Analyse de schémas de communications MPI pour le placement d'applications sur architectures parallèles

Sujet de Mémoire de Master 2

1er décembre 2011

Mots-clés : schéma de communication, analyse statique, placement, calcul haute-performance, parallélisme
Lieu de thèse : Équipe Satanas, LaBRI, Bordeaux.
Encadrant Denis BARTHOU et Guillaume MERCIER.
Contact : 05 24 57 41 16 ; denis.barthou@labri.fr, guillaume.mercier@labri.fr

1 Contexte scientifique

Les besoins des applications scientifiques en termes de puissance de calcul croissent sans cesse. Les machines qui exécutent ces applications se doivent donc d'être de plus en plus puissantes. Actuellement, seules les machines possédant une architecture parallèles sont à même de répondre à ces besoins. Cependant, la mise au point de programmes à la fois performants (c'est-à-dire exploitant l'architecture matérielle sous-jacente au mieux) et portables (c'est-à-dire pouvant être portés facilement d'une architecture parallèle à une autre) est un défi majeur dans ce domaine.
Ces applications s'appuient sur des outils de programmation, qui sont souvent également des standards. Dans le domaine du calcul parallèle, le standard *Message Passing Interface* s'est imposé depuis plus de quinze ans avec un beau succès. Un nombre considérable d'applications parallèles ont été écrites à l'aide de cette interface, dont les implémentations ont été très optimisées, notamment pôur supporter les nouvelles générations d'architectures matérielles basées sur des machines multicoeur.
Ces architectures sont désormais très hiérarchiques et hétérogènes, avec notamment de multiples niveaux de caches qui ne sont pas toujours partagés entre tous les processeurs de la machine. Egalement, les temps accès à la mémoire dépendent de la localisation physique du coeur vis-à-vis du banc mémoire considéré. Tous ces aspects impactent fortement les performances et ne peuvent être résolus au niveau des implémentations MPI proprement dites, même si celles-ci peuvent fournir des mécanismes permettant de prendre en compte une partie des phénomènes.
Il devient nécessaire d'adapter l'exécution d'une application parallèle écrite en MPI à l'architecture sous-jacente. Ainsi, les applications vont gagner en performance et vont pouvoir passer plus facilement à une échelle importante (contexte exascale). L'idée de base consiste à placer les différents processus de l'application sur les différents coeurs de la machine afin d'accélérer au mieux les communications MPI [1]. Cette technique nécessite néanmoins de posséder des informations à la fois sur l'application et sur l'architecture-cible. En ce qui concerne ce deuxième point, nous avons développé au sein de l'équipe Satanas du LaBRI et en particulier de l'EPI INRIA Runtime la bibliothèque HWLOC [2] qui permet d'obtenir de façon portable la topologie d'un environnement cible. Mais en ce qui concerne les informations de l'application, il est actuellement indispensable d'effectuer une première exécution de l'application afin de générer son schéma de communication. Ce schéma est ensuite confronté aux informations de topologie physique afin de redistribuer les processus MPI sur la machine [3].

2 Sujet

L'objectif de ce mémoire est de pouvoir trouver les schémas de communication des applications MPI sans avoir à passer par une exécution préalable. En particulier, nous aimerions nous intéresser à l'analyse des codes MPI écrits en C ou en Fortran et être capable de générer au moment de la compilation les informations dont nous avons besoin pour placer optimalement les processus sur l'architecture au moment de l'exécution proprement dite.

Cette analyse des communications est similaire à une analyse des dépendances sur des tableaux, les envois de message correspondant à des écritures de messages, les receptions correspondant à des lectures de ces messages. Quelques travaux de recherche ont utilisé ces techniques d'analyse sur des codes MPI ([4, 5] ou certaines analyses du compilateur ROSE par exemple), essentiellement à des fins de debuggage. On s'appuiera sur ces travaux pour proposer une analyse plus fine des communications de structures et de tableaux. On utilisera pour ce faire l'analyse de dépendance polyhédrique[6] pour proposer une analyse des schémas de communication MPI permettant d'améliorer le placement des processus sur les coeurs.

Le travail consistera donc dans un premier temps à étudier les articles de la bibliographie sur ce sujet, concernant les méthodes d'analyse de dépendances, en particulier pour programmes MPI. Puis on adaptera les méthodes d'analyse de dépendance polyhédriques afin de déterminer le graphe des communications (en fonction des paramètres d'entrée du programme) et les volumes de communications entre chaque noeud. Cela permettra d'utiliser les algorithmes optimisant les placements des processus MPI pour minimiser les communications coûteuses en performance.

Enfin, dans un second temps, on pourra réaliser et tester cette analyse dans le compilateur LLVM[7], disposant déjà de certaines analyses de dépendances polyhédriques.

Références

[1] Guillaume Mercier and Jérôme Clet-Ortega. Towards an efficient process placement policy for mpi applications in multicore environments. In *EuroPVM/MPI*, volume 5759 of *Lecture Notes in Computer Science*, pages 104–115, Espoo, Finland, September 2009. Springer.

[2] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications. In *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*, Pisa, Italia, February 2010. IEEE Computer Society Press.

[3] Guillaume Mercier and Emmanuel Jeannot. Improving MPI Applications Performance on Multicore Clusters with Rank Reordering. In *EuroMPI*, volume 6960 of *Lecture Notes in Computer Science*, pages 39–49, Santorini, Greece, September 2011. Springer.

[4] M.M. Strout, B. Kreaseck, and P.D. Hovland. Data-flow analysis for mpi programs. In *Parallel Processing, 2006. ICPP 2006. International Conference on*, pages 175 –184, aug. 2006.

[5] Greg Bronevetsky. Communication-sensitive static dataflow for parallel message passing applications. In *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '09, pages 1–12, Washington, DC, USA, 2009. IEEE Computer Society.

[6] Paul Feautrier. Array dataflow analysis. In Santosh Pande and Dharma Agrawal, editors, *Compiler Optimizations for Scalable Parallel Systems*, volume 1808 of *Lecture Notes in Computer Science*, pages 173–219. Springer Berlin / Heidelberg, 2001.

[7] Chris Lattner. Llvm and clang : Next generation compiler technology. Keynote at the BSD Conference, May 2008.