PVM

UN OUTIL DE DISTRIBUTION DE CALCULS SUR UNE MACHINE VIRTUELLE

Sébastien Veigneau

Institut Gaspard Monge | **Université de Marne-la-Vallée**

Sommaire

INTRODUCTION

Historique

- projet PVM—version 1.0 interne—
  - Oak Ridge National Laboratory (1989).
- projet PVM—version 2.0—
  - University of Tennessee (1991).
- projet PVM—version 3.0—re-écrite
  - University of Tennessee (1993) ;
  - version 3.3.10.

Current PVM News:

- 1996 PVM Users Group Meeting – Santa Fe, NM. February 25–27. Mark your calendars! There will be a Tutorial on Sunday 25th followed by 2 days of **talks**. Abstracts are due **January 22, 1996**.
- **New patch: PVM 3.3.10 available**
  PVM 3.3.10 is now on NETLIB **(click below under source code)**
  patch 10 contains two new ports and several enhancements and fixes the shmem and PGON bugs reported in 3.3.9.
- Listing of **new features found in patch 10** and past PVM versions.
- **Recent PVM highlights**
- **Peek into the future** and see what we are working on for future releases.
- **WPVM 1.0** – Alexandre Alves of University of Coimbra – PORTUGAL has created a PVM version for Microsoft Windows.
- **Some new items added to this page** performance graphs of PVM on SP–2, PGON, etc (look under project members), added CODINE to list of Schedulers that work with PVM.

Current articles from PVM news group comp.parallel.pvm

`http://www.epm.ornl.gov/pvm/pvm_home.html`

Survol de PVM (1)

- cadre de développement de programmes // ;
- collection de machines hétérogènes vues comme une seule machine virtuelle // ;
- machines mono et multi-processeurs :
  - prise en compte des routines spécifiques ;
- prise en compte :
  - du routage des messages ;
  - conversion de données ;
  - gestion des tâches ;

  au sein d'un réseau de machines à architectures souvent incompatibles ;
- application ≡ collection de tâches (coopèrent) ;
- les tâches accèdent aux ressources PVM via la librairie de routines d'interface :
  - création, terminaison de tâches ;
  - communication et synchronisation entre les tâches.

Survol de PVM (2)

- 2 parties :
  - démon `pvmd3` (`pvmd`) :
    - présent sur toutes les machines ;
    - installation personnelle ;
    - plusieurs utilisateurs à la fois sur des machines virtuelles différentes.
  - librairie de routines PVM d'interface :
    - gestion de la coopération entre tâches ;
    - échange de messages ;
    - création de tâches ;
    - coordination de tâches ;
    - gestion de la machine virtuelle.
- C/C++ (`libpvm3.a`) ;
- FORTRAN (`libfpvm3.a`).

Sommaire

LA MACHINE VIRTUELLE

Définition de la machine virtuelle

- définition personnelle ou globale ;
- fichier de description ;
- une ligne par processeur virtuel ;

