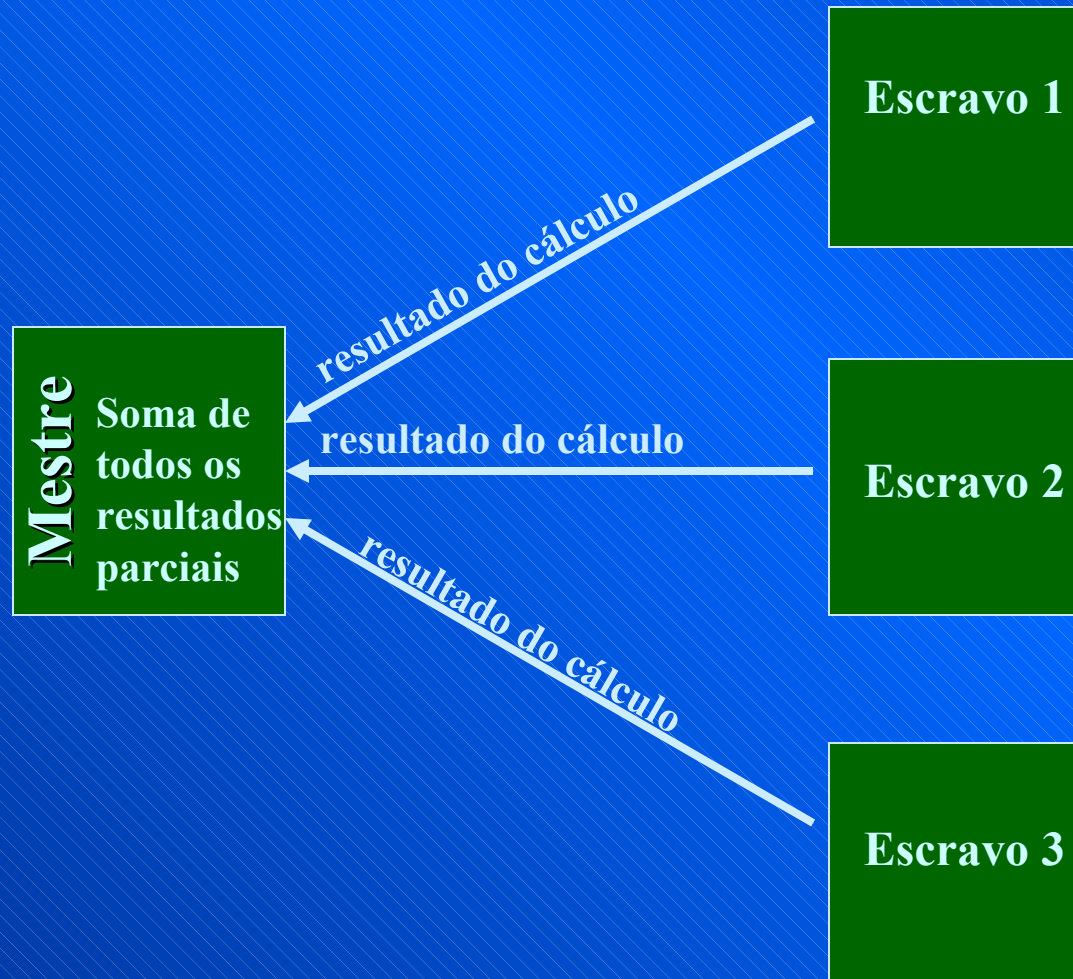
Exemplos de Aplicações em PVM e MPI (LAM)

