

P-RAM et réseaux linéaires de processeurs

1 Composantes connexes sur une P-RAM

On souhaite concevoir un algorithme CREW qui permette de calculer les composantes connexes d'un graphe $G = (V, E)$ dont les sommets sont numérotés de 1 à n . Plus précisément, on cherche un algorithme qui renvoie un tableau C de taille n tel que $C(i) = C(j) = k$ si et seulement si i et j sont dans la même composante connexe et k est le plus petit indice des sommets de cette composante.

Définition 1. À toute étape de l'algorithme, on appellera pseudo-sommet étiqueté par i l'ensemble de sommets $j, k, l, \dots \in V$ tels que $C(j) = C(k) = C(l) = \dots = i$. On assimilera le pseudo-sommet i étiqueté par i au sommet étiqueté par i .

Un des invariants de l'algorithme est que le plus petit indice des sommets constituant un pseudo-sommet étiqueté par i est i et que les sommets appartenant à un pseudo-sommet sont dans la même composante connexe. Cette assertion est donc vraie si on initialise C par : pour tout $i \in V = \llbracket 1, n \rrbracket$: $C(i) = i$. Ceci signifie que chaque processeur se considère au départ comme sommet de référence de sa composante connexe. L'objectif de l'algorithme est de modifier ce point de vue égocentrique.

Définition 2. Une arborescence k -cyclique ($k \geq 0$) est un graphe orienté faiblement connexe (c'est-à-dire tel que le graphe non orienté sous-jacent est connexe) tel que :

- tout sommet a un degré sortant égal à 1 et
- il existe exactement un circuit de longueur $k + 1$.

On appelle étoile une arborescence 0-cyclique dans laquelle toutes les arêtes sont incidentes à la racine et l'indice de la racine est le plus petit indice dans l'étoile.

L'invariant précédent est donc que le graphe orienté $(V, \{(i, C(i)) \mid i \in V\})$ est constitué d'étoiles. On peut donc identifier pseudo-sommets et étoiles, le centre de l'étoile étant l'indice du pseudo-sommet. Le calcul des composantes connexes s'effectue en enchaînant plusieurs fois de suite les deux fonctions suivantes :

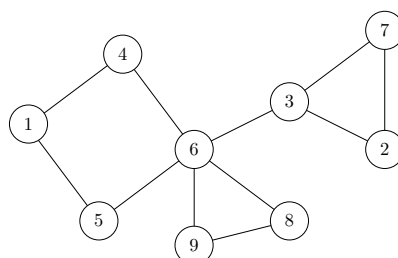
```

GATHER()
1: Pour tout  $i \in V$  en parallèle :
2:    $T(i) \leftarrow \min \{C(j) \mid \{i, j\} \in E, C(j) \neq C(i)\}$ 
   {si l'ensemble est vide, on associe  $C(i)$ }
3: Pour tout  $i \in V$  en parallèle :
4:    $T(i) \leftarrow \min \{T(j) \mid C(j) = i, T(j) \neq i\}$ 
   {si l'ensemble est vide, on associe  $C(i)$ }

JUMP()
5: Pour tout  $i \in V$  en parallèle :
6:    $B(i) \leftarrow T(i)$ 
7: Répéter  $\log n$  fois
8:   Pour tout  $i \in V$  en parallèle :
9:      $T(i) \leftarrow T(T(i))$ 
10: Pour tout  $i \in V$  en parallèle :
11:    $C(i) \leftarrow \min \{B(T(i)), T(i)\}$ 

```

▷ **Question 1** On considère le graphe suivant.



Appliquer la fonction `GATHER` sur ce graphe, puis la fonction `JUMP`, puis la fonction `GATHER`, et ainsi de suite. Il sera instructif d'observer l'effet des opérations sur les graphes orientés $(V, \{(i, T(i)) \mid i \in V\})$ et $(V, \{(i, C(i)) \mid i \in V\})$.

▷ **Question 2** Montrer qu'après l'application de la fonction `GATHER`, les composantes connexes contenant plusieurs pseudo-sommets induisent des arborescences 1-cycliques dans le graphe orienté $(V, \{(i, T(i)) \mid i \in V\})$. On notera également que le plus petit pseudo-sommet d'une arborescence 1-cyclique appartient au cycle.

