Processus concurrents et parallélisme

Chapitre 8 - Algorithmes parallèles

Gabriel Girard

18 octobre 2022

Chapitre 8 - Algorithmes parallèles

Objectif

- L'objectif est d'analyser la performance d'algorithmes parallèles
- Nous avons besoin de modèles et d'un cadre de travail pour présenter et analyser les algorithmes

Algorithmique séquentiel

- Le modèle communément accepté est un processeur avec une mémoire centrale (RAM)
- Les instructions sont : accès mémoire (load, store), arithmétiques, logiques, ...
- Le succès du modèle est basé sur sa simplicité et son habileté à capturer les performances des algorithmes séquentiels.

Algorithmique parallèle

- Il y a plusieurs modèles et la situation est plus complexe
- La performance dépend de facteurs tels :
  - la concurrence
  - l'allocation des processeurs et la planification
  - la communication
  - la synchronisation

Modèles

- Plusieurs modèles sont utilisés pour l'analyse et le développement
- Les plus connus sont :
  - les graphes orientés acycliques (DAG)
  - le modèle à mémoire partagée (PRAM)
  - le modèle réseau

Contexte de nos analyses

- Un algorithme parallèle s'exécute sur un ordinateur parallèle pour résoudre un problème donné
- Ordinateur parallèle = multiprocesseurs et des réseaux locaux
- L'objectif est de diminuer le temps de traitement

Objectifs

- Déterminer si un algorithme est bon ou non pour le calcul parallèle
- Critères pour le séquentiel : temps d'exécution, utilisation de l'espace et facilité de programmation
- Critères pour le parallèle : temps d'exécution, utilisation de l'espace, facilité de programmation, nombre de processeurs, capacité des mémoires locales, modèle de communication et les protocoles de synchronisation

Mesures pour l'évaluation : accélération

- Soit Q un problème et n la taille des données en entrée
- La complexité séquentielle de Q est $T^*(n)$
- Soit A un algorithme parallèle qui résout Q en temps $T_p(n)$ sur une machine parallèle avec p processeurs
- Accélération $S_p(n) = \frac{T^*(n)}{T_p(n)}$
- Idéalement $S_p(n) \equiv p$ (réalité $S_p(n) \leq p$)

Mesures pour l'évaluation : accélération

- Facteurs qui nuisent à l'accélération
  - insuffisamment de concurrence
  - délais de communication
  - surcharge du système due à la synchronisation et au contrôle
- **Note importante** : $T_1(n)$ peut être différent de $T^*(n)$

Mesures pour l'évaluation : accélération

- Il y a 5 définitions de l'accélération
  - Accélération relative $RS_p(n) = \frac{T_1(n)}{T_p(n)}$
  - Accélération réelle $S_p(n) = \frac{T^*(n)}{T_p(n)}$
  - Accélération absolue $AS_p(n) = \frac{meilleur\ \ algo + meilleur\ \ pcsr}{T_p(n)}$
  - Accélération réelle asymptotique $ARS_p(n) = \frac{O(T^*(n))}{O(T_p(n))}$
  - Accélération relative asymptotique $AVS_p(n) = \frac{O(T_1(n))}{O(T_p(n))}$
- En pratique on peut aussi pondérer l'accélération avec le coût de l'achat, l'installation et l'entretien des systèmes

Mesures pour l'évaluation : accélération

Exemple 1

- $T^*(n) = 20$
- $T_{10}(n) = 4$
- $T_1(n) = 25$
- $S_{10}(n) = \frac{20}{4} = 5$
- $RS_{10}(n) = \frac{25}{4} = 6,25$

Mesures pour l'évaluation : efficacité (E_p)

- L'efficacité fournit une indication sur l'utilisation effective de tous les processeurs
- $E_p(n) = \frac{T_1(n)}{pT_p(n)}$
- $E_p(n) = 1$ indique que $T_p(n)$ s'exécute p fois plus vite
  Cela signifie que les p processeurs font du travail utile pendant tout le calcul

- Exemple 1 : $E_{10}(n) = \frac{25}{10 \times 4} = 0.625$

Mesures pour l'évaluation : efficacité E_p

- Il existe une borne inférieure au temps d'exécution noté $T_\infty(n)$ au-delà duquel aucune accélération n'est possible peu importe le nombre de processeurs utilisés
- En pratique $T_p(n) \geq T_\infty(n) \quad \forall p$
- En général $E_p(n) \leq \frac{T_1(n)}{pT_\infty(n)}$
- L'efficacité se dégrade rapidement lorsque p devient plus grand que $\frac{T_1(n)}{T_\infty(n)}$

Mesures pour l'évaluation : efficacité E_p

Exemple 1 :

- Soit $T_\infty(n) = 3$
- $E_{10}(n) = \frac{25}{10\times4} = 0.625$
- $E_{10}(n) \leq \frac{25}{10\times3} = 0.833$
- $E_{20}(n) \leq \frac{25}{20\times3} = 0.416$
- $\frac{T_1(n)}{T_\infty(n)} = 25/3 = 8,333$

Généralités

- Énoncés : assignation, bloc, sélection, itération, fin
- La limite de l'utilisation des ressources (temps et mémoire) est mesurée en fonction de la taille des données en entrée
- Généralement on analyse le pire cas

Mesures

- Soit n la taille des données en entrée
- Les limites sont exprimées de façon asymptotique en utilisant les notations suivantes :
  1. $T(n) = O(f(n))$ *si* $\exists$ c *et* n_0 *tel que* $T(n) \leq c \times f(n)\ \forall n \geq n_0$
  2. $T(n) = \Omega(f(n))$ *si* $\exists$ c *et* n_0 *tel que* $T(n) \geq c \times f(n)\ \forall n \geq n_0$
  3. $T(n) = \Theta(f(n))$ *si* $T(n) = O(f(n))$ *et* $T(n) = \Omega(f(n))$

Mesures

- Le temps d'exécution est estimé selon le nombre d'opérations de base
- Les opérations de base sont : lecture, écriture et opérations arithmétiques et logiques
- La taille des mots n'affecte pas le coût
- Le modèle utilisé est RAM

Critères

- On veut des modèles utiles à l'analyse des algorithmes
- Ils doivent être : simples et implantables
- Les modèles connus sont :
  - graphes orientés acycliques
  - PRAM
  - réseau
  - arbres de comparaison
  - réseaux de tri (sorting networks)
  - circuits booléen
  - ...

Présentation DAG

- Noeud sans arc en entrée = entrée
- Noeud avec arc en entrée provenant d'autres noeuds = opération
- Degré intérieur est au plus deux
- Noeud avec degré extérieur = 0 représente une sortie
- Un algorithme est représenté par une famille de graphes $\{G_n\}$ où n correspond à la taille de l'entrée

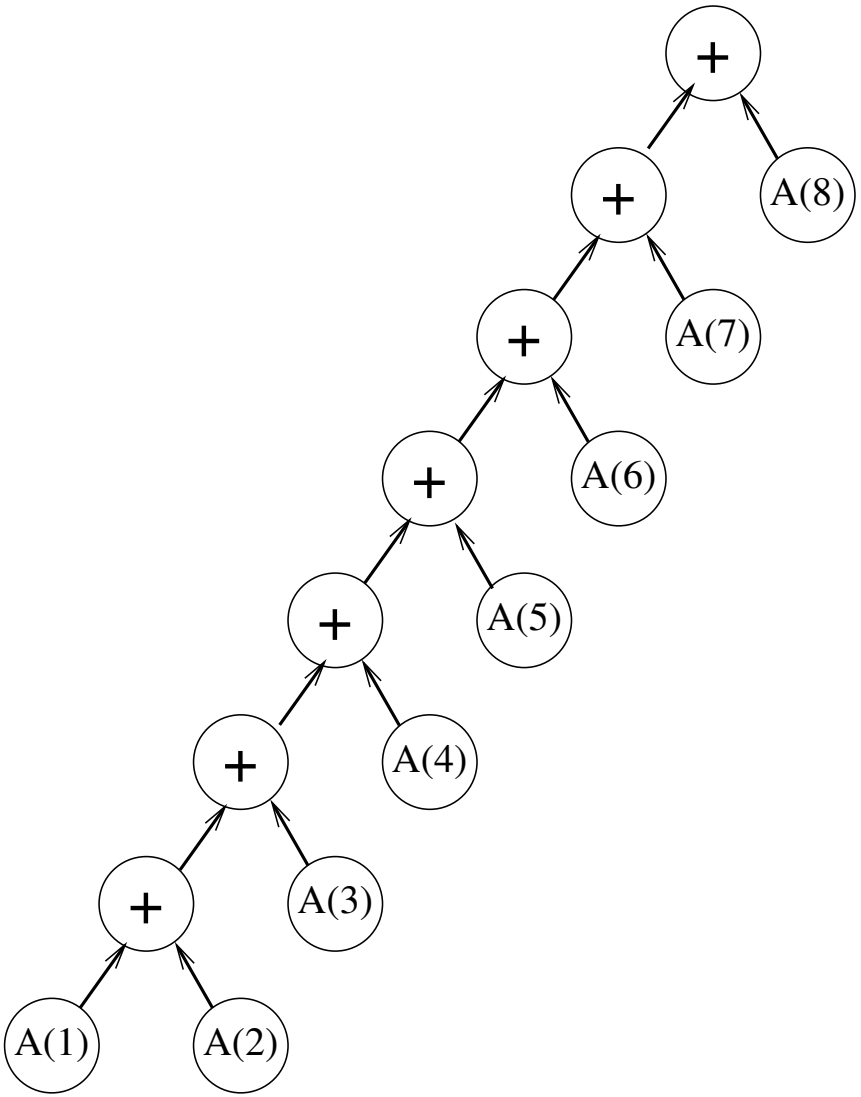
Présentation DAG

- DAG avec n noeuds d'entrée = traitement sans branchement
- Instructions de branchement = boucle qui dépendent de n
- On déroule la boucle en répétant l'instruction
- DAG spécifie opérations et contraintes de précédence
- Utilisé en analyse numérique

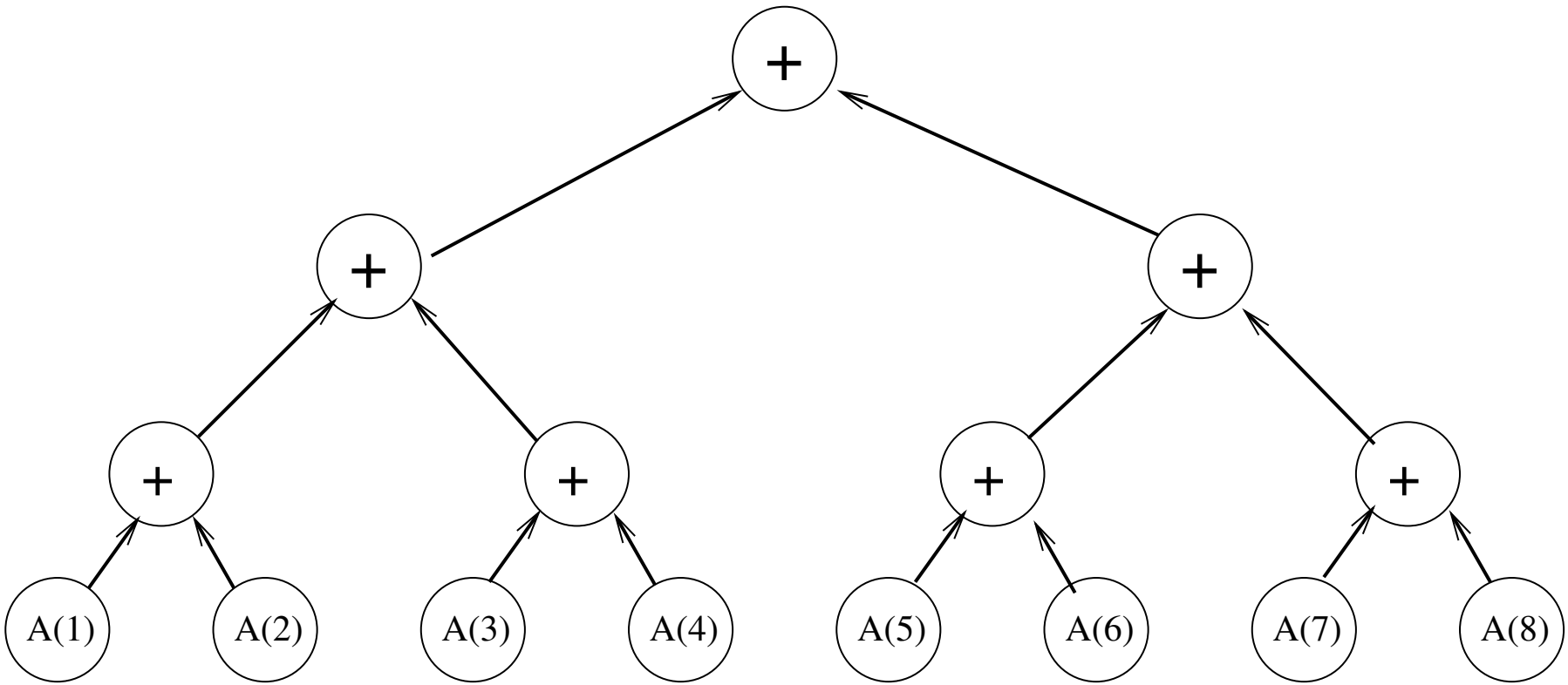
Exemple : somme de n valeurs

- On veut calculer la somme des n éléments d'un tableau
- Il y a plusieurs représentations
- Une représentation sert à analyser la performance en supposant que chaque processeur peut accéder les données des autres sans coût additionnel
- La planification consiste à associer chaque noeud à un processeur

Exemple : somme de n valeurs (modèle)



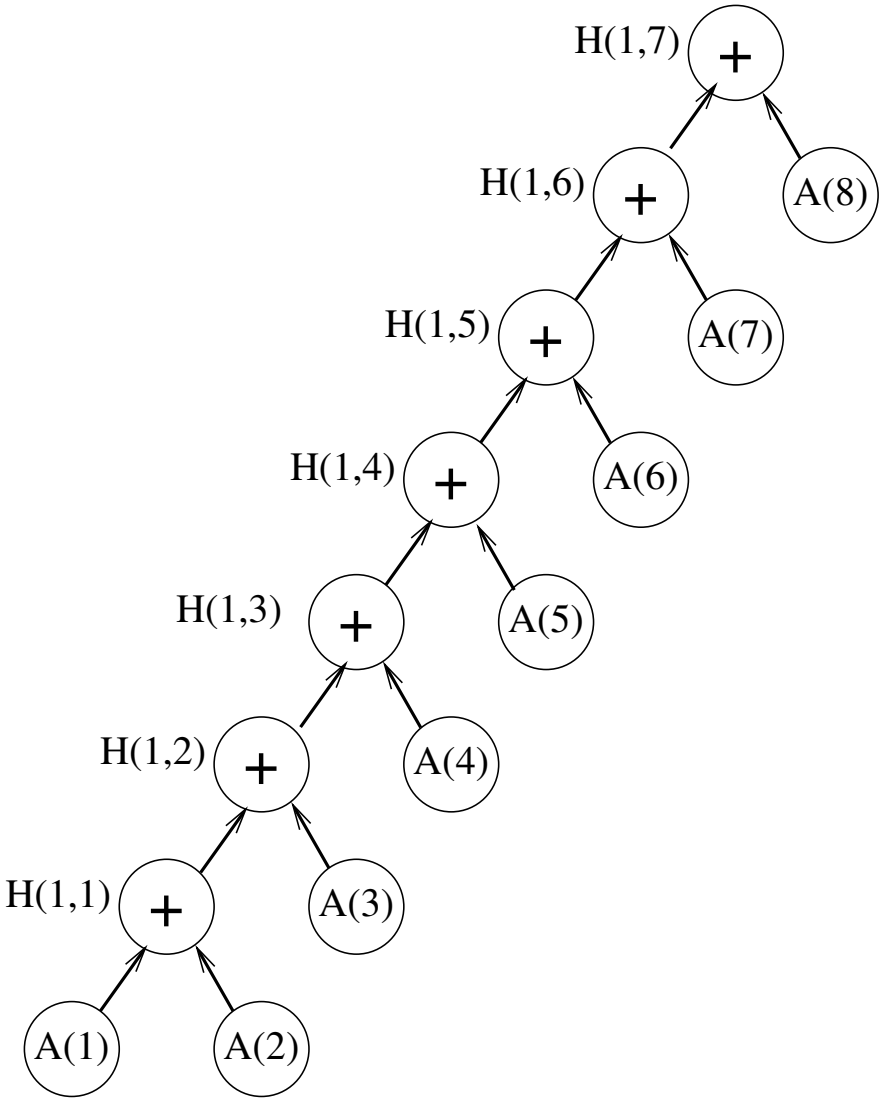
Exemple : somme de n valeurs (modèle)



