TREINAMENTO:

INTRODUÇÃO AO PVM

ÍNDICE

1 - INTRODUÇÃO & CONCEITOS

1.1 - COMPUTAÇÃO DE ALTO DESEMPENHO

"Integração de vários conceitos de *hardware* e *software* no intuito de se resolver grandes desafios em computação num menor período de tempo."

Hardware:
- Processadores: 32, 64 bits
- Memória RAM, CACHE
- "Clock"
- Processadores Vetoriais
- Processadores RISC
- Processadores em Paralelo
- Memória Distribuída

Software:
- "Time-sharing"
- Processamento Distribuído
- Sistemas Operacionais
- Compiladores Vetoriais

"Alta performance não depende exclusivamente na utilização de "hardware" mais eficiente, mas inclui novas técnicas de processamento."

- Programação Vetorial
- Técnicas de Otimização
- Pré-processadores
- Programação Paralela
- "Message-Passing Libraries"

1.2 - OBJETIVOS

Velocidade - Reduzir o tempo de processamento.

Desempenho - Vários agentes operando junto:

- Processadores Vetoriais
- Processadores Paralelos
- Processador Gráfico
- Memória cache

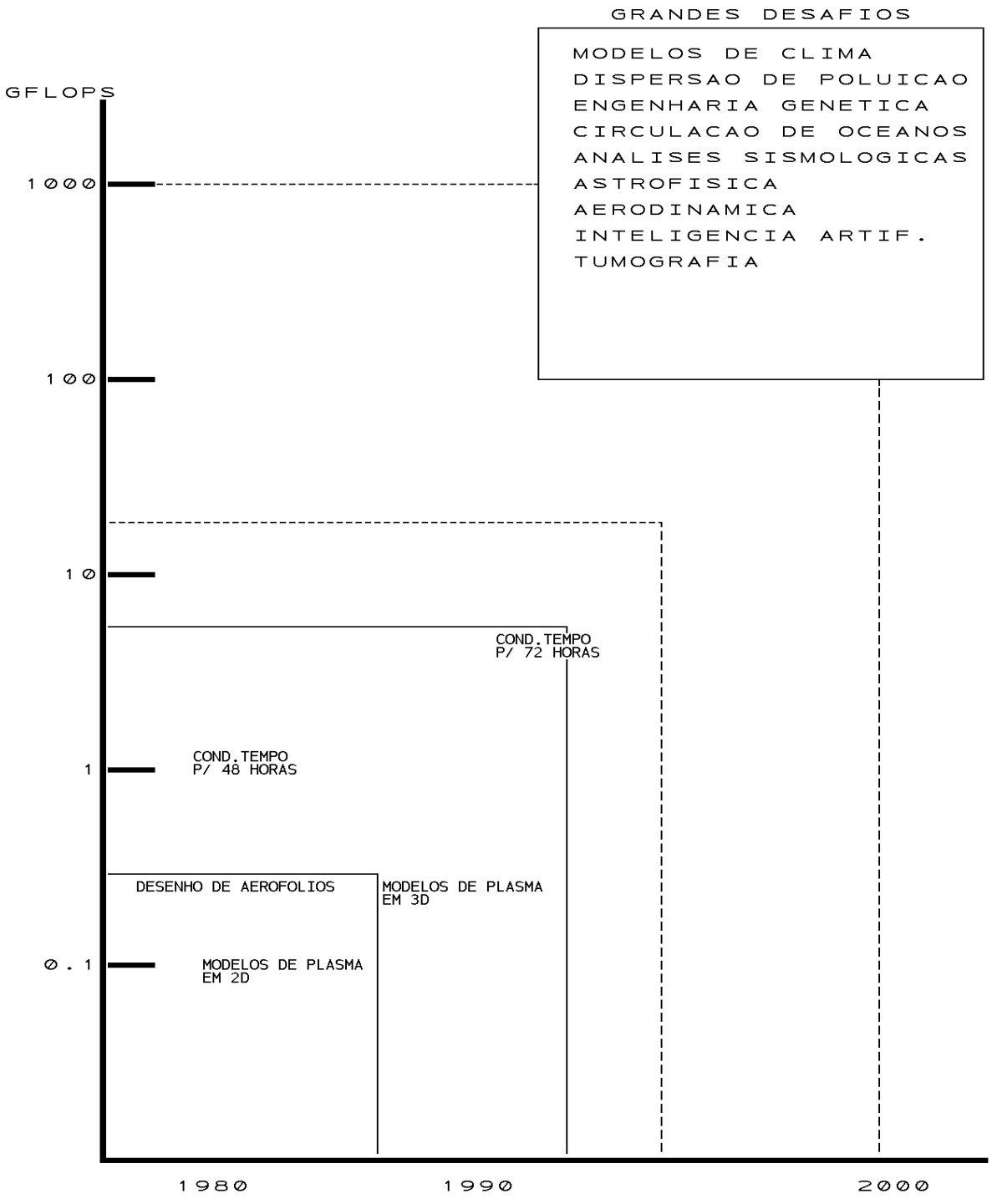
Modularidade - Possibilidade de expansão:

- Mais processadores
- Mais memória
- Mais disco

Desafios computacionais- Manipulação de imensas bases de dados:

- Simulações de climas
- Circulação de correntes marítimas
- Análise de Elementos Finitos
- Aerodinâmica
- Inteligência Artificial
- Exploração Sismológica
- Modelagem de Reservas de Óleo
- Força da Fusão do Plasma
- Engenharia Genética
- Mecânica Quântica
- Química dos Polímeros
- Dinâmica dos Fluidos

PERFORMANCE PARA GRANDES DESAFIOS
COMPUTACIONAIS



1.3 - CONCEITOS BÁSICOS

"Task" - Uma tarefa lógica de um trabalho computacional.

"Parallel Tasks" - São tarefas lógicas que independem uma das outras, sendo assim, todas poderão ser executadas simultaneamente, dando resultados corretos.

"Shared Memory" - Ambiente com vários processadores onde cada um compartilha de uma única memória central.

"Distributed Memory" - Ambiente com vários processadores onde cada um possui sua própria memória.

Execução Serial - É a execução de um programa sequencialmente, uma instrução por vez.

Execução Paralela - É a execução de um programa, no qual, diversas tarefas ("Parallel Tasks") são distribuidas por entre vários processadores, sendo todas, executadas simultaneamente.

"Message-Passing" - Método de comunicação utilizado em processamento paralelo. Se baseia na transmissão de dados (*send/Receive)*, via uma rede de interconexão, seguindo as regras de um protocolo de rede.

1.4 - IDÉIAS DE PARALELISMO

- Problema que pode ser ***paralelizado***.

"**Calcular o potencial energético para cada grupo de milhares de combinações independentes de uma molécula;** quando feito, achar a conformação mínima".

- Problema que ***não*** pode ser ***paralelizado***

"Cálculo de Séries de Fibonacci"

(1, 1, 2, 3, 5, 8, 13, 21, ...)

F(K+2) = F(K+1) + F(K)

- Tipos de paralelismo:

Paralelismo de Dados: Programa paralelo, no qual, o mesmo código é executado em diversos processadores, mas utilizando dados diferentes.

Paralelismo Funcional: Programa paralelo, no qual, diferentes códigos são processados em diversos processadores, utilizando um, ou diversos conjuntos de dados.

- Sincronização

É a coordenação na execução de tarefas que estão sendo executadas em paralelo, num determinado instante, aguardam a finalização mutua para coordenar os resultados ou trocar dados e reinicializar novas tarefas em paralelo.

OBS: **Sincronização** pode implicar em acréscimos no tempo total de execução de um programa.

Sincronização, normalmente, implica na maioria dos custos e problemas em processamento paralelo.

- "Parallel Overhead"

É o tempo dispensado para coordenar tarefas que executam em paralelo. Ex.:

- Tempo para inicializar uma tarefa
- Tempo para finalizar uma tarefa
- Tempo para sincronizar tarefas
- Tempo de comunicação entre tarefas

- "Granularity"

  É uma medida da razão entre a quantidade de tarefas computacionais realizadas numa tarefa paralela, pela quantidade de comunicação efetuada.

  "Fine-Grained" (Grânulos Finos)

  Pouca tarefa computacional e **alta** comunicação.

  "Coarse-Grained" (Grânulos Grosseiros)

  Alta tarefa computacional e **pouca** comunicação.

- "Massively Parallel System"

  Sistema paralelo com muitos processadores. Acima de 1000 processadores.

- "Scalable Parallel System"

  Sistema paralelo, no qual, a adição de processadores para execução de um programa paralelo, pode proporcionar um aumento de performance na execução. Depende do *Hardware*, do *Algorítmo Paralelo* e do *Código do Usuário*.

1.5 - CLASSIFICAÇÃO DE ARQUITETURAS

Método de Flynn - 1966

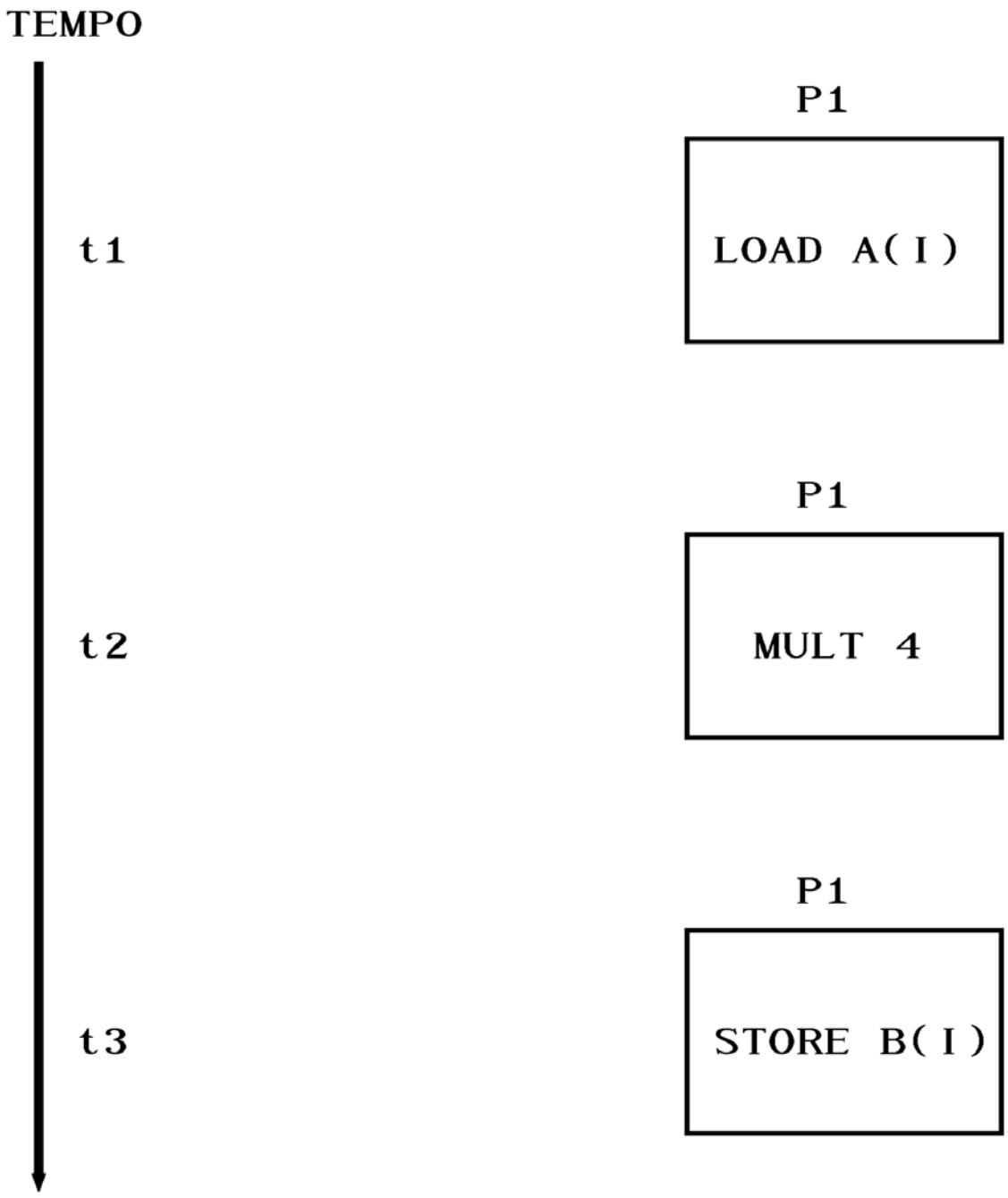
Avalia a relação entre o número de instruções de um programa com os dados.

SISD ("**S**ingle **I**nstruction, **S**ingle **D**ata")

- Maioria dos computadores convencionais;

- Uma instrução é executada a cada ciclo em um único dado;

B(I) = A(I) * 4	**LOAD A(I)** **MULT 4** **STORE B(I)**


TEMPO
t1
t2
t3
P1
LOAD A(I)
P1
MULT 4
P1
STORE B(I)

SIMD ("**S**ingle **I**nstruction, **M**ultiple **D**ata")

- Uma instrução é executada a cada ciclo em mais de um dado;

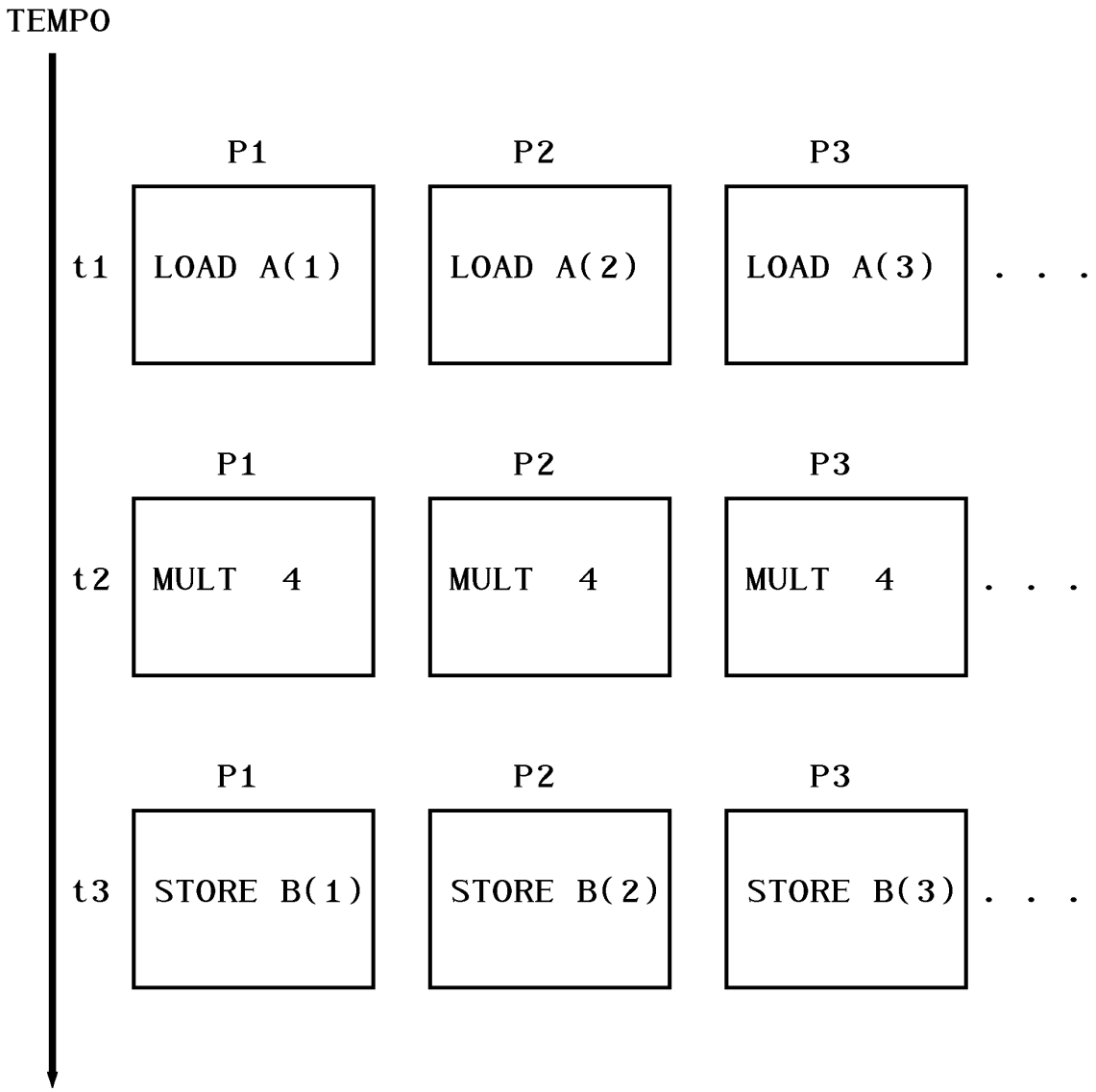
- As instruções são sincronizadas;

- Processadores vetoriais e paralelos;

- **EX.:** Cray1, NEC SX-2, Fujitsu VP, IBM9000, CM-2.

B(I) = A(I) * 4

```
LOAD A(I)
MULT  4
STORE B(I)
```

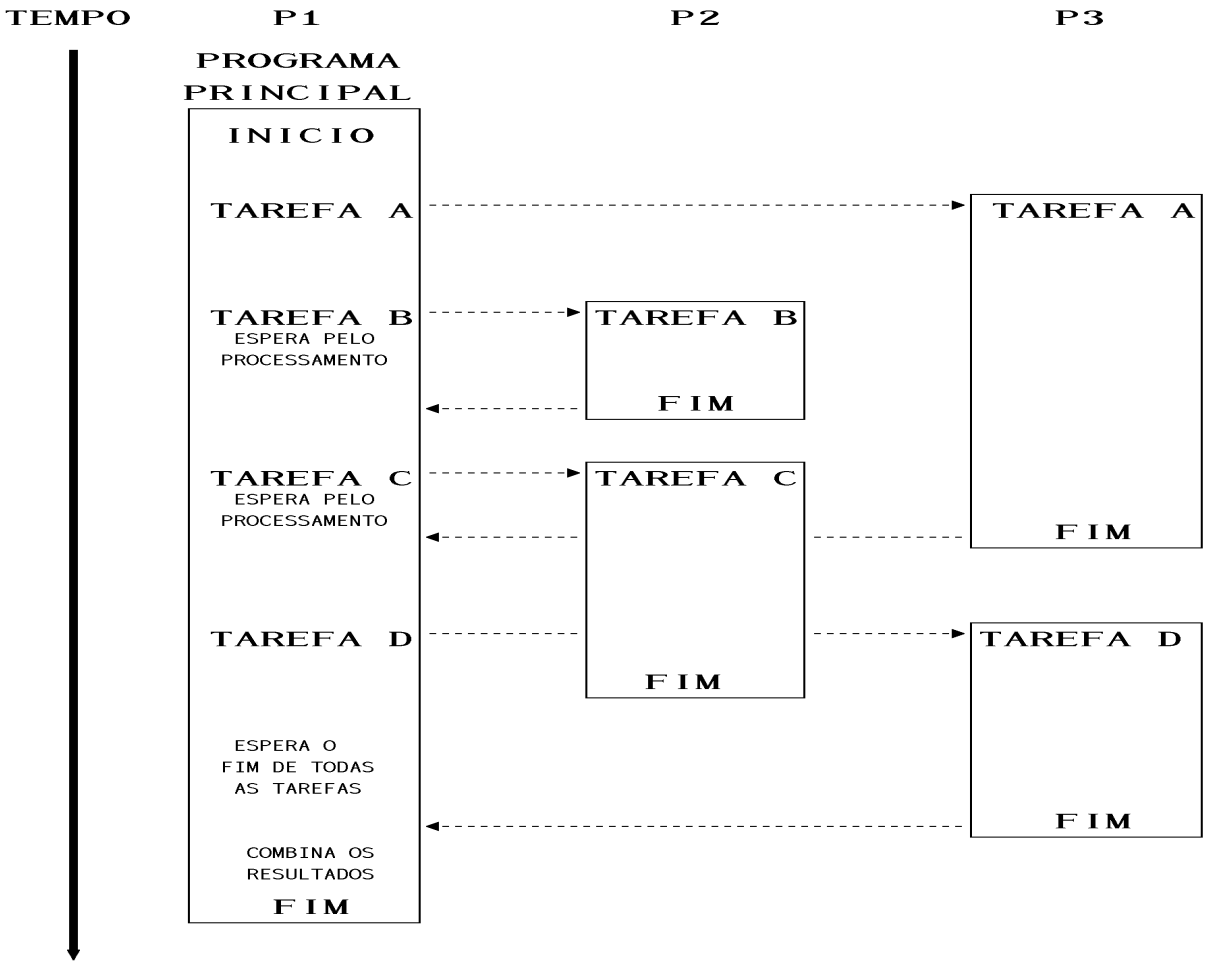

TEMPO
P1
P2
P3
t1
LOAD A(1)
LOAD A(2)
LOAD A(3)
. . .
P1
P2
P3
t2
MULT 4
MULT 4
MULT 4
. . .
P1
P2
P3
t3
STORE B(1)
STORE B(2)
STORE B(3)
. . .

MIMD ("**M**ultiple **I**nstruction, **M**ultiple **D**ata")

- Paralelismo efetuado conectando-se multiplos processadores;

- Cada processador executa um conjunto de instruções em um conjunto de dados, independente dos outros processadores.

