Ambientes de troca de mensagens - PVM

Aleardo Manacero Jr.

Ambientes de troca de mensagens

- Programação paralela em ambientes multicomputadores exige que toda a cooperação entre computadores ocorra através de troca de mensagens
- Isso pode ser feito manualmente, através de *sockets* ou *pipes, ...*

  ou através de ambientes de troca de mensagens, como PVM, MPI, Linda...

Ambientes de troca de mensagens

- Um ambiente de troca de mensagens proporciona uma interface entre o programador e as primitivas das bibliotecas de comunicação, como os sockets
- Normalmente são também bibliotecas, porém com funcionalidades já adaptadas ao paralelismo

O PVM

- Parallel Virtual Machine, ou PVM, foi desenvolvido dentro do Oak Ridge National Laboratory, no início dos anos 90 (www.csm.ornl.gov/pvm/)
- Transforma uma rede de computadores numa máquina paralela para os programas que utilizem a biblioteca PVM

Funcionamento do PVM

- O problema a ser resolvido em paralelo é programado em C, Fortran ou Java
- Pontos de comunicação, sincronismo e demais interações entre os processos paralelos são construídos através de primitivas do PVM
- Essas primitivas são incorporadas ao programa como funções da biblioteca PVM

Compilando um programa PVM

- Como o PVM é capaz de executar em ambientes heterogêneos, então a primeira coisa a fazer é construir uma árvore de diretórios, com o raiz contendo os fontes e um subdiretório para cada tipo de máquina
- Compila-se então os fontes para cada um dos sistemas definidos

Compilando um programa PVM


```
$ gcc –o mast ../mast.c –lpvm3
$ mv mast $HOME/pvm3/bin/LINUX64
$
$ gcc –o slav ../slav.c –lpvm3
$ mv slav $HOME/pvm3/bin/LINUX64
```

Executando um programa PVM

- A execução de programas PVM exige a configuração do ambiente paralelo (nossa máquina virtual)
- Exige também a existência de processo *daemon* (pvmd) executando em cada máquina do ambiente
- O *daemon* permite o controle da máquina virtual pelo usuário

Executando um programa PVM


```
$ pvm hostfile
 pvmd already running
 pvm> conf
 1 host, 1 data format
     HOST      DTID     ARCH   SPEED
   frodo       40000 SUN4SOL2    1000
   gollum      40001 SUN4SOL2    1000
   mordor      40002 SUN4SOL2    1000
```

Executando um programa PVM


```
pvm> ps
    HOST      TID   FLAG 0x COMMAND
 frodo      40042        6/c,f pvmgs
pvm> reset
pvm> ps
    HOST      TID   FLAG 0x COMMAND
pvm>
```

Modos de paralelismo

- O PVM habilita a implementação em três modelos de paralelismo:
  - Mestre-escravo
  - Distribuição-agrupamento
  - SPMD

Modelo distribuição-agrupamento

- Segue o padrão mestre-escravo, isso é, existe um processo controlando os demais

- A diferença é que nesse modelo os dados são transmitidos por broadcasting

Comandos PVM

- Os manuais do PVM (tanto de programação quanto instalação e uso) podem ser encontrados na internet e em instalações do Linux

- Nos próximos slides aparece uma lista dos comandos usados durante os exemplos de programas PVM

Comandos PVM


```
pvm_spawn(slave,char,int,char,NUMPROC,tids);
/* dispara NUMPROC processos do tipo slave */
/* diferente do MPI, em que os processos
  paralelos são disparados externamente */
/* O segundo parâmetro fornece a lista de
  argumentos de entrada dos escravos */
/* O terceiro (PvmTaskDefault(Host)(Arch)
  indica o nó/arquitetura a ser usada */
/* O quarto especifica qual é ele(a)*/
```

Comandos PVM


```
pvm_exit();  /* finaliza processos PVM */

mytid = pvm_mytid(); /* recebe um
  identificador para o processo durante a
  execução dos programas PVM */
```

