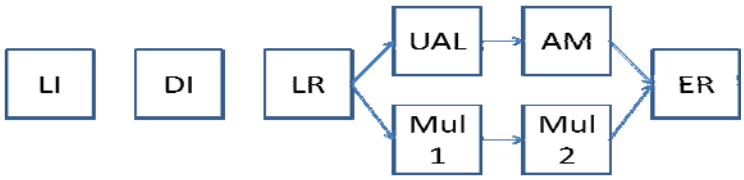
D. Etiemble, J. Falcou

ARCHITECTURES PARALLELES

Corrigé Examen Mai 2014
2 H – Tous documents autorisés
Les questions sont indépendantes

1. Pipeline

La figure ci-dessous donne le pipeline d'un processeur (on suppose qu'il utilise le jeu d'instructions MIPS32)



La signification des étapes du pipeline sont

- LI : lecture de l'instruction
- DI : décodage de l'instruction
- LR : lecture des registres
- UAL : exécution UAL ou calcul adresse
- Mul 1 et 2 : étapes de la multiplication
- AM accès cache de données
- ER : écriture du résultat.

Q 0) Donner les latences des instructions suivantes, en supposant que tous les court-circuits nécessaires sont implantés.

	Instruction	Producteur	Consommateur
a)	ADD	ADD **R1**,R2,R3	SUB R5,R6,**R1**
b)	ADD	ADD **R1**,R2,R3	SW **R1**,4(R4)
c)	ADD	ADD **R1**,R2,R3	SW R4,[4(R1] !
d)	ADD	ADD **R1**,R2,R3	MUL R5,R6,**R1**
e)	LW	LW **R1,** [R2], #4	SUB R5,R6,**R1**
f)	MUL	MUL **R1**,R2,R3	SUB R5,R6,**R1**

ADD **R1**,R2,R3							SUB R5,R6,**R1**
LI	DI	LR	**EX**	AM	ER		
	LI	DI	LR	**EX**	AM	ER	

1 cycle

ADD **R1**,R2,R3							STR **R1**,[R4 +4] !
LI	DI	LR	**EX**	AM	ER		
	LI	DI	LR	EX	**AM**	ER	

1 cycle

ADD **R1**,R2,R3	STR R4,[**R1** +4] !

LI DI LR **EX** AM ER
LI DI LR **EX** AM ER
1 cycle

ADD **R1**,R2,R3	MUL R5,R6,**R1**

LI DI LR **EX** AM ER
LI DI LR **M1** M2 ER
1 cycle

LDR	LDR **R1,** [R2], #4	SUB R5,R6,**R1**

LI DI LR EX **AM** ER
LI DI LR **EX** AM ER
2 cycles

MUL **R1**,R2,R3	SUB R5,R6,**R1**

LI DI LR **M1** M2 ER
LI DI LR **EX** AM ER
2 cycles

2. Optimisation de programmes

On utilise un processeur scalaire dont les instructions sont définies dans l'annexe 1 (Table 2 et Table 3). Les latences des instructions sont données dans la troisième colonne des tables. On rappelle qu'une latence de n signifie que si une instruction I démarre au cycle n, une instruction qui utilise son résultat ne peut démarrer qu'au cycle c+n (une latence de 1 signifie qu'elle peut démarrer au cycle suivant). Les sauts et branchements ne sont pas retardés et l'on suppose une prédiction de branchement parfaite (l'instruction BNE a une latence de 1 cycle).

La table 1 présente un programme C et le programme assembleur correspondant. Les tableaux X et Y sont rangés successivement à partir de l'adresse 0x1000 0000, qui est contenue dans le registre R3 au démarrage du programme. S est rangé à l'adresse 0x1000 ;

```
float X[100], Y[100], Z[100], A , B, S;
int i;
S=0.0;
for (i=0; i<100; i++)
S+= A*X[i] + B*Y[i];
```

```
        ADDI R5,R3,400
        LF F1, A          //A est chargé dans F1
        LF F2,B           //B est chargé dans F2
        FSUB  F0,F0,F0
Loop :  LF F3,0(R3)
        FMUL F3,F3,F1
        LF F4, 400(R3)
        FMUL F4,F4,F2
        FADD F3,F3,F4
        FADD F0,F0,F3
        ADDI R3,R3,4
        BNE R3,R5,Loop
        SF F0, 0x1000(R0)
```

Table 1 : Programme C et programme assembleur.