-Ex: IBM RS6000 Cluster, IBM SP2, Cray C90, KSR1, nCUBE, CM-5, iPSC/2


TEMPO
P1
P2
P3
PROGRAMA PRINCIPAL
INICIO
TAREFA A
TAREFA B
ESPERA PELO PROCESSAMENTO
TAREFA C
ESPERA PELO PROCESSAMENTO
TAREFA D
ESPERA O FIM DE TODAS AS TAREFAS
COMBINA OS RESULTADOS
FIM
TAREFA B
FIM
TAREFA C
FIM
TAREFA A
FIM
TAREFA D
FIM

1.6 - CUSTOS

Tempo dispensado em analisar o código para paralelizar;

Tempo dispensado para recodificar o programa;

Tempo para depuração;

Perda de portabilidade;

Aumento do "Overhead";

- início e término das "tasks".
- sincronização das "tasks".
- comunicação entre as "tasks".

Aumento do tempo total de CPU;

Aumento da necessidade de memória pelo programa;

Longa espera por melhores resultados.

2 - INTRODUÇÃO AO PVM

2.1 - O QUE É PVM ?

Parallel **V**irtual **M**achine

Biblioteca de rotinas, utilizada para efetuar a comunicação entre processos paralelos.

Opera em ambientes heterogêneos de máquinas.

Opera em diversos tipos de rede, desde que, possuam o protocolo IP.

Comunicação por **"message-passing"**.

Para operação e execução, é definido dois componentes básicos:

- O processo pvm (**daemon pvmd3**)
- Bibliotecas (**libpvm3.a, libfpvm3.a, libgpvm3.a**)

Pode trabalhar com **FORTRAN** ou **C**.

Software de domínio público desenvolvido por **Oak Ridge National Laboratory** em 1989.

2.2 - HISTÓRICO

PVM 1.0 - Oak Ridge National Laboratory - Verão de 1989
Apenas para testes de laboratório.

PVM 2.0 - University of Tennessee - Fevereiro de 1991
Versão difundida para uso geral.

PVM 3.0 - Fevereiro de 1993
Alterado os nomes das rotinas. Incompatibilidade com a versão anterior.

PVM 3.3.8 - Adicionado os métodos de comunicação entre processos utilizando MPI, arquitetura SP2MPI.

PVM 3.3.11 - Setembro de 1996, última versão.

2.3 - PORQUE USAR PVM ?

Promessa de efetuar computação paralela, utilizando-se de qualquer conjunto de computadores, disponíveis em laboratórios;

Reduz o tempo total de execução de um programa **("wall clock")**;

Paralelização escalável;

Fácil de instalar e usar;

Software de domínio público;

Grande aceitação e utilização;

Flexível:

- Variedade de arquiteturas;
- Variedade de redes de trabalho;
- Programação em FORTRAN e C;
- De fácil atualização;

Ferramentas de auxílio para desenvolvimento de programs que utilizem o pvm.

2.4 - COMPONENTES DO PVM

2.4.1 - PVM daemon

Processo **pvmd3**.

Processo no ambiente unix que inspeciona a operação de um processo do usuário, que contém uma aplicação com pvm. Coordena as comunicações entre os processadores e o sistema pvm.

Em cada máquina inicializada para executar uma aplicação com o pvm, roda um **"daemon" pvmd3**.

Cada usuário possuirá seu próprio **"daemon" pvmd3**.

O esquema de controle para os **"daemons"**, baseia-se na idéia de : daemon mestre-daemon escravo ou local-remoto.

Cada **"daemon"** mantêm uma tabela de configuração e processa a informação relativa a aplicação pvm do usuário.

Os processos criados numa aplicação pvm se comunicam entre-si através dos **"daemons"**.

Cada máquina deve possuir sua própria versão de **pvmd3** instalada, que é dependente da arquitetura do sistema.

2.4.2 - BIBLIOTECA PVM

libpvm3.a - Biblioteca de rotinas para interface com a linguagem **C**.

libfpvm3.a - Biblioteca de rotinas para interface com a linguagem **FORTRAN**.

libgpvm3.a - Biblioteca de rotinas necessária para se trabalhar com o conceito de grupos dinâmicos (Sincronização de processos).

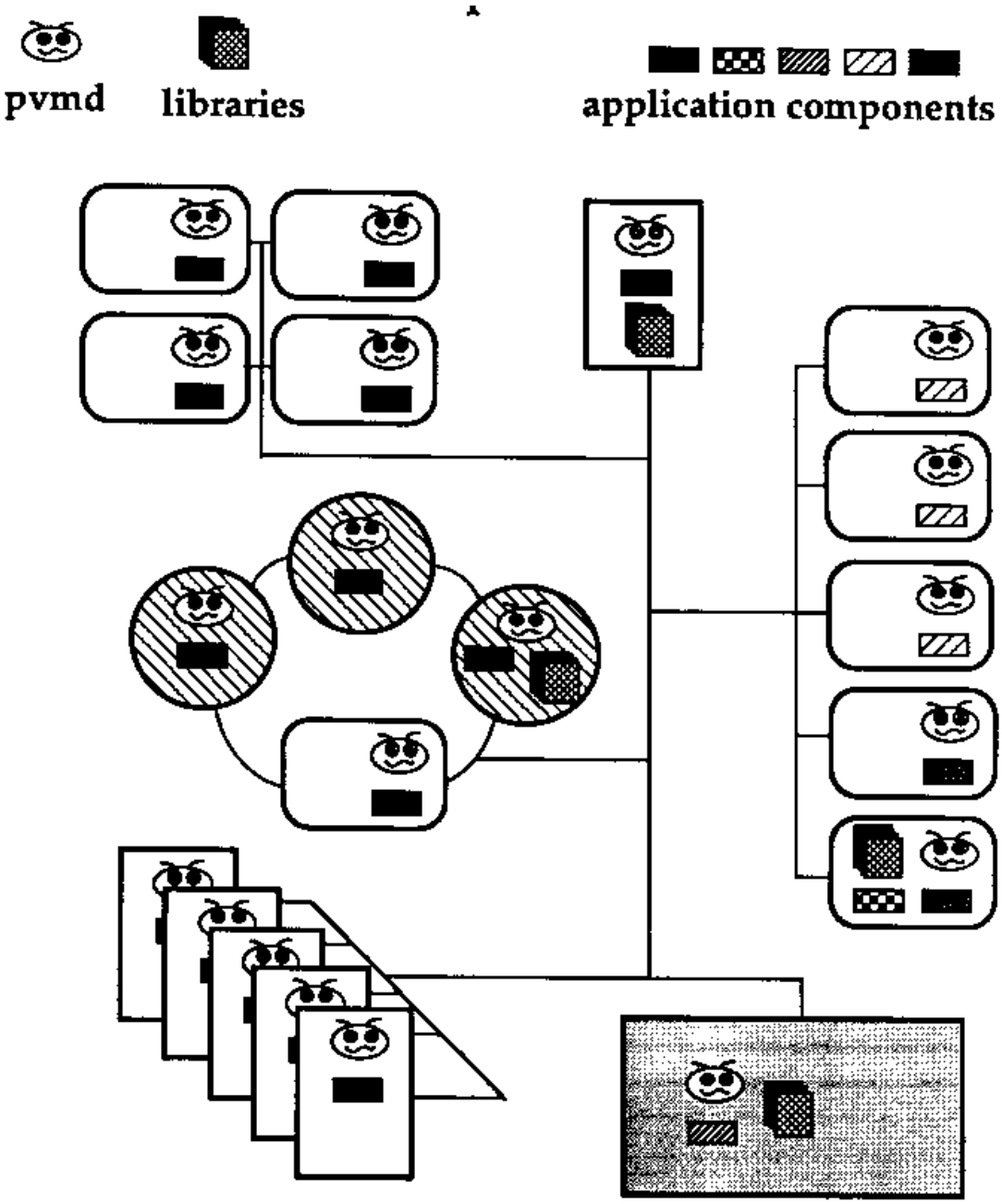
Simples chamadas as rotinas, que devem ser inseridas no código fonte dos programas que se propõem a trabalhar em paralelo. Estas rotinas te possibilitam:

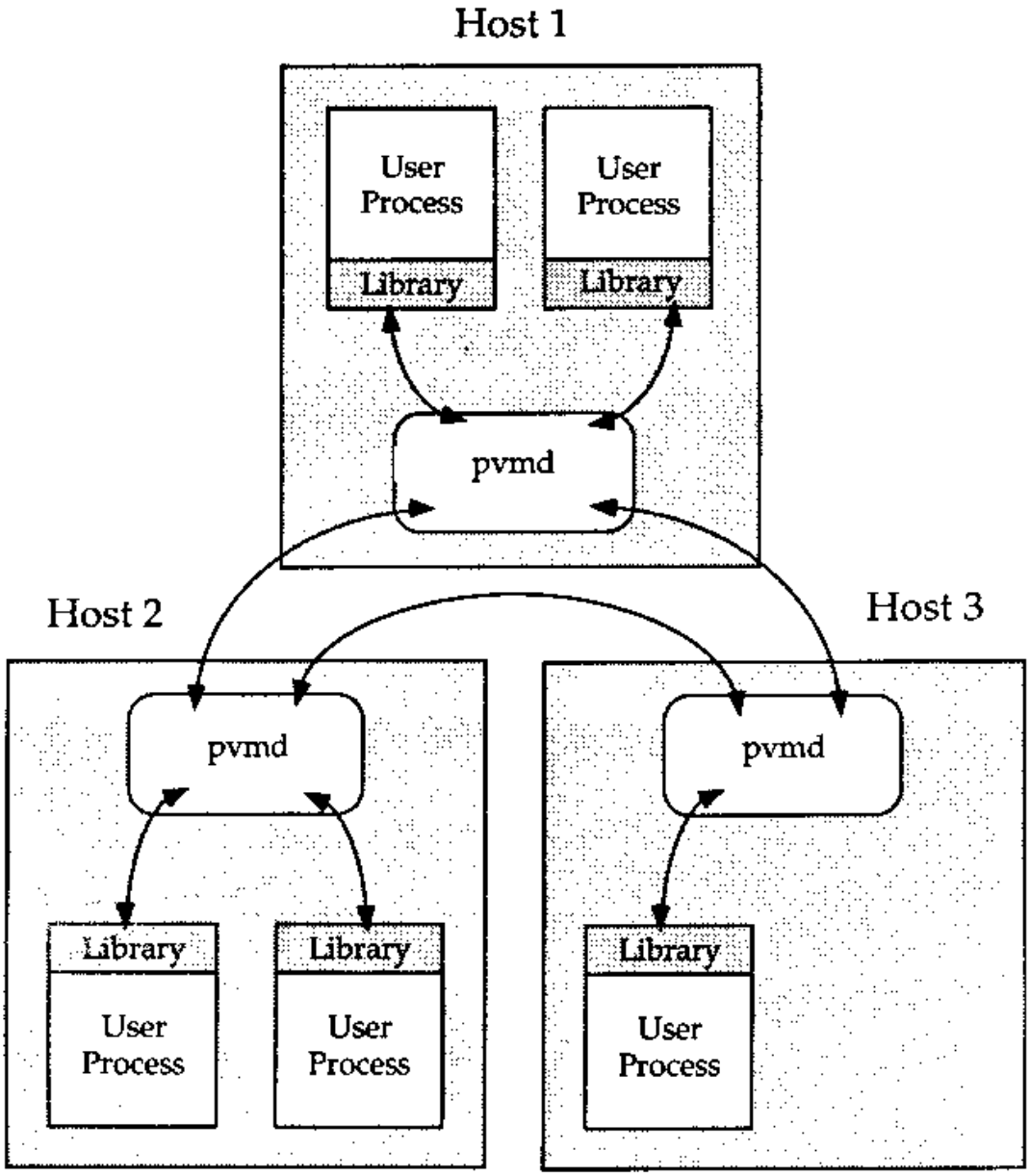
- Iniciar e terminar processos;
- Codificar, enviar e receber mensagens;
- Sincronização, por inserção de barreiras;

A comunicação entre os processos é , normalmente, feita entre os **"daemon"**.

Quando a transmissão é feita entre máquinas de arquiteturas diferentes é feita uma conversão dos dados pelo formato **xdr**, automáticamente.

2.5 - ARQUITETURA PVM




Host 1
User Process
Library
User Process
Library
pvmd
Host 2
Host 3
pvmd
pvmd
Library
User Process
Library
User Process
Library
User Process

2.6 - CONFIGURAÇÃO DO PVM

Variáveis de ambiente:

PVM_ARCH Define a arquitetura da máquina.

setenv PVM_ARCH LINUX

PVM_ROOT Define a localização do diretório **pvm3** instalado na máquina.

setenv PVM_ROOT /home/soft/pvm3

OBS: Essas variáveis deverão ser definidas no arquivo **.cshrc** de cada usuário.

Diretório de trabalho do PVM:

~/pvm3/bin/LINUX Diretório padrão do PVM, aonde devem ficar os arquivos executáveis do usuário.

Acesso as máquinas da arquitetura PVM:

Arquivo **.rhosts** Arquivo com o nome das máquinas que compoêm uma arquitetura PVM. Definido no *"home"* de cada usuário.

2.7 - UTILIZAÇÃO DO PVM

2.7.1 - ADAPTAÇÃO DO PROGRAMA

Depure o programa e verifique quais os **procedimentos, loops e rotinas** que mais consomem cpu;

Analise e avalie se esses **procedimentos, loops e rotinas,** podem ser executados concorrentemente em outros processadores;

Verifique a necessidade de transmissão de dados entre esses procedimentos (*enviar/receber* dados)

Verifique se existe a necessidade de sincronização (espera por algun resultado);

Verifique se podem surgir dependências para uma específica arquitetura (precisão, memória, ...);

Inclua as rotinas apropriadas das bibliotecas pvm em seu programa fonte.

2.7.2 - COMPILAÇÃO

Copie os programas fontes ou edite-os no diretório de trabalho do pvm (**~/pvm3/bin/LINUX**).

Na compilação dos programas será sempre necessário incluir a biblioteca pvm **libpvm3.a**. No caso de programas em FORTRAN, incluir tambêm, a biblioteca **libfpvm3.a**

% **f77 -o <programa> <programa.f>**
-I/usr/local/pvm3/include -L/usr/local/pvm3/lib -lfpvm3 -lpvm3

% **cc -o <programa> <programa.c>**
-I/usr/local/pvm3/include -L/usr/local/pvm4/lib -lpvm3

O usuário pode editar um arquivo executável com os comandos de compilação e linkedição para facilitar o seu trabalho. Este arquivo é denominado de **"makefile"**.

EXEMPLO DE "MAKEFILE"

```
# ARQUIVO: <NOME>
# DESCRIÇÃO: <DESCRIÇÃO>
# VERSÃO DO PVM:  3.3.7
# AUTOR: <NOME>

COMP   =  <COMPILADOR A SER UTILIZADO>
EXEC   =  <NOME DO PROGRAMA EXECUTÁVEL>
FONTE  =  <NOME DO PROGRAMA FONTE>
INC    =  -I/home/soft/pvm3/include
BIBS   =  -L/home/soft/pvm3/lib -lfpvm3 -lpvm3

doit :   ${EXEC}

${EXEC} : ${FONTE}
    ${COMP} ${FONTE} ${INC} ${BIBS} -o ${EXEC}
```

Para executar o arquivo "makefile", basta dar o comando:

% **make -f <nome do arquivo "makefile">**

2.7.3 - INICIALIZAÇÃO DO AMBIENTE PVM

O ambiente PVM pode ser inicializado a partir da definição de um arquivo de máquinas.

Esse arquivo tem por conteúdo, o **endereço internet** de cada máquina e **opções** de execução, na máquina:

opções **dx=** Localização do arquivo **pvmd.** Por "default" o arquivo pvmd será criado no diretório **/tmp**, com o nome de pvmd.<uid>.

ep= Localização dos arquivos executáveis do usuário. Por "default", todo arquivo executável deverá estar no diretório **~/pvm3/bin/LINUX**.

Ex.:

% vi maquina1

```
spirit.cna.unicamp.br          dx=~/teste  ep=~/exec
mafalda.cna.unicamp.br               dx=~/teste
                                     ep=~/exec
snoopy                         ep=~/exec/fortran
calvin
```

2.7.4 - EXECUÇÃO

Para executar o programa adaptado ao pvm, será necessário inicializarmos o **"daemon" pvmd3** nas máquinas escolhidas para a paralelização.

OBS: Este comando deverá ser executado sempre, em "background" (&).

% **pvmd3 <arquivo máquinas> &**

Somente um único **"daemon" pvmd3** deverá ser executado em cada máquina para um mesmo usuário.

Execute o seu programa.

% **<programa executável>**

2.7.4 - FINALIZAÇÃO DO AMBIENTE PVM

ATENÇÃO:Finalizações anormais do pvm poderão deixar arquivos no diretório **/tmp**, que por sua vez impedirão a inicialização dos **"daemons" pvmd3** novamente. Será necessário apagar manualmente todos os arquivos do diretório **/tmp/pvm*.<uid>** em todas as máquinas que foi inicializado os **"daemons" pvmd3.**

```
% id -u
10045
% rm  /tmp/pvm*.10045
```

Existe disponível no ambiente CENAPAD-SP um programa que possibilita a eliminação dos "daemons" pvmd3 sem ser necessário acessar cada um das máquinas da configuração pvm utilizada. Este programa se encontra no seu diretório de trabalho (**clean).**

```
% clean <arquivo de máquinas>
```

OBS: Lembre-se de um detalhe, a máquina que voce "logou" sempre fará parte da configuração de "daemons" pvm, mesmo que ela não esteja presente no seu arquivo de máquinas. Sendo assim é provável que a execução do **clean** não funcione completamente, sendo necessário executar o procedimento manual para a máquina logada.

2.8 - CONSOLE PVM

O pvm possui uma alternativa de auxílio na criação, deleção, monitoração e execução de um programa, chamada de **console do pvm**.

Execute o comando **pvm** para inicializar a console.

% **pvm**
pvm>

ou

% pvm <arquivo de máquinas>
pvm>

Comandos auxiliares:

conf Permite visualizar a configuração de máquinas.

quit Cancela a console pvm sem cancelar a configuração.

halt Cancela a console pvm, e também cancela toda a configuração, cancelando a execução dos programas.

add Adiciona uma ou mais máquinas a configuração.

delete Elimina uma ou mais máquinas da configuração..

reset Elimina todos os processos PVM existentes e limpa todas as tabelas internas do PVM.

spawn Possibilita executar um programa a partir da console.

-maq Executa o processo mestre na máquina especificada.

-> Redireciona as saídas dos processos para a console.

->arq Redireciona as saídas dos processos para um arquivo.

2.9 - EXEMPLO DE UM PROGRAMA COM PVM

PROGRAMA MESTRE (FORTRAN)

```
C**********************************************************
C ARQUIVO: hello.master.f
C DESCRIÇÃO: Exemplo simples de programa mestre
C AUTOR: Blaise Barney
C**********************************************************
      program  hello_master
      include  'fpvm3.h'

      parameter  (NTASKS=6)
      parameter  (HELLO_MSGTYPE=1)
      integer mytid, tids(NTASKS), i, msgtype, info
      character*12  helloworld/'HELLO WORLD!'/

      print *, ' Enrolling master task in PVM . . . '
      call pvmfmytid (mytid)
      print *, ' Spawning worker tasks . . . '
      do 10 i=1, NTASKS
      call pvmfspawn ("hello.worker", PVMDEFAULT, " ", 1, tids( i ), info)
      print *, ' Spawned worker task id = ', tids( i )
10    continue
      print *, ' Sending message to all worker tasks '
      msgtype=HELLO_MSGTYPE
      call pvmfinitsend ( PVMDEFAULT, info )
      call pvmfpack ( STRING, helloworld, 12, 1, info )
      do 20 i=1, NTASKS
      call pvmfsend (tids( i ), msgtype, info )
20    continue
      print *, ' All done.  Leaving  hello.master '
      call pvmfexit (info)
      end
```

PROGRAMA ESCRAVO (FORTRAN)