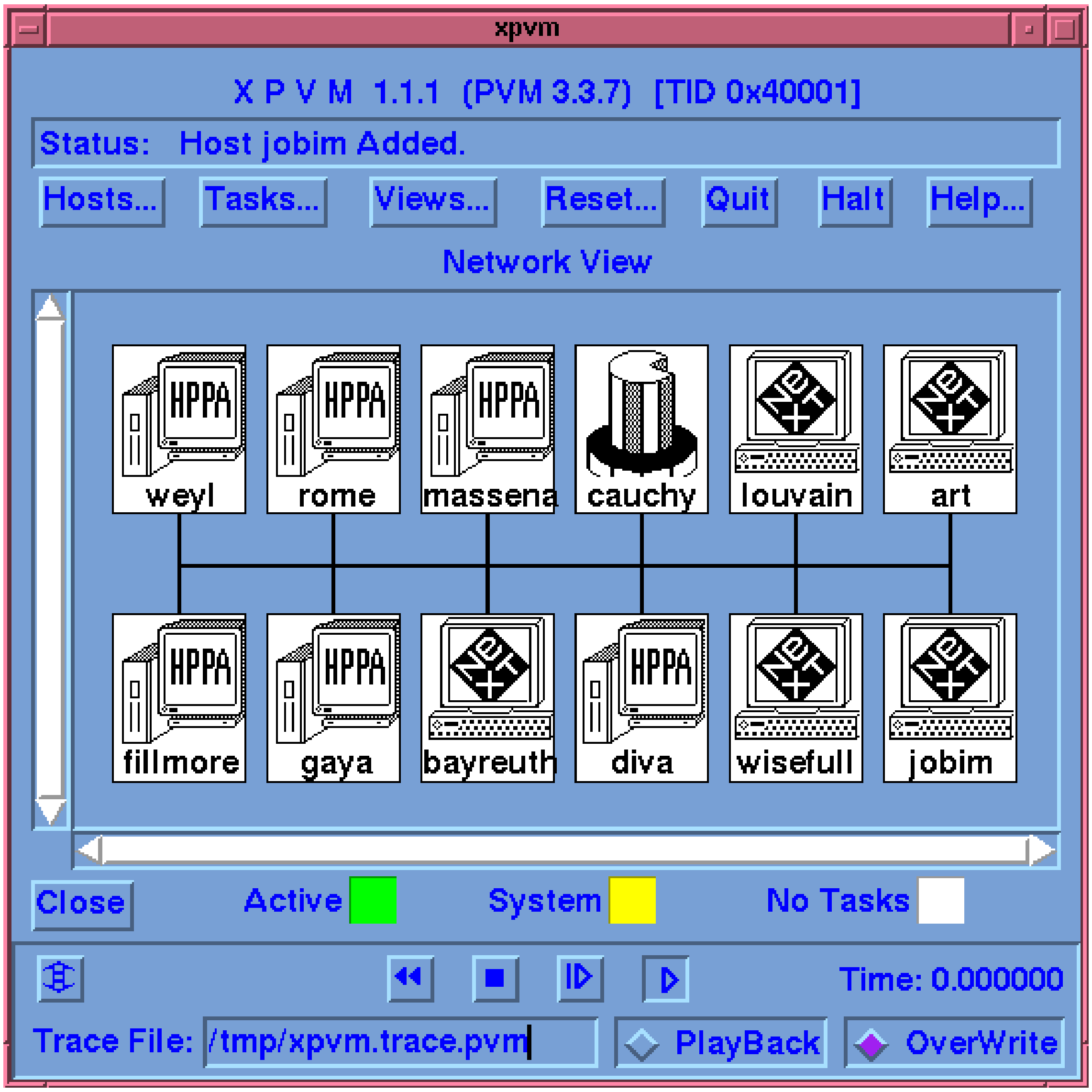
```
#####################
# Architecture HPPA #
#####################
&weyl     ep=/users/guest/pvm/usr/PVM/bin lo=pvm\
          dx=/usr/local/lib/pvm3/lib/pvmd
&fillmore ep=/users/guest/pvm/usr/PVM/bin lo=pvm\
          dx=/usr/local/pvm3/lib/pvmd
&rome     ep=/users/guest/pvm/usr/PVM/bin lo=pvm\
          dx=/usr/local/pvm3/lib/pvmd
&gaya     ep=/users/guest/pvm/usr/PVM/bin lo=pvm\
          dx=/usr/local/pvm3/lib/pvmd
&massena  ep=/users/guest/pvm/usr/PVM/bin lo=pvm\
          dx=/usr/local/pvm3/lib/pvmd
#####################
# Architecture NEXT #
#####################
&bayreuth ep=/clients/pvm/usr/PVM/bin lo=pvm\
          dx=/clients/pvm/usr/pvm3/lib/pvmd
#####################
# Architecture CRAY #
#####################
&cauchy ep=/u/veigneau/usr/PVM/bin lo=veigneau\
        dx=/u/veigneau/usr/pvm3/lib/pvmd
#####################
# Architecture SGI5 #
#####################
&free0 ep=/usr/people/pvm/usr/PVM/bin lo=pvm\
       dx=/usr/people/pvm/usr/pvm3/lib/pvmd
```

```
&free1 ep=/usr/people/pvm/usr/PVM/bin lo=pvm\
       dx=/usr/people/pvm/usr/pvm3/lib/pvmd
&free2 ep=/usr/people/pvm/usr/PVM/bin lo=pvm\
       dx=/usr/people/pvm/usr/pvm3/lib/pvmd
&free3 ep=/usr/people/pvm/usr/PVM/bin lo=pvm\
       dx=/usr/people/pvm/usr/pvm3/lib/pvmd
&free4 ep=/usr/people/pvm/usr/PVM/bin lo=pvm\
       dx=/usr/people/pvm/usr/pvm3/lib/pvmd
&free5 ep=/usr/people/pvm/usr/PVM/bin lo=pvm\
       dx=/usr/people/pvm/usr/pvm3/lib/pvmd
#
&onyx  ep=/export1/people2/thesard/veigneau/onyx/usr/PVM/bin\
lo=veigneau dx=/export1/people2/thesard/veigneau/onyx/usr/pvm3/lib/pvmd
&indigo1 ep=/usr/people2/thesard/veigneau/indigo/usr/PVM/bin\
  lo=veigneau dx=/usr/people2/thesard/veigneau/indigo/usr/pvm3/lib/pvmd
&indigo2 ep=/usr/people2/thesard/veigneau/indigo/usr/PVM/bin\
  lo=veigneau dx=/usr/people2/thesard/veigneau/indigo/usr/pvm3/lib/pvmd
&indy  ep=/usr/people2/thesard/veigneau/indy/usr/PVM/bin\
    lo=veigneau dx=/usr/people2/thesard/veigneau/indy/usr/pvm3/lib/pvmd
#####################
# Architecture HPPA #
#####################
&monge  ep=/disc1/guests/pvm/usr/PVM/bin\
        lo=pvm dx=/usr/local/pvm3/lib/pvmd
&merlin ep=/usr/users/guest/pvm/usr/PVM/bin\
        lo=pvm dx=/usr/local/pvm3/lib/pvmd
&diva   ep=/usr/users/prov/pvm/usr/PVM/bin\
        lo=pvm dx=/usr/local/pvm3/lib/pvmd
#####################
# Architecture NEXT #
#####################
&louvain   ep=/pvm/usr/PVM/bin lo=pvm dx=/pvm/usr/pvm3/lib/pvmd
&wisefull  ep=/Users/pvm/next/usr/PVM/bin lo=pvm\
           dx=/Users/pvm/next/usr/pvm3/lib/pvmd
&art       ep=/Users/pvm/next/usr/PVM/bin lo=pvm\
           dx=/Users/pvm/next/usr/pvm3/lib/pvmd
###################
# NEXTSTEP SUR HP #
###################
&jobim     ep=/Users/pvm/jobim/usr/PVM/bin lo=pvm\
           dx=/Users/pvm/jobim/usr/pvm3/lib/pvmd
```

