Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document constitue une fraude caractérisée.
Nom, date et signature :

Insert your title here

Steven QUINITO MASNADA
·
Supervised by : Arnaud LEGRAND ·
Luka STANISIC

June 2015

Résumé Dans le domaine des supercalculateurs, la course à la performance est un point crucial. Actuellement, le calculateur le plus puissant (le TianHe-2) est capable d'effectuer environ 33.86 Peta d'opérations flotantes par secondes. Cependant cette course est freinée par un facteur qui prend désormais d'une importance capitale, le coût énergétique. En effet, reprennons l'exemple du supercalculateur chinois, la consommation du TianHe-2 atteint presque les 18MW et avec la génération exascale la consommation estimée sera entre 20MW et 40MW. Dans l'état des fait, ce n'est pas réalisable et pour pouvoir atteindre l'exaflops, il nécessaire d'optimiser d'autres points que la puissance des puces. Evidemment des optimisations peuvent être faites au niveau matériel afin de réaliser des composants à hautes efficacités énergétiques. On peut également optimiser le rendement en utilisant au mieux les capacités du matériel. Cette optimisation ce fait donc du côté logiciel et pour cela il nous faut envisager un changement de méthode programmation, c'est cette dernière que nous allons étudier. L'objectif de mon stage au sein de l'équipe MESCAL, sous la tutelle d'Arnaud Legrand, est donc de tenter de mesurer le gain d'une telle solution.

Pour cela nous allons, dans une première partie, voir comment est effectuée en générale la programmation en HPC, quels sont différents les standards et pourquoi nous nous sommes concentrés sur MPI. Nous discuterons ensuite du principe et de l'intérêt d'un nouveau paradigme de programmation et de la librairie StarPU. Nous constaterons ensuite que malgrès les apports de cette méthodes des difficultés subsites et les mesures peuvent-être compliquées a ef-

F. Author
first address
Tel. : +123-45-678910
Fax : +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

fectuées. C'est pourquoi dans une seconde partie, nous étudierons les différents approches pour évaluer les performances d'applications HPC et nous justifierons notre choix pour la simulation/émulation et en particulier pour l'outils Simgrid. Dans une troisième partie nous examinerons en détail Simgrid et StarPU ainsi que les différents problèmes que nous avons rencontrés. Dans une quatrième partie, nous verrons les méthodes employées. En cinquième partie, nous verrons les modifications apportés à Simgrid afin de pouvoir effectuer les mesures. Ensuite dans une sixième partie, nous verrons comment ces changements ont été validés. Et pout finir nous conclurons sur les résultats que nous avons réussit à obtenir.

1 Introduction

La majorité des supercalculateurs actuels sont des clusters massivement parallèles et souvent de type hétérogènes(CPU-GPU). De ce fait certains standard ce sont imposés.

Il y a tout d'abord la norme MPI (Message Passing Interface), qui est une API de communication basée sur l'envoi et la récéption de message. Elle réputée pour être performante et portable, et elle est de plus haut niveau que les sockets.

Ensuite, il y a l'API OpenMP qui est une interface de multihreading de plus haut niveau de PThread...

Enfin, l'API CUDA permet tirer partie de la puissance de calculer des GPU. Mais pour cela il est nécessaire de spécifier explicitement de ce que l'on veut envoyer aux GPUs et on doit également gérer la synchronisation

L'objectif étant optimiser le rendemment d'une application afin que cette dernière tire partie de toute la puissance disponible, il faut faire en sorte d'occuper au maximum le plus d'unités de calcul possible. Le problème est que l'on se retrouve à devoir utiliser plusieurs paradigme à la fois ce qui complique grandement la programmation.

Généralement, on procède soit en délèguant tout les calculs aux GPUs, laissant les CPUs en idle. Soit on réparti la charge entre les CPUs et les GPUs de manière complètement statique. L'inconvéniant est que la mise en pratique très difficile car il est hardu de trouver un bon équilibrage.

