UNIVERSITE PIERRE ET MARIE CURIE6 Paris 6
Laboratoire d'informatique LiP6

Rapport périodique d'activité

Portage de PVM sur MPC

Karim Mana
Octobre 1998

Résumé :
Ce rapport décrit le portage de l'environnement de programmation parallèle PVM sur la machine MPC. La méthodologie adoptée, les choix faits et les difficultés rencontrées sont présentés.
Nous y présentons, tour à tour, les aspects intéressants du fonctionnement de la bibliothèque PVM et de la communication dans MPC. Nous terminons par une description générale du système et des communications au stade actuel de l'implémentation.

INTRODUCTION

La machine parallèle MPC, développée au laboratoire MASI, est constituée d'un ensemble de micro-ordinateurs (Pentium / Pentium-Pro) reliés par un réseau d'interconnexion rapide à la norme IEEE 1335 HSL à un gigat bits par seconde [1]. Ce réseau est pourvu de routeurs rapides , appelés R-Cube, conçus au MASI.

Au-dessus de cette architecture matérielle, un ensemble de couches logicielles ont été développées [2] [3]. Elles implémentent principalement, le modèle de communication par passage de message utilisant le protocole « d'écriture distante » également appelé « de dépôt direct mémoire ».

MPC promet donc de très bonnes performances via son réseau d'interconnexion et les fonctionnalités qu'elle offre. Cependant, ces dernières (fonctionnalités) restent à ce jour disponibles uniquement au niveau du noyau du système. Les applications utilisateurs ne peuvent donc en bénéficier facilement. D'où le besoin d'un environnement de programmation parallèle sur MPC.

Dans un premier temps, notre choix s'est porté sur le système PVM. Ce choix est motivé par la large utilisation de PVM, ce qui permet du même coup aux applications tournant au dessus de PVM de tourner sur MPC.

Nous commencerons ce rapport par une introduction à PVM. Son organisation, et sa gestion des communications. Nous regarderons particulièrement le cas d'une configuration PVM à démon unique. Ce qui nous rapprochera de l'implémentation sur MPC.
Nous discuterons ensuite des propriétés de MPC et de la manières de les exploiter (et de les adapter) pour PVM. Nous exposerons nos choix de portage et leur motivation, puis finirons par une étude des performances du système PVM-MPC.

I . PVM : Brève introduction

PVM (Parallèle Virtual Machine) est un système, constitué d'une librairie et d'un démon, de communication permettant aux applications d'utiliser un ensemble de machines, potentiellement hétérogènes et de puissances variables, interconnectées par un réseau de communication, comme un seul ordinateur, rendant ainsi, les communications indépendantes du système d'exploitation.

Développé à la fin des années 80 à l'Oak Ridge National Laboratory [4], PVM atteint aujourd'hui une large popularité et est considéré comme un standard pour l 'écriture de programmes parallèles.

1 . Communication sous PVM

PVM garantit que chaque tâche de la machine virtuelle peut émettre un nombre quelconque de messages vers tout autre tache de la machine. Ceci sans limitation de nombre, ni de taille des messages.

L'échange de message entre taches PVM est réalisé grâce à deux primitives de base : **pvm_send()** et **pvm_recv()**. La première est exécutée par l'émetteur et a pour effet de transmettre les données vers le récepteur. Ce dernier n'en prendra connaissance que s'il exécute un **pvm_recv** (ou scrute sa boite aux lettres). Entre ces deux moments, le système (PVM), se charge de préparer les données, de les transférer vers le site destinataire puis de les remettre à leur destinataire final.

Le transfert proprement dit, est donc indépendant de l'état d'exécution des deux taches qui dialoguent. Plusieurs messages peuvent être émis d'un émetteur **E** vers un destinataire **D** sans que ce dernier n'en consomme aucun, ni que l'émetteur en soit informé.

PVM offre un ensemble de variantes des deux primitives précédentes permettant chacune de réaliser un type de synchronisation.

- **Send Asynchrone** : ne retourne la main que lorsque les données ont été mises sur le réseau.
- **Receive Bloquant** : ne retourne la main au récepteur que lorsque le message attendu est disponible. Les données sont alors mises à la disponibilité du récepteur dans le tampon courant.
- **Receive non Bloquant** : ou encore **pvm_nrecv()**, retourne la main au récepteur avec les données si elles sont disponibles. Dans le cas contraire, retourne immédiatement une valeur indiquant qu'elle ne le sont pas.

D'autres sémantiques sont spécifiées [5] et peuvent être implémentées pour une utilisation plus optimisée des ressources offertes par l'architecture sous-jacente. Ces sémantiques standard ne sont pas toujours disponibles.
Dans le cas, d'une implémentation à démon unique, les appels sont tous bloquants sur les machines autres que celle du démon. Par exemple, il sera lancer à partir d'une console pvm grâce à la commande add pour « add host ». (ex: add mpc.lip6.fr).

2 . Fonctionnement général de la librairie PVM

La libraire PVM est constituée d'un ensemble de taches distribuées sur des machines hôtes qui constituent la machine virtuelle. Ces taches sont gérées par un ou plusieurs démons selon le type d'implémentation. Pour notre portage, nous choisissons une configuration avec un démon unique, comme pour le cas de la machine parallèle Paragon d'Intel , qui gère l'ensemble des taches de la machine virtuelle comme si elles étaient locales à son site. Ce type d'implémentation est retrouvé dans les machines massivement parallèle et qui disposent d'un réseau d'interconnexion rapide [8] .
Cette configuration peut faire partie d'une plus grande machine virtuelle où le démon local serait non primaire.
Aussi, dans cette configuration, nous distinguons la machine portant le démon des autres machines du réseau. Cette distinction nous permet d'identifier les étapes effectuées pour un échange de message pour chacun des deux cas.

Lorsqu'une tache, se trouvant sur la machine du démon, veut émettre un message, elle ne fait que l'insérer dan la file des messages à émettre le confiant ainsi au démon. Ce dernier se charge alors de le router, ou de le délivrer à sa tache destinataire. La tache émettrice est donc libre de retrouver son travail de fond.
Si une tache destinataire est sur un noeud avec démon, celui ci reçoit le message et l'insère dans la file des messages reçus pour la tache en question. Cette dernière peut alors en disposer lorsqu'elle effectuera un **pvm_recv**.

En l'absence d'un démon PVM sur le site hôte de la tache émettrice, celle ci fait appel à la librairie PVM pour effectuer le transfert. C'est donc le même processus Unix qui s'occupe de réaliser toutes les étapes du transfert jusqu'à la mise sur le réseau. La tache PVM ne peut donc reprendre la main qu'au retour des appels à la librairie.

De même, si une tache réceptrice se trouvant sur un site sans démon exécute un **pvm_recv()** (notamment bloquant), elle scrutera le buffer de dépôt du réseau (ou du système de communication) jusqu'à la réception du message. Si un mécanisme de sommeil n'est pas implémenté, l'attente active risque de coûter très cher en temps CPU. Par ailleurs, si l'attente est de courte durée, le coût d'un changement de contexte peut être excessif. Il est donc important d'estimer le temps moyen de cette attente avant d'opter pour le mode de son implémentation.

3 . Vison conceptuelle du système PVM

La bibliothèque PVM est constituée de deux principales parties (voir figure **A**). La première, indépendante du système d'exploitation, regroupe toutes les fonctions de gestion des tâches PVM, des groupes ainsi que les fonctions de communication de haut niveau telle que l'organisation des fragments de données à échanger ou encore à échanger ou encore les retours d'appel aux fonctions de la bibliothèque.

La seconde partie, de plus bas niveau, comprend les fonctions de transfert des paquets sur le réseau natif. Elle consiste en une traduction des appels PVM de bas niveau en des appels système. Cette partie reconnaît le système de la machine hôte et s'adapte à son architecture. Elle sera notre point de départ pour le portage de PVM sur la machine MPC et pour l'optimisation du coût des communications dans le cas de cette dernière.

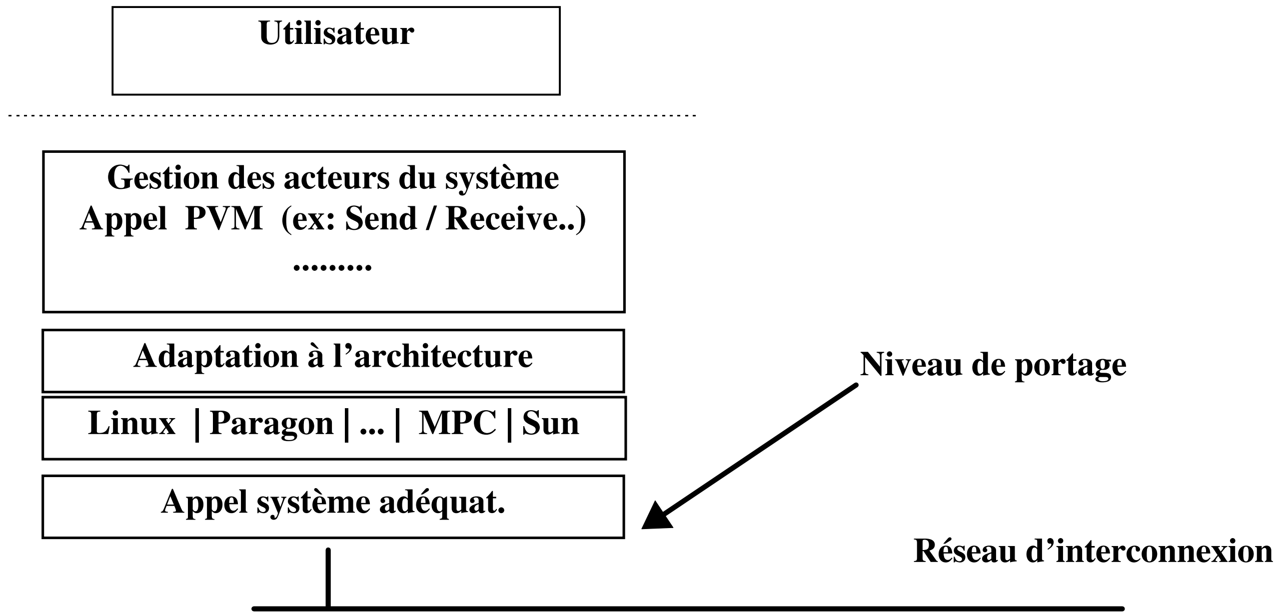


Figure A : Couches du système PVM.

