TD AEV

FICHE 1

Exercice 1

Le programme suivant, composé de 6 instructions, doit être exécuté 64 fois pour évaluer l'expression arithmétique : $D(I) = A(I) + B(I) \times C(I)$ avec $I \in [0..63]$

```
Load R1, B(I) /R1 ← Mem (β + I )/
Load R2, C(I) /R2 ← Mem (χ + I)/
Mult R1, R2          /R1 ← (R1) x (R2)/
Load R3, A(I) /R3 ← Mem (α + I)/
Add R3, R1           /R3 ← (R3) + (R1)/
Store D(I), R3 /Mem (δ + I) ← (R3)/
```

R1, R2 et R3 sont des registres du CPU, (Ri) le contenu du registre Ri, α, β, χ, δ sont les adresses respectives de A, B, C, D. On suppose que le Load/Store prend 4 cycles, le Add 2 cycles et le Mult 8 cycles.

1) Calculez le nombre total de cycles pour exécuter le code sur un processeur SISD.
2) On considère une machine SIMD composée de 64 processeurs qui fonctionnent de manière synchrone. Calculez le temps d'exécution du programme sur cette machine.
3) quel est le facteur d'accélération de la machine SIMD par rapport à la SISD.
4) Même question avec $I \in [0..127]$
5) Même question avec $I \in [0..31]$
6) Même question avec $I \in [0..99]$

Exercice 2

Le programme Fortran suivant est exécuté sur un monoprocesseur. La version parallèle est exécutée sur un multiprocesseur à mémoire partagée.

```
L1:     Do 10 I = 1, 1024
L2:         SUM(I) = 0
L3:         Do 20 J = 1, I
L4: 20      SUM(I) = SUM(I) + J
L5: 10 Continue
```

On suppose que les instructions 2 et 4 prennent chacune 2 cycles machine, y compris CPU et accès mémoire. On ignore le coût causé par le contrôle de la boucle (instructions 1,3 et 5) ainsi que les surcoûts du système lui-même.

1) Quel est le temps d'exécution du programme sur le monoprocesseur?
2) On partage l'exécution de la boucle sur 32 processeurs de la façon suivante: le processeur 1 exécute les 32 premières itérations, le processeur 2 les itérations 33 à 64 etc...Quel est le temps d'exécution et le facteur d'accélération par rapport au monoprocesseur?
3) Comparez avec une distribution cyclique des itérations sur les 32 processeurs.
4) Proposer un algorithme qui assure un bon équilibrage de charge sur les 32 processeurs. (Même nombre d'addition sur chaque processeur)
5) Quel est le temps d'exécution de l'algorithme bien équilibré? Quel est maintenant le facteur d'accélération?

TD AEV

FICHE 2

On utilise une machine SIMD de 64 Processeurs Elémentaires (PE) à mémoire distribuée. Chaque processeur possède les registres A,B,C,D,I,R.
une unité de contrôle (ACU) pilote l'ensemble des PE. Et dispose de ses propres registres INX1, INX2, INX3, INX4, INX5. Les PE sont reliés par un réseau de communication.
On dispose des instructions suivantes

Instruction ACU :

MVI INXi ,#	move immédiat dans INXi
INC INXi	incrément de INXi : INXi <= INXi + 1
JLT INXi, INXj, Label	Branchement conditionnel si INXi<INXj goto Label
	LT less than GT greater than etc...

Instruction SIMD :

VLOAD r	Adressage indirect indexé : r <= ((Di) + (Ii))
(3cycles)	r ∈ {Ai,Bi,Ci,Ri}
VMOV r1, r2	Transfert registres r1 <= (r2)
(1cycle)	r1,r2 ∈ {Ai,Bi,Ci,Di,Ii,Ri}
VMVI r,#	move immédiat r <= #
(2cycles)	r ∈ {Ai,Bi,Ci,Di,Ii,Ri}
VSTORE r	Adressage indirect indexé : (Di) + (Ii) <= (r)
(3cycles)	r ∈ {Ai,Bi,Ci,Ri}
VADD r1, r2	Addition registres r1 <= (r1) + (r2)
(2cycles)	r1,r2 ∈ {Ai,Bi,Ci,Di,Ii,Ri}
VADDI r,#	Addition immédiat r <= (r) + #
(3cycles)	r ∈ {Ai,Bi,Ci,Di,Ii,Ri}
LCYCLE r	left cyclic Ri-s <= (Ri) avec s = 2^d et d = (r)
(3cycles)	

Soit trois vecteurs A, B, C rangé dans les mémoires locales de chaque PE.

1) Dessinez la machine SIMD avec les registres en place.

Donner les codes assembleur des boucles suivantes en ayant précisé le rangement initial de A, B et C.

