Évaluation et critères de performances d'un calcul parallèle

Mesurer les performances
et optimiser un
algorithme parallèle

Introduction

➢ Programme séquentiel : évaluation de la performance par la mesure du temps d'exécution;

➢ Programme parallèle : évaluation plus complexe

Les performances d'un code parallèle vont dépendre :

– du nombre de processeurs;

– de la machine sur laquelle il va s'exécuter;

– et bien d'autres critères (cf plus loin)…

➢ En outre,

– Un programme séquentiel rapide peut s'avérer, une fois parallélisé, très peu efficace ;

– Inversement, un programme séquentiel "moins rapide" peut s'avérer mieux parallélisable;

Mesures des performances

Charge de travail d'un algorithme parallèle

➢ La charge de travail d'un algorithme séquentiel $\mathbf{W_{séquentiel}}$ se décompose en deux parties :

$$\mathbf{W_{séquentiel} = \underbrace{W_{fixe}}_{\text{Travail non parallélisable (coût fixe)}} + \underbrace{W_{parallélisable}}_{\text{Travail pouvant être parallélisé}}}$$

➢ La charge de travail d'un algorithme parallèle $\mathbf{W_{parallèle}}$ avec **p** processeurs est donc :

$$\mathbf{W_{parallèle}(p) = W_{fixe} + W_{parallélisable}/p}$$

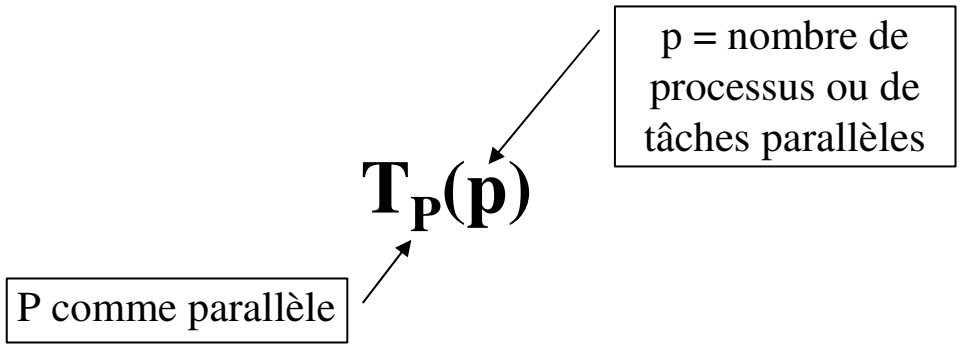
➢ Dans un "vrai" programme parallèle, il faut prendre en compte une surcharge de travail $\mathbf{W_{com}}$ due aux synchronisations des tâches :

$$\mathbf{W_{parallèle}(p) = W_{fixe} + W_{parallélisable}/p + W_{com}}$$

Temps d'exécution (*elapsed time*)

➢ Définition : temps écoulé entre le début du calcul parallèle et la terminaison de la dernière tâche;

➢ Notation :



Coût total (*cost*)

- Définition : somme des coûts de tous les processeurs;
- Mesure un temps "équivalent séquentiel" ;
- Notation :

$$C_T(p) = p \times T_P(p)$$

Surcoût parallèle (*Parallel Overhead*)

$$\mathbf{T_O(p) = \underbrace{pT_P(p)}_{\text{Temps consommé par tous les processeurs}} - \underbrace{T_S}_{\text{Temps séquentiel}}}$$

➢ Mesure le surcoût en temps total consommé du calcul parallèle par rapport au calcul séquentiel;

➢ Normalement, un calcul parallèle coûte plus cher qu'un calcul séquentiel :