Comandos PVM

- `pvm_pkXXX(&var,count,tag);`

```
/* empacota valores do tipo XXX */
```

- `pvm_pkint(&var,count,tag);`

```
/* empacota count dados inteiros */
```

- `pvm_upkXXX(&var,count,tag);`

```
/* desempacota valores tipo XXX */
```

- `pvm_upkint(&var,count,tag);`

```
/* desempacota count dados inteiros e
  os coloca em var */
```

Comandos PVM

- `pvm_initsend(coding); /* inicia os canais de comunicação de dados com codificação do tipo PvmCodeDefault ou PvmCodeRaw */`
- `pvm_send(procid,type); /* envia os dados empacotados para procid, sendo type o código da mensagem */`
- `pvm_recv(sourceid,type); /* recebe os dados enviados por sourceid usando o código type */`

Comandos PVM

- `pvm_joingroup(group,gid); /* se integra a um grupo de processos no modelo SPMD */`
- `pvm_lvgroup(group); /* sai de um grupo de processos no processamento SPMD */`
- `pvm_barrier(group,NPROC); /* para numa barreira de sincronismo esperando por NPROC processos do grupo group */`
- `pvm_gettid(group,groupids); /* recebe os identificadores (dentro do grupo) dos demais processos */`

Outros comandos PVM

- `pvm_nrecv /* receive não bloqueante */`
- `pvm_trecv /* receive temporizado */`
- `pvm_psend /* send com empacotamento */`
- `pvm_precv /* receive c/ empacotamento */`
- `pvm_mcast /* multicast de um pacote */`
- `pvm_bcast /* broadcast de um pacote */`
- `pvm_halt  /* parada do ambiente PVM */`
- `pvm_kill  /* parada de uma tarefa */`

Exemplo mestre-escravo

- O exemplo de programa mestre-escravo aqui é apenas demonstrativo do modelo de paralelismo

- Programas reais têm um corpo maior, envolvendo situações de sincronismo e tomam, evidentemente, muito mais tempo de processamento

Mestre-escravo

- O programa mestre:

```
#include <stdio.h>
#include "pvm3.h"
#define MAXPROC 5
#define JOBS 20

main() {
 int mytid, info;
 int tids[MAXPROC];
 int tid, input, output, answers, work;
```

Mestre-escravo


```
  mytid = pvm_mytid() ;
  info = pvm_spawn("slav", (char**)0, 0, "",
                           MAXPROC, tids);

// Envia a primeira tarefa a todo trabalhador

  for(work=0; work<MAXPROC; work++)
  { pvm_initsend(PvmDataDefault);
    pvm_pkint(&work, 1, 1 );
    pvm_send(tids[work],1);  /* 1 = msgtype */
  }
```

Mestre-escravo


```
// Envia restante das tarefas a pedido
  work = MAXPROC;
  for(answers=0; answers < JOBS ; answers++)
  { pvm_recv( -1, 2 ); /* -1=any task; 2=msgtype */
    pvm_upkint(&tid, 1, 1 );
    pvm_upkint(&input, 1, 1 );
    pvm_upkint(&output, 1, 1 );
    printf("Thanks to %d 2*%d=%d\n", tid, input,
                output);
```

Mestre-escravo


```
  pvm_initsend(PvmDataDefault);
    if (work < JOBS) {/* Envia outra tarefa */
        pvm_pkint(&work, 1, 1);
        work++;   }
      else {/* Avisa que terminou */
        input = -1;
        pvm_pkint (&input, 1, 1);       }
     pvm_send(tid,1);
  } /* Final do laço for(answers…) */
  pvm_exit();
} /* Final da função main */
```

Mestre-escravo

- O programa escravo:

```
#include <stdio.h>
#include "pvm3.h"

/* Programa simples para dobrar inteiros */
main() {
 int mytid;
 int input, output;
```

Mestre-escravo


```
mytid = pvm_mytid();
 while(1) { /* recebe e desempacota dados */
   pvm_recv( -1, 1); /* -1=any task;  1=msgtype */
   pvm_upkint(&input, 1, 1);
   if ( input == -1 ) break; /* Terminei!! */
  /* executo minha tarefa caso contrário */
   output = input * 2;
```

