Annexe E : Comparaison des bibliothèques de communication hétérogènes

Souvent, l'unique moyen de faire coopérer des programmes s'exécutant sur des ordinateurs hétérogènes et distants est d'adopter un modèle de programmation par processus communiquants [Hoare 78]. Dans ce modèle, les données d'un processus sont privées et la coopération entre processus n'est réalisable que par le biais de communications explicites. Ainsi, si l'exécution d'un processus nécessite l'accès à des données non locales, une requête doit être adressée au processus détenant les données désirées. L'échange de messages à travers le réseau d'interconnexion entre les processus est réalisé en mode point à point ou en mode de diffusion totale (*broadcast*) ou sélective (*multicast*) vers un groupe de processus (établi statiquement ou dynamiquement).

L'un des moyens les plus efficaces pour construire des applications client-serveur parallèles et/ou réparties basées sur un modèle de programmation par processus communiquants est d'utiliser des bibliothèques de communication par passage de messages (*Message Passing Libraries*). Ces bibliothèques facilitent la programmation et la collaboration d'applications s'exécutant et se synchronisant sur des machines hétérogènes (supercalculateur, station de travail, ordinateur de bureau).

Les quatre principales bibliothèques de communications hétérogènes utilisées sont :

1) **PVM** (***P**arallel **V**irtual **M**achine)* [Geist & al. 93] qui est **le standard de fait**. C'est un produit du domaine public issu des recherches du laboratoire de Oak Ridge National Laboratory. Le projet a débuté en 1989 et s'est déplacé dans les universités du Tennessee et de Carnegie Mellon.
2) **MPI** (*Message Passing Interface*) [MPIF 93 & 94] est une proposition de standard de bibliothèques de communication par passage de messages. Le but ultime est de proposer MPI comme un standard utilisable, c'est pourquoi il rassemble les meilleures fonctionnalités des bibliothèques déjà existantes.
3) **P4** [Butler & al. 94] (*Portable Programs for Parallel Processors)* est un produit du domaine public issu des recherches du laboratoire de Argone National Laboratory.
4) **Linda et Network Linda** [Carriero & al. 94 - Corbel & al. 95] sont des produits commerciaux issus des recherches du laboratoire de l'université de Yale. Une version du domaine public est disponible, mais elle est restreinte au monde Unix.

Principe de fonctionnement

Les bibliothèques de communications par passage de messages rendent transparentes, au niveau applicatif, les spécificités des machines réelles composant les noeuds de calcul et formant une machine dite virtuelle.

L'utilisation de ces bibliothèques se fait soit dynamiquement à l'exécution, soit statiquement lors de la compilation, soit de manière mixte. Ainsi, parmi les quatre bibliothèques précédemment citées :

- PVM utilise un environnement dynamique instanciable à l'exécution. Un démon (*daemon* au sens Unix du terme) est alors lancé avant toute exécution d'application. PVM est donc une surcouche du système d'exploitation et s'exécute dans l'espace mémoire de l'utilisateur. Tout accès aux primitives de la bibliothèque de PVM est interceptée par le démon, qui gère les accès distants (concurrents ou non), les communications, les synchronisations et la cohérence de l'état de la machine virtuelle.

- un programme MPI [Gropp & al. 94] consiste en un ensemble de processus autonomes, exécutant leur propre code. Les codes exécutés par chaque processus n'ont pas à être identiques. Par contre, MPI ne spécifie pas de le modèle d'exécution de chaque processus, car MPI a été spécifié au niveau sémantique, le choix des détails d'implémentation sont donc laissés à la charge des équipes qui l'implémentent.
- P4 supporte à la fois le modèle de mémoire partagée (en utilisant des moniteurs) et le modèle de mémoire répartie (basé sur des passages de message). P4 n'utilise aucun système de redirection des appels, si bien que chaque appel de primitive interfère directement avec le système. C'est à l'intérieur même des fonctions appelantes de P4 que sont décrites les caractéristiques de l'utilisation de la fonction.
- Linda a choisi une politique mixte. Dans l'environnement Linda, un compilateur spécial est chargé de produire du code source C ou Fortran, pour générer des programmes natifs. Les caractéristiques du compilateur sont identiques à celles du compilateur de P4, mais Linda y adjoint un gestionnaire de réseaux chargé de la cohérence des données, ainsi que des politiques d'ordonnancement et de la gestion de la machine virtuelle. A la différence de PVM, les fonctions appelantes ne sont pas redirigées, malgré l'utilisation d'une surcouche du système proche d'un processus serveur de type démon.