■ Método do Trapézio Composto



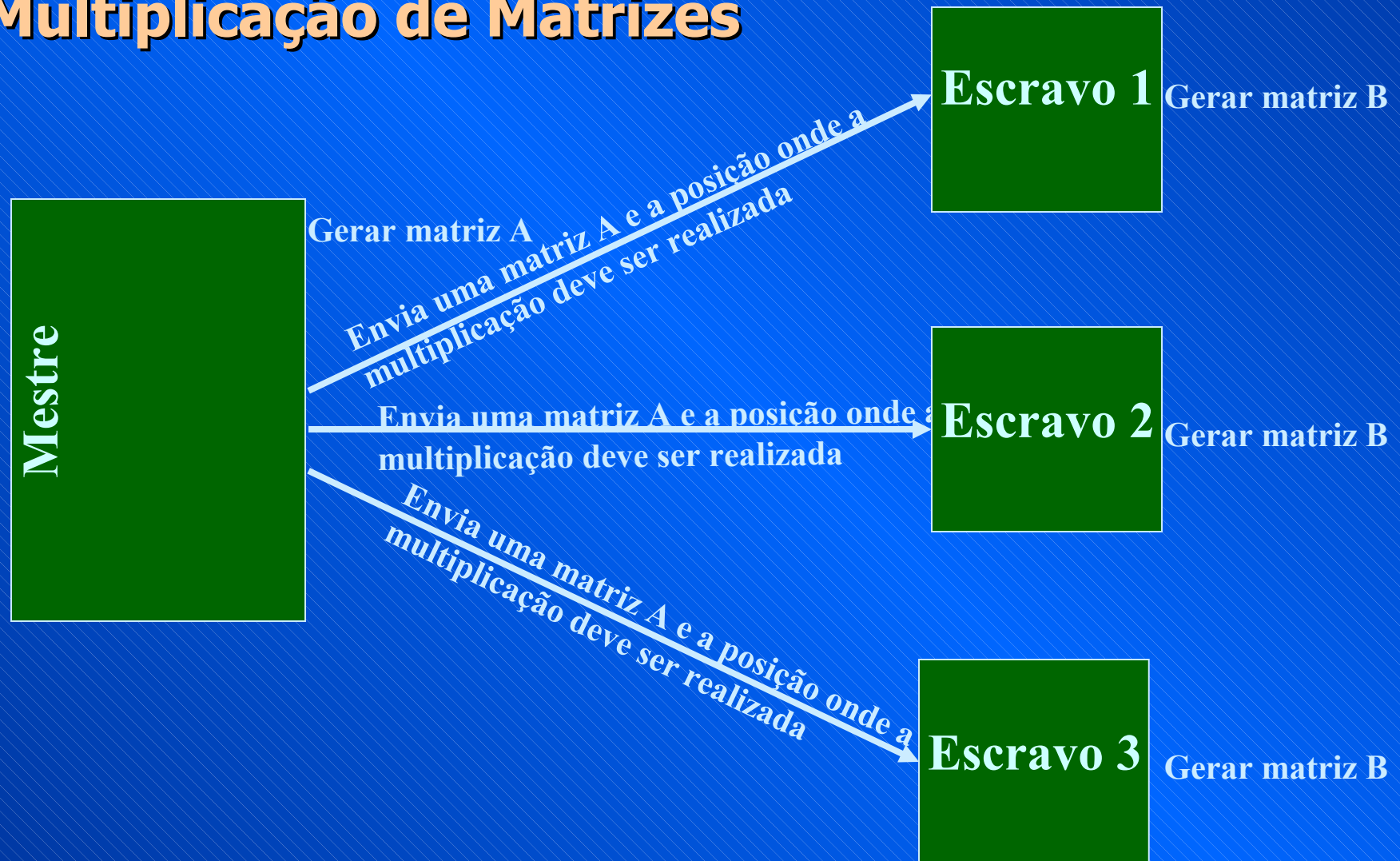
Exemplos de Aplicações em PVM e MPI (LAM)

