

Algorithmique et architectures parallèles
 Partiel du 10 novembre 2009
Et avec votre hypercube, qu'est-ce que j'vous sers?

(Durée: 2 heures)

Benjamin DEPARDON Anne BENOIT

1 Réseau hyper-Petersen

Vous avez étudié en cours l'hypercube. Nous allons maintenant étudier un nouveau réseau, le réseau *Hyper-Petersen*, basé sur deux types de graphes : les hypercubes et les graphes de Petersen.

1.1 Quelques définitions

Définition 1 (Graphe Petersen). *Un graphe Petersen est composé de 10 nœuds, numérotés à l'aide des couples suivants : $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{1, 5\}$, $\{2, 3\}$, $\{2, 4\}$, $\{2, 5\}$, $\{3, 4\}$, $\{3, 5\}$ et $\{4, 5\}$. Il existe un lien entre deux nœuds $\{i, j\}$ et $\{k, l\}$ si et seulement si les ensembles $\{i, j\}$ et $\{k, l\}$ sont disjoints. La Figure 1 présente un tel réseau.*

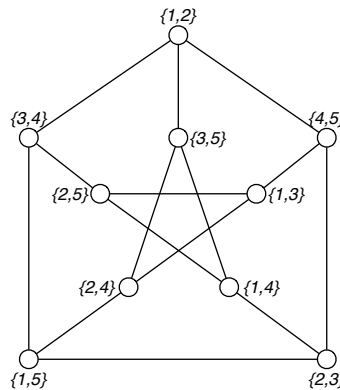


FIG. 1: Réseau Petersen

Notation 1. *Nous utiliserons les notations suivantes :*

- P : graphe de Petersen,
- Q_n : hypercube de dimension n ,
- K_n : graphe complet de n nœuds.

Définition 2 (Graphe Hyper-Petersen). *On définit un graphe Hyper-Petersen (HP_n) pour $n \geq 3$, comme étant le produit cartésien d'un hypercube de dimension $n-3$ et d'un graphe de Petersen P : $HP_n = Q_{n-3} \times P$.*

▷ **Question 1** *Dessinez un réseau $HP_4 = Q_1 \times P$ en expliquant votre méthode.*

▷ **Question 2** *En utilisant le produit cartésien, donnez une définition récursive du graphe HP_n .*

1.2 Propriétés topologiques

▷ **Question 3** *Donnez le degré, le diamètre et le nombre de nœuds d'un réseau HP_n .*

Soit κ la connectivité du réseau, *i.e.*, le nombre de nœuds qu'il faut enlever du réseau pour avoir un réseau déconnecté. La connectivité du réseau est une mesure de la résistance aux pannes d'un réseau.

▷ **Question 4** *Quelle est la connectivité κ d'un réseau HP_n ?*

▷ **Question 5** *Rappelez les caractéristiques d'un Hypercube Q_n (degré, diamètre, nombre de nœuds, connectivité). Quels avantages/inconvénients voyez vous entre un réseau Hyper-Petersen HP_n et un Hypercube Q_n ?*

1.3 Routage

Une topologie ne peut être efficace que s'il est possible de définir des algorithmes de communications point à point et de communications collectives efficaces.

▷ **Question 6** *Proposez un algorithme efficace de routage pour envoyer un message entre deux nœuds distincts du réseau. Combien d'étapes sont nécessaires à l'acheminement du message ?*

▷ **Question 7** *Écrivez un algorithme de broadcast. Combien d'étapes sont nécessaires ?*

1.4 Plongements

Nous donnons tout d'abord les définitions de *dilatation* d'un plongement, ainsi que de deux opérations pouvant être utile pour réaliser des plongements : la *fusion* et la *déconnexion*.

Définition 3 (Dilatation). *Lorsque l'on plonge un graphe G_1 dans un graphe G_2 , la dilatation d'une arête $e \in E(G_1)$ est la longueur du chemin $\psi(e)$ dans G_2 . La dilatation d'un plongement est le maximum pris sur l'ensemble des arêtes de G_1 .*

On s'autorise ainsi des plongements où une arête dans G_1 peut être remplacée par un chemin dans G_2 .

