

Chap1

Limite Von Neumann : si on augmente vitesse proc. → chaleur → propriété semi conducteurs
Flops : mesure performances

Performance : maj proc (technologie), architecture (améliorer différents étapes de calculs), algorithme.

Optimiser son code, inventer algo

Comment voir efficace :

R/p où R est puissance de calcul, p la puissance consommé.

1988-2000 : boom machine parallèle, SIMD, MIMO, MPP

1995 : cluster de PC

2010 : GPU

Système parallèle : machine avec plusieurs processeurs pour solution d'un même problème.

Système répartis : plusieurs processeurs interconnectés : résolution simultanée de plusieurs problèmes.

	Couplage entre proc	Mise en commun délibéré	Hétérogénéité des ressources	Hypothèse sur le système	Connaissance mutuelle	Fiabilité	Problème de sécurité
Parallèle	Fort	Oui	Non	Oui	Oui	Oui	Non
Réparti	Faible	Non	Oui	Non	Non	Non	Oui

Séquentiel : Mme Dupont fait un a un. $T_{seq} = N * 4 * T$

Pipeline : une personne fait une tâche $T_{pipeline} = 4 * T + (N-1) * T$ (il reste N-1 qui sont prêts une à une chaque étape t_n vaut $N * t$ si N est grand). 4 fois plus vite NB : temps peut être perdu
Pas de tolérance aux pannes

Single Instruction flow, multiple data flow

SIMD : supervisé par Mme. Dupont, les autres font $T_{simd} = 4 * T * p$ (p travailleur) = $(4 * t * N) / p$
Chacun a ses ressources (**mémoire distribués**)

MIMD : chacun a sa recette mais ressources partagées (→ perte performance), bcp de programme

Single Program Multiple Data flow

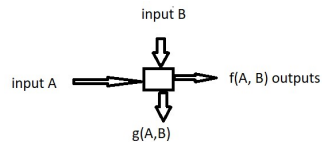
SPMD : comme MIMD

Chap2

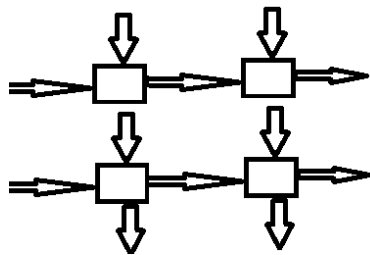
		Flow of Data	
		Single	Multiple
Instruction Flow	Single	SISD (von Neumann)	SIMD
	Multiple	MISD (pipeline ?)	MIMD

FIGURE 2.1 – Diagramme représentant la classification de Flynn

Réseaux systoliques :



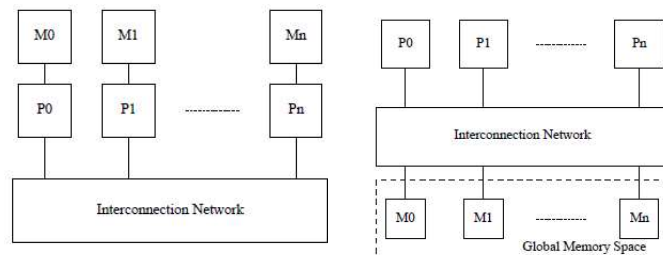
réseau systolique : architecture SIMD de type particulier
réseau systolique : réseau interconnecté de cellules simples
(cellules qui travaillent en parallèle) :



chaque cycle d'horloge correspond à un calcul+communication

réseau systolique : intéressant pour traitement de signal des systèmes embarqués

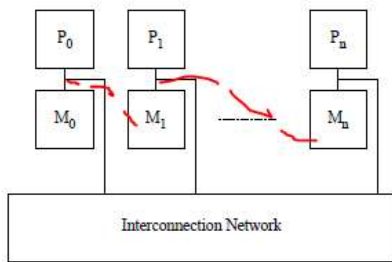
MIMD : mémoire distribués (tout le monde a ses ressources) comme MPI / partagé



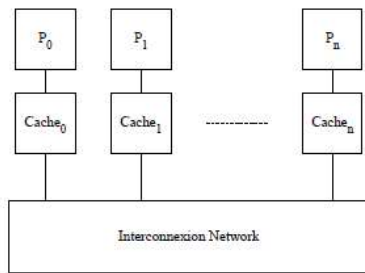
un proc uniquement peut accéder A LA FOIS à la mémoire
difficulté avec architecture mémoire partagé est que processeurs ont mem caches. Variable peut donc être copié à plusieurs endroits et modifié de façon inconstitents.