Mestre-escravo


```
/* prepara e envia o resultado */

   pvm_initsend( PvmDataDefault );
   pvm_pkint( &mytid, 1, 1 );
   pvm_pkint ( &input, 1, 1 );
   pvm_pkint( &output, 1, 1 ) ;
   pvm_send( pvm_parent() , 2 );
  } /* Final do laço while */
  pvm_exit();
} /* Final da função main */
```

Resultados


```
Thanks to 262204 2*0=0
Thanks to 262205 2*1=2
Thanks to 262206 2*2=4
Thanks to 262207 2*3=6
Thanks to 262204 2*5=10
Thanks to 262205 2*6=12
Thanks to 262206 2*7=14
Thanks to 262207 2*8=16
Thanks to 262204 2*9=18
Thanks to 262205 2*10=20
```

```
Thanks to 262206 2*11=22
Thanks to 262207 2*12=24
Thanks to 262205 2*14=28
Thanks to 262207 2*16=32
Thanks to 262205 2*17=34
Thanks to 262207 2*18=36
Thanks to 262208 2*4=8
Thanks to 262204 2*13=26
Thanks to 262206 2*15=30
Thanks to 262205 2*19=38
```

Exemplo SPMD

Problema de transferência de calor

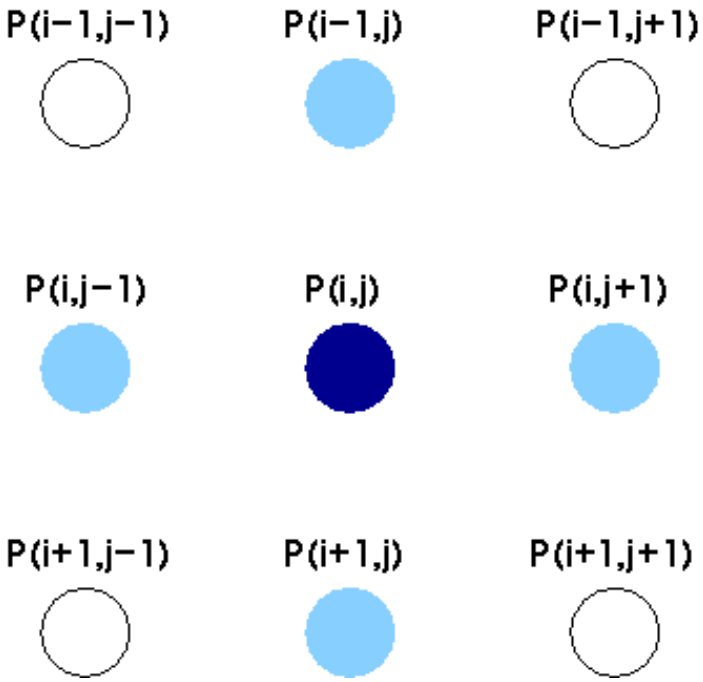
- Suponha uma chapa em que suas bordas são mantidas em temperatura constante, assim como pontos isolados da chapa
- Como determinar as temperaturas em qualquer ponto da chapa?
- Conhece algo semelhante ? ? ?

Transferência de calor

- Pode-se usar algoritmos de aproximações sucessivas, como o chamado algoritmo *red & black*
- Nesse caso, cada ponto da chapa tem sua temperatura, numa iteração, como sendo a média das temperaturas de seus vizinhos e de sua própria na iteração anterior

Transferência de calor

- Os pontos a serem considerados como vizinhos em cada iteração são os vizinhos na vertical e na horizontal, como em:



$$P(i,j) = \frac{P(i-1,j) + P(i,j-1) + P(i,j+1) + P(i+1,j) + P(i,j)}{5}$$

Transferência de calor

- Assim, a chapa pode ser vista como uma enorme matriz, em que cada elemento determina a temperatura da chapa num ponto

- Os valores de contorno e constantes são também considerados elementos da matriz

Transferência de calor

- O algoritmo Red & Black faz o cálculo das temperaturas numa iteração, considerando valores da matriz red, e armazena os novos resultados na matriz black

- Na iteração seguinte a matriz red passa a ser o destino dos resultados calculados a partir da matriz black (daí o nome do algoritmo!)

