

CentraleSupélec  

Mineure CalHau I

Métriques, mesure et analyse de performances

Stéphane Vialle

 université PARIS-SACLAY  Sciences et technologies de l'information et de la communication (STIC)  

Stephane.Vialle@centralesupelec.fr
<http://www.metz.supelec.fr/~vialle>

CentraleSupélec

Métriques, mesure et analyse de performances

1. Métriques de performance (T, S, e)
2. Métriques de *size up*
3. Critères de *passage à l'échelle*
4. Modélisations sur machines idéales
 1. Loi d'Amdahl
 2. Loi de Gustafson
 3. Liens Amdahl-Gustafson
5. Méthodologie de mesure du temps d'exécution

CentraleSupélec

1 – Métriques de performance (T, S, e)

- Observation de la courbe du T_{exec}
- Métrique d'accélération (*Speedup*)
- Métrique d'efficacité (*Efficiency*)
- Choix de la référence séquentielle
- Sources de perte de performances

Métriques de performance

Observation de la courbe du T_{exec}

Difficultés pour accélérer une application (de taille fixée)

Quand les surcoûts de gestion du parallélisme (*Tsynchro*, *Tcomm*...) sont grands...

...l'accélération est faible !

Exec Time (s)

Seq. Algo

Nb of resources

Quand l'algorithme parallèle a une complexité beaucoup plus grande que le meilleur algorithme séquentiel...

...le gain final reste faible !

Exec Time (s)

Seq. Algo

Nb of resources

Métriques de performance

Observation de la courbe du T_{exec}

Courbe de temps prometteuse

Quand l'algorithme parallèle a :

- un temps d'exécution qui décroît significativement
- un surcoût limité sur une seule ressource

...le gain final sera important !

Exec Time (s)

Seq. Algo

Nb of resources

...et on poursuit l'analyse

→

T_{exec} vs T_{exec} idéal
Speedup
Efficiency
Size up
Scalability...

Métriques de performance

Observation de la courbe du T_{exec}

Quelle représentation adopter pour un temps d'exécution ?

Remarque Générale :

- Il est préférable de connaître de l'allure de la courbe attendue...
- ...et de choisir une représentation qui permet de visualiser des droites ou des formes géométriques simples

Cas d'un temps d'exécution parallèle :

- courbe idéale : $T(P) = T(1)/P$
→ une hyperbole
- l'hyperbole est mal identifiée par l'œil...
- ...on la confond facilement avec une courbe d'une autre loi !

T_{exec} (s)

Nb of tasks

measures

idéal

Métriques de performance

Observation de la courbe du T_{exec}

Quelle représentation adopter pour un temps d'exécution ?

Remarque Générale :

- Il est préférable de connaître de l'allure de la courbe attendue...
- ...et de choisir une représentation qui permet de visualiser *des droites* ou des *formes géométriques simples*

Cas d'un temps d'exécution parallèle :

- courbe idéale : $\log(T(P)) = \log(T(1)) - \log(P)$
- en échelle log une hyperbole est une droite de pente -1
- on détecte facilement :
 - un écart complet à la théorie
 - un écart dû à une loi proche

Métriques de performance

Métrique d'accélération

Speedup :

$$S(P) = \frac{T(1)}{T(P)}$$

$\left\{ \begin{array}{l} S(P) < 1 : \text{on ralentit !} \\ \text{mauvaise parallélisation} \\ 1 < S(P) < P : \text{"normal"} \\ P < S(P) : \text{hyper-accélération} \\ \text{analyser \& justifier} \end{array} \right.$

Métriques de performance

Métrique d'accélération

Speedup :

$$S(P) = \frac{T(1)}{T(P)}$$

$\left\{ \begin{array}{l} S(P) < 1 : \text{on ralentit !} \\ \text{mauvaise parallélisation} \\ 1 < S(P) < P : \text{"normal"} \\ P < S(P) : \text{hyper-accélération} \\ \text{analyser \& justifier} \end{array} \right.$

Cas standard :

Il y a toujours des sources de perte de performance...