▷ **Question 3** Montrer que la fonction `JUMP` transforme une arborescence 1-cyclique en étoile (ou pseudo-sommet).

▷ **Question 4** Montrer qu'après $\lceil \log n \rceil$ enchaînements des fonctions `GATHER` et `JUMP`, les composantes connexes du graphe sont représentées par les pseudo-sommets induits par `C`.

▷ **Question 5** Quelle est la complexité de l'algorithme ? Combien de processeurs sont utilisés ?

2 Réseaux linéaires - Rotations de Givens

Pour triangulariser une matrice A d'ordre n de façon numériquement stable, on peut utiliser les rotations de Givens. L'opération de base $\text{ROT}(i, j, k)$ consiste à combiner les deux lignes i et j , qui doivent toutes deux commencer par $k - 1$ zéros, pour annuler l'élément en position (j, k) :

$$\begin{pmatrix} 0 & \dots & 0 & \mathbf{a}'_{i,k} & a'_{i,k+1} & \dots & a'_{i,n} \\ 0 & \dots & 0 & \mathbf{0} & a'_{j,k+1} & \dots & a'_{j,n} \end{pmatrix} \leftarrow \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & \mathbf{a}_{i,k} & a_{i,k+1} & \dots & a_{i,n} \\ 0 & \dots & 0 & \mathbf{a}_{j,k} & a_{j,k+1} & \dots & a_{j,n} \end{pmatrix}$$

Nous laissons au lecteur le soin de déterminer l'angle θ permettant d'effectuer cette opération. :-)

L'algorithme séquentiel peut s'écrire :

```
GIVENS(A)
1:  Pour k = 1 to n - 1 :
2:     Pour i = n downto k + 1 step -1 :
3:         ROT(i - 1, i, k)
```

On considère qu'une rotation $\text{ROT}(i, j, k)$ s'exécute en temps unité, indépendamment de k .

▷ **Question 6** Mettre en œuvre cet algorithme sur un réseau linéaire de n processeurs.

▷ **Question 7** Mettre en œuvre cet algorithme sur un réseau linéaire comportant seulement $\lfloor \frac{n}{2} \rfloor$ processeurs.

3 Facteur d'accélération

On discute diverses lois (Amdahl, Gustafson) sur le facteur d'accélération et l'efficacité. Ces lois font désormais partie du bagage de tout honnête paralléliseur !

▷ **Question 8** Soit un problème à résoudre, qui comporte un pourcentage f d'opérations intrinsèquement séquentielles. Montrer que le facteur d'accélération est limité par $1/f$, quel que soit le nombre de processeurs utilisés. Quelle leçon en tirer pour la parallélisation d'un problème de taille fixe ?

▷ **Question 9** Pour un problème matriciel de taille $n \times n$, on suppose que :

- le nombre d'opérations arithmétiques à exécuter est n^α , où α est une constante ;
- le nombre d'éléments à stocker en mémoire est $w_1 n^2$, où w_1 est une constante ;
- le nombre d'opérations d'entrées-sorties (purement séquentielles) est $w_2 n^2$, où w_2 est une constante.

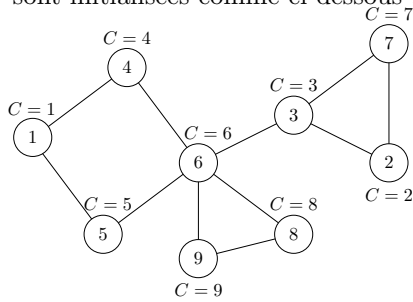
Comment estimer l'accélération obtenue avec p processeurs sur un problème de grande taille ? Quelle leçon en tirer pour la parallélisation d'un problème de taille variable ?

▷ **Question 10** Donner des exemples de facteur d'accélération superlinéaire, c'est-à-dire d'efficacité strictement supérieure à 1.

