ARCHITECTURE DES ORDINATEURS
Examen Janvier 2010
2 H – Tous documents autorisés -

On utilise un sous-ensemble du jeu d'instructions MIPS donné en annexe.

TEMPS D'EXECUTION DE BOUCLES

Soit le programme P1. R1 contient l'adresse de début d'un tableau X[N] et R2 l'adresse de début d'un tableau Y[N].

```
        ADDI R4,R1,400H
        ADD R3,R1,R0
Boucle:
        LF F1, 0(R1)
        LF F2, 0(R3)
        FADD F2,F1,F2
        SF F2, 0(R2)
        ADDI R1,R1,4
        ADDI R2,R2,4
        ADDI R3,R3,1024  // 1024₁₀ = 400H
        BNEQ  R1,R4,Boucle
```

Q 1) Donner le code C correspondant au programme P1

Q 2) Quel le temps d'exécution, en nombre de cycles d'horloge, de la boucle de P1 optimisé (mais sans déroulage de boucle)

Q 3) Quel est le temps d'exécution de la boucle de P1 optimisé avec un déroulage de boucle d'ordre 2 (2 itérations de la boucle initiale par corps de boucle) ?

Q 4) Le processeur ayant 32 registres flottants, quel est le facteur maximal de déroulage et quel est alors le nombre de cycles par itération de la boucle initiale ?

CACHES

On considère un cache de 16Ko, associatif 2 voies (2 blocs par ensemble) avec des blocs de 64 octets. Le cache est à écriture simultanée « non allouée » (pas de défauts de cache en écriture).

Soit le programme P2

Int A[1024*1024], int X = 0 ; // A[0] est implanté à l'adresse 0x0 ;

```
for (i=0 ; i<1024 ; i++)
      X+=A[i] ;
```

Q 5) Quel est le taux d'échecs (nombre d'échecs/nombre d'accès) ?

Soit le programme P3

```
for (i=0 ; i<1024 ; i++)
X+=A[i] + A[i*1024] ;
```

Q 6) Quel est maintenant le taux d'échec ?

SIMD

Soit le programme P4 ci-dessous

```
unsigned char X[1024];
define m constante;
for (i=m ; i<1024; i++)
      X[i] += X[i-m];
```

Q 7) Pour quelles valeurs de m peut on utiliser les instructions SIMD ?

Q 8) Donner le programme C correspondant au programme SIMD P5 ci-dessous

unsigned char X[2048];
_m128i XS, A,B,C ;
XS=X ;

```
A=ld16 (&X[0])
For (i=1; i<128; i++) {
      B=ld16(&XS[i]);
      C= add8(A,B)
      st16(&(XS[i],C);
      A=B; }
```

SIMD et OpenMP

Soit le programme P6 ci-dessous

```
#define N 4*2048
Float X[N], Y[N], A ;

For (i=0 ;i<N ;i++)
Y[i] += A*X[i] ;
```

Q 9) En utilisant les intrinsics ou les « define » fournis dans l'annexe 2, donner la version SIMD du programme P6

On suppose qu'on dispose maintenant d'un quadri-processeur (P0, P1, P2, P3) °.

Q 10) Donner une version OpenMP du programme P6 qui minimise les défauts de cache.

Q 11) Par rapport à la version initiale du programme P6, quelle est l'accélération maximale en utilisant à la fois les instructions SIMD et la programmation OpenMP sur le quadri-processeurs ?

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
#definemulps(a,b)	_mm_mul_ps(a,b)	Multiplication de 4 floats
#define ld16(&a)	_mm_load_si128(&a)	Chargement aligné de 128 bits
#define st16(&a, b)	_mm_store_si128(&a, b)	Rangement aligné de 128 bits
#define add8 (a,b)	_mm_add_epi8(a,b)	Addition signé de 16 octets