Ce qui nous intéresse comme mesure de l'efficacité du parallélisme est le speed up, et l'efficacité (combien de speed up on a obtenu par rapport au speed up qu'on voudrait avoir...)

Speed up est égal au degré de parallélisme moyen

Loi d'Amdahal

W et une fraction α qui ne se parallélise pas.

$$T_{par} = \frac{\alpha W}{R} + \frac{(1-\alpha)W}{pR}$$

$$S = \frac{T_{seq}}{T_{par}} \leq \frac{1}{\alpha}$$

Loi de Gustafson (weak scaling)

On a p processeurs qui travaillent pendant un temps T_{par} et qui réalisent un travail W.

Durant une fraction β de T_{par}, un seul processeur peut être exploité. Et durant une fraction $1-\beta$, tous les p processeurs travaillent. On a donc:

$$W = \beta T_{par} R + (1-\beta) T_{par} pR$$

$$T_{seq} = \frac{W}{R} = \beta T_{par} + (1-\beta) T_{par} p$$

$$S = \frac{T_{seq}}{T_{par}} = \beta + (1-\beta)p$$

$\beta = 10$ %, p = 1000, $S = 900$

Cela donne une vision optimiste du parallélisme: $S = O(p)$

Le travail augmente avec le nombre de processeurs vs autre loi -> travail fixe....

La différence fondamentale avec Amdahl est qu'ici on ne considère pas un travail fixe qu'on veut paralléliser, mais un travail qui augmente à mesure qu'on ajoute des processeurs.

Idée: donner plus de travail aux processeurs...

C'est souvent difficile d'accélérer un travail donné en le parallélisant, mais on peut toujours faire plus de travail avec plus de processeurs.

=> Le parallélisme est bien pour résoudre des problèmes plus grands plutôt qu'un problème donné plus rapidement...

Strong scaling

Étude du speed up en fonction du nombre de PE, pour un travail fixe.

Weak scaling

Étude du speed up en fonction de p en gardant le travail par processeurs constant.

Speed up superlinéaire

En général, on a que $S = \frac{T_{seq}}{T_{par}} \leq p$ car $W_{par} \geq W_{seq}$

Mais dans certains cas particuliers, on peut observer $S > p$: cela arrive si $W_{par} < W_{seq}$.

On peut distinguer 2 cas typiques:

- cela peut être un effet de mémoire cache: un problème pourrait ne pas rentrer dans le cache d'un seul processeur, mais une fois partitionné sur p processeurs, les données rentrent dans le cache. Donc ce serait un cas où $R_{eff} > R$ aver R_{eff} en parallèle et R en séquentiel.

  C'est un effet hardware...

- Il y a aussi un cas algorithmique qui est par exemple la recherche d'un élément particulier dans un vecteur, qui obéit à une propriété donnée.

  Chaque processeur s'occupe de $\frac{N}{p}$ données et fait la recherche dans ce sous-ensemble. Dès qu'un PE a trouvé, on s'arrête. On a donc: $T_{par} = n$ trouvé par P_k $T_{seq} = \frac{N}{p(k-1)}$ (Toutes les données des k - 1 processeurs).

  $$S = \frac{T_{seq}}{T_{par}} = \frac{N}{n}\frac{k-1}{p} + 1$$

  Cas limites:

  - $k = p, n = 1 => S = N >> p$
  - k = 1 => S = 1

En moyenne, on aura un speed up avoisinant p...

Ici, on a toujours comparé un speed up avec un processeur et plusieurs processeurs de puissance identique...

Toutes les petites formules se retrouvent en gros dans les systèmes électriques... Ex: courant = charge de travail etc... Il obéit aux lois d'Ohm...

Chapitre 5: modèles de performance

But: analyser des algorithmes parallèles simples et comprendre les performances qu'on peut espérer en fonction de n la taille du problème, et p le nombre de

processeurs.

Somme de n nombres sur p processeurs

On va ici considérer une architecture à mémoire distribuée.

shéma cours... (on a p processeurs, chacun contenant une donnée a_i.)

On a C: temps de communication. On a à chaque étape le temps de calcul plus de temps de calcul T_{cal}

On a donc: $T_{par} = (C + T_{cal}) \times log_2 p$ avec p une puissance de 2.

On a $T_{seq} = (p-1)T_{cal}$

Donc le speedup est donné par:

$$S = \frac{T_{seq}}{T_{par}} \approx \frac{pT_{cal}}{(C + T_{cal})log_2 p} = \frac{p}{(1 + \frac{C}{T_{cal}})log_2} < p$$

Le terme $(1 + \frac{C}{T_{cal}})$ est plutôt grand...

Même si $C = 0$, on a encore $S < p$, $S = \frac{p}{log_2 p}$

Pourquoi n'avons nous pas $S = p$?

La raison est un problème de "**load balancing**", il n'y a pas de travail pour tous les processeurs durant tout le calcul.

On peut voir que $\frac{p}{log_2 p}$ est le nombre moyen de processeurs utilisés.

Cas n » p

Dans ce cas, on espère un meilleur speedup car le problème devient beaucoup plus grand.

Mauvais algorithme

On pourrait considérer $\frac{n}{p}$ couches de données dans chaque processeur, les sommer l'une après l'autre par l'algorithme $n = p$, puis p_0 qui a récolté toutes ces sommes, fait l'addition finale.

$$T_{par} = \frac{n}{p}(C + T)log_2 p + (\frac{n}{p} - 1)T$$

avec $T = T_{cal}$

Ce qui est environ égal à $\frac{n}{p}(C + T)log_2 p + \frac{n}{p}T$ car n est grand...

$T_{seq} = (n-1)T \approx nT$

Speedup:

$$S = \frac{T_{seq}}{T_{par}} = \frac{nT}{\frac{n}{p}(C+T)log_2 + \frac{n}{p}T} =? \frac{p}{1+(1+\frac{C}{T})log_2 p}$$

Autre méthode: chaque processeur somme ses $\frac{n}{p}$ valeurs, et ensuite on applique l'algorithme $n = p$.

$$T_{par} = (n/p - 1)T + (C+T)log_2 p = \frac{n}{p}T + (C+T)log_2 p$$

à comparer avec $T_{seq} = nT$

$$S = \frac{nT}{\frac{n}{p}T + (C+T)log_2 p} = \frac{p}{1+(1+\frac{C}{T})\frac{plog_2 p}{n}}$$

Donc ici c'est bien mieux qu'avant car si n est assez grand, le terme $(1+\frac{C}{T})\frac{plog_2 p}{n}$ peut être rendu aussi petit qu'on veut.

On voit donc à nouveau que le parallélisme est bénéfique si chaque processeur a suffisamment de travail par rapport à l'overhead de la parallélisation...

$T_{par} = \frac{nT}{p} + (C+T)log_2 p$ avec $(C+T)log_2$ l'overhead de la parallélisation.

$\frac{T_{seq}}{p}$: calcul utile.

Ici, on voit que l'overhead croît avec $log_2 p$, mais si n est assez grand, cela sera masqué.

Remarques

Plus il y a de processeurs, plus l'overhead rique d'augmenter.

Supposons qu'on ait $N = 19$ tâches à répartir sur $p = 6$ processeurs.On peut répartir comme cela:

4 3 3 3 3 3 (nb tâches sur chacun des 6 processeurs...) ou comme cela: 4 4 4 4 3 0 (5 processeurs. On peut espérer que l'overhead sera plus bas, et que le temps de calcul sera le même.