Transferência de calor

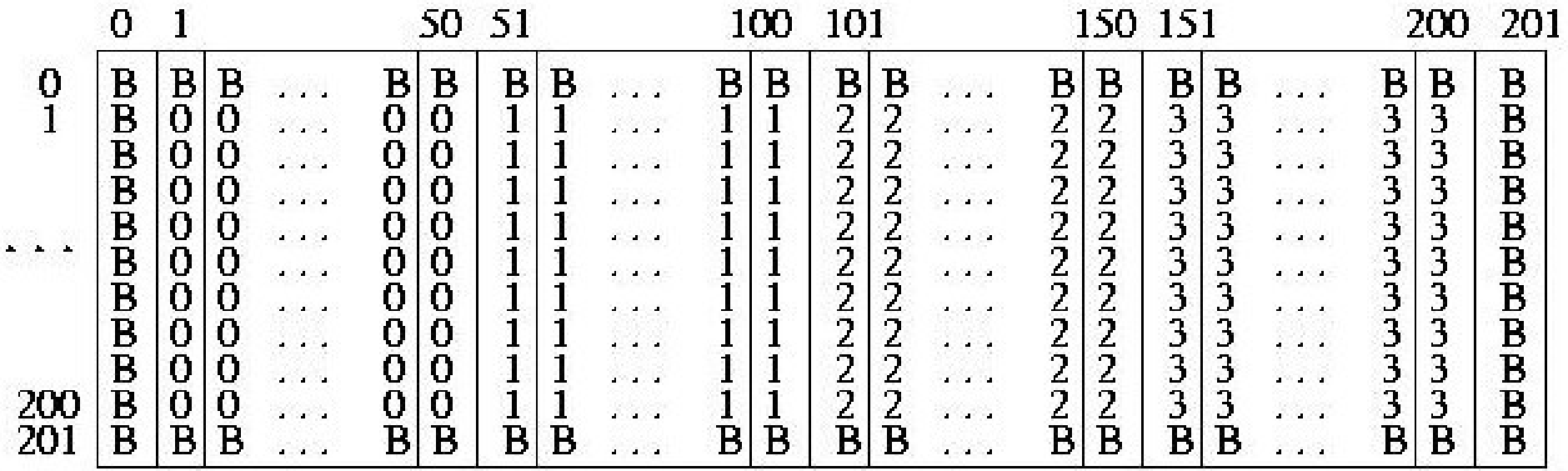
- A programação usando o modelo SPMD é feita dividindo-se a matriz em vários grupos de linhas ou colunas (dependendo da linguagem usada), um para cada processo

- Algumas dessas linhas (ou colunas), que estão nas bordas, são compartilhadas entre pares de processos (vizinhos dentro do grupo)