La modification dynamique de paramètres du système n'est possible qu'en cas de liaison dynamique des programmes avec les bibliothèques. Ainsi, un processus peut être dynamiquement ajouté à un groupe de processus créé au lancement de l'application. La création d'une surcouche au système, ainsi que l'interception par la couche logicielle des fonctions appelées entraînent en général une perte d'efficacité.

La gestion de processus et de groupes de processus

Si au départ ces bibliothèques de programmation ne devaient résoudre que des problèmes liés aux calculs répartis et/ou parallèle, on s'aperçoit qu'elles offrent, pour la plupart, une gestion des processus dans leurs systèmes. En effet, seul MPI déroge à cette règle et ne dispose pas de support pour la manipulation des processus et des processus légers (son modèle de processus est statique).

Dans chacune des autres bibliothèques étudiées, les créations de processus sont explicites, c'est à dire que c'est à l'utilisateur de demander la création d'un processus. Lors de la création d'un processus, la possibilité de demander un placement de celui-ci sur un noeud particulier de la machine virtuelle est offerte au programmeur. Dans PVM, c'est le démon qui se chargera du placement, alors que dans P4 ou Linda, c'est le processus créateur, avec la collaboration du système, qui effectuera cette tâche. Il est à noter que cette approche de placement est beaucoup plus lourde que dans PVM. Notons que P4 distingue les processus maître des autres processus appelées esclaves hiérarchisables sur plusieurs niveaux pour obtenir des conteneurs (*clusters*) d'exécution.

Linda dispose de plus d'une commande de création implicite de processus (avec la commande *eval*) s'approchant de la fonction *future* du Lisp. Cette commande est un envoi de messages avec une évaluation de ses uplets. Si un de ceux-ci correspond au résultat d'une fonction, un processus est créé pour évaluer celle-ci et le processus émetteur reprend son exécution sans attendre.

P4, PVM et MPI intègrent totalement la notion de groupes de processus, permettant une gestion plus efficace des ressources et des envois de message sélectifs (*multicast*). Dans P4, les groupes sont définis statiquements et restent immuables durant l'exécution. Dans PVM et MPI l'insertion et la suppression d'un processus dans un groupe est dynamique et un processus peut même appartenir à plusieurs groupes simultanément. A cause des groupes dynamiques, chaque communication collective fait appel à un serveur de groupe, ce qui entraîne un impact sur les performances.

Seul Linda n'implémente pas la notion de groupe. Celle-ci peut néanmoins être émulée par l'utilisateur en ajoutant une en-tête spéciale à tous les messages envoyés (cette en-tête correspond au nom du groupe).

La communication

Toutes le bibliothèques de cette étude proposent des envois et des réceptions synchrones et asynchrones. Le modèle de communication par RPC [Birrel & al. 84] n'est défini dans aucun des langages. MPI, P4 et PVM utilisent la technique du passage de messages alors que Linda s'appuie sur un espace de tuple.

Linda est basée sur une version dérivée de la communication générale appelée espace de tuple. Un espace de tuple peut être vu comme un container ou sont placés des n-uplets. Chaque élément est de type simple et sa valeur est passée sans aucun échange de pointeur. L'adjonction d'un n-uplet à l'espace de tuple se fait par trois commandes *out*, *outd*, *eval*. La récupération d'un message s'effectue par l'opérateur *match* sur le message. Certains éléments du tuple voulant être récupérés sont précédés d'un «*?*». Le (ou les) éléments sur lequel (lesquels) la fonction de correspondance est effectuée sont clairement déclarés. La réception d'un message se fait par les trois commandes *in*, *ind* et *rd*. Les deux premières ont pour effet de supprimer le n-uplet de l'espace de tuple. La dernière laisse le n-uplet à l'intérieur de l'espace de tuple. Un gestionnaire est associé à chaque machine faisant partie de la machine virtuelle. Un espace de tuple est associé à chaque gestionnaire. Ces derniers coopèrent entre eux par échanges de messages afin de garantir l'intégrité et la cohérence de tous les espaces de tuple. L'algorithme associé à cette cohérence n'est présenté dans aucune des publications trouvées et consultées. Ceci est dû probablement au caractère commercial de Linda.