Métriques de performance


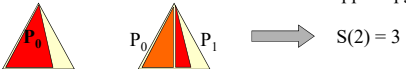
Métrique d'accélération

Cas d'hyper-accelération :

Ce n'est pas *magique*, et ce n'est pas *normal*
 → On doit analyser le phénomène et l'expliquer
 → Corriger une erreur ou exploiter une optimisation

Exemples d'explications :

- on ne fait plus les bonnes opérations (résultat faux)
- les données tiennent dans le cache total des P processeurs
- on a modifié l'algorithme de départ et on converge plus vite (ex. de l'algorithme génétique optimisé !)
- on cherche une solution dans un arbre et on stoppe le pgm

Métriques de performance

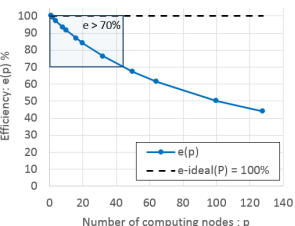
Métrique d'efficacité

Efficacité :


$$e(P) = \frac{S(P)}{P}$$

Taux d'utilisation des ressources, ou fraction obtenue de l'accélération idéale

- $e(P) \in [0;1]$, $\in [0\%;100\%]$
- $e(P) > 100\% \Leftrightarrow$ hyper-accelération



L'utilisateur s'intéresse à l'accélération obtenue
 L'acheteur de la machine s'intéresse à l'efficacité des applications exécutées
 Le développeur s'intéresse aux deux



Métriques de performance

Choix de la référence séquentielle

A quel programme et exécution séquentielle se comparer ?

1) Même programme lancé sur un seul processeur ?
 Même algorithme implanté en séquentiel ?
 Meilleur algorithme séquentiel connu ?

2) Compilation séquentielle avec le même compilateur ?
 Compilation avec le meilleur compilateur séquentiel ?

3) Optimisations séquentielles autorisées par la parallélisation ?
 Optimisations séquentielles maximales ?

4) Exécution sur un seul processeur de la machine parallèle ?
 Exécution sur la meilleure machine séquentielle ?

Métriques de performance

Choix de la référence séquentielle

Tous les choix sont plausibles :

Chaque choix de référence séquentielle correspond à :

- un point de vue différent,
- une préoccupation différente,
- un objectif d'analyse différent

→ Faire le choix correspondant à sa problématique
 → Énoncer clairement ce choix

Exemples :

Utilisateur final →
 SON pgm séquentiel sur SA machine séquentielle

Développeur de code parallèle →
 SON pgm parallèle sur UN proc de SA machine parallèle

Métriques de performance

Choix de la référence séquentielle

La référence séquentielle peut être obtenue avec :

Même algo, même langage, même optim seq., même proc → $S_1(P)$ **Bonnes perfs faciles à obtenir**

↕

Meilleur algo, meilleur langage, meilleur optim seq., meilleur proc → $S_2(P)$ **Bonnes perfs très difficiles à obtenir**

Métriques de performance

Sources de perte de performances

- Sous-optimisation séquentielle ↔ Sous-utilisation de chaque coeur
- Impossibilité de vectoriser les boucles
- Fraction séquentielle ↔ Sous-utilisation des noeuds de calcul
- Surcoût des synchronisations
- Surcoût des communications
- Déséquilibre de charge entre tâches
- IO séquentielles/re-séquentialisées ↔ Plate-forme trop faible
- Réseau d'interconnexion trop faible

Rmq : certains algorithmes nécessitent des plates-formes très performantes (d'autres non...)

2 – Métriques de *size up*

- Conception d'un code apte au *size up*
- Métrique de *size up* en temps d'exécution
- Métrique de *size up* en temps et ressources
- Bilan : *Size Up* en 3 objectifs successifs
- Exemple expérimental

Métriques de *size up*

Conception d'un code apte au *size up*

1^{er} objectif : pouvoir traiter des problèmes plus gros sur plus de rsracs

Un code qui **réplique** la plupart de ses données sur toutes les machines sera toujours limité par la taille mémoire d'**une** machine...
... et ne sera **pas apte au *size up***

Métriques de *size up*

Conception d'un code apte au *size up*

1^{er} objectif : pouvoir traiter des problèmes plus gros sur plus de rsracs

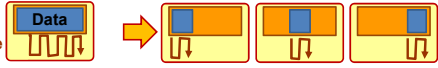
Un code qui **répartit** la plupart de ses données sur toutes les machines pourra stocker plus de données sur plus de machines...
... et sera **apte au *size up***

