Nathanaël COTTIN
http://www.ncottin.net

Découverte de PVM

Les TP sont réalisés sous *Solaris*.

1 Définition, architecture

PVM ("Parallel Virtual Machine") est un ensemble d'outils et de librairies permettant de connecter des systèmes hétérogènes afin de réaliser des traitements parallèles. Pour ce faire, PVM utilise des primitives (PVM supporte actuellement les langages C, C++ et Fortran) permettant le transfert et la récupération de données entre processus exécutés sur différentes machines.

PVM se compose de deux parties complémentaires :

- Un démon (au sens *daemon* des systèmes `UNIX`) baptisé `pvmd3` (ou plus simplement `pvmd`) exécuté sur toutes les machines enregistrées participant aux traitements
- Une librairie de routines (`pvm_*`) permettant d'indiquer les traitements à réaliser

Pour résumer, PVM offre une couche d'abstraction au programmeur en lui masquant le transport des messages (empaquetage des données, transmission, réception, désempaquetage). De plus, PVM se charge de répartir automatiquement les processus sur les machines enregistrées (à condition qu'elles puissent accéder aux programmes compilés par le biais du système de fichiers).

2 Configuration de l'environnement

2.1 Accès à PVM

Par défaut, les comptes ne reconnaissent par la commande `pvm` permettant d'exécuter l'environnement PVM. Pour ce faire, il suffit d'effectuer les manipulations suivantes :

1. Ouvrir (ou créer) un fichier `.bashrc_perso` sous votre racine (`~` ou `${HOME}`)
2. Ajouter les lignes suivantes :

```
export PVM_ROOT=/soft/pvm3
. $PVM_ROOT/lib/bashrc.stub
export MANPATH=${MANPATH}:${PVM_ROOT}/man
```

3. Changer les droits d'accès à ce fichier : `chmod u+x .bashrc_perso`
4. Activer les nouvelles variables avec la commande `source .bashrc_perso`
5. Vérifier que PVM est maintenant reconnu avec les commandes `man pvm` et `pvm`

Terminer l'exécution de l'environnement PVM avec la commande `halt`.

Ces étapes ne sont à effectuer qu'une seule fois, le fichier `.bashrc_perso` étant automatiquement chargé lors de la phase de login.

En option : préparer `ssh` comme indiqué sur le document "Utilisation de PVM au sein du GI" accessible depuis le WEB GI (effectuer une recherche du mot-clé PVM après identification afin d'obtenir la page HTML concernée). Seules les manipulations fournies dans la section relative à la

configuration de `ssh` doivent être suivies. Cette étape facultative permet d'éviter de systématiquement saisir un mot de passe lors des connexions sécurisées avec les machines distantes référencées dans le document cité précédemment.

2.2 Arborescence de travail

Il est recommandé de travailler au sein d'un dossier `pvm3` situé directement à la racine. Les TP seront par exemple identifiés par des sous-dossiers `<REPERTOIRE_TP>/` où `<REPERTOIRE_TP>` prendra par exemple les valeurs `TP1`, `TP2`, ...

Chacun de ces sous-dossiers contiendra les sources des programmes `master.c` et `slave.c`.

Il est possible de travailler au sein d'un autre dossier, cependant par défaut PVM recherche les fichiers compilés dans le dossier `${HOME}/pvm3/bin/${PVM_ARCH}`. Ce dossier est automatiquement créé par le script fourni dans ce document.

3 Premier programme

Ce programme consiste à envoyer une valeur entière incrémentale à `NB_PROC` processus chargés de la retourner. Le processus maître affiche alors les différentes valeurs retournées par les esclaves.

D'autres exemples sont fournis par défaut dans le dossier d'installation de PVM.

Du fait que chaque processus s'exécute dans son propre environnement, la sortie standard des esclaves ne correspond pas à la sortie standard du maître. Ceci explique l'incapacité des esclaves à afficher du texte avec `printf`...

3.1 Code source

3.1.1 Maître

Contenu du fichier `master.c` :

```
1  #include <stdio.h>
   #include "pvm3.h"
3  #define ERR       -1
   #define OK         0
5  #define NB_PROC    5
   #define SEND_TAG   0
7  #define RECV_TAG   1
   #define FROM_ALL  -1
9  #define SLAVE    "slave"

11 int main(int argc, char** argv) {
     int tids[NB_PROC];          /* Liste des esclaves */
13   int val;                    /* Valeur a envoyer */
     struct pvmhostinfo *host; /* Machine hebergeant le maitre */
15   int nhost, narch, nb;
```

```
    int bufId, size, tag, id;

    pvm_config(&nhost, &narch, &host);

    nb = pvm_spawn(SLAVE, (char **)0, PvmTaskDefault, host->hi_name,
      NB_PROC, tids);

    if (nb != NB_PROC) {
      pvm_exit();
      return ERR;
    }

    for (nb=0; nb<NB_PROC; nb++) {
      val = nb+1;
      pvm_initsend(PvmDataDefault);
      printf("Envoi de %d a %d.\n", val, tids[nb]);
      pvm_pkint(&val, 1, 1);
      pvm_send(tids[nb], SEND_TAG);
    }

    for (nb=0; nb<NB_PROC; nb++) {
      bufId = pvm_recv(FROM_ALL, RECV_TAG);
      if (bufId < 0) {
        pvm_exit();
        return ERR;
      }

      /*
       * Decodage des informations du message recu
       */
      if (pvm_bufinfo(bufId, &size, &tag, &id) < 0) {
        pvm_exit();
        return ERR;
      }

      pvm_upkint(&val, 1, 1);
      printf("J'ai recu %d de la part de %d.\n", val, id);
    }

    pvm_exit();
    return OK;
}
```

