ARCHITECTURE DES ORDINATEURS
Examen Novembre 2011
2 H – Tous documents autorisés

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

TEMPS D'EXECUTION DE BOUCLES

Soit les programmes assembleur P1 et P2.
R1 contient initialement l'adresse d'un tableau X[128] de flottants simple précision.
R2 contient initialement l'adresse d'un tableau Y[128] de flottants simple précision.
R3 contient initialement la valeur 126.

P1 :

```
Boucle :   LF F1, 0(R1)
           LF F2, 4(R1)
           LF F3, 8(R1)
           FADD F0,F1,F2
           FADD F0,F0,F3
           SF F0, 4(R2)
           ADDI R1,R1,4
           ADDI R2,R2,4
           ADDI R3,R3,-1
           BGTZ R3, Boucle
```

P2

```
Boucle :   LF F1, 0(R1)
           LF F2, 4(R1)
           LF F3, 8(R1)
           LF F4,12(R1)
           FADD F5, F2,F3
           FADD F6, F1, F5
           FADD F7, F4, F5
           SF F6, 4(R2)
           SF F7, 8(R2)
           ADDI R1,R1,8
           ADDI R2,R2,8
           ADDI R3, R3, -2
           BGTZ R3, Boucle
```

Q 1) Donner le code C correspondant aux programme P1 et P2

Les latences des instructions flottantes sont
- LF : 2 cycles
- FADD : 3 cycles
- SF : 1 cycle

Q 2) Quel le temps d'exécution en nombre de cycles d'horloge par itération de la boucle de P1 optimisée

Q 3) Quel est le temps d'exécution en nombre de cycles d'horloge par itération de la boucle de P2 optimisée ?

CACHES

On considère un cache de données de 8 Ko, à correspondance directe, avec des blocs de 64 octets. L'écriture est allouée (il y a des défauts de caches en écriture)

Soit le programme C suivant

```
float A[256], B[256] ;   //&A[0] = 1000 0000H et &B[0]=1000 0400H.
for (i=1 ; i<255 ; i++)
        B[i] = ( A[i-1] + A[i+1] ) * 0.5;
```

Q 4) Quel est le nombre de défauts de caches pour exécuter le programme en lecture et en écriture ?

Q 5) Les deux tableaux étant rangés successivement à partir de &A[0] = 1000 0000H, pour quelle valeur de N puissance de 2 a-t-on deux défauts par itération ? Quelle est alors en hexadécimal l'adresse de B[0]

SIMD lA-32

On utilise le jeu d'instructions lA-32, avec les « define » ci-dessous pour les intrinsics IA-32.

#define ld4f(&a)	_mm_load_ps(float * p)	chargement aligné 4 floats
#define st4f(&a, b)	_mm_store_ps(float *p, a)	rangement aligné 4 floats
#define maxf(a,b)	_mm_max_ps(a, b)	max 4 floats
#define minf(a,b)	_mm_min_ps(a, b)	min 4 floats
#define exten(w)	_mm_set_ps1(w)	Création d'un mot de 4 floats de valeur x

Soit le programme

```
__m128 X[1024],  W, Y, Z //  mots de 128 bits (4 floats)
float A, B, XS[4] ; int i;
Y = ld4f (&X[0];
Z=Y;
for (i=1; i<1024,;i++){
        W= ld4f (&X[i]);
Y= maxf (W, Y) ;
Z= minf (W, Z)}
XS=&Y ;
A=XS[0] ;
for (i=1 ;i<4 ; i++)
        if (A < XS[i])  A=XS[i];
XS = &Z;
B=XS[0];
for (i=1 ;i<4 ; i++)
        if (B> XS[i])  B=XS[i];
```

Q 6) Que fait le programme ? (Que contiennent les variables A et B en fin d'exécution du programme ?)

Q 7) Donner la version scalaire du programme.