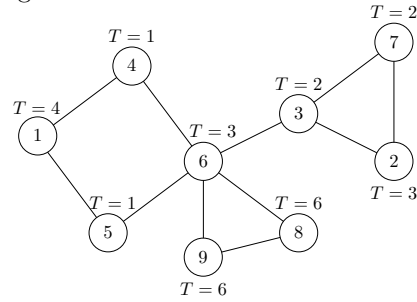
4 Réponses aux exercices

▷ Question 1, page 1

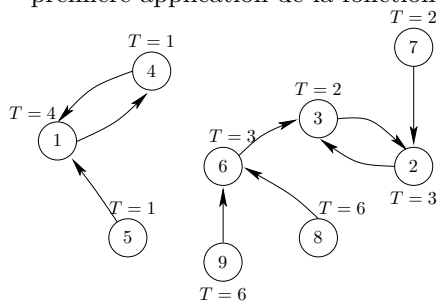
Avant le début de l'algorithme, les valeurs de C sont initialisées comme ci-dessous :



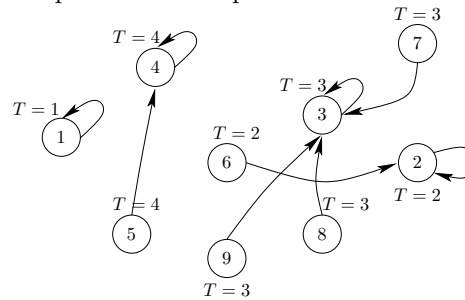
Après l'application de la fonction GATHER, les valeurs de C sont inchangées et celles de T sont les suivantes :



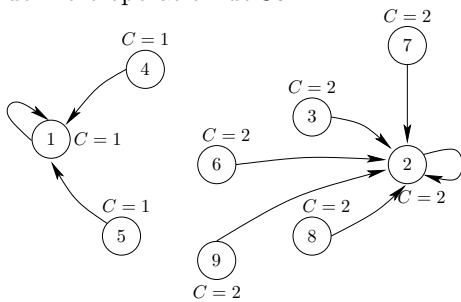
Graphe orienté $(V, \{(i, T(i)) \mid i \in V\})$ après la première application de la fonction GATHER :



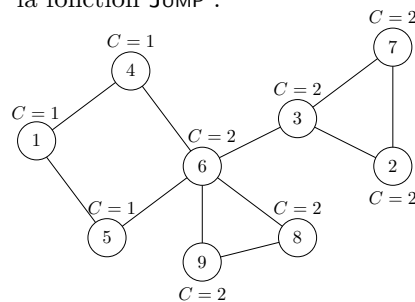
Après le saut de pointeur de la fonction JUMP :



Le graphe $(V, \{(i, C(i)) \mid i \in V\})$ après de la dernière opération de JUMP :

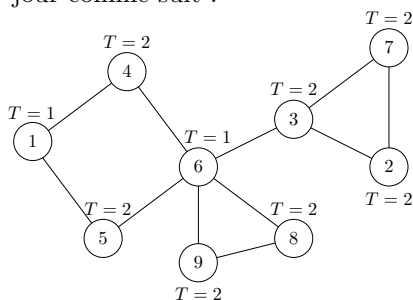


Il ne reste plus que deux pseudo-sommets. On se retrouve dans la situation suivante à la fin de la fonction JUMP :

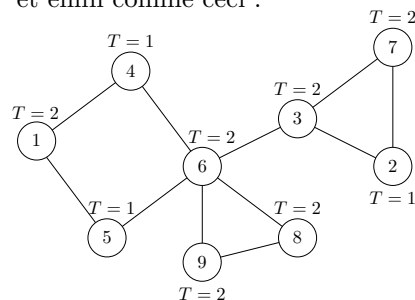


(fin de la première itération GATHER + JUMP)

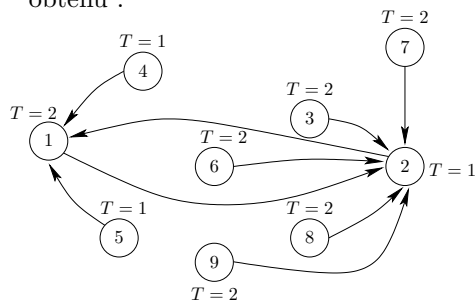
Après la première étape de GATHER, T est mis à jour comme suit :



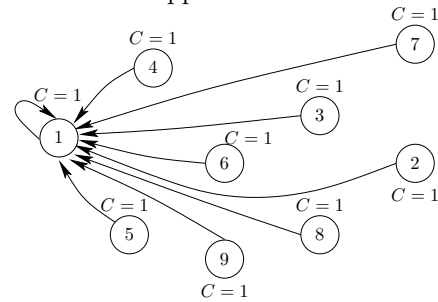
et enfin comme ceci :



Voici le graphe orienté $(V, \{(i, T(i)) \mid i \in V\})$ obtenu :



Les pseudo-sommets 1 et 2 sont donc fusionnés à la fin de l'appel à JUMP :



On a bien calculé les composantes connexes du graphe.