II. LA MACHINE MPC

1. Le model de communication

Le model de communication implémenté dans la machine MPC, est le passage de message sous le mode ***Receiver-driven***. Autrement dit, le transfert du message se déroule sous le contrôle du récepteur.
Le transfert est initié par le récepteur du message en exécutant la primitive ***slrpv_receive***. Par cette primitive, le récepteur spécifie au système les zones mémoires destinées à la réception de

son message et provoque l'émission d'un message de contrôle de type ***receive***. Par ce dernier, l'émetteur est invité à transférer ses données dans la zone mémoire spécifiée par le récepteur. Ceci même si les données sont prêtes à être émises et l'émetteur à effectuer un appel un appel ***slrpv_send()***.

La nature synchrone de ce mode de communication fait ce qu'il n'est pas nativement utilisable pour les communications de bas niveau du système PVM. Cette difficulté sera surmontée en partie grâce à une acceptation préalable du récepteur d'un message potentiel de la part de tous les émetteurs qu'il connaît sur le réseau (la machine virtuelle). Par contre, cette technique nous contraint à sérialiser la réception d'un message (voire paquet ou fragment pvm) par une tâche et l'émission du suivant vers cette dernière par le même émetteur.

Par ailleurs, la communication dans MPC suppose l'existence d'une information commune à l'émetteur et au récepteur. Il s'agit du canal de communication. Ce canal, sert à distinguer parmi les destinataires potentiels d'un message se trouvant sur un même noeud.
Une gestion rigoureuse de ces canaux est nécessaire pour adapter ce mode de communication aux échanges effectués sous PVM. Par ailleurs, l'information relative au canal ouvert doit être maintenue par les tâches qui dialoguent.

2 . Choix des fonctionnalités utilisées pour le portage

Parmi les fonctionnalités offertes par les couches de communication MPC, nous utilisons celles de la couche SLRV [2]. Notre choix est fondé sur un travail précédent [5] réalisé dans le cadre d'une évaluation des performances et du coût de chaque niveau. Ce travail montre que le gain en performance lors de l'utilisation de la couche SLRP ne justifie pas les contraintes d'un adressage physique des données à transférer.

III. LA COMMUNICATION POINT A POINT VIA LE RESEAU HSL

Le but de cette section est d'illustrer le transfert d'un paquet de données entre deux tâches PVM situées sur deux noeuds différents de la machine MPC. L'une des deux tâches pouvant être le démon PVM.

Pour que ce transfert puisse se faire, une connexion entre ces deux tâches est préalablement réalisée. Le protocole de connexion, dont nous présenterons le détail un peu plus loin, assure que chacune des deux tâches alloue deux tampons verrouillés en mémoire physique appelés « buffer de réception et buffer d'émission ». Elle garde trace aussi du noeud de la tâche distante ainsi que du canal ouvert pour cette connexion.

La tâche émettrice recopie les données à émettre dans le buffer d'émission correspondant à la tâche destinataire, puis effectue un **slrpv_send().** Ce dernier ne peut être effectif que si la tâche réceptrice a effectuer un **slrpv_recv()** au préalable. L'opération aura pour résultat le transfert des données dans le buffer de réception de la tâche destinataire (sur le noeud distant).

A la prochaine scrutation du buffer de réception, la tâche réceptrice recopie les données avant d'effectuer un autre **slrpv_recv** acceptant ainsi le prochain message - paquet - . Les données reçues sont alors transmises à la partie indépendante du système de la bibliothèque PVM qui s'occupera de leur traitement.

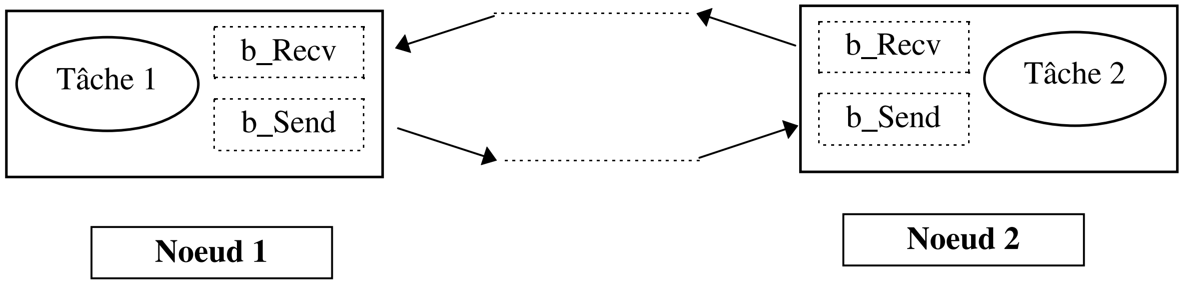


Figure B : Connexion HSL entre tâches distantes

La généralisation de ce schéma, fait l'hypothèse que chaque tâche garde trace des connexions qu'elle ouvre avec les autres tâches de la machine virtuelle.
Ces informations sont regroupées dans le contexte de la bibliothèque associée à la tâche. une structure de données connexion est donc introduite;

```
typedef struct  _connection {
        int tid;                 /* Tid de la tâche avec laquelle je suis connecté */
        int node_number;         /* Numéro du noeud , si la tâche est distante */
        int type_connection;     /* Distante ou locale */
        int desc_ connection;
        caddr_t  send_buffer;
        caddr_t  buffer_buffer;
        struct _ connection   *l_link;      /* Lien bidirectionnel gauche  */
        struct _ connection   *l_rlink;     /* Lien bidirectionnel  droit    */
        } connection ;
connection  *active_conn;
```

II. ARCHITECTURE ET FONCTIONNEMENT DE PVM-MPC

L'implémentation de la machine virtuelle PVM sur MPC n'utilise qu'un seul démon pou toute la machine virtuelle. Ce démon initialise la machine virtuelle et assure la gestion de toutes les tâches comme si elles étaient locales à son noeud.

Aussi, le noeud sur lequel tourne le démon est le noeud de démarrage. Les autres noeuds MPC sont des noeuds de calculs pour la machine virtuelle.

Dans cette implémentation, une première tâche peut être lancer sur le noeud de démarrage et intégrer la machine virtuelle par l'établissement d'une connexion *Unix Socket* avec le démon. Dans la suite de ce document nous appellerons cette tâche ‘‘ la tâche maître’’.

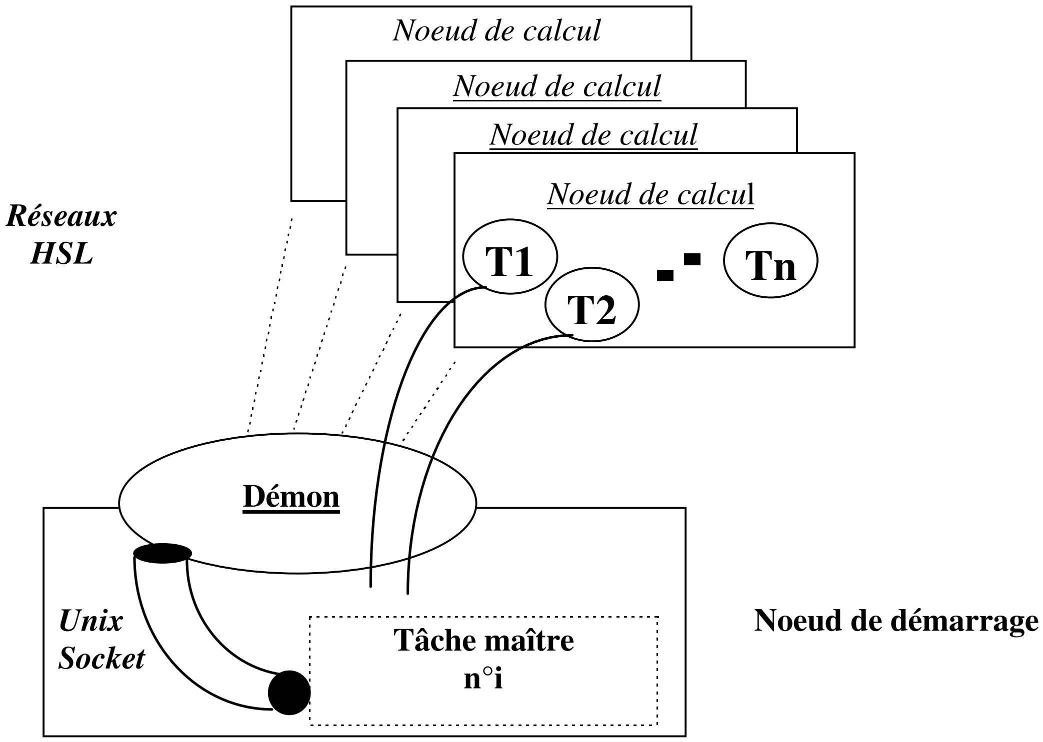


Figure C: Architecture générale de PVM-MPC

Par ailleurs, la seule manière de lancer des tâches sur les noeuds de calcul est d'effectuer un **pvm_spawn()** via la tâche maître, sur le noeud de démarrage.
Le choix des noeuds de calcul devant supportés les tâche de calcul est laissé à la charge du démon pour un meilleur rendement. Un protocole de placement statique est utilisé.