Définition 4 (Fusion et Déconnexion). *Nous nous autoriserons les deux opérations suivantes sur le graphe dans lequel le plongement est réalisé. Ces opérations permettent de modifier le graphe en fusionnant des nœuds, ou en enlevant des liens.*

*Soit u et v deux nœuds d'un graphe, on définit *fusion*(u, v) l'action d'enlever l'arête (u, v), et de fusionner les deux nœuds u et v en un seul nœud ayant pour voisins tous les voisins de u et de v . On définit également *deconnecte*(u, v) l'action consistant à enlever le lien (u, v). La Figure 2 présente ces deux opérations sur P .*

Attention, contrairement à tous les plongements que nous avons vu en TD qui étaient de dilatation 1, ici, si la dilatation n'est pas indiquée elle peut être quelconque.

▷ **Question 8** *Montrez qu'il existe un plongement d'un anneau de taille 1.25×2^n dans HP_n , de dilatation 1, pour tout $n \geq 4$.*

▷ **Question 9** *Expliquez comment réaliser le plongement d'une grille de taille 3×3 dans un graphe de Petersen P . Vous pourrez au besoin utiliser les opérations fusion et deconnecte. Quelle est la dilatation du plongement ? Puis, en vous appuyant sur l'exemple d'une grille 6×6 , expliquez comment vous plongeriez une grille de taille $a \times a$ dans HP_n .*

▷ **Question 10** *Plus difficile. Comment plonger un arbre binaire complet à $2^n - 1$ sommets dans un HP_n pour $n \geq 4$, avec une dilatation de 1 ? Présentez le plongement pour $n = 5$.*

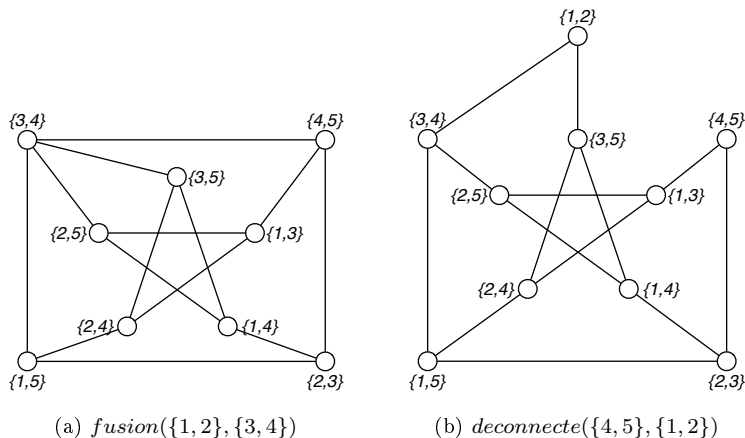


FIG. 2: $\text{fusion}(\{1, 2\}, \{3, 4\})$ et $\text{deconnecte}(\{4, 5\}, \{1, 2\})$

2 Anneaux de processeurs

Nous nous intéressons à la résolution d'un système linéaire $Ax = b$, où A est une matrice inférieure triangulaire d'ordre n , et b est un vecteur de n éléments. La topologie que nous utiliserons est un anneau de p processeurs P_0, P_1, \dots, P_{p-1} . Nous considérerons que n est choisi de telle sorte qu'il soit divisible par p .

▷ **Question 11** *On veut distribuer les colonnes de A aux processeurs. Quelle stratégie choisissez-vous ? Donnez un algorithme parallèle adapté à cette distribution.*

▷ **Question 12** *On veut maintenant distribuer les lignes de A aux processeurs. Quelle stratégie choisissez-vous ? Donnez un algorithme parallèle adapté à cette distribution.*

3 PRAM

▷ **Question 13** *Soit A et B deux matrices de taille $n \times n$. Proposez un algorithme CREW permettant de multiplier A et B en $O(\log n)$ sur n^3 processeurs.*

Références

- [1] SK Das and AK Banerjee. Hyper Petersen network : Yet another hypercube-like topology. In *Frontiers of Massively Parallel Computation, 1992., Fourth Symposium on the*, pages 270–277, 1992.