Métriques de *size up*

Conception d'un code apte au *size up*

1^{er} objectif : pouvoir traiter des problèmes plus gros sur plus de rsres

→ Conception d'une répartition initiale des données, et d'un schéma de communication à volume minimal

RAM  Code

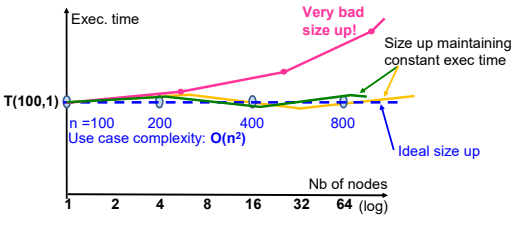
- **Besoin de faire circuler les données initiales entre les nœuds** : pour qu'un nœud puisse poursuivre ses calculs sur d'autres données que les siennes
- **Besoin de faire circuler les résultats intermédiaires entre les nœuds** : pour qu'un nœud puisse poursuivre les calculs d'un autre, avec ses propres données

Métriques de *size up*

Métrique de *size up* en temps d'exec.

2^{ème} objectif : maintenir constant T_{exec}

$T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$
avec : T (taille pb, nb rsres)



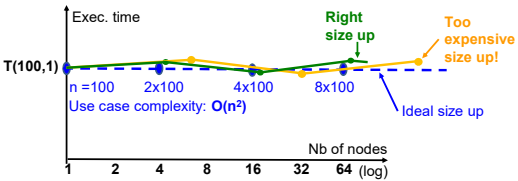
→ **Objectif non atteint !**

Métriques de *size up*

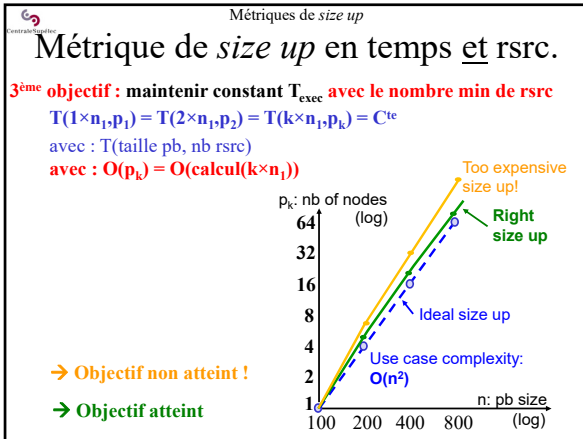
Métrique de *size up* en temps et rsres.

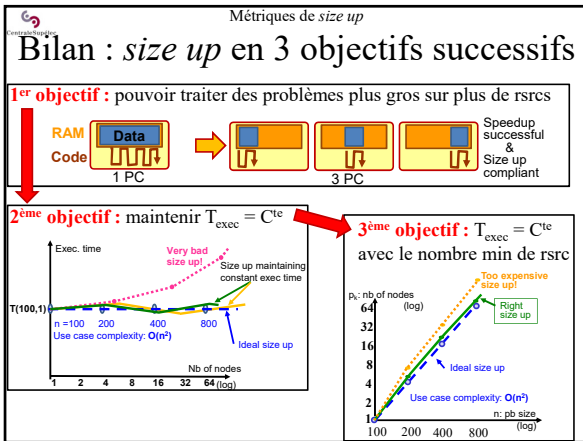
3^{ème} objectif : maintenir constant T_{exec} avec le nombre min de rsres

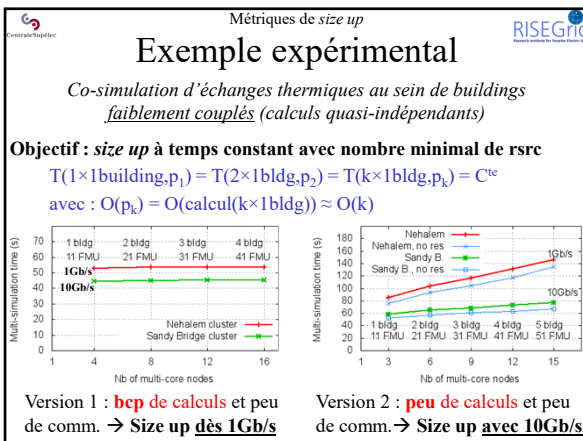
$T(1 \times n_1, p_1) = T(2 \times n_1, p_2) = T(k \times n_1, p_k) = C^{te}$
avec : T (taille pb, nb rsres)
avec : $O(p_k) = O(\text{calcul}(k \times n_1))$