Dans PVM, les communications entre processus sont réalisées en mode point à point direct (ie. sans passer par le démon) ou en utilisant la redirection et le routage des messages par l'intermédiaire du démon. C'est notamment le cas pour une diffusion générale (*broadcast*) ou sélective (multicast). PVM oblige l'utilisateur à utiliser des tampons de communication, dont la gestion n'est pas toujours aisée. La version 3.3, résoud en partie ce problème en autorisant l'émetteur à ne plus utiliser de tampons d'émission (*In Place Packing*).

Dans MPI, les communications sont réalisées en mode point à point ou vers un groupe. Les envois et les réceptions de messages, comme dans PVM, peuvent être bloquants ou non bloquants. Par contre, à la différence de PVM, MPI :

- dispose de trois modes d'envoi de messages : prêt, standard, synchrone. En mode prêt, l'envoi ne se fait que si le destinataire est bloqué en réception. En mode standard, l'envoi est réalisé quelque soit l'état du destinataire. Enfin, en mode synchrone, l'appelant est bloqué, tant que le destinataire n'est pas en attente de message.
- n'assure pas le contrôle de flux et c'est à l'utilisateur de s'assurer que le processus destinataire est prêt à recevoir. On évite ainsi un stockage intermédiaire des messages tout en augmentant les performances.

Dans P4, les connexions sont point-à-point comme dans PVM et la diffusion générale n'est ni plus ni moins que la création de n liens de communication avec les n processus du groupe.

La synchronisation

La synchronisation relève du domaine de l'échange de connaissance. Pour réaliser cet échange, on a recourt à trois paradigmes de synchronisation [Beauquier & al. 93] : la sémaphore, le moniteur et le rendez-vous.

PVM et MPI implémentent le rendez-vous alors que P4 et Linda n'offrent aucune primitive de synchronisation. P4 implémente néanmoins, pour des machines parallèles à mémoire partagée, des moniteurs et des sémaphores. Dans le cas le plus général, la synchronisation reste tout de même possible en utilisant des primitives d'envoi de messages synchrones.

L'hétérogénéité des données

Rendre l'accès aux données distantes transparent implique, sur des machines hétérogènes, une gestion fine des différences de représentation. En effet, chaque gamme d'ordinateurs possède un alignement de données spécifique. Toutes les bibliothèques de cette étude utilisent le protocole XDR (*eXternal Data Representation*) [Sun 87]. Ce protocole décrit l'ensemble des données selon un format intermédiaire, interprétable par toutes les machines. Toutes les données sont codées lors de l'envoi et décodées à la réception.

Le principal défaut de ce protocole résulte du fait qu'il encode systématiquement les données. Or deux machines possédant un alignement de données similaires pourrait se passer de l'encodage. Un travail inutile et coûteux en temps est donc effectué. Pour palier à cet inconvénient MPI, PVM et P4 fournissent à l'utilisateur des primitives outrepassant cette gestion :

- dans PVM, c'est lors de l'écriture des données dans le tampon que la demande explicite de non encodage est effectuée.
- dans MPI et P4 on utilise des fonctions spécifiques.

Néanmoins, une gestion dynamique et transparente de ce mécanisme serait souhaitable, ce qui déchargerait le programmeur. Cette dernière n'est pas possible avec PVM, car un tampon d'émission ou de réception n'est pas attaché à un lien de communication particulier. N'importe quel tampon peut être envoyé sur n'importe quel lien de communication. Par contre, avec MPI et P4, comme les envois sont statiques, cette gestion pourrait être effectuée. Linda quant à lui est dans l'impossibilité d'outrepasser cette gestion. En effet, tous les messages sont susceptibles d'être lus par n'importe quel processus sur n'importe quelle machine. Tout envoi est anonyme. Linda est donc dans l'incapacité de connaître la machine destinatrice et donc d'adapter le codage correspondant. De plus, même si la machine virtuelle est basée sur un ensemble de machines de même type, à tout moment une nouvelle machine d'un autre type peut s'adjoindre au réseau. C'est pour ces raisons que l'encodage est obligatoire dans Linda.

Les protocoles supportés

Toutes les bibliothèques disposent au moins du protocole IP (*Internet Protocol*), ce qui permet une interopérabilité au niveau réseau. Sur IP deux protocoles de transports sont possibles, soit TCP (mode connecté avec contrôle de bout en bout), soit UDP (mode non connecté non fiable).