4 Réponses aux exercices

▷ Question 1, page 1

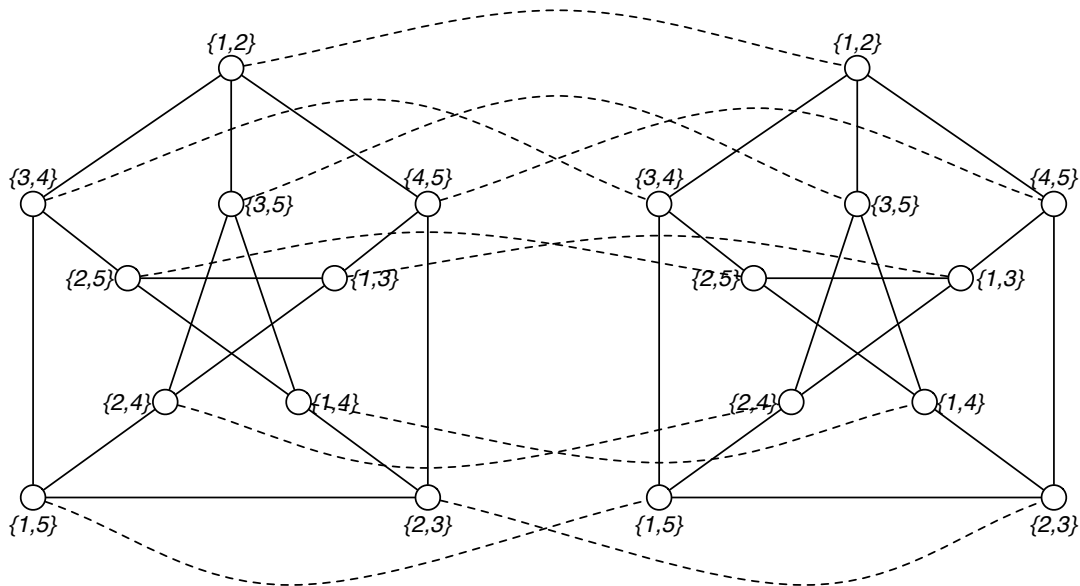


FIG. 3: Réseau $HP_4 = Q_1 \times P$

▷ Question 2, page 1

$HP_n = Q_{n-3} \times P = Q_{n-4} \times K_2 \times P = HP_{n-1} \times K_2$, pour $n \geq 4$. Ainsi on peut construire un HP_n en connectant 2 réseaux HP_{n-1} .

▷ Question 3, page 1

- degré : n . 3 dans le graphe Petersen, et $n - 3$ dans l'hypercube.
- diamètre : $n - 1$, soit deux nœuds x et y dans deux graphes de Petersen distincts dans $HP_n : P^x$ et P^y . Le nombre de bits différents dans leurs préfixes est d'au plus $n - 3$. Ainsi, en utilisant la technique du complément de 1 bit à chaque étape, en au plus $n - 3$ étapes P^y peut être contacté par P^x . Il faut ensuite au plus 2 étapes pour joindre y dans P^y .
- nombre de nœuds : $1.25^n = 10 \cdot 2^{n-3}$

▷ Question 4, page 2

$\kappa = n$: il y a $H(x, y)^1$ chemins sans arêtes communes de taille $H(x, y)$ entre deux graphes de Petersen où x et y sont. Pour chacun de ces chemins, il existe 3 chemins différents dans le graphe P^y . D'où le résultat pour la connectivité.

¹ $H(x, y)$ est la distance de Hamming entre deux nœuds x et y , *i.e.*, le nombre de bits différents dans leur préfixe.

▷ **Question 5, page 2**

Pour Q_n : degré = n , diamètre = n , nombre de nœuds = 2^n , et connectivité = n .

