Cahier des charges : My Little Cluster

Cluster team

Table des matières

1 Présentation du groupe

1.1 Quentin de Laroussilhe

Je suis très motivé dans la réalisation ce projet, en effet, cela fait plusieurs mois que je m'intéresse aux techniques de parallélisme et je vais enfin me lancer dans une application concrête pour commencer à appréhender toutes les contraintes qu'imposent les architectures parallélisées.

FIGURE 1 – Quentin de Laroussilhe

1.2 Camille Dollé

Bonjour ! Je suis très enthousiaste à propos du projet que nous allons tenter de réaliser lors de ce second semestre. Tout d'abord, c'est un projet à réaliser en C ! Ce projet me permettra donc d'apprendre à manipuler plus en détails les différents aspects du C. Ce projet comporte également un fort aspect réseau, or je suis une débutante en matière de réseau, contrairement aux autres membres du projet. J'aurai donc l'occasion d'apprendre une montagne de notions sur le réseau lors de ce projet. Le sujet que nous a proposé Underflow m'a tout de suite intéressée, le projet représente pour moi un vrai challenge car il y a une vraie difficulté à le réaliser. D'une part, car il y a beaucoup de connaissances à acquérir avant de les mettre en pratique et d'autre part car le réseau est un domaine (presque) inconnu pour moi.

Le projet représente donc pour moi une compétence de plus à acquérir en informatique bien que notre projet impose beaucoup de recherches et de travail.

1.3 Guillaume Sanchez

Cet energunene n'a qu'une seule idee en tete, biffler ses congeneres.

1.4 François Boiteux

Étudiant à EPITA en 2ème année de cycle préparatoire je décide cette année pour le projet libre du second semestre de rejoindre un groupe de projet qui s'interessera à la paralélisation ! C'est un domaine qui offre de multiples possibilités !

D'un côté personnel, j'ai vingt ans car j'ai effectué un an en faculté avant de rejoindre EPITA ! J'aime l'informatique, ma copine, mes amis et c'est ce qui compte le plus pour moi. Mes hobbies sont les jeux videos, la musique et les soirées.

Je m'occuperais de la partie serveur du projet ce qui me permettera d'en apprendre certainement plus sur les sockets et les fonctionnements server/client.

FIGURE 2 – boiteu_f

2 La parallélisation

2.1 Pourquoi paralléliser un calcul ?

Certains algorithmes peuvent demander des temps de calcul importants ou très important. Paralléliser un algorithme permet de distribuer les tâches entre plusieurs noeuds de calcul et ainsi diviser le temps nécessaire à sa réalisation.

Par exemple, un algorithme permettant de retrouver un mot de passe à base de sa signature nécessite de tester une à une toutes les combinaisons possibles jusqu'à trouver le hash recherché. Si la fonction de hashage est sûre, il n'existe pas de méthode de recherche plus optimisée. L'intérêt de diviser le calcul si l'on dispose de plusieurs machines pour effectuer un tel calcul est alors évident.

2.2 Les contraintes de la parrallélisation

Pour faire fonctionner un programme sur plusieurs clusters de calcul travaillant en collaboration on doit avoir pensé le programme pour un tel mode de fonctionnement.

2.2.1 Les effets de bords

Les algorithmes itératifs non designés pour fonctionner sur des architectures parallélisées contiennent des effets de bord, c'est à dire que l'opération d'une instruction executée à un instant t dépend des instructions éxecutées précédemment.

Ainsi, si nous possédons plusieurs noeuds de calcul et que nous décidions d'envoyer à chaque noeud une instruction différente aucun parallélisme ne serait possible car chaque noeud devrait attendre que les noeuds en charge des instructions précédentes aient fini d'executer leurs instructions. Cela revient strictement à une éxecution séquentielle des instructions.

La première difficulté est donc de minimiser les effets de bords pour minimiser les temps d'attente pour executer les instructions.

2.2.2 Les problématiques d'accès mémoire

Si tous les noeuds travaillent sur le même espace mémoire, il peut se produire des problèmes d'accès si deux tâches tentent d'accéder au même emplacement.

3 Calcul distribué en réseau

3.1 Computer cluster

Ce terme va jouer un grand rôle dans notre projet, c'est pourquoi nous allons définir maintenant ce qu'on nous désignons par celui-ci.

En anglais, le terme "cluster" signifie "grappe". "Computer cluster" désigne donc un regroupement de plusieurs ordinateurs indépendants. Chaque ordinateur sera qualifié de "noeud" et sera utilisé comme un serveur indépendant. Mais la "grappe" finale sera considérée comme une seule et même entité. Les buts d'une telle configuration sont multiples : dépasser les limitations d'un seul ordinateur, augmenter la disponibilité... En bref, le "computer cluster" sert à obtenir de plus grandes performances. En effet, la principale utilisation du "computer cluster" est le calcul parallèle. Comment regrouper plusieurs ordinateurs entre eux ? Le procédé est très simple : on relie les ordinateurs via le réseau, en général un réseau local rapide.