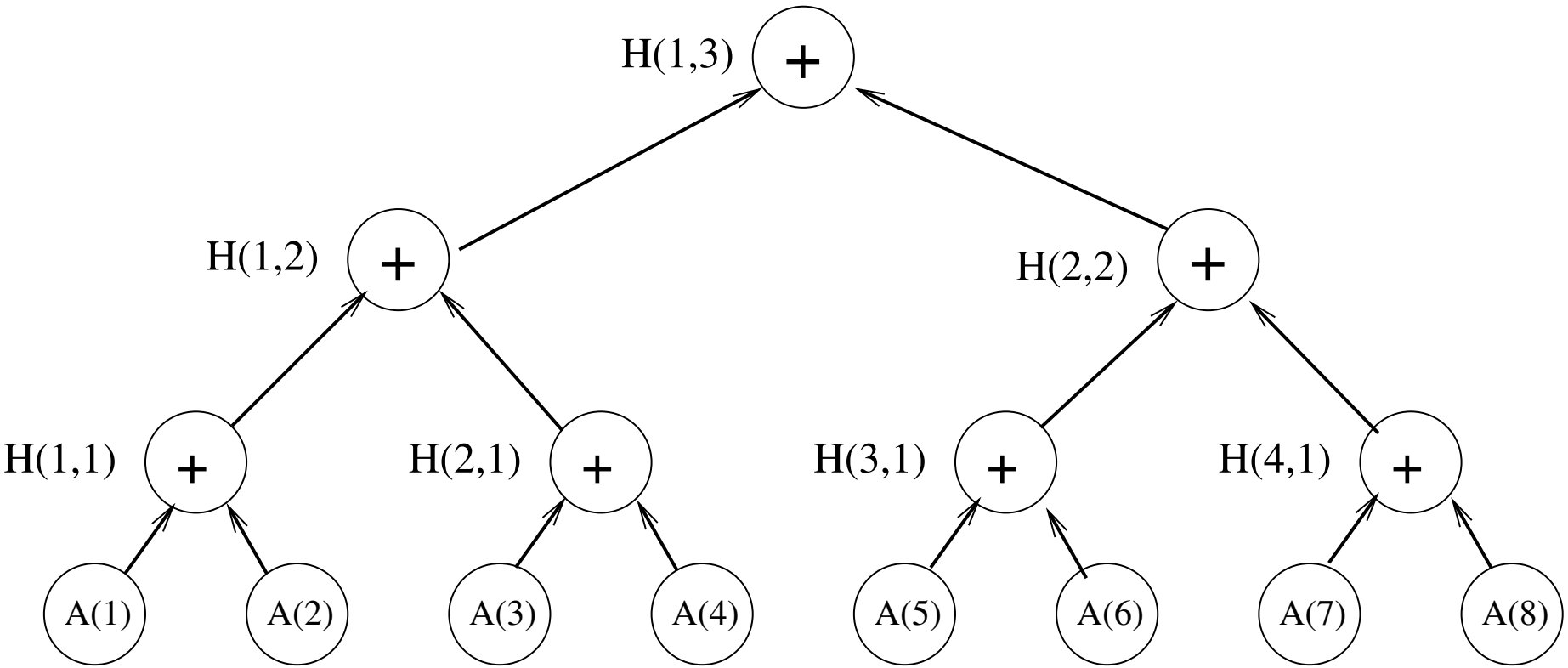
Exemple : somme de n valeurs (planification)

- Soit p processeurs
  On associe à chaque noeud interne i une paire (j_i, t_i) où $j_i < p$ est un numéro de processeur et t_i est une unité de temps
- Les conditions suivantes doivent être respectées :
  1. Si $t_i = t_k$ pour $i \neq k$ alors $j_i \neq j_k$
  2. Si (i, k) est un arc alors $t_k \geq t_i + 1$
- Le temps t_i du noeud de départ est 0 et aucun processeur n'est associé à ce i
- On appelle la séquence $\{(j_i, t_i) | i \in n\}$ un horaire

Exemple : somme de n valeurs



Exemple



Exemple : somme de n valeurs

t_i	Horaire graphe 1	Horaire graphe 2
i=1	$p_1 \rightarrow B_1 := A_1 + A_2$	$p_1 \rightarrow B_1 := A_1 + A_2$ $p_2 \rightarrow B_2 := A_3 + A_4$ $p_3 \rightarrow B_3 := A_5 + A_6$ $p_4 \rightarrow B_4 := A_7 + A_8$
i=2	$p_1 \rightarrow B_1 := B_1 + A_3$	$p_1 \rightarrow C_1 := B_1 + B_2$ $p_2 \rightarrow C_2 := B_3 + B_4$
i=3	$p_1 \rightarrow B_1 := B_1 + A_4$	$p_1 \rightarrow D_1 := C_1 + C_2$
i=4	$p_1 \rightarrow B_1 := B_1 + A_5$	
i=5	$p_1 \rightarrow B_1 := B_1 + A_6$	
i=6	$p_1 \rightarrow B_1 := B_1 + A_7$	
i=7	$p_1 \rightarrow B_1 := B_1 + A_8$	

Exemple : somme de n valeurs

- Pour chaque horaire le temps pour exécuter le programme est $Max_{i \in n} t_i$
- Le temps d'exécution correspond à la profondeur du graphe
- La complexité parallèle du DAG est
  $T_p(n) = Min(Max_{i \in n} t_i)$
  où Min est choisi parmi tous les horaires possibles

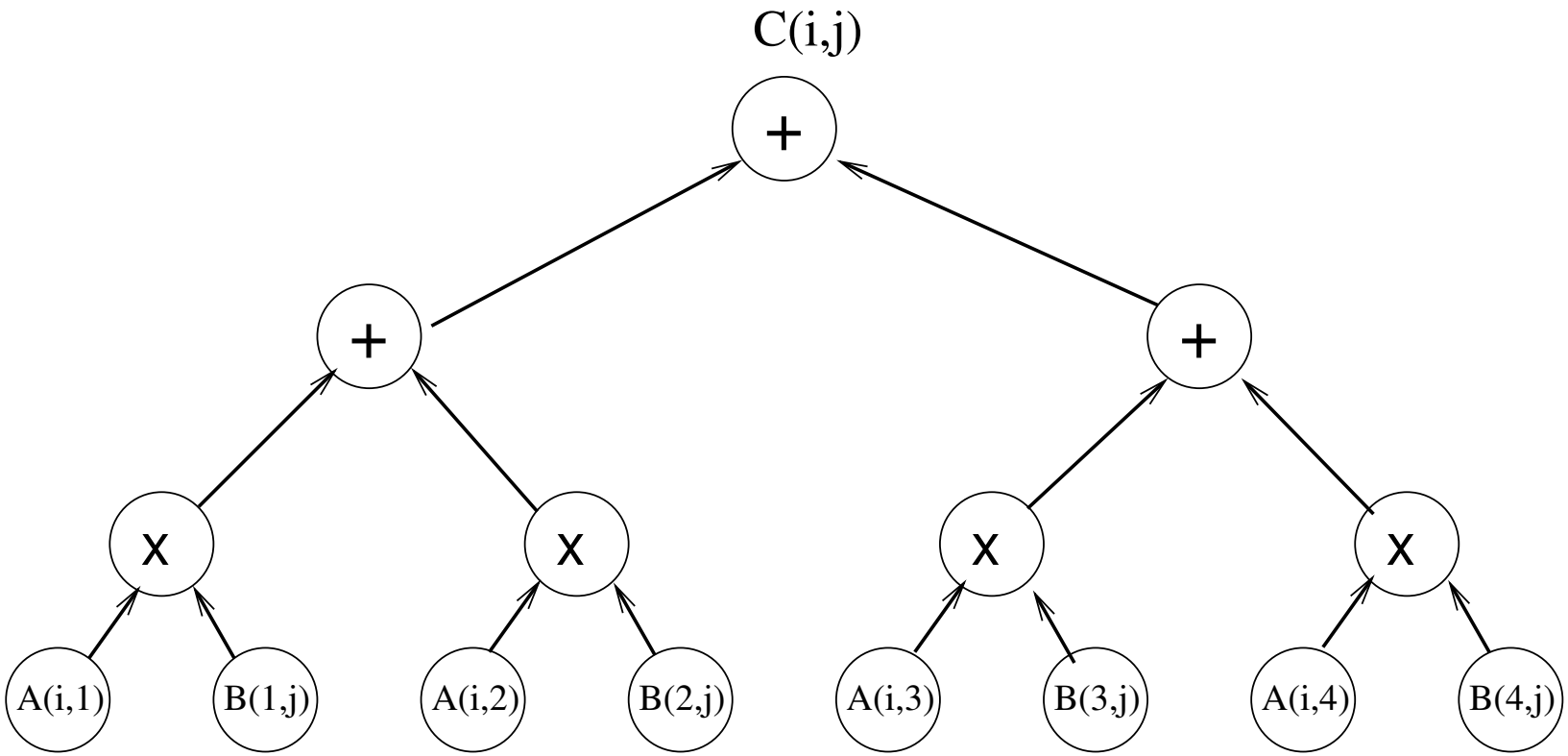
Exemple : somme de n valeurs

- Performance du premier DAG : $O(n)$ peu importe le nombre de processeurs
- Performance du deuxième DAG : $O(log_2 n)$ avec $n/2$ processeurs

Exemple : multiplication de matrices

- Soit A et B deux matrices $n \times n$.
- On veut $C = A \times B$
- On doit calculer n^2 produits scalaires $C(i, j)$
- Chaque $C(i, j) = \sum_{l=1}^{n} A(i, l) \times B(l, j)$
- On obtient un DAG de profondeur $log_2 n$ pour chaque produit scalaire

Exemple



Exemple : multiplication de matrices

- Avec n^3 processeurs : $O(log_2 n)$
- Avec n^2 processeurs : ?
- Avec n processeurs : ?
- Avec 1 processeur ?

Présentation PRAM

- Extension naturelle du modèle séquentiel RAM
- Plusieurs processeurs avec une mémoire locale privée et un mémoire globale partagée
- Chaque processeur exécute son propre programme et communique via la mémoire commune

Modes d'opérations

- Synchrone ou asynchrone
- SIMD ou MIMD

Opérations

- Global read(X,Y)
- Global write(U,V)
- La taille des données transférées représente la quantité de communication

Exemple : multiplication matrice-vecteur

- Soit A une matrice $n \times n$ et X un vecteur d'ordre n
- On veut calculer $Y = A \times X$
- Complexité de l'algorithme séquentiel $= O(n^2)$
- On a $P < n$ processeurs tel que $r = n/p$ est entier
- On opère de façon asynchrone

Exemple : multiplication matrice-vecteur

- On partitionne A pour obtenir $A_1, A_2, ..., A_p$ des matrices $r \times n$
- Chaque processeur p_i lit A_i et X
- Chaque processeur exécute $z = A_i \times X$
- Chaque processeur emmagasine z dans Y

Exemple : multiplication matrice-vecteur

```
Entrée : A : matrice
         X : vecteur
         n : taille de la matrice et du vecteur
         i : le numero du processeur
         p : le nombre de processeurs
         r : n/p
Sortie : z : composants (i-1)r+1, ..., ir de Y
```

Exemple : multiplication matrice-vecteur

```
1. global read(X, w)
2. global read(A((i-1)r+1:ir, 1:n), B)
3. z = B x w
4. global write(w, Y((i-1)r+1:ir)
```

Si $n = 10, p = 5$ et $r = 2$:

$\longrightarrow P_1$: read $(A(1:2, 1:n), B)$
$\longrightarrow P_2$: read $(A(3:4, 1:n), B)$
$\longrightarrow$ ...
$\longrightarrow P_5$: read $(A(9:10, 1:n), B)$

Exemple : multiplication matrice-vecteur

- Étape 3 : $O(n^2/p)$ opérations arithmétiques
- Étape 1 et 2 transfère : $O(n^2/p)$ valeurs
- Étape 4 écrit n/p valeurs

Exemple : multiplication matrice-vecteur

- Aucune synchronisation requise
- Un autre partitionnement aurait pu exiger de la synchronisation

Matrice 4 x 4 avec 2 processeurs ($r = n/p = 2$)

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

En pratique

Création de 2 matrices 2 x 4 (r x n où $r = n/p = 4/2 = 2$)

$$P_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 + 4 + 9 + 16 \\ 5 + 12 + 21 + 32 \end{pmatrix} = \begin{pmatrix} 30 \\ 70 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 + 10 + 18 + 28 \\ 1 + 4 + 9 + 16 \end{pmatrix} = \begin{pmatrix} 60 \\ 30 \end{pmatrix}$$

En pratique

Chaque processus P_i effectue :

- $r \times n$ multiplications
- $r \times n = n^2/p$ multiplications (car $r = n/p$)
- $r \times (n-1) = n(n-1)/p$ additions
- Total : $(n^2/p) + n(n-1)/p$ opérations $= O(n^2/p)$

Chaque processus P_i effectue (pour notre exemple) :

- $2 \times 4 = 8$ multiplications
- $4^2/2 = 8$ multiplications (car $r = n/p$)
- $4 \times 3/2 = 6$ additions
- Total : $4^2/2 + 6 = 14 \rightarrow O(n^2/p)$

Exemple : somme de n valeurs

- On veut additionner tous les éléments d'un tableau A de taille $n = 2^k$
- On a n processeurs $p_1, p_2, ...p_n$
- On veut $S = A(1) + A(2) + ... + A(n)$
- Chaque processeur exécute le même algorithme