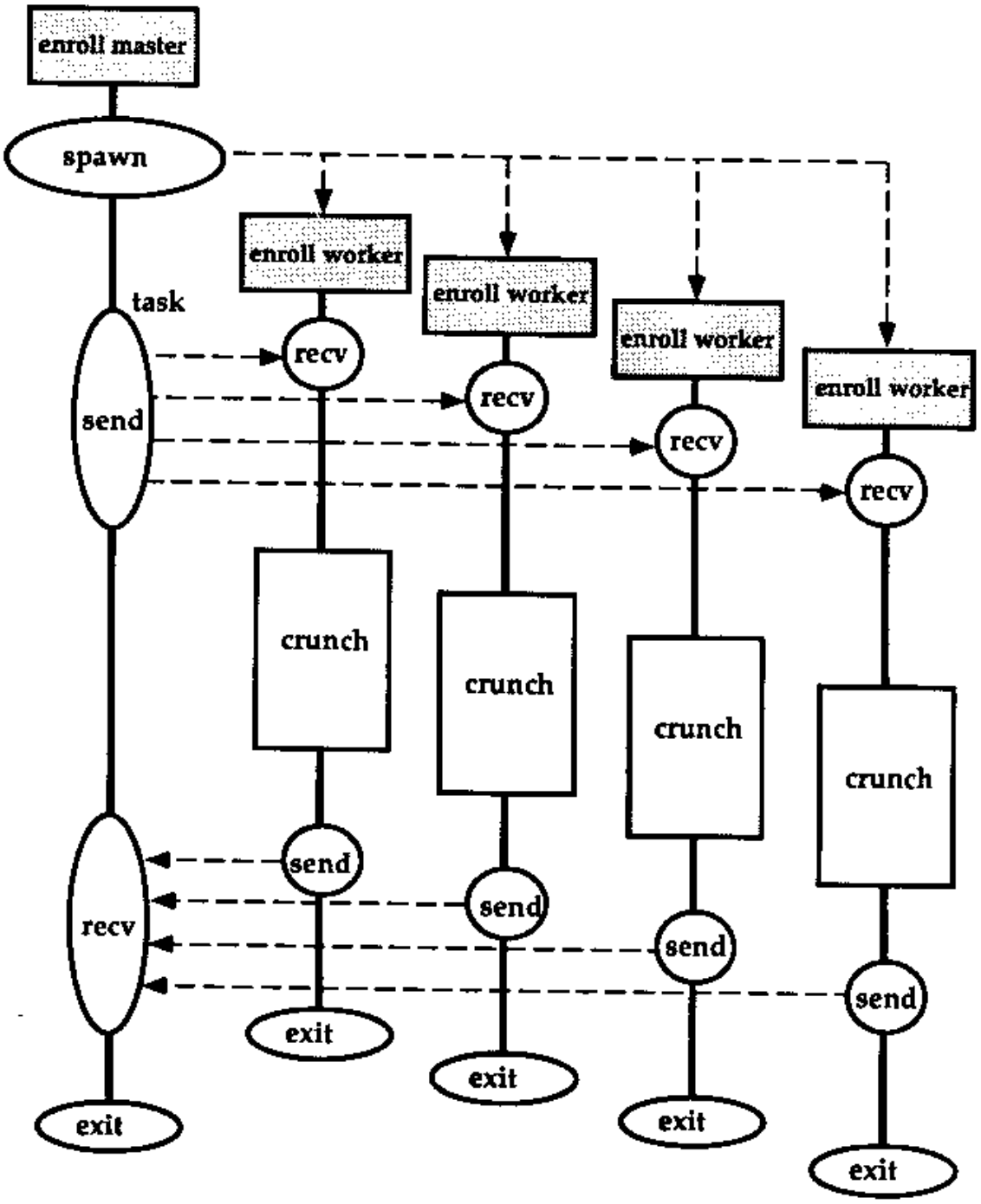
```
C**************************************************************
C ARQUIVO: hello.worker.f
C DESCRIÇÃO: Exemplo simples de programa escravo
C AUTOR: Blaise Barney
C**************************************************************
      program  hello_worker
      include  'fpvm3.h'

      parameter ( HELLO_MSGTYPE=1 )
      integer  mytid, msgtype, info
      character*12  helloworld

      call pvmfmytid (mytid)
      msgtype=HELLO_MSGTYPE
      call pvmfrecv ( -1, msgtype, info )
      call pvmfunpack ( STRING, helloworld, 12, 1, info )
      print *, '*** Reply to: ', helloworld, ' : Hello back! '
      call pvmfexit ( info )
      end
```

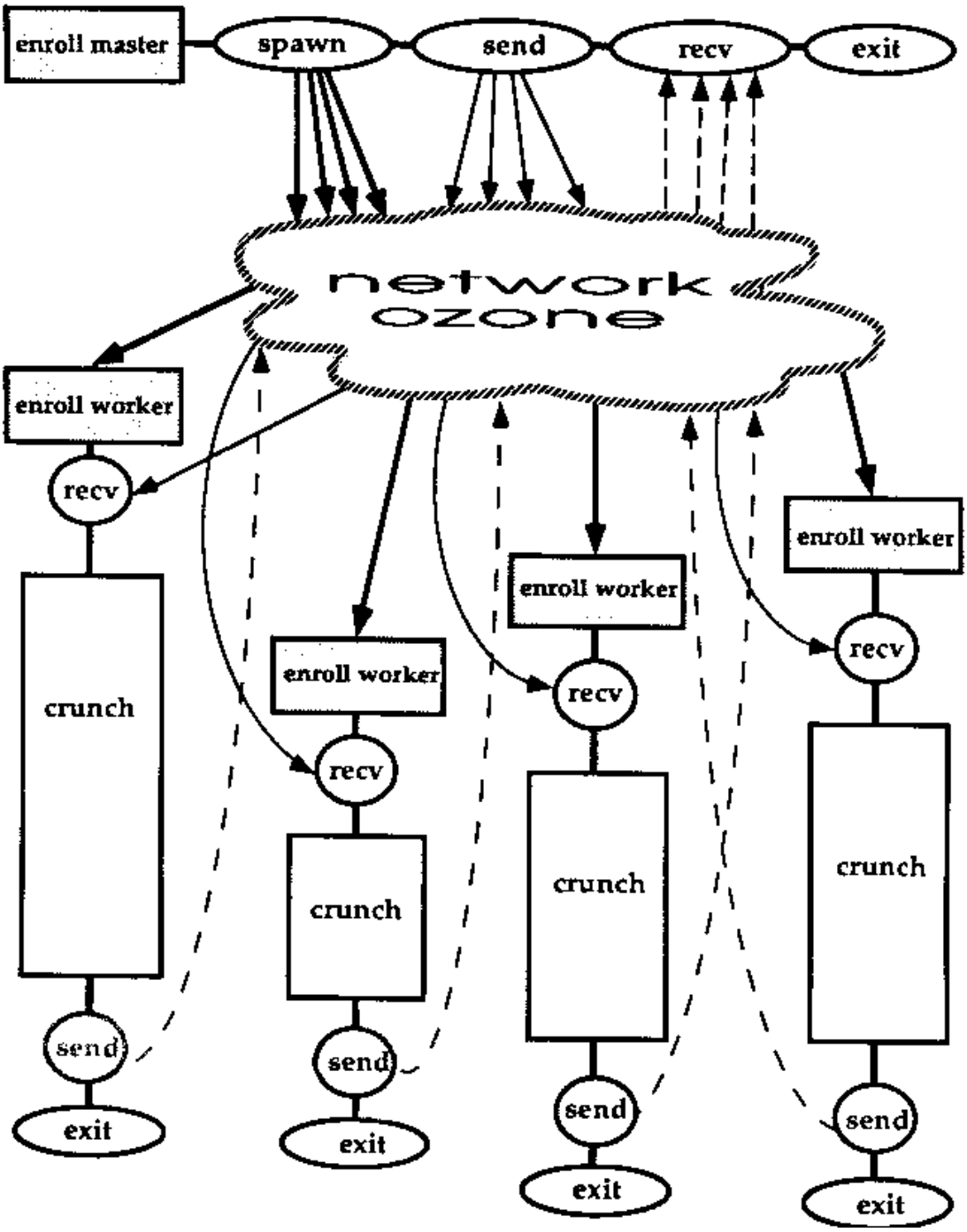
PROGRAMA MESTRE (C)

```
/**************************************************************************
*                    PVM TEMPLATE CODES
* FILE: hello.master.c
* OTHER FILES: hello.worker.c  make.hello.c
* DESCRIPTION: Trivial PVM example - master program.  C version
**************************************************************************/
#include <stdio.h>
#include "pvm3.h"              /* include file needed by PVM version 3 */
#define NTASKS       6
#define HELLO_MSGTYPE   1

main() {
int     mytid,
          tids[NTASKS],
          i,
       msgtype,
          rc;
char    helloworld[13] = "HELLO WORLD!";
printf("Enrolling master task in PVM...\n");
mytid = pvm_mytid();
printf("Spawning worker tasks ...\n");
for (i=0; i<NTASKS; i++) {
  rc = pvm_spawn("hello.worker", NULL, PvmTaskDefault, "", 1, &tids[i]);
  printf("   spawned worker task id = %d\n", tids[i]);
  }
 printf("Sending message to all worker tasks...\n");
msgtype = HELLO_MSGTYPE;
rc = pvm_initsend(PvmDataDefault);
rc = pvm_pkstr(helloworld);
for (i=0; i<NTASKS; i++)
  rc = pvm_send(tids[i], msgtype);
printf("All done. Leaving hello.master.\n");
rc = pvm_exit();
}
```

PROGRAMA ESCRAVO (C)

```
/*****************************************************************************
*                    PVM TEMPLATE CODES
* FILE: hello.worker.c
* OTHER FILES: hello.master.c  make.hello.c
* DESCRIPTION: Trivial PVM example - worker program.  C version
*****************************************************************************/
#include       <stdio.h>
#include "pvm3.h"              /* include file needed by PVM version 3.0 */
#define HELLO_MSGTYPE   1

main() {
int     mytid,
          msgtype,
          rc;
char    helloworld[13];
mytid = pvm_mytid();
msgtype = HELLO_MSGTYPE;
rc = pvm_recv(-1, msgtype);
rc = pvm_upkstr(helloworld);
printf(" ***Reply to: %s : Hello back!\n",helloworld);
rc = pvm_exit();
}
```


enroll master
spawn
enroll worker
enroll worker
enroll worker
enroll worker
task
recv
recv
recv
recv
send
crunch
crunch
crunch
crunch
send
send
send
send
recv
exit
exit
exit
exit
exit


enroll master
spawn
send
recv
exit
network
ozone
enroll worker
recv
crunch
send
exit
enroll worker
recv
crunch
send
exit
enroll worker
recv
crunch
send
exit
enroll worker
recv
crunch
send
exit

2.10 - ROTINAS BÁSICAS DO PVM

2.10.1 - IDENTIFICAR PROCESSOS

C **int tid=pvm_mytid (void)**

FORTRAN **call pvmfmytid (tid)**

tid Variável inteira de retorno que identifica o processo**.**

Esta rotina registra o processo para o pvm, gerando um número inteiro de identificação de 32 bits, criado pelo pvmd local. Está dividido em tres campos básicos: **local pvmd address, cpu number, process id.**

OBS: É, normalmente, a primeira rotina pvm utilizada dentro de um programa.

Erro tid < 0 processo não inicializado.

2.10.2 - DISTRIBUIR PROCESSOS

C **int numt=pvm_spawn (char *task, char **argv, int flag, char *where, int ntask, int *tids)**

FORTRAN **call pvmfspawn (task, flag, where, ntask, tids, numt)**

task Variável caracter contendo o nome do arquivo executável que será inicializado no pvm.

argv Apontador para um conjunto de argumentos de entrada do arquivo executável que será inicializado. **NULL**, se não existir argumentos. Somente para programas em **C**.

flag Variável inteira contendo uma identificação da opção de distribuição.

FORTRAN	C	N	SIGNIFICADO
PVMDEFAULT	PvmTaskDefault	0	Iniciar em qualquer máquina
PVMHOST	PvmTaskHost	1	Indica que será escolhida uma máquina
PVMARCH	PvmTaskArch	2	Indica que será escolhida uma arquitetura
PVMDEBUG	PvmTaskDebug	4	Inicia processo com depurador

where Variável caracter que indica aonde ou como deverá ser inicializado o processo. Depende do valor de **flag.**

ntask Variável inteira que especifica o número de cópias do arquivo executável que será inicializado no pvm.

tids Vetor inteiro de retorno com a identificação de cada processo pvm inicializado.

numt Variável inteira de retorno que indica o número de cópias que foram inicializadas.

Esta rotina inicializa um ou mais (**ntask)** processos (**task)** no pvm, para dar início a paralelização. É retornado a identificação de cada processo iniciado no vetor **tid**.

Erro numt < 0

-2 Argumento inválido
-6 Máquina não existe na configuração
-7 Arquivo executável não localizado
-10 Não existe memória disponível
-14 pvmd não responde
-27 Sem recursos

2.10.3 - INICIALIZAR "BUFFER"

C **int bufid=pvm_initsend(int encoding)**

FORTRAN **call pvmfinitsend(encoding, bufid)**

encoding Valor inteiro que especifica a regra de codificação.

FORTRAN	C	N	SIGNIFICADO
PVMDEFAULT	PvmDataDefault	0	Codificação no padrão XDR
PVMRAW	PvmDataRaw	1	Não há codificação
PVMINPLACE	PvmDataInPlace	2	Dados transferidos direto da memória

bufid Variável inteira de retorno contendo a identificação do "buffer". Deve ser maior ou igual a zero.

Esta rotina reserva uma área específica de memória ("buffer"), para empacotar e codificar uma nova mensagem.

Erro bufid < 0

-2 Argumento inválido
-10 Não existe memória disponível

2.10.4 - EMPACOTAR DADOS

C **int info=pvm_packf(const char *fmt, . . .)**
int info=pvm_pkbyte(char *xp, int nitem, int stride)
int info=pvm_pkdouble(double *dp,int nitem, int stride)
int info=pvm_pkfloat(float *fp, int nitem, int stride)
int info=pvm_pkint(int *ip, int nitem, int stride)
int info=pvm_pklong(long *ip, int nitem, int stride)
int info=pvm_pkshort(short *jp, int nitem, int stride)
int info=pvm_pkstr(char *sp, int nitem, int stride)

FORTRAN call pvmfpack(what, xp, nitem, stride, info)

fmt Expressão que especifica o tipo de dado que será empacotado. Somente para programas em **C**.

what Variável inteira que especifica o tipo de dado que será empacotado. Somente para programas em **FORTRAN**.

STRING	**0**	**REAL4**	**4**
BYTE1	**1**	**COMPLEX8**	**5**
INTEGER2	**2**	**REAL8**	**6**
INTEGER4	**3**	**COMPLEX16**	**7**

xp Variável, de acordo com o tipo definido, que identifica a posição inicial para empacotamento de um conjunto de dados.

nitem Variável inteira que especifica o número total de elementos que serão empacotados (Não é o total de bytes).

stride Parâmetro que determina como deve ser a leitura dos dados para serem empacotados (**Vetor contínuo de dados = 1, Números Complexos = 2).**

info Variável inteira de retorno com o status da execução da rotina.

Esta rotina irá preparar ("empacotar") um conjunto de dados relativos a um determinado tipo de variável. Para cada variável ou variáveis de tipos diferente, se usa uma chamada a rotina "pack".

Erro info < 0

-10 Não existe memória disponível
-15 Não existe "buffer" ativo

2.10.5 - ENVIAR DADOS

C **int info=pvm_send(int tid, int msgtag)**

FORTRAN **call pvmfsend(tid, msgtag, info)**

tid Variável inteira que identifica o processo ao qual se destina a mensagem.

msgtag Rótulo que identifica a mensagem que será transmitida.

info Variável de retorno com o status da execução da rotina.

Esta rotina pega o conteúdo do "buffer" ativo, seu tamanho e o tipo de dado, e envia para outro processo pvm.

Erro info < 0

-2 Argumento inválido
-14 pvmd não responde
-15 Não existe "buffer" ativo

2.10.6 - IDENTIFICAR O PROCESSO PRINCIPAL

C **int tid=pvm_parent(void)**

FORTRAN **call pvmfparent(tid)**

tid Variável inteira de retorno com a identificação do processo pai.

Esta rotina retorna o número de identificação do processo pai, ou seja, o número do processo que inicializou o processo que está executando esta rotina.

Erro tid < 0

-23 Não existe processo pai

2.10.7 - RECEBER DADOS

C **int bufid=pvm_recv(int tid, int msgtag)**

FORTRAN **call pvmfrecv(tid, msgtag, bufid)**

tid Variável inteira que identifica o processo que enviou a mensagem. (**-1**, argumento significando **qualquer** processo).

msgtag Rótulo que identifica a mensagem que foi transferida. (**-1**, argumento significando **qualquer** rótulo).

bufid Variável inteira de retorno com a identificação do "buffer" de recebimento.

Esta rotina bloqueia o processo, **para o processo**, até receber uma mensagem com um rótulo específico do processo que enviou a mensagem. Coloca a mensagem num "buffer" de recebimento, apagando a mensagem anterior.

Erro bufid < 0

-2 Argumento inválido
-14 pvmd não responde

2.10.8 - DESEMPACOTAR DADOS

C **int info=pvm_unpackf(const char *fmt, . . .)**
int info=pvm_upkbyte(char *xp, int nitem, int stride)
int info=pvm_upkdouble(double *dp,int nitem, int stride)
int info=pvm_upkfloat(float *fp, int nitem, int stride)
int info=pvm_upkint(int *ip, int nitem, int stride)
int info=pvm_upklong(long *ip, int nitem, int stride)
int info=pvm_upkshort(short *jp, int nitem, int stride)
int info=pvm_upkstr(char *sp, int nitem, int stride)

FORTRAN call pvmfunpack(what, xp, nitem, stride, info)

fmt Expressão que especifica o tipo de dado que será desempacotado. Somente para programas em **C**.

what Variável inteira que especifica o tipo de dado que será desempacotado. Somente para programas em **FORTRAN.**

STRING	**0**	**REAL4**	**4**
BYTE1	**1**	**COMPLEX8**	**5**
INTEGER2	**2**	**REAL8**	**6**
INTEGER4	**3**	**COMPLEX16**	**7**

xp Variável, de acordo com o tipo definido, que identifica a posição inicial, para o desempacotar um conjunto de dados.

nitem Variável inteira que identifica o número total de elementos que serão desempacotados (Não é o total de bytes).

stride Determina como devem ser posicionados os dados para serem desempacotados (**Vetor contínuo de dados = 1, Números Complexos = 2).**

info Variável inteira de retorno com o status da execução da rotina.

Esta rotina irá desempacotar um conjunto de dados relativos a um determinado tipo de variável. Para cada variável, ou variáveis de tipos diferente, se usa uma chamada a rotina "unpack", que deve coincidir, exatamente, com os "packs" efetuados pelo processo que enviou a mensagem.

Erro info < 0

-10 Não existe memória disponível
-15 Não existe "buffer" ativo

2.10.9 - FINALIZAR PROCESSOS

C **int info=pvm_exit (void)**

FORTRAN **call pvmfexit (info)**

info Variável inteira de retorno com "status" da execução da rotina.

Esta rotina avisa ao pvmd que um processo está finalizando a sua participação na execução paralela do programa. Ela deve ser posicionada sempre ao final de um programa.

Erro info < 0 pvmd não responde.

PROGRAMA MESTRE (FORTRAN)

```
C**************************************************************************
CARQUIVO: pvm.ex1.master.f
C**************************************************************************
      program example1_master
      include  'fpvm3.h'
      integer   NTASKS, ARRAYSIZE
      parameter   (NTASKS=6)
      parameter   (ARRAYSIZE=60000)
      parameter   (FROMMASTER_MSG=1)
      parameter   (FROMWORKER_MSG=2)
      integer   tids(NTASKS), rc, i, index, tid, bufid, bytes, msgtype, chunksize
      real*4   data(ARRAYSIZE), result(ARRAYSIZE)
C*****************Identificação do processo para o PVM ******************
      print  *,  '***************** Starting PVM Example 1 ************'
      call pvmfmytid (rc)
      if (rc .lt. 0) then
            print *, 'MASTER: Unable to enroll this task.'
            print *, 'Return code=', rc,'.  Quitting.'
            stop
      else
            print *, 'MASTER:  Enrolled as task id = ', rc
      endif
C****************** Inicialização dos processos escravos ****************
      print *, 'MASTER: Spawning worker tasks . . .'
      call pvmfspawn ("pvm.ex1.worker", PVMDEFAULT, " ", NTASKS,
     &                          tids, rc)
      if (rc  .eq.  NTASKS)  then
           print *, 'MASTER: Successfully spawned ', rc,' worker tasks.'
      else
           print *, 'MASTER: Not able to spawn requested number of tasks!'
           print *, MASTER: Tasks actually spawned: ', rc, '.  Quitting.'
           stop
      endif
```

```
C********************* Inicializações *********************************
      chunksize=(ARRAYSIZE / NTASKS)
      do 20 i=1, ARRAYSIZE
            data(i) = 0.0
20    continue
C********************** Distribuição de tarefa ************************
      index=1
      msgtype=FROMMASTER_MSG
      do 30 i=1, NTASKS
            call pvmfinitsend (PVMDEFAULT, rc)
            call pvmfpack (INTEGER4, index, 1, 1, rc)
            call pvmfpack (INTEGER4, chunksize, 1, 1, rc)
            call pvmfpack (REAL4, data(index), chunksize, 1, rc)
            call pvmfsend (tids(i), msgtype, rc)
            index=index + chunksize
30    continue
C***************** Espera pelos processos escravos **********************
      print *, 'MASTER: Waiting for results from worker tasks . . .'
      msgtype=FROMWORKER_MSG
      do 40 i=1,  NTASKS
            call pvmfrecv (-1, msgtype, bufid)
            call pvmfbufinfo (bufid, bytes, msgtype, tid, rc)
            call pvmfunpack (INTEGER4, index, 1, 1, rc)
            call pvmfunpack (REAL4, result(index), chunksize, 1, rc)
            print *, '-------------------------------------------------------'
            print *, 'MASTER: Sample results from worker task id = ', tid
            print *, '        result [', index, ']=', result(index)
            print *, '        result [', index+100, ']=', result(index+100)
            print *, '        result [', index+1000, ']=', result(index+1000)
            print *, ' '
40    continue

C************************ Sair do PVM *********************************
      print *, 'MASTER: All Done!
      call pvmfexit (rc)
      end
```

PROGRAMA ESCRAVO (FORTRAN)

```
C**************FILE: pvm.ex1.worker.f*******************************
      program example1_worker
      include 'fpvm3.h'
      integer ARRAYSIZE
      parameter (ARRAYSIZE=60000)
      parameter (FROMMASTER_MSG=1)
      parameter (FROMWORKER_MSG=2)
      integer  i, materid, rc, index, msgtype, chunksize
      real*4  result(ARRAYSIZE)
C********************** Inicia o processo para o PVM ******************
      call pvmfmytid ( rc )
      if ( rc .lt. 0 ) then
        print *, 'WORKER: Unable to enroll this task.'
        print *, '       Return code=', rc, '. Quitting.'
        stop
      else
        print *, 'WORKER: Enrolled as task id=', rc
      endif
C********************** Recebe os dados do programa mestre ************
      msgtype=FROMMASTER_MSG
      call pvmfparent (masterid)
      call pvmfrecv (masterid, msgtype, rc)
      call pvmfunpack (INTEGER4, index, 1, 1, rc)
      call pvmfunpack (INTEGER4, chunksize, 1, 1, rc)
      call pvmfunpack (REAL*4, result(index), chunksize, 1, rc)
      do 10 i=index, index + chunksize
        result ( i ) = i + 1
10    continue
C************** Envio dos resultados para o programa mestre *************
      msgtype=FROMWORKER_MSG
      call pvmfinitsend (PVMDEFAULT, rc)
      call pvmfpack (INTEGER4, index, 1, 1, rc)
      call pvmfpack (REAL4, result(index), chunksize, 1, rc)
      call pvmfsend (masterid, msgtype, rc)
C********************** Sai do PVM ************************************
      call pvmfexit ( rc )
      end
```