P4 et PVM utilisent une connexion dynamique sur TCP, c'est-à-dire que la connexion est ouverte lors de la première demande d'envoi de messages et non au lancement du processus. Avec cette technique, on évite d'ouvrir de multiples connexions entraînant un gâchis de ressources systèmes et une baisse des performances. Le type de connexion étant point à point, la diffusion vers un groupe se réalise en envoyant n fois le même message. Se pose alors le problème de l'ordonnancement de ces messages. PVM utilise alors une connexion UDP et le routage dynamique des messages est réalisé par le démon.

La connexion dans Linda est basée sur UDP. En fait, cette connexion est réalisée pour résoudre le problème de cohérence des données entre les gestionnaires de cohérences. L'avantage immédiat de l'utilisation d'UDP est d'éviter, pour des communications de faible taille et occasionnelles, l'établissement d'une connexion TCP.

Performances

En ce qui concerne les performances, les conclusions de notre étude sont les suivants :

- la gestion des tampons de communication est primordiale pour obtenir de bonnes performances ;
- plus les schémas de communication (ie la topologie) deviennent compliqués, plus les différences de performance entre environnement disparaissent ;
- les langages utilisant des schémas de communication synchrones deviennent beaucoup moins efficaces avec le nombre de stations ;
- par rapport à l'environnement de programmation, les communications à travers un réseau local, même dans le cas de FDDI, restent lent par rapport aux temps de calcul. La recherche du bon grain est donc primordiale pour ce genre d'applications.

L'augmentation des performances n'est en général pas linéaire avec le nombre de machines utilisées. Ainsi, l'utilisation de PVM entraîne un surcoût dû aux processus démons et à la gestion de la machine virtuelle. Pour remédier à cela, on a recourt à des extensions pour optimiser les communications sur des réseaux hauts débits (comme ATM [Zhou & al. 95a] ou FDDI [Lewis & al. 93]), offrir la migration transparente de processus (comme dans MPVM [Casas & al. 95]) et la gestion des processus légers (comme TPVM [Ferrari & al. 94] ou LPVM [Zhou & al. 95b]).

Les implémentations spécifiques de MPI sur différentes machines parallèles semblent plus efficaces que celles de PVM. Les différences tendent néanmoins à s'amenuiser [Casanova & al. 95]. Notons que PVM reste plus performant que MPI sur des réseaux de machines hétérogènes [Dongarra & al. 95].

Quant à Linda, il dispose avec Piranha [Bjornson & al. 91] d'un ensemble de fonctions de répartitions de charge, qui distribuent automatiquement le travail sur des machines ayant une charge de travail nulle (ou inférieure à un seuil).

Conclusion

Ces systèmes ne sont pas toujours conçus pour répondre aux demandes d'utilisateurs néophytes, dont le principal souci reste l'aide à la création de programmes client-serveur répartis et performants.

Les langages supportés par les bibliothèques de communication sont variés et on y trouve toujours le C, le C++ et le Fortran. L'utilisateur n'a donc pas à réapprendre un langage de programmation. PVM adjoint au langage de base 35 fonctions, P4 quant à lui en adjoint 50, Linda en adjoint 6 et MPI 125 (dont 24 différentes manières d'envoyer un message). Offrir trop de primitives est souvent un frein au succès d'un langage, car le temps d'apprentissage est long et l'impact sur les performances du choix de telle ou telle primitive non négligeable. A l'opposé, l'utilisateur attend d'une bibliothèques des fonctionnalités suffisantes pour éviter, à chaque fois, de réinventer la roue (ce qui d'ailleurs réduirait son caractère portable).

Le développement d'applications client-serveur demande des outils puissants pour la programmation, le test, le déboggage et la mise au point. PVM est une environnement riche en extensions. Il est notamment complété par HeNCE [Dongarra & al. 94], un environnement puissant de spécification et de contrôle de programmes intégrant l'interface graphique XPVM [Kohl & al. 94]. Linda, quand à lui, est livré avec un débogueur convivial et performant, ce qui rend ce système attrayant.