Contrôle de la machine virtuelle

```
$ pvm hostfile
pvm> help
HELP   - Print helpful information about a command
Syntax:  help [ command ]
Commands are:

.......

  ADD    - Add hosts to virtual machine
  ALIAS  - Define/list command aliases
  CONF   - List virtual machine configuration
  DELETE - Delete hosts from virtual machine
  HALT   - Stop pvmds
  QUIT   - Exit console

.......

pvm> conf
12 hosts, 2 data formats
                    HOST     DTID     ARCH   SPEED
                    weyl    40000     HPPA    1000
                fillmore    80000     HPPA    1000
                    rome    c0000     HPPA    1000
                    gaya   100000     HPPA    1000
                 massena   140000     HPPA    1000
                bayreuth   180000     NEXT    1000
                  cauchy   1c0000     CRAY    1000
                    diva   380000     HPPA    1000
                 louvain   3c0000     NEXT    1000
                wisefull   400000     NEXT    1000
                     art   440000     NEXT    1000
                   jobim   480000     NEXT    1000
```

Contrôle de la machine virtuelle (xpvm)



Sommaire

UTILISATION DE PVM

Le point sur...

...le modèle PVM :

- chaque tâche peut envoyer un message à toutes les autres tâches ;
- pas de limite sur la taille ou le nombre des messages ;
- modèle de communication :
  - émission asynchrone bloquante :
    - bloquante : termine dès que le *buffer* d'envoi est libre pour une autre utilisation ;
    - asynchrone : ne dépend pas d'un éventuel récepteur ;
  - réception asynchrone bloquante ;
  - réception non-bloquante ;
  - supporte le *multicast* ;
  - supporte le *broadcast* vers un groupe de tâches.

Le point sur...

...les primitives d'échange de messages :

- modèle d'échange de messages, explicite ;
- orientées opérations hétérogènes ;
- constructions fortement typées pour
  - la *bufferisation* des données ;
  - la transmission ;
- primitives de :
  - réception,
  - émission ;

  de structures de données ;
- primitives de plus haut niveau :
  - *broadcast* ;
  - synchronisation par barrière ;
- allocation dynamique des *buffers* $\leadsto$ taille maximale de message limitée par la quantité de mémoire sur une machine donnée.