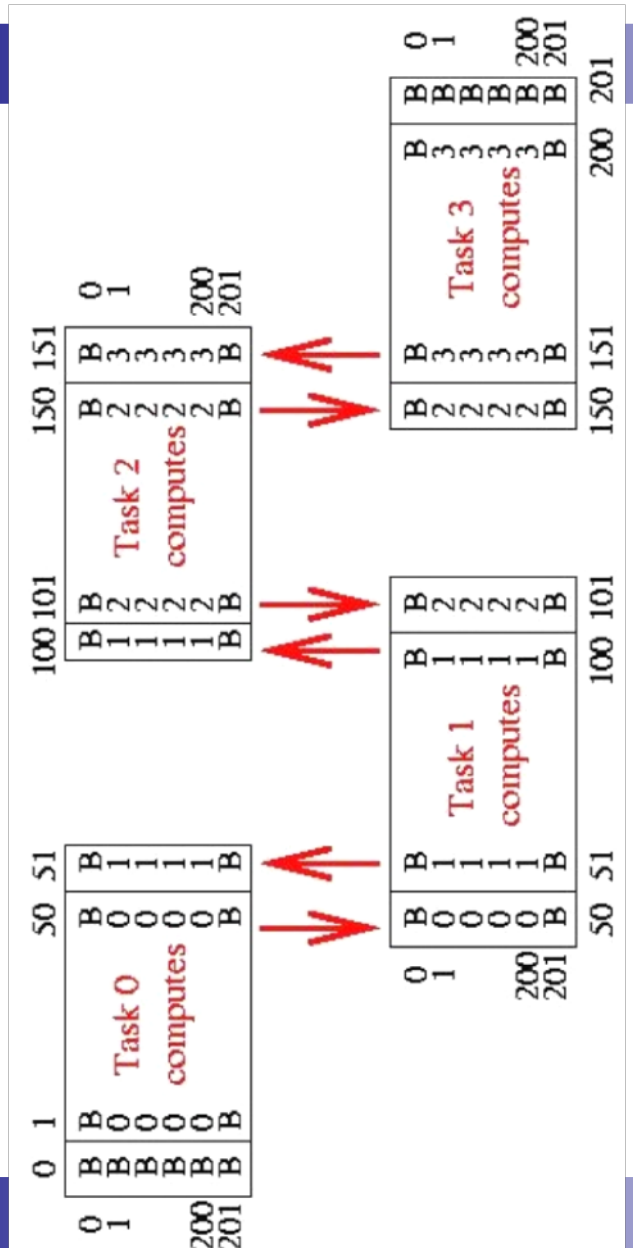
Transferência de calor

- A operação em SPMD ocorre através da criação de barreiras de sincronismo entre cada iteração
- As barreiras são criadas através da função pvm_barrier()
- As informações sobre as linhas compartilhadas são trocadas entre os processos vizinhos a cada iteração