Démonstration : hypothèse : temps **T** proportionnel au coût **W**

$$T_P(p) = T_{fixe} + T_{com} + T_{parallélisable}/p$$

$$\Rightarrow \quad pT_P(p) = pT_{fixe} + pT_{com} + T_{parallélisable}$$

$$= (p-1)T_{fixe} + pT_{com} + T_{fixe} + T_{parallélisable}$$

$$> 0 + T_S$$

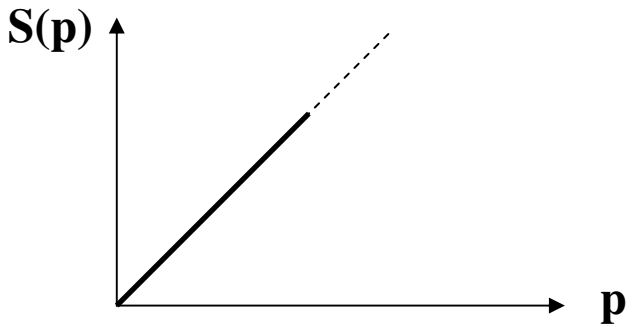
$$\Rightarrow \quad T_O(p) > 0$$

Accélération Relative (*speed-up*)

$$S(p) = T_P(1) / T_P(p)$$

- Mesure l'accélération, le bénéfice d'un calcul parallèle par rapport au même algorithme sur un processeur;

- On dit qu'un programme est extensible (*scalable*) si et seulement si **S(p) ~ p** (ou encore mieux si **S(p) = p**) :
  - Difficile à obtenir ;
  - Souvent extensible que sur un intervalle de **p** ;



À suivre…

…Suite

Accélération Relative (*speed-up*)

- En théorie,
  - $S(p) \leq p$
  - Autrement dit, un calcul parallèle sur p processeurs ne peut pas s'exécuter p fois plus vite que le meilleur algorithme séquentiel ;

- Démonstration :

$$S(p) = \frac{T_{fix} + T_{par}}{T_{fix} + T_{par}/p + T_{com}}$$

$$= \frac{p(T_{fix} + T_{par})}{T_{fix} + T_{par} + (p-1)T_{fix} + pT_{com}}$$

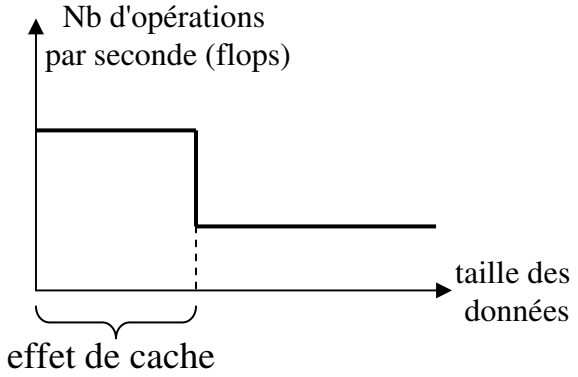
$$= \frac{p}{1 + ((p-1)T_{fix} + pT_{com})/(T_{fix} + T_{par})} \leq \frac{p}{1+0} = p \qquad (*)$$

À suivre…

…Suite

Accélération Relative (*speed-up*)

- En pratique, il est existe des cas où **S(p) > p** !
  - ⇨ On dit alors que le programme est **superscalaire**;
- Pourquoi un programme peut être superscalaire ?



⇨ Grâce aux effets de cache

Effets de cache en séquentiel :

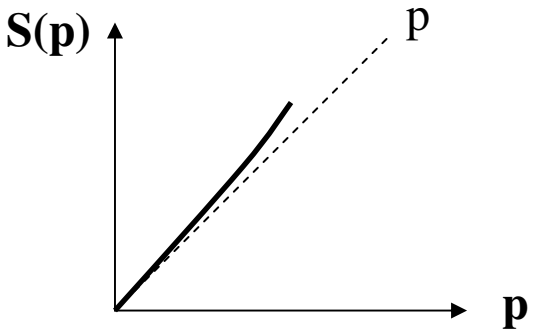
la vitesse de traitement d'un vecteur "long"
est moins élevée que celle d'un vecteur "court"
Conséquence : l'hypothèse **T** proportionnel à
W est fausse !