■ Método do Trapézio Composto



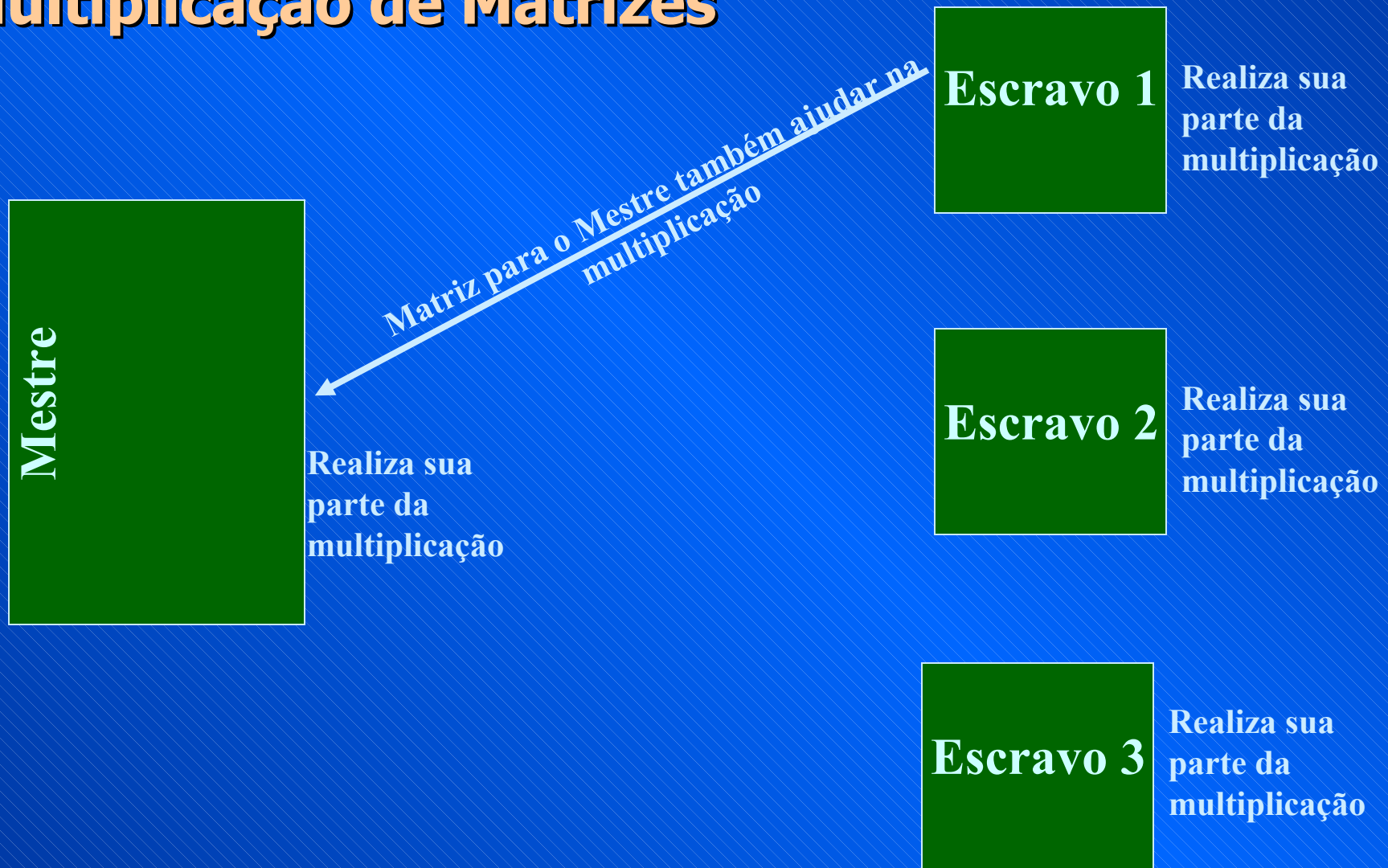
Exemplos de Aplicações em PVM e MPI (LAM)

■ Multiplicação de Matrizes



Exemplos de Aplicações em PVM e MPI (LAM)

■ Multiplicação de Matrizes



Exemplos de Aplicações em PVM e MPI (LAM)