Il faut ajouter code pour invalider ces modifs et admettre uniquement modif. Atomique
Processeurs continuellement en communication pour assurer ctte cohérence → peu scalable

NUMA : Non Uniform Memory access : mémoire virtuellement partagée, physiquement distribué (on accède mémoire des autres éléments = lent)

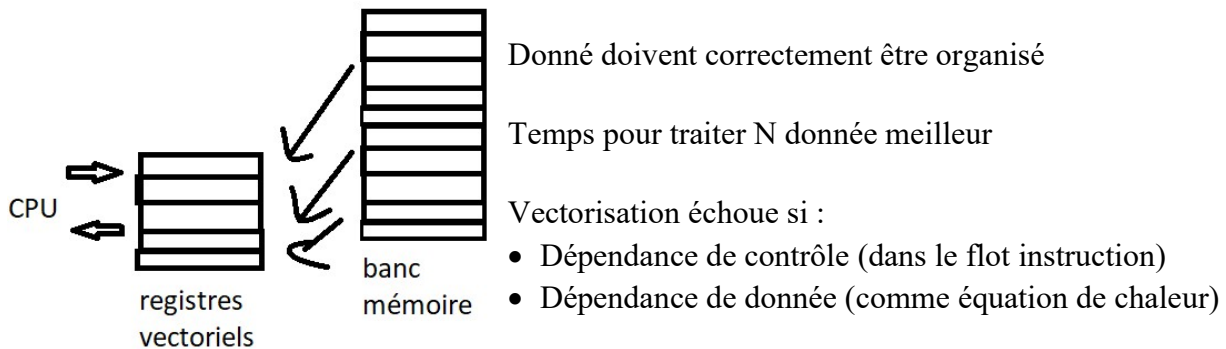


COMA : Cache Only Memory Access : mémoire associative adressés par le contenu.
Données n'ont pas d'emplacement fixe. Elles migrent de processeurs en processeurs selon besoin. Système hybride ou hiérarchique



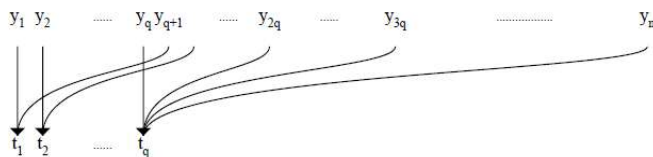
Architecture Vectoriels :

- Registre vectoriels : accéder à la mémoire par des vecteurs : plusieurs données à la fois. (fait face au goulet d'étranglement de Von Neumann)



Dépendance de contrôle : les conditions comme if détruit le flow

Dépendance de données : des boucles for avec dépendance de donnée, on effectue technique de déroulement de boucle



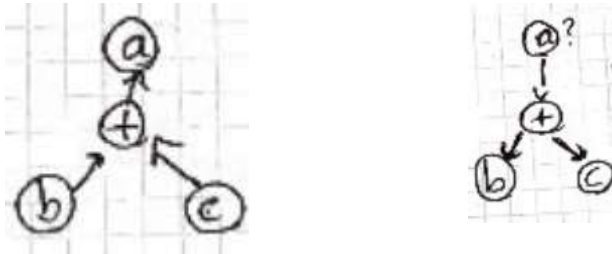
$\log_2(n)$ pour dérouler la boucle

- Pipeline
- Unités d'exécutions multiples

Architecture Dataflow :

Data Driver : disponibilité de b et c, va activer opération qui donne c.

Demande Driver : demande de a, va activer opération de recherche de b et c.



Parallélisme interne : mémoire cache pour réduire goulet d'étranglement

Dans processus modernes, il y a d parallélisme au niveau des instructions (Instruction Level Parallelism)

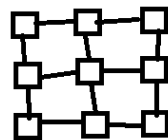
Il faut faire du pipelining : utiliser une pipeline d'exécution

Chap3

Machine parallèle : processeurs qui coopèrent et ils le font en communiquant.

Communication interprocesseurs : réseau d'interconnexion.

