Modelisation et simulation d'applications dynamique pour plateformes Exascale

Steven QUINITO MASNADA

Encadrants : Arnaud LEGRAND · Luka STANISIC

Juin 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 calcul 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'inconvénient 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

En HPC, il y a trois grandes approches possible pour évaluer les performances d'application.

2.1 Test sur systèmes réels

Cette approche consiste à lancer la vrai application sur le système réel afin d'effectuer les mesures. Cependant cette méthode peut se révéler très couteuse et il n'est pas toujours possible d'avoir accès à la plateforme. De plus comme les expérimentations ne peuvent être effectuées sur que sur un petit nombre de plateforme notamment à cause de coût, on ne peut pas vraiment extrapoler les résultats. Dernier point important, nous n'avons pas de contrôle sur les décisions d'ordonnancements, d'une exécution à l'autre on peut avoir des résultats différents ce qui fait que les expériences ne sont pas reproductibles.

2.2 L'approche par rejeu de trace

Cette méthode consiste à exécuter une première l'application sur un système réel pour ensuite pour ensuite rejouer la trace post-mortem. Elle est couramment employé dans le contexte d'application MPI mais est ici totalement inadaptés car nous avons à faire des programmes qui sont non déterministes. En effet, on ne pourra pas connaitre les autres actions qu'il était possible d'effectuer plutôt qu'un autre, ni leur impact.

2.3 La simulation/émulation

On a d'une part la simulation où l'on crée un faux environnement proche de la réalité et où les actions ne sont pas réellement effectués. Dans notre cas on simulerait donc la plateforme de même que l'OS. Ainsi, les expérimentations peuvent être effectuées à partir de n'importe quel système, il n'est plus

nécessaire d'avoir accès à la plateforme, ce qui rend cette approche peu coûteuse. Par ailleur il est facile d'extrapoler les résultats car on peut simuler un nombre important de plateformes. Ensuite la simulation permet d'avoir un temps d'exécution plus court qu'avec des tests réels car on n'effectue que certains traitements ce qui nous permet pouvoir effectuer un grand nombre de mesures. Enfin comme la simulation nous permettrait d'avoir un contrôle sur l'ordonnancement, nous pourrions avoir un système déterministe qui nous permettrait d'avoir des expériences qui peuvent être reproduites.

Et d'autre part l'émulation où l'on exécuterait en vrai le programme programme sur le système simulé.

Nous allons tenter de voir si nous pouvons conciller les deux approches.

Pour cela nous allons utiliser le logiciel Simgrid qui est un simulateur de systèmes distribués, de grilles de calculs, de systèmes peer to peer et cloud.

3 Analyse du problème

3.1 Simgrid : Les processus

Sous Simgrid, les processus sont modélisés par des threads, ce qui signifie que leur espace d'adressage est partagé. Afin que ces derniers ce comportent comme des processus UNIX, il est nécessaire que chaque processus n'ait pas accès aux variables d'un autre, c'est pourquoi un système de privatisation a été mis en place. L'approche est la suivante : pour chaque processus, une zone mémoire est allouée dans le tas grâce à un mmap. Cette zone est le nouveau segment données du processus, et à chaque changement de contexte, on fait pointer vers cette zone.

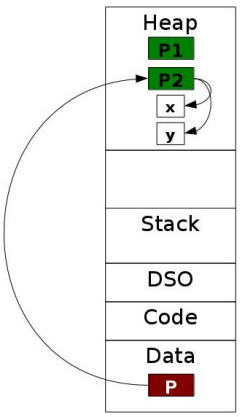

Figure 1 Privatisation du segment données

3.2 SimGrid/MPI : Architecture générale

Simgrid est composé de plusieurs modules, ceux auxquels nous nous intéressons sont les suivant :

3.2.1 MSG qui permet de simuler couche classique logiciels ? ? ?

3.2.2 SMPI

Cette API permet de simuler la couche MPI. Actuellement, la majeur partie des fonctionnalités MPI ont été implémentées. Le fonctionnement est le suivant :

- l'application que l'on veut tester est compilée en remplaçant le mpi.h classique par le mpi.h de Simgrid
- à l'édition de lien on remplace le main de l'application par celui de Simgrid.
- Ce dernier a pour rôle de préparer l'éxécution du simulateur en créant la plateforme et en déployant les processus SMPI qui exécuterons chacun le main de l'application MPI.

3.3 StarPU-SG : Architecture générale

StarPU a été modifié afin de pouvoir fonctionner au dessus du simulateur Simgrid et est basé sur l'API MSG. L'application est exécutée réellement mais les allocations mémoires des tâches ne sont pas effectuées, les codes de calcul sont simulés et remplacés par un délais de même pour les transferts CUDA.

3.4 Ce qui coince

Comme en MPI on est dans un contexte d'espace mémoire distribuée, les processus MPI d'un même noeud doivent partager les données donc il faut faire en sorte que le segment data d'un processus soit rattaché à celui qui les a crées.

De plus, une autre difficulté vient du fait qu'à la base MSG et SMPI n'ont pas été prévus pour fonctionner en ensemble. il nous faut arriver à correctement initialiser en la partie MSG et SMPI.