■ Multiplicação de Matrizes



Exemplos de Aplicações PVM

■ Ordenação de vetores (algoritmo QuickSort): Mestre

```
#include "pvm3.h"
#define MaxVet 300000 // numero de elementos para ordenar
#define NUM_PROC 4    // numero de escravos
#define ESCRAVO "/home/tests/pvm/quick"

int main(int argc, char **argv)
{
    my_tid = pvm_mytid();
    tot_subvet = MaxVet/NUM_PROC;

    cc = pvm_spawn(ESCRAVO, (char **)0, PvmTaskDefault, NULL,
                  NUM_PROC, tid_sort);

    if (cc < NUM_PROC) {
        printf ("ERRO:pvm_spawn nao gerou todas as tasks[gerou %d]...\n", cc);
        exit (-1);
    }
    for (t = 0; t < NUM_PROC; t++)
    {
        pvm_initsend (PvmDataRaw);
        pvm_pkint (&vetor[(tot_subvet * t)], tot_subvet, 1);
        pvm_send (tid_sort[t], 1);
    }
}
```

Identificação do processo

Divisão pelo número de escravos

Gerando os escravos

Enviando uma parte do vetor para cada escravo

Exemplos de Aplicações PVM

■ Ordenação de vetores (algoritmo QuickSort): Mestre

```
...  
for (t = 0; t < NUM_PROC; t++)  
{  
    pvm_recv (-1, 2);  
    pvm_upkint (&vetor[(tot_subvet * t)], tot_subvet, 1);  
    indices[t] = (tot_subvet * t);  
}  
merge(vetor, indices);  
pvm_exit();  
return(0);
```

Recebendo o vetor ordenado de cada escravo

Informa sua saída ao pvmd

}

Exemplos de Aplicações PVM

■ Ordenação de vetores (algoritmo QuickSort): Escravo

```
#include "pvm3.h"

#define MaxVet 300000 // numero de elementos para ordenar
#define NUM_PROC 4    // numero de escravos (quick)

int main(void) {

    tid_parent = pvm_parent();
    tot_subvet = (MaxVet/NUM_PROC);

    buf_rcv = pvm_rcv (tid_parent, 1),
    pvm_upkint (a, tot_subvet, 1);
    ordena (a, 0, (tot_subvet - 1));

    pvm_initsend(PvmDataRaw);
    pvm_pkint(a, tot_subvet, 1);
    pvm_send(tid_parent, 2);

    pvm_exit();
    return(0);
}
```

Identificação do processo
Mestre

Divisão pelo número de escravos

Recebendo uma parte do vetor para ordenar

Enviando o vetor ordenado para o mestre

Informa sua saída ao pvmd

Exemplos de Aplicações LAM-MPI

■ Ordenação de Vetores: Mestre

```
#include "mpi.h"
#define MaxVet 5000
#define NUM_PROC 2
#define WORKTAG 1
int main(int argc, char **argv)
{
    MPI_Status status;
    tot_subvet=MaxVet/NUM_PROC;
    x=MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    for (x=1;x<=NUM_PROC;x++)
        MPI_Send(&vetor[(tot_subvet*(x-1))],tot_subvet,MPI_INT,x,WORKTAG,
                MPI_COMM_WORLD);

    for (x=0;x<NUM_PROC;x++)
        MPI_Recv(&vetor[(tot_subvet*x)],tot_subvet,MPI_INT,MPI_ANY_SOURCE,
                WORKTAG+1,MPI_COMM_WORLD,&status);

    merge(vetor);
    MPI_Finalize();
    return 0;
}
```

Objeto status utilizado apenas na recepção de mensagens

Registra o processo na sessão LAM

Obtém o número de processos na sessão LAM

Obtém o ID do processo atual

Envia a mensagem <vetor> para o processo <x>

Aguarda o recebimento de uma mensagem vinda de qualquer processo

Informa sua saída de uma sessão LAM

Exemplos de Aplicações LAM-MPI

■ Ordenação de Vetores: Escravo

```
#include "mpi.h"

#define MaxVet 5000
#define NUM_PROC 2
#define WORKTAG 1
int main(int argc, char **argv)
{
    MPI_Status status;
    tot_subvet = (MaxVet/NUM_PROC);

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    MPI_Recv(a, tot_subvet, MPI_INT, 0, WORKTAG, MPI_COMM_WORLD, &status);
    ordena(a, 0, (tot_subvet-1));
    MPI_Send(a, tot_subvet, MPI_INT, 0, WORKTAG+1, MPI_COMM_WORLD);

    MPI_Finalize();
    return 0;
}
```

