C2S@Exa
Computer and Computational Sciences at Exascale

Ph.D project proposal

Conception d'un modèle de composants logiciels avec ordonnancement de tâche pour architecture parallèle à base de many-cœurs, application au code Gysela5D

Conception of a software component model with task scheduling for many-core based parallel architecture, application to the Gysela5D code

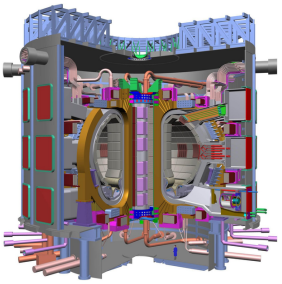
---

AVALON team	Inria Grenoble – Rhône-Alpes research center
MAISON DE LA SIMULATION	CEA Saclay
RUNTIME team	Inria Bordeaux – Sud-Ouest research center
IRFM	CEA Cadarache

Contacts: Christian Pérez and Julien Bigot and Olivier Aumage and Guillaume Latu
E-mails: christian.perez@inria.fr / julien.bigot@cea.fr / olivier.aumage@inria.fr / guillaume.latu@cea.fr

Keywords: Large-scale parallelism, Software components, Dynamic task scheduling, Hardware accelerators

1 Contexte



Coupe d'un Tokamak: réacteur pour la fusion

Un besoin majeur pour maîtriser la fusion nucléaire dans des réacteurs comme ITER est de comprendre les mécanismes qui influent sur le confinement de la chaleur au sein du plasma où a lieu la fusion. **Gysela5D**[1] [8] est un code utilisé pour simuler les instabilités qui se développent dans le plasma et ont un grand impact sur ce confinement. L'exécution de telles simulations nécessite des ressources de calcul particulièrement importantes qui ne peuvent être obtenues qu'en tirant parti du parallélisme massif des plus grands supercalculateurs disponibles. Une exécution récente sur la machine Juqueen a permis d'atteindre 91% d'efficacité relative avec 1 835 008 threads en parallèle.

L'architecture de processeur/accélérateur Intel MIC (Xeon PHI) a été spécifiquement conçue pour ces machines massivement parallèles et est en passe d'être adoptée dans un nombre croissant de supercalculateurs. Un avantage important de cette architecture pour un code comme GYSELA5D est la bande passante mémoire élevée qu'elle offre, permettant apriori d'améliorer les performances des cœurs de calculs "memory-bound" du code.

Malgré la possibilité d'utiliser les modèles de parallélisation classiques, le portage de codes comme GYSELA5D sur MIC reste complexe. La faible quantité de mémoire implique de favoriser l'utilisation de threads plutôt que de processus lourds pour favoriser le partage de données. Il est aussi nécessaire de porter une attention particulière à l'écriture des boucles pour permettre au compilateur de tirer le meilleur parti de l'unité de calcul vectoriel du processeur. Il peut

[1] **5D gy**rokinetic **se**mi-**la**grangian code: http://gyseladoc.gforge.inria.fr/

aussi être nécessaire d'optimiser l'organisation en mémoire de manière spécifique. Ces divergences dans le code posent un problème d'autant plus important que pour profiter au mieux des supercalculateurs, il est nécessaire d'utiliser au sein d'une même application à la fois les processeurs multi-cœurs (ex. Xeon classique) et many-cœurs (ex. Xeon PHI) présents sur la machine.

Les modèles de composants logiciels ont été conçus pour permettre de faciliter la modularisation de code. Grâce à ces modèles il est possible de séparer les différents aspects d'une application, et de séparer par exemple un ensemble de composants mettant en œuvre la logique applicative et un autre gérant la parallélisation de l'application. Le modèle de composant **L²C**[2] [3] est particulièrement intéressant puisqu'il permet de décrire l'architecture de l'application parallèle complète tout en imposant des sur-coûts minimaux à l'exécution. Grâce à ce type d'approche, on peut envisager d'offrir des mises en œuvres de composant différentes pour many-cœurs et pour multi-cœurs.

Un autre approche intéressante pour ce type d'architecture est offerte par les moteurs d'ordonnancement et d'exécution dynamiques de tâches. Traditionnellement, les modèles de programmation parallèles demandent de la part du programmeur l'organisation de son application en fils d'exécution distincts (appelés threads), en nombre potentiellement variable entre le début et la fin de l'application. Or le choix du nombre de threads et du travail que chacun d'eux réalise dépend notamment des caractéristiques de la plateforme utilisée pour l'exécution.