La console pvm

```
$ pvm hostfile
pvm> help
HELP   - Print helpful information about a command
Syntax:  help [ command ]
Commands are:
  ADD    - Add hosts to virtual machine
  ALIAS  - Define/list command aliases
  CONF   - List virtual machine configuration
  DELETE - Delete hosts from virtual machine
  ECHO   - Echo arguments
  HALT   - Stop pvmds
  HELP   - Print helpful information about a command
  ID     - Print console task id
  JOBS   - Display list of running jobs
  KILL   - Terminate tasks
  MSTAT  - Show status of hosts
  PS     - List tasks
  PSTAT  - Show status of tasks
  QUIT   - Exit console
  RESET  - Kill all tasks
  SETENV - Display or set environment variables
  SIG    - Send signal to task
  SPAWN  - Spawn task
  TRACE  - Set/display trace event mask
  UNALIAS - Undefine command alias
  VERSION - Show libpvm version
pvm> add cauchy
1 successful
                    HOST     DTID
                  cauchy   140000
pvm> conf
2 hosts, 2 data formats
                    HOST     DTID     ARCH   SPEED
                    weyl    40000     HPPA    1000
                  cauchy   140000     CRAY    1000
```

Contrôle de processus

- `int tid = pvm_mytid( void );`
  - retourne le `TID` d'un processus ;
  - processus ⇝ tâche PVM.
- `int info = pvm_exit( void );`
  - informe le démon local que cette tâche quitte PVM ;
  - ne tue pas le processus (revient un processus "normal").
- `int numt = pvm_spawn( char *task, char **argv, int flag, char *where, int ntask, int *tids );`
  - création de `ntask` copies de `task` ;
  - `argv` contient les arguments (NULL à la fin) ;
  - `flag` spécifie des options ;
  - `numt` = nombre de tâches créées.
- `int info = pvm_kill( int tid );`

Informations...

- `int tid = pvm_parent( void )`;
  - retourne le `TID` du processus qui a créé la tâche par un `pvm_spawn`;
  - `PvmNoParent` si non créée par `pvm_spawn`.
- `int dtid = pvm_tidtohost( int tid )`;
  - retourne le `TID` du démon tournant sur le même hôte que la tâche;
- `int info = pvm_config( int *nhost, int *narch, struct pvmhostinfo **hostp )`;
  - `nhost` = nombre d'hôtes;
  - `narch` = nombre de format de données différents;
  - `hostp` = tableau de "structures" :
    - `TID` du démon;
    - nom de l'hôte;
    - nom de l'architecture;
    - vitesse CPU relative.

Echange de messages (1)

Trois étapes :

1. initialiser un *buffer* d'envoi ;
2. y stocker le message à envoyer (`pack`) ;
3. envoyer le message.

Initialisation

- `int tid = pvm_initsend(int encoding)` ;
  - remise à zéro du *buffer* ;
  - `encoding` =
    - `PvmDataDefault` ;
    - `PvmDataRaw` ;
    - `PvmDataInPlace`.
- gestion de plusieurs *buffers*.

Echange de messages (2)

Stockage du message :

- tableau ⇝ *buffer* actif d'émission ;
- `pvm_pkbyte(char *cp, int nitem, int stride)` ;
- `pvm_pkint(int *np, int nitem, int stride)` ;
- `pvm_pklong(long *np, int nitem, int stride)` ;
- `pvm_pkstr(char *cp)` ;
  - stockage d'une chaîne de caractères terminée par le caractère NULL.

(Extraction du message) :

- *buffer* actif de réception ⇝ tableau ;
- `pvm_upkbyte(char *cp, int nitem, int stride)` ;
- `pvm_upkint(int *np, int nitem, int stride)` ;
- `pvm_upklong(long *np, int nitem, int stride)` ;
- `pvm_upkstr(char *cp)` ;

Echange de messages (3)

Envoi du message :

- `int info = pvm_send( int tid, int msgtag );`
- `int info = pvm_mcast( int *tids, int ntask, int msgtag );`
- `int info = pvm_psend( int tid, int msgtag , void *vp, int cnt, int type );`

◦ `type` =

PVM_STR	PVM_FLOAT
PVM_BYTE	PVM_CPLX
PVM_SHORT	PVM_DOUBLE
PVM_INT	PVM_DCPLX
PVM_LONG	PVM_UINT
PVM_USHORT	PVM_ULONG

Echange de messages (4)

Réception du message :

- `int bufid = pvm_recv( int tid, int msgtag )` ;
  - réception bloquante ;
  - `msgtag` = -1 pour signifier n'importe quel `msgtag` ;
  - `TID` = -1 pour signifier n'importe quel identificateur de tâche ;
  - efface le précédent *buffer* de réception ;
  - crée un nouveau *buffer* de réception.
- `int bufid = pvm_nrecv( int tid, int msgtag )` ;
  - réception non-bloquante ;
  - message pas arrivé $\leadsto$ `bufid` = 0 ;
- `int bufid = pvm_probe( int tid, int msgtag )` ;
  - non-bloquant ;
  - message pas arrivé $\leadsto$ `bufid` = 0 ;
  - la réception n'a pas lieu : c'est juste un test.