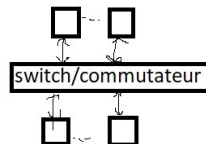
Topologies :



Un carré c'est un proc+routeur

Statique :

graphe dont les nœuds sont les proc + routeurs et les arcs sont les fils de connexion.



Dynamique :

Les processeurs, la périphérie d'un switch et d'établir des connexions entre paire de processeur.

But : grande bande passante, faible latence

Propriétés d'un réseau d'interconnexion :

- Le diamètre D : longueur minimal qui relie deux processeurs les plus éloignés
- Le degré : nombre de liens (fil) qui arrive sur un processeur
- Bande passante : Débit du réseau en byte/sec

- Latence : temps qui sépare msg envoyé et arrive a destination (latence physique + latence préparation du message)
- Largeur bisectionnelle : mesure débit global dans réseau (il faut diviser le réseau en 2 moitié égales et en comptant le nombre de liens entre ces 2 moitiés) : nombre de lien quand tu sépare réseau en deux.
- Prix et/ou complexité : mesure en fonction du nombre de processeurs, qté de matériel
- Scalabilité : peut on tjrs faire croitre ou ya limite ?

Routage : on utilise connaissance du réseau d'interconnexion pour améliorer ça.

Technicien de commutation : comment les messages se déplacent-ils à travers le réseau.

STORE AND FORWARD (bad)

- Temps pour m bytes = $m \cdot l \cdot W$ avec l longueur du chemin

COMMUTATION PAR TRAIN DE BITS (good) (worm hole/cut through)

CUT THROUGH : réserve canal et on continue de passer (1 passe de A à B, quand 2 passe de A à B, 1 passe de B à C)

- $T_{\text{cut through}} = 1/W + (m-1)/W$ (à chaque cycle m-1 morceaux restant arrive) $1/w$ temps du premier morceau (donc environ M/W)

Primitives Communications

Communication point-à-point : proc à proc. MPI Send, Recv

Permutation : envoie d'un proc à un autre proc déterminé par calcul $i \rightarrow j(i)$

Broadcast : one to all : MPI_Bcast

Réduction (many to one, all to one) : MPI Reduce avec une opération

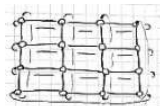
Gather : rassemble, scatter : divise






Echange total : multi broadcast : chaque proc envoie a tous les autres proc

Echange total personnalisé : tous les proc. Envoie donnée différente à tout les autres : multi scatter

Opération de scan ou préfixe : combinent communication et calcul de sorte que la donnée final du ième proc soit la somme (pour éviter qu'un proc fait la somme total)

Réseaux statiques :

Anneau	Diamètre : $N/2$ Degré : 2 (indépendant de N), Largeur bisectionnel : 2 Prix : $O(n)$ scalabilité : Oui	
Grille 2D		Diamètre : $O(\sqrt{N})$ degré : 4 largeur b. : \sqrt{N} prix : $O(4N)$ = nb connexion scalabilité : oui

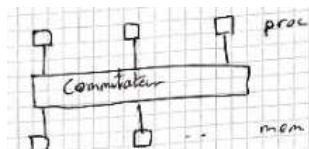
Grille 3D	 Diamètre : $O(N^{1/d})$ degré : $2d$ largeur bisectionnel : $N^{1/d}^{(d-1)}$ Prix : $O(b*d)$ scalabilité : Oui Routage : d'abord, on s'aligne à j puis on monte à i
Hypercube	<p> $D = 0$: 1 seul proc $D = 1$: 2 proc  $D = 2$:  4 proc $D = 3$:  8 proc $D = 4$:  16 proc </p> <p>Le nombre de processeurs (nœuds) N est relié à la dimension $N = 2^d \Leftrightarrow d = \log_2(N)$</p> <p>Pour construire un hypercube de dim. K, on prend deux hypercubes de dim $k-1$ et on relie les nœuds correspondants.</p> <p>Diamètre : $d = \log_2(N)$ donc diamètre : $O(\log(N))$ Croissance la plus faible vu jusqu'à maintenant</p> <p>Degré : $\log_2(N)$ donc augmente avec la taille Largeur bisectionnelle : $N/2$ car par construction d'un hypercube de dim $d-1$ avec $2^{(d-1)} = N/2$ nœuds. Comme on l'a dit, on relie deux cubes de dim $d-1$, donc ça correspond à ça Prix : dN (nombre de lien par nœud fois nombre de nœud) = $O(N*\log(N))$ croissance plus rapide que linéaire Scalabilité : non car il faut des nœuds avec de + en + de lien et donc on ne peut pas simplement combiner.</p>

Hypercube :

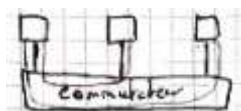
On envoie à voisin de dim. Adresse origine et XOR avec destination. Les 1 indiquent dimensions à traverser.