Cependant même si l'on arrive à équilibrer les charges correctemment, on peut avoir des cas où certaines unités de caluls ne sont pas occupées alors qu'elles le pourraient. Cela se produit quand par exemple lorsque certaines unités de calculs attendent la terminaisons de certain traitements alors que d'autres auraient put être effectuer en attendant. Cela est dû au fait que l'exécution soit statique et induit un idle time artificiel. De plus cette solution n'est pas portable car le découpage des traitements ce fait en fonction de la plateforme cible.

La solution serait donc d'avoir une gestion dynamique des charges. Mais cela s'avère bien plus ardue, voir impossible à réaliser directement avec ces méthodes de programmation. Alors essayons en une autre.

La librairie StarPU est un système runtime qui permet une répartition des traitements de manière dynamique et opportuniste. Pour ce faire elle introduit un nouveau paradigme basé sur les tâches. StarPU génère un graphe de dépendance permettant d'optimiser le l'ordonnancement de ces dernières.

La première version de StarPU a été conçu spécialement pour des architectures hybrides. Une version récente (StarPU MPI) a été réalisée pour bénéficier d'un ordonnance et d'une exécution qui soit à la fois dynamique et opportunistes dans un contexte distribuée afin de répartir la charge entre les différents noeuds.

Nous allons donc voir comment évaluer les performances d'applications basés sur StarPU MPI.

2 État de l'art

2.1 sparse solvers

MUMPS, QR-MUMPS, ...

2.2 Runtimes

- Dague, Quark, StarSS
- StarPU has the right level of abstraction and organization (synchronization, software architecture), is stable and has good performances

2.3 Simulation for HPC

Most tools focud on simulation of MPI applications and are based on trace replay. SimGrid provides thread-like API, which simplifies application porting.

2.4 Conclusion, we use the best of these three domains

3 Presentation of SimGrid/StarPU/Qrm_Starpu

3.1 Qrm_Starpu

How does it work. DAG, eliminition tree, main kernels (INIT, CLEAN, DO_SUBTREE, GEQRT). A priori complexity of some of the kernels known by the developpers.

3.2 StarPU/SimGrid

Emulation/simulation/... based on Europar and CCPE.

4 Coarse grain Porting qrm_starpu on top of StarPU/SimGrid :

Intro..

In order to run it with SimGrid, we had to do only few changes to the initial qrm_starpu code. Compilation process along with the necessary environment variables had to be adapted to the simulation mode of StarPU. Main `qrm_test` program used to execute factorization with qrm_starpu had to be changed for the `starpu_main` subroutine, as SimGrid has its own main function and this is the easiest way to avoid problems with having two mains.

This solution includes several major differences comparing to the already published study of StarPU and SimGrid on dense linear algebra applications [**?**]. Most notably, in qrm_starpu code there is a bigger number of different StarPU tasks (also named *kernels* in the rest of this paper) called with wide variety of input parameters. When solving dense matrices, during an application run, one kernel type (e.g., *dgemm*) is always working with the same block size, which makes its duration on the same processing unit very stable. In qrm_starpu factorization, the amount of work that has to be done by a kernel depends hugely on its input parameters and thus the duration will greatly vary. Hence, we had to rework the task submission model to make sure all kernel parameters are explicitly given to the StarPU so that they can be propagated to SimGrid. Additionally, we were required to extend tracing so that such kernel parameters are traced as well. This was indispensable for analyzing and comparing traces for both real and simulation executions.

On the other hand, so far we had worked with the qrm_starpu implementation that does not involve code running on GPUs, which facilitates the simulation. The reason is that as there are no critical need to model PCI contention and heterogeneity, thus there are less elements and approximations to take into account with SimGrid. Still, following very good results on hybrid machines with dense linear algebra applications, we believe that it would not be so difficult to extend this qrm_starpu study to the fully heterogeneous architectures.

Explain current memory allocation shortcoming..

5 Conclusion

— Previous work for dense with GPUs with different schedulers but the current version of the code is not ready yet and schedulers barely matter for CPU only executions.
— Still goes faster than real executions and does not require actual machines.
— Still allows for extrapolation, which can be useful
— Open the path to performance studies that were not possible earlier

Acknowledgments

This work is partially supported by the SONGS ANR project (11-ANR-INFRA-13). Experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from LABRI and IMB and other entities : Conseil Régional d'Aquitaine, Université de Bordeaux and CNRS.