Sommaire

EXEMPLES

Programme `hello.c`

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

main()
{
     int cc, tid; char buf[100];

     printf("i'm t%x\n", pvm_mytid());

     cc = pvm_spawn("hello-other",
                    (char**)0, 0, "", 1, &tid);

     if (cc == 1) {
        cc = pvm_recv(-1, -1);
        pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
        pvm_upkstr(buf);
        printf("from t%x: %s\n", tid, buf);

     } else printf("can't start hello-other\n");

     pvm_exit();
     exit(0);
}
```

Programme hello-other.c

```
#include "pvm3.h"

main()
{
     int ptid;
     char buf[100];

     ptid = pvm_parent();

     strcpy(buf, "hello, world from ");
     gethostname(buf + strlen(buf), 64);

     pvm_initsend(PvmDataDefault);
     pvm_pkstr(buf);
     pvm_send(ptid, 1);

     pvm_exit();
     exit(0);
}
```

Un exemple d'un programme maître

```
#ifdef _ANSI_
  void main(int argc, char * argv[])
#else /* _ANSI_ */
  void main(argc, argv) int argc; char * argv[];
#endif /* _ANSI_ */
{
  .......

  int mytid;                                                     /* my tid */
  int tasktid;                                                 /* task tid */
  int max_nbtasks=0;                             /* maximal number of tasks */
  int nbtasks=0;                             /* real number of active tasks */
  int i,j;                                         /* variables for loops */
  char err_msg[256];                                      /* error message */
  int parameters_size=0;          /* number of elements in parameters */
  char * parameters[2*PARTITION_MAX_NBPARTS+6];
  ExecGr exec_graph;                                    /* execution graph */
  int spawn_result=0;                               /* the result of a spawn */
  int buffer_id;              /* the id of the received message buffer */
  int route;                                      /* answer of pvm_setopt */
  int pvm_return_value;                        /* pvm_ call returned value */
  int msg_tag;                                              /* message tag */
  long nbsucess=0;                /* number of branchs that succeeded */

  init_from_argv(argc, argv, &max_nbtasks);

  read_data(INPUT_DATA_FILE, &mu, &lambda, &k);

  .......

  /* DECLARING MYSELF AS A PVM TASK */
  mytid=pvm_mytid();
  if (mytid==PvmSysErr)
    Error("Pvmd not responding", EXIT_PVMD_NOT_RESPONDING);
```

```
/* initialization of parameters */
for(i=0;i<2*PARTITION_MAX_NBPARTS+6-1;i++)
{  parameters[i]=(char *)malloc(3*sizeof(char));
   assert_memory(parameters[i]!=NULL);
}
/* to send parameters, we need to have the last one set to NULL */
parameters[2*PARTITION_MAX_NBPARTS+6-1]=NULL;

sprintf(parameters[0], "%d", mytid);
sprintf(parameters[1], "%d", k);
sprintf(parameters[2], "%d", lambda_conj_staircase.size);
for(i=0;i<lambda_conj_staircase.size;i++)
  sprintf(parameters[3+i], "%d", lambda_conj_staircase.data[i]);
sprintf(parameters[3+i], "%d", mu.size-1);
sprintf(parameters[4+i], "%d", mu.size);
for(j=0;j<mu.size;j++)
  sprintf(parameters[5+i+j], "%d", mu.data[j]);
parameters_size=5+i+j;
parameters[parameters_size]=NULL;

/* creation of the execution graph */
InitExecGraph(lambda_conj_staircase, mu, &exec_graph);

.......

/* creation of the first task */
spawn_result=pvm_spawn(TASK_NAME, parameters, PvmTaskDefault,
                       "", 1, &tasktid);
switch(spawn_result)
{ case 1  : nbtasks++;
            break;

  case 0  : Error("Can't create first task", EXIT_CREATE_FIRST_TASK);

  default : Error("pvm_spawn for first task", spawn_result);
}

/* requesting direct route between tasks */
route=pvm_setopt(PvmRoute, PvmRouteDirect);
```

```
  /* while there are sub-tasks */
  while(nbtasks)
  {
    /* waiting for a message from any task */
    buffer_id=pvm_recv(-1, -1);

    /* extracting message tag */
    if ((pvm_return_value=pvm_upkint(&msg_tag, 1, 1))<0)
      Error("pvm_upkint", pvm_return_value);