Finalement, ces systèmes restent surtout utilisés pour leurs capacités de portabilité et leur disponibilité sur un nombre toujours plus grand de plate-formes matérielles et logicielles. Ils gèrent un réseau hétérogène de machines comme une seule machine parallèle virtuelle, ce qui plaît au programmeur. Malheureusement, en masquant l'hétérogénéité, on

rend plus difficile l'obtention d'un programme performant, car le nombre de paramètres à ajuster est considérablement réduit. Si PVM reste un standard de fait, MPI est un standard réfléchi, qui évolue vers le futur MPI 2. Notons tout de même qu'il existe une version de MPI au dessus de PVM, ce qui fait bénéficier les utilisateurs de MPI des outils tiers développés autour de PVM.

Références

[Beauquier & al. 93] J. Beauquier & B. Bérard, «Systèmes d'exploitation : concepts et algorithmes», Ediscience International Eds., 541 pages, 1993.

[Bjornson & al. 91] R. Bjornson, N. Carriero, T. Mattson & A. Sherman, «Parallel Applications Using Linda on Networks of Workstation», Presented at the Sun Users Group Meeting, San Jose, 1991.

[Butler & al. 94] R. Butler & E. Lusk, «Monitors, Messages and Cluster : The P4 Parallel Programming System», Parallel Computing, Vol. 20, 1994

[Carriero & al. 94] N. Carriero, D. Gelernter, G. Mattson & A. Sherman, «The Linda alternative to message-passing systems», Parallel computing, Volume 20, 1994, Internet : ftp://ariadne.csi.forth.gr

[Casanova & al. 95] H. Casanova, J. Dongarra & W. Jiang, «The performance of PVM on MPP Systems», Technical Report CS-95-301, University of Tennessee, August 1995.

[Casas & al. 95] J. Casa, D. Clark, R. Konuru, S. Otto, & al., «MPVM: A migration transparent version of PVM», Technical Report CSE-95-002, Oregon Graduate Institute of Science and Technology, February 1995.

[Corbel & al. 95] A. Corbel & F. Fleter, «LINDA : un Modèle de Programmation Parallèle», Calculateurs Parallèles, Hermès Eds., Vol. 7 (2), pp. 159-172, 1995.

[Dongarra & al. 94] J. Dongarra, G. Geist & A. Beguelin, «HeNCE 2.0 User's Guide», Site : <http://www.netlib.org>.

[Dongarra & al. 95] J. Dongarra & T Dunigan, «Message Passing Performance on Various Computers», Technical Report CS-95-299, University of Tennessee, July 1995.

[Ferrari & al. 94] A. ferrari & V.S. Sunderam, «TPVM: Distributed concurent computing with lighweight processes», Dept. of Mathematics and Computer Science, Emory University, 1994.

[Geist & al. 93] A. Geist, J. Dongarra & R. Manchek, «PVM3 User's Guide and Reference Manual», Oak Ridge National Laboratory and University of Tenessee, May 1993.

[Gropp & al. 94] W. Gropp, E. Lusk & A. Skjellum, «Using MPI: Portable Parallel Programming with the Message Passing Interface», MIT Press, 1994.

[Hoare 78] C. Hoare, «Communicating Sequential Processes», Communications of the ACM, Vol. 21 (8), August 1978.

[Kohl & al. 94] J.A. Kohl & G.A. Geist, «XPVM: A Graphical Console And Monitor For PVM», In 2nd PVM User's Group Meeting, Oak Ridge, TN, May 1994.

[Lewis & al. 93] M. Lewis & R. Cline «PVM Communication performance in a switched FDDI heterogeneous distributed computing environment», In Proc. of the IEEE Workshop on Advances in Parallel and Distributed Systems, October 1993.

[MPIF 93] MPI Forum, «Document for a Standard Message Passing Interface», Technical Report CS-93-214, University of Tennessee, Site : <http://www.mcs.anl.gov/mpi/mpi-report>, November 1993.

[MPIF 94] MPI Forum, «MPI: A Message Passing Interface Standard», Technical Report CS-94-230, University of Tennessee, April 1994. Est paru dans International Journal of Supercomputer Applications, Vol 8 (3/4), 1994.

[SUN 87] Sun Microsystems Inc., «XDR: External Data Representation», Technical Report RFC 1014, June 1987.

[Zhou & al. 95a] H. Zhou & A. Geist, «Fast Message Passing in PVM», 9th Intel Parallel Processing Symposium: Workshop on High Speed Network Computing, Santa Barbara, April 1995.

[Zhou & al. 95b] H. Zhou & A. Geist, «LPVM: A step towards Multithreaded PVM», Oak Ridge National Laboratory, 1995.