1 . Création des tâches de calcul

Une tâche de calcul (située sur un noeud sans démon) est constituée d 'un couple : processus unix et contexte d'exécution PVM (figure **D** suivante). Le premier est le processus créé pour effectuer un travail de fond pour l'application utilisatrice de PVM. Quant à la seconde partie, il s'agit d l'ensemble des informations nécessaire à l'exécution de ce processus au sein de la machine virtuelle, contenant entre autre les paramètres de ses connexions directes avec les autres tâches. Ce contexte est initié lors du premier appel de la bibliothèque PVM. Il sera maintenu tout au long de l'exécution jusqu'à l'appel de **pvm_exit()**
par la tâche -ou d'un **pvm_kill()** par le démon - où cette dernière ne fera plus partie de la machine virtuelle.

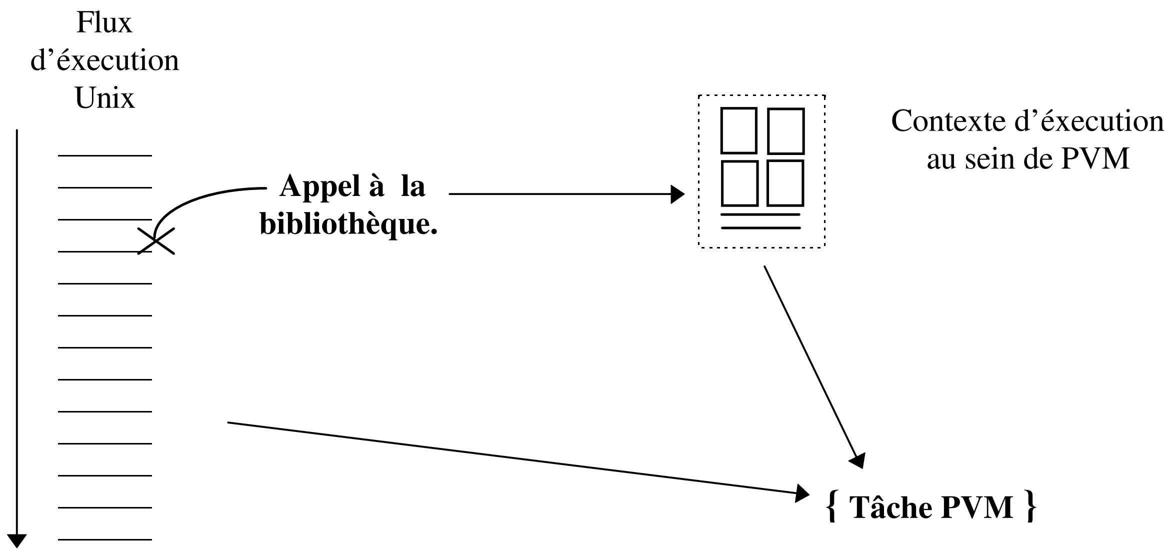


Figure D : Contexte d'une tâche PVM

Une tâche de calcul est créée via l'exécution , par le démon, d'un **rsh** (remote shell) sur le noeud distant choisi pour l'accueillir (figure **E**). Le démon se met alors en attente jusqu'à la réception d'une demande de contexte de la part de la tâche lancée. Cette demande est produite lors de l'exécution d'un appel à la bibliothèque PVM. Le contexte de la tache, contenant entre autre son TID (Task Identifier géré par PVM), lui est alors communiqué. Du même coup, une connexion HSL telle que décrite précédemment, est établie entre cette tâche et le démon. Cette connexion est maintenue tout au long de la durée de vie de la tâche au sein de la machine virtuelle. Elle servira aux messages de contrôle avec le démon ainsi qu'aux échanges de données avec la tâche maître.

Notons toutefois, qu'en n'utilisant que le réseau HSL même pour l'initialisation des bibliothèques liées aux tâches de calcul, un canal des communications initiales est nécessaire. Ce canal est négocié pour chaque noeud MPC en fonction du nombre de tâches déjà sur ce dernier. Ceci pour éviter tout conflit entre communications initiales pouvant conduire à un indéterminisme bloquant.

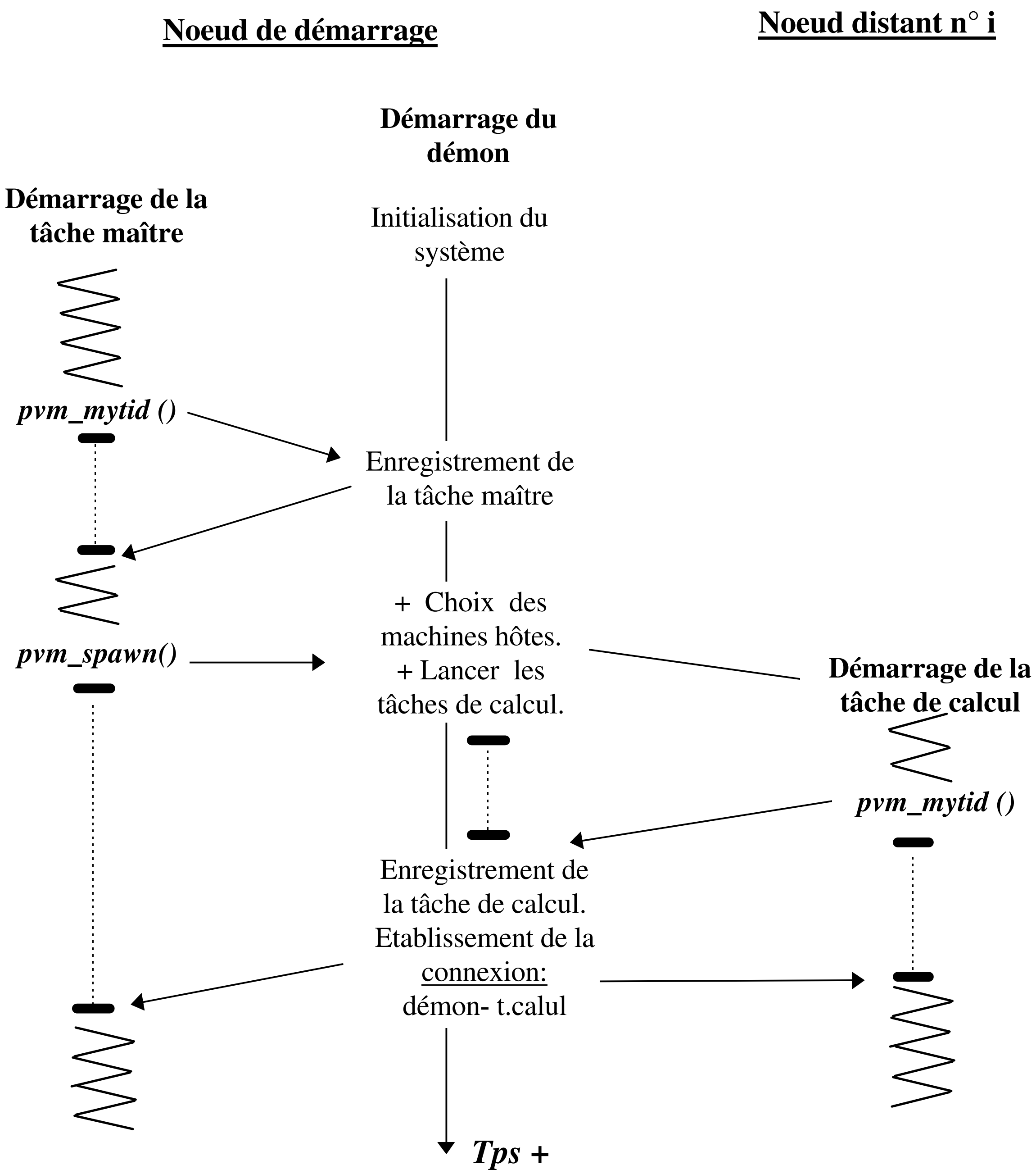


Figure E : Création et initialisation des tâches PVM

2 . Le TID : Task Identifier

Le TID, ou Task Identifier, est le seul moyen pour le démon de distinguer les tâches tournant sur la machine virtuelle, les unes des autres. Cet identifiant, entre autre sert pour le routage des messages et pour le contrôle de l'état des processus applicatifs.

