Rapport de Projet

Branch and bound Parallel

MENORET Clément, RULLIER Noémie
8 avril 2013

UNIVERSITÉ DE NANTES

Table des matières

1 Introduction

L'objectif de ce projet fut de paralléliser l'algorithme *branch and bound* permettant de calculer le minimum d'une fonction réelle par le découpage en boites de domaine.

Nous allons donc ici présenter les différentes décisions prise afin de paralléliser l'algorithme ainsi qu'une comparaison des résultats obtenus entre la version parallèle et séquentielle.

2 Les étapes de parallélisation

Afin de paralléliser ce programme, nous avons utilisé MPI et OpenMP. Ils permettent respectivement d'exploiter des ordinateurs distants, afin d'exécuter des portions de programme et de traiter les données, sur une architecture à mémoire partagée.

2.1 MPI

Nous avons ici décidé d'utiliser quatre ordinateurs grâce à MPI. Nous avons donc, dans un premier temps, regroupé tous les machines dans le même *Communicator* et initialisé le tout. A ce stade, les quatre ordinateurs vont exécuter la suite de la fonction main. Nous avons donc préciser que seul l'ordinateur de rang 0 exécute le premier appel à *minimize_first*.

Cette première fonction permet de faire un premier découpage en quatre intervalles. Chaque machine recevra ensuite l'ensembles des informations permettant d'exécuter à leur tour la fonction *minimize* ; chaque machine l'exécutera pour un des quatre intervalles. Ces informations seront envoyées via la fonction *MPI_Send()*. Elle permet d'envoyer des données à un ordinateur en lui précisant son rang. Afin de lui envoyer toutes les données nécessaires, nous avons créé de nouvelles structures :

- **interv** : cette structure permet de stocker l'intervalle qui va être envoyé. Elle est composée des attributs x et y de type *interval* ;
- **consts** : cette structure permet de stocker l'ensemble des paramètres nécessaires à la fonction *minimize* ;
- **package** : cette structure permet d'envoyer à la fois l'intervalle et les autres paramètres. Elle est composée d'un attribut *inter* de type *interv* et d'un attribut *constantes* de type *consts*.

Nous avons décidé de créer deux structures distinctes car pour chaque machine, l'ensemble des paramètres, autre que l'intervalle, sont identiques. Afin que notre structure soit bien envoyée via la fonction *MPI_Send()*, nous avons définit le type de données à envoyer comme étant des *MPI_BYTE*.

Nous avons donc, dans la suite de la fonction main, fait appel à la fonction *MPI_Recv()*. Celle-ci permet aux différentes machines de recevoir les messages envoyés par la machine de rang i (ici $i = 0$). Ces machines vont ensuite exécuter la fonction *minimize*.

De plus, dans le but de récupérer le résultat final, nous avons ajouté la fonction *MPI_Reduce()*. Celle-ci nous permettra de trouver le minimum de tous les minimums calculés par chaque machine.

2.2 OpenMP

Nous avons ajouté un niveau supplémentaire de parallélisation avec l'ajout d'OpenMP. En effet, sur chaque machine traitant une instance de *minimize* nous avons parallélisé les appels récursif à la fonction *minimize*. Nous avons donc défini quatre sections (*#pragma omp section*), une pour chaque appel à *minimize*. Nous avons choisi des sections plutôt que des tâches car elles permettent d'avoir une barrière à la fin de leur exécution.

Nous avons, de plus, ajouté une réduction sur le minimum obtenu dans toutes ces exécutions (*#pragma omp parallel sections reduction (min :min_ub)*).

3 Les résultats

4 Conclusion générale

Ce projet nous a permis de réfléchir à la façon de paralléliser un programme séquentiel tout en s'initiant à la programmation avec MPI et OpenMP. Bien que nous ayons regretté de ne pas pouvoir tester ce programme sur plusieurs machines physiques, nous avons pu constater à quel point l'utilisation du parallélisme sur un projet devient crucial à partir du moment où l'on souhaite obtenir un résultat calculé avec une bonne précision en un minimum de temps sur cet algorithme.