→ **Objectif non atteint !** → **Objectif atteint**







3 – Critères de passage à l'échelle

- Une métrique basée sur le T-exec
- Démarche de *Size Up* et de *Speedup*
- Graphique de passage à l'échelle

Critères de passage à l'échelle

Une métrique basée sur le T_{exec}

Quand la taille du pb croît on ne peut plus faire d'exéc séquentielle :

- pas assez de RAM, pas assez de disque...
- exécution trop longue, rsrc non monopolisable...

→ on ne peut pas mesurer de *référence séquentielle* !
 → on ne peut pas calculer de *Speedup* !!

→ **Bâtir une analyse seulement sur des mesures du T_{exec}**

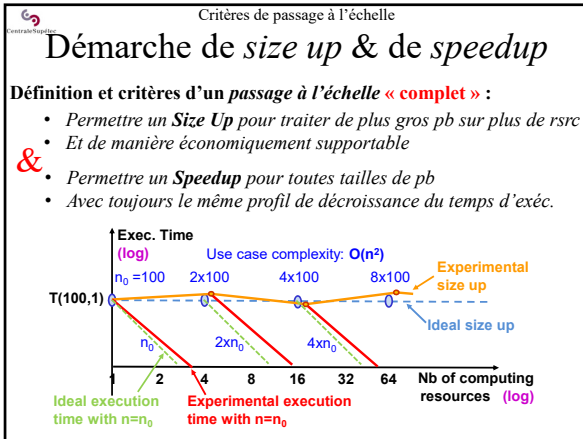
Critères de passage à l'échelle

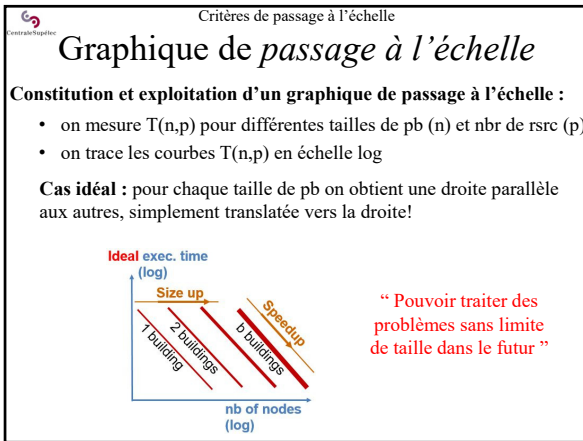
Démarche de *Size Up* & de *Speedup*

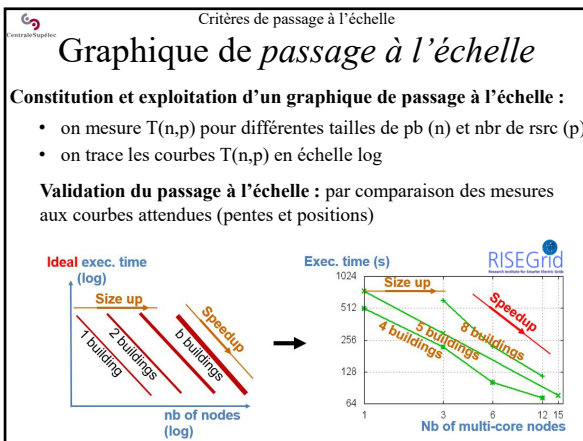
Définition et critères d'un passage à l'échelle « simple » :

- Permettre un *Size Up* pour traiter de plus gros pb sur plus de rsrc
- Et de manière économiquement supportable

→ *Size Up* avec temps d'exéc. constant et nbr minimal de rsrcs







Critères de passage à l'échelle

Graphique de *passage à l'échelle*

Constitution et exploitation d'un graphique de passage à l'échelle :

- on mesure $T(n,p)$ pour différentes tailles de pb (n) et nbr de rsrc (p)
- on trace les courbes $T(n,p)$ en échelle log

Utilisation en « abaque » : Pour une taille de pb donnée on peut identifier le nb de rsrcs à utiliser pour respecter un T -exec maximum

Une solution qui *pass*e à l'échelle *complètement* permet de :

- répondre aux besoins de calculs
- nécessiter le minimum de rsrc
- quantifier les rsrcs nécessaires
- planifier les dépenses associées

4 – Modélisations sur machines idéales

4.1 - Loi d'Amdahl

Modélisations sur machines idéales

Loi d'Amdahl : Motivations

Programme séquentiel

Programme parallèle

Fraction séquentielle du programme ...