2) For I = 0 to 63 A(I) = B(I) + C(I) dimension de A, B et C de 64 éléments
3) For I = 0 to 639 A(I) = B(I) + C(I) dimension de A, B et C de 640 éléments
4) Proposez une instruction supplémentaire qui permette de sélectionner certains des 64 PEs. Puis
   For I = 0 to 31 A(I) = B(I) + C(I) dimension de A, B et C de 64 éléments
5) For I = 0 to 630 A(I) = B(I) + C(I) dimension de A, B et C de 640 éléments
6) For I = 0 to 63 S = S + A(I) dimension de A de 64 éléments

Question ouverte….
For I = 0 to 63 For J = 0 to I A(I) = A(I) + B(J) dimension de A, B de 64 éléments

TD AEV

FICHE 3

Exercice 1

On désire calculer en parallèle la somme des quatre éléments du produit de deux matrices A, B 2x2. L'algorithme se compose donc de deux phases: le produit de matrice C = AxB puis la somme des quatre éléments de C. Pour réaliser ce programme il faut 8 multiplications et 7 additions.

1) Construisez le graphe du programme parallèle.
2) Sachant qu'une multiplication prend 101 cycles, une addition 8 cycles et une phase de communication entre processeurs 212 cycles, donnez les temps d'exécution en séquentiel et en parallèle sur 8 processeurs. On pourra à l'aide d'un schéma représenter dans le temps le déroulement du programme. Quel est le speedup obtenu?
3) Proposez une mise en œuvre de l'algorithme sur 4 processeurs. Quel est le temps de calcul, le speedup? Analysez les résultats de 2) et 3).
4) Sur 2 processeurs ?
5) Comment obtenir la somme sur tous les processeurs ?

Exercice2

Un monoprocesseur peut fonctionner en séquentiel ou parallèle. En mode parallèle, les calculs sont exécutés 9 fois plus vite qu'en séquentiel. Un programme de benchmark prend T cycle pour s'exécuter sur ce processeur. Une étude du code a montré que 25% du temps T est attribué au mode parallèle. Pour le reste, le processeur fonctionne en mode séquentiel.

1)Calculer le speedup effectif de ce processeur par rapport au même processeur sans mode parallèle. Donner le pourcentage de code qui a été parallélisé.
2)En supposant que l'on arrive à doubler le ratio des vitesses par des améliorations hardware, que devient le speedup effectif ?
3)En supposant que le même speedup doit être obtenu par une amélioration du compilateur au lieu du hardware. Quel doit être le pourcentage de code que le compilateur doit paralléliser pour obtenir ce speedup avec le même programme ?

TD AEV

FICHE 4

Exercice 1

1. Dessiner les réseaux de Benès à 4, 8 et 16 entrées.
2. Donnez tous les chemins possibles entre l'entrée 1 et la sortie 1 pour un benes 8x8
3. Donnez une configuration sur le 8x8 permettant les connexions simultanées (1,0), (2,2), (4,6) et (7,4)
4. Y-a-t-il une cinquième connexion qui demande un réarrangement ?

Exercice 2

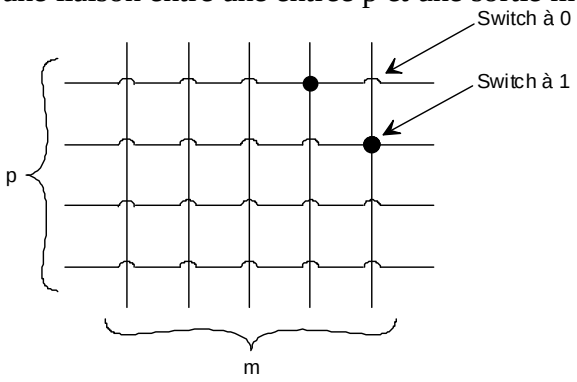
On dispose d'un commutateur 2 entrées, 2 sorties. Dans l'état de commande 0 (resp 1) les lignes sont reliées directement (resp en se croisant).

Etat 0 Etat 1

L'exercice consiste à réaliser un crossbar à partir de ce commutateur élémentaire.

1) Construisez un crossbar 2x3 avec 6 commutateurs, en utilisant les commutateurs comme connecteurs.

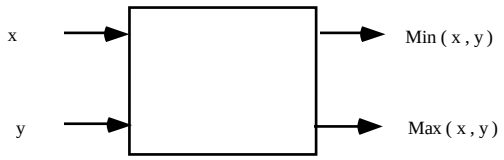
2) Construisez un crossbar N x P et précisez les commandes de chaque commutateur ij pour établir une liaison entre une entrée p et une sortie m



Exercice 3

Nous allons nous intéresser aux algorithmes de tri et de fusion parallèles: Algorithmes de Batcher.

La fusion de 2 listes triées X et Y permet d'obtenir une liste résultat triée Z telle que chaque élément z_i de Z appartienne à X ou Y et que chaque x_i et y_i apparaissent exactement une fois dans Z. On dispose d'un comparateur élémentaire 2 entrées et 2 sorties.



1) Proposer un réseau qui fusionne 2 listes triées de 2 éléments
2) Proposer un réseau qui fusionne 2 listes triées de 4 éléments
3) Proposer un réseau qui trie 1 liste non triée de 8 éléments

4) Proposer une méthode de construction itérative pour un réseau triant des listes de 2^n éléments. Quel est le nombre de comparateur total et le nombre de comparateur à traverser pour un des éléments de la liste Z.

TD AEV

FICHE 5

Exercice 1

L'ensemble de l'exercice concerne les communications entre processeurs, plus particulièrement la communication en échange total sur une machine de P processeurs: chaque processeur doit communiquer son propre message à tous les autres (chacun se retrouve avec P messages).