Un TID au sens PVM est constitué de deux parties. La première comprend les champs S, G, et HOST. Les champs S et HOST du TID d'une tâche permettent de reconnaître le démon qui s'occupe de sa gestion au sein de la machine virtuelle. Dans un adressage de messages, ces mêmes champs désignent un sous ensemble de tâches PVM. Un adressage hiérarchique est

alors possible. Ce mode d'adressage est particulièrement intéressant dans le cas d'une machine à plusieurs démon chacun gérant un sous ensemble de tâches et qui tournent sur des machines hétérogènes.
Le champs G indique que l'adressage est un adressage de groupe.

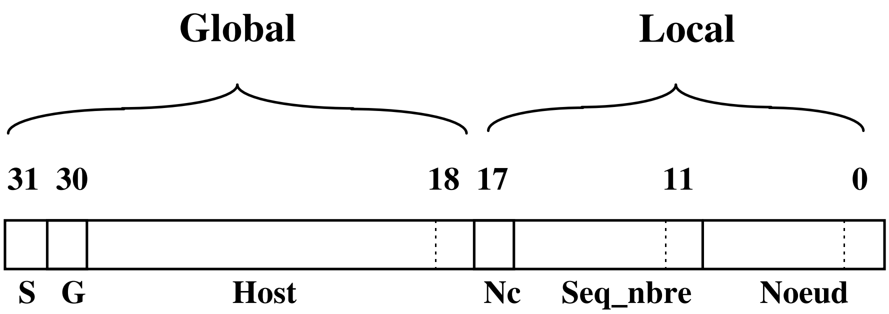


Figure F : Construction du TID : Task Identifier

Le champ LOCAL porte une signification interne pour chaque implémentation du démon. Pour notre cas, nous le divisons en trois champs qui auront la signification suivante:
Noeud : Le numéro de noeud MPC, local à la tâche.
Seq_nbre : Le numéro de la tâche sur le noeud, pour la distinguer des autres.
Nc : Vaut 1 si noeud de calcul,..0 sinon. Cette information est nécessaire car la communication avec la tâche maître (se trouvant sur le noeud du démon) se fait toujours au travers du démon et des sockets Unix.

3 . <u>Communication : tâche de calcul-tâche maître</u>

La tâche maître se trouve sur le noeud du démon. Toutes ses communications avec les tâches distantes se feront via la connexion de ce dernier avec ces tâches. Les messages que la tâche maître veut émettre sont confiés au démon qui les transfert à la tâche destinataire grâce à sa connexion HSL avec cette tâche. De même, les messages destinés à la tâche maître sont reçus par le démon puis délivrés à cette dernière. rappelons que la communication démon-tâche maître est réalisée par sockets unix locales .

Ce choix d'implémentation est justifié par le fait que le démon conserve une connexion avec chaque tâche et qui n'est pratiquement utilisée que pour les messages de contrôle. De plus, la tâche maître aura besoin du même nombre de connexion que le démon pour dialoguer avec les taches de calcul. Ce qui double les besoins en mémoire pour les tampons d'échange et nécessite de maintenir un contexte d'exécution supplémentaire pour la tâche maître.

Par ailleurs, le passage systématique par les sockets UNIX pour les communications avec la tâches maître pourrait constituer un goulot d'étranglement pour le reste du système.

4 . <u>Communication entre tâches de calcul</u>

Nous distinguons deux situations suivant que les deux tâches sont distantes ou se trouvent sur le même noeud MPC.

Le premier cas revient à la communication point à point HSL décrite plus haut. Quant à l'établissement de la connexion, le protocole sera décrit dans ce qui suit.
La communication entre taches du même noeud est réalisée par recopie de buffer entre les espaces d'adressages des deux tâches communicantes. Une connexion entre les deux tâches est aussi nécessaire et pour cela un protocole est introduit.

V. LA COMMUNICATION ENTRE TACHES DE CALCUL DISTANTES

1. Etablissement d'une connexion entre tâches distantes :

Une tâche désirant émettre un message à une autre, pour une première fois, commence par établir une connexion directe avec celle ci.
Cette connexion se fera via le démon. Après avoir créer deux tampons verrouillés en mémoire physique (l'un pour l'émission, l'autre pour le réception), elle envoie une requête de connexion au démon portant le TiD de la tâche à laquelle elle veut se connecter. Elle se met alors en attente d'une réponse de la part du démon avec le numéro de canal attribué à la connexion. Puis d'une acceptation de la connexion de la part de la tâche distante concernée.
De son côté le démon reçoit la première requête, attribue la canal à la connexion, puis répond à la première tâche avec le numéro de canal, et envoie un message à la seconde avec le numéro de ce canal et le TID de la tâche désirant s'y connecter.

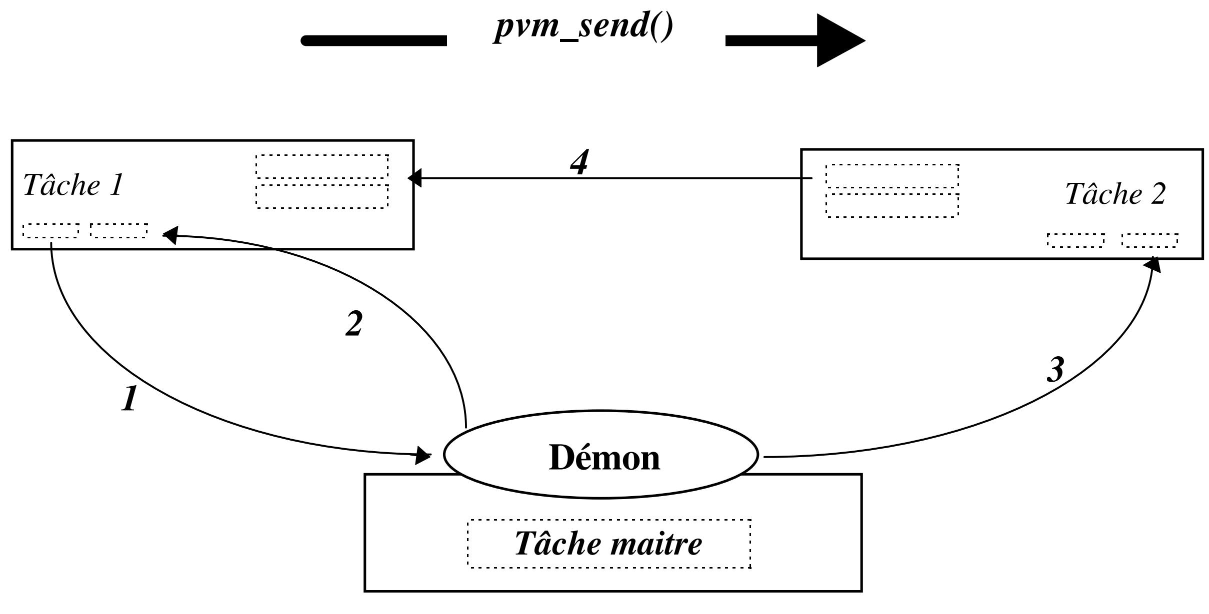


Figure G : Protocole de connexion HSL

A la réception, de cette requête par la seconde tâche, elle alloue les deux tampons nécessaires à la connexion puis répond à la tâches émettrice lui indiquant qu'elle peut émettre son message. Voir figure G.
Chacune de ces deux tâches garde trace de cette connexion et l'utilise pour ses communications avec la tâche homologue tout au long de leur vie.

2. L'attribution des canaux

La distribution des canaux est centralisée au niveau du démon. Il les attribue au fur et à mesure que des connexion inter-noeuds sont requises et garde trace des canaux alloués.
Un canal est associé à deux tâches(donc à deux TiD) et est bidirectionnel.
Le nombre maximal des canaux pouvant être ouvert entre deux noeuds est fonction (carré) du nombre maximal de tâches pouvant être supportées par un noeud MPC.
Le démon a un canal ouvert avec chaque tâche de la machine virtuelle, vu qu'il est connecté à toutes les tâches dès l'initialisation de leurs bibliothèques.

3. L'attente active

Lors d'une demande de connexion, une tâche se met en attente d'un message de réponse de la tâche connectée. Cette attente est faite de scruttant le buffer de réception de cette dernière.
Une tâche en attente active risque de ne pas répondre aux messages de contrôle arrivés du démon. Une boucle de requêtes risque donc de créer un interblocage. Il est donc impératif que les attentes actives acceptent les messages de contrôle de la part du démon. Ce que nous avons réalisé.

4. La communication directe : vision SLRPV

L'émission d'un message PVM correspond au niveau SLRPV à une correspondance entre un ***slrpv_recv()*** - acceptation préalable du message de la part du récepteur- et le ***slrpv_send()*** effectué.
Le processus PVM est alors mis en attente jusqu'à l'émission effective des données. Cette synchronisation du cote de l'émetteur permet d'éviter que le contenu du buffer d'émission ne soit écrase par des données du message suivant.
Du coté du récepteur, la réception d'un message n'est détectée que lors de la scrutation du buffer de réception. Un message, ***slrpv_recv()*** est alors émis pour en accepter le suivant.