Q 1) Donner l'exécution cycle par cycle de la boucle optimisée (mais sans déroulage de boucle). Quel est le nombre de cycles par itération de la boucle ? Quel est le temps total d'exécution du programme ?

```
P1      ADDI R5,R3,400
P2      LF F1, A        //A est chargé dans F1
P3      LF F2,B         //B est chargé dans F2
P4      FSUB  F0,F0,F0
1 Loop :LF F3,0(R3)
2       LF F4, 400(R3)
3       FMUL F3,F3,F1
4       FMUL F4,F4,F2
5       ADDI R3,R3,4
6
7
8       FADD F3,F3,F4
9
10
11
12      FADD F0,F0,F3
13      BNE R3,R5,Loop
F1
F2
F3      SF F0, 0x1000(R0)
```

13 cycles par iterations.
Temps total : 4 + 1300 + 3 = 1307 cycles

```
P1      ADDI R5,R3,400
P2      LF F1, A        //A est chargé dans F1
P3      LF F2,B         //B est chargé dans F2
P4      FSUB  F0,F0,F0
P5      FSUB F5,F5,F5
1 Loop :LF F3,0(R3)
2       LF F4, 400(R3)
3       FMUL F3,F3,F1
4       FMUL F4,F4,F2
5       ADDI R3,R3,4
6
7       FADD F0,F0,F3
8       FADD F5,F5,F4
9       BNE R3,R5,Loop
F1      FADD F0,F0,F5
F2
F3
F4
F5      SF F0, 0x1000(R0)
```

9 cycles par itération. Temps d'exécution total : 5 + 900 + 5 = 910 cycles.

Q 2) Avec un déroulage de boucle d'ordre 4, quel est le nombre de cycles par itération de la boucle initiale ? Quel est alors le temps total d'exécution du programme ? (le code de la boucle déroulée n'est pas demandé)

Par rapport à la boucle optimisée version 1, il y a
8 LF, 8 FMUL, 8 FADD, 1ADDI et 1 BNE, sans suspension soit un total de 26 cycles/4 itération = 6,5 cycles/itération.
Le prologue a 1 ADDi, 2 LF et 4 FSUB soit 7 cycles.
L'épilogue est le suivant (10 cycles)

FADD F0,F0,F11
FADD F12F12,F13
-
-
-
FADD F0,F0,F12
-
-
-
SF F0, 0x1000(R0)

Le temps d'exécution total est 7 + 650+10 = 667 cycles

3. Caches

Un processeur utilise un cache de données de 32 Ko, avec des lignes de 16 octets, à correspondance directe. Le cache utilise la réécriture avec écriture allouée (il y a des défauts de cache en écriture). Le processeur a des adresses sur 32 bits.

On considère l'extrait de programme C suivant, pour lequel les tableaux X et Y sont rangés successivement en mémoire à partir de l'adresse $1000\ 0000_H$ (adresse de X[0].)

```
float X[2052], Y[2048];
for (i=1 ; i<2048 ; i++)
      Y[i] = X[i+4] – X[i] ;
```

Q 3) Quel est le nombre de bits pour l'adresse dans la ligne, l'index et l'étiquette ?

Il y a 2^{11} lignes de 16 octets soit

- 4 bits d'adresse dans la ligne
- 11 bits d'index
- 17 bits d'étiquette

Q 4) Quelles sont les adresses hexadécimales de Y[0] et X[0] ? Dans quelles lignes du cache vont les données correspondant à X[0] et Y[0] ?
Adresse de X[0] 0x1000 0000
Adresse de Y[0) 0x1000 2000 = 0001 0000 0000 0000 0|010 0000 0000| 0000
X[0]va dans la ligne 0.
Y[0] va dans la ligne 512
Q 5) Quel est le nombre de défauts de cache par itération de la boucle et le nombre total de défauts de cache.
I= 1 : deux défauts de caches en lecture et un défaut de cache en écriture.