▷ **Question 2, page 2**

Tout d'abord, il est clair que lorsqu'une composante connexe ne contient qu'un seul pseudo-sommet, l'étoile correspondante est transférée dans T sans modification.

Si une composante connexe contient plusieurs pseudo-sommets, par contre, T décrira un ensemble d'arborescences 1-cycliques contenues dans cette composante. En effet, tout pseudo-sommet de cette composante contient au moins un sommet adjacent à un sommet d'un autre pseudo-sommet. GATHER fait pointer le représentant de chaque pseudo-sommet – *via* T – vers le représentant d'un autre pseudo-sommet, tout en laissant pointer les autres sommets vers leur pseudo-sommet initial. En clair, si deux groupes se touchent, leurs représentants se retrouvent liés dans le graphe orienté induit par T et les autres sommets continuent de pointer vers leur représentant respectif. Chaque composante de ce graphe orienté doit contenir au moins une boucle car le degré sortant de chaque sommet vaut 1. Il y a au plus une boucle car sinon il y aurait un pseudo-sommet avec deux valeurs pour T . Enfin la boucle en question ne peut être que de longueur 2, sinon (si elle était de longueur 1) i et $T(i)$ seraient identiques ou (si elle était de longueur supérieure à 2) il existerait un sommet i sur la boucle tel que $T(i)$ n'est pas le plus petit indice des pseudo-sommets adjacents au pseudo-sommet i .

▷ **Question 3, page 2**

Cette étape fusionne tous les sommets d'une même arborescence 1-cyclique en une étoile indexée par le sommet de plus petit indice en utilisant la technique de saut de pointeur. En effet, étant donné la configuration d'une étoile, à l'issue des sauts de pointeur, chaque sommet a pour valeur de T l'une des anciennes valeurs d'un des sommets de la boucle. La dernière étape permet donc d'assigner à tous les sommets la plus petite valeur des sommets de l'arborescence à laquelle ils appartiennent.

▷ **Question 4, page 2**

Il suffit de montrer que le nombre de pseudo-sommets diminue au moins de moitié à chaque étape. Concentrons-nous sur les représentants des pseudo-sommets et intéressons-nous au graphe induit par T sur ces sommets. Dans un tel graphe, deux pseudo-sommets i et j sont connectés si, et seulement si, il existe deux sommets k et l connectés dans le graphe initial et tels que $C(k) = i$ et $C(l) = j$. Dans l'exemple précédent, cela revient à avoir un graphe composé des pseudo-sommets 1 et 2 reliés par une arête après la première application de GATHER. La fonction JUMP fusionne tous les pseudo-sommets ainsi reliés en un seul pseudo-sommet. Ainsi le nombre de pseudo-sommets dans une même composante connexe diminue au moins de moitié à chaque étape et $\lceil \log n \rceil$ enchaînements de GATHER et JUMP suffisent à calculer les composantes connexes.

▷ **Question 5, page 2**

Premièrement, on peut remarquer que la boucle séquentielle de la fonction JUMP implique que le temps de calcul total est au moins $O(\log^2 n)$, et ce quel que soit le nombre de processeurs. On va d'abord montrer que ce temps peut être atteint avec $O(n^2)$ processeurs.

On peut déjà remarquer qu'avec autant de processeurs, la première et la dernière boucle de JUMP prennent un temps $O(1)$ et le saut de pointeur un temps $O(\log n)$. En fait, $O(n)$ processeurs suffisent

pour arriver à un tel temps de calcul de la fonction JUMP. Si on veut arriver à un temps total de l'ordre de $O(\log^2 n)$, on va donc devoir montrer que la fonction GATHER peut s'exécuter en temps $O(\log n)$.