Nous considérons ici que tous les processeurs exécutent le même algorithme mode SPMD. On pourra définir l'algorithme en un seul processeur.

Partie 1

On considère des réseaux de communications statiques où les communications se font par envoi de messages (procédure envoyer et recevoir) sur des liens. Un seul message à un instant donné peut circuler dans chaque sens sur un lien bidirectionnel reliant deux processeurs. La procédure envoyer n'est pas bloquante alors que recevoir est bloquante. Cela veut dire que recevoir ne sera terminée que lors de la réception du message le premier arrivé sur le lien.

1) Proposer un algorithme d'échange total sur un réseau en anneau qui fait circuler les messages toujours dans le même sens .

On dispose des procédures envoyer à droite et recevoir de gauche. Combien de phases de communication sont nécessaires?

2) Proposer un algorithme d'échange total sur une grille 2D torique (P= p^2 processeurs). On dispose des procédures envoyer au Nord Sud Est et Ouest (idem pour recevoir). Combien de phases de communication sont nécessaires?

3) Proposer un algorithme d'échange total sur un hypercube de dimension 3, puis de dimension N. Combien de phases de communication sont nécessaires?

Partie 2

On considère un réseau de communication dynamique non bloquant. A chaque phase de communication un processeur peut établir une communication avec un autre processeur si les deux sont libres. On dispose d'un procédure de synchronisation qui en plus permet d'établir la permutation centralisée du réseau : permut ({i,j,k,…,l,m,n}) ou {i,j,k,…,l,m,n}) est une permutation de {1,2,3,…,N}, la même permutation doit être exécutée sur tous les processeurs pour être valide.

1) Proposez un algorithme SPMD d'échange total où chaque processeur envoie successivement son message aux autres processeurs. Donnez le nombre de phases de communication. Explicitez cette solution pour 8 processeurs.

2) Proposez une solution SPMD où chaque processeur envoie son message à un seul processeur. Celui-ci transmet ensuite le message vers un autre processeur etc... jusqu'à l'échange total. Donnez le nombre de phases de communication. Explicitez cette solution pour 8 processeurs.

TD AEV

FICHE 6

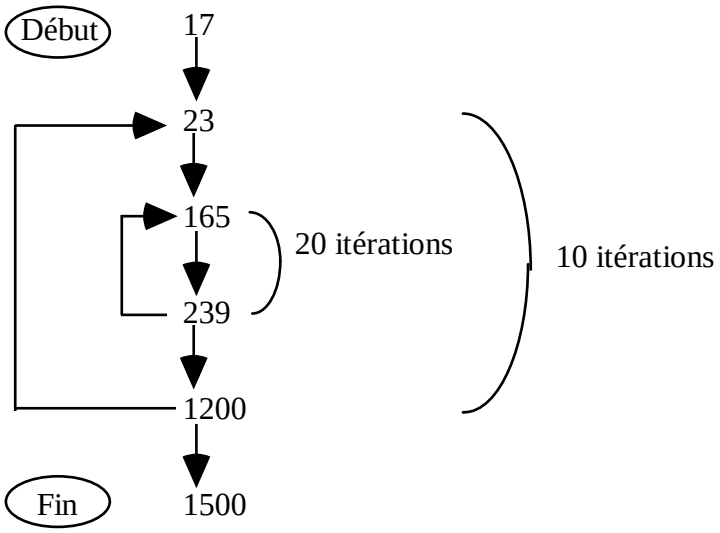
Exercice 1

On dispose d'une mémoire principale de 2^{16} octets. Le transfert minimal entre la mémoire et le cache est de 8 octets. On utilise un cache direct mapping de 32 lignes.

1) Définissez le nombre de champs de l'adresse et la taille de chaque champ.
2) Dans quelle ligne se trouvent les mots dont les adresses sont :
   0001000100011011
   1100001100110100
   1101000000011101
   1010101010101010
3) Supposez que l'octet dont l'adresse est 0001 1010 0001 1010 soit rangé dans le cache. Quelles sont les autres adresses rangées dans la même ligne ?
4) Que range-t-on avec la donnée et pourquoi ?
5) Quelle est la taille effective du cache?

Exercice 2 Cache d'instruction

Un programme se compose de deux boucles FOR imbriquées, une petite boucle interne et une plus grande externe. La structure générale du programme est la suivante:



Les adresses mémoires décimales données déterminent l'emplacement des deux boucles ainsi que le début et fin du programme. Tous les emplacements mémoire contiennent des instructions devant être exécutées séquentiellement. Le programme s'exécute sur un processeur avec cache mémoire. Le cache est organisé en direct mapping. La mémoire est de 64K mots, le cache est de 1K mots , il est organisé en bloc de 128 mots.

a)Précisez les différents champs d'une adresse ainsi que leur taille.

b)En ne tenant compte que de l'accès mémoire pour le chargement d'une instruction, donnez le temps de chargement moyen d'une instruction de ce programme. On pose M le temps d'accès à la mémoire d'un bloc et m le temps d'accès au cache d'une instruction

Exercice 3. Cache données

Soit le programme suivant, qui effectue la normalisation des colonnes d'une matrice X[8][8] : chaque élément de la colonne est divisé par la moyenne des valeurs de cette colonne.

```
float X[8][8], sum, ave;
sum = 0.0;
for (j = 0; j < 8; j++) {
```

```
     for (i=0; i<8, i++)
          sum+= X[i][j];
     ave = sum/8;
     for (k=7; k>=0; k--)
     X[k][j] = X[k][j]  / ave
}
```

On suppose que l'on a un cache de 128 octets avec des blocs de 16 octets (soit 8 blocs pour le cache). L'adresse de X[0][0] est F000H (sur 16 bits), la matrice est rangée ligne par ligne, puis colonne par colonne. On répondra aux questions pour ces deux rangements.