Pour un même degré au niveau des nœuds, un réseau HP_n contient 1.25 fois plus de nœuds, et a un diamètre inférieur d'une unité.

▷ **Question 6, page 2**

Chaque nœud de HP_n peut être représenté par un couple : $[B(u), P(i, j)]$, où $B(u)$ est le code binaire (de longueur $(n - 3)$ bits) d'un nœud u dans un hypercube, pour $0 \leq u \leq 2^{n-3} - 1$, et $P(i, j)$ est un nœud dans le graphe de Petersen (représenté par $\{i, j\}$). $B(u)$ est le préfix du nœud, et son graphe de Petersen correspondant sera noté $P^{B(u)}$. Nous avons ainsi 2 types de connexions dans un graphe HP_n :

- *Connexion Petersen* : un nœud $[B(u), P(i, j)]$ est adjacent à un nœud $[B(u), P(k, l)]$ si et seulement si $\{i, j\} \cap \{k, l\} = \emptyset$, pour $1 \leq i, j, k, l \leq 5$. Cela donne l'adjacence dans $P^{B(u)}$.
- *Connexion Hypercube* : un nœud $[B(u), P(i, j)]$ est adjacent à un nœud $[B(v), P(i, j)]$ si et seulement si $B(u)$ et $B(v)$ ne diffèrent que d'un bit. Cela donne l'adjacence entre les nœuds *correspondant* de $P^{B(u)}$ et $P^{B(v)}$.

Soit $S = [B(u), P(i, j)]$ la source, et $D = [B(v), P(k, l)]$ la destination, avec $S \neq D$. Si $u = v$, le message doit seulement circuler au sein de $P^{B(u)}$ du nœud $P(i, j)$ à $P(k, l)$, et nécessite seulement 2 hops, *i.e.*, le diamètre du réseau Petersen. Si $u \neq v$, en complémentant bit à bit, et en choisissant le voisin qui a la distance de Hamming la plus faible par rapport à $B(v)$ pour router le message, on arrive ainsi à router le message de $P^{B(u)}$ à $P^{B(v)}$. Il ne reste plus qu'à router le message dans $P^{B(v)}$. Il faut ainsi $(n - 1)$ hops pour router le message.

Le routage dans le Petersen peut se faire simplement en sélectionnant à qui on envoie le message de la façon suivante (message de (i, j) à (a, b)) :

- si $(i, j) = (a, b)$, on ne fait rien
- si $\{i, j\} \cap \{a, b\} = \emptyset$, alors (a, b) est voisin direct, on peut lui envoyer le message
- sinon, choisir le voisin (x, y) tel que $\{i, j\} \cap \{a, b\} \cap \{x, y\} = \emptyset$ (il existe forcément).

▷ **Question 7, page 2**

Deux réponses possibles. Soit on considère qu'on est en modèle 1-port, et du coup chaque nœud ne peut envoyer qu'un seul message à chaque étape, soit il peut envoyer en parallèle.

En 1-port : Il faut $(n - 3) + 4$ étapes pour diffuser un message sur un HP_n : $(n - 3)$ pour atteindre tous les graphes de Petersen, puis 4 étapes pour diffuser au sein des graphes de Petersen (le P le plus loin est atteint au bout de $(n - 3)$ étapes).

En multiport : Il faut autant d'étapes dans l'hypercube, mais par contre il faut 2 étapes dans le Petersen, soit $(n - 1)$ étape au total.

▷ **Question 8, page 2**

On part d'un HP_4 . On peut trouver un chemin hamiltonien dans P^0 , ayant pour préfix 0, qui part du nœud $[0, x]$ et terminant en $[0, y]$. On cherche également un chemin hamiltonien dans P^1 commençant en $[1, y]$ et terminant en $[1, x]$. Puisque $[0, y]$ et $[1, y]$ sont des nœuds "identiques" dans P^0 et P^1 , ils sont donc adjacents. Il en est de même pour $[0, x]$ et $[1, x]$. On a ainsi un cycle hamiltonien dans HP_4 : $[0, x], \dots, [0, y], [1, y], \dots, [1, x]$.