    /* extracting task tid */
    if ((pvm_return_value=pvm_upkint(&tasktid, 1, 1))<0)
      Error("pvm_upkint", pvm_return_value);

    switch(msg_tag)
    {
      case TAG_DYING :

          nbtasks--;

#ifdef _DEBUG_
          fprintf(stderr, "[%s]", __FILE__);
          fprintf(stderr,
                " : pvm_rec : TAG_DYING=%d, TASK=%d : nbtasks=%d\n",
                      msg_tag, tasktid, nbtasks);
#endif
          break;

     case TAG_DYING_SUCCESS :

#ifdef _DEBUG_
          fprintf(stderr, "[%s]", __FILE__);
          fprintf(stderr,
          " : pvm_rec : TAG_DYING_SUCCESS=%d, TASK=%d : nbtasks=%d\n",
              msg_tag, tasktid, nbtasks);
#endif

          nbsucess++;
          break;
```

Gestion de la création des tâches

Dans cette version de notre programme, chaque tâche demande au programme maître l'autorisation de créer un certain nombre de nouvelles tâches. Le programme maître

- analyse les demandes ;
- met à jour ses tables de gestion des tâches ;
- répond aux demandes.

```
      case TAG_CREATING :

#ifdef _ONE_CREATING_MSG_
          /* we receive one message for all cases */
          pvm_return_value=pvm_upkbyte(&(choose.nb_entries), 1, 1);
          pvm_return_value=pvm_upkbyte(choose.data,
                    choose.nb_entries*lambda_conj_staircase.size, 1);

          AnswerChooseRoot(&choose);
          nbtasks+=choose.nb_positive_answers;

          buffer_id=pvm_initsend(PvmDataDefault);
          msg_tag=TAG_CREATING_ANSWER;
          pvm_return_value=pvm_pkint(&msg_tag, 1, 1);
          pvm_return_value=
                     pvm_pkbyte(&(choose.nb_positive_answers), 1, 1);
          pvm_return_value=pvm_pkbyte(choose.answers,
                             choose.nb_positive_answers, 1);

          pvm_send(tasktid, TAG_CREATING_ANSWER);
#endif /* _ONE_CREATING_MSG_ */

          break;
```

Terminaison/gestion des erreurs

- gestion des problèmes réseaux ;
- gestion des problèmes systèmes ;
- `pvm_exit()` et `exit(EXIT_SUCCESS)`.

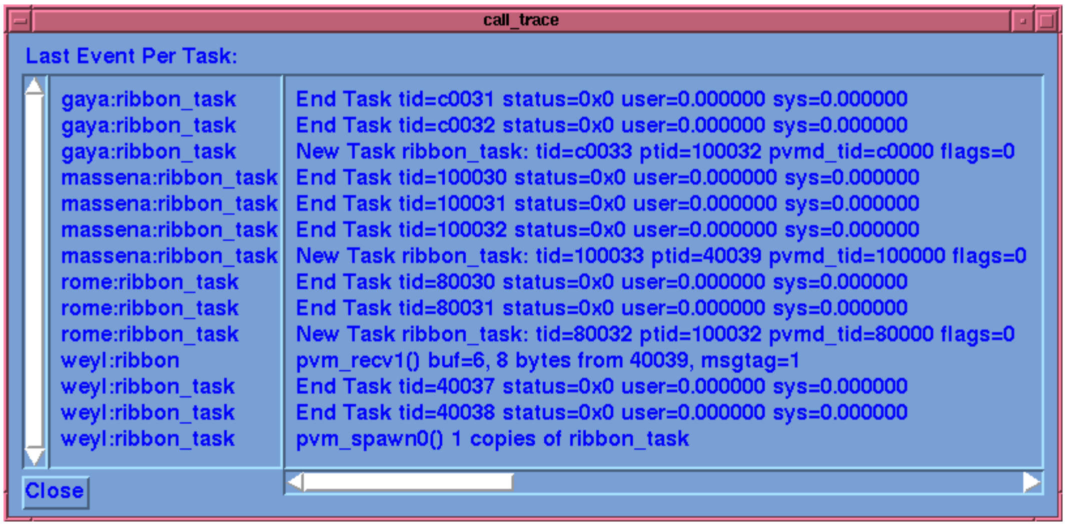
```
  .......

      case TAG_PVMD_NOT_RESPONDING :
          Error("one task does not succeed in contacting its parent",
               TAG_PVMD_NOT_RESPONDING);
          break;
    }
  }

  /* no more sub-task : printing the result */
  fprintf(stderr, "that's all folks\n");
  fprintf(stderr, "Number of solutions : %ld\n", nbsucess);
  pvm_exit();
  exit(EXIT_SUCCESS);
}