4 Méthodologie

Comme nous travaillons avec Simgrid et StarPU à la fois, nous utilisons un dépôt complexe comprenant les deux et gérer avec l'outils submodule de git. Ce dernier nous de gérer des sous dépôt indépendemment, ainsi il est plus aisé de traiter les mises à jours de ces derniers.

Afin de pouvoir retracer le cheminement de mon travail, mais aussi de pouvoir garder le fil d'un jour à l'autre, un cahier de laboratoire est tenu en org-mode et est hébergé sur github. Cela permet également à mon tuteur de stage de savoir chaque jours l'avancement du projet et des difficultés rencontrées.

Comme on l'a vu précédemment il est nécessaire d'apporter quelques modifications au niveau du simulateur. Dans ce but, il a été dans un premier temps nécessaire de consulter la documentation afin de comprendre le fonctionnement et l'architecture de Simgrid. Ensuite il a fallut explorer le code afin de déterminer où et comment apporter les modifications. Pour cela les outils tels que GDB, Valgrind, les etags et CGVG ont été d'une aide précieuse.

5 Contribution

La toute première chose à réaliser afin de pouvoir effectuer des mesures, a été la gestion du partage du segment de données au niveau du simulateur. Comme la mémoire est partagée au sein d'un noeud, nous avons fait en sorte que les processus d'un même noeud aient leurs segment données en commun. Le principe est le suivant, il y a dans un premier temps, les processus SMPI qui sont créés au lancement de l'application avec leur propre espace de données. Puis ces dernier peuvent à leurs tours créer de nouveau processus. Ceux-ci héritent donc du segment de données du processus qui les a créés. Nous avons donc fait pointés le segment données des processus fils sur celui du père et un switch est effectué au changement de contexte.

Une fois la privatisation mise en place, nous avons constaté qu'il y avait un cas que nous n'avions pas pris en compte : celui des librairies dynamiques. En effet, nous n'avons privatiser que le segment données des applications or, les variables globales des librairies dynamiques ne se trouvent pas dans le segment données du processus et se retrouvent donc partagées.

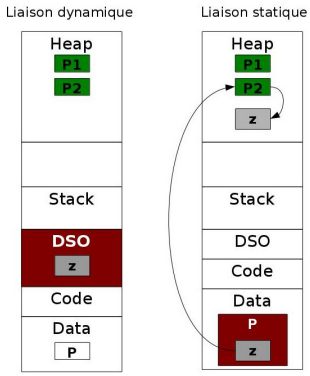

Figure 2 Emplacement en mémoire des bibliothèques

La solution qui nous avons employé est d'utiliser donc une version statique de la librairie. Ainsi, les variables globales se retrouvent dans le segment données du processus et ainsi la privatisation s'effectue grâce au mécanisme précédent. Cependant cette solution comporte une limitation car elle nécessite de changer la chaîne de compilation des applications utilisant StarPU, mais cela nous sera suffisant pour effectuer nos tests.

Ensuite comme nous l'avons évoqué tout à l'heure, MSG et SMPI

6 Validation

6.1 Test simple

Dans le but de tester le bon fonctionnement des modifications apportées, un test illustrant le fonctionnement de StarPU a été fourni et enrichi. Ce dernier permet ainsi d'isoler le problème afin de pouvoir nous concentrer dessus. Ce test, initialise Simgrid et la partie SMPI comme cela est fait du côté de StarPU et fait appel à une bibliothèque dynamique et manipule des variables globales. Ainsi lors de l'exécution de ce test, on doit pouvoir constater que pour des processus appartenant à un même noeuds, les valeurs des variables globales du programme et des bibliothèques dynamiques sont bien identiques. Ce qui après plusieurs correction a été le cas.

6.2 Test de StarPU - SMPI

Comme les résultats du test simples étaient ceux attendu, nous sommes passé à un test utilisant cette fois la vrai bibliothèque StarPU. Cette dernière est fourni avec des exemples de programme MPI notamment d'algèbre linéaire tel que l'algorithme de Cholesky. Nous nous sommes servi de ces dernier afin de valider les modifications. Cependant, malgré les ajouts apportés au test, ce dernier était incomplet et il semble avoir des soucis au niveau de l'initialisation de Simgrid côté StarPU. Malheureusement par manque de temps il n'a pas été possible de corriger le problème et donc les mesures n'ont pas encore pu être effectuées.

7 Conclusion

Afin de pouvoir conclure sur la question de comment évaluer les performances de StarPU-MPI et il faudra finir de valider les modifications de Simgrid afin de pouvoir faire fonctionner StarPU SMPI. Ensuite nous pourrons effectuer les simulations et les mesures. Pour ce faire les mesures seront faites sur le logiciel Chameleon (un solveur d'algèbre linéaire basé sur StarPU). Enfin, dans le but de valider le résultat des expérimentations, un test grandeur nature sera fait sur Grid5000. C'est pour atteindre cet objectif que j'ai choisi de prolonger mon stage.

Please give a shorter version with: \authorrunning and \titlerunning prior to \maketitle

Acknowledgments

Je souhaite remercier...