Dans le cas d'un message constitué de plusieurs paquets (fragments), cette sérilaisation se rapporte au niveau transfert de paquets. La figure H, montre un exemple de transfert de message vu au niveau PVM et Slrpv. Dans cet exemple, le message pvm est constitué de plusieurs fragments. Chacun de ces derniers est transféré par un appel ***Slrpv_send*** précédé d'un ***Slrpv_recv*** autorisant ce transfert. Aussi pour éviter un éventuel interblocage des tâches communicantes, nous introduisons une priorité pour la consommation des paquets disponibles avant d'en traiter ceux prêts à l'émission.

En effet, une tâche désirant émettre un paquets (frag) effectue un appel ***Slrpv_send*** et ne reprends le cours de son exécution que lorsque ce paquet est effectivement émis. Durant ce temps d'attente tous les messages (paquets) disponibles dans les tampons de réception sont récupères et enfilés avec les messages prêts à être délivrer. Ainsi, tout message en provenance d'une tache distante est consomme au plus tard immédiatement après un message émis vers elle et une fluidité des messages Slrpv_recv est assurée entre les deux taches communiquantes.

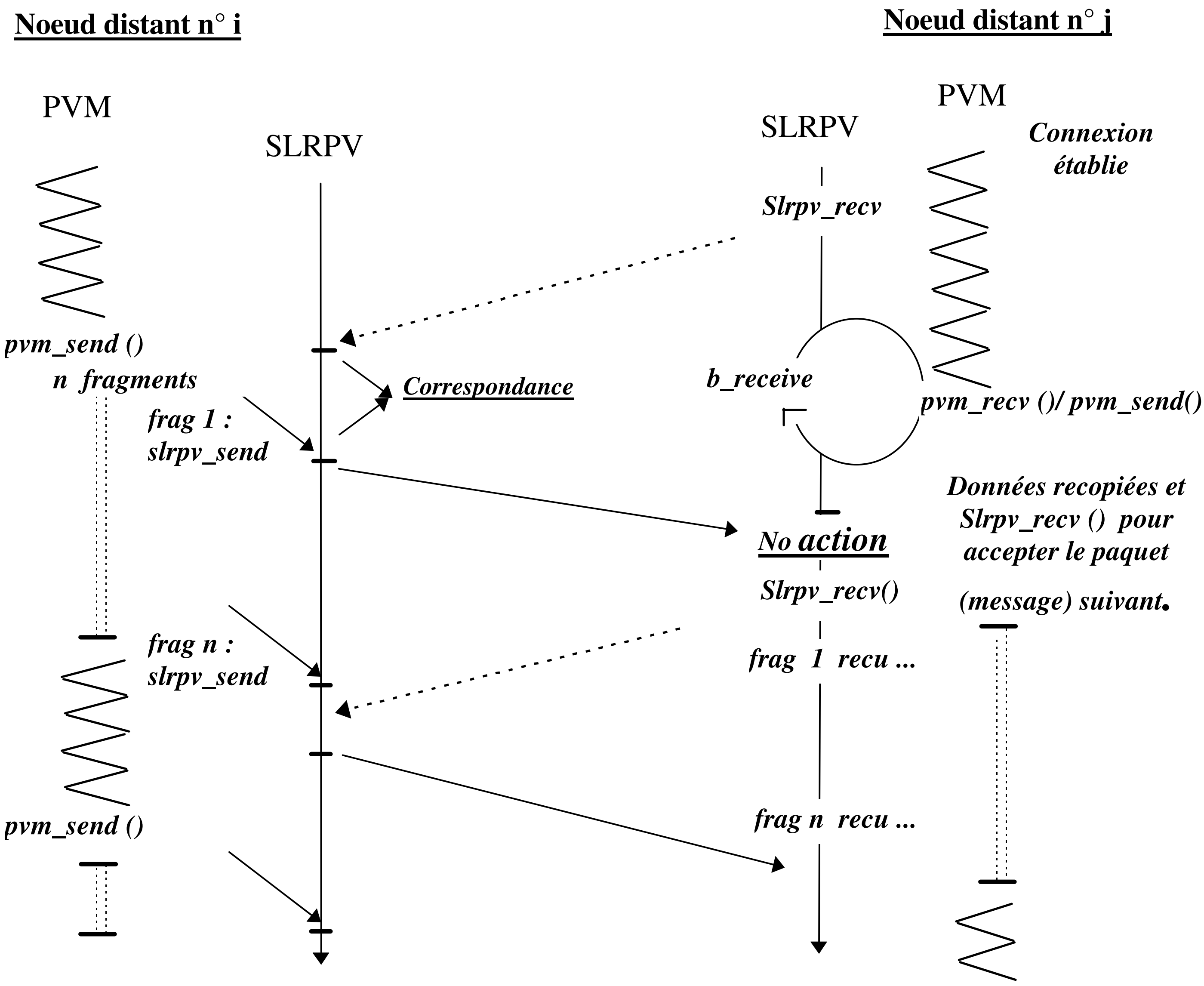


Figure H : Tranfert PVM et transferts SLRPV

VI. LA COMMUNICATION ENTRE TACHES DE CALCUL LOCALES A UN MEME NOEUD MPC :

L'adaptation des communications pvm aux seules fonctionnalités de communication de la machine MPC, supposent que les tâches communicantes se trouvent sur des noeuds distants.
Une première implémentation a porté sur la limitation des taches de calcul au nombre de noeuds. Cependant, ceci étant trop contraignant pour les applications, si le nombre de noeuds de la machine est trop réduit, les communications locales ont été introduites.
Elles obéissent aux même traitement que les communications distantes, par soucis de conserver l'existant. Cependant, la mise sur le réseau des données dans le cas des communications distantes, correspondent dans le cas des communications locales , à une recopie de données.

Une connexion locale nécessite deux buffers (et non pas quatre comme pour le cas des connexion HSL). Un buffer correspond à un sens de communication, la première tâche y écrit alors que la seconde y lit les données. Un protocole de producteur consommateur garanti l'intégrité et la cohérence des données. Quant à l'accès aux emplacement se trouvant dans deux espaces d'adressage différents, les mécanismes de partage de mémoire offerts par FreeBSD permettent le mapping des zones mémoires des processus dans des espaces appartenant à d'autres processus. Le partage est explicite et pour cela un protocole de connexion est nécessaire.

1. Etablissement de la connexion :

De même que pour les connexions distantes, ces dernières passent aussi par le démon. Une tâche qui désirent se connecter à une autre se trouvant sur le même noeud, alloue un buffer dans la mémoire partagée et en transmet la clé au démon qui à son tour la retransmet à la tâche destinataire. Cette dernière mappe le buffer distant dans son espace d'adressage, et à son tour alloue un buffer partagé dont elle transmet la clé directement à la première tâche.
Un champ dans la structure de données **_connection** distingue les connexions internes et externes.

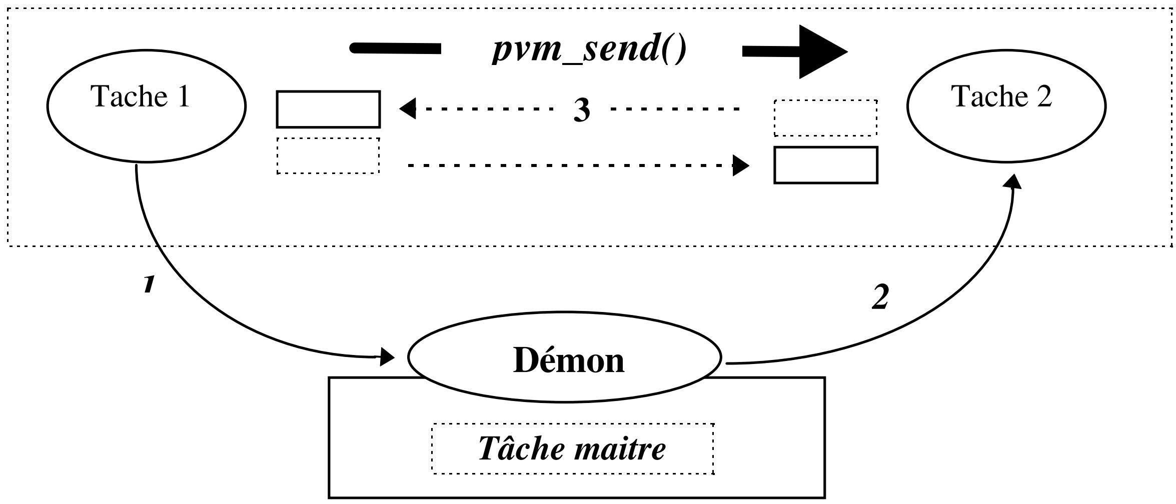


Figure I : Protocole de connexion locale.