#ifdef _ANSI_
  void Error(char * string, int exit_status)
#else /* _ANSI_ */
  void Error(string, exit_status) char * string; int exit_status;
#endif /* _ANSI_ */
{ char err_msg[259];
  strcpy(err_msg, "** "); strcat(err_msg, string);
  fprintf(stderr,err_msg);
  pvm_exit();
  exit(exit_status);
}
```

Sommaire

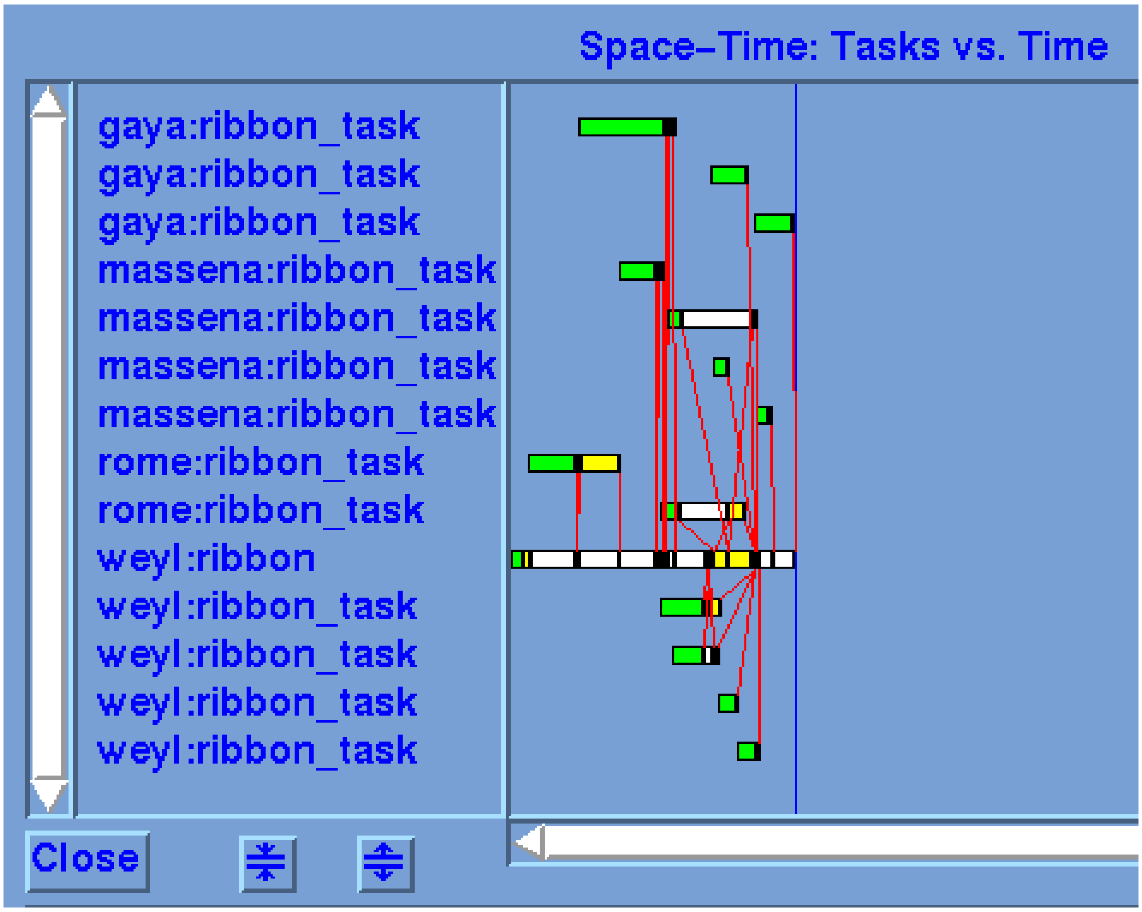
AUTOUR DE PVM

XPVM (1)

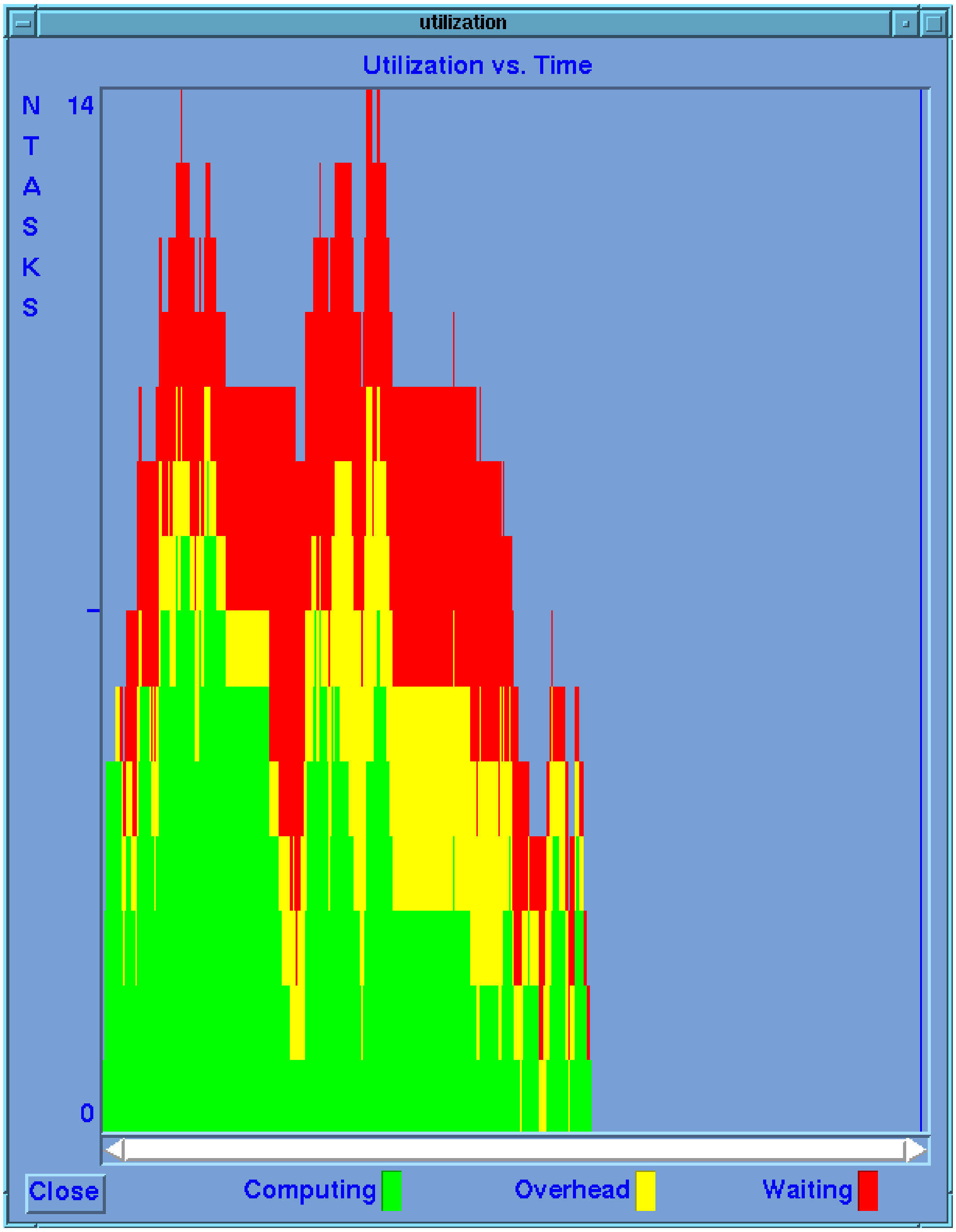
- console graphique de contrôle de PVM ;
- plusieurs vues de l'exécution ;
  - interaction entre les tâches ;
  - aide au déboggage ;
  - détection d'un processeur virtuel défectueux ;
- enregistrement d'une exécution :
  - possibilité d'interruption de l'exécution ;
  - visualisation de l'exécution post-mortem.



XPVM (2)


Space–Time: Tasks vs. Time
gaya:ribbon_task
gaya:ribbon_task
gaya:ribbon_task
massena:ribbon_task
massena:ribbon_task
massena:ribbon_task
massena:ribbon_task
rome:ribbon_task
rome:ribbon_task
weyl:ribbon
weyl:ribbon_task
weyl:ribbon_task
weyl:ribbon_task
weyl:ribbon_task
Close

XPVM (3)


utilization
Utilization vs. Time
NTASKS
14
0
Close
Computing
Overhead
Waiting

PVM sur Internet

1. Anonymous Ftp,

   ```
   ftp netlib2.cs.utk.edu
   cd pvm3/book
   get pvm-book.ps
   quit
   ```

2. Email, `netlib@ornl.gov`, and in the message type:
   `send pvm-book.ps from pvm3/book`

3. Http,
   `http://www.epm.ornl.gov/pvm/pvm_home.html`

4. Http,
   `http://www.netlib.org/pvm3/book/pvm-book.html`

5. Rcp,
   `anon@netlib2.cs.utk.edu:pvm3/book/pvm-book.ps`