Réseaux dynamiques :

Souvent switches ou commutateurs qui relient proc.



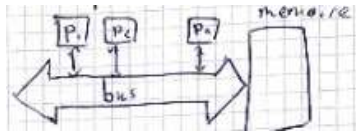
Mém partagé



le carée a meme et proc

3 catégories de réseaux dynamiques :

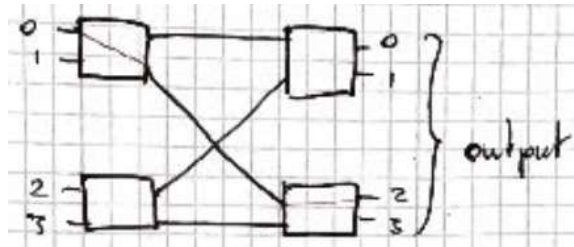
- Connexion par bus : beaucoup utilisé en informatique mais pas vraiment pour le parallélisme (mémoire partagée)



plusieurs proc partagent même bus pour accéder mémoire.
 \Rightarrow un seul proc à la fois
 Bande passante $O(1/N)$, donc décroît, plus il y a de processeur. PAS SCALABLE

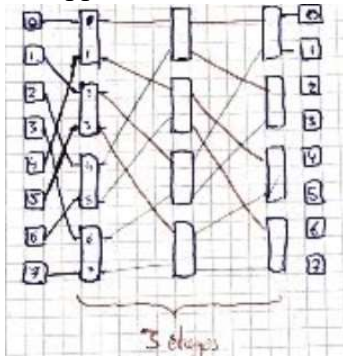
En plus, il faut mettre en place, un FIFO pour accès mémoire.

■ Réseaux multi-étages : bon compromis entre coût et performance.



Même switch 2×2 que tout à l'heure
 Chaque entrée peut être connectée à n'importe quel sortie.
 Mais un lien entrée sortie contraint les autres possibilités.
 Si $0 \rightarrow 2$, 1 ne pourra aller que sur 0 ou 1. C'est pourquoi, on dit qu'il est **bloquant**.
 Solution pour diminuer cela : ajouter des chemins avec couches supplémentaires.

On appelle cela un réseau **OMEGA**. C'est un parmi plusieurs réseaux multi-étages possible.



Nombre d'étage $\log_2(N)$
 Il faut réfléchir en binaire pour les connexions : ex pour aller de 2 à 5 ($010 \rightarrow 101$)
 1^{er} étage : 1 0 1
 2^{ème} étage : 0 1
 3^{ème} étage : 1
 C'est un shift circulaire vers la gauche des bits d'entrées.
 Cette permutation : **shuffle inverse**.

Algo routage : adresse destination : chaque étage, on regarde bit dominant, 0 sort par haut, 1 par bas peu importe l'origine

Preuve : Soit adresse $0102 \dots 0k$ où $k = \log_2(N)$ ici $k=3$

Soit $d_1 d_2 \dots d_k$ adresse destination

1^{er} ligne : permutation cyclique à gauche

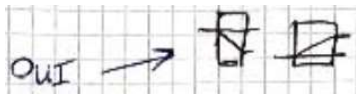


Haut
 Bas
 Shuffle
 Sortie commutation

Cela montre qu'il faut un nombre k d'étage (nombre de bit d'adresse) $= \log_2(N)$
 Réseau oméga reste bloquant car un choix de chemin empêche certains autres d'être réalisés.

