Exercices pour le cours ALPA

P. Quinton, T. Risset

5 janvier 2000

Version Corrigée

Les exercices difficiles sont indiqués par une étoile ($\star$)

1 algorithmique systolique

Prérequis : algorithmique systolique. (source: [QR89])

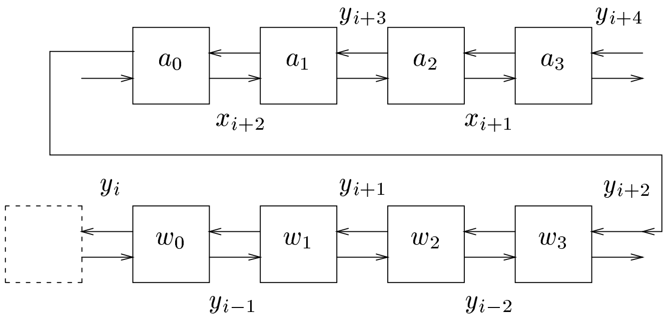
1.1 Convolution généralisée (15 mn)

- Décrire l'architecture systolique pour la convolution généralisée

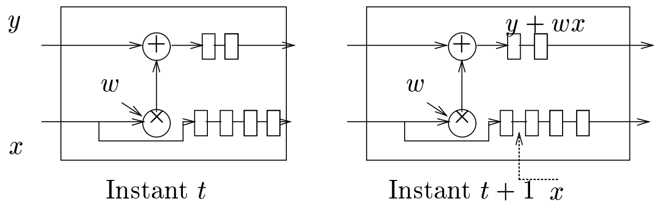
$$y_i = \sum_{k=0}^{3} a_k x_{i-k} + \sum_{k=0}^{3} w_k y_{i-k+1}.$$

- Expliquer pourquoi on est naturellement conduit à utiliser des réseaux de convolution bidirectionnels.
- Expliquer ce qu'on peut faire pour utiliser, malgré tout, un réseau de type unidirectionnel pour la partie non récursive de la convolution.

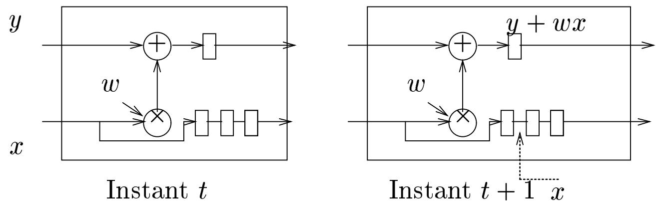
Corrigé: *Le réseau est le suivant, on utilise le réseau de la convolution récursive pour la partie récursive ($\sum_{k=0}^{3} w_k y_{i-k+1}$), on a donc une période de 2 pour les coefficients y. Du coup, on doit utiliser un réseau bidirectionnel aussi pour la partie non récursive car on ne peut pas "accélérer" les y entre les deux réseaux.*



On peut, malgrès tout, utiliser un réseau unidirectionnel (le réseau vu en cours) pour la partie non récursive, mais pour respecter la période des coefficients y, il faut que l'horloge *contrôlant les registres de ce réseau soit deux fois plus lente que l'horloge du réseau bidirectionnel. Si on veut conserver une seule horloge, on ralentit le réseau en doublant le nombre de registres.*



On s'aperçoit alors que l'on peut optimiser cette architecture. En effet, entre deux cellules successives, les y rencontrent deux registres et les x quatre. Le fait de retirer un registre sur les deux canaux en même temps ne modifiera pas les positions relatives des coefficients x et y (cette notion est formalisée par ce qu'on appelle le "théorème de la coupe"). On peut donc passer là l'architecture suivante qui effectue le même traitement que la précédente mais dont les sorties arrivent plus tôt:



1.2 Produit de matrices (15 mn)

- Décrire l'architecture systolique (réseau carré) pour le produit de matrices carrées $C = AB$.
- Expliquer comment modifier ce réseau et son utilisation, de façon que les coefficients de C soient évacués vers la gauche, afin de reboucler les résultats.
- Examiner aussi le cas ou les résultats C sont rebouclés, lorsqu'ils sortent (normalement) par la droite du réseau. Donner le délai nécessaire entre l'entrée de deux matrices successives, et expliquer quel dispositif de mémorisation doit être utilisé.