PROGRAMA MESTRE (C)

```
/**********************FILE: pvm.ex1.master.c**********************************/
#include <stdio.h>
#include "pvm3.h"                    /* PVM version 3.0 include file */
#define NTASKS              6
#define ARRAYSIZE   60000
#define FROMMASTER_MSG      1
#define FROMWORKER_MSG      2
#define WORKERTASK      "pvm.ex1.worker"
main() {
int    tids[NTASKS],
       rc,                          /* for catching PVM return codes */
       i,                   /* loop variable */
       index,                       /* index into the array */
       tid,                         /* PVM task id */
       bufid,                       /* PVM message buffer id */
       bytes,                       /* number bytes recv'd in PVM message buffer */
       msgtype,                     /* PVM message type */
       chunksize;                   /* for partitioning the array */
float  data[ARRAYSIZE],     /* the intial array */
       result[ARRAYSIZE];   /* for holding results of array operations */
/************************** enroll this task in PVM ****************************/
printf("\n************ Starting PVM Example 1 ************\n");
rc = pvm_mytid();
if (rc < 0) {
  printf("MASTER: Unable to enroll this task.\n");
  printf("   Enroll return code= %d.  Quitting.\n", rc);
  exit(0);
  }
else
  printf("MASTER: Enrolled as task id = %d\n", rc);
/************************** spawn worker tasks ***********************************/
printf("MASTER: Spawning worker tasks...\n");
rc = pvm_spawn(WORKERTASK, NULL, PvmTaskDefault, "", NTASKS, tids);
if (rc == NTASKS)
  printf("MASTER: Successfully spawned %d worker tasks.\n", rc);
else {
```

```
  printf("MASTER: Not able to spawn requested number of tasks!\n");
  printf("MASTER: Tasks actually spawned: %d.  Quitting.\n",rc);
  exit(0);
  }
/************************ initializations ***********************************/
chunksize = (ARRAYSIZE / NTASKS);
for(i=0; i<ARRAYSIZE; i++)
  data[i] =  0.0;
/******************** send array chunks to each worker task **********************/
printf("MASTER: Sending data to worker tasks...\n");
index = 0;
msgtype = FROMMASTER_MSG;
for (i=0; i<NTASKS; i++) {
  rc = pvm_initsend(PvmDataDefault);
  rc = pvm_pkint(&index, 1, 1);
  rc = pvm_pkint(&chunksize, 1, 1);
  rc = pvm_pkfloat(&data[index], chunksize, 1);
  rc = pvm_send(tids[i], msgtype);
  index = index + chunksize;
  }
/******************* wait for results from all worker tasks **********************
printf("MASTER: Waiting for results from worker tasks...\n");
msgtype = FROMWORKER_MSG;
for(i=0; i<NTASKS; i++){
  bufid = pvm_recv(-1, msgtype);
  rc = pvm_bufinfo(bufid, &bytes, &msgtype, &tid);
  rc = pvm_upkint(&index, 1, 1);
  rc = pvm_upkfloat(&result[index], chunksize, 1);
  printf("-------------------------------------------------\n");
  printf("MASTER: Sample results from worker task = %d\n",tid);
  printf("   result[%d]=%f\n", index, result[index]);
  printf("   result[%d]=%f\n", index+100, result[index+100]);
  printf("   result[%d]=%f\n\n", index+1000, result[index+1000]);
  }
/************************** exit from PVM ***********************************/
printf("MASTER: All Done! \n");
rc = pvm_exit();
}
```

PROGRAMA ESCRAVO (C)

```
#include       <stdio.h>
#include     "pvm3.h"       /* PVM version 3.0 include file */
#define ARRAYSIZE       60000
#define FROMMASTER_MSG   1
#define FROMWORKER_MSG   2
main() {
int     masterid,             /* PVM task id for master process */
        rc,                  /* for catching PVM return codes */
        i,                  /* loop variable */
        index,               /* index into the array */
        msgtype,             /* PVM message type */
        chunksize;           /* for partitioning the array */
float   result[ARRAYSIZE];      /* for holding results of array operations */
rc = pvm_mytid();
if (rc < 0) {
  printf("WORKER: Unable to enroll this task.\n");
  printf("   Enroll return code= %d.  Quitting.\n", rc);
  exit(0);
  }
else {
  printf("WORKER: Enrolled as task id = %d\n", rc);
  }
msgtype = FROMMASTER_MSG;
masterid = pvm_parent();
rc = pvm_recv(masterid, msgtype);
rc = pvm_upkint(&index, 1, 1);
rc = pvm_upkint(&chunksize, 1, 1);
rc = pvm_upkfloat(&result[index], chunksize, 1);
for(i=index; i < index + chunksize; i++)
  result[i] = i + 1;
msgtype = FROMWORKER_MSG;
rc = pvm_initsend(PvmDataDefault);
rc = pvm_pkint(&index, 1, 1);
rc = pvm_pkfloat(&result[index], chunksize, 1);
rc = pvm_send(masterid, msgtype);
rc = pvm_exit();
}
```

2.11 - XPVM

O **xpvm** é uma interface gráfica, que funciona como um auxílio na depuração e na visualização da execução de um programa que utilize o pvm. É possível observar a comunicação entre os processos e depurar o acesso e o funcionamento das rotinas pvm.

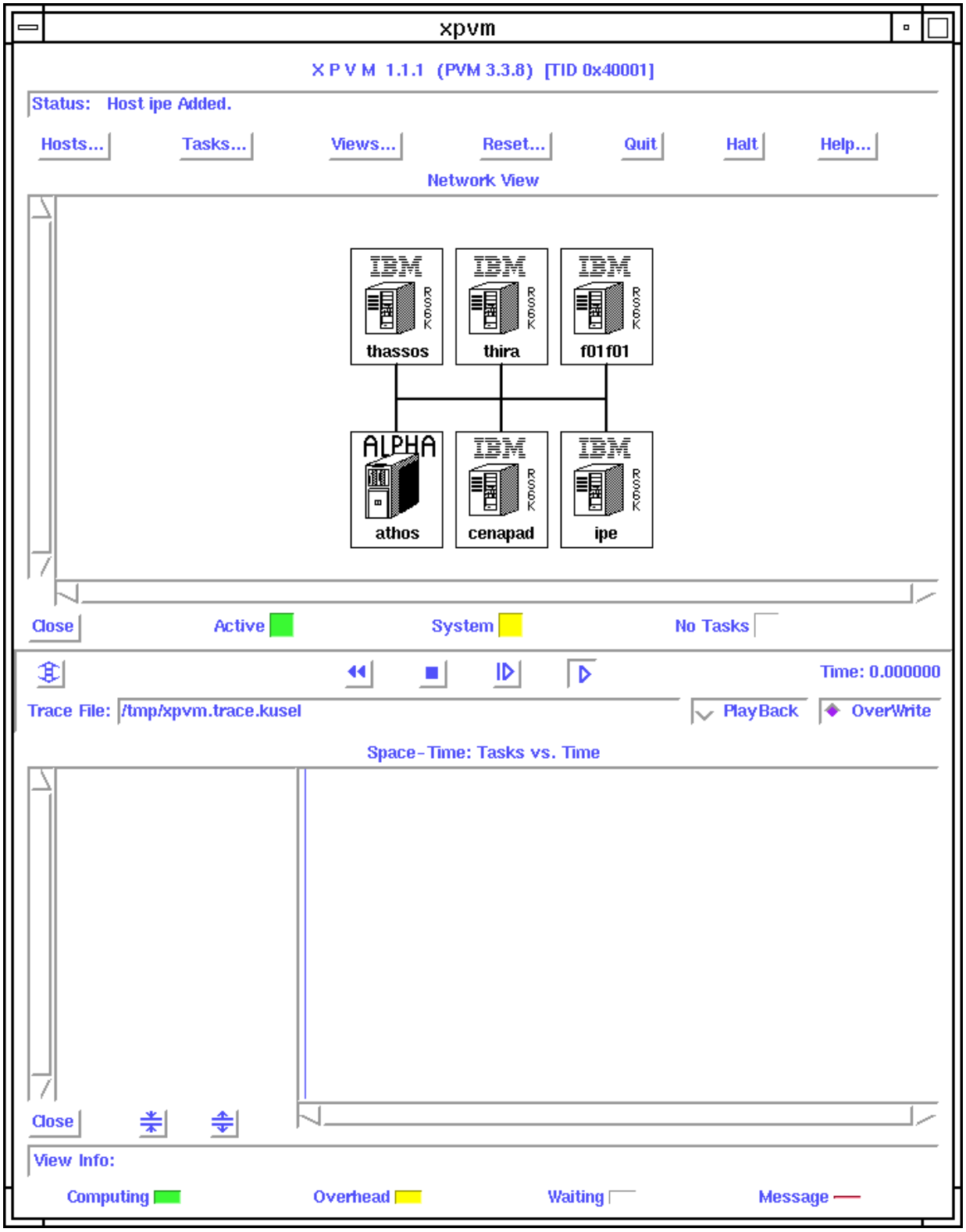
O **xpvm** possui várias janelas de visualização:

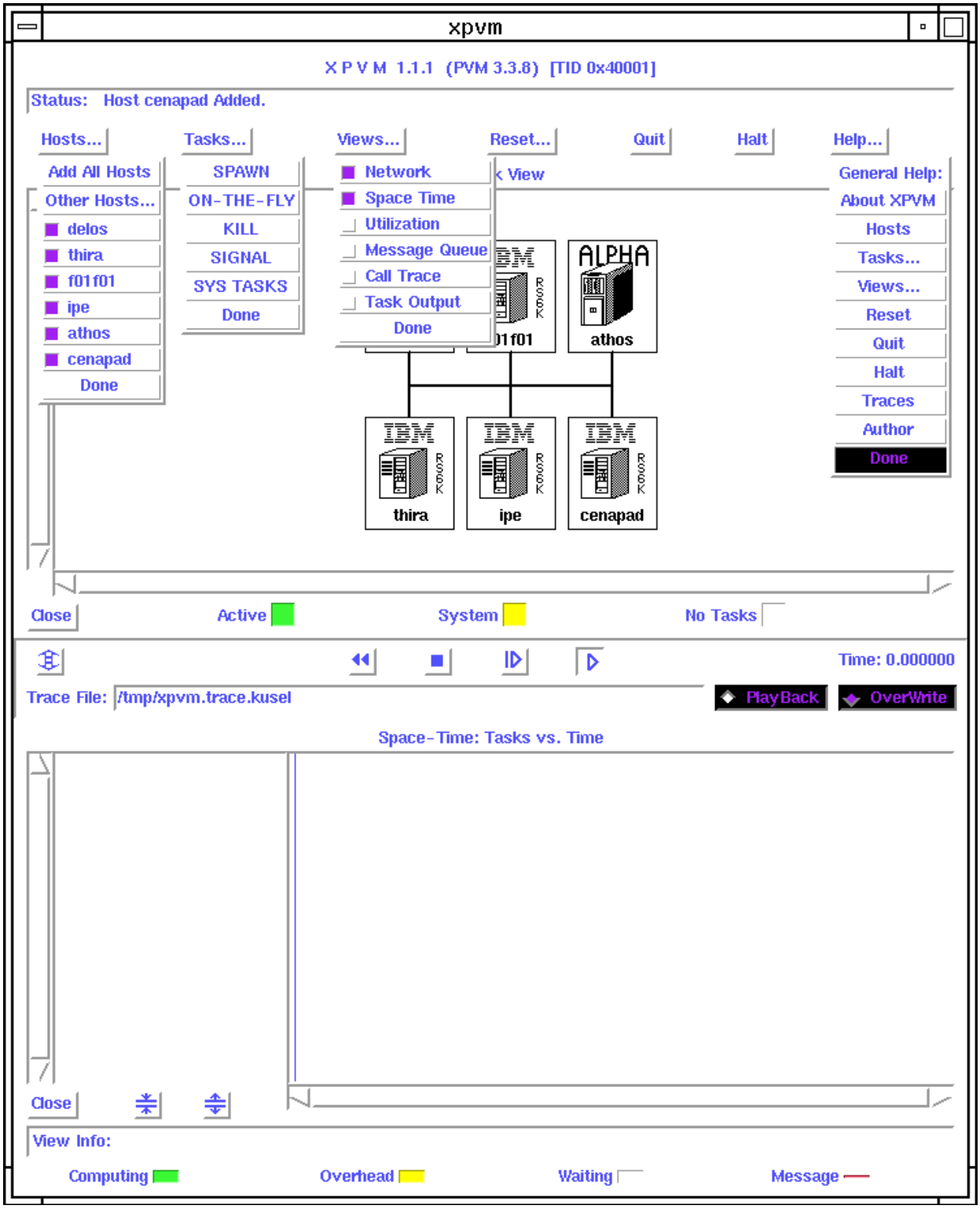
- Configuração de máquinas;
- Relação tempo x execução x máquina;
- Gráfico de utilização de cpu;
- Gráfico com o número de mensagens em fila;
- "Debugg" das rotinas pvm ("trace file");
- Saída de resultados.

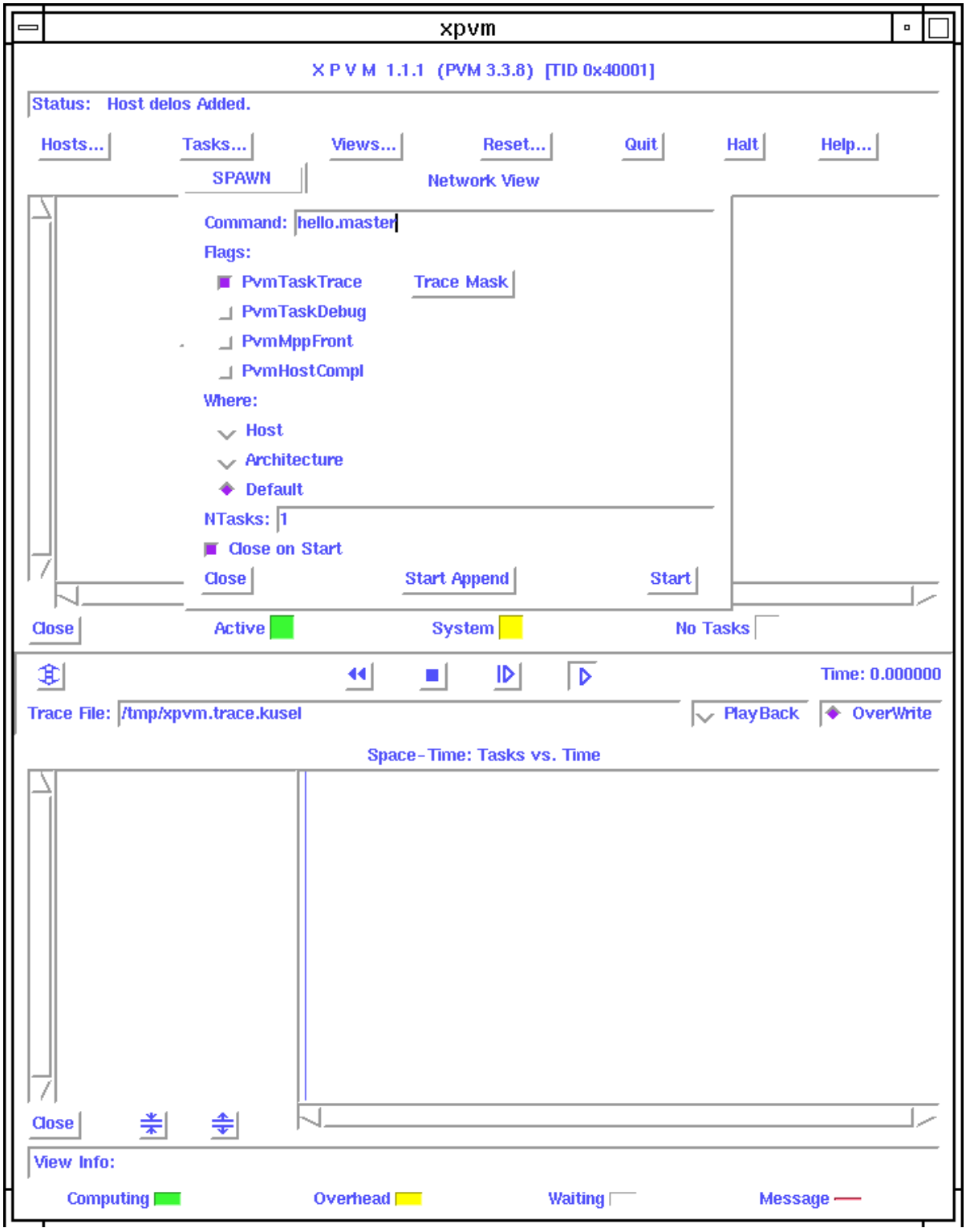
O **xpvm** procura, por "default", um arquivo especial de configuração de máquinas, **.xpvm_hosts**, que fica localizado no seu diretório principal. Esse arquivo é opcional, não sendo necessário para a execução do xpvm.

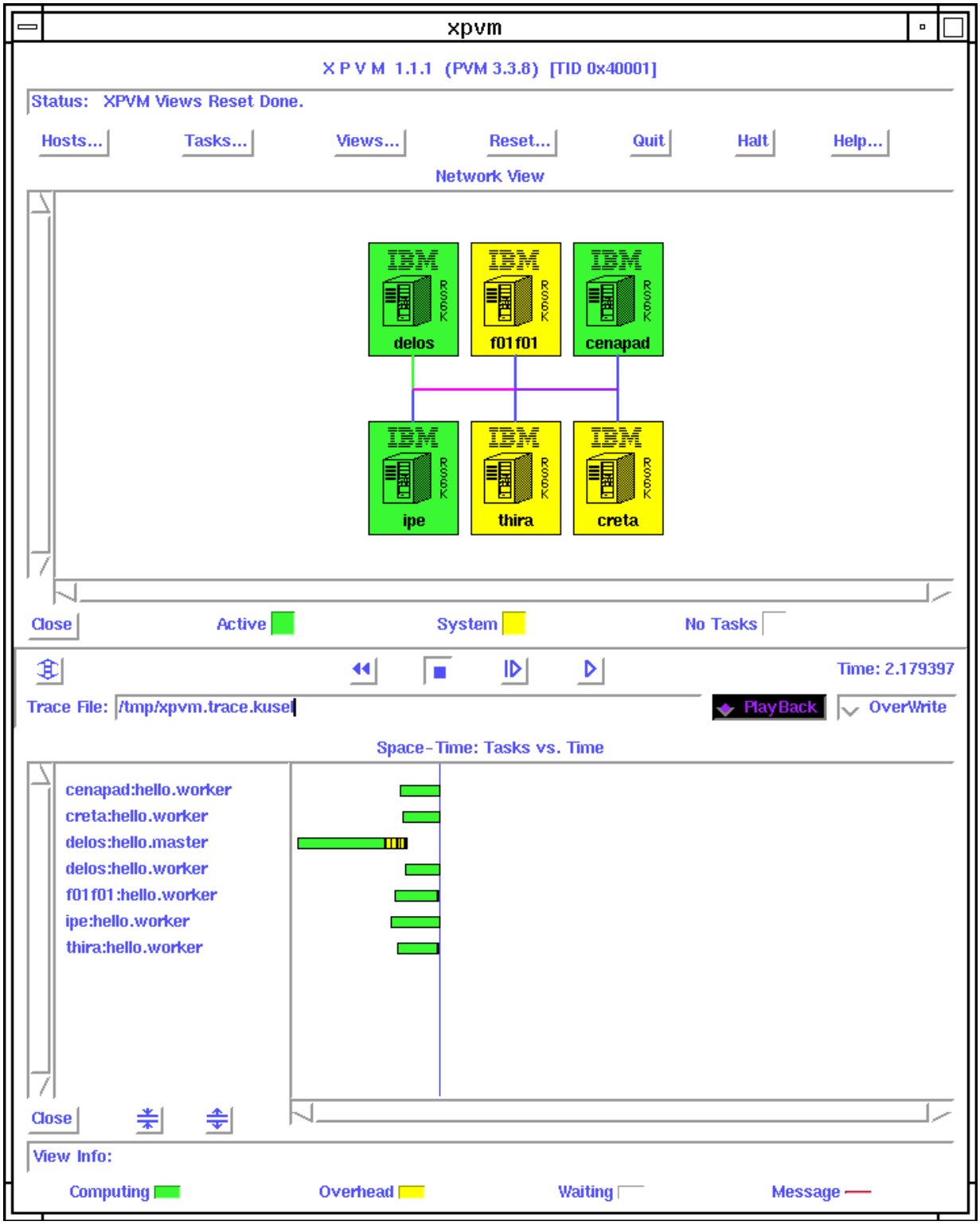
% **xpvm**

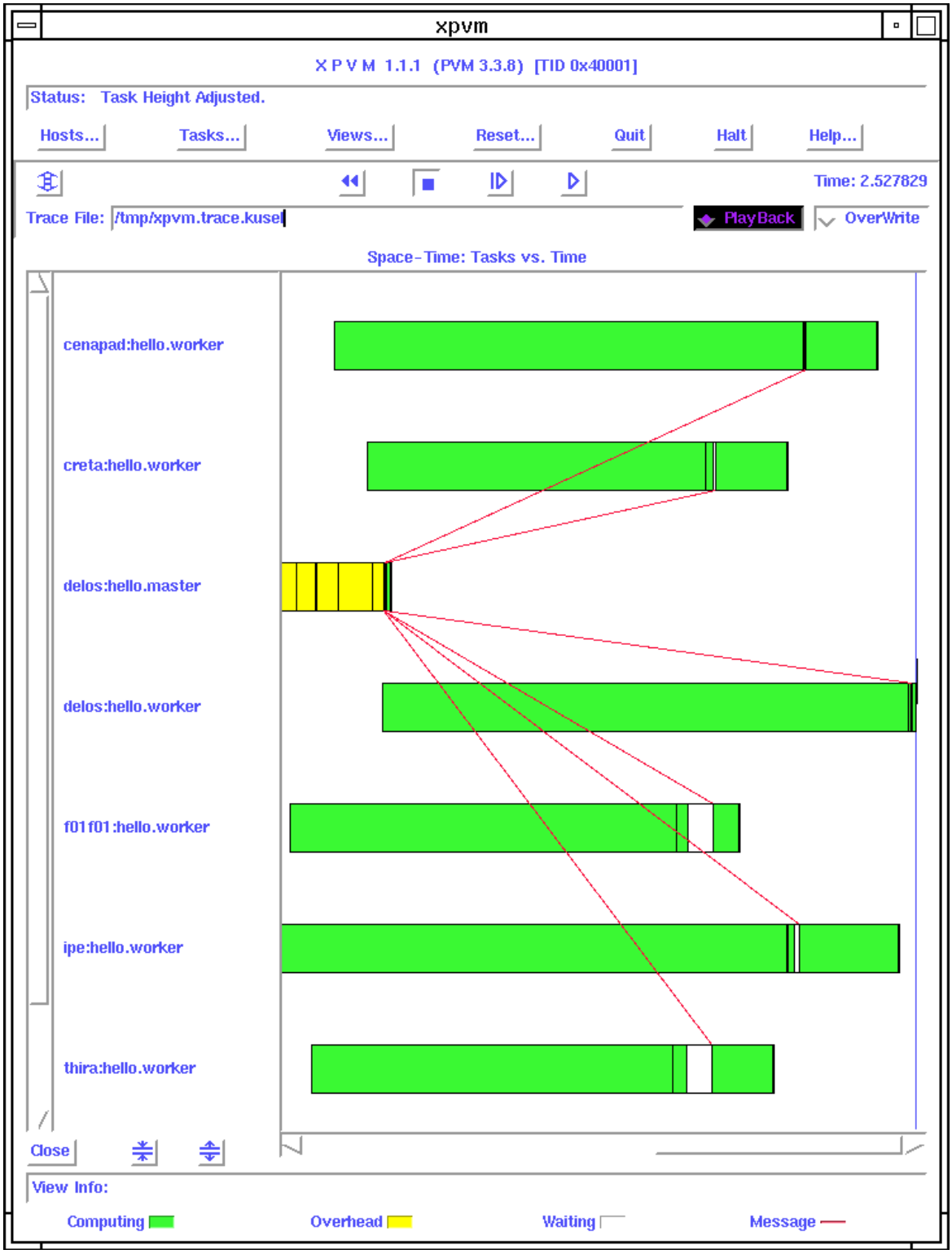
OBS: Lembre-se só adicione as máquinas que possibilitam o processamento interativo.

