Université Paris Sud 2012-2013
M2R NSI - SETI

ARCHITECTURE DES ORDINATEURS
Examen Novembre 2012
3 H – Tous documents autorisés
5 parties indépendantes.

1. 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 1.
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

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];
	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)

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 ?

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 ?

2. 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

Q 4) Dans quelles lignes du cache vont les flottants X[0] et X[2048] qui sont aux adresses 1000 0000_H et 1000 4000_H. ?

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

```
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];
```

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 ?

3. 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 threads qui minimise les défauts de cache.

4. SIMD (IA-32 – SSE2)

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

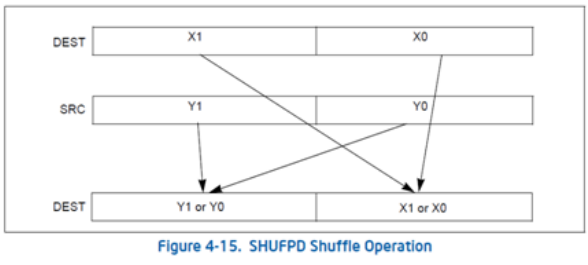

Figure 4-15. SHUFPD Shuffle Operation

Figure 1 : Instruction SIMD SHUFPD

_mm_shuffle_pd (a,b,c) est l'instrinsics correspondant . Si a et b sont des variables 128 bits (a1, a0, b1 et b0 étant des doubles) avec a=a1|a0 et b=b1|b0, 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'instrinsics _mm_shuffle_pd (x, x, 1) ?

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);
          d=setdouble(z);
          e=setdouble(c1);
for (i=0;i<= etapes; i+=2){
           k=(double)(2*i);
           b=setdouble(k);
           c=addpd(a,b);
           c=divpd(e,c);
           d=addpd(d,c);
 }
           b=perm(d,d,1);
           d=subpd(b,d);
          stld(res,d);
}
```

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 ?

5. 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

6. 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 Table 2 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)

Mnémo	Syntaxe	Latence	Pipeline	Effet
LF	LF Fi, dép.(Ra)	2	E0 ou E1	Fi ← M (Ra + dépl.16 bits avec ES)
SF	SF Fi, dép.(Ra)	1	E0	Fi -> M (Ra + dépl.16 bits avec ES)
ADD	ADD Rd,Ra, Rb	1	E0 ou E1	Rd ←Ra + Rb
ADDI	ADDI Rd, Ra, IMM	1	E0 ou E1	Rd ←Ra + IMM-16 bits avec ES
SUB	SUB Rd,Ra, Rb	1	E0 ou E1	Rd ←Ra - Rb
FADD	FADD Fd, Fa, Fb	3	FA	Fd ←Fa + Fb
FMUL	FMUL Fd, Fa, Fb	3	FM	Fd ←Fa x Fb
BEQ	BEQ Ri, Rj, dépl	1	E1	si Ri=Rj alors CP ← NCP + depl
BNE	BNE Ri, Rj, dépl	1	E1	si Ri≠Rj alors CP ← NCP + depl

Table 2 : Instructions disponibles