Objeto status utilizado apenas na recepção de mensagens

Registra o processo na sessão LAM

Obtém o número de processos na sessão LAM

Obtém o ID do processo atual

Aguarda o recebimento de uma mensagem vinda do Mestre (0)

Envia a mensagem para o Mestre (0)

Informa sua saída de uma sessão LAM

Exemplos de Aplicações PVM

Método dos Trapézios Composto: Mestre

```
#define N 1200000
#define TASKS 2;
#define ESCRAVO "/home/tests/pvm/integralpar_pvm"

int main(int argc, char **argv)
{
    my_id=pvm_mytid();
    id_parent = pvm_parent();
    char *hosts[]={"maq01","maq02"};
    if (id_parent==PvmNoParent)
    {
        for (i=0;i<TASKS;i++)
        {
            cc = pvm_spawn(ESCRAVO, (char**)0, PvmTaskHost, hosts[i],
                          workers, &tids[i]);

            if (cc < workers) {
                printf ("ERRO:pvm_spawn nao gerou todas as tasks[gerou %d]...\n", cc);
                exit (-1);
            }
        }
        a = 0.0; b = 1.2; Ndiv = N/numtasks;
        x = (double)(b - a)/numtasks;

        limite_inferior = a;
        limite_superior = a + x;
    }
}
```

Identificação do processo

Identificação do processo
Mestre

Verifica se o processo atual é
mestre ou escravo

Gera os processos escravos

Exemplos de Aplicações PVM

■ Método dos Trapézios Composto: Mestre

```
for (i=0;i<workers;i++)
```

```
{  
    pvm_initsend (PvmDataRaw);  
    pvm_pkdouble (&limite_inferior,1,1);  
    pvm_pkdouble (&limite_superior,1,1);  
    pvm_send (tids[i], 1);  
    limite_inferior = limite_superior;  
    limite_superior = limite_superior + x;  
}
```

Envia os limites
para cada um dos
escravos

Calcula a integral

```
res = integral(limite_inferior,limite_superior,Ndiv);
```

```
for (i=0; i<workers;i++)
```

```
{  
    pvm_recv (-1, 2);  
    pvm_upkdouble (&valor_retorno,1, 1);  
    sum = sum + valor_retorno;  
}
```

Recebe os resultados de
cada um dos escravos

```
res = res + sum;
```

Soma todos os resultados

```
}
```

Exemplos de Aplicações PVM

■ Método dos Trapézios Composto: Escravo

```
else{  
  
    Ndiv = N/numtasks;  
  
    buf_rcv = pvm_rcv (id_parent, 1);  
    pvm_upkdouble (&a, 1, 1);  
    pvm_upkdouble (&b, 1, 1);  
    pvm_freebuf (buf_rcv);  
  
    res = integral(a,b,Ndiv);  
  
    pvm_initsend(PvmDataRaw);  
    pvm_pkdouble(&res, 1, 1);  
    pvm_send(id_parent, 2);  
}  
  
pvm_exit();  
return(0);  
}
```

Recebe os limites

Calcula a integral

Envia os resultados para o mestre

Informa sua saída ao pyemd

Exemplos de Aplicações LAM-MPI

■ Método dos Trapézios Composto: Mestre

```
#include "mpi.h"
#define MASTER 0 /* ID do primeiro processo (MESTRE) */
#define N 1200000

MPI_Status status;
main(int argc, char **argv)
{
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    numworkers = numtasks-1;
    if (taskid == MASTER) {
        a = 0.0; b = 1.2;
        Ndiv = N/numtasks; x = (double)(b - a)/numtasks;
        limite_inferior = a; limite_superior = a + x;
        for (i=1; i<=numworkers; i++) {
            MPI_Send(&limite_inferior, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
            MPI_Send(&limite_superior, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
            limite_inferior = limite_superior;
            limite_superior = limite_superior + x;
        }
    }
}
```