Direct mapping

Définir dans quels blocs du cache va chaque élément de la matrice.

En déduire le nombre de défauts de cache pour l'exécution du programme ?

Quel serait le nombre de défauts de cache en écrivant la seconde boucle interne

```
     for (k=0; k<8, k++)
```

TD AEV

FICHE 7

Exercice 1

Un processeur adresse une mémoire de 2^{32}mots. On désire utiliser une mémoire cache de 16 Kmots de capacité. Les mots sont de 32 bits. Pour chaque construction donnez le nombre de champs de l'adresse et la taille de chacun, puis calculez la taille totale du cache en bits (TAG + data + ...)

a) En direct mapping
b) En direct mapping par bloc de 16 mots
c) En Full associative
d) En set-associative par ensemble de 16 mots
e) En set-associative par ensemble de 16 blocs de 8 mots

Exercice 1

Lors d'une exécution d'un programme, on observe la trace des pages suivantes : P = r1, r2, r3, ... rk, rk+1, ... où rk est le numéro de la page qui contient la kième référence du programme.

Exemple

trace	a	b	c	b
défaut	*	*	*	
cache	a	a	c	c
	#	b	b	b

Pour la trace suivante P = abacabdbacd

1)Donner le contenu de la mémoire cache, la présence de défaut de cache pour chaque référence. La mémoire cache contient 2 pages.

a) Pour un algorithme de remplacement FIFO
b) Pour un algorithme de remplacement LRU
c) Est-on loin de l'optimal ?

2) Pour une mémoire cache contenant 3 pages, même question.

3)LRU et FIFO semblent de " bons " algorithmes. L'algorithme MRU Most Recently Used semble intuitivement mauvais. Comparer avec les résultats précédents. Peut-on juger de l'efficacité d'un algorithme sur une seule trace de programme ?

Exercice 3

Ce problème consiste à distribuer une structure de données sur une machine pipe-line vectorielle à mémoire entrelacée. Soit Mat une matrice 8x8.

La mémoire entrelacée est composée de huit bancs indépendants (accès en parallèle). Chaque banc a un temps de lecture de d cycles processeur.

1) Proposer un rangement de la matrice en mémoire permettant un accès efficace par ligne puis par colonne. Donner les temps d'accès à une ligne puis une colonne pour les deux rangements proposés.
2) Proposer un rangement unique permettant le même temps d'accès (le meilleur) pour une ligne et une colonne. Quel est le temps d'accès à la diagonale?
3) Que se passe-t-il si l'on prend 9 bancs au lieu de 8? Préciser les avantages et inconvénients.

TD AEV

FICHE 8

Exercice1

Un micro processeur pipe-line possède 5 étages : (IF) Instruction Fetch, (ID) Instruction Decode,(OF) Operand Fetch , (OE) Operation Execute, (OS) Operand Store. Le tableau suivant représente les unités utilisées pour certaines instructions.

Instruction	T(IF)	T(ID)	T(OF)	T(OE)	T(OS)
Load mem/reg	1	1	1	0	0
Load reg/reg	1	1	0	1	0
Store reg/mem	1	1	0	0	1
Add reg/reg	1	1	0	1	0

Calculer le temps d'exécution des programmes suivants.

1)

```
MOV  AX ,[100]      ;copie mémoire 100 sur registre AX
MOV  BX ,[200]
MOV  CX ,[300]
MOV  DX ,[400]
```

2)

```
MOV  AX ,[100]
MOV  BX ,[200]
ADD  AX , BX
MOV  [200] ,AX      ;copie AX vers mémoire 200
MOV  BX, [200]
```

Exercice 2

A partir d'un programme en assembleur, on veut exploiter le maximum de parallélisme sur un puis deux processeurs RISC. On suppose qu'il n'y a pas de conflits d'accès aux ressources communes et que le processeur est multi-unités fonctionnelles. Par simplification le processeur n'est pas pipeline et toutes les instructions s'exécutent en 1 cycle machine.