Latence $= O(\log_2(N))$

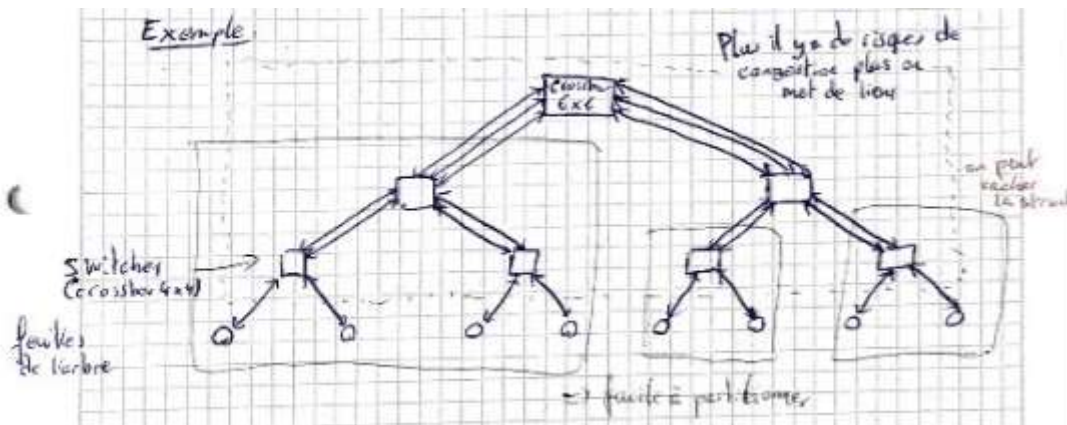
Comment faire un broadcast ? Il faut broadcast sur commutateurs élémentaires



Prix : $(n/2 \text{ par étage}) * \log_2(N) \text{ (nb étage)} = O(N * \log(N))$

Scalabilité : il faut tout recablé car entre N et $2N$, le shuffle est différent. On ne peut pas prendre 2 oméga de taille N pour en faire un de taille $2N$

FATTREE : Un type de réseau multi-étage couramment utilisée :



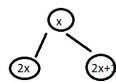
Chaque nœud de l'arbre est un commutateur.

Feuille : input et output

Arbre gras car branches s'épaississent à mesure que l'on monte vers la racine, ou la bande passante est de plus en plus nécessaire.

Réseau qui permet aussi de facilement créer de partitions de processeurs indépendants les uns des autres.

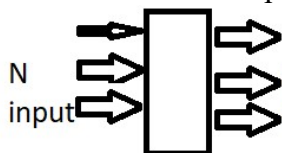
Routage dans un arbre binaire :



(en binaire : valeur binaire : deux enfants : valeur parent suivi de 0 ou 1)

Pour aller d'un nœud à un autre (12 à 7), il faut remonter jusqu'à ancêtre commun, (même préfixe) et on redescend selon la valeur.

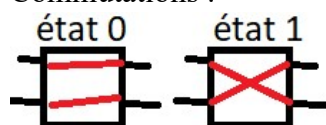
■ Crossbar : cher mais performant



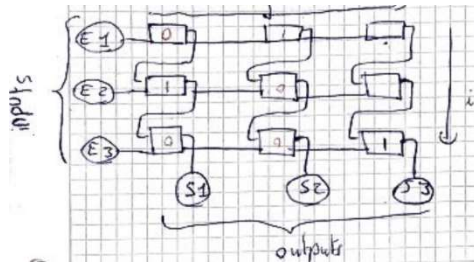
Il peut relier les input aux outputs : toutes les permutations de N entrées sur les N sortie sont réalisables.

Si permutation $i \leftrightarrow j$, pas de conflits entre chemin. $N!$ pattern de connexions. Mais il ne peut pas avoir deux entrées sur la même sortie.

Commutations :



Exemple d'un crossbar 3×3



On peut ainsi faire toutes les permutations. Il n'y aura jamais deux commutateurs de la même colonne dans l'état 1 car deux entiers ne peuvent pas se connecter à la même sortie.

Dans un vrai, il faut ajouter gestion de conflits et files d'attente, donc coût plus élevée.

Désavantage : prix croît comme N^2 puisqu'il contient

$N \times N$ éléments. Solution courante pour relier cœurs d'un proc multicœur à mémoire.

Il y a eu des 64×64 crossbar

Chap4

Puissance du proc. : R (vitesse)

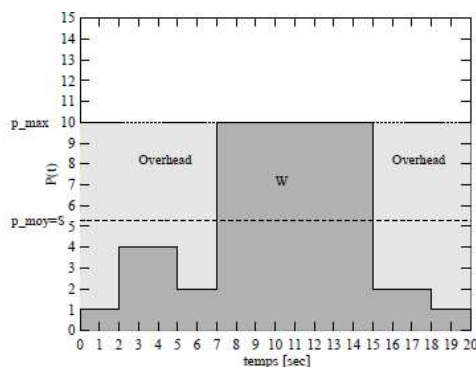
Travail : W (travail nécessaire pour résoudre un problème).