... quel est son impact sur les performances ?

Modélisations sur machines idéales

Loi d'Amdahl : Définitions

Hypothèses de départ :

$$T(1) = T_{seq} = T_s^a + T_p^a$$

↓ Temps de la partie *séquentielle* du programme (au sens de Amdahl)

↓ Temps de la partie *parallélisable* du programme (au sens de Amdahl)

$$T(P) = T_s^a + T_p^a / P + \text{Overhead}$$

↓ Hyp : machine parallèle idéale

$$T(P) = T_s^a + T_p^a / P$$

- Synchro sans surcoût !
- Msgs instantanés !
- ...

Modélisations sur machines idéales

Loi d'Amdahl : Définitions

Définition de la fraction séquentielle au sens de Amdahl :

$$f_s^a = \frac{T_s^a}{T(1)} = \frac{T_s^a}{T_s^a + T_p^a} \in [0;1]$$

Et la fraction parallèle: $f_p^a = \frac{T_p^a}{T(1)} = \frac{T_p^a}{T_s^a + T_p^a} \in [0;1]$

D'où : $f_s^a + f_p^a = 1$

Réécriture du temps d'exécution:

$$T(P) = T_s^a + T_p^a / P$$

$$T(P) = f_s^a \cdot T(1) + (1 - f_s^a) \cdot T(1) / P$$

Modélisations sur machines idéales

Loi d'Amdahl : Définitions

Conséquence sur le speedup :

Par définition du Speedup : $S^a(P) = \frac{T(1)}{T(P)}$

Donc : $S^a(P) = \frac{T(1)}{f_s^a \cdot T(1) + (1 - f_s^a) \cdot T(1) / P}$

Soit : $S^a(P) = \frac{1}{f_s^a + \frac{1 - f_s^a}{P}}$ $\lim_{P \rightarrow \infty} S^a(P) = \frac{1}{f_s^a}$ $S^a(P) \leq \frac{1}{f_s^a}$

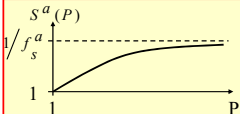
$$S^a(P) = P \cdot \frac{1}{1 + f_s^a \cdot (P-1)}$$

$$S^a(P) = S^{ideal}(P) \cdot \frac{1}{1 + f_s^a \cdot (P-1)}$$

Modélisations sur machines idéales

Loi d'Amdahl : Impact sur le *speedup*

Allure du *speedup* :



Exemple (a.n.) :

$$f_s^a = 1\% \Rightarrow \begin{cases} S^a(P) < 100 \\ S^a(100) = 50.3 \end{cases}$$

Petite fraction séquentielle →
grosse limitation pour P grand!

Modélisations sur machines idéales

Loi d'Amdahl : Confrontation à la réalité

En réalité les overhead ne sont pas négligeables :

$$T(P) = T_s^a + T_p^a / P + \text{Overhead}$$

⇒ $S^{\text{réel}}(P) < S^a(P)$

- Temps de communication
- Temps de synchronisation
- Temps d'ordonnancement
- ...

Pourtant on n'observe pas de si mauvais résultats (en général) !

“On sait obtenir de bon *speedup* sur de grosses machines parallèles”

→ Les fractions séquentielles sont généralement faibles ?

Le modèle ne correspond pas toujours à la réalité ?

Loi de Gustafson

4 – Modélisations sur machines idéales

4.2 - Loi de Gustafson

Modélisations sur machines idéales

Loi de Gustafson : Motivation

1 – Impact de la fraction séquentielle (selon Amdahl) ?

Programme séquentiel → Programme parallèle

2 – Mais on observe qu'Amdahl n'est pas réaliste (trop pessimiste) :

Construire un modèle plus proche de la réalité ...