Le programme est le suivant:

```
1    Load  R1 , A        /R1 <- Mem(A)/
2    Load  R2, B
3    Mul   R3, R1, R2    /R3 <- (R1) * (R2)/
4    Load  R4, D
5    Mul   R5, R1, R4
6    Add   R6, R3, R5
7    Store X, R6         /Mem(X) <- (R6)/
8    Load  R7, C
9    Mul   R8, R7, R4
10   Load  R9, E
11   Add   R10, R8, R9
12   Store Y, R10
13   Add   R11, R6, R10
14   Store U, R11
15   Sub   R12, R6, R10
16   Store V, R12
```

TD AEV

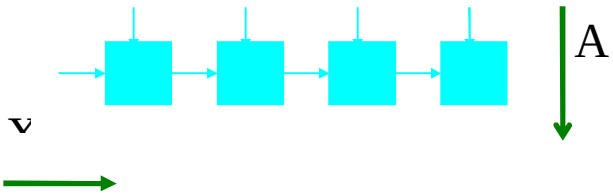
1) Dessinez un graphe qui représente les dépendances de données entre les 16 instructions (les arcs du graphe représentent la réutilisation par une instruction d'un résultat produit par une autre instruction).
2) On considère que le processeur est 3-issues, il peut donc à chaque cycle exécuter 3 instructions: une instruction d'accès mémoire, une instruction Add/Sub et une instruction Mul. Construisez la liste des triplets d'instructions qui seront exécutées à chaque cycle sur ce processeur pour ce programme.
3) On utilise maintenant 2 processeurs de ce même type avec une mémoire partagée. Proposez un partitionnement du programme en deux parties équilibrées, U et V seront produits sur des processeurs différents. Vous pourrez insérer des instructions load/store pour assurer les communications entre les deux processeurs. Redessinez les deux graphes correspondants (avec 2 couleurs différentes svp).Précisez le déroulement de la liste des triplets d'instructions sur chaque processeur. Quel est le gain obtenu par cette parallélisation?

TD AEV

FICHE 9

Ce problème concerne le produit de matrice/vecteur puis matrice/matrice

1) On dispose d'une ligne de processeurs. Chacun dispose de sa propre mémoire locale, de son propre programme, de liens de communications avec deux voisins. Chaque processeur possède deux liens en entrée. Ils sont connectés de la façon suivante.



On veut réaliser le produit $Yi = \Sigma_{j=1,n}$ Aij Xj. Les éléments du vecteur X sont envoyés successivement en entrée du premier processeur. Les éléments de la première ligne sont envoyés successivement sur le premier processeur, la seconde ligne sur le second etc. Tous les processeurs effectuent le même algorithme:

recevoir des données de la gauche, du haut
calculer
envoyer des données vers la droite

Le résultat Y du produit mat/vect sera réparti sur les différents processeurs. L'idée générale de l'algorithme consiste à se faire rencontrer le couple(Xj, Aij) en même temps sur le proc i.

a) Définissez le flux d'entrée de chaque processeur (les Aij) en tenant compte des décalages dans le temps. Pour un produit de dimension 4, dessinez les flux de données entre les différentes étapes de calcul. Ex  a b . c . signifie envoyer rien puis c puis rien puis b puis a.

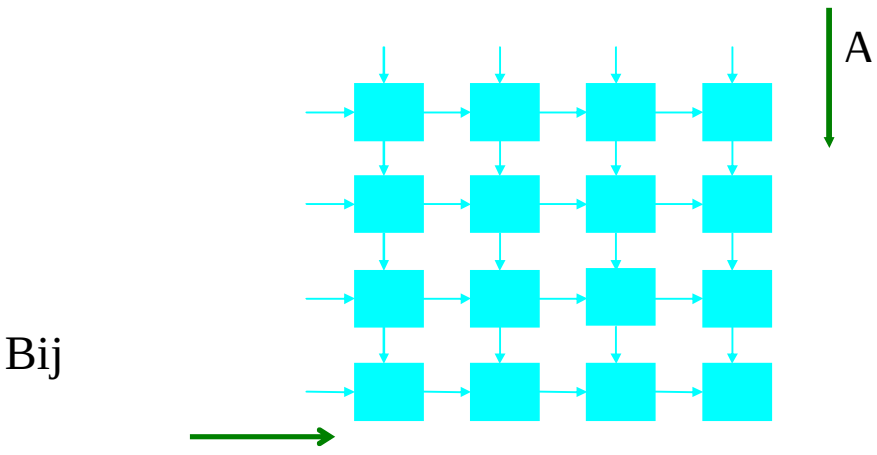
b) Ecrire l'algorithme pour un processeur. Quel est le temps d'exécution total?

2) On veut maintenant calculer un produit matrice/matrice. On dispose pour cela d'une grille 2 dimensions.

Les colonnes de B sont envoyées depuis la gauche, les lignes de A sont envoyées depuis le haut. Là encore il s'agit de réunir les couples (Aik, Bkj) sur le processeur Pij.

a) Précisez les flux d'entrée en haut et à gauche de la grille de processeurs en tenant compte des décalages.

b) Ecrire l'algorithme d'un processeur. Donnez le temps total de calcul du produit de matrices.



TD AEV

3) A partir du même algorithme, on veut définir un algorithme de produit de matrices sur la MasPar. On utilise le réseau de voisinage Nord, Sud, Est, Ouest qui est torique. Les deux matrices A et B sont ici rangées sur les processeurs.

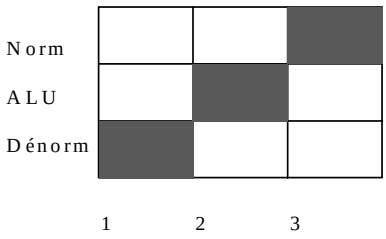
a) Donnez pour une MasPar de 4x4 PE la répartition initiale de Aij et Bij sur les processeurs Pi'j' pour un produit de matrices 4x4.(sur un dessin)

b) Pour une topologie de N x N processeurs et une multiplication de deux matrices N x N calculez le nombre de phases de communication et de phases de calcul.