Degré de parallélisme $p(t)$: nombre de proc. Actifs au temps t

$P = p_{\max}$, max de proc utilisé

T_{seq} , T_{par}

$\Delta W = R \times \Delta T$: travail fait par 1 proc. Pendant ΔT est égal ΔW .



$$W = \int_{t_{\min}}^{t_{\max}} p(t) R dt$$

t_{\min} : nb proc actifs
 t_{\max} : nb d'instructions résolues par proc dt

$T_{\text{seq}} = W/R$

$T_{\text{par}} = T_{\text{end}} - T_{\text{start}}$

Speedup = $T_{\text{seq}}/T_{\text{par}}$:

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}} = \frac{(W/R)}{t_{\max} - t_{\min}} = \frac{1}{(t_{\max} - t_{\min})} \int_{t_{\min}}^{t_{\max}} P(\tau) d\tau$$

$= P_{\text{moyen}}$ = degré de parallélisme moyen

Algo séquentiel : faut prendre le meilleur

Efficacité : $E = S/P = T_{\text{seq}}/(p \times T_{\text{par}})$, idéalement on aimerait 1

Amdhal : α pourcent du code non parallélisable.

Vision pessimiste

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

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

Gustafson : fraction β du temps parallèle

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

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

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

Amdhal : essaye de parallélise un travail donné

Gustafon : augmenter travail proportionnellement à P

Parallélisme : pas uniquement aller vite mais aussi bonne façon de traiter problèmes plus grand.

Weak scaling : augmenter W de sorte que le travail par proc soit constant

Strong scaling : on garde W constant et on augmente p

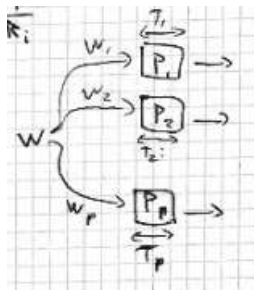
Speedup superlinéaire : $S > P$:

- 1) Matériel : problème trop grand pour rentrer en mémoire cache d'un proc, donc plus de proc, on rentre : plus rapide qu'en séquentiel.

Système hétérogènes

Si proc différent. On choisit le plus lent pour calculer speedup.

Soit W, le travail total, divisé en p morceaux, W_i



Idéalement, on aimerait ça.

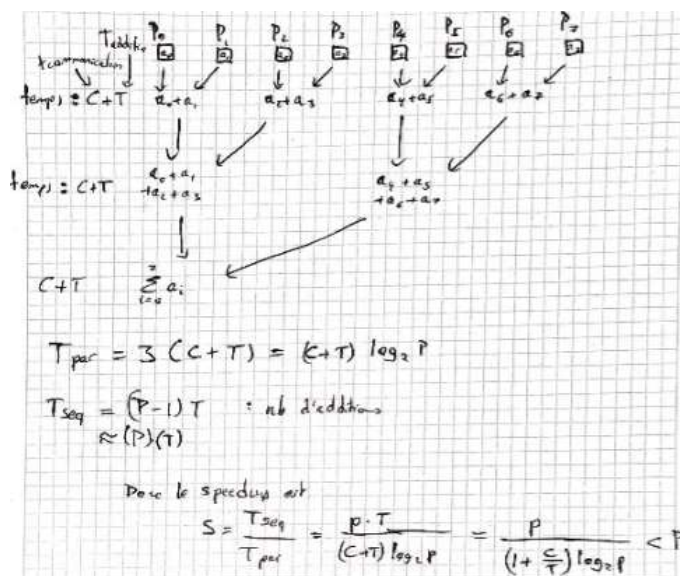
Si T_i plus court que T_j , p_j donne à p_i

On peut alors définir l'efficacité $E = T_{opt}/T_{par} = W/\text{somme}(R_i) * 1/T_{par}$
 $= W/(P * R_{mag} * T_{par}) = T_{seq}/p T_{par}$ ou $T_{seq} = W/R_{mag}$

Chap5

Sommer n nombres répartis sur p processeurs

1^{er} cas : $n = p$.

