

ARCHITECTURE DES ORDINATEURS

Corrigé Examen Novembre 2012

3 H – Tous documents autorisés

OPTIMISATIONS DE PROGRAMME

On utilise le processeur superscalaire défini dans l'annexe 1. On suppose une version pipelinée du processeur utilisant les instructions MIPS. La latence des instructions est donnée dans la deuxième colonne de la Table 2. On rappelle qu'une latence de n signifie que si une instruction I démarre au cycle c , une instruction qui utilise le résultat de I ne peut démarrer qu'au cycle $c+n$. (une latence de 1 signifie qu'elle peut démarrer au cycle suivant).

La Table 1 présente un programme C et le programme assembleur MIPS correspondant. Les tableaux X , Y , W et Z sont rangés successivement à partir de l'adresse $0x10000000$, qui est contenue au démarrage dans le registre $R3$.

Table 1 : Programme C et programme assembleur

<pre>float X[100], Y[100], Z[100] S; int i; for (i=0; i<100; i++) S+=X[i]*Y[i] +W[i]* Z[i];</pre>	<pre>ADDI R5, R3, 400 FSUB F0, F0, F0 Boucle :LF F1,(R3) LF F2,400 (R3) LF F3,800(R3) LF F4,1200(R3) FMUL F1,F1,F2 FMUL F3,F3,F4 FADD F1,F1,F3 FADD F0,F0,F1 ADDI R3,R3,4 BNEQ R3,R5, Boucle SF F0, (adresse de S)</pre>
--	--

Q 1) Donner l'exécution cycle par cycle de la boucle optimisée (mais sans déroulage de boucle) en plaçant les instructions dans les différents pipelines E0, E1, FA et FM. Quel est, en nombre de cycles, le temps d'exécution par itération de la boucle ?

	E0	E1	FA	FM
1	LF F1,(R3)	LF F2,400 (R3)		
2	LF F3,800(R3)	LF F4,1200(R3)		
3	ADDI R3,R3,4			FMUL F1,F1,F2
4				FMUL F3,F3,F4
5				
6				
7			FADD F1,F1,F3	
8				
9				
10		BNEQ R3,R5, Boucle	FADD F0,F0,F1	

10 cycles /itération

	E0	E1	FA	FM
1	LF F1,(R3)	LF F2,400 (R3)		
2	LF F3,800(R3)	LF F4,1200(R3)		
3	ADDI R3,R3,4			FMUL F1,F1,F2
4				FMUL F3,F3,F4
5				
6			FADD F0,F0,F1	
7				
8				
9		BNEQ R3,R5, Boucle	FADD F0,F0,F3	

9 cycles /itération

Q 2)) Donner l'exécution cycle par cycle de la boucle optimisée avec un déroulage de boucle d'ordre 4 en plaçant les instructions dans les différents pipelines E0, E1, FA et FM. Quel est, en nombre de cycles, le temps d'exécution par itération de la boucle ?

	E0	E1	FA	FM
1	LF F1,(R3)	LF F2,400(R3)		
2	LF F5,4(R3)	LF F6,404(R3)		
3	LF F9,8(R3)	LF F10,408(R3)		FMUL F1,F1,F2
4	LF F13,12(R3)	LF F14,412(R3)		FMUL F5,F5,F6
5	LF F3,800(R3)	LF F4,1200(R3)		FMUL F9,F9,F10
6	LF F7,804(R3)	LF F8,1204(R3)		FMUL F13,F13,F14
7	LF F11,808(R3)	LF F12,1208(R3)		FMUL F3,F3,F4
8	LF F15,812(R3)	LF F16,1212(R3)		FMUL F7,F7,F8
9	ADDI R3,R3,16			FMUL F11,F11,F12
10			FADD F1,F1,F3	FMUL F15,F15,F16
11			FADD F5,F5,F7	
12			FADD F9,F9,F11	
13			FADD F13,F13,F15	
14			FADD F0,F0,F1	
15			FADD F17,F17,F5	
16			FADD F18,F18, F9	
17		BNEQ R3,R5, Boucle	FADD F19,F19,F13	