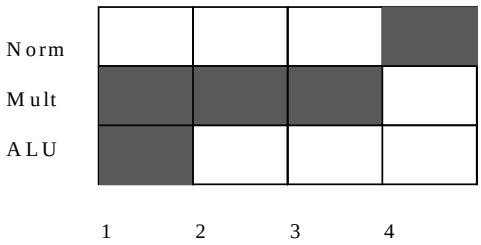
TD AEV

FICHE 10

La mise en œuvre d'un calcul sur un pipe-line demande plusieurs ressources successives, chacune correspondant à un étage du pipe-line. Plusieurs étages d'un même pipe-line peuvent demander la même ressource et donc entraîner des **collisions** lors de l'exécution.
Pour éviter ces collisions, on utilise les tables de réservation. Elles permettent de représenter l'état d'occupation des ressources matérielles en fonction des périodes d'horloge. Par exemple pour un additionneur flottant, le premier étage correspond au décalage des mantisses et à la comparaison des exposants (*Dénorm*), le second est l'addition des mantisses décalées(*ALU*), le troisième la normalisation des résultats(*Norm*). La table de réservation comporte trois colonnes.



Pour la multiplication, il faut additionner les exposants, multiplier les mantisses ce qui prend 3 cycles (en même temps que l'addition) puis normaliser le résultat.



A l'évidence, on peut déclencher une addition à chaque cycle, par contre on ne peut déclencher une multiplication que tous les trois périodes d'horloge.

Question 1 :

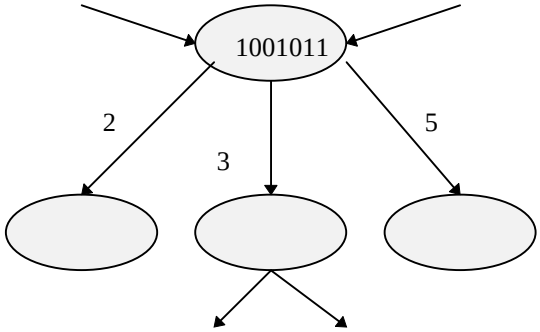
On désire réaliser un pipe-line qui effectue une multiplication et une addition en série(quand la multiplication est terminée) de type `a * x + y`. Donnez la table de réservation. Combien de cycles d'attente sont nécessaires entre deux déclenchements (on appelle ce nombre la latence du pipe-line)?
Proposez une autre construction de pipe-line où l'on anticipe le calcul de l'addition (en parallèle avec la multiplication). Que deviennent la table de réservation et la latence ? Comment utiliser ce pipe-line pour calculer `a * X + Y` ou `X` et `Y` sont des vecteurs de n éléments ?

Pour automatiser le déclenchement de nouveaux calculs dans le pipe-line, on utilise le vecteur de collision. Le vecteur de collision associé à un pipe-line est un vecteur binaire dont la $i^{ième}$ composante indique s'il y a conflit en cas de lancement après i cycles. Le bit correspondant vaut 1 en cas de conflit et 0 sinon. Dans le cas de la multiplication, le vecteur de collision est 110 (C_1,C_2,C_3), il faut 3 cycles pour éviter les collisions sur mult.

Question 2 :

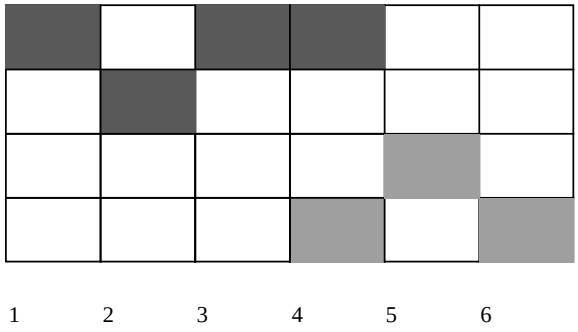
On dispose d'un registre d'état à décalage. Celui-ci doit permettre à chaque top de connaître directement s'il est possible ou non de déclencher un nouveau calcul. Précisez la valeur d'initialisation de ce vecteur d'état lors du lancement du premier calcul, la transformation de ce vecteur à chaque top d'horloge, et la mise à jour de ce vecteur lors du lancement d'un nouveau calcul. Donnez sous forme d'algorithme, le fonctionnement de notre déclencheur de calcul. Décrivez le début de l'algorithme avec le vecteur de collision 1001011 en essayant de lancer un calcul dès que possible.

La stratégie que nous avons pour l'instant adoptée est dite gloutonne, c'est à dire qu'une requête est exécutée dès que possible. Ce n'est pas toujours la meilleure. A partir de l'exemple précédent, vecteur de collision 1001011, on peut construire le diagramme de tous les états possibles si l'on commence une nouvelle opération à tous les instants possibles. Pour notre exemple, on peut lancer un nouveau calcul soit à l'instant 2, soit à l'instant 3, soit à l'instant 5. Puis pour chaque vecteur d'état obtenu, on peut réessayer tous les déclenchements possibles. On obtient ainsi un graphe où les sommets sont les vecteurs d'état et les arcs les transitions étiquetées par les délais.