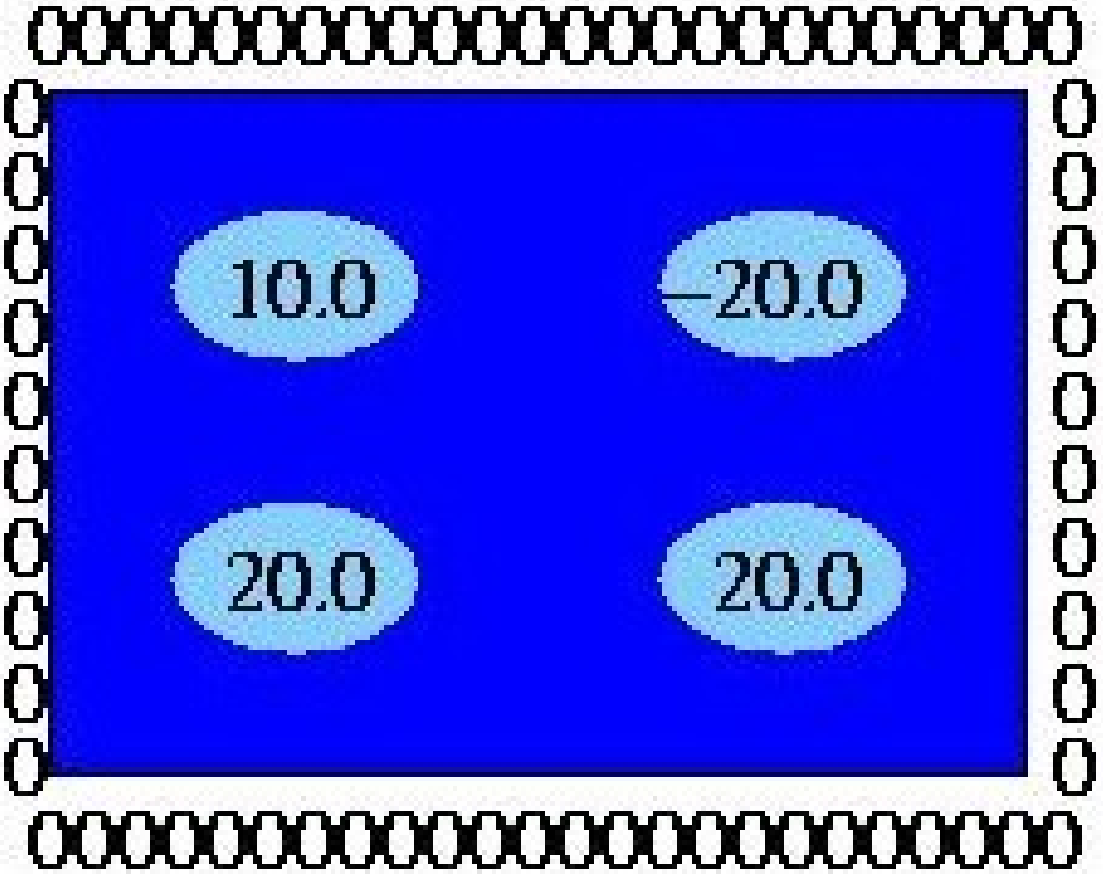
Transferência de calor



Transferência de calor



Exemplo de chapa



Código.....


```
#include “stdio.h”
#include “../include/pvm3.h”
#define NPROC 4
#define MAXTIME 200
#define COLS 200
#define TOTROWS 200
#define ROWS (TOTROWS/NPROC)+3

double RED[ROWS+2][COLS+2],BLACK[ROWS+2][COLS+2];
int TIDS[NPROC-1], OFFSET, IERR, INFO;
int TICK, MAXTIME, IAMFIRST=0, IAMLAST=0;
char FNAME[30];
```

Código.....


```
main() {int I, R, C, INUM;
/* Junta-se ao grupo pheat para iniciar o SPMD */
   INUM = pvm_joingroup(“pheat”);
/* Se sou o primeiro no grupo, então crio pares */
   if ( INUM == 0 )
     for (I=1; I<NPROC; I++)
       INFO = pvm_spawn(“pheat”,0,PvmTaskDefault,
                                ”anywhere”,1,TIDS[I]);
/* Barreira para ter certeza de que todos os processos
já começaram */
      pvm_barrier( “pheat”, NPROC );
```

Código.....


```
/* Identifica parceiros e pega seus TIDs para poder
 enviar dados */
    for (i=0; I<NPROC; I++)
        TIDS[I] = pvm_gettid(“pheat”, I);

/*   Linha 0 = Recebida do vizinho de cima          *
 *   Linha 1 = Enviada para vizinho de cima         *
 *   Linhas 1..mylen = Minha área de cálculo        *
 *   Linha mylen = Enviada para vizinho de baixo    *
 *   Linha mylen+1 = Recebida do vizinho de baixo   */
```

Código.....


```
    if (INUM == 0) IAMFIRST = 1;
    if (INUM == NPROC-1) IAMLAST = 1;
    OFFSET = (ROWS/NPROC * INUM );
    MYLEN = ROWS/NPROC;
    if ( IAMLAST )
          MYLEN = TOTROWS - OFFSET ;
    printf("INUM: %d  Local: 1 %d Global: %d %d\n",

                INUM, MYLEN, OFFSET+1, OFFSET+MYLEN);
/* Começa Frio  */
    for (R=0; R<=ROWS+1; R++)
      for (C=0; C<=COLS+1; C++)
          BLACK[R][C] = 0.0;
```

Código.....


```
/* Laço para contar os ciclos executados */
    for (TICK=1; TICK<=MAXTIME; TICK++)
      {
/* Determina as fontes constantes de calor */
        STORE (BLACK,OFFSET,MYLEN,ROWS/3,
               TOTCOLS/3,10.0,INUM);
        STORE (BLACK,OFFSET,MYLEN,2*ROWS/3,
               TOTCOLS/3,20.0,INUM)
        STORE (BLACK,OFFSET,MYLEN,ROWS/3,
               2*TOTCOLS/3,-20.0,INUM)
        STORE (BLACK,OFFSET,MYLEN,2*ROWS/3,
               2*TOTCOLS/3,20.0,INUM)
```