Une approche différente a été initiée par les travaux autour du langage Cilk [6], puis reprise par des outils tels que Intel TBB et plus récemment introduite dans le langage OpenMP [2, 9]. Cette approche propose qu'au lieu de définir des fils d'exécution, le programmeur définisse plutôt des **tâches** élémentaires au sein de son application ainsi que les **relations de dépendance** existant entre ces tâches [4]. Un ordonnanceur se charge ensuite de la répartition du travail représenté par les tâches. **StarPU**[3] [1, 7] est un ordonnanceur de tâches. Il est responsable de choisir dynamiquement sur quel processeur exécuter chaque tâche en fonction d'un modèle de coûts prenant en compte des éléments comme le temps d'exécution et le temps de transfert des données. En transférant la responsabilité de la répartition du travail depuis l'application vers l'ordonnanceur, les modèles de programmation parallèles à base de tâches permettent de préserver la portabilité de l'application.

2 Description du sujet de thèse

Le sujet de thèse consiste à définir et à valider un modèle de programmation intégrant la description d'architectures logicielles et un ordonnancement dynamique de tâches [5]. Par exemple, il s'agit de combiner les avantages de modèles tels que L²C et StarPU. L'objectif final est de proposer un modèle capable de supporter des applications telles que Gysela5D sur les architectures parallèles actuelles et futures. Ainsi, il est attendu de valider d'abord le modèle sur des applications synthétiques, et des versions simplifiées de Gysela5D sur un nœud de calcul parallèle (multi-cœur, many-cœurs, etc.) avant de viser des architectures à large échelle des supercalculateurs. Pour ces validations, le doctorant aura accès à des clusters très variés, ainsi qu'à des supercalculateurs comportant des accélérateurs.

Ce sujet nécessite d'intégrer des connaissances réparties dans plusieurs équipes (modèles à composants, ordonnancement de graphes de tâches, Gysela5D, parallélisation de grands codes, etc.). Il demandera donc au candidat une grande autonomie et de fortes interactions avec les quatre sites impliquées. De ce fait, la thèse devrait se dérouler sur les sites de Saclay puis de Lyon et de nombreux déplacement sur les autres sites sont à prévoir.

[2] Low Level Components: http://llcmcpp.gforge.inria.fr/
[3] http://runtime.bordeaux.inria.fr/StarPU

3 English Description of the Thesis Subject

The goal of the thesis is to define and validate a programing model that combines the description of software architecture with dynamic task scheduling [5]. For example by integrating the advantages of the L²C and StarPU models. The final goal is to propose a model that enables the description of applications such as Gysela5D on current and future parallel architectures. Hence, the candidate will first have to validate the model on synthetic and simplified use-cases from Gysela5D on a single parallel node (multi-core, many-core, etc.) before targeting large scale architectures of supercomputers. For these validations, the candidate will have access to various clusters and supercomputers including accelerators.

The subject requires the candidate to integrate knowledge from multiple teams (software component models, workflow scheduling, Gysela5D, large-scale parallelization, etc.) This requires a high capacity for autonomy and interactions with the four partners. For this reason, the thesis will most likely take place in Saclay and then Lyon with multiple visits to the other sites.

References

[1] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, 23:187–198, February 2011.

[2] Eduard Ayguade, Nawal Copty, Alejandro Duranl, Jay Hoeflinger, Yuan Lin, Federico Massaioli, Ernesto Su, Priya Unnikrishnan, and Guansong Zhang. A proposal for task parallelism in OpenMP. In *Third International Workshop on OpenMP (IWOMP 2007)*, Beijing, China, 2007.

[3] Julien Bigot, Zhengxiong Hou, Christian Pérez, and Vincent Pichon. A Low Level Component Model Enabling Performance Portability of HPC Applications. In *5th International Workshop on Multi-Core Computing Systems*, Salt Lake City, United States, Nov. 2012.

[4] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Herault, Pierre Lemarinier, and Jack Dongarra. Dague: A generic distributed dag engine for high performance computing. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 1151–1158, Washington, DC, USA, 2011. IEEE Computer Society.

[5] Hinde Bouziane. *De l'abstraction des modèles de composants logiciels pour la programmation d'applications scientifiques distribuées.* Thèse de doctorat, Université de Rennes 1, IRISA/INRIA, Rennes, France, February 2008.

[6] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI '98, pages 212–223, New York, NY, USA, 1998. ACM.

[7] Andra Hugo, Abdou Guermouche, Raymond Namyst, and Pierre-André Wacrenier. Composing multiple StarPU applications over heterogeneous machines: a supervised approach. In *Third International Workshop on Accelerators and Hybrid Exascale Systems*, Boston, USA, May 2013.

[8] G. Latu, N. Crouseilles, V. Grandgirard, and E. Sonnendrücker. Gyrokinetic Semi-lagrangian Parallel Simulation Using a Hybrid OpenMP/MPI Programming. In Franck Cappello, Thomas Herault, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 4757 of *Lecture Notes in Computer Science*, pages 356–364. Springer Berlin Heidelberg, 2007.

[9] Judit Planas, Rosa M. Badia, Eduard Ayguadé, and Jesus Labarta. Hierarchical task-based programming with starss. *Int. J. High Perform. Comput. Appl.*, 23(3):284–299, August 2009.