Question 3 :
Dessinez le graphe complet pour le vecteur de collision 1001011. Identifiez tous les cycles de ce graphe et montrez que l'un d'entre eux permet d'obtenir le meilleur rendement (nombre de calcul / nombre de top d'horloge). Précisez la technique de calcul du meilleur cycle pour un graphe quelconque.

Dans certain cas, lorsque les dépendances entres les étages du pipe-line le permettent, la restructuration de la table de réservation permet d'améliorer le débit du pipe-line. Considérons la table de réservation suivante.



Question 4 :
Dessinez le graphe des états et donnez le débit maximum de ce pipe-line. Si l'on suppose qu'il n'y a pas de dépendance entre les gris clair et les gris foncé,

proposez une transformation de la table de réservation qui permette d'augmenter le débit maximum. Vérifiez par le nouveau graphe obtenu, la valeur du débit maximum.

Les résultats de cet exercice sont publiés dans le livre de M. Cosnard et D. Trystram " Algorithmes et architectures parallèles " chapitre VI.2 InterEditions.

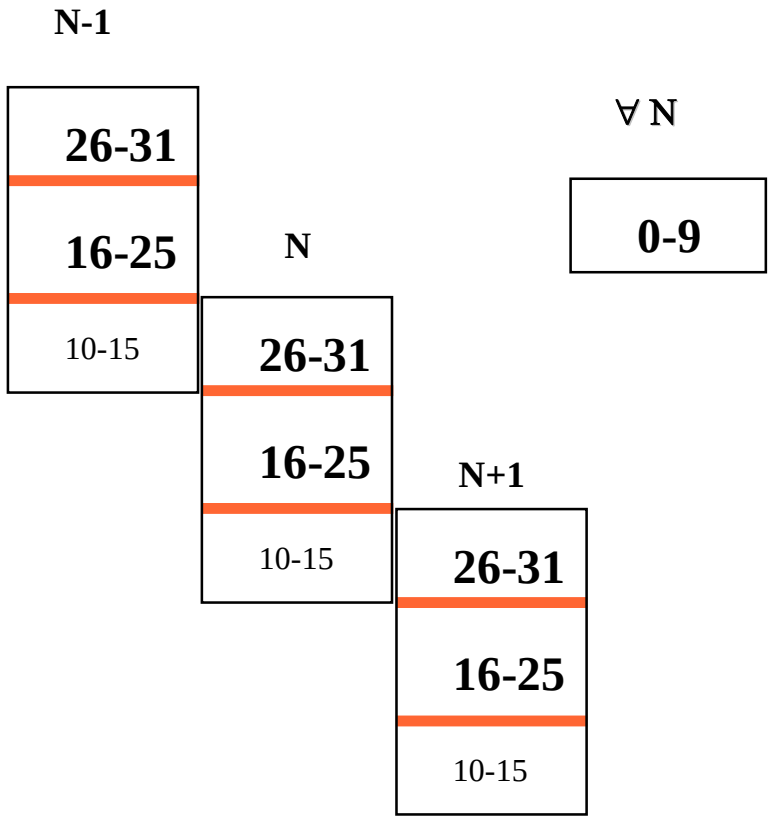
TD AEV

FICHE 11

Exercice 1 : (6pts)

Voici une copie d'un des transparents du cours concernant la gestion des registres sur un processeur RISC. Répondre aux questions suivantes :

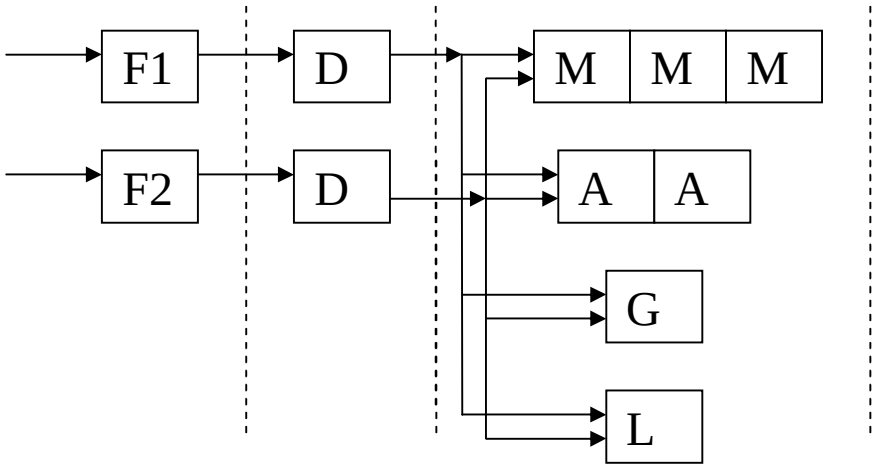
1)Comment passe-t-on du niveau N à N+1 et de N à N-1 ?
2) A quoi servent les 10 registres numérotés 0-9 ?
3) A quoi servent les registres 26-31, puis 16-25 et 10-15 ?
4) Sur un processeur à 192 registres, à quel moment se produit un dépassement de capacité ? Comment peut-on alors fonctionner pour le passage de N à N+1 ?
5) Pour un niveau N donné, dans quel cas se produit un dépassement de capacité ? Comment peut-on le gérer ?
6) Quel est l'avantage majeur dans cette gestion de registres ?