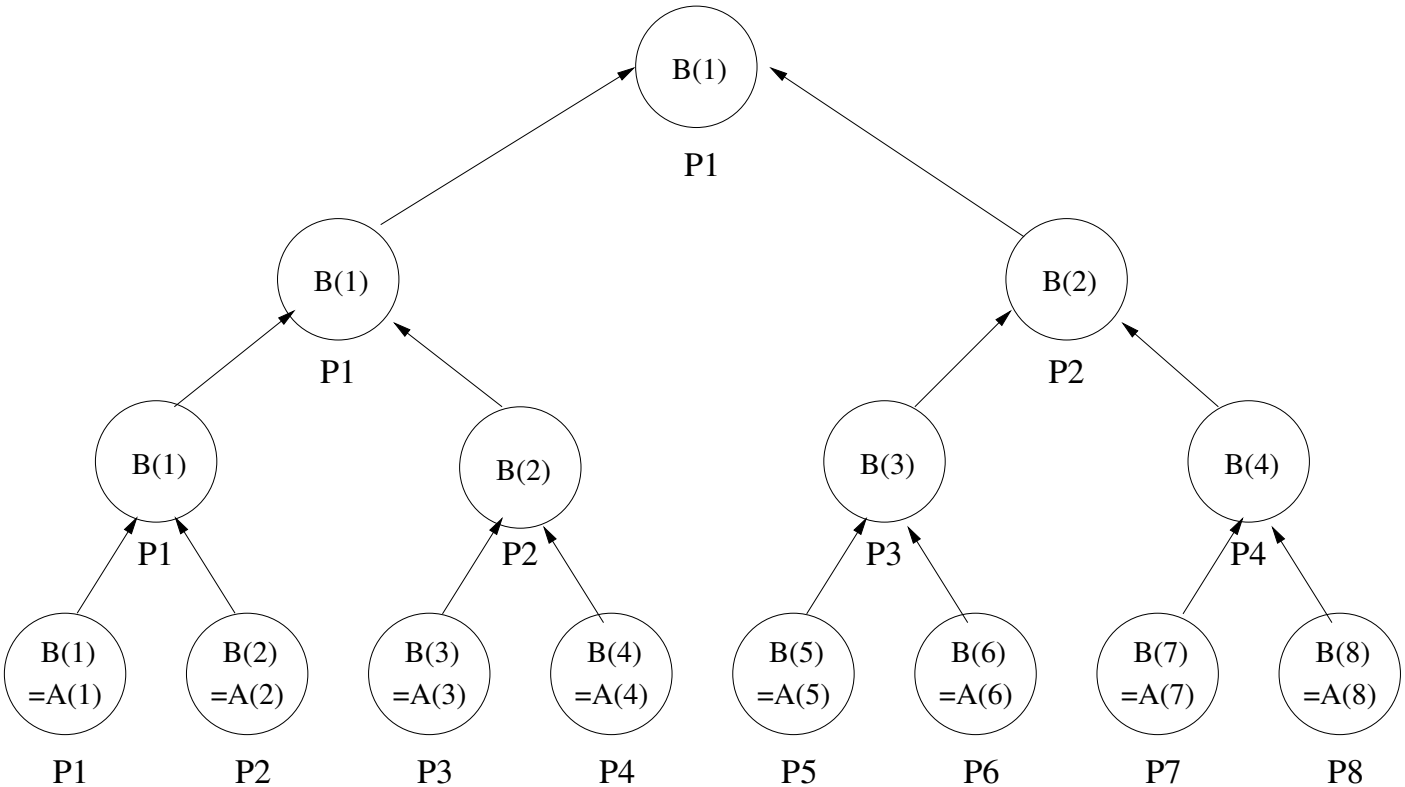
Exemple : somme de n valeurs

```
Entrée : A : vecteur de taille n
         n : taille de la matrice et du vecteur
         i : le numéro du processeur
Sortie : S : la somme
```

Exemple : somme de n valeurs

```
1. global read(A(i), a)
2. global write(a, B(i))
3. For h=1 to log n  do
    if (i <= n/2^h)
    begin
       global read(B(2i-1), x)
       global read(B(2i), y)
       z = x+y
       global write(z, B(i))
    end --- barrière ----
4. If i=1 global write(z,S)
```

Exemple : somme de n valeurs



Modes d'opérations

- Plusieurs modèles existent basés sur la capacité d'accès concurrents
  - EREW
  - CREW
  - CRCW (common, arbitrary, priority)
- Puissance : CRCW $>$ CREW $>$ EREW (mais de peu $\leq log_2 n$)

Exemple : Multiplication de matrices carrés

- On veut $C = A \times B$
- Les matrices sont n X n ou $n = 2^k$
- On a n^3 processeurs numéroté $P_{i,j,l}$
- Chaque $P_{i,j,l}$ calcule $A(i,l) \times B(l,j)$
- Pour chaque paire (i,j) les n processeurs calculent le produit scalaire $(O(log_2 n))$

Exemple : multiplication matrices carrés

```
Entrée : A : matrice de taille  n x n
         B : matrice de taille  n x n
         n : taille de la matrice et du vecteur
         i,j,l : le no du processeur
Sortie : C : A x B
```

Exemple : multiplication matrices carrés

```
// on omet les global read et global write
1. C'(i,j,l) := A(i,l) x B(l,j)
2. For h=1 to log n  do
    if (l <= n/2^h)
       C'(i,j,l) = C'(i,j,2l-1) + C'(i,j,2l)
4. If l=1 C(i,j) = C'(i,j,l)
```

Exemple : Multiplication de matrices carrés

- On doit avoir un PRAM de type CREW
- Performance : $O(log_2 n)$
- Si on retire le if de l'étape 3 : requiert common CRCW

Modèle réseau : présentation

- Un réseau peut être vu comme un graphe $G = (N, E)$
- Chaque noeud $i \in N$ représente un processeur
- Chaque arc (i, j) représente un lien de communication bidirectionnel

Modèle réseau : mode d'opération

- Chaque processeur possède sa mémoire
- Les processeurs opèrent de façon synchrone ou asynchrone
- Aucune mémoire partagée n'est disponible
- Communication et synchronisation par messages
- Communications : Send(x,i) et Receive(y,j)
- Routage possible entre source et destination

Modèle réseau : topologie

- Ce modèle incorpore la topologie du réseau
- Paramètres pour évaluer la topologie :
  - diamètre
  - degré maximum
  - connectivité
- Topologies populaires :
  - réseau linéaire
  - maillage 2D
  - hypercube

Réseau linéaire

- Soit n processeurs $p_1, p_2, ..., p_n$
- Chaque p_i est connecté à p_{i-1} et p_{i+1}
- Si on connecte p_n à p_1 on obtient un anneau
- Diamètre $= n - 1$
- Degré maximum $= 2$

Exemple : multiplication matrice-vecteur

- On veut faire le produit $Y = A \times X$
- A est une matrice n x n et X est un vecteur d'ordre n
- On a p processeurs organisés en anneau tel que $r = n/p$ est un entier
- On partitionne A et X en blocs
  $A = (A_1, A_2, ..., A_p)$
  $X = (X_1, X_2, ..., X_p)$
- Chaque A_i est de taille n x r et X_i est d'ordre r
- Chaque processeur p_i calcule $z_i = A_i \times X_i$
- À la fin on accumule la somme $z_1 + z_2 + ... + z_p$

Matrice 4 x 4 avec 2 processeurs ($r = n/p = 2$)

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 1 & 2 & 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Matrice 4 x 4 avec 2 processeurs (r = n/p = 2)

$$P_1 = \begin{pmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \\ 1 & 2 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 & + & 4 \\ 5 & + & 12 \\ 9 & + & 20 \\ 1 & + & 4 \end{pmatrix} = \begin{pmatrix} 5 \\ 17 \\ 29 \\ 5 \end{pmatrix} = Z_1$$

$$P_2 = \begin{pmatrix} 3 & 4 \\ 7 & 8 \\ 11 & 12 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 9 & + & 16 \\ 21 & + & 32 \\ 33 & + & 48 \\ 9 & + & 16 \end{pmatrix} = \begin{pmatrix} 25 \\ 53 \\ 81 \\ 25 \end{pmatrix} = Z_2$$

Matrice 4 x 4 avec 2 processeurs ($r = n/p = 2$)

$$P_1 = \begin{pmatrix} 5 \\ 17 \\ 29 \\ 5 \end{pmatrix} + \begin{pmatrix} 25 \\ 53 \\ 81 \\ 25 \end{pmatrix} = \begin{pmatrix} 30 \\ 70 \\ 110 \\ 30 \end{pmatrix}$$

Exemple : multiplication matrice-vecteur

```
Entrée : B : matrice de taille n x r
                  -> A(1:n, (i-1)r+1:ir)
         w : vecteur d'ordre r -> X((i-1)r+1:ir)
         p : le nombre de processeurs
         i : le no du processeur
Sortie : Y : Y = A1*X1 + A2*X2 + ... + Ai*Xi (Pi)
         Y : Y = A*X (P1)
```

Exemple : multiplication matrice-vecteur

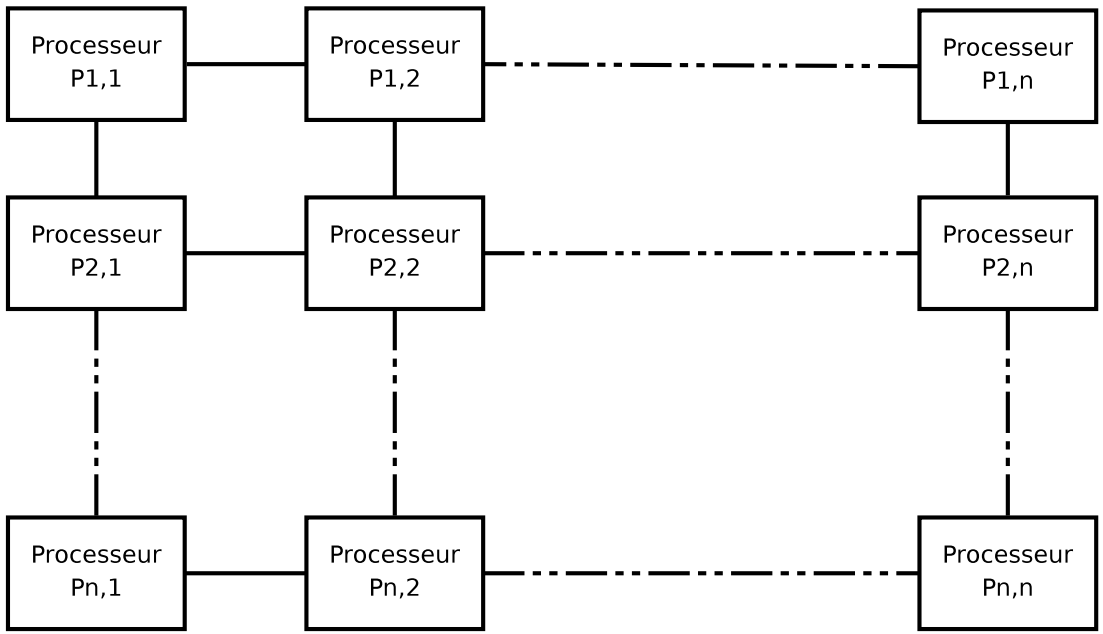
```
1. z := B * w
2. If i=1 then Y := 0
          else receive(y, left)
3. Y := Y + z
4. send(Y, right)
5. If i=1 then receive(y, left)
```

Exemple : complexité

- Calcul aux étapes 1 et 3 : $O(n^2/p)$
- Cependant chaque calcul doit attendre le résultat du précédant pour la somme
  - A_p attend que A_{p-1} soit fini (étape 5)
  - A_{p-1} attend que A_{p-2} soit fini (étape 5)
  - ...
- Temps de communication $= p \times comm(n)$
- $comm(n) = \sigma + n\tau$ où
  - σ est le temps pour initialiser la transmission
  - τ est le taux de transmission
- Temps d'exécution total $=$
  $T = T_{calcul} + T_{comm} = \alpha(n^2/p) + p(\sigma + n\tau)$

Maillage 2D

- Version 2D d'un réseau linéaire
- Contient $p = n^2$ processeurs organisés dans un grille $n \times n$ tel que le processeur $P_{i,j}$ est connecté aux processeurs $P_{i\pm1,j}$ et $P_{i,j\pm1}$



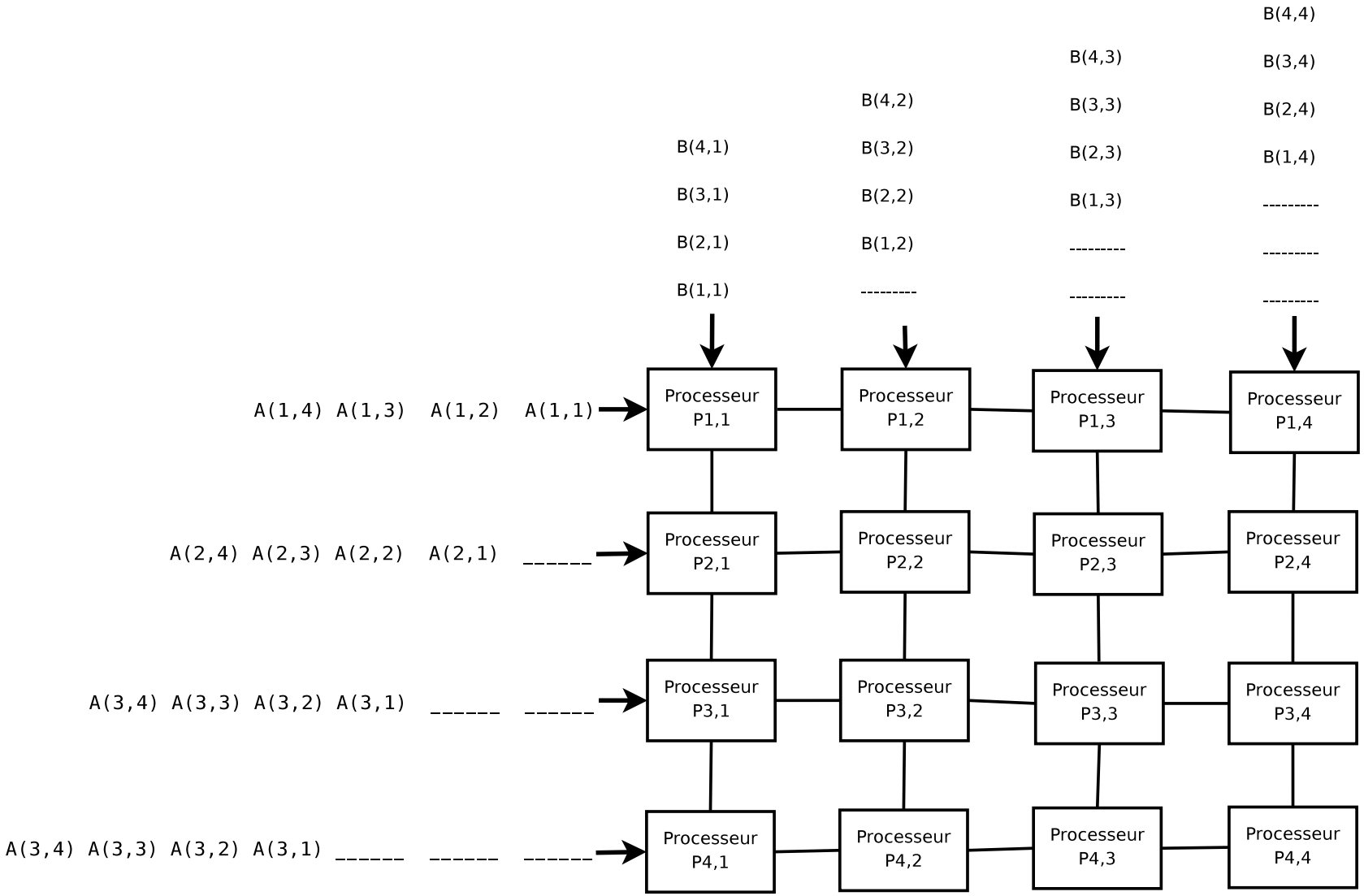
Maillage 2D

- Diamètre $= 2\sqrt{p} - 2$
- Degré maximum $= 4$
- Caractéristiques intéressantes
  - supporte plusieurs types d'algorithmes synchrones et asynchrones
  - simplicité
  - régularité
  - capacité d'expansion
  - correspond bien à la structure de calcul de plusieurs applications
- Toutefois, étant donné son diamètre, tous les calculs non triviaux requiert $\Omega(\sqrt{p})$ étapes

Exemple : multiplication de matrices carrés

- On veut calculer $C = A \times B$ avec des matrices $n \times n$
- On le résout avec un modèle systolique
  - Les rangées de A entrent de façon synchrone à gauche
  - les colonnes de B entrent de façon synchrone au sommet
- Quand $P_{i,j}$ reçoit deux entrées $A(i, l)$ et $B(l, j)$
  - calcule $C(i, j) = C(i, j) + A(i, l) \times B(l, j)$
  - envoie $A(i, l)$ à sa droite
  - envoie $B(l, j)$ vers le bas
- Après n étapes $P_{i,j}$ contiendra les résultat $C_{i,j}$