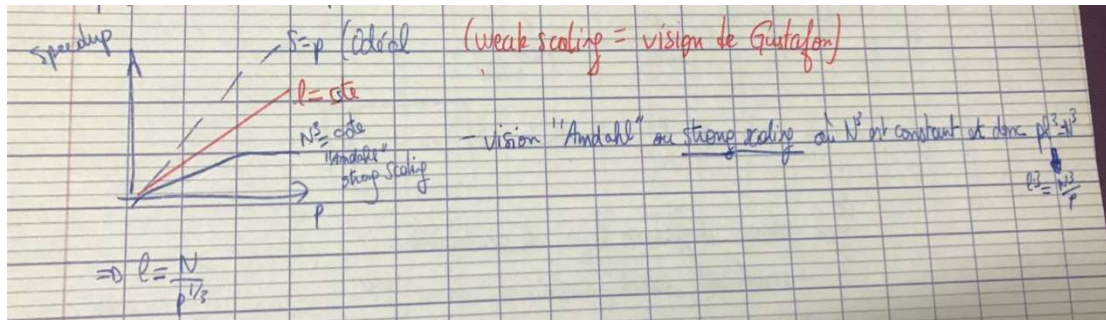


Problème : bcp de proc inutilisés après. $T_{com} > T_{cal}$

Car moins en moins de proc : cycles perdu

2^{ème} cas : $n \gg p$, p divise n

Maivaise algo : répété n/p fois l'algo précédent avec 1 val. Faut faire somme locale



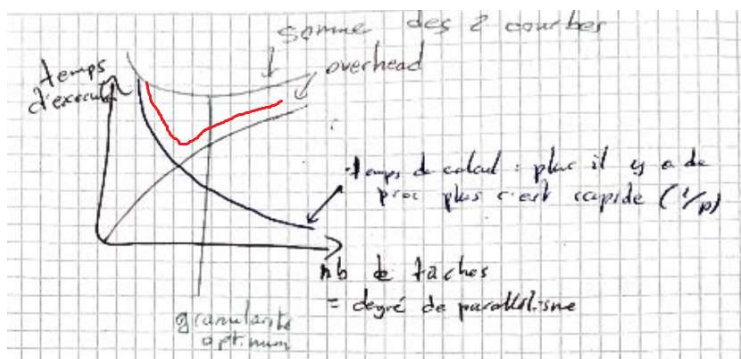
Fonction d'iso-efficacité : donne taille n du problème qu'il faut en fonction de p pour garder efficacité $E = S/p$ constante.

Cela donne différents type de scalabilité :

- Scalabilité idéal/linéaire : $n = \gamma * p \rightarrow$ favorable au parallélisme
- Scalabilité polylogarithmique : $n = \gamma * p * \log^k(p)$ $k \geq 1 \rightarrow$ favorable au parallélisme
- Scalabilité faible : $n = \gamma * p^k$ $k > 1$
- Non-scalabilité : fonction d'iso-efficacité n'existe pas.

Chap6

Si deux tâches, il y a un ordonnancement $T_i \rightarrow T_j$



Il faut trouver placement qui minimise overhead

Scheduling : temps auquel chaque tâche peut s'exécuter sur le proc. Qui la détient de sorte à ne pas violer la dépendance. Méthode des temps au plus tôt et au plus tard.

Tâche	T1	T2	T3	T4	T5	T6	T7	T8	T9
Au Plus tot	0	1	1	3	3	2	7	7	5 (t3), 11 (t8), 14 (t7)

Au plus tard	0	1	10	3	3 (T2)	11	7	10	14 (temps optimum de au plut tôt)
--------------	---	---	----	---	--------	----	---	----	-----------------------------------

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
P1	T1	T2	T2	T4	T4	T4	T7	T7	T7	T7	T7	T7	T7	T7	T9
P2			T3	T5	T5	T5	T5	T6	T6	T6	T8	T8	T8		

2 proc suffisent et rentabilité max. Mais pas 100%

Equilibrage de charge

Statique : on peut estimer charge à l'avance. En utilisant des algos qui découpe en morceaux de travail égal. (METIS pour partitionnement de graphe).