VII - EVALUATION DES PERFORMANCES DU SYSTEME :

Afin d'évaluer les performance du système, nous procédons à une série de test suivant la méthodologie proposée par Dongarra and Dunigan [6]. Pour cela, sont utilisés deux processus s'exécutant sur deux noeuds distants, reliés par une connexion HSL, suivant le pseudo code :

Processus 1	**Processus 2**
Initialisation;	
Top timer;	Initialisation;
Pour i=1 à N faire	Tant que VRAI faire
émettre 1 message à M octets;	recevoir 1 message à M octets;
recevoir 1 message à M octets	émettre 1 message à M octets
fait;	fait;

Les résultats sont synthétisés dans les courbes des figures 9 (a-b-c) et 10 (a-b) qui décrivent les transfert de données à deux niveau d'application : PVM et SLRPv.

La figure 9 décrit la latence de notre implémentation de PVM. Nous pouvons remarquer que le coût additionnel de l'utilisation de PVM au dessus de SLRPv est relativement constant et égal à 100 microsecondes. Notons toutefois que la latence SLRPv citée ci dessus est composée de la latence du réseau HSL et de l'overhead des couche logicielles sous-jacentes à SLRPv.
La figure 10 compare la bande passante disponible au dessus de SLRPv à celle disponible au dessus de PVM. Ce qui donne une estimation de la bande passante du réseau qui reste inutilisée et montre que le portage à été efficacement réalisé.

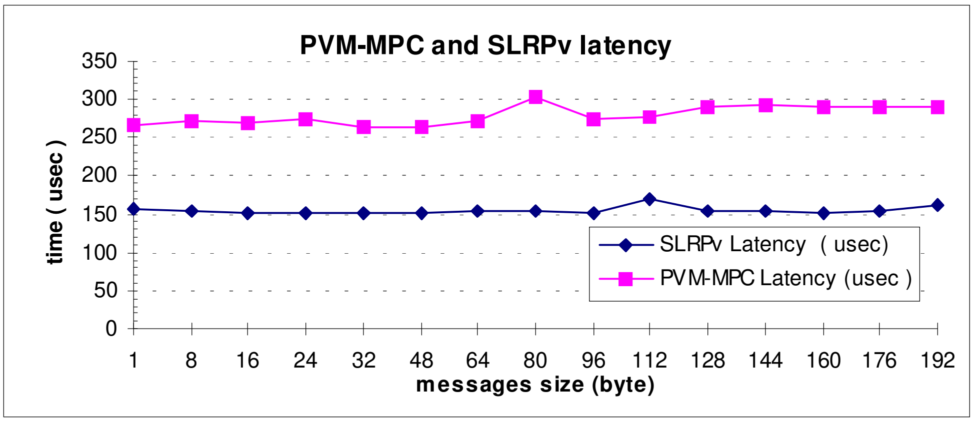


Figure 9-a

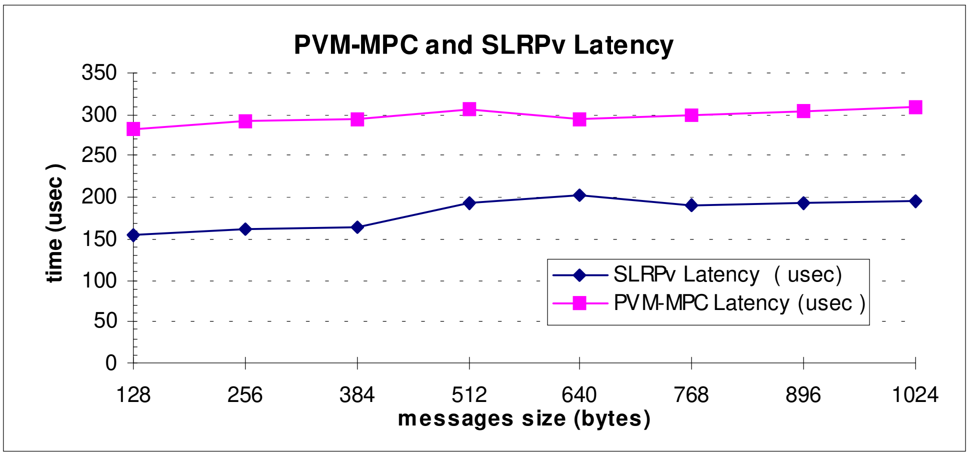


Figure 9-b

Toutes ces mesures ont été réalisées sur la plate-forme expérimentale MPC du laboratoire ASIM-LIP6 composée de PC pentium-133 Mhz et disposant de 64 Mbytes de mémoire vive. Aussi dans ces tests, le buffer démission est de 16 Koctets. L'émission d'un message de taille T nécessite donc (T / 16K + 1) appels à SLRV_Send(). Ce qui explique la réduction périodique des performances tous les 16 Koctets observée sur les graphiques.

La taille du buffer d'émission est un paramètre du système. Afin d'obtenir les meilleurs performances avec les messages de grandes tailles, il suffit de fixer T à une valeur conséquente. Toutefois, mettre une valeur trop grande réduit le nombre de buffer d'émission potentiels et donc le nombre de connexion possible pour chaque tache PVM.

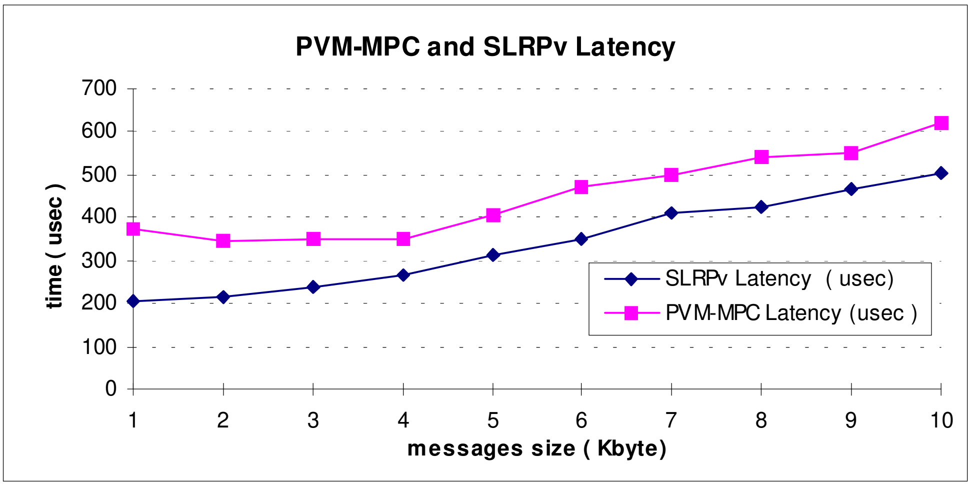


Figure 9-c

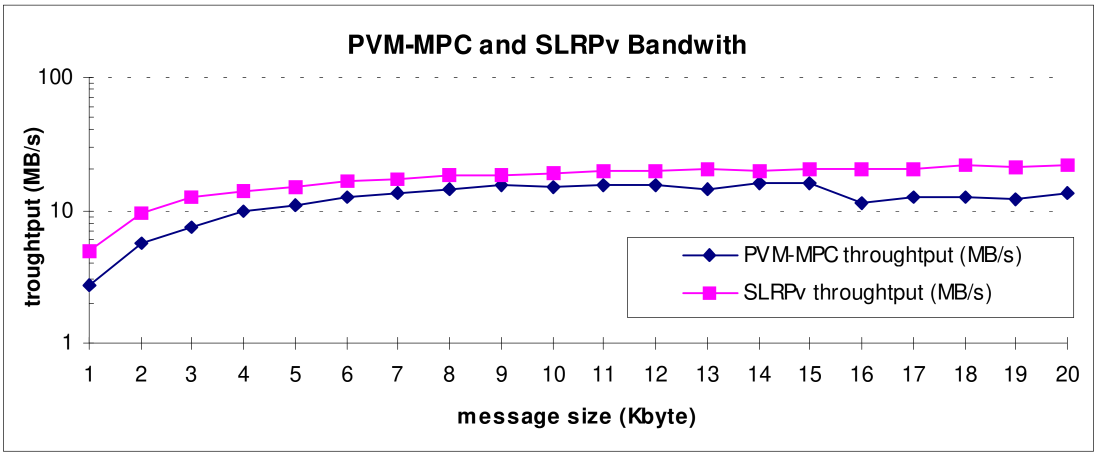


Figure 10-a

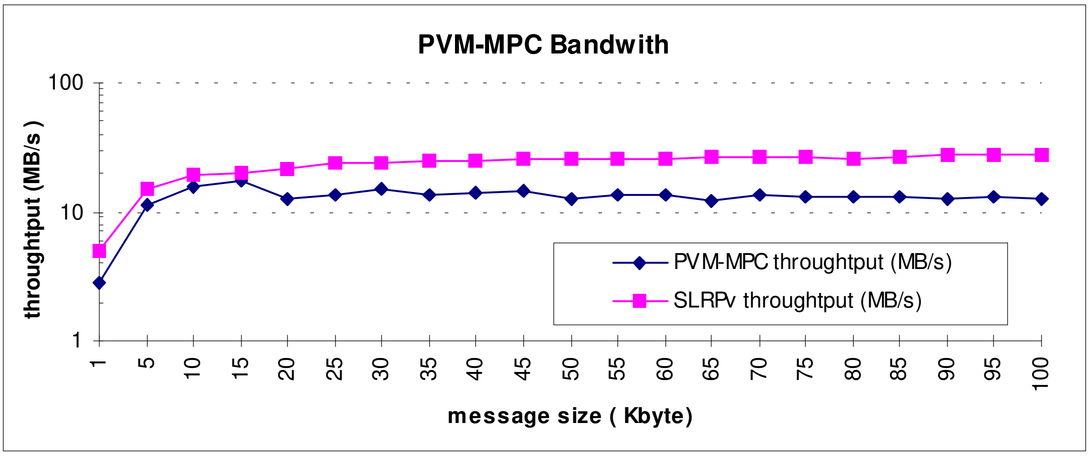


Figure 10-b

Bibliographie

[1] IEEE draft std p1355 standard for heterogeneous interconnect . Tecnical report, Institute of Electrical and Elictronic Engineers, Inc, 345 East 47th Street New York NY 10017, USA August 1994.