Modélisations sur machines idéales

Loi de Gustafson : Définitions

Idee/Constataion de base :

En pratique : on ne veut pas diminuer le temps d'exécution T_0 , mais augmenter la quantité de travail W traitée en T_0

↓

- On traite W_0 en séquentiel en : $T(1, W_0)$
- On traite W en parallèle en : $T(P, W)$

Et on choisit P et W tels que : $T(P, W) = T(1, W_0) = T_0$

Modélisations sur machines idéales

Loi de Gustafson : Définitions

Temps que durerait le traitement séquentiel de W :

$$T(1, W) = T_s^g(P, W) + P \times T_p^g(P, W) - \text{Overhead}$$

↓ Temps de la partie séquentielle du programme (au sens de Gustafson)

↓ Temps de la partie parallélisée sur P processeurs (au sens de Gustafson)

Modélisations sur machines idéales

Loi de Gustafson : Définitions

Hypothèse : partie séquentielle constante

$$T_s^g(P_1, W_1) = T_s^g(P_2, W_2) = T_s^g(P, W) = Cte$$

Vrai notamment pour des calculs scientifiques itératifs

The diagram shows a Gantt chart for parallel execution. A vertical axis represents time, with 0 at the top. A horizontal axis represents the number of processors, with 0 on the left and 1 on the right. A green bar represents the sequential part of the computation, which is constant in length across all processor counts. An orange bar represents the parallel part, which is longer for more processors. The total length of the bars represents the total execution time $T(P, W)$. Labels include $T(1, W)$ for the total time on 1 processor, $T(P, W)$ for the total time on P processors, and $T_s^g(P, W)$ for the sequential part.

Modélisations sur machines idéales

Loi de Gustafson : Définitions

Introduction de la fraction séquentielle de Gustafson :

$$f_s^g / T_s^g = f_s^g \times T(P, W), \quad f_s^g \in [0, 1]$$

avec : $P = P_g(W)$

$T(P, W) = Cte$, par définition
 $T_s^g = Cte$, par hypothèse

$f_s^g = Cte$

Nouvelle expression du temps séquentiel (estimé) :

$$T(1, W) = T_s^g(P, W) + P \times T_p^g(P, W)$$

$$T(1, W) = f_s^g \times T(P, W) + P \times (1 - f_s^g) \times T(P, W)$$

Modélisations sur machines idéales

Loi de Gustafson : Définitions

Speedup fonction de la fraction séquentielle :

$$S(P, W) = \frac{T(1, W)}{T(P, W)}$$

$$S_s^g(P, W) = \frac{f_s^g \cdot T(P, W) + P \cdot (1 - f_s^g) \cdot T(P, W)}{T(P, W)}$$

$$S_s^g(P, W) = f_s^g + P \cdot (1 - f_s^g)$$

$S^g(P, W) = f_s^g + P \cdot (1 - f_s^g)$ Speedup au sens de Gustafson

Mais la modélisation de Gustafson considère uniquement des couples (P_i, W_i) tels que $T(P_i, W_i) = T(1, W_0)$

$$S^g(P(W)) = f_s^g + P(W) \cdot (1 - f_s^g)$$

Modélisations sur machines idéales

Loi de Gustafson : Impact sur le *speedup*

Speedup non borné :

$$\lim_{P \rightarrow \infty} S^G(P) = \infty$$

Allure du speedup :

Bilan de la loi de Gustafson :

- Adopte le **point de vue utilisateur** : augmenter W à T -exec constant
- **Hypothèse optimiste** de partie séquentielle constante ($T_s^G(P, W) = Cte$)

4 – Modélisations sur machines idéales

4.3 - Lien Amdahl-Gustafson

Modélisations sur machines idéales

Amdahl – Gustafson : Comparaison

Amdahl et Gustafson aboutissent à des conclusions différentes :

Amdahl

Etude **l'extensibilité forte** (*strong scalability*) : capacité à décroître T -exec qd seule la taille de la machine croit.

Gustafson

Etude **l'extensibilité faible** (*weak scalability*) : capacité à maintenir T -exec qd les tailles de la machine et du pb croissent.