Corrigé: *à faire*

1.3 Synthèse par équations: transformée en ondelettes (45 mn) ⋆

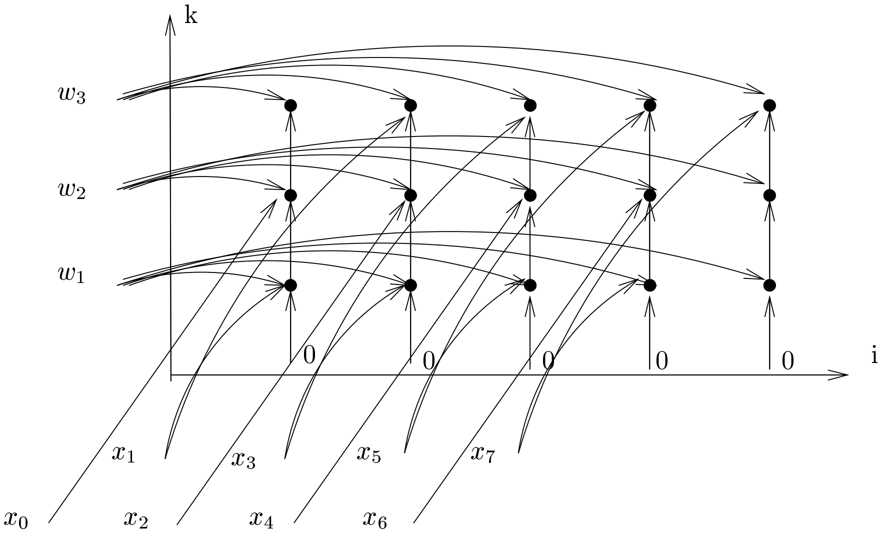
Pré-requis : Synthèse automatique de réseau systoliques

La transformée en ondelettes à récemment beaucoup été étudiée. Elle permet de faire un passage dans le domaine des fréquences de manière hiérarchique. Le calcul se fait en plusieurs *niveaux*, chaque niveau rendant successivement une information *de moins en moins précise*. À chaque niveau, on peut modéliser le calcul effectué lors d'une transformée en ondelettes unidimensionnelle (à N coefficients et par un facteur de deux) par la formule suivante:

$$y(i) = \sum_{k=1}^{N} w(k) * x(2i - k) \tag{1}$$

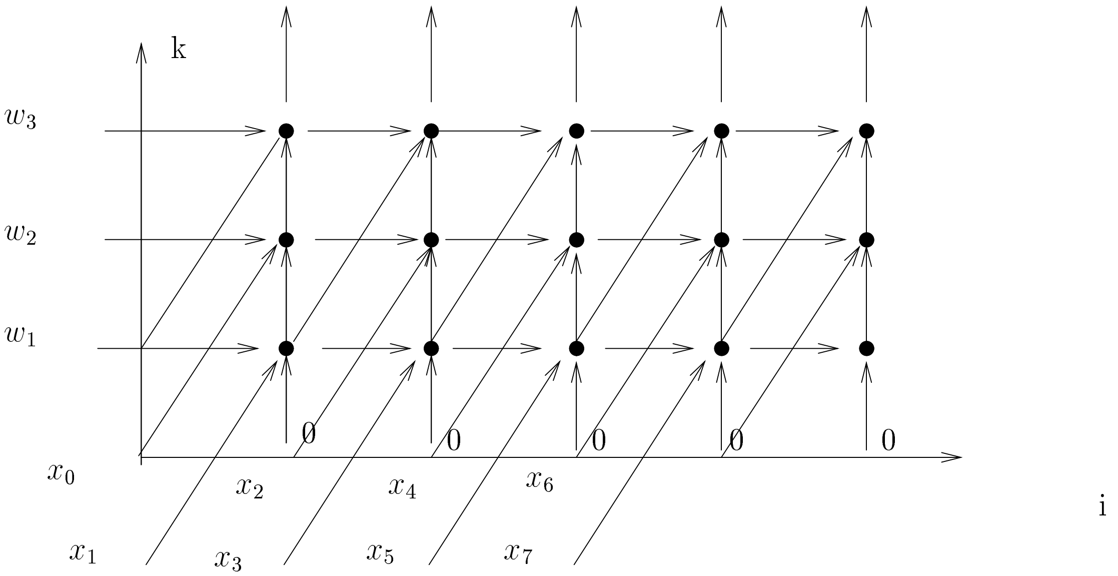
dans laquelle, les $x(j)$ sont des coefficients donnés en entrée, les $y(i)$ sont les résultats, les $w(k)$ sont N coefficients fixés. On va rechercher une architecture systolique pour ce problème. On ne vous demandera pas d'écrire les équations récurrentes au cours de la dérivation de l'architecture, mais d'expliquer informellement sur le graphe de dépendance.