Le calcul du maximum de n valeurs peut se faire en temps $O(1)$ avec n^2 processeurs. Cependant, ce sont les maxima de plusieurs ensembles que l'on veut calculer et il faut raffiner donc l'approche

```

1:  Pour tout  $i \in V$  en parallèle :
2:     $T(i) \leftarrow \min \{C(j) \mid \{i, j\} \in E, C(j) \neq C(i)\}$ 
      {si l'ensemble est vide, on associe  $C(i)$ }

La boucle précédente peut être transformée en le code suivant

3:  Pour tout  $i, j \in V$  en parallèle :
4:    Si  $\{i, j\} \in E$  And  $C(i) \neq C(j)$  Alors
       $Temp(i, j) \leftarrow C(j)$ 
5:    Sinon  $Temp(i, j) \leftarrow \infty$ 
6:  Pour tout  $i \in V$  en parallèle :
7:     $Temp(i, 1) \leftarrow \min \{Temp(i, j) \mid j \in V\}$ 
8:  Pour tout  $i \in V$  en parallèle :
9:    Si  $Temp(i, 1) = \infty$  Alors  $T(i) \leftarrow C(i)$ 
10:   Sinon  $T(i) \leftarrow Temp(i, 1)$ 
    
```

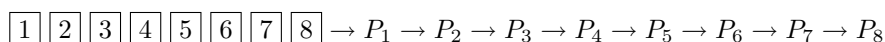
Algorithme 1: Précision sur la mise en œuvre des calculs de maxima.

La première boucle se fait clairement en temps $O(1)$ avec $O(n^2)$ processeurs (en réalité avec $O(|E|)$ processeurs) sur une CREW. Les deux boucles suivantes se font en temps $O(\log n)$ avec $O(n^2)$ processeurs en divisant pour régner.

Le calcul des composantes connexes s'effectue donc bien en temps $O(\log^2 |V|)$ avec $O(|V| + |E|)$ processeurs. Cependant, la fonction JUMP gaspille des ressources (le «diviser pour régner» est trop gourmand en ressources). Les calculs de minima peuvent également être optimisés en terme de ressources. Le théorème de Brent nous permet donc de diminuer le nombre de processeurs à $O\left(\frac{n^2}{\log n}\right)$ (en réalité à $O\left(\frac{|E|}{\log |V|} + |V|\right)$ processeurs) sans modifier le temps de calcul.

▷ **Question 6, page 2**

La mise en œuvre avec n processeurs numérotés P_1 à P_n est aisée : le processeur P_k va être responsable de toutes les rotations $ROT(i-1, i, k)$. On a en entrée du réseau (où \boxed{i} représente la ligne i de la matrice) :



La première ligne qui arrive dans un processeur s'y installe pour un top. Quand une autre ligne arrive, elle est combinée avec la ligne en mémoire, puis prend sa place. La ligne qui était en mémoire poursuit sa route vers la droite. On obtient le schéma d'exécution suivant pour $n = 8$:

Top	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
$t = 1$	$\boxed{8}$							
$t = 2$	$ROT(7,8,1)$							
$t = 3$	$ROT(6,7,1)$	$\boxed{8}$						
$t = 4$	$ROT(5,6,1)$	$ROT(7,8,2)$						
$t = 5$	$ROT(4,5,1)$	$ROT(6,7,2)$	$\boxed{8}$					
$t = 6$	$ROT(3,4,1)$	$ROT(5,6,2)$	$ROT(7,8,3)$					
$t = 7$	$ROT(2,3,1)$	$ROT(4,5,2)$	$ROT(6,7,3)$	$\boxed{8}$				
$t = 8$	$ROT(1,2,1)$	$ROT(3,4,2)$	$ROT(5,6,3)$	$ROT(7,8,4)$				
$t = 9$	$\boxed{1}$	$ROT(2,3,2)$	$ROT(4,5,3)$	$ROT(6,7,4)$	$\boxed{8}$			
$t = 10$	$\boxed{1}$	$\boxed{2}$	$ROT(3,4,3)$	$ROT(5,6,4)$	$ROT(7,8,5)$			
$t = 11$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$ROT(4,5,4)$	$ROT(6,7,5)$	$\boxed{8}$		
$t = 12$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$\boxed{4}$	$ROT(5,6,5)$	$ROT(7,8,6)$		
$t = 13$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$\boxed{4}$	$\boxed{5}$	$ROT(6,7,6)$	$\boxed{8}$	
$t = 14$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$\boxed{4}$	$\boxed{5}$	$\boxed{6}$	$ROT(7,8,7)$	
$t = 15$	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$\boxed{4}$	$\boxed{5}$	$\boxed{6}$	$\boxed{7}$	8

Au top $2n - 1$, le processeur P_k contient la ligne k . Il faudrait éventuellement vider le réseau.