17 cycles pour 4 itérations soit 4,25 cycles/itération

	E0	E1	FA	FM
1	LF F1,(R3)	LF F2,400(R3)		
2	LF F3,800(R3)	LF F4,1200(R3)		
3	LF F5,4(R3)	LF F6,404(R3)		FMUL F1,F1,F2
4	LF F7,804(R3)	LF F8,1204(R3)		FMUL F3,F3,F4
5	LF F9,8(R3)	LF F10,408(R3)		FMUL F5,F5,F6
6	LF F11,808(R3)	LF F12,1208(R3)		FMUL F7,F7,F8
7	LF F13,12(R3)	LF F14,412(R3)	FADD F1,F1,F3	FMUL F9,F9,F10
8	LF F15,812(R3)	LF F16,1212(R3)		FMUL F11,F11,F12
9	ADDI R3,R3,16		FADD F5,F5,F7	FMUL F13,F13,F14
10			FADD F0,F0,F1	FMUL F15,F15,F16
11			FADD F9,F9,F11	
12			FADD F17,F17,F5	
13			FADD F13,F13,F15	
14			FADD F18,F18, F9	
15				
16		BNEQ R3,R5, Boucle	FADD F19,F19,F13	

16 cycles pour 4 itérations = 4 cycles/itération

CACHES.

On suppose qu'un processeur a un cache données de 64 Ko, avec des blocs de 64 octets. 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.

Q 3) Quel est pour ce cache le nombre de bits pour l'adresse dans la ligne, le nombre de bits d'index et le nombre de bits d'étiquette pour un cache à correspondance directe

Adresse dans la ligne = 6 bits
Index = 10 bits (1024 lignes)
Etiquette = 16 bits

Q 4) Dans quelles lignes du cache vont les flottants X[0] et X[2048] qui sont aux adresses 1000 0000_H et 1000 2000_H. ?

X[0] dans la ligne 0

X[2048]

1000 | 2000_H soit 0010 0000 00 | 00 0000 pour index et adresse dans la ligne
Index = 0010000000 = $2^7 = 128$

Q 5) Quel est le nombre total de défauts de caches lors de l'exécution des boucles ci-dessous pour le cache à correspondance directe pour les trois programmes suivants?

On considère les programmes suivants, pour lequel le tableau X est rangé en mémoire à partir de l'adresse 1000 0000_H (adresse de X[0].)

```
float x[4096] ;
```

```
1) for (i=0 ; i<2048 ; i++)
    S+= X[i];
2) for (i=0 ; i<2032 ; i++)
    S+= X[i] + X[i+16];
3) for (i=0 ; i<2048 ; i++)
    S+= X[i] + X[i+2048];
```

1 ligne = 16 floats

- 1) 1 défaut toutes les 16 itérations d'où $2048/16 = 128$ défauts
- 2) La première itération charge les lignes 1 et 2. Ensuite, il y a un défaut pour l'accès $i+16$. On est dans la même situation que pour la boucle 1) soit 128 défauts
- 3) $X[i]$ et $X[i+2048]$ vont dans deux lignes différentes. Il y a donc deux défauts toutes les 16 itérations, soit 256 défauts

On suppose maintenant un cache de 16 Ko à correspondance directe avec des lignes de 64 octets et la boucle

```
float X[4096] ;
for (i=0 ; i<2048 ; i++)
    S+= X[i] + X[i+2048];
```

Q 6) Quel est maintenant le nombre de défauts de cache lors de l'exécution de la boucle ?