Maillage 2D

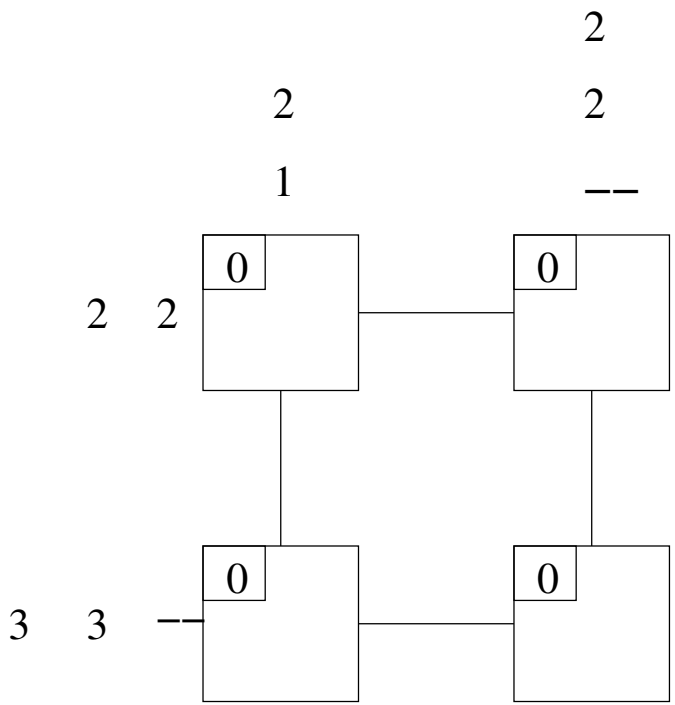

B(4,4)
B(3,4)
B(2,4)
B(1,4)

B(4,3)
B(3,3)
B(2,3)
B(1,3)

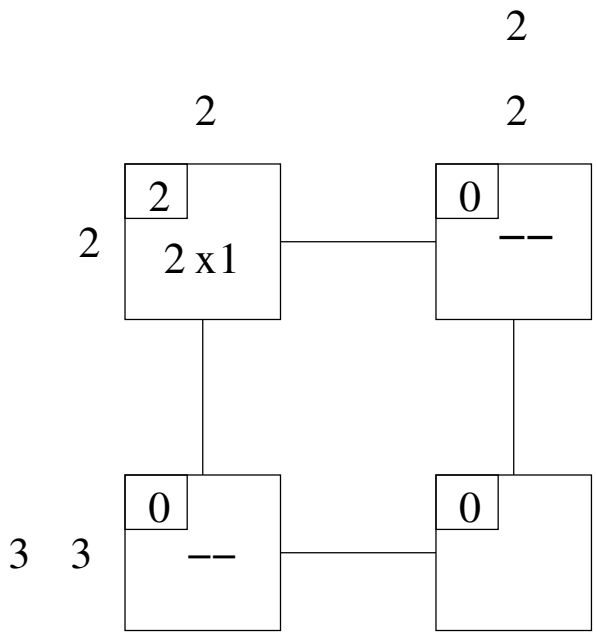
B(4,2)
B(3,2)
B(2,2)
B(1,2)

B(4,1)
B(3,1)
B(2,1)
B(1,1)
A(1,4) A(1,3) A(1,2) A(1,1)
A(2,4) A(2,3) A(2,2) A(2,1) ______
A(3,4) A(3,3) A(3,2) A(3,1) ______ ______
A(3,4) A(3,3) A(3,2) A(3,1) ______ ______ ______
Processeur P1,1
Processeur P1,2
Processeur P1,3
Processeur P1,4
Processeur P2,1
Processeur P2,2
Processeur P2,3
Processeur P2,4
Processeur P3,1
Processeur P3,2
Processeur P3,3
Processeur P3,4
Processeur P4,1
Processeur P4,2
Processeur P4,3
Processeur P4,4

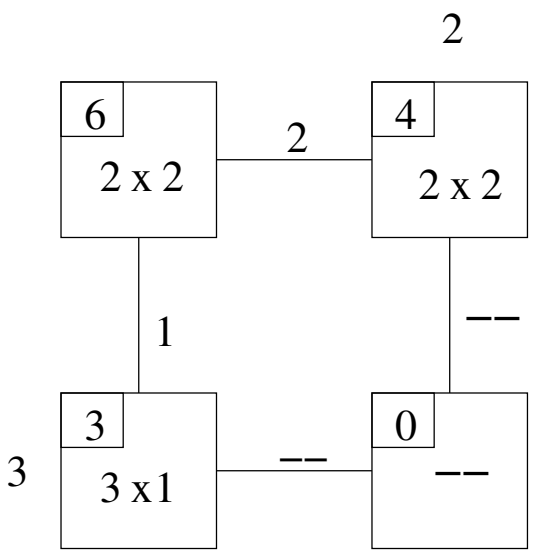
Exemple



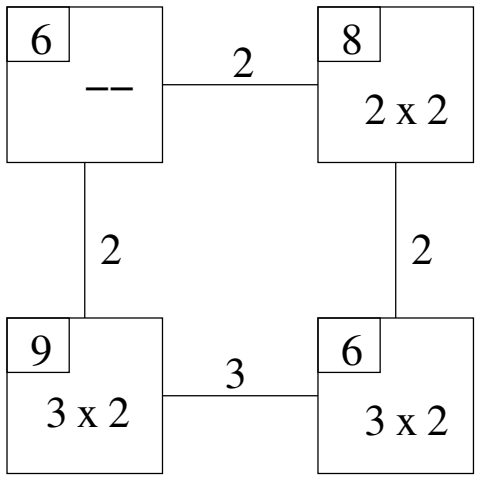
Exemple



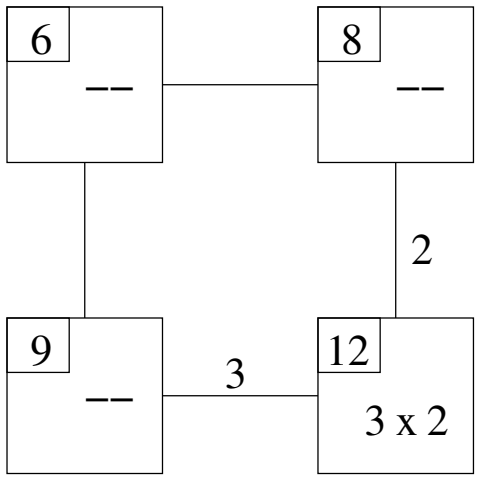
Exemple



Exemple



Exemple



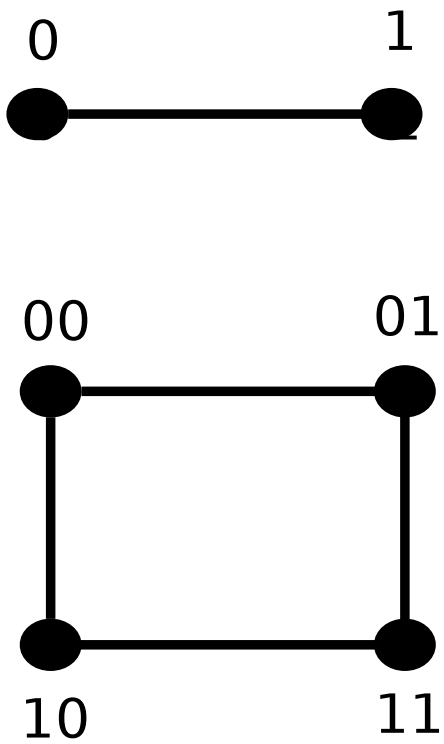
Hypercube

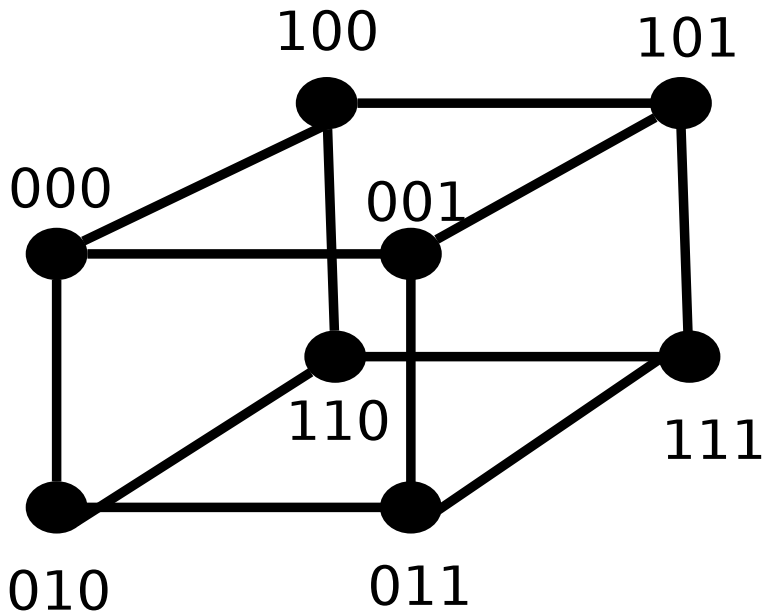
- Consiste en $p = 2^d$ processeurs connectés en un cube booléen de dimension d
- Pour chaque p_i on a la représentation binaire suivante de i
  $i_{d-1} i_{d-2} \ldots i_0$
- Le processeur p_i est connecté aux processeurs $p_{i^{(j)}}$ où
  $i^{(j)} = i_{d-1} \ldots \overline{i_j} \ldots i_0$
  où $\overline{i_j} = 1 - i_j$ pour $0 \leq j \leq d - 1$
- Deux processus sont connectés si leurs indices différent d'un bit

Hypercube

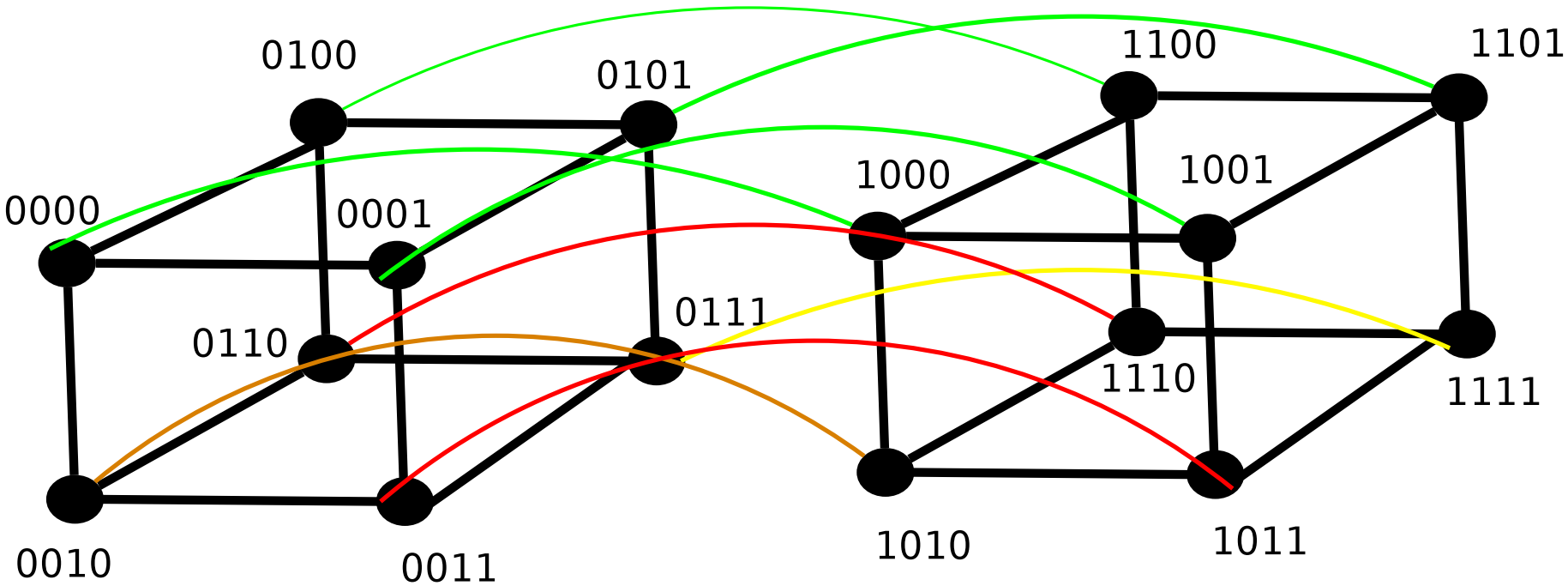
- C'est une structure récursive : on crée la dimension $d+1$ en connectant deux cubes de dimension d : un cube a le bit le plus significatif à 0 et l'autre à 1
- Diamètre : $log_2 p$
- Degré maximum $= log_2 p$
- Caractéristiques : régularité, petit diamètre, rapide en calcul, propriétés reliées à la théorie des graphes