OpenMP

Soit le programme C suivant :

```
#define N 65636
float x[N] ;
float sum, ave ;

for (i=0 ; i<N; i++)
      sum += x[i];
ave= sum/N;
```

Q 8) Sur un multiprocesseur à 4 processeurs, paralléliser le programme avec des directives OpenMP

Loi d'Amdahl

Q 9) Quelle est le pourcentage maximal de la partie non parallélisable d'un programme de taille fixe pour obtenir une efficacité parallèle de 75% avec 16 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.
ZIMM est une constante sur 32 bits, avec 16 zéros suivis de IMM (extension de zéros)
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)
ADDIU	1	ADDIU rt, rs, IMM	rt ← rs + SIMM (le contenu des registres est non signé)
ADDU	1	ADDU rd, rs, rt	rd ← rs + rt (le contenu des registres est non signé)
AND	1	AND rd, rs, rt	rd ← rs and rt
ANDI	1	ANDI rt, rs, IMM	rt ← rs and ZIMM
BEQ	1	BEQ rs,rt, IMM.	si rs = rt, branche à ADBRANCH
BGEZ	1	BGEZ rs,IMM.	si rs ≥ 0, branche à ADBRANCH
BGEZAL	1	BGEZAL rs, IMM.	adresse de l'instruction suivante dans R31 si rs ≥ 0, branche à ADBRANCH
BGTZ	1	BGTZ rs,IMM.	si rs > 0, branche à ADBRANCH
BLEZ	1	BLEZ rs,IMM.	si rs ≤ 0, branche à ADBRANCH
BLTZ	1	BLTZ rs,IMM.	si rs < 0, branche à ADBRANCH
BLTZAL	1	BLTZAL rs, IMM.	adresse de l'instruction suivante dans R31. si rs < 0, 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
LUI	1	LUI rt, IMM	Place IMM dans les 16 bits de poids fort de rt. Met 0 dans les 16 bits de poids faible de rt
LB	2	LB rt, IMM(rs)	Rt $_{7\text{-}0}$ ← MEM8 [rs + SIMM] ; Rt 31-8 ← extension de signe
LBU	2	LBU rt, IMM. (rs)	Rt $_{7\text{-}0}$ ← MEM8 [rs + SIMM] ; Rt 31-8 ← extension de zéros.
LW	2	LW rt, IMM.(rs)	rt ← MEM [rs + SIMM]
OR	1	AND rd, rs, rt	rd ← rs or rt
ORI	1	ANDI rt, rs, IMM	rt ← rs or ZIMM
SLL	1	SLL rd, rt, nb	Décale rt à gauche de nb bits et range dans rd
SLT	1	SLT rd, rs, rt	rd ← 1 si rs < rt avec rs signé et 0 autrement
SLTI	1	SLTI rt, rs, IMM	rt ← 1 si rs < SIMM avec rs signé et 0 autrement
SLTIU	1	SLTIU rt, rs, IMM	rt ← 1 si rs < ZIMM avec rs non signé et 0 autrement
SLTU	1	SLTU rt, rs, rt	rd ← 1 si rs < rt avec rs et rt non signés et 0 autrement
SRA	1	SRA rd, rt, nb	Décaler (arithmétique) rt à droite de nb bits et ranger dans rd
SRL	1	SRL rd, rt, nb	Décaler (logique) rt à droite de nb bits et ranger dans rd.
SUB	1	SUB rd, rs, rt	rd ← rs - rt (signé)
SUBU	1	SUBU rd rs, rt	rd ← rs - rt(non signé)
SW	1	SW rt, IMM.(rs)	rt => MEM [rs + IMM]
XOR	1	XOR rd, rs, rt	rd ← rs xor rt
XORI	1	XORI rt, rs, IMM	rt ← rs xor ZIMM

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	3	FADD fd, fs,ft	fd ← fs + ft (addition flottante simple précision)
FMUL	3	FMUL fd, fs,ft	fd ← fs * ft (multiplication flottante simple précision)
FSUB	3	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)