Dans notre cas, par exemple si nous voulions “cracker” un mot de passe, chaque noeud/ordinateur effectuera une tâche, par exemple ici, un noeud aura pour tâche d'essayer des combinaisons de mots de passe sur une certaine plage, et un autre s'occupera en même temps d'une autre plage.

4 Etat de l'art

Les premiers ordinateurs étaient séquentiels, exécutant les instructions l'une après l'autre. Le parallélisme se manifeste actuellement de plusieurs manières : en juxtaposant plusieurs processeurs séquentiels ou en exécutant simultanément des instructions indépendantes.

4.1 Différents types de parallélisme : la taxinomie de Flynn

La taxinomie de Flynn, proposée par l'américain Michael J. Flynn est l'un des premiers systèmes de classification des ordinateurs créés. Les programmes et les architectures sont classés selon le type d'organisation du flux de données et du flux d'instructions.

*Les machines les plus simples traitent une donnée à la fois : ces systèmes sont dits « séquentiels ». Ce type de fonctionnement était prédominant pour les ordinateurs personnels jusqu'à la fin des années 1990. On parle d'architectures SISD (Single Instruction, Single Data).*Les systèmes traitant de grandes quantités de données d'une manière uniforme ont intérêt à être des SIMD (Single Instruction, Multiple Data) ; c'est typiquement le cas des processeurs vectoriels ou des unités de calcul gérant le traitement du signal comme la vidéo ou le son. *Les systèmes utilisant plusieurs processeurs ou un processeur multi-cœur sont plus polyvalents et pleinement parallèles, ce sont des MIMD (Multiple Instructions, Multiple Data).*Le type MISD a été beaucoup plus rarement utilisé, il semble néanmoins adapté à certains problèmes comme les réseaux neuronaux et aux problèmes temps-réel liés. L'architecture appelée Systolic array est un type d'architecture MISD.

4.2 Efficacité du parallélisme

Représentation graphique de la loi d'Amdahl. L'accélération du programme par la parallélisation est limitée par le nombre d'exécutions parallèles possible au sein de l'algorithme. Par exemple, si un programme peut être parallélisé à 90 %, l'accélération maximale théorique sera de x 10, quel que soit le nombre de processeurs utilisés.

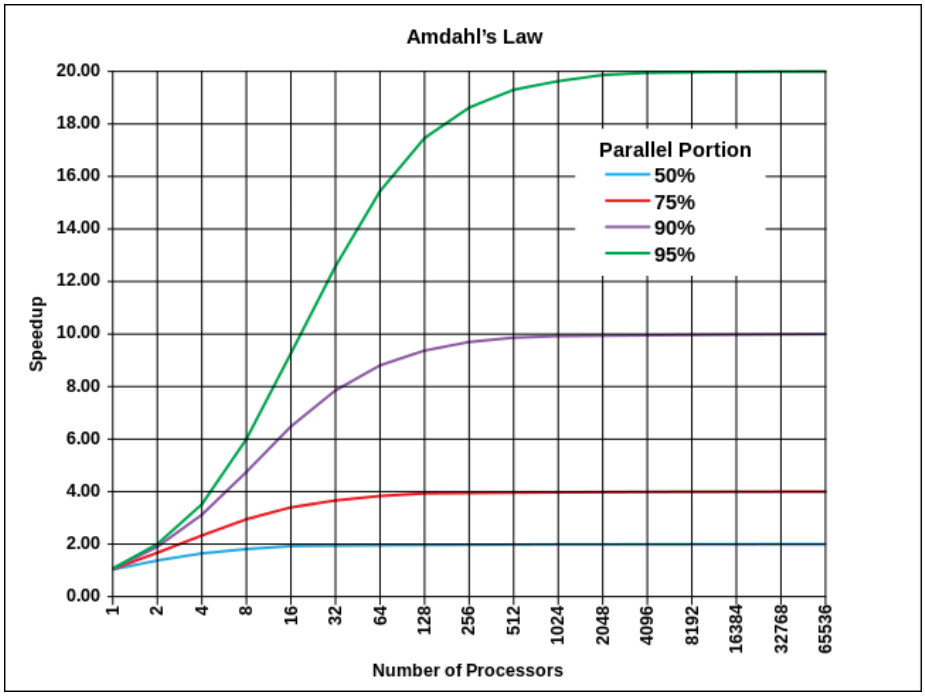

FIGURE 3 – Loi de Amdahl's