Hypercube : structure

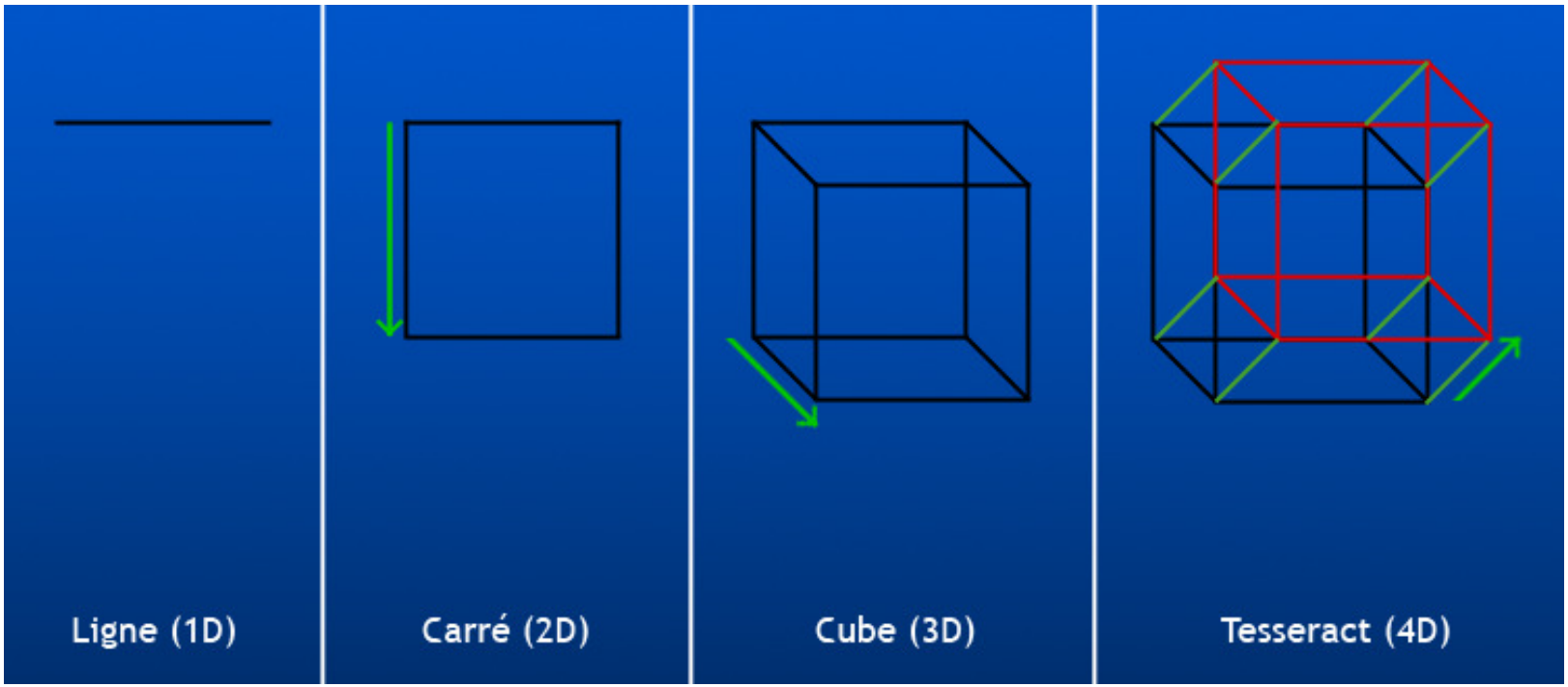




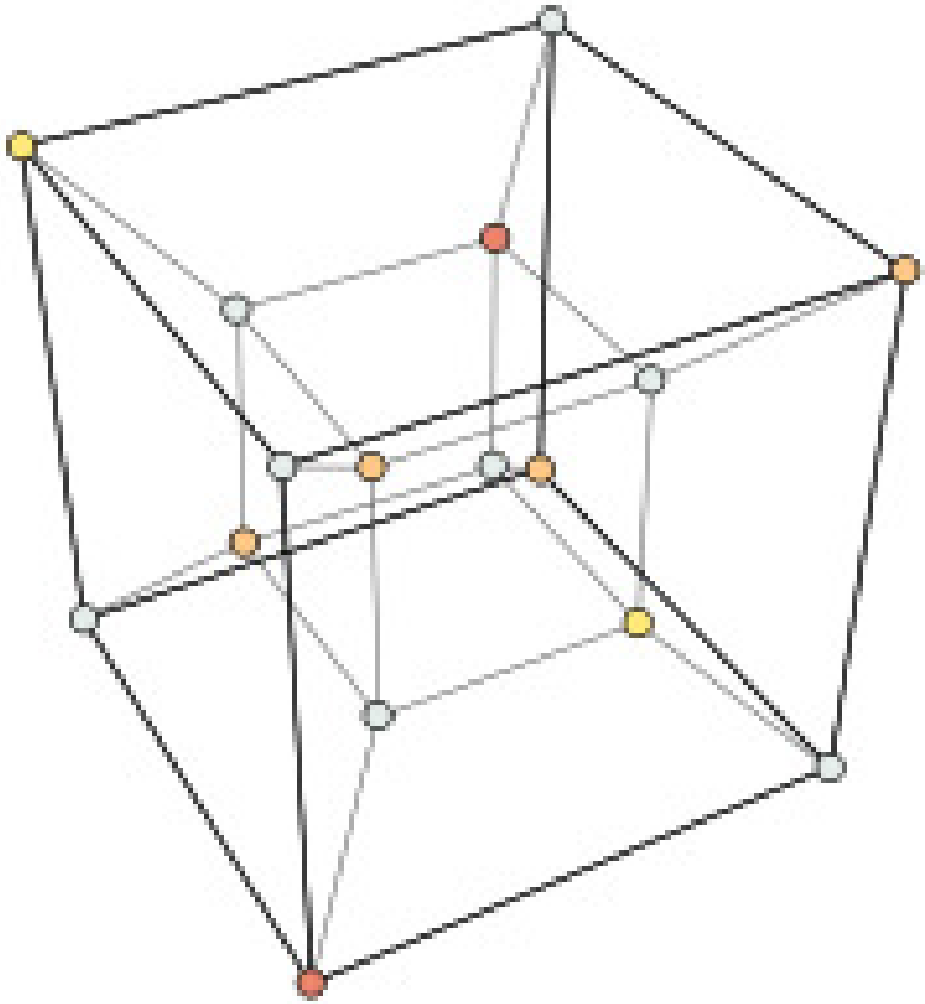
Hypercube : structure



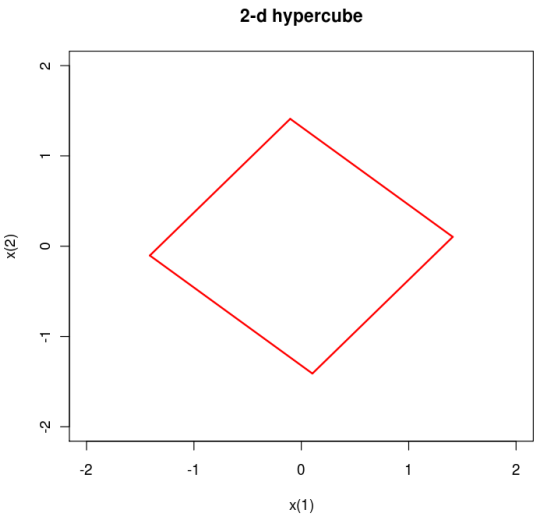
Hypercube : structure

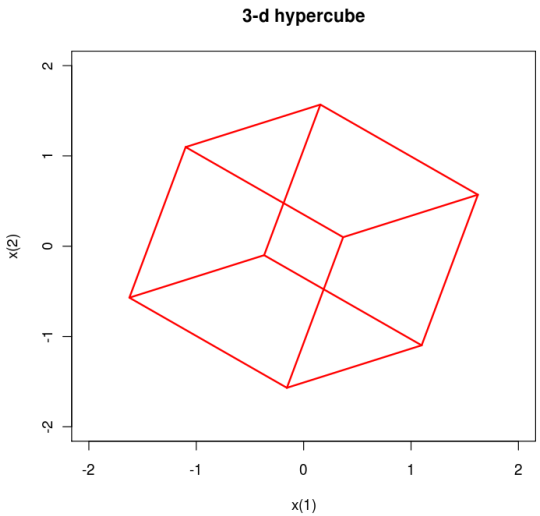


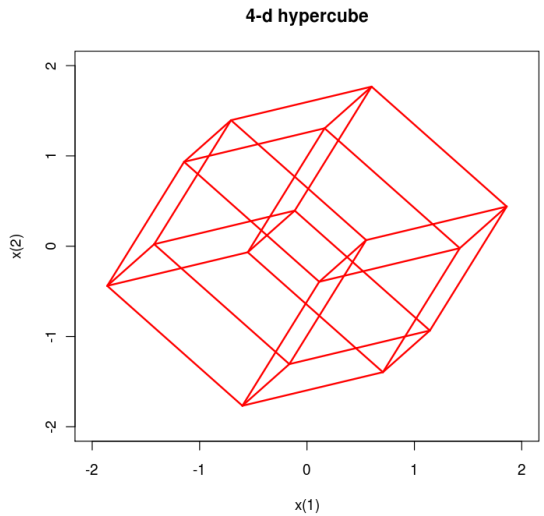
Hypercube : structure

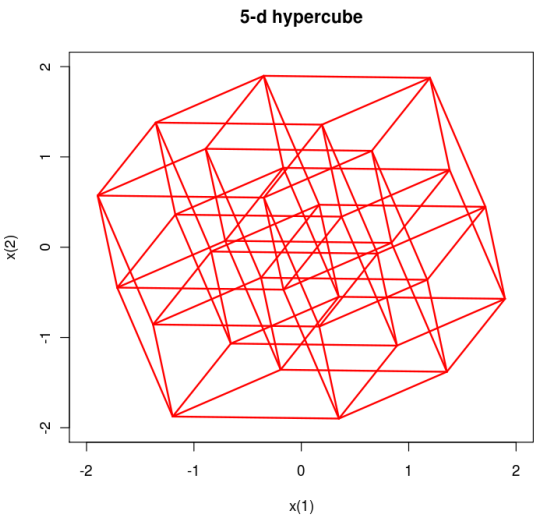


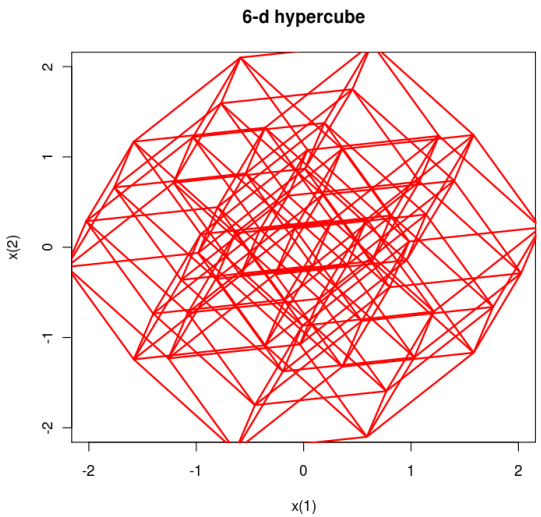
Hypercube : structure

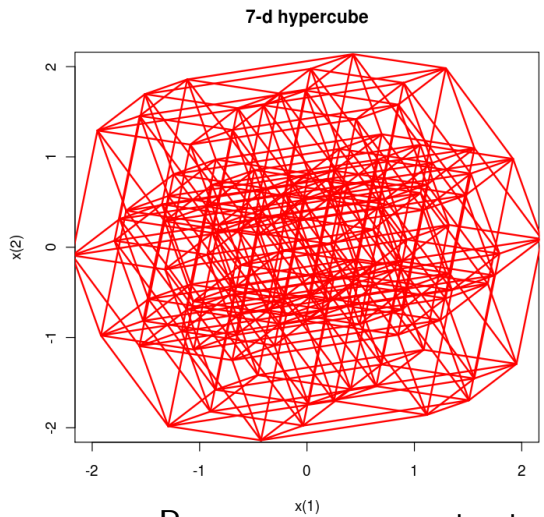












Exemple : somme sur un hypercube

- Soit A un tableau de n éléments
- On a n processeurs ($n = 2^d$)
- On veut $S = \sum_{i=0}^{n-1}(A_i)$ sur le processeur p_0
- On emmagasine chaque A_i sur un processeur p_i
- On fait d itérations
  1. calcule la somme de paires (sur le cube dD) - les sommes sont calculées et emmagasinées dans le cube $(d-1)D$ dont le bit le plus significatif est à 0.
  2. calcule la somme de paires (sur le cube $(d-1)D$) - les sommes sont calculées et emmagasinées dans le cube $(d-2)D$ dont les deux bits les plus significatifs sont à 0
  3. ...

Exemple : somme

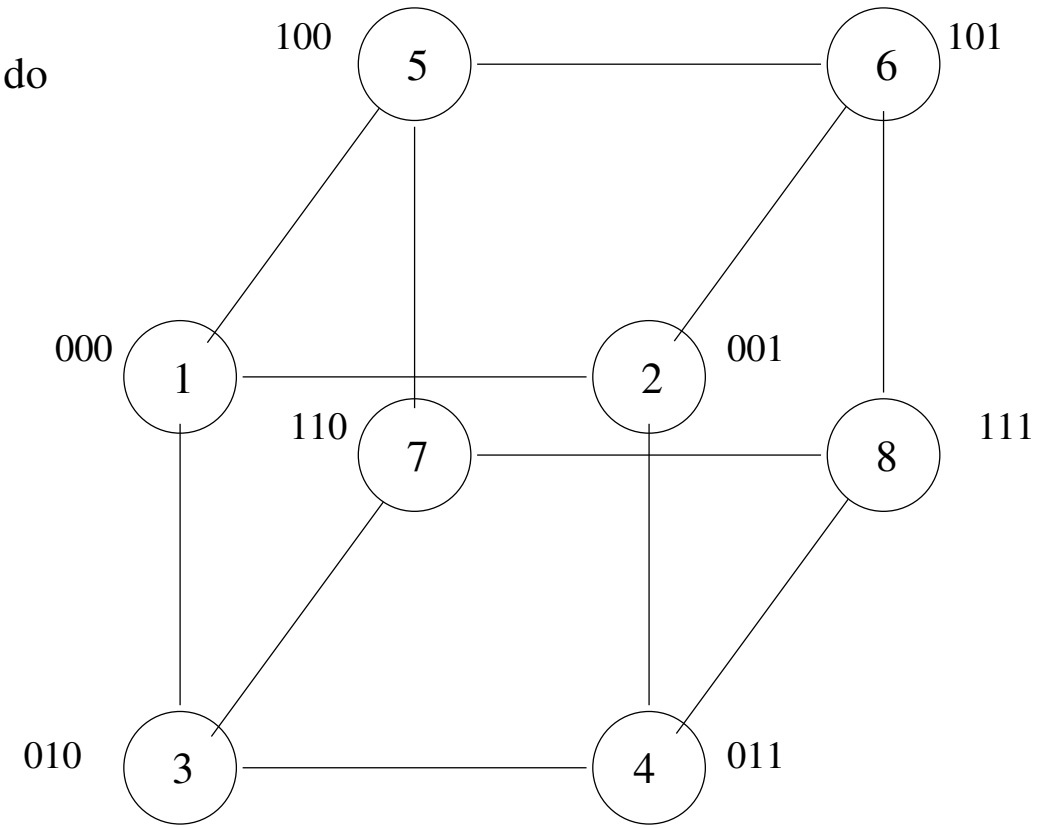
```
Entrée : A(i) sur chaque pi
Sortie : S : somme des A(i) sur p0
begin
  For l = d-1 to 0 do
     if (0<= i <= (2^l)-1) do
          A(i) := A(i) + A(i~l))
end
```

- $A(i) := A(i) + A(j)$ implique que p_i copie p_j vers p_i puis fait l'addition
- `2^l` $\rightarrow 2^l$ (2 à la puissance `l`)
- `i~l` $\rightarrow$ Complément du bit en position i (`000~1 = 010`)

d = 3

i~l = complément du bit en position l

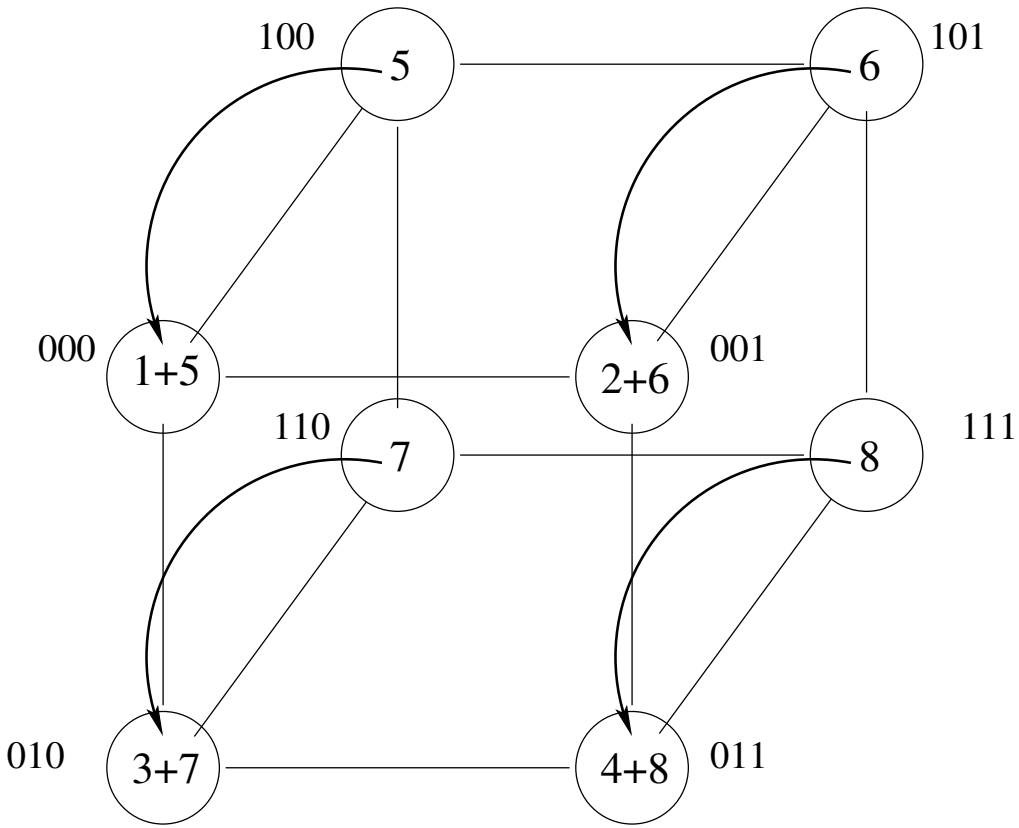
```
for l = d−1 to 0 do
   if ( 0 <= i <= (2^l)−1 ) do
      A[i] = a[i] + A[i~l]
```



d = 3
l = 2
i~2 = complément du bit en position 2

```
for l = 2 to 0 do
   if (0 <= i <= 3) do
      A[i] = A[i] + A[i~2]
```