1. De quelle manière peut-on sérialiser la réduction de l'équation (1). Choisissez une de ces possibilités et représentez le graphe de dépendance étendu pour $N = 3$, et $M = 5$ (M est le nombre de coefficient y que l'on calcule).
   Corrigé*:*
   On peut sérialiser dans l'ordre croissant des Y ou dans l'ordre décroissant. Dans l'ordre croissant, cela donne:

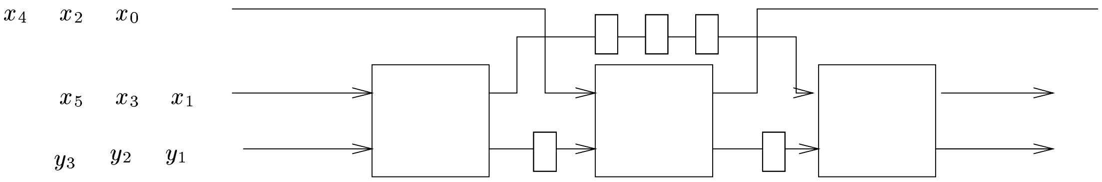


2. énumérez les possibilités pour pipeliner les variables x et w dans ce graphe. Quelles sont, selon vous, les cas à éviter. Choisissez une de ces possibilité et représenter le graphe de dépendance étendu uniformisé pour $N = 3$, et $M = 5$.
   corrigé: *On doit pipeliner les $w(k)$ dans la direction* (0,1) *(deux sens possibles), et les $x(j)$ dans la direction* (1,2) *(deux sens possibles). Il faut éviter de pipeliner en* (0,1) *et* $(-1,-2)$ *ni en* $(0,-1)$ *et* (1,2) *car on ne pourra pas trouver d'ordonnancement affine On choisit (Par exemple) de pipeliner les $w(k)$ selon la direction* (1,0) *(de gauche à droite) et les $x(j)$ selon la direction* (1,2) *(de bas en haut). Le nouveau système d'équations récurrentes obtenu est le suivant:*

$$
\begin{array}{ll}
\{i,k \mid 1 \le i \le M; k = 0\}: & y(i,k) = 0 \\
\{i,k \mid 1 \le i \le M; 1 \le k \le N\}: & y(i,k) = y(i,k-1) + W(i,k) * X(i,k) \\
\{i,k \mid i = 0; 1 \le k \le N\}: & W(i,k) = w(k) \\
\{i,k \mid 1 \le i \le M; 1 \le k \le N\}: & W(i,k) = W(i-1,k) \\
\{i,k \mid 1 \le i \le M; 1 \le k \le N\}: & X(i,k) = X(i-1,k-2) \\
\{i,k \mid -1 \le k \le 0; -1 \le i \le M\}: & X(i,k) = x(2i-k) \\
\{i,k \mid -1 \le i \le 0; 1 \le k \le N\}: & X(i,k) = x(2i-k)
\end{array}
\tag{2}
$$



3. Donnez les contraintes que doit vérifier une fonction de temps pour le graphe trouvé à la question 2. Choisissez une fonction de temps respectant ces contraintes. Donnez une fonction d'allocation des calculs compatible avec la fonction de temps choisie.
   Corrigé*:*
   $\tau_i + 2\tau_k \geq 1$
   $\tau_k \geq 1$
   $\tau_i \geq 1$
   la fonction de temps: $T(i,k) = i + k$ *est valide car* (1,1) *respecte les trois vecteurs de dépendance:* (1,2),(1,0)et(0,1). *la fonction d'allocation* $A(i,k) = k$ *est valide car* (1,1).(0,1) $= 1 \neq 0$.
4. Décrivez l'architecture obtenue avec les fonctions de temps et d'allocation trouvées à la question précédente. On pourra faire un schéma de l'architecture.
   Corrigé*:*