$X[i]$ et $X[i+2048]$ vont toujours dans deux lignes différentes, donc 256 défauts

PROGRAMMATION OPENMP

On reprend le code C de la question 1

```
float X[1000], Y[1000], W[1000], Z[1000], S;
int i ;
for (i=0; i<100; i++)
    S+=X[i]*Y[i]+ W[i]* Z[i];
```

Q 7) Donner une version OpenMP de ce programme pour 4 processeurs qui minimise les défauts de cache.

```
float X[1000], Y[1000], W[1000], Z[1000], S=0.0, SP[4];
int i,k ;
    omp_set_num_thread(4);
#pragma omp parallel
{
    int i;
    id = omp_get_thread_num();
    SP[id]=0.0;
    for (i =id*250 ; i<(id+1)*250 ; i++)
        SP[id]+=X[i]*Y[i]+ W[i]* Z[i];
}
for (k=0;k<4;i++)
    S+= SP[i];
```

SIMD (IA-32 - SSE2)

La Figure 1 présente l'effet de l'instruction SIMD SHUFFPD sur des registres 128 bits contenant 2 flottants en double précision.

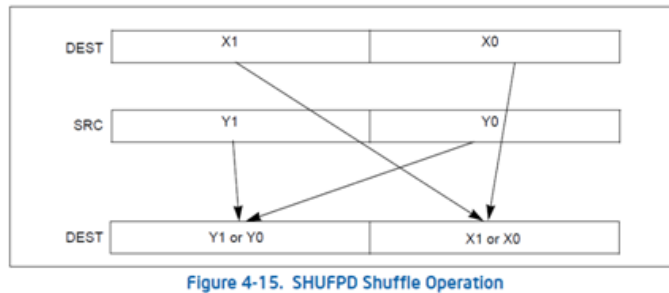


Figure 1 : Instruction SIMD SHUFFPD

`_mm_shuffle_pd (a,b,c)` est l'intrinsics correspondant . Si a et b sont des variables 128 bits (a_1, a_0, b_1 et b_0 étant des doubles) avec $a = a_1 | a_0$ et $b = b_1 | b_0$, l'action de cette instruction est :

```

_mm_shuffle_pd (a,b,0) = b0 | a0
_mm_shuffle_pd (a,b,1) = b0 | a1
_mm_shuffle_pd (a,b,2) = a1 | a0
_mm_shuffle_pd (a,b,3) = b1 | b0

```

Q 8) Quel est l'effet de l'intrinsics `_mm_shuffle_pd (x, x,1)` ?

Si $x = x_1 | x_0$ alors `_mm_shuffle_pd (x,x,1) = x0 | x1`

Soit l'extrait de programme C avec des « define » et la fonction XSIMD :

```

#define addpd(a,b) _mm_add_pd(a,b)           // addition simd 2 doubles + 2 doubles
#define subpd(a,b) _mm_sub_pd(a,b)          // soustraction simd 2 doubles - 2 doubles
#define divpd(a,b) _mm_div_pd(a,b)          // division simd 2 doubles / 2 doubles
#define setdouble(a) _mm_set1_pd(a)         // creation 2 doubles a | a
#define set2double(a,b) _mm_set_pd(a,b)     // creation 2 doubles a | b
#define perm(a,b,c) _mm_shuffle_pd(a,b,c)   // voir question précédente
#define stld(p,a) _mm_storel_pd(&p,a)       // rangement à l'adresse p du double (poids
                                                faible) de a

```

```

XSIMD ()
{
    int i;
    double x, res, k, c1=1.0, c2=3.0, z=0.0 ;
    __m128d a,b,c,d, e; // variables 128 bits (flottants)
    a=set2double(c1,c2); // a=1.0|3.0
    d=setdouble(z);      // d= 0.0|0.0
    e=setdouble(c1);     // e=1.0|1.0
    for (i=0;i<= etapes; i+=2){
        k=(double)(2*i); //k=2*i

```

```

    b=setdouble(k);           // b=k | k
    c=addpd(a,b);            // c=1.0+2*i | 3.0=2*i
    c=divpd(e,c);           // c= 1 / (1.0+k|3.0+k)
    d=addpd(d,c);           // d= Σ 1 / (1.0+k) | 1 / (3.0+k)
}
    b=perm(d,d,1);          // b = Σ 1 / (3.0+k) | 1 / (1.0+k)
    d=subpd(b,d);           // d= b-d
    stld(res,d);           // res = 1 / (1.0+k) - 1 / (3.0+k)
}

```

Q 9) Que fait la fonction XSIMD ? (On pourra soit donner le programme C scalaire équivalent, soit indiquer le calcul effectué à chaque itération et le résultat final). Quelle valeur contient la variable res à la fin d'exécution de la fonction ?