▷ **Question 7, page 2**

On voit bien que le réseau précédent peut être replié, pour obtenir un réseau bidirectionnel :

$$\boxed{1} \boxed{2} \boxed{3} \boxed{4} \boxed{5} \boxed{6} \boxed{7} \boxed{8} \Leftrightarrow P_1 \Leftrightarrow P_2 \Leftrightarrow P_3 \Leftrightarrow P_4 .$$

Au top n , on reverse le sens de circulation, pour obtenir sur l'exemple avec $n = 8$:

Top	P_1	P_2	P_3	P_4
$t = 1$	$\boxed{8}$			
$t = 2$	Rot(7,8,1)			
$t = 3$	Rot(6,7,1)	$\boxed{8}$		
$t = 4$	Rot(5,6,1)	Rot(7,8,2)		
$t = 5$	Rot(4,5,1)	Rot(6,7,2)	$\boxed{8}$	
$t = 6$	Rot(3,4,1)	Rot(5,6,2)	Rot(7,8,3)	
$t = 7$	Rot(2,3,1)	Rot(4,5,2)	Rot(6,7,3)	$\boxed{8}$
$t = 8$	Rot(1,2,1)	Rot(3,4,2)	Rot(5,6,3)	Rot(7,8,4)
$t = 9$	Rot(2,3,2)	Rot(4,5,3)	Rot(6,7,4)	$\boxed{8}$
$t = 10$	Rot(3,4,3)	Rot(5,6,4)	Rot(7,8,5)	
$t = 11$	Rot(4,5,4)	Rot(6,7,5)	$\boxed{8}$	
$t = 12$	Rot(5,6,5)	Rot(7,8,6)		
$t = 13$	Rot(6,7,6)	$\boxed{8}$		
$t = 14$	Rot(7,8,7)			
$t = 15$	$\boxed{8}$			

La première ligne ressort du réseau par la gauche au top $t = n + 1$, et la dernière ligne ressort au top $2n$.

▷ **Question 8, page 2**

Soit T_1 le temps d'une résolution avec un processeur, décomposé en $T_1 = T_{\text{seq}} + T_{\text{par}}$, où T_{seq} est la partie séquentielle et T_{par} la partie parallélisable. Par hypothèse, $T_{\text{seq}} = f \cdot T_1$. Avec p processeurs, le temps nécessaire pour exécuter la partie parallélisable sera, au mieux, divisé par p (avec un parallélisme parfait, sans aucun surcoût). Le temps d'exécution total T_p avec p processeurs vérifie donc :

$$T_p \geq T_{\text{seq}} + \frac{T_{\text{par}}}{p} = f \cdot T_1 + \frac{(1-f)T_1}{p}$$

et le facteur d'accélération est borné par :

$$S_p = \frac{T_1}{T_p} \leq \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}$$

Ainsi, si $f = 5 \%$, le facteur d'accélération est limité à 20, même avec 1 000 processeurs. Pour un problème de taille fixe, le degré de parallélisme est borné indépendamment du nombre de processeurs. C'est la loi d'Amdahl [?], qui décourage le recours au parallélisme massif.

▷ **Question 9, page 2**

Le temps d'exécution séquentiel pour un problème de taille n est $T_1(n) = n^\alpha \tau_a + w_2 n^2 \tau_{e/s}$, où τ_a et $\tau_{e/s}$ sont des paramètres machine. La taille maximale de résolution avec 1 processeur vérifie $w_1 (n_{\text{max}}(1))^2 = M$, où M est la mémoire disponible (en nombre d'éléments) sur un processeur.