I=2,3 = succès
Puis un défaut de cache en lecture et 1 en écriture toutes les 4 itérations.
Au total, il y a 2048/4 + 1 défauts soit 513 défauts en lecture et 2048/4 en écriture.
Total : 1025

4. SIMD

Soit deux matrices A et B contenant 2 x 2 réels en double précision. On se propose de vectoriser le produit matriciel de A par B en utilisant les instructions SIMD SSE2. On considère que les données de A et B sont stockées de la façon suivante :

```
double A[2][2], B[2][2], C[2][2];
```

Q 6) Donner le code C entièrement déroulé qui remplit la matrice C avec le résultat du produit matriciel de A par B.

```
C[0][0] = A[0][0]B[0][0] +A[0][1]B[1][0];
C[0][1]= A[0][0]B[0][1]+A[0][1]B[1][1];
C[1][0] = A[1][0]B[0][0]+A[1][1]B[1][0];
C[1][1]=A[1][0]B[0][1]+A[1][1]B[1][1];
```

Q 7) Proposez une implantation SSE2 de ce calcul en utilisant les fonctions suivantes :

```
#define  lpdu (p) __m128d _mm_loadu_pd(double *p)          //renvoit (p[0], p[1])
#define lpd1(p)   __m128d _mm_load1_pd(double *p)         // renvoit (p[0], p[0])
#define addpd(a,b)          __m128d _mm_add_pd( __m128d a, __m128d b )    //renvoit (a0+b0, a1+b1)
#define mulpd(a,b)          __m128d _mm_mul_pd( __m128d a, __m128d b )    //renvoit (a0*b0, a1*b1)
#define spud (p,a)          void _mm_storeu_pd( double *p, __m128d a )    //effectue p[0]=a0, p[1]=a1
```

```
C[0][0]|C[0][1] = (A[0][0]|A[0][0] * B[0][0]|B[0][1])+ (A[0][1]|A[0][1] * B[1][0]|B[1][1]) ;
C[1][0]|C[1][1] = (A[1][0]|A[1][0] * B[0][0]|B[0][1])+ (A[1][1]|A[1][1] * B[1][0]|B[1][1]) ;

Spud(&C[0][0], addpd (mulpd (lpd1(&A[0][0]), lpdu&(B[0][0])), (mulpd (lpd1(&A[0][1]), lpdu(&B[1][0]))));
Spud(&C[1][0],  addpd (mulpd (lpd1&(A[1][0]), lpdu(&B[0][0])), (mulpd (lpd1(&A[1][1]), lpdu(&B[1][0]))));
```

En décomposant avec des variables 128 bits X00, X01,X10,X 11 et Y00,Y10

```
X00= lpd1(&A[0][0] ; X01  = lpd1(&A[0][1] ; X10  = lpd1(&A[1][0]; X11  = lpd1(&A[1][1];
Y00= lpdu(&B[0][0] ; Y10  = lpdu(&B[1][0];

X00 = mulpd (X00,Y00) ; X01 = mulpd(X01,Y10) ;
X00 = addpd(X00,X01) ;
Spdu (&C[0,0], X00) ;

X10 = mulpd (X10,Y00) ; X11 = mulpd(X11,Y10) ;
X10 = addpd(X10,X11) ;
Spdu &(C[1,0], X10) ;
```

Q 8) Si les matrices A et B contenaient des réels simples précisions, quelle devrait être leur taille afin de pouvoir de la même façon calculer leur produit matriciel déroulé entièrement.

Taille [4][4]

5. Annexe 1

JEU D'INSTRUCTIONS (extrait)

Mnémo	Syntaxe	Latence	Effet
ADDI	ADDI Rd, Ra, IMM	1	Rd ←Ra + IMM-16 bits avec ES
BEQ	BEQ Ri, Rj, dépl	1	si Ri=Rj alors CP ← NCP + depl
BGE	BNE Ri, Rj, dépl	1	si Ri≠Rj alors CP ← NCP + depl

Table 2 : instructions entières disponibles

Mnémo	Syntaxe	Latence	Effet
LF	LF Fi, dép.(Ra)	2	Fi ← M (Ra + dépl.16 bits avec ES)
SF	SF Fi, dép.(Ra)	1	Fi -> M (Ra + dépl.16 bits avec ES)
FADD	FADD Fd, Fa, Fb	4	Fd ←Fa + Fb
FMUL	FMUL Fd, Fa, Fb	4	Fd ←Fa x Fb

Table 3 : Instructions flottantes