Programme C

```

{      int i;   double x, res=0.0, k ;

      for (i=0;i<= etapes; i+=2){
          k=(double)(2*i);
          x = 1/(k+1) -1/(k+3);
          res +=x;}

```

res = 1/1 - 1/3 + 1/5 - 1/7+...

res = $\pi/4$

ACCELERATION PARALLELE

Soit un problème de taille fixe.

Q 10) Quelle est le nombre de processeurs nécessaires pour avoir une accélération de 4 si la partie non parallélisable représente 20% du code séquentiel

$$\text{Acc} = \frac{S+P}{S+P/n} = \frac{1}{0,2+0,8/n} = 4$$

$$0,2 + \frac{0,8}{n} = 0,25$$

$$\frac{0,8}{n} = 0,05$$

$$n/0,8 = 20$$

$$n = 16.$$

Annexe 1

Soit un processeur superscalaire à ordonnancement statique qui a les caractéristiques suivantes :

- les instructions sont de longueur fixe (32 bits)
- Il a 32 registres entiers (dont R0=0) de 32 bits et 32 registres flottants (de F0 à F31) de 32 bits.
- Il peut lire et exécuter 4 instructions par cycle.
- L'unité entière contient deux pipelines d'exécution entière sur 32 bits, soit deux additionneurs, deux décaleurs. Tous les bypass possibles sont implantés.

- L'unité flottante contient un pipeline flottant pour l'addition et un pipeline flottant pour la multiplication. - L'unité Load/Store peut exécuter jusqu'à deux chargements par cycle, mais ne peut effectuer qu'un load et un store simultanément. Elle ne peut effectuer qu'un seul store par cycle.

- Il dispose d'un mécanisme de prédiction de branchement qui permet de "brancher" en 1 cycle si la prédiction est correcte. Les sauts et branchements ne sont pas retardés.

La **Erreur ! Source du renvoi introuvable.** donne les instructions disponibles et le pipeline qu'elles utilisent : E0 et E1 sont les deux pipelines entiers, FA est le pipeline flottant de l'addition et FM le pipeline flottant de la multiplication. Les instructions peuvent être exécutées simultanément si elles utilisent chacune un pipeline séparé. L'addition et la multiplication flottante sont pipelinées. La division flottante n'est pas pipelinée (une division ne peut commencer que lorsque la division précédente est terminée). L'ordonnancement est statique.

JEU D'INSTRUCTIONS (extrait)

LF	LF Fi, dép.(Ra)	2	E0 ou E1	$Fi \leftarrow M(Ra + \text{dépl.16 bits avec ES})$
SF	SF Fi, dép.(Ra)	1	E0	$Fi \rightarrow M(Ra + \text{dépl.16 bits avec ES})$
ADD	ADD Rd,Ra, Rb	1	E0 ou E1	$Rd \leftarrow Ra + Rb$
ADDI	ADDI Rd, Ra, IMM	1	E0 ou E1	$Rd \leftarrow Ra + \text{IMM-16 bits avec ES}$
SUB	SUB Rd,Ra, Rb	1	E0 ou E1	$Rd \leftarrow Ra - Rb$
FADD	FADD Fd, Fa, Fb	3	FA	$Fd \leftarrow Fa + Fb$
FMUL	FMUL Fd, Fa, Fb	3	FM	$Fd \leftarrow Fa \times Fb$
BEQ	BEQ Ri, Rj, dépl	1	E1	si $Ri=Rj$ alors $CP \leftarrow NCP + \text{depl}$
BNE	BNE Ri, Rj, dépl	1	E1	si $Ri \neq Rj$ alors $CP \leftarrow NCP + \text{depl}$

Table 2 : instructions disponibles