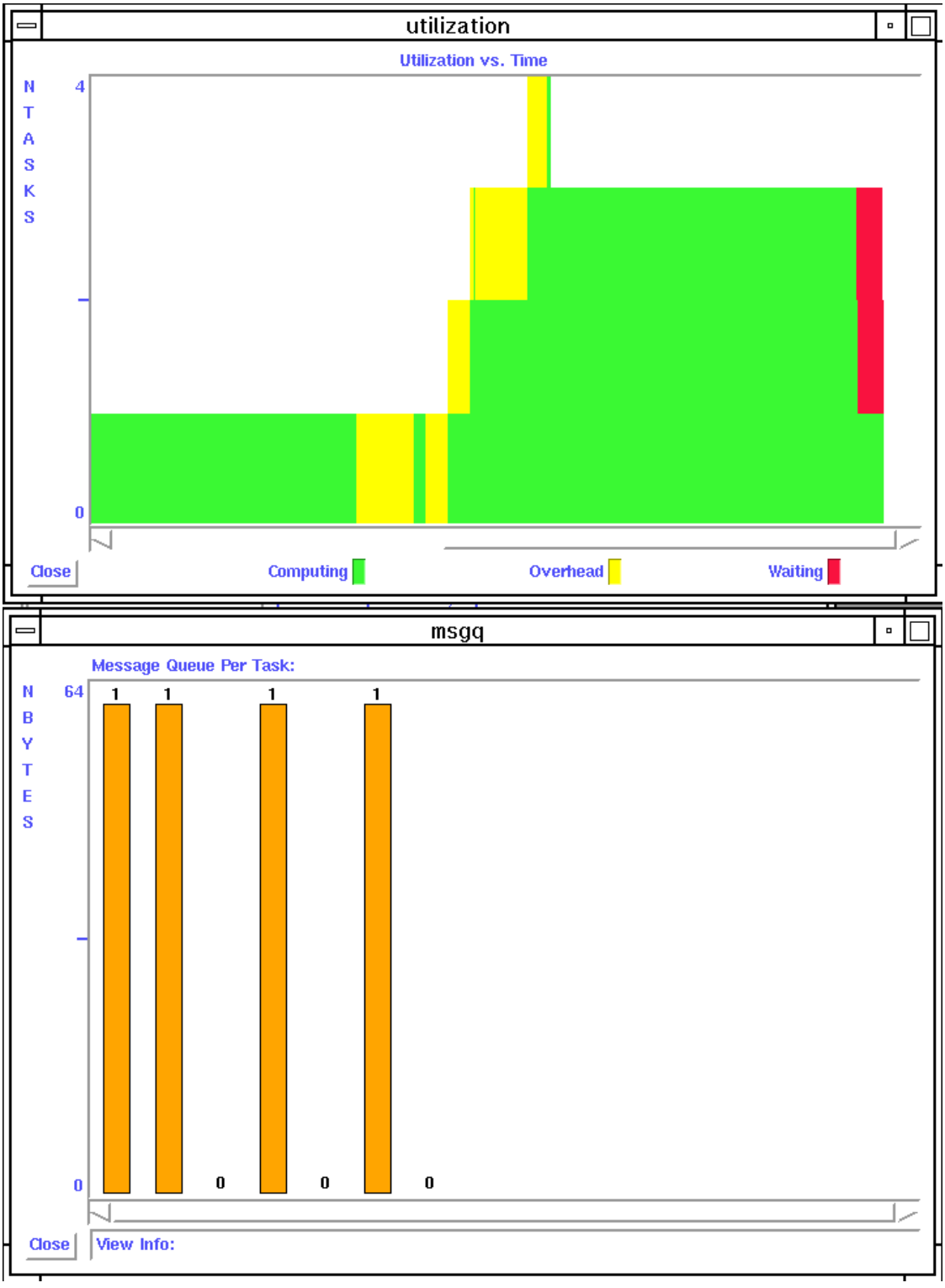

xpvm
X P V M 1.1.1 (PVM 3.3.8) [TID 0x40001]
Status: Host ipe Added.
Hosts...
Tasks...
Views...
Reset...
Quit
Halt
Help...
Network View
IBM
RS6K
thassos
thira
f01f01
ALPHA
athos
cenapad
ipe
Close
Active
System
No Tasks
Time: 0.000000
Trace File: /tmp/xpvm.trace.kusel
PlayBack
OverWrite
Space-Time: Tasks vs. Time
Close
View Info:
Computing
Overhead
Waiting
Message

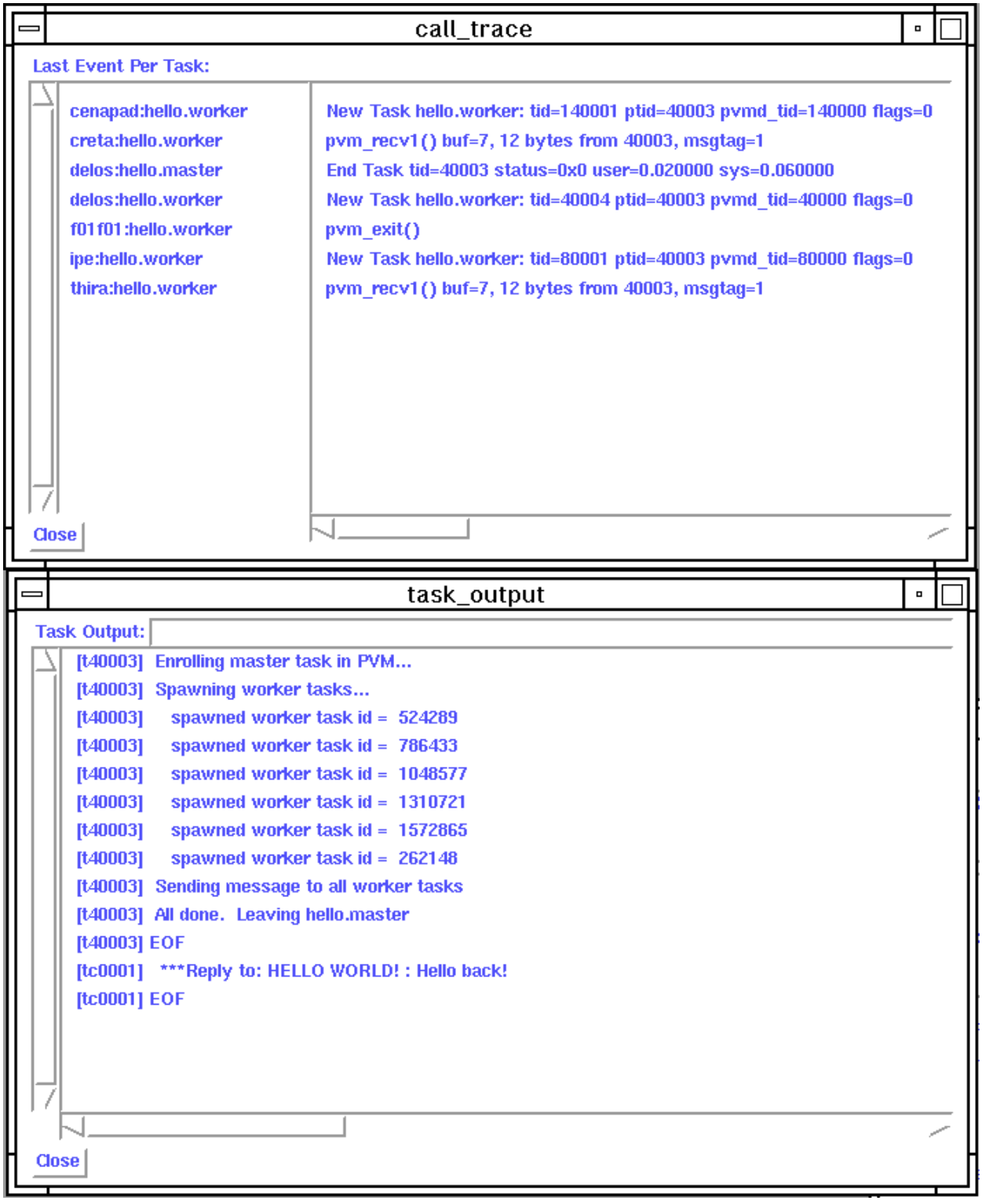

xpvm
XPVM 1.1.1 (PVM 3.3.8) [TID 0x40001]
Status: Host cenapad Added.
Hosts...
Tasks...
Views...
Reset...
Quit
Halt
Help...
Add All Hosts
Other Hosts...
delos
thira
f01f01
ipe
athos
cenapad
Done
SPAWN
ON-THE-FLY
KILL
SIGNAL
SYS TASKS
Done
Network
Space Time
Utilization
Message Queue
Call Trace
Task Output
Done
k View
General Help:
About XPVM
Hosts
Tasks...
Views...
Reset
Quit
Halt
Traces
Author
Done
ALPHA
athos
IBM
RS6K
thira
ipe
cenapad
Close
Active
System
No Tasks
Time: 0.000000
Trace File: /tmp/xpvm.trace.kusel
PlayBack
OverWrite
Space-Time: Tasks vs. Time
Close
View Info:
Computing
Overhead
Waiting
Message


xpvm
X P V M 1.1.1 (PVM 3.3.8) [TID 0x40001]
Status: Host delos Added.
Hosts...
Tasks...
Views...
Reset...
Quit
Halt
Help...
SPAWN
Network View
Command: hello.master
Flags:
PvmTaskTrace
Trace Mask
PvmTaskDebug
PvmMppFront
PvmHostCompl
Where:
Host
Architecture
Default
NTasks: 1
Close on Start
Close
Start Append
Start
Close
Active
System
No Tasks
Time: 0.000000
Trace File: /tmp/xpvm.trace.kusel
PlayBack
OverWrite
Space-Time: Tasks vs. Time
Close
View Info:
Computing
Overhead
Waiting
Message


xpvm
XPVM 1.1.1 (PVM 3.3.8) [TID 0x40001]
Status: XPVM Views Reset Done.
Hosts...
Tasks...
Views...
Reset...
Quit
Halt
Help...
Network View
delos
f01f01
cenapad
ipe
thira
creta
Close
Active
System
No Tasks
Time: 2.179397
Trace File: /tmp/xpvm.trace.kusel
PlayBack
OverWrite
Space-Time: Tasks vs. Time
cenapad:hello.worker
creta:hello.worker
delos:hello.master
delos:hello.worker
f01f01:hello.worker
ipe:hello.worker
thira:hello.worker
Close
View Info:
Computing
Overhead
Waiting
Message


xpvm
X P V M 1.1.1 (PVM 3.3.8) [TID 0x40001]
Status: Task Height Adjusted.
Hosts...
Tasks...
Views...
Reset...
Quit
Halt
Help...
Time: 2.527829
Trace File: /tmp/xpvm.trace.kusel
PlayBack
OverWrite
Space-Time: Tasks vs. Time
cenapad:hello.worker
creta:hello.worker
delos:hello.master
delos:hello.worker
f01f01:hello.worker
ipe:hello.worker
thira:hello.worker
Close
View Info:
Computing
Overhead
Waiting
Message


utilization
Utilization vs. Time
N
T
A
S
K
S
4
0
Close
Computing
Overhead
Waiting
msgq
Message Queue Per Task:
N
B
Y
T
E
S
64
0
1
1
0
1
0
1
0
Close
View Info:


call_trace
Last Event Per Task:
cenapad:hello.worker
creta:hello.worker
delos:hello.master
delos:hello.worker
f01f01:hello.worker
ipe:hello.worker
thira:hello.worker
New Task hello.worker: tid=140001 ptid=40003 pvmd_tid=140000 flags=0
pvm_recv1() buf=7, 12 bytes from 40003, msgtag=1
End Task tid=40003 status=0x0 user=0.020000 sys=0.060000
New Task hello.worker: tid=40004 ptid=40003 pvmd_tid=40000 flags=0
pvm_exit()
New Task hello.worker: tid=80001 ptid=40003 pvmd_tid=80000 flags=0
pvm_recv1() buf=7, 12 bytes from 40003, msgtag=1
Close
task_output
Task Output:
[t40003] Enrolling master task in PVM...
[t40003] Spawning worker tasks...
[t40003] spawned worker task id = 524289
[t40003] spawned worker task id = 786433
[t40003] spawned worker task id = 1048577
[t40003] spawned worker task id = 1310721
[t40003] spawned worker task id = 1572865
[t40003] spawned worker task id = 262148
[t40003] Sending message to all worker tasks
[t40003] All done. Leaving hello.master
[t40003] EOF
[tc0001] ***Reply to: HELLO WORLD! : Hello back!
[tc0001] EOF
Close

2.12 - COMO MELHORAR A PERFORMANCE

2.12.1 - NA DISTRIBUIÇÃO DE PROCESSOS

Toda vez que se faz uma chamada a uma rotina, ocorre um **overhead** devido ao acesso a biblioteca PVM.

Para se inicializar **n** processos escravos, utiliza-se da rotina **spawn, n** vezes, que irá proporcionar um **overhead, n** vezes maior.

Utilize a rotina **spawn** uma única vez. Se for necessário inicializar **n** processos escravos, utilize-se do parâmetro **ntask** da rotina.

```
int numt=pvm_spawn ( char  *task, char  **argv, int flag,
                     char *where, int ntask, int *tids )

call pvmfspawn ( task, flag, where, ntask, tids, numt )
```

ntask Variável inteira que especifica o número de cópias do arquivo executável a ser iniciado no pvm.

tids Vetor inteiro de retorno com a identificação de cada processo pvm inicializado por esta rotina.

2.12.2 - CONFIGURAR OPÇÕES DO PVM

Toda comunicação entre os processos de uma aplicação PVM, é feita entre os **daemons** de cada máquina, que ocasiona uma perda de performance. Essa perda pode ser amenizada reconfigurando a opção de roteamento do PVM.

C **int oldval=pvm_setopt(int what, int val)**

FORTRAN **call pvmfsetopt(what, val, oldval)**

what Variável inteira que determina o parâmetro que será alterado.

FORTRAN	C	N	SIGNIFICADO
PVMROUTE	PvmRoute	1	Rotear mensagens

val Variável inteira que define a nova opção do parâmetro escolhido pela variável **what**.

FORTRAN	C	N	SIGNIFICADO
PVMDONTROUTE	PvmDontRoute	1	pvmd-pvmd
PVMALLOWDIRECT	PvmAllowDirect	2	task-task (TCP, Específico)
PVMROUTEDIRECT	PvmRouteDirect	3	task-task (TCP, Geral)

oldval Variável inteira que retorna a antiga opção do parâmetro.
OBS: Essa rotina deve ser utilizada logo após a rotina **mytid**.

2.12.3 - ENVIAR DADOS PARA TODOS OS PROCESSOS

C **int info=pvm_mcast(int *tids, int ntask, int msgtag)**

FORTRAN **call pvmfmcast (ntask, tids, msgtag, info)**

ntask Variável inteira que especifíca o número de processos que receberão mensagens.

tids Vetor inteiro com a identificação dos processos que receberão as mensagens.

msgtag Rótulo que identifica a mensagem que será transferida.

info Variável inteira com o status da execução da rotina.

Esta rotina distribui, simultaneamente, para vários processos identificados, a mensagem arquivada no "buffer" ativo de envio.

Erro info < 0

-2 Argumento inválido
-14 pvmd não responde
-15 Não existe "buffer" ativo

2.12.4 - EMPACOTAR E ENVIAR DADOS

C **int info=pvm_psend(int tid, int msgtag, char *buf, int len, int datatype)**

FORTRAN **call pvmfpsend(tid, msgtag, buf, len, datatype, info)**

tid Variável inteira que identifica o processo ao qual se destina o dado.

msgtag Rótulo que identifica a mensgem que será transmitida.

buf Variável que identifica um conjunto de dados que serão empacotados e enviados. Pode ser uma variável simples, um vetor ou uma matriz.

len Tamanho do parâmetro **buf**. (Número de elementos X tamanho do tipo de dado).

datatype Tipo do dado que será empacotado e enviado. Em **FORTRAN**, será o mesmo utilizado nas rotinas de "pack". Em **C**, segue a tabela abaixo:

TIPO DE DADO EM C	**datatype**
string	**PVM_STR**
byte	**PVM_BYTE**
short	**PVM_SHORT**
int	**PVM_INT**
real	**PVM_FLOAT**
double	**PVM_DOUBLE**
long integer	**PVM_LONG**
complex	**PVM_CPLX**

info Variavel inteira de retorno com o status da execução da rotina.

Esta rotina, pega o conteúdo do campo **buf**, o **len** e o **datatype**, empacota-os e envia para o processo identificado. Qualquer rotina de recebimento de dados poderá receber os dados enviados.

Erro info < 0
-2 Argumento inválido
-14 pvmd não responde

2.12.5 - REDUZIR "BUFFERING"

No PVM, é necessário criar um "buffer" para colocar os dados, especificar o tamanho, o tipo, empacotar, e envia-los para o processo destino.

O PVM fixa o tamanho do "buffer" em **4K**.

Para conteúdos acima de **4K**, ocorre o procedimento de **"buffering",** várias vezes ocorrerá a cópia dos dados do endereço de memória da variável para o "buffer".

Esse procedimento proporciona uma perda de performance na execução do programa.

Se possível, evite o **"buffering"** utilizando a opção de empacotamento no endereço de memória da variável, **PvmDataInPlace / PVMINPLACE**, da rotina **initsend.**

Ex.:

C int bufid=pvm_initsend(**PvmDataInPlace**)
FORTRAN call pvmfinitsend(**PVMINPLACE**, bufid)

OBS: Não será possível utilizar essa opção, se o ambiente de máquinas, for heterogêneo.

2.12.6 - RECEBER E DESEMPACOTAR DADOS

C **int info=pvm_precv(int tid, int msgtag, char *buf, int len, int datatype, int atid, int atag, int alen)**

FORTRAN **call pvmfprecv(tid, msgtag, buf, len, datatype, atid, atag, alen, info)**

tid Variável inteira que identifica o processo que enviou os dados.

msgtag Rótulo que identifica a mensgem que foi transmitida.

buf Variável que identifica um endereço, aonde serão armazenados e desempacotados os dados. Pode ser uma variável simples um vetor ou uma matriz.

len Tamanho do parâmetro **buf**. (Número de elementos X tamanho do tipo de dado).

datatype Tipo do dado que será recebido e desempacotado. Em **FORTRAN**, será o mesmo utilizado nas rotinas de "pack". Em **C**, segue a tabela abaixo:

TIPO DE DADO EM C	datatype
string	**PVM_STR**
Byte	**PVM_BYTE**
Short	**PVM_SHORT**
Int	**PVM_INT**
Real	**PVM_FLOAT**
double	**PVM_DOUBLE**
long integer	**PVM_LONG**
complex	**PVM_CPLX**

atid Variável inteira de retorno com a identificação do processo que enviou os dados.

atag Variável inteira de retorno com o rótulo da mensagem.

alen Variável inteira de retorno com o tamanho da mensagem.

info Variavel inteira de retorno com o status da execução.