5. Quelle sont: la période du flot de y, la latence (temps total pour l'exécution d'une instance du problème), la macro période (temps entre l'exécution de deux instances successives de problème)?
   corrigé: *période: 1 (un résultat tous les tops.)*
   latence: $N + M$ *(*`a voir`*)*
   efficacité $\frac{NM}{(N+M)*N} = \frac{M}{N+M}$ *(*`a voir`*)*
   macro période: $M + 1$

2 Arithmétique entière

Prérequis : addition avec anticipation de retenue sur un arbre

2.1 addition (15 mn)

Une architecture est composée d'un arbre binaire complet de processeurs (N feuilles). Chaque processeur feuille possède $Log(N)$ bits d'un nombre de $NLog(N)$ bits. Peut-on avec cette architecture additionner deux nombres de $Nlog(N)$ bits en $\mathcal{O}(Log(N))$ étapes?

Corrigé: *(source: [Lei92], p 239)*

Le problème vient uniquement de la retenue à transmettre de proche en proche. On va utiliser une approche similaire a l'algorithme d'addition avec propagation de la retenue. On a vu que l'on pouvait, avec une structure d'arbre binaire complet dont chaque feuille possède un bits, informer chaque feuille de la retenue entrante à cette feuille en $\mathcal{O}(Log(N))$ étapes. La seule différence ici réside dans le calcul effectif des feuilles ($\mathcal{O}(Log(N))$ au lieu de $\mathcal{O}(1)$ étapes). Considérons l'algorithme suivant:

1. *Chaque feuille détermine si elle génère, transmet ou stoppe une retenue venant de la feuille précédente.*
2. *A partir de cette information, on transmet la retenue entrante à chaque feuille en utilisant l'algorithme "classique" de propagation de la retenue vu en cours.*
3. *Chaque feuille effectue réellement son addition.*

Si on suppose que l'algorithme de propagation de retenue est correct, alors cet algorithme est correct. La complexité est la suivante: 1), $Log(N)$ étapes (il suffit de calculer l'information en partant des bits de poids faible vers les bits de poids fort). 2) $\mathcal{O}(Log(N))$ (vu en cours), 3) $Log(N)$ étapes car il s'agit d'additionner $Log(N) + 1$ bits. On effectue donc bien l'addition en $\mathcal{O}(Log(N))$.

Remarque: On obtient un efficacité très bonne: $\mathcal{O}(N)$ processeurs, $\mathcal{O}(Log(N))$ étapes, d'où un travail de $W = \mathcal{O}(NLog(N))$ et donc un efficacité de $E = \mathcal{O}(1)$) puisque l'algorithme séquentiel pour additionner deux nombre de $Nlog(N)$ bits est en $\mathcal{O}(NLog(N))$.

2.2 Calcul de préfixe (30 mn) $\star$

On rappelle le calcul du préfixe pour une opération $\otimes$ sur une suite x_i ($0 \leq i \leq N$): calcul de $\otimes_{i=1,j} x_i$ pour tout j ($0 \leq j \leq N$).

1. Le calcul de la retenue dans l'algorithme d'addition avec anticipation de retenue peut être vu comme un calcul de préfixe. Pour cela quelle opération préfixons nous?
   Corrigé: *(source: [Lei92], p 37) On préfixe l'opération qui génère les signaux 's' et 'g' dans les feuilles. En effet, l'information présente au bout des $2Log(N)$ étapes est: "arrivé à la feuille courante, une retenue a été générée par toutes les feuilles précédentes (ou pas)". L'opération elle, a lieu sur les trois signaux mais on est sur que le résultat sera 's' ou 'g' car on a introduit un s dans la première feuille. Cette opération de base dit: un 's' annule toujours la retenue ('s') un 'g' génère toujours une retenue ('g') et un 'p' transmet la retenue arrivant. l'opération que nous préfixons est*

donc:

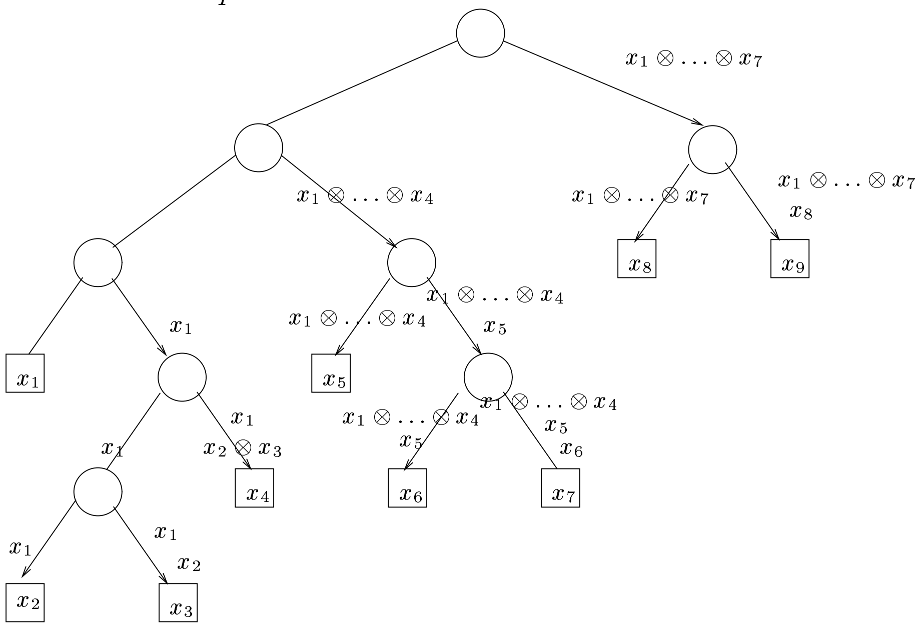
$\otimes$	s	p	g
s	s	s	g
p	s	p	g
g	s	g	g

Notons qu'elle n'est pas commutative, mais qu'elle est associative.

2. En se basant sur l'algorithme d'addition avec propagation de la retenue, proposer un algorithme pour le calcul de préfixe.

Corrigé: *L'algorithme fonctionne en deux phases, d'abord chaque noeud calcul le produit de ses descendants, puis ces produits sont redescendus pour que les feuilles connaissent le produits des feuilles précédentes.*

Plus précisément: On considère un arbre binaire à N feuilles (pas forcément équilibré, c.f. l'arbre ci-dessous), chaque feuille contenant une donnée x_i (on numérote les feuilles de gauche à droite). On doit calculer $y_i = \otimes_{j=1,i} x_i$ (dans l'autre sens que pour l'addition) . Lors du premier pas de l'algorithme, les feuilles passent leur donnée à leur père. Puis, récursivement jusqu'à ce que les données remontent à la racine, chaque noeud non feuille calcule le produit des données transmises par ses fils. Ces noeuds transmettent la donnée calculée à leur père. Chaque noeud non feuille recevant une donné transmise par son fils gauche la transmet à son fils droit et, recevant une donnée de son père, la transmet à ses deux fils. Chaque feuille recevant une donnée de son père la concatène à la donnée qu'elle possède et mémorise le résultat. Sur l'arbre ci dessous, on a représenté la valeur des feuilles au début de l'algorithme ainsi que les valeurs successives qui redescendent vers les noeuds.



Pour prouver que l'algorithme est valide, on raisonne par récurrence sur la profondeur de l'arbre. On doit montrer que la feuille i calcule bien $y_i = x_1 \otimes \ldots \otimes x_i$. On considère que l'algorithme fonctionne pour tous les arbres de profondeur $D-1$ et on considère un arbre de profondeur D. Si $x_1 \ldots x_k$ sont les descendants à gauche de la racine et $x_{k+1} \ldots x_n$ les descendants à droite, si l'on ignore le rôle de la racine, l'algorithme *calculera pour* $0 \leq i \leq k$, $y_i = x_1 \otimes \ldots \otimes x_i$ *et pour* $k+1 \leq i \leq n$, $y_i = x_{k+1} \otimes \ldots \otimes x_i$. *En tenant compte de l'activité de la racine, la seule action de celle ci sera de transmettre* $x_1 \otimes \ldots \otimes x_k$ *de son fils gauche vers son fils droit. Cette valeur devient la dernière valeur reçue par les feuilles (cela car la racine ne transmet cette valeur qu'après avoir reçu la donnée de son fils droit, donc toutes les données devant arriver aux feuilles du fils droit arriverons avant la donnée de la racine) qui la concatènent à leur valeur stockée:* $x_{k+1} \otimes \ldots \otimes x_i$ *et l'on obtient le bon résultat. Le cas D=1 est évident à traiter.*