← Inicializar junto à sessão MPI

← Obtém o ID do processo atual

← Obtém o número de processos na sessão

LAM

← Envia para cada escravo os limites do cálculo

Exemplos de Aplicações LAM-MPI

■ Método dos Trapézios Composto: Mestre/Escravo

```
...
res = integral(limite_inferior,limite_superior,Ndiv);
for (i=1; i<= numworkers; i++) {
    MPI_Recv(&valor_retorno,1,MPI_DOUBLE,i,1,MPI_COMM_WORLD,&status);
    sum = sum + valor_retorno;
}
res = res + sum;
}
/*ESCRAVO*/
if (taskid > MASTER) {
    Ndiv = N/numtasks;
    source = MASTER;

    MPI_Recv(&a, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, &status);
    MPI_Recv(&b, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, &status);
    res = integral(a,b,Ndiv);
    MPI_Send(&res, 1, MPI_DOUBLE, MASTER, 1,MPI_COMM_WORLD);
}
MPI_Finalize();
}
```

Aguarda o recebimento do resultado do cálculo de todos os escravos

Verifica se é o processo escravo

Recebe as posições onde o cálculo da integral deve ser realizado

Envia para o mestre o resultado do cálculo de integrais

Exemplos de Aplicações PVM

■ Multiplicação de Matrizes: Mestre

```
#include "pvm3.h"
#define TAM 420
#define TASKS 4
#define ESCRAVO "/home/testes/pvm/matrixpar_pvm"

int main(int argc, char **argv)
{
    my_id=pvm_mytid();
    id_parent = pvm_parent();
    if (id_parent==PvmNoParent)
    {
        cc = pvm_spawn(ESCRAVO, (char **)0, PvmTaskDefault, NULL, workers, tids);
        if (cc < workers) {
            printf ("ERRO:pvm_spawn nao gerou todas as tasks[gerou %d]...\n", cc);
            exit (-1);
        }
        gera_matriz(A,1235);
        pos = 0;
        for(i=0;i<workers;i++){
            pvm_initsend (PvmDataRaw);
            pvm_pkdouble (&A[0][0], (TAM*TAM), 1);
            pvm_pkint (&pos,1,1);
            pvm_send (tids[i], 1);
            pos = pos + n_lin;
        }
    }
}
```

Identificação do processo

Identificação do processo
Mestre

Verifica se o processo atual é
mestre ou escravo

Gera os processos escravos

Envia a matriz A e a posição
onde a multiplicação deve
ser realizada

Exemplos de Aplicações PVM

■ Multiplicação de Matrizes: Mestre

```
pvm_recv (-1, 2);  
pvm_upkdouble (&B[0][0], (TAM*TAM), 1);
```

← Recebe a matriz B de qualquer escravo

```
for(k=pos;k<pos+n_lin;k++){  
    for(i=0;i<TAM;i++){  
        for(j=0;j<TAM;j++){  
            C[k][i] = C[k][i] + (A[k][j] * B[j][i]);
```

← Calcula sua parte da matriz C

```
pos = 0;  
for (i=0;i<workers;i++){  
    pvm_recv (-1, 2);  
    pvm_upkdouble (&C[pos][0], (n_lin*TAM), 1);  
    pos = pos + n_lin;  
}
```

← Recebe a parte calculada de todos os escravos

Exemplos de Aplicações PVM

■ Multiplicação de Matrizes: Escravo

```
}  
else{  
    gera_matriz(B,1235);  
    buf_rcv = pvm_rcv (id_parent, 1);  
    pvm_upkdouble (&A[0][0], (TAM*TAM), 1);  
    pvm_upkint (&pos,1,1);  
    if (pos==0)  
    {  
        pvm_initsend(PvmDataRow);  
        pvm_pkdouble(&B[0][0], (TAM*TAM), 1);  
        pvm_send(id_parent, 2);  
    }  
    for(k=pos;k<pos+n_lin;k++){  
        for(i=0;i<TAM;i++){  
            for(j=0;j<TAM;j++){  
                C[k][i] = C[k][i] + (A[k][j] * B[j][i]);  
            }  
        }  
    }  
    pvm_initsend(PvmDataRow);  
    pvm_pkdouble(&C[pos][0], (TAM*TAM), 1);  
    pvm_send(id_parent, 2);  
}  
pvm_exit();  
return(0);  
}
```