Effets de cache en parallèle :

– Pour un problème de taille fixée,
lorsque le nombre de processeurs augmente,
la taille des données par processeur diminue ;

– le calcul séquentiel manipule des données
volumineuses;

– à partir d'un certain nombre de processeurs,
le calcul parallèle manipule des données qui tiennent
dans les caches des cpus, d'où une suraccélération;



À suivre…

Accélération Relative (*speed-up*)

- En outre,

$$\forall\ p,\quad S(p) \le 1/\alpha \qquad \textit{(Loi d'Amdahl)},$$

où $\alpha = T_{fix}/T_S$ est la part du travail non parallélisable sur le travail total

Autrement dit, l'accélération d'un algorithme parallèle est bornée par une constante, et ceci quelque soit le nombre de processeurs

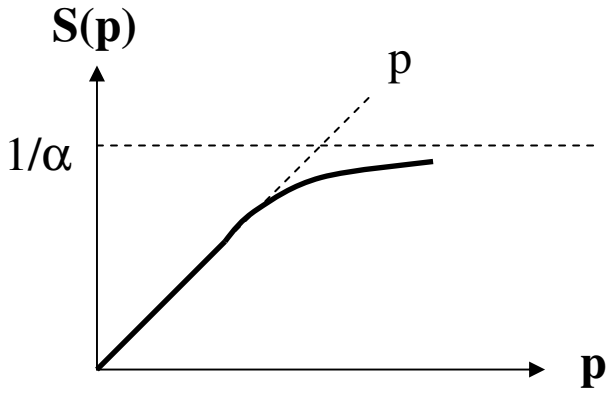
- Démonstration : d'après (*),

$$S(p) = \frac{p}{1+((p-1)T_{fix}+pT_{com})/(T_{fix}+T_{par})}$$

$$= \frac{p}{1+((p-1)T_{fix}+pT_{com})/T_S} \qquad (**)$$

$$\le \frac{p}{1+(p-1)\alpha} = \frac{1}{1/p+(1-1/p)\alpha}$$

$$\xrightarrow[p\to\infty]{} \frac{1}{\alpha}$$



À suivre…

…Suite

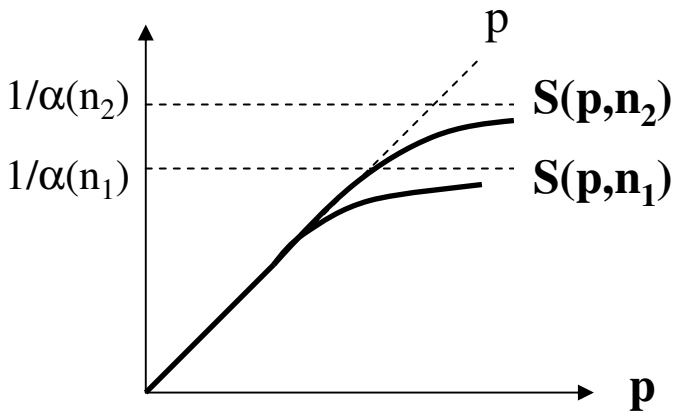
Accélération Relative (*speed-up*)

➢ Pour être exact, l'accélération dépend du nombre de processeurs mais également de la taille des données **n** à traiter

$$\mathbf{W_{séquentiel}(n) = W_{fixe} + W_{parallélisable}(n)}$$

En règle générale, $\mathbf{W_{parallélisable}(n)}$ augmente quand **n** augmente
donc, $\alpha(n) = \mathbf{W_{fixe}} / \mathbf{W_{séquentiel}(n)} \rightarrow 0$ quand $n \rightarrow \infty$

Pour un nombre de processeurs fixé, l'accélération sera d'autant meilleure que la taille des données à traiter sera importante :



À suivre…

…Suite

Accélération Relative (*speed-up*)

- Il ne faut pas oublier l'influence des synchronisations des tâches :

  Plus il y'a de processeurs, plus le temps pris par les communications est important :

  $$\mathbf{T_{com}(p)} \rightarrow \infty \text{ quand } \mathbf{p} \rightarrow \infty$$