Exercice 2 (8 points)

Soit le processeur super-scalaire suivant :

TD AEV



Ce processeur est composé de deux pipe-lines de 3 étages : Fetch (F1, F2 1 cycle), Decode (D1, D2 1 cycle), Execute (multiplieur de 3 cycles M1,M2,M3, Addition de 2 cycles A1, A2, unité logique G et Load/Store LS de 1 cycle), ces unités fonctionnelles ne sont pas pipe-linées elles-même. Chaque pipe-line possède sa propre unité de fetch et de decode, par contre les 4 unités fonctionnelles sont partagées dynamiquement entre les deux pipe-lines si elles sont libres. Les registres Destination sont mis à jour à la fin de l'étage d'exécution (attention aux dépendances !!). Le Fetch et le Decode sont toujours exécutés dans l'ordre d'écriture du programme. Ces deux étages sont synchronisés pour les deux pipelines et avancent nécessairement en même temps.

On utilise le programme suivant : R1, R2 etc sont des registres du processeur, A, B etc sont des adresses mémoire. (Format à 2 adresses : OP SRC/DEST, SRC)

```
1) Load R1, A
2) Add R2, R1
3) Add R3, R4
4) Mul R4, R5
5) Comp R6          /complément de R6 sur unité logique/
6) Mul R6, R7
```

1) Quelles sont les dépendances dans ce programme?
2) On applique un déclenchement de l'exécution dans l'ordre (ne pas déclencher i avant i+1). En respectant les dépendances et l'accès aux ressources du processeur, décrire le déroulement du programme top après top. On pourra utiliser une table telle que celle présentée en cours, en voici les deux premières lignes…

F1	F2	D1	D2	M	A	LS	G
1	2						
3	4	1	2				

3) Même question mais en autorisant une exécution dans le désordre.
4) On voudrait en plus autoriser un déclenchement des instructions dans le désordre, que faut-il ajouter au processeur ? Décrire le déroulement dans ce cas.
5) Pour les trois solutions proposées, peut-on réorganiser le code à la compilation de façon à gagner des cycles à l'exécution ?

TD AEV

FICHE 12
Question 1: 3pts
Donner la liste de toutes les dépendances dans les morceaux de code suivants:

```
a)
sub r1,r3,r2            | r2 <- r1 - r3
and r2,r5,r12           | r12 <- r2 and r5
or r6,r2,r13            | r13 <- r6 or r2
add r2,r2,r14           | r14 <- r2 + r2
move r2,r15             | r15 <- r2
b)
sub r1,r3,r2            | r2 <- r1 - r3
and r2,r5,r4            | r4 <- r2 and r5
or r2,r4,r4             | r4 <- r2 or r4
add r4,r2,r9            | r9 <- r4 + r2
c)
add r4,r5,r2            | r2 <- r4 + r5
add r2,r5,r4            | r4 <- r2 + r5
load 100(r2),r5         | r5 <- M[r2+100]
add r4,r2,r3            | r3 <- r4 + r2
```

Exercice 1
On dispose d'une machine vectorielle disposant d'une mémoire entrelacée à accès pipeliné. Sur chaque unité fonctionnelle de type load : on envoie une adresse à la fois vers la mémoire et on récupère pendant le même cycle une autre donnée de la mémoire que l'on range dans le registre vectoriel destination.
Pour les unités fonctionnelles de type store : on envoie un couple (adresse, donnée) à la fois vers la mémoire à partir d'une donnée extraite du registre vectoriel source.
Le temps d'accès à la mémoire est de 4 cycles pour les lectures et écritures. On considère que la mémoire ne produit jamais de conflit d'accès (pas d'accès à un banc mémoire déjà occupé).
Les instructions manipulent des vecteurs de 4 éléments.

Le processeur peut déclencher une nouvelle instruction à chaque cycle en respectant l'ordre du programme, les dépendances entre les instructions et la disponibilité des unités fonctionnelles. Les pipelines Add et Mul sont de 4 étages, ils sont connectés à tous les registres vectoriels sans avoir à se partager le chemin de données entre registres et pipelines. Ils produisent un résultat par cycle.

Format des instructions et programme test :

(Opération, source 1, [source 2,] destination),
A,B, C et D sont des vecteurs contenant les 4 adresses mémoires,
V1, V2, V3, V4, V5, V6, V7 sont des registres vectoriels de 4 éléments.

```
t0 :    Vload  A, V1
t1 :    Vload  B, V2
t2 :    Vmul   V1, 3, V3
t3 :    Vadd   V2, 4, V4
t4 :    Vmul   V3, V4, V2
t5 :    Vload  C, V1
t6 :    Vadd   V2, V1, V3
t7 :    Vstore V3, D
```

1) Définir toutes les dépendances entre les instructions.

2) Renommer les registres en utilisant V5,V6 etc afin de ne garder que les dépendances de flot.

La figure représente un exemple d'architecture avec 2 unités fonctionnelles Load/store, une unité pipelinée de Add et une de Mul.