- Bisection récursive
- Partitionnement de graphe
- Découpage cyclique ou modulo

Dynamique : pour problème que l'on peut pas estimer (Ensemble de Julia)

Il faut :

- Détecter déséquilibre (mesurer charge au cours d'exécution)
- Décider : faut-il rééquilibrer les charges ? est-ce-que ça vaut la peine ? Est-ce que tps de répartition se justifie ?
- Migrer tâche si on pense que rééquilibrage nécessaire.

2 types d'applications :

- **Problèmes itératifs**: même calcul répété sur plusieurs étapes, et tout les proc. Doivent se synchro à chaque étape.
Processus de **loadbalancing** sera activé après chaque itération. Une itération est un bon prédicteur de la charge de la prochaine itération.
- **Problèmes non-itératifs** : chaque donnée du problème n'est calculé qu'une seule fois (ensemble de Julia).

Approche centralisé : liste avec travail restant dans espace mémoire. Et on demande quand on a fini. (version locale : un proc qui demande travaille à un autre proc. De groupe.)

Chap7

Il y a plusieurs modèles principaux de programmation parallèle :

- Echange de message (MPI, PVM, ...)
- Multithreading/sharedMemory (thread PQSIX, Open MP)
- Data parallelism (HPF) : structure de données parallèles (ex : vecteurs), flot d'instruction séquentielle mais instruction qui git sur des structures de données parallèles.

Le modèle par échange de message :

- Coopération entre processeur se fait par des communications
- Le problème doit être partitionné sur chaque processeur

- Pas d'espace mémoire globale, espace fragmenté sur les processeurs
- Modèle « do it yourself » mais assez bien contrôlé par l'utilisateur
- Il faut changer sa façon de voir le problème, penser « parallèle »
- Modèle clair et on sait ce qui se passe
- Solution scalable (→ millions de cœurs)
- Matériel pas excessivement rendu compliqué par le parallélisme
- Coordination naturelle entre processeur par l'envoi/réception de message (on ne peut pas recevoir un message avant qu'il n'ait été envoyé, pas de donnée partagée)

Thread : coopération par modification de valeur partagé. Proche de séquentiel la programmation. Coordination pour éviter race condition (ressources)

Contrôle de séquence : il faut empêcher un proc. D'aller trop vite et de commencer à utiliser des variables qui n'ont pas encore été mises à jour par d'autres processeurs. Avec barrière de synchro

Primitives de synchronisation : barriere, sémaphore, test and set, compare and swap

Fetch and add

Pour :

- Equilibrage de charge dynamique
- Section critique
- Barrière de synchronisation
- Historique de PC ?
- Différence entre système parallèle et système répartis ?
- Expliqués différents types de truc que l'on a vu ? (séquentiel, Pipeline, SIMD, MIMD, SPMD)
- Qu'est ce qu'un réseau systolique ?
- Expliquer le NUMA et le COMA ?
- A quoi sert l'architecture vectoriels ? Quels sont les avantages et les inconvénients ?
- Explique les deux types de dataflow ? (datadriven et demand driver)
- Topologie statique et dynamique ?
- Décrivez les 6 propriétés d'un réseau d'interconnexion ?
- Décrivez technique de communication (store and forward, cut through)
- Décrivez les primitives de communications (point à point, permutation, broadcast, réduction, gather, échange total, échange total personnalisé)
- Réseaux statiques : que pouvez vous me dire ? (anneaux, grille 2D, 3D, Hypercube)
- Catégories de réseaux dynamiques ? (connexion par bus, multi étage (omega, fattree), crossbar)
-
- Weak scaling : augmenter nombre itérations
- Strong scaling : augmenter le nombre de CPU
-
- Qu'est ce que l'overhead ?
- Qu'est-ce-qu'un speedup ? Quel est la différence entre Amdhal et Gustafon ?
- Weak scaling ? strong scaling ?
- Qu'est ce qu'un speedup superlinéaire ? Pourquoi observe-t-on cela ?

- Qu'est ce qu'un système hétérogène ?
- Présentez les systèmes d'équilibrage de charge ? (statique, dynamique)
- Il y a deux types d'applications. Que sont-ils ? Quels sont les contraintes ?
- Qu'est ce que l'approche centralisé pour l'équilibrage de charge dynamique ?
Différence entre ça et la version locale ?
- Citez moi quelques modèles principaux de programmation parallèle ?
- Donner les caractéristiques du modèle par échange de message ?
- Qu'est ce que le contrôle de séquence ?