On suppose maintenant qu'un anneau de taille $1.25 \times 2^{n-1}$ puisse être plongé dans un HP_{n-1} . HP_n peut être décomposé en 2 HP_{n-1} , et on peut trouver un chemin hamiltonien dans chacun d'eux, l'un étant le miroir de l'autre, on peut alors les connecter à l'aide des connexions dans l'hypercube. La dilatation est bien de 1.

On peut donc plonger des anneaux de taille $5 \leq l \leq 1.25 \times 2^n$ dans HP_n .

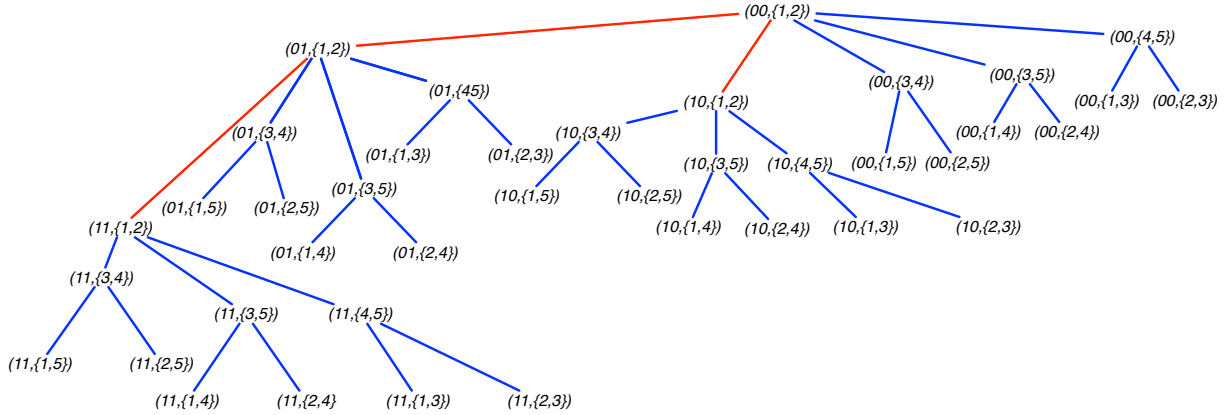


FIG. 4: Broadcast sur HP_5 . Les liens rouges représentent les communications pour faire transiter le message d'un graphe de Petersen à un autre (communications dans l'hypercube), et les liens bleus sont les communications au sein des graphes de Petersen.

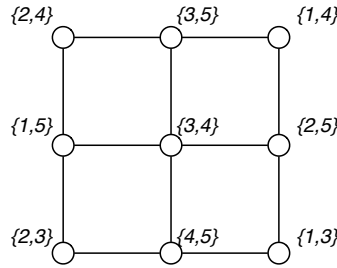


FIG. 5: Plongement d'une grille 3×3 dans P . $\text{fusion}(\{1, 2\}, \{3, 4\})$ et déconnection sur les bords.

▷ **Question 9, page 2**

Il suffit de fusionner 2 nœuds comme présenté Figure 2a. Puis de ne garder que les liens comme présenté Figure 5. La dilatation est de 2

Le cas de la grille 6×6 est donné Figure 6.

La dilatation est toujours de 2.

▷ **Question 10, page 2**

Plus compliqué celui là . . .

Pour plus de clarté on renumérote les nœuds de P comme indiqué Figure 7, et on utilise la notation décimale pour le numéro du nœud dans l'hypercube.

Le plongement dans HP_5 est réalisé comme présenté Figure 8. On part du nœud $(0, 0)$, jusqu'au niveau $n - 3$, à chaque niveau, le fils gauche est un nœud du même graphe de Petersen, alors que le fils droit est le nœud de même indice dans le graphe de Petersen, mais dans un autre P de l'hypercube, ne différant que d'un bit. Avec cette méthode on peut plonger un arbre binaire complet de hauteur $n - 3$, il nous faut donc deux niveaux de plus.

Voir [1] pour la suite.