Esta rotina bloqueia o processo, **para o processo**, até que uma determinada mensagem com o rótulo **msgtag**, tenha chegado de **tid**. Desempacota e armazena a mensagem em **buf**. Essa rotina recebe, de qualquer rotina que envia dados.

2.12.7 - RECEBER DADOS SEM BLOQUEAR PROCESSO

C **int bufid=pvm_nrecv(int tid, int msgtag)**

FORTRAN **call pvmfnrecv(tid, msgtag, bufid)**

tid Variável inteira que identifica o processo que está enviando a mensagem. (**-1**, argumento significando **qualquer** processo).

msgtag Rótulo que identifica a mensagem que foi transferida. (**-1**, argumento significando **qualquer** rótulo).

bufid Variável inteira de retorno com a identificação do "buffer" de recebimento.

Esta rotina verifica, sem **bloquear o processo**, se existe alguma mensagem **msgtag** do processo **tid.** Se a mensagem tiver chegado, armazena-a em **bufid**. Ao usuário caberá verificar, em algum momento, se **bufid** >0, para desempacotar os dados e armazena-los em variáveis.

Erro bufid<0

-2 Argumento inválido
-14 pvmd não responde

2.13 - ROTINAS PARA INFORMAÇÃO E CONTROLE

2.13.1 - IDENTIFICAR O AMBIENTE PVM

C **int info=pvm_config(int *nhost, int *narch, struct pvmhostinfo **hostp)**

struct pvmhostinfo {
int hi_tid;
char *hi_name;
char *hi_arch;
int hi_speed;
} hostp;

FORTRAN **call pvmfconfig(nhost, narch, dtid, name, arch, speed, info)**

nhosts Variável inteira de retorno com o número de máquinas do ambiente PVM.

narch Variável inteira de retorno com o número de arquiteturas existentes no ambiente PVM.

hostp Apontador para um vetor de estruturas, contendo as informações que retornam de cada maquina do ambiente PVM. Somente para programas em **C.**

dtid Variável inteira de retorno com a identificação do **daemon**.

name Variável caracter de retorno com o nome da máquina.

arch Variável inteira de retorno com o nome da arquitetura.

speed Variável inteira de retorno com a velocidade relativa da máquina.

info Variável inteira de retorno com o status da execução da rotina.

Esta rotina retorna com informações de cada máquina do atual ambiente PVM. Em C, uma única chamada a rotina, retorna informações de todo o ambiente. Em FORTRAN, será necessário chamar a rotina **n** vezes, relativo a **n** máquinas.

Erro info<0

-14 pvmd não responde

2.13.2 - IDENTIFICAR OS PROCESSOS PVM

C **int info=pvm_tasks (int where, int *ntask, struct taskinfo **taskp)**

```
struct taskinfo {
                    int ti_tid;
                    int ti_ptid;
                    int ti_host;
                    int ti_flag;
                    char *ti_a_out;
                    } taskp;
```

FORTRAN **call pvmftasks(where, ntask, tid, ptid, dtid, flag, aout, info)**

where Variável inteira que especifíca qual, ou quais, os processos tarefas que deverão retornar informações.

0	**Todos os processos**
pvmd tid	**Todos os processos de uma máquina**
tid	**Um processo específico**

ntask Variável inteira que retorna o número de processos.

taskp Apontador para um vetor de estruturas, contendo as informações que retornam de cada processo. Somente para programas em **C**.

tid Variável inteira que retorna com o número de identificação de um processo.

ptid Variável inteira que retorna com o número de identificação do processo mestre do **tid**.

dtid Variável inteira que retorna o número de identificação do **daemon** pvm do **tid**.

flag Variável inteira que retorna com o status da execução do processo **tid**.

aout Variável caracter que retorna com o nome do processo **tid**.

info Variável inteira que retorna o status da execução da rotina.

Esta rotina retorna informações a respeito da execução dos processos que estão sendo executados. Em C, uma única chamada a rotina, retorna informações de todo os processos. Em FORTRAN, será necessário chamar a rotina **n** vezes, relativo a **n** processos inicializados.

Erro info < 0
-2 Argumento inválido
-6 Não existe a máquina na configuração
-14 pvmd não responde

2.13.3 - IDENTIFICAR O "BUFFER"

C **int info=pvm_bufinfo(int bufid, int *bytes, int *msgtag, int *tid)**

FORTRAN **call pvmfbufinfo(bufid, bytes, msgtag, tid, info)**

bufid Variável inteira que identifica o "buffer", no qual se deseja verificar o status. O **bufid**, normalmente, é fornecido pelas rotinas de **recv**.

bytes Variável inteira de retorno com o tamanho, em bytes, da mensagem no "buffer".

msgtag Variável inteira de retorno com o rótulo da mensagem no "buffer".

tid Variável inteira de retorno com a identificação do processo que enviou a mensagem.

info Variável inteira de retorno com o status da execução da rotina.

Essa rotina retorna informações do "buffer" solicitado. Normalmente, é utilizado logo após um **recv**.

Erro info < 0
-2 Argumento inválido
-15 Não existe o "buffer"

2.13.4 - ADICIONAR MÁQUINAS

C **int info=pvm_addhosts(char **hosts, int nhost, int *infos)**

FORTRAN **call pvmfaddhosts(host, info)**

hosts Apontador para um vetor caracter, contendo o nome das máquinas que serão adicionadas. Somente para programas em **C**.

nhost Variável inteira com o número de máquinas que serão adicionadas.

infos Apontador para um vetor de inteiros, contendo o status da adiçao de cada máquina. Somente para programas em **C**.

host Variável caracter contendo o nome da máquina que será adicionada.

info Variável inteira de retorno com o status da execução.

Esta rotina adiciona uma ou mais máquinas à configuração PVM. Em C, uma única chamada a rotina, adiciona várias máquinas. Em FORTRAN, será necessário chamar a rotina **n** vezes, relativo a **n** máquinas que se deseja adicionar.

Erro info<0
- -2 Argumento inválido
- -6 Não existe máquina
- -28 Máquina já existe na configuração

2.13.5 - REMOVER MÁQUINAS

C **int info=pvm_delhosts(char **hosts, int nhost, int *infos)**

FORTRAN **call pvmfdelhosts(host, info)**

hosts Apontador para um vetor caracter, contendo o nome das máquinas que serão removidas. Somente para programas em **C**.

nhost Variável inteira com o número de máquinas que serão removidas.

infos Apontador para um vetor de inteiros, contendo o status da remoção de cada máquina. Somente para programas em **C**.

host Variável caracter, contendo o nome da máquina que será removida.

info Variável inteira de retorno com o status da execução.

Essa rotina remove uma ou mais máquinas da configuração PVM atual. Em C, uma única chamada a rotina, remove várias máquinas. Em FORTRAN, será necessário chamar a rotina **n** vezes, relativo a **n** máquinas que se deseja remover.

Erro info<0
-2 Argumento inválido
-14 pvmd não responde

2.13.6 - VERIFICAR O STATUS DE UMA MÁQUINA

C **int mstat=pvm_mstat(char *host)**

FORTRAN **call pvmfmstat (host, mstat)**

host Variável caracter contendo o nome da máquina.

mstat Variável inteira de retorno com o status da máquina.

VALOR	N	SIGNIFICADO
PvmOk	0	Máquina OK
PvmNoHost	-6	Máquina não existe no ambiente PVM
PvmHostFail	-22	Máquina com problemas

Esta rotina retorna com o status de atividade da máquina **host**, que se imagina estar executando uma processo PVM.

2.13.7 - CANCELAR O AMBIENTE PVM

C **int info=pvm_halt(void)**

FORTRAN **call pvmfhalt(info)**

info Variável inteira de retorno com o status da execução da rotina.

Esta rotina cancela todo o ambiente PVM, cancelando os processos remotos, os **daemons** remotos, os processo locais e o **daemon** local.

Erro info<0

-14 pvmd não responde

2.14 - EXEMPLO DE UM PROGRAMA SPMD

PROGRAMA SERIAL (FORTRAN)

```
      program karp
c This simple program approximates pi by computing pi = integral
c from 0 to 1 of 4/(1+x*x)dx which is approximated by sum from
c k=1 to N of 4 / ((1 + (k-1/2)**2 ).  The only input data required is N.
c NOTE: Comments that begin with "cspmd" are hints for part b of the
c lab exercise, where you convert this into a PVM program.
c Each process could be given a chunk of the interval to do.
      real err, f, pi, sum, w
      integer i, N
      f(x) = 4.0/(1.0+x*x)
      pi = 4.0*atan(1.0)
c Now solicit a new value for N.  When it is 0, then you should depart.
c This would be a good place to unenroll yourself as well.
5     continue
      print *,'Enter number of approximation intervals:(0 to exit)'
      read *, N
      if (N .le. 0) then
        call exit
      endif
      w = 1.0/N
      sum = 0.0
      do i = 1,N
            sum = sum + f((i-0.5)*w)
      enddo
      sum = sum * w
      err = sum - pi
      print *, 'sum, err =', sum, err
      go to 5
      end
```

PROGRAMA PARALELIZADO (FORTRAN)

```
      program karp2

c This simple program approximates pi by computing pi = integral
c from 0 to 1 of 4/(1+x*x)dx which is approximated by sum from
c k=1 to N of 4 / ((1 + (k-1/2)**2 ).  The only input data required is N.

      include 'fpvm3.h'
      parameter (maxproc=100)
      real err, f, pi, sum, w
      integer i, N, tids(0:maxproc)
      common tids

      f(x) = 4.0/(1.0+x*x)
      pi = 4.0*atan(1.0)

c All instances call the startup routine to get their instance number (mynum)

      call startup(nprocs,mynum)

c -------  Each new approximation to pi begins here. -------------------
c (Step 1) Get value N for a new run

5     call solicit (N,nprocs,mynum)

c Step (2): check for exit condition.  Parallel versions: also call fexit().

      if (N .le. 0) then
          call pvmfexit(info)
          call exit
      endif
```

```
c Step (3): do the computation in N steps
c Parallel Version: there are "nprocs+1" instances participating.  Each
c instance should do 1/(nprocs+1) of the calculation.  Since we want
c i = 1..n but mynum = 0, 1, 2..., we start off with mynum+1.

      w = 1.0/N
      sum = 0.0
      do i = mynum+1,N,nprocs+1
         sum = sum + f((i-0.5)*w)
      enddo
      sum = sum * w

      if (mynum.eq.0) then
          print *,'host calculated x=',sum
          do i = 1,nprocs
             call pvmfrecv(-1,222,info)
             call pvmfunpack(REAL4,x,1,1,info)
             print *,'host got x=',x
             sum=sum+x
          enddo
          err = sum - pi
          print *, 'sum, err =', sum, err
      else
          call pvmfinitsend(PVMDEFAULT,info)
          call pvmfpack(REAL4,sum,1,1,info)
          if (info .lt. 0) then
             print *, 'process',mynum,' failed to fpack REAL4'
             stop
          endif
          call pvmfsend(tids(0),222,info)
          if (info .lt. 0) then
             print *,'instance no.', mynum, ' failed to fsend'
             stop
          endif
          print *,'instance',mynum,' sent partial sum',sum, ' to instance 0'
      endif
      go to 5
      end
```

```
      subroutine startup (nprocs,mynum)

      character*32 iproc

      include 'fpvm3.h'
      parameter(maxproc=100)
      integer tids(0:maxproc)
      common tids

      call pvmfmytid(mytid)
      if (mytid.lt.0) then
           print *,'failure in enrolling, error=',mytid
           stop
      endif
      print *,'enrolled, mytid=', mytid

      call pvmfparent(tids(0))
      if (tids(0).lt.0) then
           tids(0)=mytid
           mynum=0
          print *,'How many node programs (1-32)?'
          read *, nprocs
          if (nprocs .gt. 0) then
              call pvmfspawn("pi",PVMDEFAULT,"RS6K",nprocs,tids(1),numt)
              if (numt.ne.nprocs) then
                  print *,'Error in spawning, numt=', numt
                  stop
              endif
              print *,numt,' additional processes were initiated:'
              do i=1,numt
                 print *, tids(i)
              enddo
          else
              print *, 'Thats all folks!'
              call pvmfexit(info)
              stop
          endif
c Broadcast tid info to all tasks
```

```
      call pvmfinitsend(PVMDEFAULT,info)
      call pvmfpack(INTEGER4,nprocs,1,1,info)
      call pvmfpack(INTEGER4,tids,nprocs+1,1,info)
      call pvmfmcast(nprocs,tids(1),333,info)
    else
      call pvmfrecv(tids(0),333,info)
      call pvmfunpack(INTEGER4,nprocs,1,1,info)
      call pvmfunpack(INTEGER4,tids,nprocs+1,1,info)
      do i=1, nprocs
         if (mytid.eq.tids(i)) mynum = i
      enddo
    endif
10  return
    end

    subroutine solicit (N,nprocs,mynum)

    include 'fpvm3.h'
    parameter (maxproc=100)
    integer tids(0:maxproc)
    common tids
    if (mynum .eq. 0) then
      print *,'Enter number of approximation intervals:(0 to exit)'
      read *, N
      call pvmfinitsend(PVMDEFAULT,info)
      call pvmfpack(INTEGER4,N,1,1,info)
      call pvmfpack(INTEGER4,nprocs,1,1,info)
      call pvmfmcast(nprocs,tids(1),111,info)
    else
      call pvmfrecv(-1,111,info)
      call pvmfunpack(INTEGER4,N,1,1,info)
      call pvmfunpack(INTEGER4,nprocs,1,1,info)
    endif
    return
    end
```

PROGRAMA SERIAL (C)

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define f(x) ((float)(4.0/(1.0+x*x)))
#define pi ((float)(4.0*atan(1.0)))
main()
{
    /* This simple program approximates pi by computing pi = integral
     * from 0 to 1 of 4/(1+x*x)dx which is approximated by sum
     * from k=1 to N of 4 / ((1 + (k-1/2)**2 ).  The only input data
     * required is N.
     * Each process is given a chunk of the interval to do. */
float err, sum, w;
int i, N;
void startup();
     /* startup(&mynum, &nprocs)
      * Now solicit a new value for N.  When it is 0, then you should depart.
      * This would be a good place to unenroll yourself as well.*/
printf ("Enter number of approximation intervals:(0 to exit)\n");
scanf("%d",&N);
while (N > 0)
  {
   w = 1.0/(float)N;
   sum = 0.0;
   for (i = 1; i <= N; i++)
     sum = sum + f(((float)i-0.5)*w);
   sum = sum * w;
   err = sum - pi;
   printf("sum, err = %7.5f, %10e\n", sum, err);
   printf ("Enter number of approximation intervals:(0 to exit)\n");
   scanf("%d",&N);
   }
}
```

PROGRAMA PARALELIZADO (C)

```
/* This simple program approximates pi by computing pi = integral
 * from 0 to 1 of 4/(1+x*x)dx which is approximated by sum
 * from k=1 to N of 4 / ((1 + (k-1/2)**2 ).  The only input data required is N. */
 #include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "pvm3.h"          /* PVM 3.0 include file */
#define f(x) ((float)(4.0/(1.0+x*x)))
#define pi ((float)(4.0*atan(1.0)))
#define MAXPROCS      32          /* max number of node programs */
main() {
float   err,
        sum,
        w,
        x;
int     i,
        N,
        info,
        mynum,
        nprocs,
        tids[MAXPROCS+1];
void    startup(),
        solicit();

startup(&mynum, &nprocs, tids);

printf(""); fflush(stdout);

solicit (&N, &nprocs, mynum, tids);

if (N <= 0) {
   printf("node %d left\n", mynum);
   pvm_exit();
   exit(0);
   }
while (N > 0) {
```

```
        w = 1.0/(float)N;
        sum = 0.0;
        for (i = mynum+1; i <= N; i+=nprocs+1)
             sum = sum + f(((float)i-0.5)*w);
        sum = sum * w;
        if (mynum==0) {
          printf ("host calculated x = %7.5f\n", sum);
          for (i=1; i<=nprocs; i++) {
               info = pvm_recv(-1, 222);
               info = pvm_upkfloat(&x, 1, 1);
               printf ("host got x = %7.5f\n", x);
               sum=sum+x;
           }
          err = sum - pi;
          printf("sum, err = %7.5f, %10e\n", sum, err); fflush(stdout);
          }
        else {
          info = pvm_initsend(PvmDataDefault);
          info = pvm_pkfloat(&sum, 1, 1);
          if (info < 0) {
             printf ("process %d failed to putnfloat\n", mynum);
             exit(0);
           }
          info = pvm_send(tids[0], 222);
          if (info < 0) {
             printf ("instance no. %d failed to snd\n", i);
             exit(0);
          }
          printf ("inst %d sent partial sum %7.2f to inst 0\n",mynum, sum);
          fflush(stdout);
        }
    solicit (&N, &nprocs, mynum, tids);
}
pvm_exit();
}
```

void startup (pmynum, pnprocs, tids)

```
int *pmynum, *pnprocs, tids[MAXPROCS+1];
{
    int     i,
            mynum,
            nprocs,
            info,
            mytid,
            numt,
            parent_tid;

    mytid = pvm_mytid();
    if (mytid < 0) {
      printf("failure in enrolling instance\n");
      exit(0);
     }

    parent_tid = pvm_parent();
    if (parent_tid == PvmNoParent) {
      mynum = 0;
      tids[0] = mytid;
      printf ("How many node programs (1-32)?\n");
      scanf("%d", &nprocs);
      if (nprocs > MAXPROCS) {
        printf("Number of node programs exceeds limit...try again!\n");
        exit(0);
      }
      if (nprocs < 0) {
        printf("Number of node programs is below zero...try again!\n");
        exit(0);
      }
      numt = pvm_spawn("pi", NULL, PvmTaskDefault, "", nprocs, &tids[1]);
      if (numt != nprocs) {
        printf ("Error in spawning, numt= %d\n",numt);
        exit(0);
      }
      printf ("%d additional processes were initiated\n", numt);
```

```
    for (i=0; i<=nprocs; i++)
      printf("task %d tid = %d \n",i,tids[i]);
    info = pvm_initsend(PvmDataDefault); /* broadcast tid info to all tasks */
    info = pvm_pkint(&nprocs,1,1);
    info = pvm_pkint(tids,nprocs+1,1);
    info = pvm_mcast(&tids[1], nprocs, 333);
  }
  else {
    info = pvm_recv(parent_tid, 333);
    info = pvm_upkint(&nprocs, 1, 1);
    info = pvm_upkint(tids, nprocs+1, 1);
    for (i = 1; i <= nprocs; i++) {
      if (mytid == tids[i]) mynum=i;
     }
  }
}
```

void solicit (pN, pnprocs, mynum, tids)

