ARCHITECTURE DES ORDINATEURS
Corrigé Examen Décembre 2010
2 H – Tous documents autorisés
Les questions sont indépendantes

On utilise un sous-ensemble du jeu d'instructions MIPS donné en annexe. Les latences des instructions sont données dans l'annexe (figures 1 et 2)

EXECUTION DE BOUCLES

Soit le programme assembleur P1, qui travaille sur un tableau de flottants simple précision (float) X[N] rangé en mémoire, avec N = 512. L'adresse de A[0] est initialement contenue dans le registre R1. F0 contient la constante flottante 1.0.

```
        ADDI R2 ;R1, 1024
Boucle :LF F1, 0(R1)
        FADD F1,F1,F0
        SF F1, 0 (R1)
        LF F2, 1024(R1)
        FSUB F2,F2,F0
        SF F2, 1024(R1)
        ADDI R1,R1,4
        BNE R1,R2, Boucle
```

```
        ADDI R2 ;R1, 1024
Boucle :LF F1, 0(R1)
        FADD F1,F1,F0
        SF F1, 0 (R1)
        LF F2, 1024(R1)
        FADD F2,F2,F0
        SF F2, 1024(R1)
        ADDI R1,R1,4
        BLT R1,R2, Boucle
```

Question 1) Donner le code C correspondant au programme P1

```
float X[512]; i ;
for (i = 0; i < 256; i++) {
      X[i] = X[i] + 1.0;
      X[i+256] = X[i+256] - 1.0;
}
```

Question 2) En optimisant, montrer l'exécution cycle par cycle du programme assembleur P1 et donner le nombre de cycles par itération de la boucle du programme assembleur en supposant qu'il n'y a ni pénalité de branchement, ni défauts de cache. Quel est le temps d'exécution total en cycles d'horloge ?

```
        ADDI R2, R1,1024₁₀
1Boucle :LF F1, 0(R1)
2       LF F2, 1024(R1)
3       FADD F1,F1,F0
4       FSUB F2,F2,F0
5       ADDI R1,R1,4
6
7       SF F1, -4(R1)
8       SF F2, 1020 (R1)
9       BLT R1,R2, Boucle
```

9 cycles

Temps total : 1 + 256 *9 = 2305 cycles

Question 3) Quel est le nombre de cycles par itération de la boucle avec déroulage d'ordre 4, en supposant qu'il n'y ni pénalité de branchement, ni défauts de cache ?

```
        ADDI R2, R1,512₁₀
1Boucle :LF F1, 0(R1)
2       LF F3, 4(R1)
3       LF F5, 8(R1)
4        LF F7, 12(R1)
5       LF F2, 1024(R1)
6       LF F4, 1028 (R1)
7       LF F6, 1032 (R1)
8       LF F8, 1036 (R1)
9       FADD F1,F1,F0
10      FADD F3,F3,F0
11      FADD F5,F5,F0
12      FADD F7,F7,F0
13      FSUB F2,F2,F0
14      FSUB F4,F4,F0
15      FSUB F6,F6,F0
16      FSUB F8,F8,F0
17      SF F1, 0(R1)
18      SF F3, 4(R1)
19      SF F5, 8(R1)
20      SF F7, 12(R1)
21      SF F2, 1024(R1)
22      SF F4, 1028 (R1)
23      SF F6, 1032 (R1)
24      SF F8, 1036 (R1)
25      ADDI R1,R1,16
26      BLT R1,R2, Boucle
```

Soit 26/4 =6,5 cycles /itération

Temps d'exécution total : 1 +6.5*256 = 1665 cycles

CACHES

Soit un cache de 1 Ko à correspondance directe, avec des lignes (blocs) de 16 octets. Le processeur a des registres de 32 bits et des adresses de 32 bits. Le cache est à réécriture (write back) et allocation d'écriture (il y a des défauts de caches en écriture).

Question 4) Quels sont les nombres de bits nécessaire pour l'index, l'étiquette et le déplacement (adresse dans la ligne) ?