Código.....


```
/* Envia para cima e para baixo (se não estou na ponta) */
     if ( ! IAMFIRST )
     { pvm_initsend(PvmDataDefault);
       pvm_pkdouble(BLACK[1][0], COLS, 1 );
       pvm_send( TIDS[INUM-1], 1);
     }
     if ( ! IAMLAST )
     { pvm_initsend(PvmDataDefault);
       pvm_pkdouble(BLACK[MYLEN][0], COLS, 1 );
       pvm_send( TIDS[INUM+1], 2);
     }
```

Código.....


```
/* Recebe primeiro de baixo de depois de cimam*/
  if ( ! IAMLAST )
    { pvm_recv ( TIDS[INUM+1], 1, BUFID );
      pvm_upkdouble(BLACK[MYLEN+1][0], ROWS, 1);
    }

  if ( ! IAMFIRST )
    { pvm_recv ( TIDS[INUM-1], 2, BUFID );
      pvm_upkdouble(BLACK[0][0], ROWS, 1);
    }
```

Código.....


```
/* Cálculo o fluxo em uma iteração   */
   for (R=1; R<= MYLEN; R++)
     for (C=1; C<= ROWS; C++)
       RED[R][C] = ( BLACK [R][C] +
              BLACK [R][C-1] + BLACK [R-1][C] +
              BLACK [R+1][C] + BLACK [R][C+1] ) / 5.0;
```

Código.....


```
/* Copia a matriz – Normalmente faria-se uma
 versão black do laço */
        for (R=1; R<=MYLEN; R++)
          for (C=1; C<=COLS; C++)
            BLACK[R][C] = RED[R][C];
}
/* Fim do laço principal
    “for (TICK=1; TICK<=MAXTIME); TICK++)” */
```

Código.....


```
SENDCELL(OFFSET,MYLEN,INUM,TIDS(0),ROWS/2,TOTCOLS/2);
/* Despeja dados para verificação   */
if ( ROWS <= 20 )
   {FILE *fp;
     strcpy(FNAME,”/tmp/pheat.out”);
     fp = fopen(FNAME,”w”);
     for (R=1; R<=MYLEN; R++)
       fprintf(fp,“%f12.6 ”,BLACK[R][1]);
     fclose(fp);
   }
```

Código.....


```
/* Espera todos os processos terminarem   */
   pvm_barrier ( “pheat”, NPROC );
   pvm_exit ( );
} /* Fim da função main */
```

Código.....


```
void SENDCELL(int OFFSET, int MYLEN, int INUM,
                  int PTID, int R, int C)
{double CENTER;
 int I;

/* Verifica se linha a ser impressa é desse processo */
      I = C – OFFSET;
```

Código.....


```
if ( I >= 1 && I <= MYLEN )
  if ( INUM == 0 )
      printf("Master has %lf %d %d %d",
                RED[R][I], R, C, I);
    else
       { pvm_initsend (PvmDataDefault);
         pvm_pkdouble( RED[R][I], 1, 1 );
         printf("INUM: %d Returning %d %d %lf
                  %d",INUM,R,C,RED[R][I],I );
         pvm_send( PTID, 3 )
       }
```

Código.....


```
else
  if ( INUM == 0 )
    { pvm_recv( -1 , 3, BUFID )
      pvm_upkdouble ( CENTER, 1, 1 )
      printf("Master Received %d %d %lf",R,C,CENTER);
    }
return();
}
```

Código.....


```
void  STORE (int OFFSET, int MYLEN, int R, int C,
double VALUE, int INUM)
{int I;
  I = R – OFFSET;
  if ( I > 0 && I < MYLEN )
      BLACK[R][I] = VALUE;
  return();
}
```

Comentários finais

- O PVM, assim como o MPI, permite a construção de programas paralelos para ambientes distribuídos com uma certa simplicidade
- A maior ou menor eficiência de um programa escrito usando bibliotecas de troca de mensagens depende do seu particionamento, granulosidade, e experiência do projetista