```
int *pN, *pnprocs, mynum, tids[MAXPROCS+1];
{
  int info;
  if (mynum == 0) {
    printf("Enter number of approximation intervals:(0 to exit)\n");
    scanf("%d", pN);
    info = pvm_initsend(PvmDataDefault);
    info = pvm_pkint(pN,1,1);
    info = pvm_pkint(pnprocs,1,1);
    info = pvm_mcast(&tids[1], *pnprocs, 111);
  }
  else {
    info = pvm_recv(tids[0], 111);
    info = pvm_upkint(pN, 1, 1);
    info = pvm_upkint(pnprocs,1, 1);
  }
}
```

2.15 - GRUPOS DE PROCESSOS DINÂMICOS

2.15.1 - DEFINIÇÕES

É possível definir um ou vários grupos de processos, e dá nome a esses grupos.

Facilita e simplifica a comunicação com, e entre os grupos.

Simplifica a sincronização dos processos.

Um processo pode pertencer a vários grupos, simultâneamente.

Um processo pode se juntar ou deixar um grupo, em qualquer instante.

Básicamente, se o seu programa necessitar que haja uma sincronização entre processos para efetuar algum procedimento ou dar início a novos processos, será necessário a definição de grupos de processos.

2.15.2 - COMPONENTES DO PVM

É necessário acessar uma biblioteca especial do pvm para se trabalhar com grupos de processos dinâmicos.

No momento da compilação deve-se incluir mais uma biblioteca para linkedição - **libgpvm3.a**

Assim como o processo **daemon**, no momento da execução de uma aplicação PVM, que utilize grupo de processos dinâmicos, será inicializado o processo **pvmgs**, que será o servidor de grupos.

Esse processo será inicializado automáticamente e aparecerá como um outro processo normal.

O servidor de grupos mantém uma tabela com informações dos grupos:

- Nome do grupo;
- Número de processos dentro do grupo;
- O número de identificação dos processos;
- O status da sincronização dos processos.

2.15.3 - INICIALIZAR E ADERIR A UM GRUPO

C **int inum=pvm_joingroup (char *group)**

FORTRAN **call pvmfjoingroup (group, inum)**

group Variável caracter com o nome de um grupo a ser criado ou que já exista.

inum Váriavel inteira que retorna o número de identificação do processo dentro do grupo.

Esta rotina insere um processo num denominado grupo, retornando um número de identificação do processo, no grupo. Se for o primeiro processo no grupo, ele criará o grupo e inicializará o servidor de grupos **pvmgs**.

Erro info < 0

-2 Argumento inválido
-14 pvmd não responde
-18 Processo já faz parte do grupo

2.15.4 - SAIR E FINALIZAR UM GRUPO

C **int inum=pvm_lvgroup (char *group)**

FORTRAN **call pvmflvgroup (group, info)**

group Variável caracter com o nome de um grupo a ser criado ou que já exista.

inum Váriavel inteira que retorna com o status da execução da rotina.

Essa rotina tira um processo de um denominado grupo. Se for o último processo no grupo, ele finalizará o grupo, no entanto, o servidor de grupos, **pvmgs**, permanece até ser cancelado o ambiente PVM.

Erro info < 0

-2 Argumento inválido
-14 pvmd não responde
-19 Não existe grupo

2.15.5 - VERIFICAR O TAMANHO DE UM GRUPO

C **int size=pvm_gsize(char *group)**

FORTRAN **call pvmfgsize(group, size)**

group Variável caracter que identifica um grupo que já exista.

size Variável inteira de retorno com o número de processos presentes no grupo.

Esta rotina retorna o número de processos de um denominado grupo, no instante em que a rotina foi chamada.

Erro size<0

-2 Argumento inválido
-14 pvmd não responde

2.15.6 - SINCRONIZAR PROCESSOS

C **int info=pvm_barrier(char *group,int count)**

FORTRAN **call pvmfbarrier (group, count, info)**

group Variável caracter com o nome de um grupo que já exista. O processo que chama a rotina, deve fazer parte desse grupo.

count Variável inteira que especifica o número de processos dentro do grupo identificado. (**-1**, todos os processos do grupo).

info Variável inteira de retorno com o status de execução da rotina.

Esta rotina bloqueia a execução do processo até que todos os outros processos, do mesmo grupo, também, façam uma chamada a rotina **pvmfbarrier**. O valor de **count** deverá ser o mesmo para todas as chamadas.

Erro info < 0

-2 Argumento invalido
-14 pvmd não responde
-19 Não existe grupo
-20 Não existe o processo no grupo

2.15.7 - ENVIAR MENSAGENS PARA UM GRUPO

C **int info=pvm_bcast(char *group, int msgtag)**

FORTRAN **call pvmfbcast(group, msgtag, info)**

group Variável caracter com o nome de um grupo que já exista.

msgtag Variável inteira com o rótulo da mensagem que será enviada.

info Vriável inteira de retorno com o status de execução da rotina.

Esta rotina envia uma mensagem armazenada no "buffer", para todos os processos de um denominado grupo. O conteúdo da mensagem é distingüido pelo valor de **msgtag**.

Erro info < 0

-2 Argumento invalido
-14 pvmd não responde
-19 Grupo não existe

2.15.8 - EFETUAR UMA OPERAÇÃO DE REDUÇÃO

C **int info=pvm_reduce(void (*func)(), void *data, int count, int count, int datatype, int msgtag, char *group, int root)**

FORTRAN **call pvmfreduce(func, data, count, datatype, msgtag, group, root, info)**

func Função que define a operação que será efetuada num determinado dado dos processos. Pode-se definir uma função ou utilizar funções pré-definidas pelo PVM:

PvmMax	PvmMin	PvmSum	PvmProduct

data Variável na qual se deseja efetuar uma operação de redução. Se for um vetor ou matriz, especificar a posição inicial.

count Número de elementos da variável **data**, se for um vetor ou matriz.

datatype Tipo do dado da variável **data**.

C	FORTRAN
PVM_BYTE	BYTE1
PVM_SHORT	INT2
PVM_INT	INT4
PVM_FLOAT	REAL4
PVM_CPLX	COMPLEX8
PVM_DOUBLE	REAL8

msgtag Variável inteira com o rótulo da mensagem que será enviada.

group Variável caracter com o nome de um grupo que já exista.

root Variável inteira com o número de identificação do processo, no grupo, que receberá o resultado da operação de redução.

info Variável inteira de retorno com o status de execução da rotina.

Esta rotina executa uma operação de redução global em todos os processos de um grupo. Todos os processos tem que efetuar uma chamada a essa rotina, com os dados pertinentes ao seu processo. O resultado é acumulado no campo **data,** do processo especificado pelo campo **root.**

OBS: Esta rotina não bloqueia a execução do processo.

Erro info<0

-2 Argumento inválido
-14 pvmd não responde
-21 Processo não pertence ao grupo

2.16 - PVM EM AMBIENTE HETEROGÊNEO

Para trabalhar com o PVM em um ambiente heterogêneo de máquinas será necessário seguir alguns procedimentos:

1 - Instalar a mesma versão do PVM em todas as arquiteturas que compoêm o ambiente heterogêneo de máquinas;

2 - Compilar todos os programas: mestre, escravo ou SPMD, em todas as arquiteturas do ambiente PVM heterogêno;

3 - Definir no programa principal, na rotina **spawn**, quais os processos, e onde serão executados;

4 - Utilizar uma rotina **spawn**, para cada arquitetura diferente.

No ambiente CENAPAD-SP, existem duas arquiteturas onde é possível executar uma aplicação PVM: **ALPHA** e **RISC/6000**.

1 - Execução de uma aplicação PVM a partir da arquitetura RS6K

- Altere o programa principal (processo "pai"), modificando a rotina **pvm_spawn/pvmfspawn**, para identificar a arquitetura onde serão executados os processos "filhos";

```
program pai
include "fpvm3.h"
parameter (NTASKS=6)
...
call pvmfspawn("filho",PVMARCH,"RS6K",4,tids(1),erro)
call pvmfspawn("filho",PVMARCH,"ALPHA",2,tids(5),erro)
...
```

- Compile uma cópia do programa secundário (processo "filho"), na arquitetura ALPHA;

- Inicializar uma máquina da arquitetura ALPHA no ambiente de daemons do PVM;

```
% pvm
pvm> add athos.cna.unicamp.br
```

2 - Execução de uma aplicação PVM a partir da arquitetura ALPHA

- Altere o programa principal (processo "pai"), modificando a rotina **pvm_spawn/pvmfspawn**, para identificar a arquitetura onde serão executados os processos "filhos";

```
program pai
include "fpvm3.h"
parameter (NTASKS=6)
...
call pvmfspawn("filho",PVMARCH,"ALPHA",4,tids(1),erro)
call pvmfspawn("filho",PVMARCH,"RS6K",2,tids(5),erro)
...
```

- Compile uma cópia do programa secundário (processo "filho"), na arquitetura RS6K;

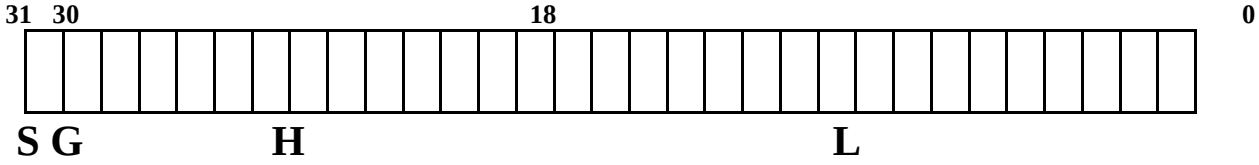
- Inicializar uma máquina da arquitetura RS6K no ambiente de daemons do PVM;

```
% pvm
pvm> add delos.cna.unicamp.br
```

2.17 - COMO O PVM TRABALHA

Identificação de Processos

O PVM utiliza um *identificador de processo* **(TID)**, para endereçar daemons, processos e grupos. O **TID** possui quatro campos, totalizando 32 bits.



H Esse campo identifica a máquina no ambiente PVM. O número máximo de máquinas num ambiente PVM é de $2^{H}-1$ **(4095)**.

L Esse campo identifica o processo em uma máquina do ambiente PVM. O número máximo de processos por máquina é de $2^{L}-1$.

IDENTIFICAÇÃO	S	G	H	L
Processo	0	0	1...Hmax	1...Lmax
Pvmd	1	0	1...Hmax	0
Grupo	0	1	1...Hmax	0...Lmax
Error	1	1	negativo	negativo

PVM daemon

Um pvmd é executado em cada máquina do ambiente PVM;

É automáticamente configurado como ***mestre*** ou ***escravo***;

É inicializado um arquivo para mensagens de erro no **/tmp**, como **/tmp/pvml.*uid*** ;

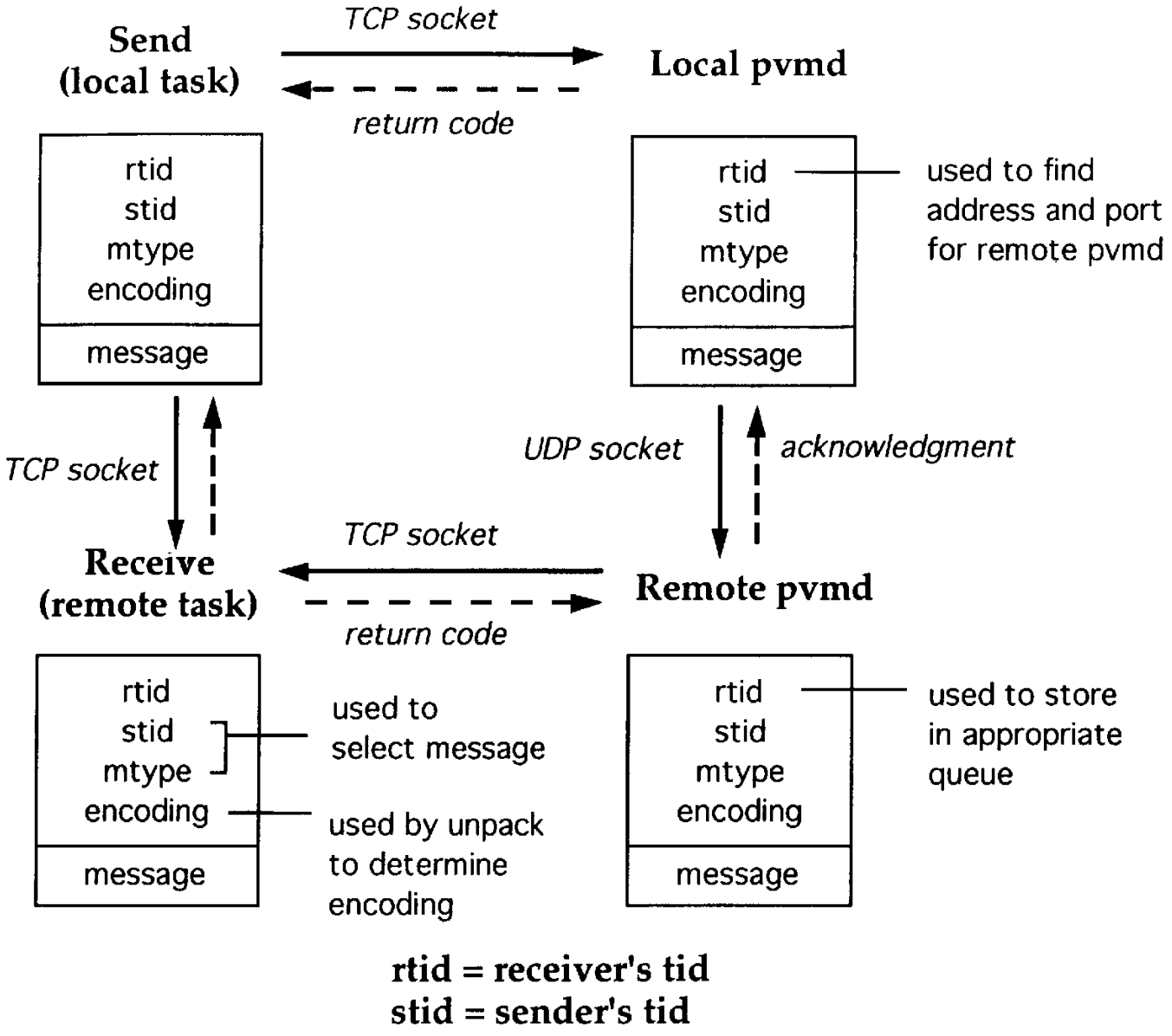
O pvmd de um usuário não interage com o pvmd de outros usuários;

Cada pvmd mantêm uma lista com a identificação de todos os processos que estão sobre o seu controle;

O pvmd trabalha como um controlador e roteador de mensagens, administrando o processo e detectando falhas no ambiente.

Protocolos

A comunicação em PVM é baseada em TCP (*Transmission Control Protocol*) ou UDP (*User Datagram Protocol*).



rtid = receiver's tid
stid = sender's tid

UDP

(+) É escalável. Uma conexão UDP pode-se comunicar com qualquer quantidade de outras conexões UDPs remotas.
(+) Overhead. A inicialização do protocolo UDP não exije comunicação.
(+) Tolerante a falhas.

(-) O serviço de envio/recebimento não é confiável;
(-) Duplica e reordena os pacotes;
(-) É necessario um mecanismo de repetição e acusação de envio/recebimento.
(-) Limita o tamanho do pacote, fragmentando mensagens grandes.

TCP

(+) A transmissão de dados é confiável devido a implementação do protocolo;
(+) Boa performance;

(-) Não é escalável;
(-) Overhead, **N** pvmds necessita de **N(N-1)/2** conexões TCP;
(-) Dificuldades para controlar falhas.

3 - AUXÍLIO NA INTERNET

http://www.unicamp.br/cenapad
(Home page do CENAPAD-SP)

http://www.epm.ornl.gov/pvm
(Home page do pvm)

consult@cenapad.unicamp.br
(e-mail de dúvidas do CENAPAD-SP)

pvm@msr.epm.ornl.gov
(e-mail de dúvidas do pvm em ORNL)

news:comp.parallel.pvm
(Notícias, trabalhos, dúvidas do grupo de usuários de pvm)

4 - CONCLUSÃO

Para se reduzir a sobrecarga de comunicação que ocorre ao se executar processos concorrentes, é necessário se ter em mente as seguintes regras:

Inicialize as "tasks" uma única vez, para se reduzir a sobrecarga que é gerado ao se inicializar ou terminar uma "task".

Maximize o trabalho que cada "task" pode efetuar.

Envie o mínimo de dados possíveis para cada "task".

Para sincronizar "tasks" utilize das primitivas do pvm.

Não utilize o format de conversão nas rotinas de envio de mensagens.

5 - BIBLIOGRAFIA

LIVROS: **Computer Architecture and Parallel Processing**
Kai Hwang, Fayé A. Briggs
McGraw-Hill Book Company - 1985

Principles of Parallel and Multiprocessing
Georges R. Desrochers
McGraw-Hill Book Company - 1987

MANUAIS: **PVM 3 User's Guide and Referance Manual**
Al Geist, Adam Beguelim, Jack Dongarra
Oak Ridge National Laboratory, Tennessee - USA, Maio - 1994

IBM AIX PVMe User's Guide and Subroutine Reference
IBM, Kingston, NY - USA
Abril - 1994

Introduction to PVM
Blaise Barney
Cornell Theory Center, Ithaca, NY - USA, Abril - 1994

Design of the PVM Daemon and Libraries
Menbers of the consulting staff
Cornell Theory Center, Ithaca, NY - USA, Março - 1994

PVM Performance Issues
Steven R. Lantz
Cornell Theory Center, Ithaca, NY - USA, Abril - 1994

PALESTRAS: **Introdução ao Processamento Paralelo**
Philippe O. A. Navaux - CPD - UFRGS
I SIMPÓSIO BRASILEIRO DE ARQUITETURA DE COMPUTADORES - 1987

Overview of Parallel Processing
Cornell Theory Center, Ithaca - NY
Abril - 1984

Parallel Performance Expectation
Cornell Theory Center, Ithaca - NY
Abril - 1984

Máquinas disponíveis para o processamento "on-line" com o PVM:

spirit.cna.unicamp.br
belinda.cna.unicamp.br
mafalda.cna.unicamp.br
sandman.cna.unicamp.br
calvin.cna.unicamp.br
snoopy.cna.unicamp.br

1º LABORATÓRIO

1 - No diretório de execução do PVM (**~/pvm3/bin/LINUX**), edite um arquivo contendo o nome das máquinas que voce escolheu para processar o seu programa com o PVM.

% **cd ~/pvm3/bin/LINUX**
% **vi < arquivo de máquinas >**

2 - Copie os arquivos exemplos do 1º laboratório.

% **cp ~curso/curso/pvm/lab01/* .**

3 - Analise os programas exemplos mestre/escravo, com relação as chamadas das rotinas PVM e da lógica utilizada.

4 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

% **make -f make.hello.c**
ou
% **make -f make.hello.f**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento paralelo:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle <**enter**> duas vezes

6 - Inicialize a console do PVM.

```
% pvm
pvm>
```

7 - Verifique se todos os "daemons" foram inicializados. Se ok, saia da console PVM sem cancelar os "daemons" com o comando "quit"

```
pvm> conf
  .     .
  .     .
pvm> quit
```

8 - Execute o programa mestre.

```
% hello.master
```

9 - Verifique e analise os resultados no arquivo de mensagens do PVM gerado no diretório temporário para o seu "user id".

```
% id -u
1010
% cd /tmp
% more pvml.1010
```

OBS: O resultado será várias mensagens de "Hello Back..." enviadas de volta por cada processo inicializado.

10 - Retorne ao seu diretório de trabalho. Execute novamente o programa, mas agora, a partir da console pvm com o comando "spawn", redirecionando a saída dos processos para a console.

```
% cd ~/pvm3/bin/LINUX
% pvm
pvm> spawn -> hello.master
          .         .
          .         .
pvm>
```

11 - Mexa na configuração dos "daemons" pvm. Adicione ou retire uma ou mais máquinas da

configuração atual. Execute novamente o programa, a partir da console.

pvm> add <máquina>

ou

pvm> delete <máquina>
pvm> spawn -> hello.master

12 - Lembre-se, o PVM continua rodando em "background". Se voce não deseja executar mais nenhum programa, cancele os "daemons" PVM:

pvm> halt
%

2º LABORATÓRIO

1 - Copie os arquivos exemplos do 2º laboratório.

```
% cd pvm3/bin/LINUX
% cp ~curso/curso/pvm/lab02/*  .
```

2 - Analise os programas exemplos mestre/escravo, com relação as chamadas das rotinas PVM e da lógica utilizada.

3 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

```
% make  -f  make.pvm.ex1.c
                ou
% make  -f  make.pvm.ex1.f
```

4 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

```
% pvmd3 <arquivo de máquinas>  &
```

OBS: tecle **<enter>** duas vezes

5 - Inicialize a console do PVM.

```
% pvm
pvm>
```

6 - Verifique se todos os "daemons" foram inicializados. Se ok, saia da console PVM sem cancelar os "daemons" com o comando "quit"

```
pvm> conf
 .     .
 .     .
pvm> quit
```

7 - Execute o programa mestre utilizando o comando "time" do AIX para verificar o tempo de execução.

% **time pvm.ex1.master**

8 - Verifique e analise os resultados que serão mostrados na sua tela.

9 - Adicione ou retire uma ou mais máquinas da configuração atual. Execute novamente o programa com o comando "time". Compare os tempos de processamento para cada configuração de máquinas que voce utilizou.

% **pvm**
pvm> add <máquina>

ou

pvm> delete <máquina>
pvm> quit
% **time pvm.ex1.master**

10 - Edite e modifique o programa mestre e o programa escravo. Altere os parâmetros NTASKS=4 e ARRAYSIZE=100000. Recompile o programa e execute-o novamente utilizando o comando "time".

% **vi <programa mestre>**
.
.
% **vi <programa escravo>**
.
.
% **make -f make.pvm.ex1.f**
ou
% **make -f make.pvm.ex1.c**
% **time pvm.ex1.master**

11 - Edite e altere novamente o programa mestre. Altere o parâmetro da rotina pvm **"initsend"** de PVMDEFAULT para PVMRAW. Recompile e execute novamente o programa mestre com o comando "timex". Compare os tempos de processamento com o item anterior.