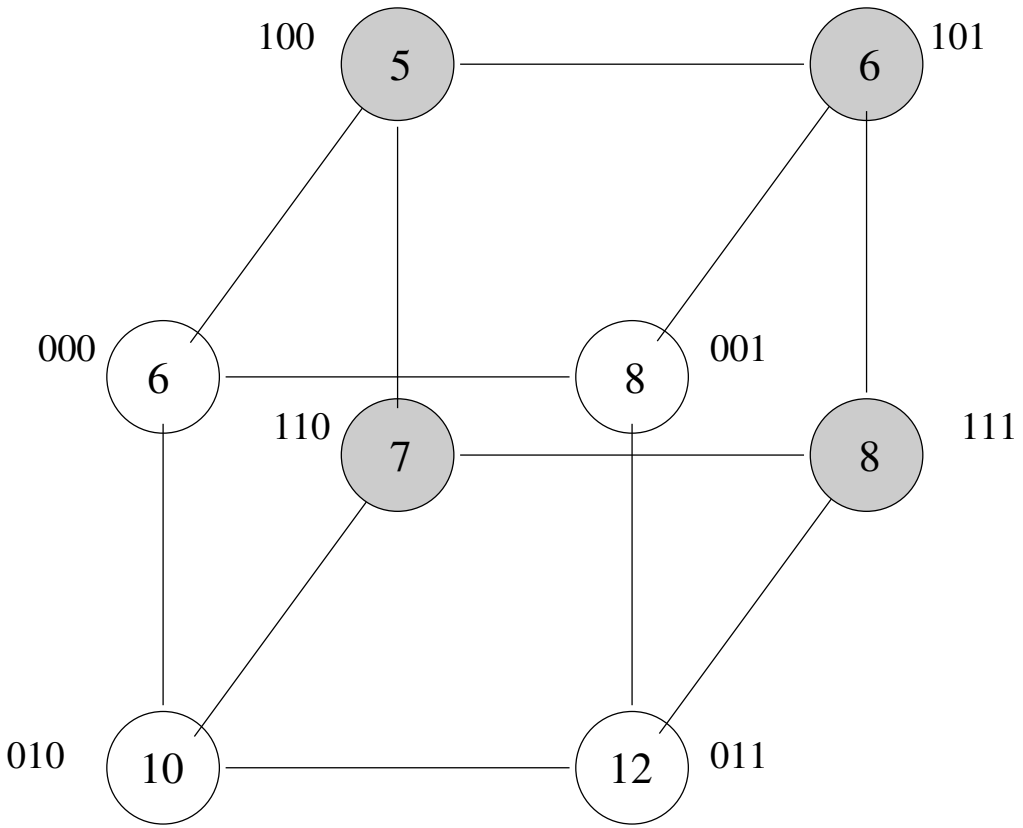
A[0] = A[0] + A[4]
A[1] = A[1] + A[5]
A[2] = A[2] + A[6]
A[3] = A[3] + A[7]



d = 3
l = 2
i~2 = complément du bit en position 2

```
for l = 2 to 0 do
   if (0 <= i <= 3) do
      A[i] = A[i] + A[i~2]
```

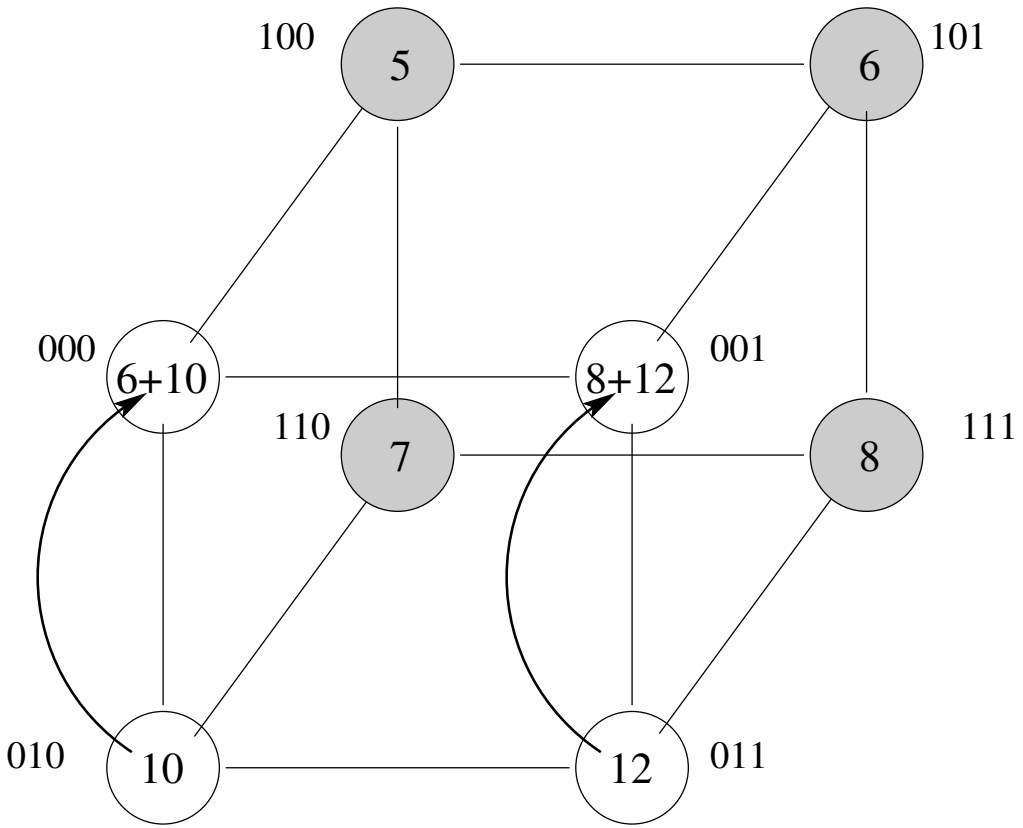
A[0] = A[0] + A[4]
A[1] = A[1] + A[5]
A[2] = A[2] + A[6]
A[3] = A[3] + A[7]



d = 3
l = 1
i~1 = complément du bit en position 1

```
for l = 2 to 0 do
   if (0 <= i <= 1) do
      A[i] = A[i] + A[i~1]
```

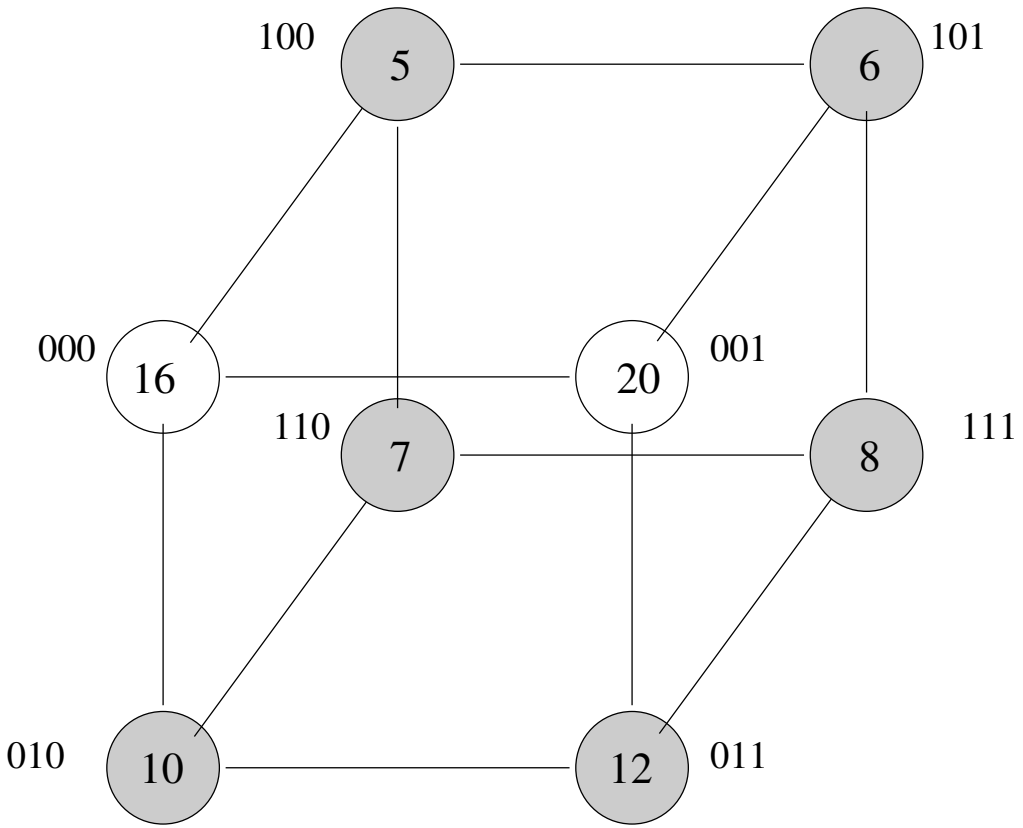
A[0] = A[0] + A[2]
A[1] = A[1] + A[3]



d = 3
l = 1
i~1 = complément du bit en position 1

```
for l = 2 to 0 do
   if (0 <= i <= 1) do
      A[i] = A[i] + A[i~1]
```

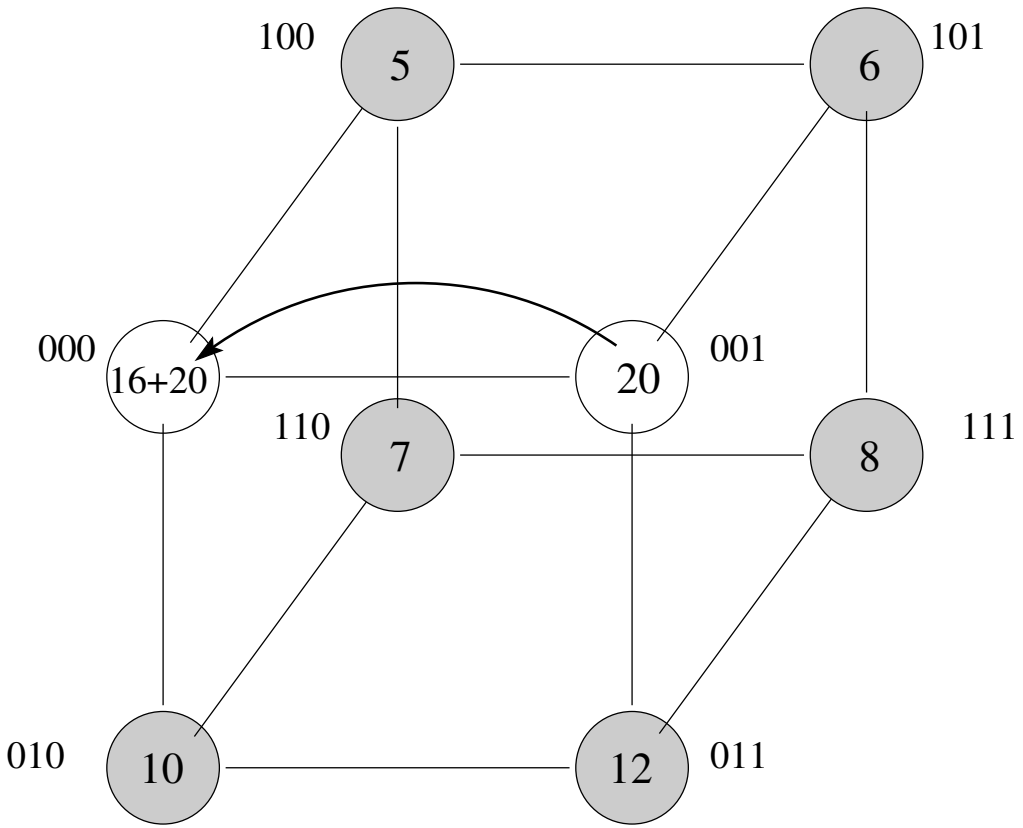
A[0] = A[0] + A[2]
A[1] = A[1] + A[3]



d = 3
l = 0
i~0 = complément du bit en position 0

for l = 2 to 0 do
  if (0 <= i <= 0) do
    A[i] = A[i] + A[i~0]

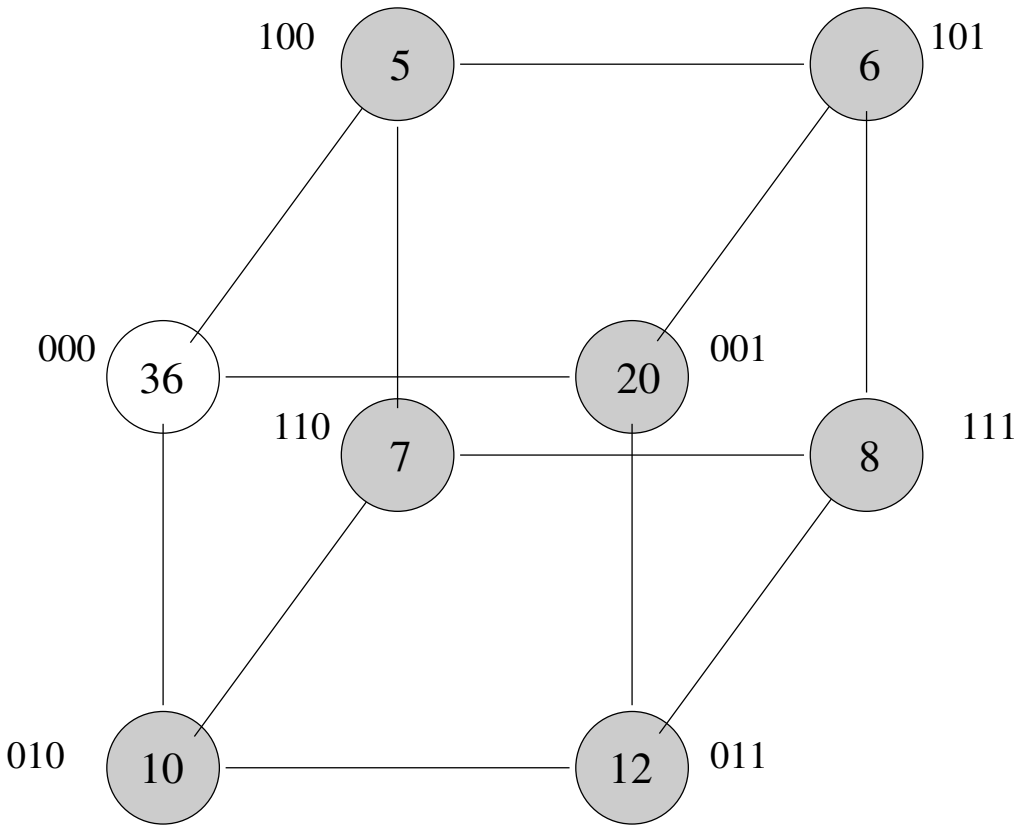
A[0] = A[0] + A[1]



d = 3
l = 0
i~0 = complément du bit en position 0

```
for l = 2 to 0 do
   if (0 <= i <= 0) do
      A[i] = A[i] + A[i~0]
```

A[0] = A[0] + A[1]



Exemple : somme sur un hypercube

- Complexité $= O(\log_2 n)$: dimension du cube
- Si $n = 8$, itération no.
  1. $[0]A(0) := A(0) + A(4), [1]A(1) := A(1) + A(5), ...$
  2. $[0]A(0) := (A(0) + A(4)) + (A(2) + A(6)), [1]...$
  3. $[0]A(0) :=$ somme

