

ARCHITECTURE DES ORDINATEURS

Corrigé - 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

Programme P1 :

```
float X[128], Y[128] ;
for (i=1 ; i<127 ; i++)          //for (i=0 ; i<126 ; i++)
    Y[i] = X[i-1]+X[i]+X[i+1];   // Y[i+1] = X[i]+X[i+1]+X[i+2];
```

Programme P2 :

```
float X[128], Y[128] ;
for (i=1 ; i<127 ; i+=2){
    Y[i] = X[i-1]+X[i]+X[i+1];
    Y[i+1] = X[i]+X[i+1]+X[i+2];}
```

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

Les latences des instructions flottantes sont

- LF : 2 cycles
- FADD : 3 cycles
- SF : 1 cycle

Boucle :	LF F1, 0(R1)
2	LF F2, 4(R1)
3	LF F3, 8(R1)
4	FADD F0,F1,F2
5	ADDI R1,R1,4
6	ADDI R2,R2,4
7	FADD F0,F0,F3
8	ADDI R3,R3,-1
9	
10	SF F0, 0(R2)
11	BGTZ R3, Boucle

11 cycles/itération

Q 3) Quel est le temps d'exécution de la boucle de P2 optimisée ?

Boucle :	LF F1, 0(R1)
2	LF F2, 4(R1)
3	LF F3, 8(R1)
4	LF F4,12(R1)
5	FADD F5, F2,F3
6	ADDI R1,R1,8
7	ADDI R2,R2,8
8	FADD F6, F1, F5
9	FADD F7, F4, F5
10	ADDI R3, R3, -2
11	SF F6, -4(R2)
12	SF F7, 0(R2)
13	BGTZ R3, Boucle

13 cycles pour 2 itérations soit 6,5 cycles/itération

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];
For (i=1 ; i<255 ; i++)
    B[i] = ( A[i-1] + A[i+1] ) * 0.5;
```

&A[0] = 1000 0000H et &B[0]=1000 0400H.

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

Le cache a 128 lignes contenant 16 floats.

A[0] va dans la ligne 0

B[0] va dans la ligne 16

Il n'y a donc pas de défauts de conflit entre les tableaux A et B

- A[0] à A[15] sont chargés par l'itération 0 dans la ligne 0, et utilisé par A[i+1] jusqu'à l'itération 14. L'itération 15 charge A[16] à A[31] dans la ligne 1. Donc A[14] et A[16] puis A[15] et A[17] sont disponibles pour les itérations correspondantes.

Ensuite, toutes les itérations $j \pmod{16} = 15$ chargent 16 floats utilisés.

Il y a donc 1 défaut de cache en lecture toutes les 16 itérations et 1 défaut de cache en écriture toutes les 16 itérations.

Au total, il y a donc 1 défaut toutes les 8 itérations, soit un total de 32 défauts.

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]

Il faut que les adresses de A[0] et B[0] soient telles que les 13 bits de poids faible de B[0] soient à 0. Toutes les valeurs de N puissance de 2 supérieures ou égales à N=11 provoquent 2 défauts/itération..

SIMD IA-32

On utilise le jeu d'instructions IA-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];
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 ?)

Calcule la valeur maximale et la valeur minimale d'un tableau de 4096 flottants simple précision. Les valeurs max et min sont dans la

Q 7) Donner la version scalaire du programme.

```
float XS[4096], MIN, MAX ;
int i ;
MIN=XS[0]; MAX=MIN;
fFor (i=1; i<4096; i++)
if (XS[i] < MIN) MIN=XS[i];
if (XS[i] > MAX) MAX=XS[i];
}
```

OpenMP

Soit le programme C suivant :

```
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

```
#include omp.h
N=65636
Float x[N] ;
Float sum, ave ;
#pragma omp parallel for reduction(+:sum)
```

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

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 ?

$$EP = Acc/n = \frac{1}{16 * (S + \frac{1-S}{16})} = 0,75$$

$$16S+1-S = 1/0,75 = 4/3$$

$$15S = 1/3$$

$$S=1/45 = 2,22\%$$

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)	$Rt_{7-0} \leftarrow MEM8 [rs + SIMM]$; $Rt_{31-8} \leftarrow$ extension de signe
LBU	2	LBU rt, IMM. (rs)	$Rt_{7-0} \leftarrow MEM8 [rs + SIMM]$; $Rt_{31-8} \leftarrow$ extension de zéros.
LW	2	LW rt, IMM.(rs)	$rt \leftarrow MEM [rs + SIMM]$
OR	1	AND rd, rs, rt	$rd \leftarrow rs \text{ or } rt$
ORI	1	ANDI rt, rs, IMM	$rt \leftarrow rs \text{ or } ZIMM$
SLL	1	SLL rd, rt, nb	Décale rt à gauche de nb bits et range dans rd
SLT	1	SLT rd, rs, rt	$rd \leftarrow 1$ si $rs < rt$ avec rs signé et 0 autrement
SLTI	1	SLTI rt, rs, IMM	$rt \leftarrow 1$ si $rs < SIMM$ avec rs signé et 0 autrement
SLTIU	1	SLTIU rt, rs, IMM	$rt \leftarrow 1$ si $rs < ZIMM$ avec rs non signé et 0 autrement
SLTU	1	SLTU rt, rs, rt	$rd \leftarrow 1$ 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	$rd \leftarrow rs - rt$ (signé)
SUBU	1	SUBU rd rs, rt	$rd \leftarrow rs - rt$ (non signé)
SW	1	SW rt, IMM.(rs)	$rt \Rightarrow MEM [rs + IMM]$
XOR	1	XOR rd, rs, rt	$rd \leftarrow rs \text{ xor } rt$
XORI	1	XORI rt, rs, IMM	$rt \leftarrow rs \text{ xor } ZIMM$

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

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

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