Modélisations sur machines idéales

Amdahl – Gustafson : Comparaison

Amdahl et Gustafson aboutissent à des conclusions différentes :

Amdahl

Gustafson

En fait Amdahl et Gustafson :

- Font des hypothèses différentes
- Définissent des fractions séquentielles différentes
- **Ne sont pas incompatibles**

Modélisations sur machines idéales

Amdahl – Gustafson : Comparaison

Différence de modélisation de la fraction séquentielle :

Gustafson-Barsis:

T_s^g remains constant when w increases

Amdahl:

$T_s^g(w)$ evolution unspecified when w increases

Modélisations sur machines idéales

Amdahl – Gustafson : Relation entre SU

Expressions complètes des speedup

$$S^a(p, w) = \frac{1}{f_s^a(w) + \frac{1 - f_s^a(w)}{p}}$$

$$S^g(p_g(w)) = f_s^g + p_g(w) \cdot (1 - f_s^g)$$


Allures des speedup


En augmentant P et W (selon Gustafson) à f_s^g constant :

→ $S^g(P)$ intersekte différent $S^a(P)$, avec des $f_s^a(w)$ décroissant

Modélisations sur machines idéales

Amdahl – Gustafson : Bilan

Amdahl : W Cte, P ↑, T-exec ↓
 Réaliste quand on travaille à W Cte 
 Réaliste quand on augmente P pour diminuer T-exec

Gustafson : W ↑, P ↑, T-exec Cte
 Réaliste quand on augmente W sur *certaines* pbs 
 Ex : plus de cycles de calculs lors de simulations

En général :
 Valeurs des courbes réelles moins bonnes à cause des *overhead*
 Allures des courbes réelles entre Amdahl et Gustafson

5 – Méthodologie de mesure du temps d'exécution

Mesure des temps d'exécution

Méthodologie de mesures


Mesures externes :

```
>time myAppli
>/usr/bin/time myAppli
>times myAppli
>timex myAppli
.....
```


12.002u	user
0.128s	system
12.150	total

Nom et fonctionnement variables selon le système utilisé !!

Fréquemment : total > user + system !!



Simple à utiliser
Pas de modifications des codes sources



Peu précis: ± 0.5s

Mesure des temps d'exécution


Méthodologie de mesures

Mesures internes :


```
time()
clock()
gettimeofday()
omp_get_wtime()
```

- Compte les clicks d'horloge
- Compte le temps écoulé en s

- Toutes ces routines ne sont pas toujours disponibles !
- "gettimeofday" est en général une bonne solution.
- Parfois il existe des outils plus précis pour mesurer de petites durées.



Plus précis que les mesures externes



Besoin de modifier le code source
Pas toujours totalement portable

Mesure des temps d'exécution

Méthodologie de mesures

Précision des outils et des mesures :

123456789012 . 1234567890123456

←

Capacité maximale de l'outil de mesure

→

Précision théorique (cf. doc)

←

Précision expérimentale, vu la fluctuation des exécutions

- Ne pas tenir compte de trop de décimales!
- Faire attention à ne pas déborder la capacité de mesure!

Mesure des temps d'exécution

Méthodologie de mesures

Problème fréquent :

Test en mode exclusif (mono-user).
Outil de mesure à 1ms de précision.

→

Fluctuation de 500ms d'une exécution à l'autre !!
Et plus encore avec la montée en fréquence automatique des procs. (effet de "chauffe")

Démarche conseillée :

- Mesurer les fluctuations, ne pas les ignorer (le *warm up* des processeurs peut perturber les premières)
- Ne pas donner que les valeurs moyennes
- Mesurer des temps > 10s si possible

Mesure des temps d'exécution

Méthodologie de mesures

Stocker des meta-données sur les conditions de mesure :

- **Date de l'exécution**
- **Auteur(s) du test**
- Outil(s) de mesure utilisé(s)
- Caractéristiques de la machine : RAM, Cache, Processeurs, ...
- OS utilisé (nom et version)
- Compilateur utilisé (nom et version)
- Options de compilation utilisées
- Test en multi-user/mono-user ?
- Présence d'IO dans le test ?
- Configuration du programme de test : taille des données, ...

On oublie souvent (et rapidement) à quel benchmark se réfère une série de mesures !

On manque souvent de détail sur les conditions de réalisation d'une série de mesures !

Métriques, mesure et analyse de performances

FIN