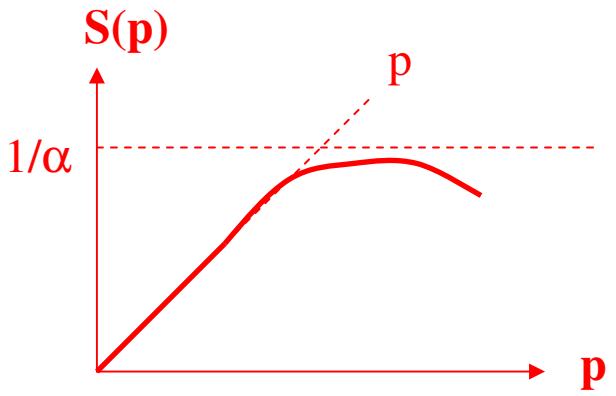
  Conséquence : $\mathbf{S(p)} \rightarrow 0$ quand $\mathbf{p} \rightarrow 0$

  Démonstration : d'après (**),

  $$S(p) = \frac{1}{1/p + (1-1/p)\alpha + T_{com}(p)/T_S} \underset{p\to\infty}{\sim} \frac{1}{\alpha + T_{com}(p)/T_S}$$

  Or, $\mathbf{T_{com}(p)} \rightarrow \infty$ quand $\mathbf{p} \rightarrow \infty$, donc

  $\mathbf{S(p)} \rightarrow 0$ quand $\mathbf{p} \rightarrow \infty$



Accélération Absolue (*speed-up*)

$$S^*(p) = T_S / T_P(p) \ ,$$

où T_S est le temps du meilleur algorithme séquentiel

- Mesure l'accélération, le bénéfice d'un calcul parallèle par rapport au meilleur algorithme séquentiel ;

- Rappel : en règle générale, $S(p) \neq S^*(p)$ car $T_P(1)$ n'est pas forcement le temps du meilleur algorithme séquentiel;

- Il est impossible que $S^*(p) > p$ (superscalaire) sinon T_S ne serait pas le temps du meilleur algorithme séquentiel (manipulation des données par bloc pour bénéficier des effets de caches);

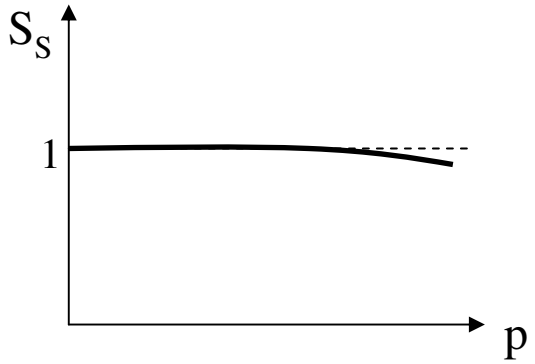
Scaled speed-up

$$S_S(p, n) = T_P(1, n) / T_P(p, p \times n) \ ,$$

où **n** est la taille des données à traiter

- Interprétation : si je multiplie par **p** le nombre de processeurs et la taille des données, est-ce que cela me prend le même temps ?

- Le programme est extensible si $S_S(p) \sim 1$;



Efficacité parallèle (*efficiency*)

$$E(p) = T_S / C_T(p)$$

- Mesure le "rendement" du calcul parallèle;

- En théorie, l'efficacité est un nombre compris entre 0 et 1 :

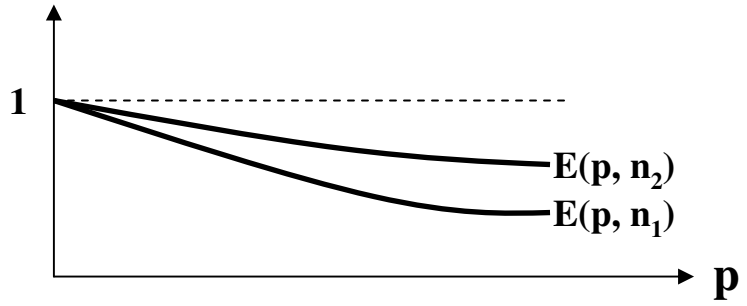
$$E(p) = T_S / C_T(p) = T_S / (pT_P(p)) = S(p) / p \leq 1$$

Mais, en pratique, l'efficacité peut être supérieure à 1 dans le cas des calculs superscalaires;

- Programme extensible si **E = 1**: 100 % des ressources parallèles ont été utilisées pour du calcul "utile" ;

- **$S(p) \leq 1/\alpha \Rightarrow E(p) \leq 1/(\alpha p)$**

(rappel : α = part de la charge non parallélisable sur la charge totale);