Recebe do Mestre a matriz A e a posição que deve ser calculada

Se for o primeiro escravo (pos==0) envie a matriz B para o Mestre

Calcular a parte da Matriz

Enviar os resultados para o Mestre

Informa a saída ao pvmd

Exemplos de Aplicações LAM-MPI

■ Multiplicação de Matrizes: Mestre

```
#include "mpi.h"
#define MASTER      0          /* ID do primeiro processo (MESTRE) */
#define TAM        120
#define TASKS       6
MPI_Status status;


main(int argc, char **argv)
{
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    numtasks=TASKS; numworkers = numtasks-1;
    n_lin = TAM/numtasks; total = TAM*TAM;
```




Indica a faixa da matriz produto que deve ser calculada por cada processo

Exemplos de Aplicações LAM-MPI

■ Multiplicação de Matrizes: Mestre

```
if (taskid==MASTER)  Verifica se é o processo mestre
{
    gera_matriz(A,1235);
    pos = 0;
    for(i=1;i<=numworkers;i++){  Envia para cada escravo a matriz e a
        MPI_Send(A, (TAM*TAM), MPI_DOUBLE, i, 0, MPI_COMM_WORLD);   
        MPI_Send(&pos, 1, MPI_INT, i, 0, MPI_COMM_WORLD);   
        pos = pos + n_lin;
    }
    MPI_Recv(B, total, MPI_DOUBLE, 1, 2, MPI_COMM_WORLD, &status);   

    for(k=pos;k<pos+n_lin;k++)
        for(i=0;i<TAM;i++)
            for(j=0;j<TAM;j++)
                C[k][i] = C[k][i] + (A[k][j] * B[j][i]);
    pos = 0;
    for(i=1;i<=numworkers;i++){
        MPI_Recv(&C[pos][0], (n_lin*TAM), MPI_DOUBLE, i, 3, MPI_COMM_WORLD,   
            &status);   
        pos = pos + n_lin;
    }
}
```

 Recebe o resultado do cálculo com a matriz C

Exemplos de Aplicações LAM-MPI

■ Multiplicação de Matrizes: Escravo

```
if (taskid > MASTER)
{
    gera_matriz(B,1235);
    MPI_Recv(A, (TAM*TAM), MPI_DOUBLE, MASTER, 0, MPI_COMM_WORLD, &status);
    MPI_Recv(&pos, 1, MPI_INT, MASTER, 0, MPI_COMM_WORLD, &status);

    if(taskid == 1)
        MPI_Send(B, (TAM*TAM), MPI_DOUBLE, 0, 2, MPI_COMM_WORLD);

    for(k=pos; k<pos+n_lin; k++)
        for(i=0; i<TAM; i++)
            for(j=0; j<TAM; j++)
                C[k][i] = C[k][i] + (A[k][j] * B[j][i]);

    MPI_Send(&C[pos][0], (n_lin*TAM), MPI_DOUBLE, 0, 3, MPI_COMM_WORLD);
}
MPI_Finalize();
}
```

Verifica se é o processo escravo

Recebe do mestre a matriz A e a "faixa" do cálculo

Se for o primeiro escravo, envie B para o mestre

Envie o resultado da matriz C para o mestre

Resultados

- Resultados obtidos a partir do tempo de execução das aplicações:
 - Multiplicação de Matrizes;
 - Método dos Trapézios Composto.
- Configuração do ambiente de testes:
 - Rede *FastEthernet* 100 Mbits/sec;
 - 4 máquinas, variando de um PIII até um PentiumMMX;
 - Sistema operacional Linux com kernel 2.2.16.
- Comparações:
 - Tempo de execução seqüencial e paralelo;
 - Aplicações PVM x LAM-MPI.