[2] Interface Logiciel MPC/1 , Manuel du programmeur, version 3.0. Alexandre Fenyö, Frederic Potter , Pierre David, Université Paris 6-MASI, 18 Dec 1996

[3] Emulateur Logiciel MPC/1: Manuel d'Installation, version 3.0, Alexandre Fenyö, Frederic Potter , Pierre David, Université Paris 6-MASI, 18 Dec 1996

[4] PVM3 User's Guide Abd Reference Manual, Al Geist et al, Oak Ridge Laboratory, Oak Ridge,Tennessee TN 37831-6367

[5] Sebastien Fernier. Septembre 1997. Memoire de stage dans le cadre du DEA SI -LIP6-, effectué à l'ENST sous la direction d'Isabelle Demeure.

[6] Message-Passing performance of various computers. Jack J.Dongara, Tom Dunigan. disponible sur le web http://www.netlib.org/utk/papaers/commperf.ps.

ANNEXE 1
Utilisation de PVM-MPC
sur la plateforme expérimentale à 4 noeuds MPC.

Il nécessaire pour cela de disposer d'un compte utilisateur sur chacune des 4 machines pmpc0, pmpc1, pmpc2, pmpc3. Chacun de ces noeuds identifie les autres grâce à un fichier de configuration utilisé par le processus client lors de la configuration a l'initialisation de la machine MPC.
Sur le poste pmpc0, noeud de démarrage de PVM-MPC, on dispose d'un script de fenêtrage permettant de se logger a partir de n'importe quelle machine du sous-domaine **lip6.fr.** Un exemplaire général est fournis sous le nom: pvm-mpc.cons.host, ou host est le nom de la machine distante.
Pour l'adapter à une machine spécifique, il s'agit de remplacer a l'intérieur du fichier toutes les occurrences de host par la machine en question. (un xhost +, préalable permet l'ouverture des fenêtres de manière adéquate)

1 _ Initialiasation de la machine MPC:

Cette étape est nécessaire a l'amorçage des couches de communications MPC. Il s'agit de lancer un couple (Client / serveur) sur chacun des noeuds MPC.
La commande **gmake serverpmpcX** (où X=0,3) permet de lancer le server et est nécessaire pour enregistrer le noeud en question dans la machine MPC. La commande **gmake clientpmpcX** (où X=0,3) lance le client en mode émulateur. En fin, **gmake hclientpmpcX** permet d'initialiser la machine MPC en mode hsl.

2 _ Initialiasation de la machine virtuelle PVM

Il s'agit de lancer le démon pvmd3, sur le noeud de démarrage, qui crée une socket unix paramètrée du userid () au sens Unix.
Dés lors, la tâche maître peut être lancé sur ce noeud et s'exécuter au sein de la machine virtuelle PVM-MPC.

3_ Erreurs d'exécution

Lors d'une exécution, des erreurs peuvent se produire. Dans ce cas, un message d'erreur est renvoyé sur la fenêtre du démon, si l'erreur est produite par une tâche de calcul ou sur la fenêtre de la tache maître si l'erreur s'y est produite. Nous distinguons deux types d'erreurs, les erreurs de la librairie PVM, signalant une insuffisance des ressources ou des comportements anormaux du système, ou des erreurs des programmes utilisateur supportés. Les premières sont précédées de la mention [Libpvm xxx] : « message d'erreur ».

4_ Les fichiers de trace :

Un fichier de trace (log) est utilisé pour garder trace de ce qu'effectue le démon... Ce fichier est nommé pvml.%uid()% Il est recommandé aux utilisateurs d'en utiliser pour chaque tâche de calcul qu'il lance. Et pour cela il faut recompiler le système PVM en remettant en utilisation la fonction **pvmlogerror()** pour le cas MPC. Cette fonction se trouve dans le fichier lpvmmimd.c

ANNEXE 2
Les modifications dans le code source du système PVM

PVM offre trois niveaux d'imbrications pour ses fonctionnalités : Les fonctions génériques (LPVMGEN.C) tous les appels définis par l'API PVM. Ces fonctions utilisent des fonctions de bas niveau PVM pour réaliser leur fonctionnalites. Ces fonctions de bas niveau sont, suivant que la tâche est sur un noeud avec démon ou sans, traduites en une séquence différente d'exécution. Ces séquences de tâche sont principalement distinguées dans LPVMMIMD.C et LPVM.C qui définissent le comportement de la tâche suivant qu'elle est liée â l'une ou l'autre des 2 bibliothèques (liées au moment de la compilation). Dans notre portage, ce comportement est soit impliqué par l'utilisation des fonctionnalités MPC soit par le fait que la tâche se trouve sur le noeud du démon.

PVMGEN.C

LPVMMIMD.C — LPVM.C

Slrpv — Socket.

Autour de ces deux sous-bibliothèques, un ensemble de modules sont nécessaires.
- HSL.C pour l'interface utilisateur des fonctions HSL utilisées par PVM. Des fonctions contenant principalement des IOCTLs y sont écrites.
- Tdpro.c / Tdpro.h : décrivent les codes des messages échanges par les tâche PVM et le démon qui les gère. Ces codes permettent de différencier les messages de contrôle de ceux portant des données.
- Task.c / Task.h définissent les comportements de haut niveau d'une tâche PVM. La structure de contrôle de la tâche y est aussi décrite.

Pour le démon :

Il s'agit principalement du module pvmdmimd.c du répertoire SRC/MPC. Ce fichier regroupe tous les comportements spécifiques aux démon PVM-MPC. Les fonctions qui y sont décrites sont appelés à partir du corps principal du démon PVM (fichier pvmd.c du répertoire SRC).

Notons toutefois que ces fichiers décrivent le comportement de la machine dés que l'architecture MPC est admise par PVM. Pour cela , il est nécessaire d'introduire dans le fichier PVMARCH.h et de le définir dans par MPC.def

ANNEXE 3
Le model de programmation sur PVM-MPC.

Le model de programmation sur PVM-MPC conserve la totalité de l'API PVM. Cependant, certaines propriétés doivent être prises en compte pour un bon fonctionnement des programmes sur cette plate-forme.

La plus importante de ces propriétés est sans doute que la robustesse de la machine virtuelle dépend de celle de chacune des tâches PVM . Aussi, une programmation cohérente et tolérante aux fautes serait très appropriée.

Le pvm_spawn() implémenté ne prend en parametres que le nom de l'exécutable et le nombre d'instances à lancer. Il renvoie en sortie les Tids affectés.
Le choix des machines hôtes n'est pas pris en compte.

Lors de la création d'une tâche de calcul, le démon Pvm reste en attente de l'établissement de la connexion avec celle-ci. Afin de minimiser cette attente, le code de chaque tache de calcul doit commencer par une cation PVM . ex: pvm_mytid()

Enfin, notre implémentation est , rappelons le, à démon unique. Ceci dit, deux comportement différents sont possibles pour les tâches suivant qu'elles retrouvent sur le noeud du démon ou bien sur un noeud de calcul. Aussi, deux librairies sont à lier à ces taches selon les cas cité. (voir exemple de Makefile ci après)

```
#           Makefile
#           .................


PVM_ARCH  =     MPC
ARCHLIB   =     -lrpcsvc
PVMDIR    =     ../..
PVMLIB    =     $(PVM_ROOT)/lib/$(PVM_ARCH)/libpvm3.a
PVMPELIB  =     $(PVMDIR)/lib/$(PVM_ARCH)/libpvm3pe.a

LIBS      =     $(PVMLIB) $(ARCHLIB)
NODELIBS  =     $(PVMPELIB) $(ARCHLIB)
GLIBS     =     $(PVM_ROOT)/lib/$(PVM_ARCH)/libgpvm3.a

SDIR      =     ..
BDIR      =     $(PVM_ROOT)/bin
XDIR      =     $(BDIR)/$(PVM_ARCH)

CC        =     cc
```

```
CFLAGS      =      -O -I../../include

$(XDIR):     $(BDIR)
        - mkdir $(XDIR)

$(BDIR):
        - mkdir $(BDIR)

default : master  slave

clean:
        rm -f  *.o

# tache devant s’execute sur le noeud de demarrage lié à la librarie LIBS
#

master: $(SDIR)/master.c  $(XDIR)
        $(CC) $(CFLAGS) -o master $(SDIR)/master.c  $(LIBS)
        mv master $(XDIR)

# tache devant s’execute sur un noeud de calcul  lié à la librarie NODELIBS
#

slave: $(SDIR)/slave.c  $(XDIR)
        $(CC) $(CFLAGS) -o slave  $(SDIR)/slave.c  $(NODELIBS)
        mv slave $(XDIR)
```

ANNEXE 4
Installation de la bibliothèque PVM-MPC

La bibliothèque est présentée comme un fichier **pvmroot.tar.gz.**

1_Configuration et Réseaux

La machine parallèle PVM-MPC est constituée d'un ensemble de stations de travail de type PC reliés par un maillage HSL et par un réseau local de type Ethernet. Chacun de ces PC portent un numéro de P-node qui permet de l'identifier sur la machine parallèle. Nous considérerons ces numéros ceux affectés par l'étiquetage du réseau HSL (lors du calcul des tables de routage).

