Hiérarchie des données en parallélisme hybride

L'avènement des processeurs multi-coeurs a provoqué une évolution des supercalculateurs vers des noeuds de calcul comprenant un grand nombre de coeurs. Il devient alors nécessaire d'adapter la manière d'exploiter de telles grappes en utilisant une programmation parallèle hybride multithread.
Une approche permettant une évolution progressive des applications est l'utilisation d'implémentations MPI (Message Passing Interface) à base de threads (thread-based MPI) en conjonction avec un support exécutif OpenMP. Néanmoins, l'intégration de ces deux approches nécessite la mise en place d'un contrôle fin du placement des données et de leur visibilité, chaque modèle proposant une construction différente pour déclarer des variables privées. Cet article propose d'étendre le mécanisme TLS (Thread-Local Storage) pour gérer le placement des données privées et ainsi permettre la programmation hybride thread-based MPI + OpenMP.

P. Carribault, M. Pérache, H. Jourdren CEA - DAM Île-de-France

Modèles de programmation à base de threads

L'évolution des architectures des supercalculateurs incite actuellement à explorer la programmation parallèle hybride à base de threads, qui consiste à utiliser plusieurs flots d'exécution au sein d'un processus en partageant le même espace mémoire. Dans ce contexte, le CEA/DAM développe la plateforme *Open Source MPC* [1] qui fournit un support exécutif de type *thread-based MPI* (modèle MPI [2] utilisant des threads à la place de processus système) et une implémentation du modèle OpenMP [3]. MPC permet ainsi une évolution progressive des codes de calcul, du modèle MPI pur vers un modèle hybride à base de threads.

Dans un tel modèle de programmation multithread à plusieurs niveaux, le placement ainsi que la visibilité des données sont cruciaux. En effet, dans les approches *thread-based MPI*, il est nécessaire de pouvoir associer des données à chaque tâche MPI qui est aussi un thread. En ce qui concerne le modèle OpenMP, la problématique est la même : il faut pouvoir associer une donnée à chaque thread. De plus, les modèles de programmation étant imbriqués, les données associées à chaque tâche MPI doivent êtres partagées entre les threads OpenMP crées à partir de cette tâche MPI. On observe alors une hiérarchie de données plaquée sur l'imbrication des modèles de programmation.

Or, les méthodes existantes de stockage privé par thread (*Thread-Local Storage* ou TLS) n'autorisent pas cette hiérarchie de données. Cet article présente donc une extension du mécanisme de TLS pour la programmation hybride *thread-based* MPI + OpenMP. Ce nouveau mécanisme est appelé Extended TLS [4].

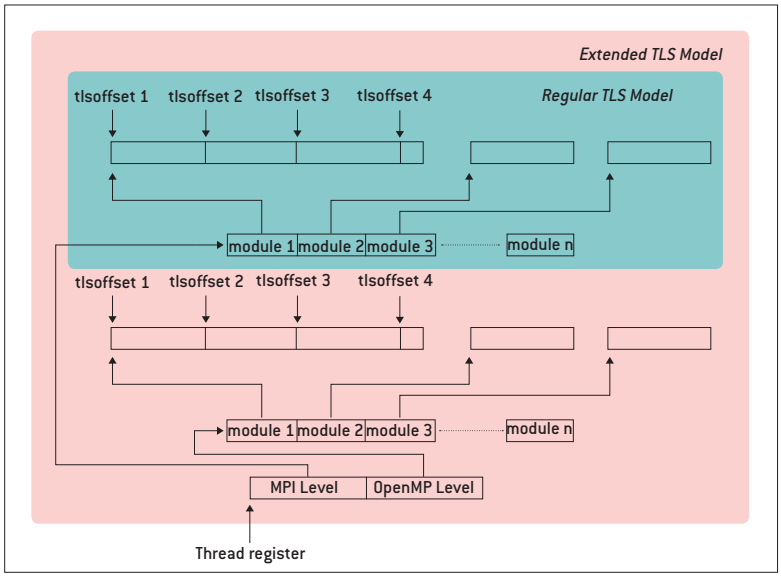

Figure 1. Modèles TLS général et étendu.

Tableau 1. Statistiques sur des benchmarks NAS Multi-Zone.

