Historique des travaux autour du problème P||Cmax

florian colas

20 mai 2020

Sommaire

1 Introduction

texte

2 Présentation du problème.

texte

2.1 Parallélisme.

Le parallélisme est un type d'architecture informatique dans lequel plusieurs processeurs exécutent ou traitent une application ou un calcul simultanément. IL aide à effectuer de grands calculs en divisant la charge de travail entre plusieurs processeurs, qui fonctionnent tous en même temps.

Il existe quatre types de parallélismes, définis par la taxonomie de Flynn[(1)]. Cette classification est basée sur deux notions : le flot d'instructions (simple ou multiple), et le flot de données (simple ou multiples) ; un algorithme est un flot d'instructions à exécuter sur un flot de données.

Instructions \ Données	Simple	Multiple
Simple	SISD premiers PC machine de Von Neumann Obsolète, car tous les PC sont désormais multi-cœur.	SIMD Machines synchrones Pipeline Exécution d'une instruction unique sur des données différentes.
Multiple	MISD Machines vectoriels Tableau de processeurs Exécute plusieurs instructions sur une même donnée.	MIMD Multi processeurs à mémoire distribuée. Multi processeurs à mémoire partagée (multi-cœur). Multi Ordinateur.

Taxonomie de Flynn

Les premières machines parallèles étaient des réseaux d'ordinateurs, et des machines vectorielles (faiblement parallèles, très coûteuses), telles que l'IBM 360, les Cray1. La plupart des machines parallèles contemporaines sont désormais MIMD.

On peut définir une machine parallèle comme un ensemble de processeurs qui coopèrent et communiquent.

IBM 360-91 (le plus rapide et le plus puissant en service en 1968) NASA. Centre de vols de Greenbelt (Md)

2.2 Ordonnancement.

Sur une machine non parallèle, les tâches sont exécutées séquentiellement, les unes après les autres. Certaines tâches, ou jobs peuvent demander plus de temps que d'autres pour être entièrement traitées. Lorsque plusieurs ressources (processeurs, machines, coeurs) sont disponibles, ou que des jobs a

exécuter ne sont pas indépendants (même traités sur un seul processeur), se pose alors, un problème d'ordonnancement. Celui-ci consiste à organiser, dans le temps, les jobs à exécuter, en les affectant à une ressource donnée, de manière à satisfaire un certain nombre de contraintes, tout en optimisant un ou des objectifs. L'ordonnancement, fait partie de la catégorie des problèmes d'optimisation combinatoire.

Les problèmes qui s'y rattachent sont très variés. Premièrement, la nature des machines parallèles doit être considérée. Celles-ci peuvent être :

- identiques. (Le même temps de traitement sera nécessaire, d'une machine à l'autre) ;
- uniformes (un quotient de vitesse qi propre à une machine est à appliquer pour chaque tâche affectée à cette machine pour déterminer le temps de traitement nécessaire) ;
- indépendantes (les temps de traitements des tâches sont ni uniformes ni proportionnels d'une machine à l'autre).

Ensuite, des contraintes peuvent affecter les jobs eux-mêmes. Dans le cas d'un problème préemptif, les taches peuvent être interrompues, et reprises ultérieurement. Il est possible que les jobs soient indépendants, ou au contraire, être liées par des relations de précédence. Ces jobs ne sont disponibles qu'à partir d'une certaine date. Ou encore, être de durée égale, ou tous de durée différente.

Pour finir, l'objectif de l'ordonnancement est d'optimiser un critère. Par exemple, minimiser la somme des dates de fin, la somme des retards, le nombre de tâches en retard, ou simplement, le retard total. Mais le plus habituel, est de chercher à minimiser le temps total de traitement de tous les jobs, i.e minimiser le makespan.

2.3 Enoncé du $P||C_{max}$

Ces diverses possibilités définissent divers problèmes d'ordonnancements différents, recensés et classifiés par Graham et al. [1], qui introduit la notation trois-champs $\alpha|\beta|\gamma$.

Le problème $P_m||C_{max}$ se définit alors ainsi :

- $\alpha = \alpha 1\ \alpha 2$, détermine l'environnement machines. $\alpha = P$: Les machines sont parallèles et identiques : Un job, une tâche prendra le même temps de traitement qu'il soit exécuté sur une machine ou une autre. Le nombre de machines (m) est variable.
- β c { $\beta 1$, $\beta 2$, $\beta 3$, $\beta 4$, $\beta 5$, $\beta 6$}, détermine les caractéristiques des jobs, ou des tâches. β est vide. Ce qui signifie que la préemption n'est pas autorisée (les jobs doivent être exécutés d'une traite, sans interruption

ni coupure) et qu'il n'y a pas de relation entre les jobs (ils sont indépendants).
— γ détermine le critère à optimiser. $\gamma = C_{max}$: on cherche à optimiser le makespan, i.e le temps de traitement total.

$P_m||C_{max}$ consiste à planifier un ensemble J = {1,2,...,n) de n jobs simultanés, pour être traités par m machines identiques et parallèles. Chaque job, qui requière une opération, peut être traité par une des m machines. Le temps de traitement de chaque job (P_i avec i $\in$ N) est connu à l'avance. Un job commencé, et complété sans interruption. Les jobs, indépendants, sont exécutés par une seule machine, et une machine ne peut traiter qu'un seul job à la fois.

2.4 Problématique

Comme l'ont démontré Garey et Johnson, $P_2||C_{max}$ est un problème NP-Difficile [4], et $P||C_{max}$ est un problème NP-Difficile au sens fort [3]. Cependant,$P_m||C_{max}$ devient un problème NP-Difficile, du moment que le nombre de machines est fixé [2], comme l'a montré Rothkopf [5], qui a présenté un algorithme de programmation dynamique.

Donner la solution optimale à un problème d'ordonnancement (dans notre cas $P_m||C_{max}$) n'est pas réaliste. Même pour un problème de taille modeste, la résolution de celui-ci demanderait un temps excessif et donc rédhibitoire.

La résolution du problème d'ordonnancement va reposer sur des méthodes d'approche, qui consistent à calculer en temps polynomial, une solution « assez » proche de la valeur optimale.

Dans la littérature, l'étude d'ordonnancement est très riche et abondante. Le but étant d'améliorer le temps de calcul, et d'approcher le résultat optimal.

3 Résoudre le problème

Comme évoqué précédemment, l'existence d'une solution qui résout le problème n'est pas pensable, à moins que P = NP.

comme la dit legrand chneck, [1]

[2]

Bibliographie

[1] Federico Della Croce and Rosario Scatamacchia. The longest processing time rule for identical parallel machines revisited. *Journal of Scheduling*, pages 1–14, 2018.

[2] Michael R Garey and David S Johnson. "strong"np-completeness results : Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3) :499–508, 1978.