1 Ko = 2^{10}.
Lignes de 16 octets : 2^4.
Nombre de lignes : 2^6 = 64.
Déplacement : 4 bits
Index : 6 bits
Etiquette : 22 bits

Soit le code C.

```
int A[512], i ;
for (i = 0; i < 256; i++) {
      A[i] = A[i] + 1;
      A[i+256] = A[i+256] - 1;
}
```

Question 5) Quel est le taux d'échec (nombre de défauts / nombre d'accès) lors de l'exécution du code ci-dessus.

Entre l'adresse de A[i] et l'adresse de A[i+256], il y a 256 * 4 = 1024 octets, et les 10 bits de poids faible de l'adresse seront identiques. Les deux adresses iront dans la même ligne et il y aura donc 1 défaut de cache à chaque accès, soit un taux d'échec de 100%.

Question 6)

a) Donner deux techniques logicielles (modification du code) et une technique matérielle (modification du cache) qui permettraient d'obtenir le taux d'échec minimal compte tenu des défauts de démarrage.

Les lignes sont de 16 octets, soit 4 entiers. Il faut donc accéder à tous les éléments d'une ligne avant qu'elle soit éjectée ;

Première technique logicielle.
Déroulage de boucle d'ordre 4, soit le programme

```
int A[512], i ;
for (i = 0; i < 256; i += 4) {
      A[i] = A[i] + 1;
      A[i+1] = A[i+1] + 1;
      A[i+2] = A[i+2] + 1;
      A[i+3] = A[i+3] + 1;
      A[i+256] = A[i+256] - 1;
```

```
      A[i+257] = A[i+257] - 1;
      A[i+258] = A[i+258] - 1;
      A[i+259] = A[i+259] - 1;
}
```

Deuxième technique logicielle
Décomposer la boucle en deux boucles distinctes

```
int A[512], i ;
for (i = 0; i < 256; i++)
      A[i] = A[i] + 1;
for (i = 0; i < 256; i++)
      A[i+256] = A[i+256] - 1;
```

Technique matérielle
Utiliser un cache associatif par ensemble (deux voies).

b) Quel est alors le taux d'échec ?
Taux d'échec = 25% (les lignes ont 4 mots. L'accès au premier mot provoque un défaut de cache).

PREDICTION DE BRANCHEMENT

Dans cette partie, on considère le comportement d'un branchement qui est pris (P) ou non pris (N).
On dispose de plusieurs types de prédicteurs
- toujours pris
- toujours non pris
- prédicteur dynamique 1 bit
- prédicteur dynamique 2 bits

Question 7) Choisir un prédicteur pour chacun des branchements suivants, en choisissant celui nécessitant le moins de ressources matérielles si plusieurs prédicteurs ont les mêmes performances

a) Branchement 1
P, P, P, P, N, P, P, P, P, P, P, N, P, P, P, P, P, P, P, N, N, N, N, N, P, N, N, N, N, N, N, P, N, N, N, N
Prédicteur 2 bits
Longues séries de P, puis de N, avec au milieu un comportement inverse. Le prédicteur 2 bits ne fera de mauvaises prédictions que pour ces comportements inverses.

b) Branchement 2
P, P, P, P, P, P, N, P, P, P, P, P, P, P, P, N, P, P, P, P, P, P, P, P, P, N, P, P, P, P, P, P, P, P, P, P, P, N
Prédicteur statique « toujours pris »
Le branchement est pratiquement toujours pris. Le prédicteur statique « toujours pris » fera aussi bien qu'un prédicteur plus coûteux

c) Branchement 3
P, P, P, P, P, N, N, N, N, N, N, N, N, P, P, P, P, P, P, P, P, P, P, P, N, N, N, N, N, P, P, P, P, P, P
Prédicteur 1 bit

Longues séries de P ou de N. Le prédicteur 1 bit ne se trompera que lors des changements de comportements.