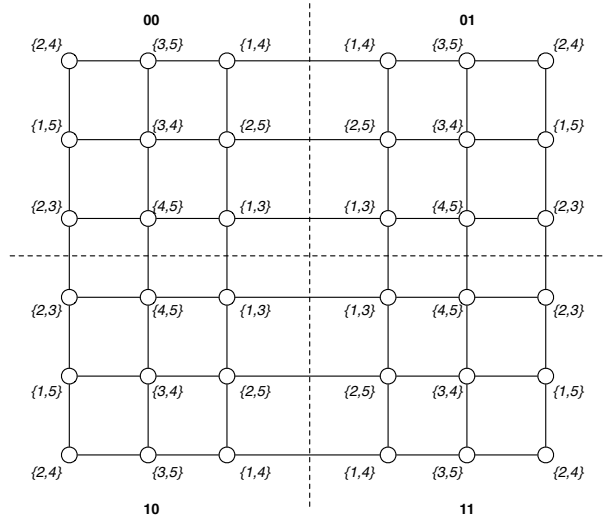


FIG. 6: Plongement d'une grille 6×6 dans HP_5 .

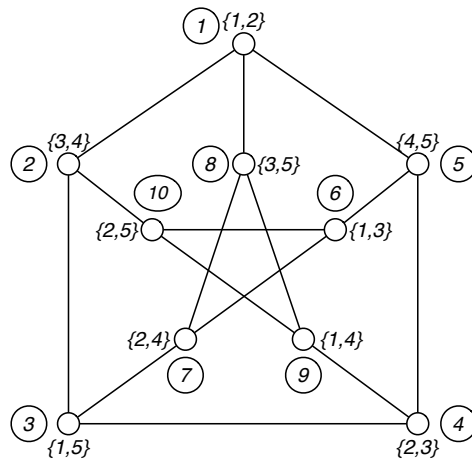


FIG. 7: Renumération

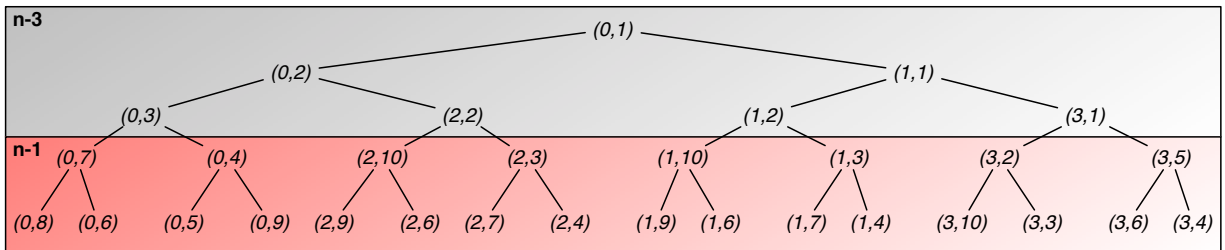


FIG. 8: Plongement d'un arbre binaire complet de $2^5 - 1$ sommets dans HP_5 .

▷ Question 11, page 3

Si on distribue les colonnes de A aux processeurs, il faut paralléliser les opérations sur les lignes ! En effet, dans un algorithme où on manipulerait les colonnes de A plutôt que les lignes, un seul processeur (celui qui posséderait la colonne courante) pourrait être actif à une étape donnée. Au contraire, il faut pouvoir opérer en parallèle sur une ligne de la matrice, chaque processeur apportant une contribution pour les colonnes qui le concernent. De manière générale, si on utilise un algorithme en ligne sur une distributions des données en ligne, on ne va rien gagner à cause du problème de la précédence des calculs ; c'est pourquoi, on préférera utiliser un algorithme colonne sur une distribution des données en ligne – et réciproquement.