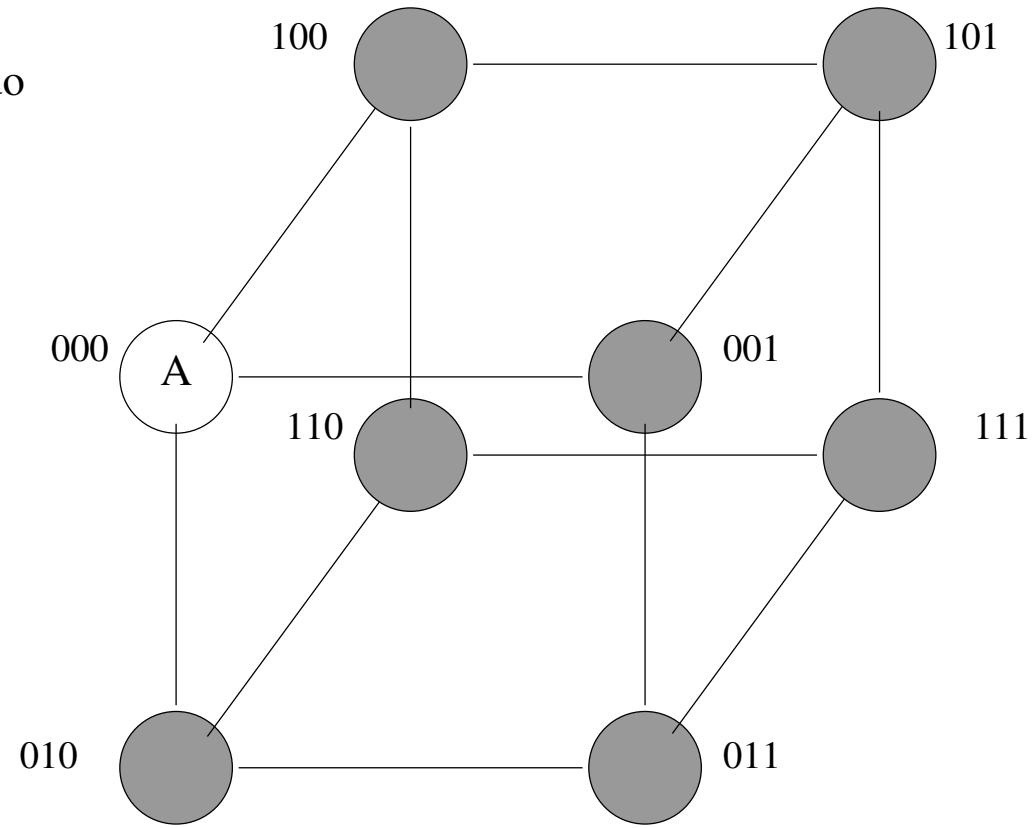
Exemple : diffusion

```
Entrée : p0 contient x dans D(0)
Sortie : pi contient x : pour tout i
begin
  For l = 0 to d-1 do
     if (0<= i <= (2^l)-1) do
          D(i~l) := D(i)
end
```

d = 3

i~l = complément du bit en position l

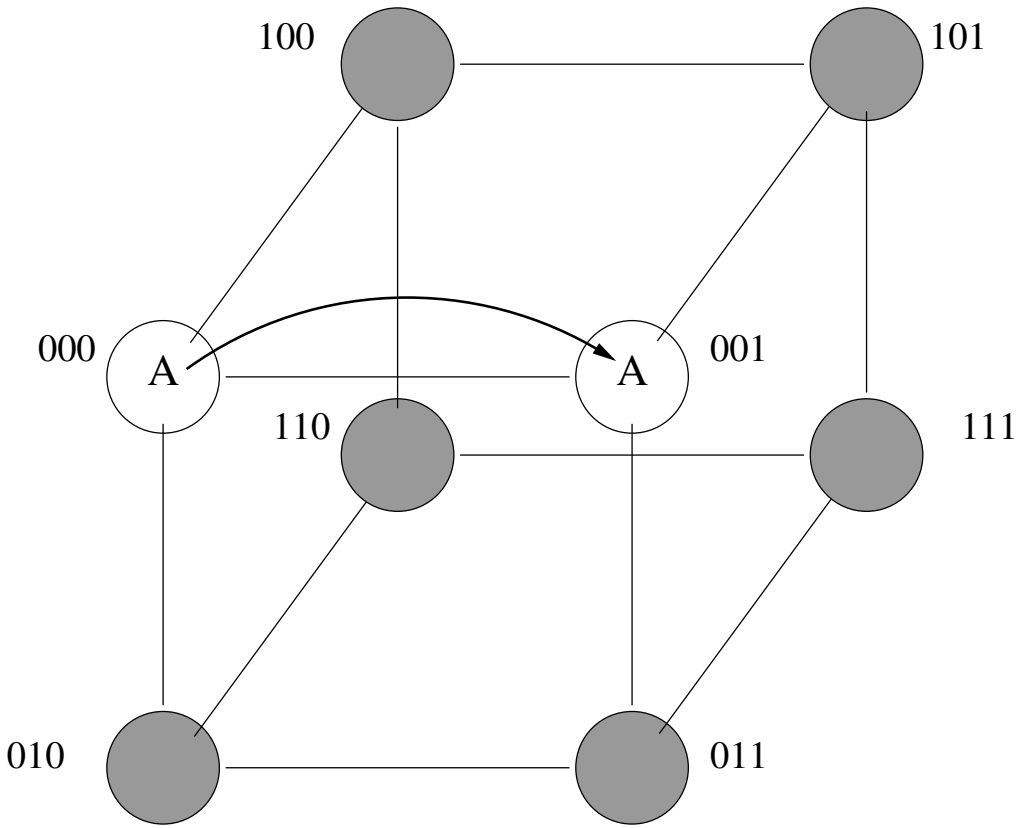
```
for l = 0 to d−1 do
   if (0 <= i <= (2^l)−1) do
      D[i~l] = D[i]
```



d = 3
l = 0
i~0 = complément du bit en position 0

for l = 0 to d−1 do
  if (0 <= i <= 0) do
    D[i~0] = D[i]

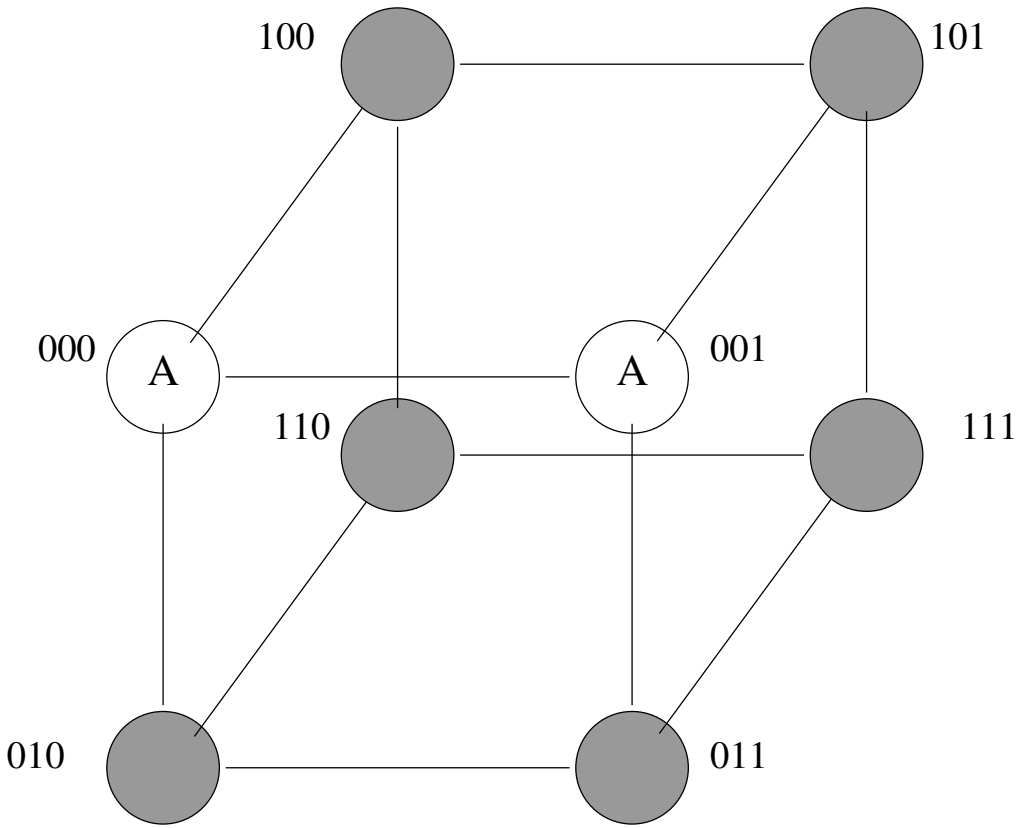
D[1] = D[0]



d = 3
l = 0
i~0 = complément du bit en position 0

for l = 0 to d−1 do
  if (0 <= i <= 0) do
    D[i~0] = D[i]

D[1] = D[0]

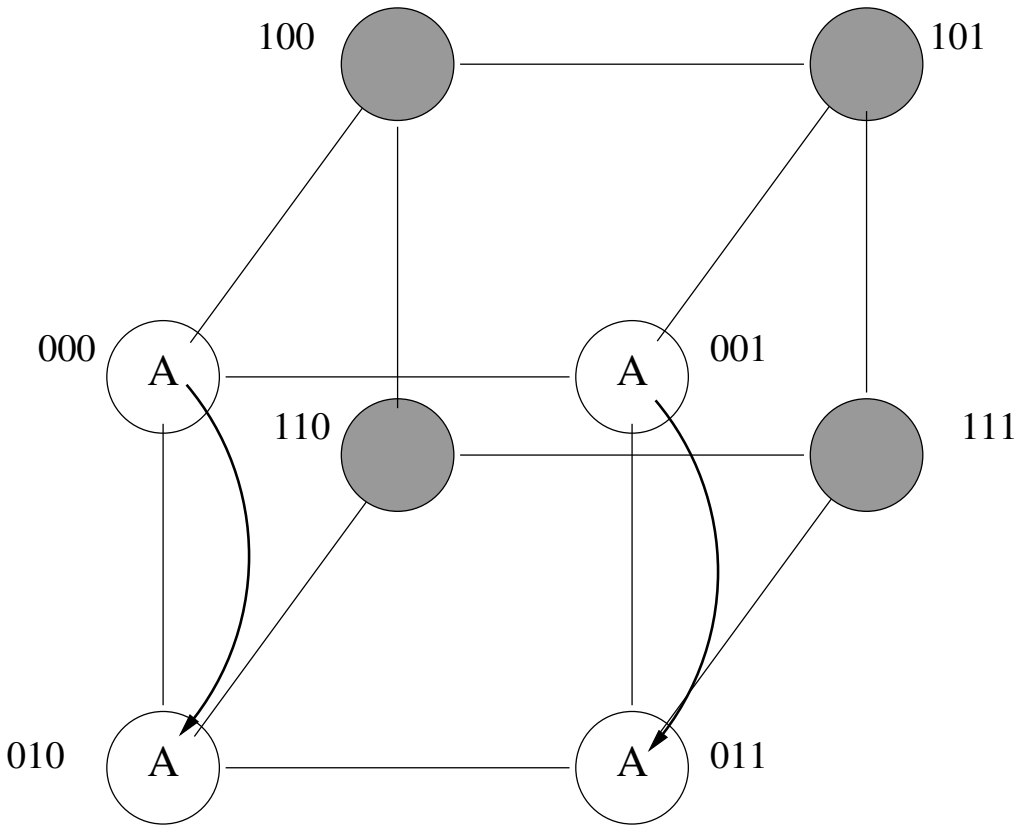


d = 3
l = 1
i~1 = complément du bit en position 1

for l = 0 to 2 do
  if (0 <= i <= 1) do
    D[i~1] = D[i]

D[2] = D[0]

D[3] = D[1]

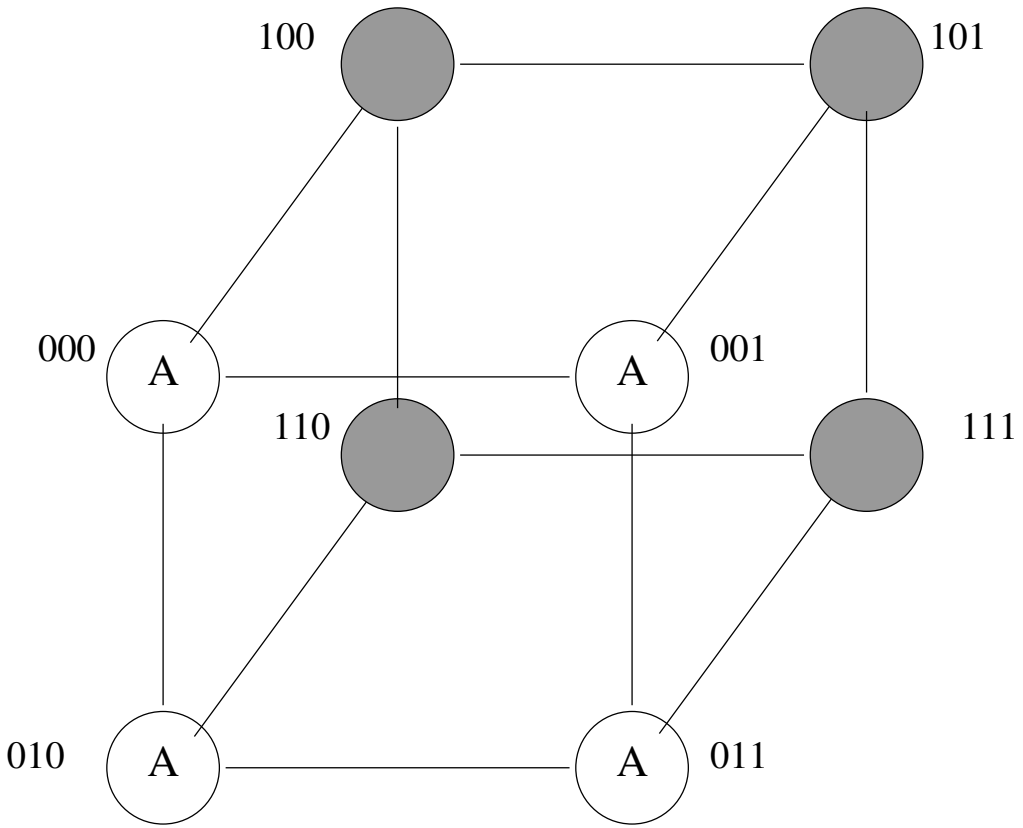


d = 3
l = 1
i~1 = complément du bit en position 1

for l = 0 to 2 do
  if (0 <= i <= 1) do
    D[i~1] = D[i]

D[2] = D[0]