Nous utiliserons la machine portant le numéro de P-node ***zéro*** comme racine de la machine parallèle PVM-MPC. Sur le disque de ce noeud (portant le numéro hsl 0) sera dédié un répertoire ***pvmroot*** visible via NFS et Ethernet par tous les autres PC constituant la machine PVM-MPC et reliés au réseau HSL.

Outre le mount de ce répertoire, il est nécessaire d'autoriser l'écriture dans le répertoire ***pvmroot/bin/MPC*** à tous les utilisateurs du système. Cette machine devra supporter la compilation des applications et contiendra les exécutables binaires des différentes applications PVM-MPC.

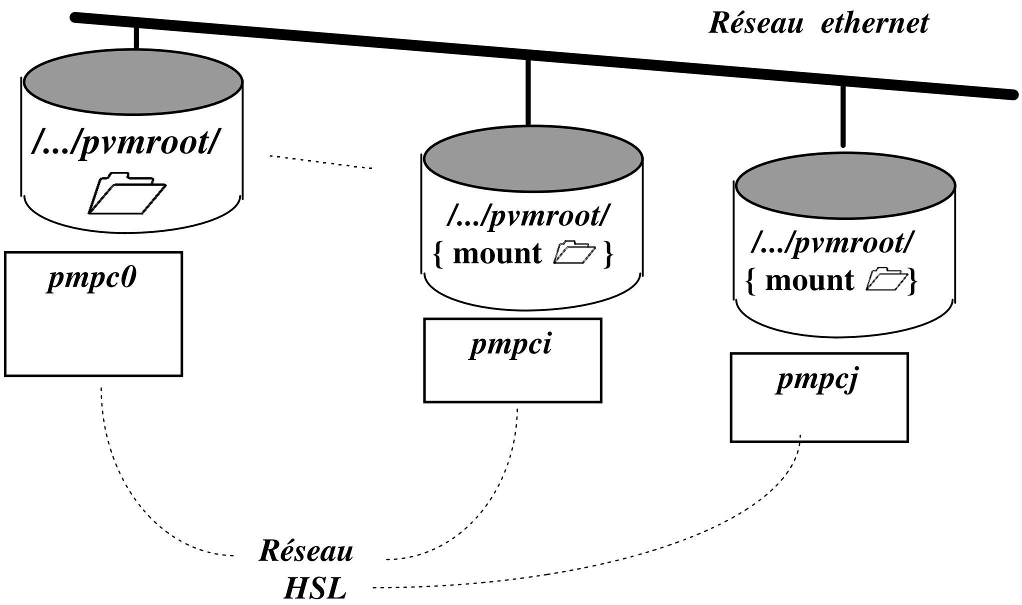


figure annexe : 4 -a

2_Installation

Pour installe la bibliothèque PVM-MPC , il suffit recopier le fichier ***pvmroot.tar.gz*** sur le noeud zéro, de le décompresser dans le répertoire partagé ***pvmroot*** puis d'effectuer un make dans chacun des répertoire suivants :

```
# cd  pvmroot/src/MPC
make
# cd pvmroot/pvmgs/MPC
make
```

Le système est alors installé et prêt a recevoir des applications.

3_Remarque

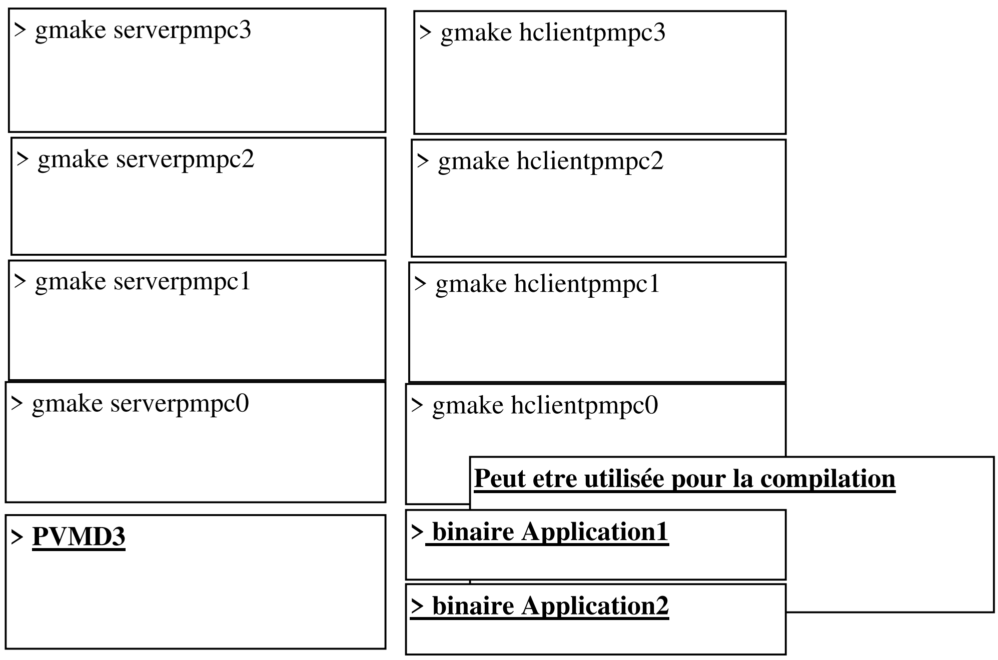
Pour des besoins de placement des taches, la bibliothèque doit reconnaître les machines appartenant au réseau HSL. Un fichier de configuration est accessible dans le répertoire ***pvmroot/src/MPC*** : C'est le fichier ***scorefile***. Ce fichier est consulté par la bibliothèque a son initialisation.

Aide mémoire pour une utilisation occasionnelle de PVM - MPC avec DSIM

DSIM, distributed simulator, est une application conçue aulaboratoire ASIM -LIP6 et qui a étét utilisée pour l'estimation des performances du sysème PVM-MPC en comparant ses résultats à ceux obtenus sur d'autres plat-formes.

1_Le fenetrage :

Dans le répertoire ~kmana sur pmpc0, on peut trouver des fichiers **pvm-mpc.cons.host** , il suffit de rempalcer host par le noeud distant d'ou on utilise MPC.
Les fenetres sont du nombre de 12, réprties suivant la figure A.



Les 8 fentres superieures serevnt a lancer le client-serveur de la machine.
Les fenetres en **caratères** sont situés sur pmpc0;
- pvmd3 dans le repertoire $PVM_ROOT/lib/MPC.
- binaire Application dans le repertoire $PVM_ROOT/bin/MPC.
- la fentre en arrière plan est initialement dans le reprtoire $PVM_ROOT/bin/MPC.

2_DSIM

ligne de commande Dsim < fichier en entrée>.

* Comment le faire marcher ? Je n'ai pas le mot de passe alors je crée le fentrage sous route, puis je fais.
su dsim dans les fentres du bas: celles necessaires à l'application (en **caratères**).

* Lancer pvmd3 : de la même manière puisqu'on est dans le bon répertoire. Reste à lancer Dsim. le path d'environnement de Dsim permet de voir le binaire dans

$PVM_ROOT/bin/MPC. Mais il est necessaire de se deplacer dans le répertoire qui contient le fichier de description du fichier à simuler. Pour cela :dans la fentre concernée , faire :

- cd pour revenir dans le repertoire root_dsim
- cd dsim/bench/ pour trouver un ensemble de reprtoire ..chacun pour un circuit. structurés de la manière suivante.

circuit / partionnement / < fichiers de description >

exple :

s5378 / part2 / s5378XXX : le circuit est S5378... le partionnement qui nous interrsse est part2 (partionné en deux), puis s5378 suffit pour retrouver tous les composants de description ::structurelles, comportementatles...

Structure du reprtoire Dsim

Dsim est visble par un ***mount*** , a partir de space1 sur eowyn.

/ space1 / Dsim / Alliance --------> à priori non introduite dans les Makefile, reste a verifier.

Dsim dev / sce repertoire de develloppement (avec des sous reerptoires pour les differentes versions) celle qui nous concerne est V330. Pour recompiler faire : make clean , puis make dans ce repertoire
(cd /space1/ dsim/dsim/dev/ sce /v330 ;
make clean; make)
Pour recompiler faire d'abord su dsim (pour les variables d'environnements).

bench est organise en sous répertoirechacun pour un circuit chaque repertoire circuit contient des sous repertoires **partX** (exple part2, part4, part8)
Il suffit de se mettre dans ces repertoire et de taper le nom du circuit sans extention.