```
% vi <programa mestre>
                .
                .
% make -f  make.pvm.ex1.f
                ou
% make -f  make.pvm.ex1.c
% time pvm.ex1.master
```

12 - Lembre-se, o PVM continua rodando em "background". Se voce não deseja executar mais nenhum programa, cancele os "daemons" PVM:

```
% pvm
pvm> halt
```

3º LABORATÓRIO

1 - Caminhe para o diretório de trabalho do PVM

% **cd ~/pvm3/bin/LINUX**

2 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle **<enter>** duas vezes

3 - Inicialize a interface gráfica do PVM.

% **xpvm**

ATENÇÃO!
A sua máquina deverá estar configurada adequadamente, para possibilitar a execução de uma Xwindow de outra máquina, e a outra máquina (máquina "logada" no CENAPAD), deverá saber para quem enviar o DISPLAY da execução do software.

Máquina residente:

% **xhost +** **ou** % **xhost <endereço da máquina do CENAPAD>**

Máquina logada:

% **setenv DISPLAY <endereço da máquina residente>:0.0**

4 - Clique no "buttom" **Views** e ative as janelas **Call Trace** e **Task Output**.

5 - Clique no "buttom" **Tasks** e depois em **Spawn**.

6 - Na janela aberta, no campo **Command**, digite o nome do arquivo executável do pvm**,** **hello.master**, e depois, clique em **Start**.

7 - Visualize a execução gráfica do programa por entre as máquinas solicitadas.

8 - Repita o mesmo procedimento, a partir do item 5, para o arquivo executável do pvm, **pvm.ex1.master**

4º LABORATÓRIO

1 - Caminhe para o diretório de trabalho do PVM

% **cd ~/pvm3/bin/LINUX**

2 - Copie os arquivos exemplos do 4º laboratório.

% **cp ~curso/curso/pvm/lab03/* .**

3 - Edite e altere os programas mestre e escravo. Subistitua os campos assinalados "====>" pelas respectivas rotinas pvm, já com os parâmetros principais indicados entre parênteses. (Programas em **FORTRAN** e **C**, listados, e anexados ao laboratório).

4 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

% **make -f make.matriz.f**
ou
% **make -f make.matriz.c**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle **<enter>** duas vezes

6 - Inicialize a console do PVM.

% **pvm**
pvm>

7 - Verifique se todos os "daemons" foram inicializados. Se ok, saia da console PVM sem cancelar os "daemons" com o comando "quit"

pvm> conf

. .
. .

pvm> quit

7 - Execute o programa mestre.

% **matriz.m**

8 - Verifique e analise os resultados que serão mostrados na sua tela.

9 - Lembre-se, o PVM continua rodando em "background". Se voce não deseja executar mais nenhum programa, cancele os "daemons" PVM:

% **pvm**
pvm> halt

PROGRAMA MESTRE (C)

```
/**************************************************************************
*                    PVM TEMPLATE CODES
* FILE: matriz.m.c
* OTHER FILES: matriz.w.c, make.matriz.c
* DESCRIPTION:  PVM matrix multiply example code master task.  C version.
*   In this template code, the master program acts as the parent and spawns
*   NPROC worker tasks.  The first worker task is spawned on a specific machine.
*   The master program performs the matrix multiply by sending all of matrix B
*   to every worker task and then partitioning rows of matrix A among the
*   workers.  The worker tasks perform the actual multiplications and send back
*   to the master task their respective results.
*   NOTE1:  C and Fortran versions of this code differ because of the way
*     arrays are stored/passed.  C arrays are row-major order but Fortran
*     arrays are column-major order.
*   NOTE2: This matrix multipy can be improved by:
*     1) more efficient memory use by the worker program (C version only)
*     2) checking of return codes
*     3) insuring that all workers are operational before message passing
*     4) reading the number of workers at run time
*     See mm2.m.c and mm2.w.c for the improved version.
* PVM VERSION: 3.x
* AUTHOR: Roslyn Leibensperger
* LAST REVISED: 5/27/93 bbarney
**************************************************************************/
#include <stdio.h>
========> Include PVM    /* PVM version 3.0 include file */

#define NPROC 4                 /* number of PVM worker tasks to spawn */
#define NRA 62                  /* number of rows in matrix A */
#define NCA 15                  /* number of columns in matrix A */
#define NCB 7                /* number of columns in matrix B */
main() {
int mtid,                       /* PVM task id of master task */
   wtids[NPROC],                /* array of PVM task ids for worker tasks */
   mtype,                    /* PVM message type */
   rows,                     /* rows of matrix A sent to each worker */
   averow, extra, offset,          /* used to determine rows sent to each worker */
   rcode, i, j;              /* misc */
double a[NRA][NCA],             /* matrix A to be multiplied */
     b[NCA][NCB],               /* matrix B to be multiplied */
```

```
      c[NRA][NCB];                    /* result matrix C */
char thishost[35];                    /* name of selected master */

/* enroll this task in PVM */
```

========> **ROTINA PVM (VARIAVEL: mtid)**

```
/* The master task now spawns worker tasks by calling pvm_spawn.  The unique */
/* worker task ids are stored in the wtids array.  The first worker task is */
/* spawned on a specific machine.  The return code tells the number of tasks */
/* successfully spawned, and in this example, is not checked for errors.     */
for (i=0; i<NPROC; i++) {
  if (i==0) {
     printf ("Enter selected hostname - must match PVM config: ");
     scanf("%s", thishost);
```

========> **ROTINA PVM (VARIAVEIS: rcode, thishost, &wtids[0])**

```
     }
  else
```

========> **ROTINA PVM (VARIAVEIS: rcode, &wtids[i])**

```
  }

/* initialize A and B */
for (i=0; i<NRA; i++)
   for (j=0; j<NCA; j++)
      a[i][j]= i+j;
for (i=0; i<NCA; i++)
   for (j=0; j<NCB; j++)
      b[i][j]= i*j;
averow = NRA/NPROC;
extra = NRA%NPROC;
offset = 0;
mtype = 1;
/* send data to the worker tasks */
for (i=0; i<NPROC; i++)       {                    /* for each worker task */
  rows = (i < extra) ? averow+1 : averow;          /* Find #rows to send from A */
  /* next call initializes send buffer and specifies to do XDR data format */
  /* conversion only in heterogenous environment */
```

========> **ROTINA PVM (VARIAVEL: rcode)**

```
  /* next four calls pack values into the send buffer */
========> ROTINA PVM ( VARIAVEIS: rcode, &offset )      /* starting pos. in matrix */
========> ROTINA PVM ( VARIAVEIS: rcode, rows )          /* #rows of A to send */
========> ROTINA PVM ( VARIAVEIS: rcode, &a[offset][0], rows*NCA ) /* rows of A */
========> ROTINA PVM ( VARIAVEIS: rcode, b, NCA*NCB )        /* all of B */

  /* send contents of send buffer to worker task  */
========> ROTINA PVM ( VARIAVEIS: rcode, wtids[i], mtype )

  offset = offset + rows;
}
/* wait for results from all worker tasks */
mtype = 2;                                          /* set message type */
for (i=0; i<NPROC; i++)        {                            /* do once for each worker */

========> ROTINA PVM ( VARIAVEIS: rcode, mtype )   /* receive message from worker*/
========> ROTINA PVM ( VARIAVEIS: rcode, &offset ) /* starting pos. in matrix */
========> ROTINA PVM ( VARIAVEIS: rcode, &rows )   /* #rows sent */
========> ROTINA PVM ( VARIAVEIS: rcode, &c[offset][0], rows*NCB) /* rows matrix C*/

   }
/* print results */
for (i=0; i<NRA; i++) {
   printf("\n");
   for (j=0; j<NCB; j++)
      printf("%6.2f   ", c[i][j]);
   }
printf ("\n");
/* task now exits from PVM */

=======> ROTINA PVM ( VARIAVEL: rcode )

}
```

PROGRAMA ESCRAVO (C)

```
/**************************************************************************
*                    PVM TEMPLATE CODES
* FILE: matriz.w.c
* DESCRIPTION: See matriz.m.c
* PVM VERSION: 3.x
* LAST REVISED: 5/27/93 bbarney
**************************************************************************/
#include <stdio.h>
#include <malloc.h>
========> Include PVM          /* PVM version 3.0 include file */

#define NRA 62                  /* number of rows in matrix A */
#define NCA 15                  /* number of columns in matrix A */
#define NCB 7                   /* number of columns in matrix B */

main() {
int wtid,               /* PVM task id of this worker program */
   mtid,                        /* PVM task id of parent master program */
   mtype,                   /* PVM message type */
   rows,                    /* number of rows in matrix a sent to worker */
   offset,                      /* starting position in matrix */
   rcode, i, j, k;          /* misc */
double a[NRA][NCA],             /* matrix A to be multiplied */
     b[NCA][NCB],               /* matrix B to be multiplied */
     c[NRA][NCB];               /* result matrix C */

/* enroll worker task */
========> ROTINA PVM ( VARIAVEL: wtid )

/* Receive message from master */
mtype = 1;                      /* set message type */

========> ROTINA PVM ( VARIAVEL: mtid )          /* get task id for master process */
========> ROTINA PVM ( VARIAVEIS: rcode, mtid, mtype )/* wait to receive from master*/
========> ROTINA PVM ( VARIAVEIS: rcode, &offset ) /* start pos. in A and C matrices */
========> ROTINA PVM ( VARIAVEIS: rcode, &rows )  /* #rows in matrix A sent */
========> ROTINA PVM ( VARIAVEIS: rcode, a, rows*NCA )  /* our share of matrix A */
========> ROTINA PVM ( VARIAVEIS: rcode, b, NCA*NCB )  /* contents of matrix B */

printf("worker task id = %d received %d rows from A\n", wtid, rows);
```

```
/* do matrix multiply */
for (k=0; k<NCB; k++)
  for (i=0; i<rows; i++) {
    c[i][k] = 0.0;
    for (j=0; j<NCA; j++)
      c[i][k] = c[i][k] + a[i][j] * b[j][k];
  }
/* Set up send message to master. */
mtype = 2;                              /* set message type */
```

========> ROTINA PVM (VARIAVEL: rcode) /* initialize send buffer */
========> ROTINA PVM (VARIAVEIS: rcode, &offset) /* pos. in result matrix */
========> ROTINA PVM (VARIAVEIS: rcode, &rows) /* number of rows being sent */
========> ROTINA PVM (VARIAVEIS: rcode, c, rows*NCB) /*part of result matrix C */

```
/* send to master */
```

========> ROTINA PVM (VARIAVEIS: rcode, mtid, mtype)

```
/* exit PVM */
```

========> ROTINA PVM (VARIAVEL: rcode)

```
}
```

PROGRAMA MESTRE (FORTRAN)

```
C**************************************************************************
C                    PVM TEMPLATE CODES
C FILE: matriz.m.f
C OTHER FILES: matriz.w.f, make.matriz.f
C DESCRIPTION:  PVM matrix multiply example code master task.  Fortran version.
C   In this template code, the master program acts as the parent and spawns
C   NPROC worker tasks.  The first worker task is spawned on a specific machine.
C   The master program performs the matrix multiply by sending all of matrix A
C   to every worker task and then partitioning columns of matrix B among the
C   workers.  The worker tasks perform the actual multiplications and send back
C   to the master task their respective results.
C   NOTE1:  C and Fortran versions of this code differ because of the way
C     arrays are stored/passed.  C arrays are row-major order but Fortran
C     arrays are column-major order.
C   NOTE2: This matrix multipy can be improved by:
C     1) more efficient memory use by the worker program (C version only)
C     2) checking of return codes
C     3) insuring that all workers are operational before message passing
C     4) reading the number of workers at run time
C PVM VERSION: 3.x
C AUTHOR: Blaise Barney - adapted from C version
C LAST REVISED: 5/27/93 bbarney
C**************************************************************************
C Explanation of constants and variables used in this program:
C   NPROC             = number of PVM worker tasks to spawn
C   NRA               = number of rows in matrix A
C   NCA               = number of columns in matrix A
C   NCB               = number of columns in matrix B
C   mtid              = PVM task id of master task
C   wtids             = array of PVM task ids for worker tasks
C   mtype             = PVM message type
C   cols              = columns of matrix B sent to each worker
C   avecol, extra     = used to determine columns sent to each worker
C   offset            = starting position within the matrix
C   rcode, i, j       = misc.
C   a           = matrix A to be multiplied
C   b                 = matrix B to be multiplied
C   c           = result matrix C
C   thishost    = name of selected master
C ----------------------------------------------------------------------
```

```
      program matriz_master
C     PVM Version 3.0 include file
```

========> **Include PVM**

```
      parameter (NPROC = 4)
      parameter (NRA = 62)
      parameter (NCA = 15)
      parameter (NCB = 7)

      integer       mtid, wtids(NPROC), mtype, cols, avecol, extra, offset,
     &              rcode, i, j
      real*8        a(NRA,NCA), b(NCA,NCB), c(NRA,NCB)
      character*35 thishost

C  Enroll this task in PVM
```

========> **ROTINA PVM (VARIAVEL : mtid)**

```
C  The master task now spawns worker tasks by calling pvm_spawn.  The unique
C  worker task ids are stored in the wtids array.  The first worker task is
C  spawned on a specific machine.  The return code tells the number of tasks
C  successfully spawned, and in this example, is not checked for errors.
      do 20 i=1, NPROC
      if (i .eq. 1) then
        write(*,9)
 9       format('Enter selected hostname - must match PVM config: ',$)
        read (*, 10) thishost
 10     format (a35)
```

========> **ROTINA PVM (VARIAVEIS: thishost wtids(1))**

```
      else
```

========> **ROTINA PVM (VARIAVEL: wtids(i))**

```
      endif
 20   continue

C  Initialize A and B
      do 30 i=1, NRA
        do 30 j=1, NCA
        a(i,j) = (i-1)+(j-1)
 30   continue
```

```
      do 40 i=1, NCA
        do 40 j=1, NCB
        b(i,j) = (i-1)*(j-1)
 40   continue
      avecol = NCB/NPROC
      extra = mod(NCB, NPROC)
      offset = 1
      mtype = 1

C  Send data to the worker tasks
C  First find #columns from B to send to each worker task
      do 50 i=1, NPROC
      if (i .le. extra) then
        cols = avecol + 1
      else
        cols = avecol
      endif

C  Next call initializes send buffer and specifies to do XDR data format
C  conversion only in heterogenous environment
```

========> ROTINA PVM (VARIAVEL: rcode)

```
C  Next four calls pack values into the send buffer - rcode not checked
C     offset      = starting position in matrix
C     cols        = number of columns of B to send
C     a           = send all of A
C     b                   = send some columns from B beginning at offset
```

========> ROTINA PVM (VARIAVEIS: offset, rcode)
========> ROTINA PVM (VARIAVEIS: cols, rcode)
========> ROTINA PVM (VARIAVEIS: a, NRA*NCA, rcode)
========> ROTINA PVM (VARIAVEIS: b(1,offset), cols*NCA, rcode)

```
C  Send contents of send buffer to worker task
```

========> ROTINA PVM (VARIAVEIS: wtids(i), mtype, rcode)

```
      offset = offset + cols
 50   continue
C  Wait for results from all worker tasks.  After setting message type,
C  loop for NPROCs.  Receive following data from each worker:
```

```
C     offset              = starting position in matrix
C     cols                = number of columns to receive
C     c(1,offset)= columns of matrix C beginning at offset
      mtype = 2
      do 60 i=1, NPROC
```

========> ROTINA PVM (VARIAVEIS: mtype, rcode)
========> ROTINA PVM (VARIAVEIS: offset, rcode)
========> ROTINA PVM (VARIAVEIS: cols, rcode)
========> ROTINA PVM (VARIAVEIS: c(1, offset), cols*NRA, rcode)

```
 60   continue

C  Print results
      do 90 i=1, NRA
        do 80 j = 1, NCB
        write(*,70)c(i,j)
 70     format(2x,f8.2,$)
 80     continue
      print *, ' '
 90   continue

C  task now exits from PVM
```

========> ROTINA PVM (VARIAVEL: rcode)

```
      end
```

PROGRAMA ESCRAVO (FORTRAN)

```
C**************************************************************************
C                     PVM TEMPLATE CODES
C FILE: matriz.w.f
C DESCRIPTION: See matriz.m.f
C PVM VERSION: 3.x
C LAST REVISED: 5/27/93 bbarney
C**************************************************************************
C Explanation of constants and variables used in this program:
C   NRA              = number of rows in matrix A
C   NCA              = number of columns in matrix A
C   NCB              = number of columns in matrix B
C   wtid             = PVM task id of this worker program
C   mtid             = PVM task id of master task
C   mtype            = PVM message type
C   cols             = columns of matrix B sent to each worker
C   offset                 = starting position within the matrix
C   rcode, i, j, k     = misc.
C   a                = matrix A to be multiplied
C   b                = matrix B to be multiplied
C   c                = result matrix C
C ---------------------------------------------------------------------
      program matriz_worker
C     PVM Version 3.0 include file
=======> PVM Include
      parameter(NRA = 62)
      parameter(NCA = 15)
      parameter(NCB = 7)
      integer       wtid, mtid, mtype, cols, rcode, offset, i, j, k
      real*8    a(NRA,NCA), b(NCA,NCB), c(NRA,NCB)

C  Enroll worker task

=======> ROTINA PVM ( VARIAVEL: wtid )

C  Receive message from master.  First set message type and determine the
C  tid of the parent process.  Then receive following data from master:
C     offset       = starting position in matrix
C     cols      = number of columns of B to receive
C     a            = receive all of matrix A
C     b        = receive some columns from matrix B
```

```
      mtype = 1

=======> ROTINA PVM ( VARIAVEL: mtid )
=======> ROTINA PVM ( VARIAVEIS: mtid, mtype, rcode )
=======> ROTINA PVM ( VARIAVEIS: offset, rcode )
=======> ROTINA PVM ( VARIAVEIS: cols, rcode )
=======> ROTINA PVM ( VARIAVEIS: a, NRA*NCA, rcode )
=======> ROTINA PVM ( VARIAVEIS: b, cols*NCA, rcode )

      write(*,10) wtid, cols
 10   format('worker task id = ',i8,'  received ',i3,' cols from B')

C  Do matrix multiply
       do 20 k=1, cols
         do 20 i=1, NRA
         c(i,k) = 0.0
           do 20 j=1, NCA
           c(i,k) = c(i,k) + a(i,j) * b(j,k)
 20        continue

C  Set up send message to master. First set message type and
C  initialize send buffer.  Then send following data elements to master:
C     offset       = starting position in result matrix C
C     cols         = number of columns to send
C     c            = our part of result matrix C
      mtype = 2

=======> ROTINA PVM ( VARIAVEL: rcode )
=======> ROTINA PVM ( VARIAVEIS: offset, rcode )
=======> ROTINA PVM ( VARIAVEIS: cols, rcode )
=======> ROTINA PVM ( VARIAVEIS: c, cols*NRA, rcode )

C  Send to master
=======> ROTINA PVM ( VARIAVEIS: mtid, mtype, rcode )

C  Exit PVM
=======> ROTINA PVM ( VARIAVEL: rcode )
      end
```

5º **LABORATÓRIO**

1 - Caminhe para o diretório de trabalho do PVM

% **cd ~/pvm3/bin/LINUX**

2 - Copie os arquivos exemplos do 5º laboratório.

% **cp ~curso/curso/pvm/lab04/* .**

4 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

% **make -f make.matriz.f**
ou
% **make -f make.matriz.c**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle <**enter**> duas vezes

6 - Inicialize o XPVM

% **xpvm**

7 - Execute o programa **matriz.m** e tente depurar o erro que ocorre no programa.

OBS: Será mais fácil de observar o erro na janela de "trace file" do XPVM

8 - Corrija o erro e execute novamente o programa.

6º LABORATÓRIO

1 - Caminhe para o diretório de trabalho do PVM

% **cd ~/pvm3/bin/LINUX**

2 - Copie os arquivos exemplos do 6º laboratório.

% **cp ~curso/curso/pvm/lab05/* .**

3 - Edite e altere o programa **mat.m.f** ou **mat.m.c.** Elimine a primeira chamada da rotina **"spawn"**, onde é informado o nome da primeira máquina a ser utilizada para execução.

4 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

% **make -f make.mat.f**
ou
% **make -f make.mat.c**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle **<enter>** duas vezes

6 - Execute o programa várias vezes (pelo menos tres) com o comando **time,** para se ter uma média do tempo de execução:

% **time mat.m**

7 - Edite e altere o programa **mat.m.f** ou **mat.m.c** com o intuito de melhorar a performance da

execução. Efetue os seguintes passos:

7.1 - Elimine o "loop" onde se encontra o comando **"spawn"**, adequando a rotina.

Compile e execute várias vezes com o comando **time**, para se ter uma média do tempo da execução.

7.2 - Modifique a comunição entre os processos, de modo a ser direta (task-task). Utilize a rotina **"setopt"**

Compile e execute várias vezes com o comando **time**, para se ter uma média do tempo da execução.

7.3 - Modifique a codificação dos dados, de modo que os dados sejam transferidos direto da memória. Altere a rotina **"initsend"**.

Compile e execute várias vezes com o comando **time**, para se ter uma média do tempo da execução.

8 - Compare todas as médias de execução do programa.

7º LABORATÓRIO

1 - Melhore a performance do programa do 1º laboratório (o programa **"hello"**). Edite o programa mestre, retire os **"loops"** da rotina **"spawn"** e da rotina **"send"**.

2 - Melhore a performance do programa do 2º laboratório (o programa **"pvm.ex1"**). Edite o programa mestre e o programa escravo, altere o modo de codificação dos dados quando da inicialização do **"buffer"**.

8º LABORATÓRIO

1 - Caminhe para o diretório de trabalho do PVM

% **cd ~/pvm3/bin/LINUX**

2 - Copie os arquivos exemplos do 8º laboratório.

% **cp ~curso/curso/pvm/lab06/* .**

3 - Analise o programa exemplo modelo SPMD, com relação as chamadas das rotinas PVM e da lógica utilizada.

4 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

% **make -f make.karp2.f**

ou

% **make -f make.karp2.c**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle <**enter**> duas vezes

6 - Inicialize a console do PVM.

```
% pvm
pvm>
```

7 - Verifique se todos os "daemons" foram inicializados. Se ok, saia da console PVM sem cancelar os "daemons" com o comando "quit"

```
pvm> conf
  . . .
pvm> quit
```

8 - Execute o programa.

```
% pi
```

9º LABORATÓRIO

1 - Caminhe para o diretório de trabalho do PVM

% **cd ~/pvm3/bin/LINUX**

2 - Copie os arquivos exemplos do 9º laboratório.

% **cp ~curso/curso/pvm/lab07/* .**

3 - Analise o programa exemplo modelo SPMD, com relação as chamadas das rotinas PVM e da lógica utilizada.

4 - Antualize o arquivo makefile com relação as opções de compilação (Compilador, Include e bibliotecas). Execute o arquivo makefile:

% **make -f make.groupIO.f**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**
OBS: tecle **<enter>** duas vezes

6 - Execute o programa utilizando o comando "time". Anote os tempos de processamento observados.

% **time groupIO**

7 - Adicione ou retire uma ou mais máquinas da configuração atual. Execute novamente o programa com o auxílio do comando "time". Compare os tempos de processamento com ao do item anterior.

% **pvm**
pvm> add <máquina> ou delete <máquina>
pvm> quit
% **time groupIO**

10º LABORATÓRIO

1 - Caminhe para o diretório de trabalho do PVM na arquitetura RISC

% **cd ~/pvm3/bin/LINUX**

2 - Copie os arquivos exemplos do 10º laboratório.

% **cp ~curso/curso/pvm/lab09/* .**

3 - Edite o programa **karp2.f** ou **karp2.c** e elimine os comandos de leitura de dados ,via console e elimine o loop do programa, **go to 5**. Fixe um valor para **nprocs** e **N**:

nprocs=2
N=100000

4 - Execute o arquivo makefile para criar os executáveis do PVM:

% **make -f make.karp2_PVM.f**

5 - Inicialize os "daemons" pvmd em "background" nas máquinas que irão participar do processamento com o PVM:

% **pvmd3 <arquivo de máquinas> &**

OBS: tecle **<enter>** duas vezes

6 - Execute o programa tres vezes para se ter uma idéia do tempo de execução:.

% **time pi**