Équilibrage de charge

- Mesure la "qualité" de la répartition de la charge de travail sur les processeurs;

Soit $\mathbf{W_i}$ la charge de travail du processeur **i**
Soit $\mathbf{W_{max}}$ la charge maximale : $\mathbf{W_{max} = max_{i \in [1,p]}\ W_i}$
Soit $\mathbf{W_{tot}}$ la charge totale de tous les processeurs : $\mathbf{W_{tot} = \Sigma_{i \in [1,p]} W_i}$
Soit $\mathbf{W_{moy}}$ la charge moyenne par processeur : $\mathbf{W_{moy} = W_{tot} \ / \ p}$

On définit l'équilibrage de charge $\boldsymbol{\varepsilon(p)}$ comme :

$$\boldsymbol{\varepsilon(p) = (W_{max} - W_{moy}) \ / \ W_{max}}$$

- Si $\boldsymbol{\varepsilon(p) = 0}$ (cas où $\mathbf{W_{max} = W_{moy}}$), alors l'équilibrage est parfait ;

Critères de performances

Critères de performances ?

Pour résoudre le même problème, le coût d'un calcul parallèle est supérieur au coût d'un calcul séquentiel;

Qu'est-ce qui engendre ce surcoût ?

- $\mathbf{W_{fixe}}$: travail non parallélisable :
  – peu de marge de manœuvre;
  – de plus, la part du travail fixe sur le travail total diminue quand le nombre de données à traiter augmente;

  ⇨ ne pas perdre son temps à diminuer $\mathbf{W_{fixe}}$!

- $\mathbf{W_{com}}$: travail dû au traitement parallèle :
  – le travail augmente quand le nombre de tâches parallèle augmente;
  – le travail augmente quand le nombre de données à traiter par tâche augmente;

  ⇨ il faut concentrer l'effort sur $\mathbf{W_{com}}$ car c'est de lui que vont dépendre en partie les bonnes performances du calcul parallèle !

Quelles sont les sources du surcoût dû au traitement parallèle ?
Comment diminuer cette surcharge de travail ?

Les communications

Souvent, c'est le critère qui a le plus d'influence sur les performances parallèles;

Une communication est un surcoût car pendant que l'on communique, on ne calcule pas …

- en général, une communication est lente par rapport à du calcul donc pénalisant pour les performances;
- temps $\mathbf{T_{msg}}$ passé dans un message de **n** octets peut être modélisé par :

$$\mathbf{T_{msg}(n) = T_{lat} + \tau \times n}$$

$\mathbf{T_{lat}}$: temps d'initialisation du message

$\mathbf{1/\tau}$: débit défini par la bande passante

- En règle générale, $\mathbf{T_{lat} >> \tau \times 1}$, mais les ordres de grandeurs dépendent des machines parallèles (surtout des réseaux):
  - ex communications lentes : réseaux de PCs ;
  - ex communications rapides : réseaux supercalculateurs parallèles
    - InfiniBand : $T_{lat} = 6\mu s$, débit = 10 Gb/s ;
    - Quadrics : $T_{lat} = 2\mu s$, débit = 7 Gb/s;
  - mais cela reste toujours lent devant du calcul !

À suivre…

…Suite

Les communications

Conséquences (et solutions):

- chaque message a un coût fixe important ⇨ **il faut le moins de messages possibles**;

- il faut des **messages les plus longs possibles** pour minimiser l'impact des latences;

- il faut souvent **réorganiser l'algorithme séquentiel** pour minimiser le nombre de communications lors de sa parallélisation;

- lorsque c'est possible, **recouvrir les communications par du calcul** (effectuer du calcul pendant que les communications se déroulent, notion de communications non bloquantes).

Remarque sur les communications collectives : elles coûtent encore plus chers que les communications point à point (coût de l'ordre de p fois un échange, où p est le nombre de processeurs).