SIMD

Soit le programme P2 ci-dessous

```
Float X[512];
_m128 *XS, A,B,C

XS=X;
For (i=0 ;i<64 ;i++){
      A = lf4(&XS[i]) ;
      B = lf4 (&XS(i+64) ;
      C = addps (A,B) ;
      sf4 (&XS[i], C) ;}
```

Question 8) Donner le programme C scalaire correspondant au programme SIMD P2

for (i=0 : i< 256; i++)
        X[i] = X[i] + X[i+256];

PROGRAMMATION OPENMP

Soit le code C ci-dessous.

```
float X[128], Y[128], Z[128] ;
for (i = 0 ; i<128 ; i++)
      Z[i]= (X[i]+Y[i])*0.5 ;
```

Question 9) Donner une version OpenMP de ce programme pour 8 processeurs qui minimise les défauts de cache.

```
float X[128], Y[128], Z[128] ;
int k=128/8;
id=get_thread_number();
#pragma omp parallel
for (i = id*k ; i<(id+1)*k ; i++)
      Z[i]= (X[i]+Y[i])*0.5 ;
```

LOI D'AMDAHL

Un programme séquentiel a 20% de son temps d'exécution qui ne peut être parallélisé. On veut l'accélérer à taille constante.

Question 10) Quel est le nombre de processeurs nécessaire pour obtenir une efficacité parallèle de 50% ? (Efficacité parallèle = Accélération / Nombre de processeur).

EP = Acc/n = $\dfrac{1}{n*(0{,}2+\dfrac{0{,}8}{n})} = 0{,}5$

0,2 n + 0,8 = 2
0,2 n = 1,2 soit n =6

ANNEXE 1

Les figures donnent la liste des instructions disponibles.
La signification des abréviations est la suivante : IMM correspond aux 16 bits de poids faible d'une instruction. SIMM est une constante sur 32 bits, avec 16 fois le signe de IMM, suivi de IMM (extension de signe)
ADBRANCH est l'adresse de branchement, qui est égale à NCP+ SIMM (NCP est l'adresse de l'instruction qui suit le branchement)

ADD	1	ADD rd, rs, rt	rd ← rs + rt (signé)
ADDI	1	ADDI rt, rs, IMM	rt ← rs + SIMM (signé)
BEQ	1	BEQ rs,rt, IMM.	si rs = rt, branche à ADBRANCH
BNEQ	1	BNEQ rs,rt, IMM.	si rs ≠ rt, branche à ADBRANCH
J	1	J destination	Décale l'adresse destination de 2 bits à gauche, concatène aux 4 bits de poids fort de CP et saute à l'adresse obtenue
JAL	1	JAL destination	Même action que J . Range adresse instruction suivante dans R31
JR	1	JR rs	Saute à l'adresse dans rs

Figure 1 : Instructions entières MIPS utilisées (NB : les branchements ne sont pas retardés)

LF	2	LF ft, IMM(rs)	rt ← MEM [rs +SIMM]
SF	1	SF ft, IMM.(rs)	ft → MEM [rs + SIMM]
FADD	4	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FMUL	4	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
FSUB	4	FSUB fd, fs,ft	fd ← fs - ft (soustraction flottante simple précision)
FDIV	12	FDIV fd,fs,ft	fd ← fs / ft (division flottante simple précision)

Figure 2 : Instructions flottantes ajoutées (Ce ne sont pas les instructions MIPS)

ANNEXE 2 : Intrinsics SIMD

On utilise les #define suivants pour le jeu d'instructions SIMD IA-32

#define lf4(&a)	_mm_load_ps(&a)	chargement aligné de 4 floats
#define lfc(&a)	_mm_load1_ps(&a)	Chargement de 4 fois le float a
#define sf4(&a, b)	_mm_store_si128(&a, b)	Rangement aligné de 4 floats
#define addps(a,b)	_mm_add_ps(a,b)	Addition de 4 floats
#define mulps(a,b)	_mm_mul_ps(a,b)	Multiplication de 4 floats