L'algorithme de résolution séquentiel en cascade est bien adapté à un traitement par lignes :

```
Pour  $i = 0$  to  $n - 1$  :  
  Pour  $j = 0$  to  $i - 1$  :  
     $b[i] \leftarrow b[i] - a[i, j] \times x[j]$   
   $x[i] \leftarrow b[i]/a[i, i]$ 
```

À l'étape i , on calcule le produit scalaire des éléments de la i -ème ligne de A situés avant la diagonale avec les éléments déjà calculés du vecteur x . On voit qu'il faut allouer de manière cyclique les colonnes de A aux processeurs pour équilibrer la charge de travail à chaque étape. On attribue donc la ligne i de A , ainsi que la i -ème composante $b[i]$ du vecteur b , au processeur d'indice $alloc(i) = i \bmod p$. Ce processeur sera responsable du calcul des mêmes composantes du vecteur x . Avec une telle allocation, on obtient le schéma de résolution parallèle :

```
Pour  $i = 0$  to  $n - 1$  :  
   $t \leftarrow 0$   
  Pour  $j \in MyCols, j < i$  :  
     $t \leftarrow t + a[i, j] \times x[j]$   
   $s = \text{GATHER}(alloc(i), t)$   
  Si  $i \in MyCols$  Alors  
     $x[i] \leftarrow (b[i] - s)/a[i, i]$ 
```

Dans ce code, *MyCols* dénote symboliquement les colonnes allouées à un processeur donné. L'opération **GATHER** est une opération de réduction : les produits scalaires partiels, calculés localement sur chaque processeur, sont regroupés et additionnés ; le résultat final est disponible dans la mémoire du processeur $alloc(i)$.

▷ Question 12, page 3

La situation est symétrique de la précédente : si on choisit de distribuer les lignes de A aux processeurs, il faut à une étape donnée pouvoir opérer en parallèle sur une colonne de la matrice, chaque processeur apportant une contribution pour les lignes qui le concernent. En inversant les deux boucles dans l'algorithme séquentiel de la question précédente, on obtient le code :

```
Pour  $j = 0$  to  $n - 1$  :  
   $x[j] \leftarrow b[j]/a[j, j]$   
  Pour  $i = j + 1$  to  $n$  :  
     $b[i] \leftarrow b[i] - a[i, j] \times x[j]$ 
```

On voit qu'il faut encore une allocation cyclique pour équilibrer la charge de travail à chaque étape. On attribue donc la colonne i de A , ainsi que la i -ème composante $b[i]$ du vecteur b , au processeur d'indice $alloc(i) = i \bmod p$. Ce processeur sera responsable du calcul des mêmes composantes du vecteur x . Avec une telle allocation, on obtient le schéma de résolution parallèle :

```

Pour  $j = 0$  to  $n - 1$  :
  Si  $j \in MyRows$  Alors
     $x[j] \leftarrow b[j]/a[j, j]$ 
    BROADCAST( $alloc(j), x[j]$ )
  Pour  $i \in MyRows, i > j$  :
     $b[i] \leftarrow b[i] - a[i, j] \times x[j]$ 

```

À chaque étape j , le processeur responsable de $x[j]$ calcule sa valeur et le diffuse aux autres processeurs. Ces derniers font alors une mise à jour de leurs données.

▷ **Question 13, page 3**

```

Pour  $i \leftarrow 1$  to  $n$  en parallèle :
  Pour  $j \leftarrow 1$  to  $n$  en parallèle :
    Pour  $k \leftarrow 1$  to  $n$  en parallèle :
       $D[i, j, k] \leftarrow A[i, j] \times B[j, k]$ 
  Pour  $i \leftarrow 1$  to  $n$  en parallèle :
    Pour  $k \leftarrow 1$  to  $n$  en parallèle :
       $C[i, k] \leftarrow D[i, 1, k] + \dots + D[i, n, k]$ 

```

Le calcul de la matrice D prend un temps constant sur n^3 processeurs (CREW). Le calcul de la matrice C prend un temps $O(\log n)$ sur n^3 processeurs : on peut calculer les sommes à l'aide de l'algorithme de calcul de préfixe en $O(\log n)$ sur n processeurs (EREW), et on a 2 boucles parallèles imbriquées, d'où les n^3 processeurs.