Statistiques	BT (Block Tridiagonal Solver)	LU (Lower-Upper Symmetric Gauss-Seidel Solver)	SP (Scalar Pentadiagonal Solver)
Langage	Fortran	Fortran	Fortran
Lignes de code	5154	4618	5085
Nombre de variables globales	173	175	132
Accès statiques aux variables globales	258	274	273
Nombre de variables privées OpenMP	11	11	5
Accès aux variables privées OpenMP	44	50	58

Extension du support des variables privées par thread

Le mécanisme de TLS permet le stockage de données privées par thread, mais il ne dispose que d'un seul niveau. Il ne permet pas d'intégrer une notion sémantique liée au modèle de programmation. Le stockage des données dans les segments TLS se fait grâce à trois niveaux d'indirection (figure 1). Cette dernière fait appel à un mode d'adressage indirect, dans lequel l'adresse d'un élément est fournie par lecture d'un autre élément. Le premier niveau est gardé par un registre spécifique, mis à jour à chaque changement de contexte entre threads. Il permet d'indexer la table des variables privées. Le second niveau permet d'identifier la bibliothèque qui a déclaré la variable. Le troisième niveau est un décalage (*offset*) dans la structure de données contenant toutes les variables privées.L'extension des TLS que nous proposons ajoute un niveau sémantique avant le choix de la bibliothèque (figure 1). Précisément, le premier niveau concerne toujours un registre tandis que le second niveau représente le modèle de programmation. Avec une gestion adaptée de la création des threads et du changement de contexte, il est alors possible de plaquer la hiérarchie des données privées sur l'imbrication des modèles de programmation.

Expérimentations

L'implémentation du mécanisme *Extended TLS* nécessite tout d'abord une modification de la chaîne de compilation. En effet, la déclaration du niveau sémantique de chaque variable se fait par mot-clé.

De plus, il est nécessaire de générer un code spécifique pour chaque accès à ces variables afin de calculer l'adresse mémoire de la variable en fonction du thread, du niveau sémantique, de la bibliothèque et du décalage. Cette opération est réalisée grâce à un appel de fonction ayant pour paramètre le numéro de la bibliothèque et le décalage.

Nous avons alors implémenté ce mécanisme dans la plateforme MPC comprenant un compilateur GNU GCC « patché » [5]. Cette implémentation a été validée par l'exécution de la suite de benchmarks NAS comprenant des mini-applications de simulation numérique parallélisées en MPI, OpenMP et MPI/OpenMP (NAS Multi-Zone).

Le tableau 1 illustre l'ampleur des modifications effectuées automatiquement par notre mécanisme dans MPC. Ces applications MPI/OpenMP Fortran comportent un nombre important de variables globales par rapport aux nombre de lignes de code. Ces variables ont alors été automatiquement placées au niveau sémantique MPI du mécanisme *Extended TLS*. Enfin, ces applications exploitent également plusieurs variables privées à chaque thread OpenMP : ces variables ont été mises au niveau sémantique OpenMP. Grâce à ces modifications, tous les accès à ces variables sont valides, respectant ainsi la sémantique parallèle du programme.

Perspectives

La mise en place d'un mécanisme étendu pour la gestion des données dans des modèles de programmation à base de threads permet la migration d'applications parallèles, depuis un modèle purement MPI vers un modèle plus souple mêlant une approche *thread-based MPI* avec des threads OpenMP. Avec les évolutions actuelles des architectures de processeurs, ce mécanisme, nommé *Extended TLS* [4], est alors une étape importante vers l'exploitation des machines de classe exaflopique.

Au niveau des perspectives, d'autres codes de calcul et les nouvelles architectures *manycore* devront être évaluées. Enfin, les modifications apportées au compilateur et au support exécutif MPC empêchent certaines optimisations statiques améliorant l'accès aux variables privées : il nous faudra porter ces optimisations (remplacement d'un appel de fonction par des mouvements de registres) pour atténuer le surcoût, qui peut actuellement s'élever à 15% selon les applications.

RÉFÉRENCES

[1] **P. CARRIBAULT, M. PERACHE, H. JOURDREN,** "Enabling Low-Overhead Hybrid MPI/OpenMP Parallelism with MPC", *Proc. International Workshop on OpenMP (IWOMP'2010)*, Tsukuba, Japan (2010).

[2] Message Passing Interface Forum, "MPI: A Message Passing Interface Standard", http://mpiforum.org (2008).

[3] **ARCHITECTURE BOARD**, "OpenMP version 3.0", http://www.openmp.org (2008).

[4] **P. CARRIBAULT, M. PERACHE, H. JOURDREN,** "Thread-Local Storage Extension to Support Thread-Based MPI/OpenMP Applications", *Proc. International Workshop on OpenMP (IWOMP'2011)*, Chicago, USA (2011).

[5] **GNU**, "GCC: GNU Compiler Collection", http://gcc.gnu.org (2011).