3.1.2 Esclave

Contenu du fichier `slave.c` :

```
1  #include <stdio.h>
   #include "pvm3.h"
3  #define ERR       -1
   #define OK         0
5  #define RECV_TAG  -1
   #define SEND_TAG   1
7
   int main(int argc, char **argv) {
9    int masterId; /* Identifiant du maitre */
     int selfId;   /* Identifiant du processus courant */
11   int val;      /* Valeur recue */

13   selfId = pvm_mytid();
     masterId = pvm_parent();
15
     /*
17    * Attente de reception d'un message quelconque emanant du maitre
      * et portant un tag donne
19    * Si le tag vaut -1, alors tous les tags sont acceptes
      */
21   if (pvm_recv(masterId, RECV_TAG) < 0) {
       pvm_exit();
23     return ERR;
     }
25
     /*
27    * Reception de l'entier envoye par le maitre
      */
29   pvm_upkint(&val, 1, 1);
     pvm_initsend(PvmDataDefault);
31   pvm_pkint(&val, 1, 1);
     if (pvm_send(masterId, SEND_TAG) < 0) {
33     pvm_exit();
       return ERR;
35   }

37   pvm_exit();
     return OK;
39 }
```

3.2 Compilation

Les fichiers de compilation (makefiles) proposés par défaut dans le dossier d'installation de PVM étant trop difficiles à modifier, voici une alternative élégante basée sur deux fichiers complémentaires `makefile` et `compile`[1].

Ces fichiers pourront être créés dans le dossier principal contenant les différents TP.

La mise en page (espaces, tabulations, retours à la ligne) doit être scrupuleusement respectée.

3.2.1 Création d'un makefile

Créer un fichier `makefile` générique comportant le code suivant :

```
CC = gcc
CPPFLAGS = -I${PVM_ROOT}/include
CFLAGS = -g
LDFLAGS = -L${PVM_ROOT}/lib/${PVM_ARCH}
LDLIBS = -lpvm3 -lnsl -lsocket

BINDIR = ${HOME}/pvm3/bin/${PVM_ARCH}

.PHONY: all clean

all: master slave
	mkdir -p ${BINDIR}
	mv -f master slave ${BINDIR}

clean:
	rm -f master slave ${BINDIR}/master ${BINDIR}/slave
```

3.2.2 Compilation des programmes

Créer un fichier `compile` disposant de droits d'exécution et comportant le code suivant :

```

#!/bin/bash

PROJECT=$1
MAKEFILE_DIR=$HOME/pvm3
MAKER=make
MAKER_OPT=-f

if [ ! -d $PROJECT ] ; then
  echo "Le repertoire $PROJECT n'existe pas."
  exit 1
```

[1] Remerciements à David Anderson pour ses sources

```
12 fi

14 (cd $PROJECT && $MAKER $MAKER_OPT $MAKEFILE_DIR/makefile)
```

La compilation d'un TP s'effectue en appelant la commande `./compile <CHEMIN_REPERTOIRE_TP>`. Le résultat de cette compilation (en cas de succès) est la création (ou mise à jour) des fichiers `master` et `slave` au sein du dossier `${HOME}/pvm3/bin/${PVM_ARCH}/` (qui sera également créé le cas échéant).

La compilation des programmes s'effectue à l'aide de l'outil `make` (ou `aimk`[a]) en lui indiquant l'étiquette à compiler. Par défaut l'étiquette `all` est appelée.
L'étiquette `clean` sert à effacer les fichiers compilés `master` et `slave`.

[a] architecture independent make

3.3 Exécution

L'interpréteur PVM (commande `pvm`) permet d'ajouter des machines participant à la distribution de l'algorithme avec la commande `add`. La liste de ces machines est indiquée dans le document relatif à la configuration de `ssh`.

Le programme maître est appelé avec la commande `spawn -> master` (en respectant les espaces situés de part et d'autre du symbole `->`).

4 Commentaires

L'envoi d'un message s'effectue en 3 étapes :

1. Initialisation du message : primitive `pvm_initsend`
2. Empaquetage (encodage) du contenu du message : `pvm_pk*`

   

   Le contenu du message à envoyer est indiqué sous forme de pointeur. La taille des données à transmettre doit donc être indiquée.

3. Envoi du message préparé (en indiquant le destinataire et une valeur de tag) : `pvm_send`

La réception d'un message s'effectue en 2 étapes (plus une étape optionnelle) :

1. Attente de réception d'un message : `pvm_recv`
2. Récupération des informations du message : `pvm_bufinfo`. Cette étape optionnelle permet de filtrer les messages en fonction de leur tag notamment
3. Désempaquetage (décodage) du contenu du message : `pvm_upk*`

5 Remarques diverses

Il est possible de quitter temporairement l'interpréteur PVM avec `quit` (et non `halt`), de sorte que le démon PVM demeure actif. Ceci permet d'exécuter le maître directement depuis une console et ainsi d'éviter de systématiquement saisir la commande `spawn -> master` depuis l'environnement PVM. Il ne faut cependant pas oublier de terminer correctement PVM avec `pvm` suivi de `halt` en fin de travail.