Avec p processeurs, on peut résoudre un problème plus grand. Il y a p mémoires de taille M , donc $n_{\text{max}}(p) = \sqrt{p} \cdot n_{\text{max}}(1)$. Comment calculer le facteur d'accélération pour un problème de taille $n_{\text{max}}(p)$, trop gros pour être exécuté avec un seul processeur ? L'idée est simple : on normalise, i.e. on calcule $A(p)$, le temps moyen d'une opération arithmétique avec p processeurs pour un problème de taille maximale $n_{\text{max}}(p)$. La nouvelle définition du facteur d'accélération (selon Gustafson [?]) est $S_p = \frac{A(1)}{A(p)}$.

Pour notre problème matriciel, en supposant une parallélisation parfaite des calculs :

$$A(1) = \frac{n^\alpha \tau_a + w_2 n^2 \tau_{e/s}}{n^\alpha} \quad \text{avec } w_1 n^2 = M$$

$$A(p) = \frac{\frac{n^\alpha \tau_a}{p} + w_2 n^2 \tau_{e/s}}{n^\alpha} \quad \text{avec } w_1 n^2 = pM$$

Si $\alpha = 2$, $A(1) = \tau_a + w_2\tau_{e/s}$ et $A(p) = \frac{\tau_a}{p} + w_2\tau_{e/s}$, on retrouve un facteur d'accélération borné indépendamment de p . Mais pour $\alpha \geq 3$, on a

$$A(1) = \tau_a + \frac{w_2\tau_{e/s}}{\left(\frac{M}{w_1}\right)^{\alpha-2}}$$

$$A(p) = \frac{\tau_a}{p} + \frac{w_2\tau_{e/s}}{\left(\frac{M}{w_1}\right)^{\alpha-2} p^{\alpha-2}}$$

Le facteur d'accélération croît linéairement avec p . Moralité, le parallélisme massif reste utile, mais pour résoudre des problèmes de grande taille, dont la fraction intrinsèquement séquentielle tend vers 0. La plupart des calculs matriciels satisfont à ces deux exigences. Ouf!

▷ Question 10, page 2

On peut donner plusieurs types de réponses :

- **Algorithmique.** L'algorithme utilisé pour résoudre le problème peut converger plus de p fois plus vite avec p processeurs qu'avec un seul.

Un premier exemple tout bête : l'exécution d'une boucle de taille 100 pour additionner deux vecteurs nécessite un temps égal au calcul de 100 additions, auquel s'ajoute le temps de contrôle de la boucle. Avec 100 processeurs, une seule addition par processeur, mais plus de contrôle, on va donc plus de 100 fois plus vite. Bien sûr, on aurait pu dérouler toute la boucle pour un seul processeur et supprimer aussi le contrôle.

Voici deux exemples plus sérieux. Le premier concerne les algorithmes pour la recherche *branch-and-bound* d'une solution satisfaisant un certain critère. Quand on explore en parallèle l'arbre de recherche, le processeur numéro 12 peut tomber immédiatement sur une solution, indépendamment du temps passé en séquentiel, d'où une accélération superlinéaire. C'est aussi le cas pour la résolution d'un système linéaire creux par une méthode itérative : une matrice de préconditionnement découpée en blocs indépendants pour faciliter la mise en œuvre du parallélisme peut s'avérer meilleure que la matrice de préconditionnement de la méthode séquentielle usuelle. Il est vrai que le facteur d'accélération doit être calculé comme le rapport du temps avec le meilleur algorithme séquentiel sur le temps de l'algorithme parallèle, et qu'après coup on peut toujours simuler l'algorithme parallèle superefficace (dans tous les sens du terme!) avec un seul processeur. Mais, en général, on ne connaît pas le meilleur algorithme séquentiel, et le recours au parallélisme conduit à inventer de nouveaux algorithmes, auxquels on n'aurait jamais pensé (comme dans le dernier exemple) sur une machine séquentielle.

- **Matériel.** Il ne faut jamais oublier que p processeurs ont plus de mémoire qu'un seul. Ainsi pour effectuer un produit matrice-vecteur avec deux processeurs, il y a toujours une taille de problème bien choisie pour que les données distribuées (la demi-matrice et le vecteur) tiennent dans la mémoire vive de chaque processeur, mais que la donnée totale (toute la matrice) nécessite des accès disque avec un seul processeur : accélération superlinéaire garantie!

Pour conclure sur une note d'humour : un seul processeur/déménageur met un temps infini à transporter le piano, mais deux déménageurs ont besoin d'une petite heure : facteur d'accélération infini!