Calculs supplémentaires

- Calculs non présents dans l'algorithme séquentiel mais indispensables dans l'algorithme parallèle ;

- Différentes sources :
  - éviter des communications :
    - dans certains cas, il est préférable que toutes les tâches parallèles effectuent le même calcul, plutôt qu'une seule tâche suivie d'une phase de communication;
    - dans quels cas ? On évalue le coût de la phase de calcul par rapport au coût d'une communication. En règle générale, le coût de la phase de calcul ne dépend pas de la taille des données à traiter;
  - cas où l'algorithme parallélisé est différent de l'algorithme séquentiel:
    - conséquence : plus d'opérations en parallèle par processeur que pour l'algorithme séquentiel;
    - dans quels cas ? L'algorithme séquentiel est difficilement parallélisable (mauvaises accélérations) ou non parallélisable.

Remarque : ces calculs supplémentaires sont inévitables voire indispensables (donc, pas de solution).

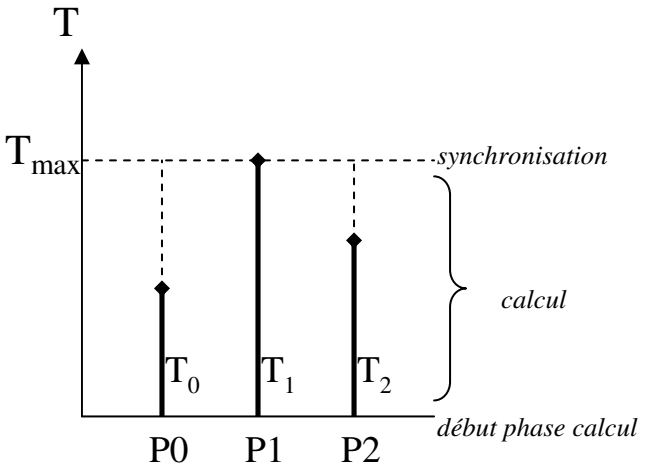
Déséquilibrage de charge

(*Imbalance workload*)

Que se passe t'il quand les tâches n'ont pas la même charge de travail ?

– temps de la phase parallèle : $T_{max} = T_1$;

– coût de la phase parallèle : $C_T = 3 \times T_1$;



Qu'ont fait P0 et P2 entre le moment où ils ont terminé et le moment où P1 a terminé ?

– ils ont attendu ! (*idling*)

– temps passé dans l'attente : $\Sigma_i (T_{max}-T_i) = (T_1-T_0)+(T_1-T_2)$;

– si on rapporte cette attente au coût total du calcul, on retrouve la mesure de l'équilibrage de charge : $\varepsilon(p) = \Sigma_i (T_{max}-T_i) / (p\ T_{max})$;

À suivre…

…Suite

Déséquilibrage de charge

(*Imbalance workload*)

- Équilibrage parfait (équidistribution du travail) ⇨ pas de perte de temps dans des attentes ;

- Un calcul est rarement parfaitement équilibré : comment répartir **n** données à traiter sur **p** tâches ?

- Un bon équilibrage sera **facile** à obtenir si :

  – **n** est grand devant **p** : on parle alors de grain fin (*fine granularity*);

  – répartition statique (i.e. ne change pas au cours du calcul);

- Un bon équilibrage sera **difficile** à obtenir si :

  – **n** est du même ordre de grandeur que **p** : on parle alors de grain grossier (*coarse granularity*);

  – la charge de travail est dynamique : la taille de la charge n'est pas prévisible car générée par l'algorithme :

  ⇨ répartition dynamique de la charge (*dynamic load balancing*), mais coûteux (=> calculs supplémentaires);

  – les tâches s'exécutent sur des unités de traitement de puissances différentes ou d'architectures différentes (réseaux hétérogènes) : ex : réseau de PCs avec des fréquences différentes, ou réseau constitué de machines d'architectures différentes …

En bref …

➢ Les performances d'un calcul parallèle s'évaluent à l'aide de plusieurs mesures comme l'accélération ou l'équilibrage ;

➢ Les performances parallèles sont essentiellement impactées par :

– les communications : il en faut le moins possible, et leurs messages doivent être les plus grands possibles ;

– le déséquilibrage de charge induit des attentes inutiles sur les processeurs les moins chargés : veillez à une répartition équitable du travail !