3 Réseaux

PRÉ-REQUIS : HYPERCUBE, COMPLEXITÉ DES OPÉRATIONS DE COMMUNICATIONS DE BASE

3.1 plongement d'un arbre binaire complet dans un hypercube

1. Montrer que l'hypercube de N noeud ne contient pas un arbre complet binaire de $N-1$ noeud ($\star$).
   Corrige: *(source: [Lei92], p 404) On utilise l'argument suivant: on considère la parité d'un noeud, c'est à dire la parité du nombre de bit à 1 dans la représentation binaire du noeud. On peut voir facilement qu'un hypercube à* N *noeuds possède* $N/2$ *noeuds pairs et* $N/2$ *noeuds impairs. Si on suppose que l'hypercube contient un arbre de* $N-1$ *noeuds (*$Log(N)$ *niveau), alors si la racine est paire (sans perte de généralité), les fils de la racine sont impairs, les petits fils sont pairs etc. On peut donc compter les noeuds pairs:* $nb_P(N) = \sum_{i=0}^{\lfloor (LogN-1)/2 \rfloor} 2^{2i}$ *et les noeuds impairs* $nb_I(N) = \sum_{i=1}^{\lfloor (LogN)/2 \rfloor} 2^{2i}$. *Si* $LogN - 1$ *est multiple de 2, on compte les sommet pairs, sinon on compte les sommets impairs.*
   Par exemple, si $LogN-1$ *est pair (exemple,* $N = 32$, $LogN = 5$*) , essayons d'évaluer* $nb_P(N) = \sum_{i=0}^{(LogN-1)/2} 2^{2i}$. *On sait que*

$$
\begin{aligned}
N-1 = \; & \sum_{i=0}^{(LogN-1)} 2^i \\
& = (\sum_{i=0}^{(LogN-1)/2} 2^{2i}) + \sum_{i=0}^{(LogN-1)/2-1} 2^{2i+1} \\
& = (\sum_{i=0}^{(LogN-1)/2} 2^{2i}) + 2\sum_{i=0}^{(LogN-1)/2-1} 2^{2i} = \\
& 3(\sum_{i=0}^{(LogN-1)/2} 2^{2i}) - 2 * 2^{(LogN-1)} \\
& = 3nb_P(N) - N
\end{aligned}
$$

   ce qui implique $nb_P(N) = (2N-1)/3 > N/2$ *qui est une contradiction, on tombe sur une contradiction similaire avec* $nb_I(N)$ *dans le cas ou* $LogN - 1$ *impair. Dans le cas* $N = 32$, *on tombe sur* $nb_P(32) = 21$ *au lieu des 16 disponible dans l'hypercube.*

2. Montrer que l'on peut plonger dans un hypercube à N noeud le "arbre à double racine" de N noeud, c'est à dire un arbre dans lequel la racine à été dédoublée, chaque "demi-racine" n'ayant qu'un seul fils.
   Corrigé: *à faire*

3.2 Procédure SPMD pour le One-to-all sur un hypercube

1. On souhaite écrire une procédure qui implémente la communication one-to-all sur un hypercube. On suppose que l'on dispose de communications élémentaires `send` et `receive` qui prennent comme arguments un message et un numéro de processeur (qui doit être un des voisins du processeur exécutant l'instruction). On veut donc une procédure qui prend en argument la dimension de l'hypercube, le numéro du processeur qui l'exécute et le numéro du processeur qui initie le one-to-all (et le message à diffuser bien sur).

   Remarque: On pourra d'abord réfléchir à la procédure spécifique du one-to-all initié par le processeur 0, puis proposer une généralisation.