D[3] = D[1]



d = 3
l = 2
i~2 = complément du bit en position 2

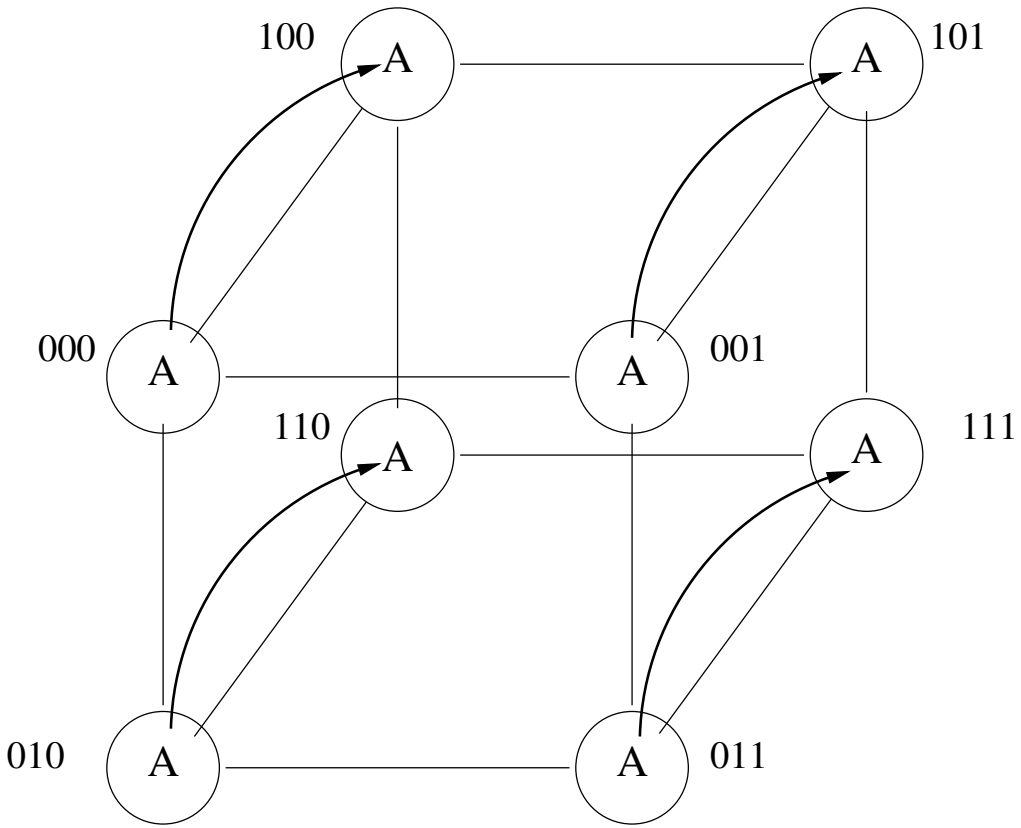
```
for l = 0 to 2 do
   if (0 <= i <= 2) do
      D[i~2] = D[i]
```

D[4] = D[0]

D[5] = D[1]

D[6] = D[2]

D[7] = D[3]



d = 3
l = 2
i~2 = complément du bit en position 2

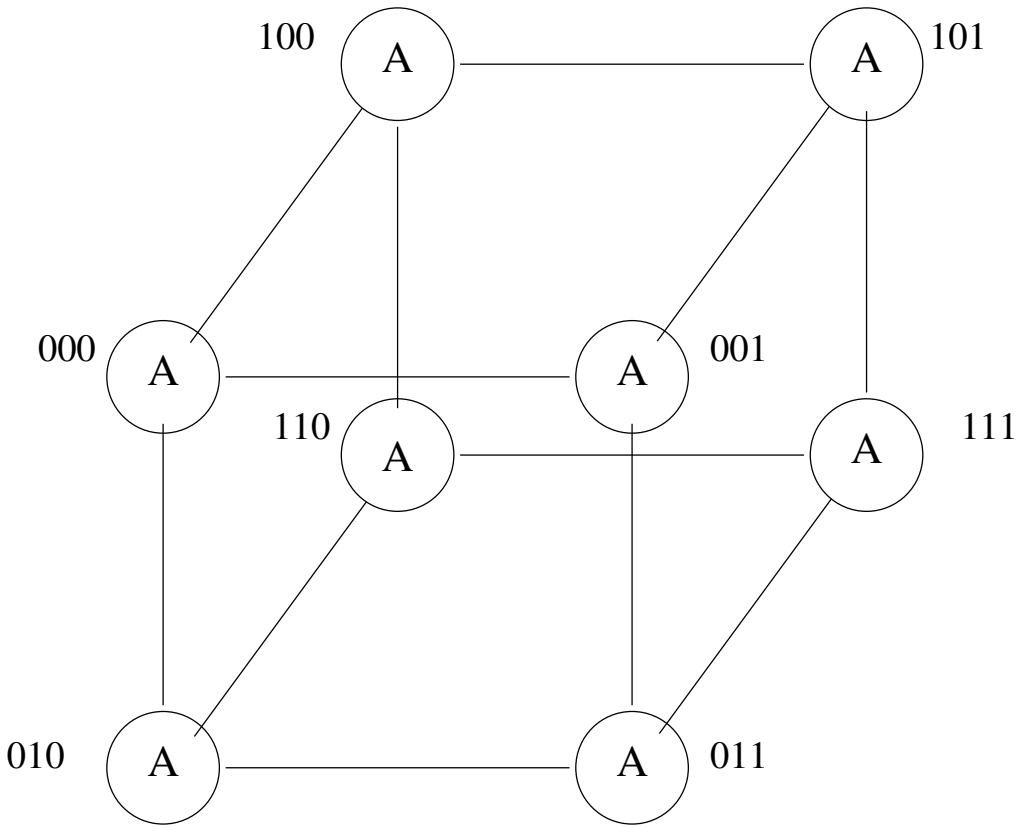
```
for l = 0 to 2 do
   if (0 <= i <= 2) do
      D[i~2] = D[i]
```

D[4] = D[0]

D[5] = D[1]

D[6] = D[2]

D[7] = D[3]



Exemple : diffusion

- Complexité $= O(\log_2 p)$

Types d'algorithmes

- Algorithmes normaux : utilise une dimension à chaque unité
- Algorithmes complètement normaux : utilise des dimensions consécutives en séquence dans le temps

Exemple : multiplication de matrices

- Soit A, B, C des matrices carrés nxn
- On veut calculer $C = A \times B$ sur un hypercube avec $p = n^3$ processeurs
- Soit $n = 2^q$ alors $p = 2^{3q}$
- Chaque processeur est indexé (l, i, j)
- $p_{l,i,j}$ représente p_r où $r = ln^2 + in + j$ les q bits les plus significatifs = l les q bits les moins significatifs = j
- Si on fixe deux des trois valeurs, on obtient un cube qD

Exemple : multiplication de matrices

- A est emmagasiné dans le cube formé par les processeurs $p_{l,i,0} : 0 \leq i, l \leq n-1$ tel que $A(i,l)$ est sur le processeur $p_{l,i,0}$
- B est emmagasiné dans le cube formé par les processeurs $p_{l,0,j} : 0 \leq j, l \leq n-1$ tel que $B(l,j)$ est sur le processeur $p_{l,0,j}$
- On veut $C(i,j) := \sum_{l=0}^{n-1} A(i,l) \times B(l,j)$, $0 \leq i, j \leq n-1$
- Trois étapes :
  1. diffusion des données : chaque $p_{l,i,j}$ contient $A(i,l)$ et $B(l,j)$
  2. $p_{l,i,j}$ calcule $C'(l,i,j) = A(i,l) \times B(l,j)$ $\forall 0 \leq i, j, l \leq n-1$
  3. $p_{l,i,j}, 0 \leq l \leq n-1$ calcule $C(i,j) = \sum_{l=0}^{n-1} C'(l,i,j)$

Exemple : multiplication de matrices

1. Diffusion (algorithme précédant) : $O(log_2 n)$
2. Multiplication sur chaque $p_{l,i,j}$: $O(1)$
3. Calcul de n^2 sommes $C(i,j)$: $O(log_2 n)$
4. Complexité totale : $O(log_2 n)$

Évaluation : DAG

- Les DAG s'applique à une classe spécialisée d'algorithmes (algorithmes réguliers)
- Les DAG représente seulement une information partielle
  - Ne parle pas de planification et d'allocation
  - Ne contient aucun mécanisme pour gérer les communications

Évaluation : Réseau

- Le modèle réseau est plus approprié que DAG pour modéliser le calcul et les communications
- La description et l'analyse des algorithmes sont complexes
- Les algorithmes dépendent de la topologie

Évaluation : PRAM

- PRAM semble le plus puissant
- Plusieurs techniques existent pour travailler les différentes classes de problèmes
- Il élimine les détails algorithmiques reliés à la synchronisation et la communication
- Il capture plusieurs paramètres du calcul parallèle : planification et allocation
- Il est robuste
  Plusieurs algorithmes réseau dérivent des algorithmes PRAM et les algorithmes PRAM peuvent être projeté sur certains types de réseau
- On peut incorporer la synchronisation et la communication dans le mode

Coût

- Soit Q un problème à résoudre sur PRAM qui s'exécute en temps $T_p(n)$ sur p processeurs pour des données de taille n
- Le **coût** de l'algorithme parallèle est $C(n) = T_p(n) \times p$
- Il peut être converti en algorithme séquentiel qui s'exécute en temps $O(C(n))$
  Un processeur simule p processeurs en temps $O(p)$ pour chaque étape $T_p(n)$

Mesures

Exemple 1

- Temps exécution séquentiel $= T^*(n) = 20$
- Temps exécution parallèle $= T_{10}(n) = 4$
- Temps exécution sur 1 pcsr $= T_1(n) = 25$
- Accélération $= S_{10}(n) = \frac{20}{4} = 5$
- Accélération relative $= RS_{10}(n) = \frac{25}{4} = 6,25$
- Efficacité $= E_{10}(n) = \frac{25}{10 \times 4} = 0.625$
- Coût $= C(n) = 4 \times 10 = 40$

Coût

Pour l'algorithme qui calcule la somme de n éléments sur PRAM, cela signifie que les énoncés suivants sont équivalents :

- $p\ (= n)$ processeurs et $O(log_2 n)$ unités de temps
- Coût $C(n) = O(p\ log_2 n)$ et temps de $O(log_2 n)$

$C(n) = O(n\ log_2 n)\ si\ p = n$

Coût

Pour l'algorithme qui calcule la multiplication de matrices sur PRAM, cela signifie que les énoncés suivants sont équivalents :

- n^3 processeurs et $O(log_2 n)$ unités de temps
- Coût $C(n) = O(n^3\ log_2 n)$ et temps de $O(log_2 n)$

WT : Worktime paradigme

- Permet de décrire un algorithme en 2 niveaux
  1. WT presentation : niveau élevé qui supprime les détails
  2. WT planification : fourni les détails de planification

WT presentation : le travail

- Décrit l'algorithme en terme d'unités de temps incluant chacune un nombre quelconque d'opérations concurrentes
- le **travail** $W(n)$ fait par un algorithme parallèle est le nombre total d'opérations utilisées
- Notation utilisée : **for** $l \leq i \leq u$ **pardo**

Exemple : somme de n nombres $n = 2^k$

```
Entree: A : un tableau contenant n nombres
Sortie: S : la somme des nombres
begin
  1 - for 1 <= i <= n pardo B(i) := A(i)
  2 - for h:=1 to log n  do
          for 1 <= i <= n/2^h  pardo
             B(i) := B(2i-1) + B(2i)
  3 - S := B(i)
```

Exemple : somme de n nombres $n = 2^k$

- Cet algorithme ne donne aucune indication sur le nombre de processeurs ni comment les opérations sont allouées aux processeurs
- Il est décrit en terme d'unités de temps, chacune contenant un nombre quelconque d'opérations

Exemple : somme de n nombres $n = 2^k$

- L'algorithme contient :
  - $log_2 n + 2$ unités de temps
  - n opérations dans la première unité de temps (étape 1)
  - $n/2^{j-1}$ opérations : $2 \leq j \leq log_2 n + 1$ à l'unité de temps j (itération $h := j - 1$ de l'étape 2)
  - une opération à la dernière unité de temps

- Le travail de l'algorithme est :

$$W(n) = n + \sum_{j:=1}^{\log_2 n}(n/2^j) + 1 = O(n)$$

- Le temps d'exécution est : $T_p(n) = O(\log_2 n)$

Travail vs Coût

- Soit un algorithme de temps $T_p(n)$ avec un nombre d'opérations total de $W(n)$
- Coût $= C(n) = T_p(n) \times p$
- $W(n)$ mesure le nombre total d'opérations et n'a rien à voir avec le nombre de processeurs
- $C(n)$ mesure le coût de l'algorithme relatif au nombre de processeurs

Travail vs Coût : somme de n valeurs

- Soit un algorithme qui calcule la somme de n nombres
- On a
  - $W(n) = O(n)$
  - $T_p(n) = O(log_2 n)$
  - $C(n) = O(p \cdot log_2 n) = O(n \cdot \ log_2 n)$ *si* $p = n$
- Avec n processeurs, au plus $n/2$ processeurs sont actifs au premier niveau, au plus $n/4$ au niveau suivant, etc.
- L'algorithme requiert $O(n)$ opérations mais n'utilise pas efficacement les processeurs

Notion d'optimalité

- Soit un problème Q dont la complexité séquentielle optimale théorique est $T^*(n)$
- En parallélisme on définit deux notions d'optimalité
  - Si le **travail** $W(n)$ satisfait $W(n) = \Theta(T^*(n))$
  - Si **l'accélération** $S_p(n) = \Theta(p)$

Exemple : Somme

- $T(n) = O(log_2 n)$
- $W(n) = O(n)$
- Comme $T^*(n) = n$ l'algorithme est optimal (pour le travail)
- L'accélération optimale est obtenue avec $p = O(n/log_2 n)$ processeurs

Complexité des communications

- C'est le pire cas de traffic entre la mémoire et un processeur queslconque
- Soit A un algorithme optimal adapté avec succès sur p processeurs
- La complexité des communications n'est possiblement pas la meilleure

Exemple : multiplication de matrices

```
Entree: A et B : matrices n x n
        n : entier = 2^k
Sortie: C = A x B
begin
  1 - for  1 <= i,j,l <n pardo
        C'(i,j,l) := A(i,l) x B(l,j)
  2 - for h=1 to log n  do
        for (1 <= i,j <= n , 1<= l <= n/2^h) pardo
            C'(i,j,k) :=  C'(i,j,2l-1) + C'(i,j,2l)
  3 - for 1 <= i,j <n  pardo
         C(i,j) :=  C'(i,j,1)
```

Exemple : multiplication de matrices

- Exécution : $O(log_2 n)$ pour $O(n^3)$ opérations
- Exécution : $O(n^2)$ avec n processeurs
- Communication : $O(n^2)$ avec n processeurs car
  1. Lit une ligne par processeur et un matrice : $O(n^2)$
  2. On lit les produit en mémoire : $O(n^2)$
  3. $O(n)$

Autre allocation possible

- Soit $\alpha = \sqrt[3]{n}$
- On partitionne les matrices A et B en blocs de α
- Taille de chaque bloc : $\sqrt[3]{n} \times \sqrt[3]{n}$
- On obtient n paires de blocs
- Pour la multiplication, chaque processeur lit une paire.
- Performance :
  - Temps de calcul : $O(n^2)$ sur n processeurs
  - Communication : $O(n^{4/3} \cdot log_2 n)$

Structure de données

- Le choix d'une bonne structure de données influence la performance des algorithmes parallèles
- Conception d'algorithmes parallèles
  Il n'y a pas une technique mais plusieurs qui peuvent aider à la conception

- Recherche dans une liste triée : $O(\frac{log(n+1)}{log(p+1)})$ (CREW PRAM)
- Tris parallèles :
  - tri bulle : $O(n)$ sur $O(n)$ processeurs (PRAM EREW)
  - tri insertion : $O(n^2)$
  - heap sort : $O(n \cdot log_2 n)$ (pire cas)
  - tri fusion : $O(log_2^2 n)$ (PRAM CREW) – d'autres variations font mieux
  - tri bitonique : $O(log_2^2 n)$ (PRAM EREW)

Conclusion

- Temps d'exécution séquentiel optimal : $T^*(n)$
- Temps d'exécution sur p processeurs : $T_p(n)$
- Temps d'exécution sur un processeur : $T_1(n)$
- Accélération absolue : $S_p(n)$
- Accélération relative : $RS_p(n)$
- Efficacité : $E_p(n)$
- Coût : $C(n)$
- Travail : $W(n)$