   Corrigé: *(source: [KGGK94], p 72) L'algorithme suivant est une procédure SPMD pour un one-to-all dont le message est originellement placé sur le processeur 0 d'un hypercude de dimension d. Ce code est exécuté en parallèle par chaque processeur, de manière assynchrone, la synchronisation est réalisée par les send et receive. Pour chaque processeur, my_id représente son numéro de noeud dans l'hypercube (compris entre 0 et* 2^d-1*). L'algorithme procède en* $d-1$ *étapes (indéxées de* $d-1$ *à* 0*). À l'étape d'index* i*, on effectue des communications uniquement sur les liens de la* i^{eme} *dimension (on a donc numéroté les dimensions de 0 à* $d-1$*). D'autre part, tous les processeurs ne peuvent pas communiquer le message à chaque étape (par exemple, à l'étape d'index* $d-1$*, c'est à dire à la première, seul le processeur 0 possède le message à diffuser). Le principe général est qu'à l'étape d'index* i*, tous les processeurs dont le numéro possède ses* i *premiers bits à 0 vont participer à la communication. Par exemple, à l'étape d'index* $d-1$ *(première étape), seuls les processeurs ayant leurs* $d-1$ *premiers bits a 0 vont participer à la communication, c'est a dire les processeurs de numéro 0* 2^{d-1}*. En effet, lors de cette première étape, le processeur 0 envoie son message au processeur* 2^{d-1}*, ce qui correspond bien au lien de dimension* $d-1$*.*

```
procedure ONE_TO_ALL_0(d,my_id,X) /* X est le message */
begin
    mask:= 2^d-1;                     /* mask contient tous ses bits à 1 */
    for i := d-1 downto 0 do
    begin                             /* les opérateurs logiques sont bits à bits */
        mask := XOR(mask,2^i);        /* mise à 0 du i^eme bit de mask */
        if AND(my_id,mask)==0 then
            if AND(my_id,2^i)==0 then
            begin                            /* je suis un envoyeur */
                destination:=XOR(my_id,2^i)   /* lien de dimension i utilisé */
                send X to destination
            endif
            else begin                       /* je suis un receveur */
                source:= XOR(my_id,2^i);
                receive X from source
            endelse
    endfor
```

18 **end** *ONE_TO_ALL_0*

3.3 One to All dans un arbre

On dispose d'une machine ayant la topologie suivante: un arbre binaire équilibré (toutes les feuilles sont à la même profondeur) dont les p feuilles sont des processeurs et dont tous les autres noeuds (noeuds internes à l'arbre) sont des noeuds d'interconnection. On suppose qu'un noeud ne peu router qu'un seul message à la fois et que un message de taille m prend le temps $t_s + t_w m + t_h l$ pour parcourir un chemin comportant $l - 1$ noeuds d'interconnection. Montrer que la diffusion simple (one to all) d'un message de taille m peut être exécutée en temps: $(t_s + t_w m + t_h(Log(p) + 1))Log(p)$, donner l'algorithme. *corrigé:*

On suppose que c'est le noeud le plus à gauche qui broadcast le message. L'algorithme est le suivant:

1. *Il envoi le message au noeud le plus à gauche du sous arbre de droite.*
2. *les deux noeud les plus a gauche des deux sous arbres recommence récursivement le one to all dans leur sous arbre, jusqu'à ce que le sous arbre soit réduit à une feuille.*

la première diffusion prend: $t_s + t_w m + t_h(2Log(p))$ *car il y a* $2Log(p) - 1$ *noeud d'interconnection à traverser. la première diffusion prend:* $t_s + t_w m + t_h(2(Log(p) - 1))$ *etc. La dernière prend*$t_s + t_w m + 2t_h$*. Il y a* $Log(p)$ *étapes. En sommant on obtient* $Log(p)(t_s + mt_w) + 2t_h \sum_{k=1}^{Log(p)} k = Log(p)(t_s + mt_w) + Log(p)(Log(p) + 1)t_h = Log(p)(t_s + t_w m + t_h(Log(p) + 1))$

Références

[KGGK94] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introction to Parallel Computing, Design and Analysis of Algorithms.* Benjamin-Cumming, 1994.

[Lei92] F. T. Leighton. *Introduction to parallel algorithm and architecture.* Cambrige University Press, 1992.

[QR89] Patrice Quinton and Yves Robert. *Systolic Algorithms and Architectures.* Prentice Hall and Masson, 1989.