Université Paris Sud 2009-2010
IFIPS-Info-2

ARCHITECTURE DES ORDINATEURS
Corrigé - 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
        BNE  R1,R4,Boucle
```

Q 1) Donner le code C correspondant au programme P1

```
For (i=0 ; i<256 ; i++)
        Y[i] = X[i] + X[i*256] ;
```

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

1 boucle	LF F0, 0(R1)
2	LF F1, 0(R3)
3	ADDI R1,R1,4
4	FADD F1,F1,F0
5	ADDI R2,R2,4
6	ADDI R3,R3,1024
7	
8	SF F1, 0(R2)
9	BNE R1,R4,Boucle

9 cycles/itération

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) ?

1 boucle	LF F0, 0(R1)
2	LF F1, 4(R1)
3	LF F2, 0(R3)
4	LF F3, 1024(R3)
5	FADD F1,F1,F0
6	FADD F3,F3,F2
7	ADDI R1,R1,8
8	ADDI R2,R2,8
9	SF F1, 0(R2)
10	SF F3, 4(R2)
11	ADDI R3,R3,2048
12	BNE R1,R4, Boucle

12 cycles/2 itérations soit 6 cycles par itérations

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 ?

Chaque itération utilisant deux registres, le facteur de déroulage maximal est donc 16
Chaque itération utilise 3 instructions mémoire (2LF et 1SF), 1 opération flottante (FADD). Il y a 4 instructions de gestion de boucle (3 ADDI et BNE)
Le temps d'exécution est donc 16 x 4 + 4 = 68 cycles/16 itérations = 4,25 cycles/itération.

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
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) ?

Il y a 16 entiers (int) par bloc. Il y a 1 échec tous les 16 accès, soit un taux d'échec de 1/16 = 6,25%.

Le programme devient

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

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

Avec un cache associatif 2 voies, il n'y a pas de conflits entre les deux accès à A pour chaque valeur de i.

Considérons i=0 à 15. Pour A[i], il y a un défaut de cache toutes les 16 itérations. Pour A[i*1024], il y a un défaut de cache par accès, sauf pour i=0. Pour 32 premiers accès, il y a donc 1+15 = 16 défauts. Pour les accès suivants (i=16 à 1024), il y a 1+16 = 17 défauts
Le taux d'échec est de (16 + 63 * 17) / (64 * 32) = 53,08 %

SIMD

Soit le programme P 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 ?

Il faut que m ≥ 16 pour éviter la récurrence. (Par ailleurs, si m est un multiple de 16, on évite les problèmes d'alignement).

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

```
unsigned char X[1024];
define m 16;
_m128i Xs, A,B,C ;
XS=X ;

A=ld16 (&X[0])
For (i=1; i<128; i++) {
        B=ld16(&X[i]);
        C= add8(A,B)
        St16( &(X[i],C);
        A=B; }

For (i=16; i<1024;i++)
        X[i] += X[i-16] ;
```

SIMD et OpenMP

Soit le programme C nommé P1 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 P1, après les déclarations

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

```
_m128 *XS, *YS, AS, T0,T1,T2;
/* version SIMD à écrire*/

#define N 4*2048
Float X[N], Y[N], A ;
_m128 *XS, *YS, AS, T0,T1,T2;

XS = X ; YS=Y ;
AS= lfc(&A) ;
for (i=0 ; i<N/4 ; i++){
      T0 = lf4 (&XS[i]);
      T1 = lf4(&YS[i°]) ;
      T0= mulps(AS,T0) ;
      T0= addps (T1,T0) ;
      Sf4 (&YS[i],T0;}
```

Peut être réécrit sous la forme

```
for (i=0 ; i<N/4 ; i++)
      sf4 (&YS[i], addps (lf4(&YS[i]), mulps(AS, lf4 (&XS[i]))))
```

On suppose qu'on dispose d'un quadri-processeur (P0, 1, P2, P3) °.

Q 10) Donner une version OpenMP du programme P1 dans la question précédente qui minimise les défauts de cache.

Programme P1

```
#define N 4*2048
Float X[N], Y[N], A ;
omp_set_num_thread (4);
#pragma omp parallel
{
id = omp_get_thread_num();
for (i =id*N/4 ; i<(id+1)*N/4 ; i++)
      Y[i] += A*X[i] ;
}
```

Q 11) Par rapport à la version initiale du programme P1, quelle accélération maximale peut on espérer en utilisant à la fois les instructions SIMD et la programmation OpenMP sur le quadri-processeurs ?

Facteur 4 (SIMD) et Facteur 4 (OpenMP) soit 16.

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} ← MEM8 [rs + SIMM] ; Rt 31-8 ← extension de signe
LBU	2	LBU rt, IMM. (rs)	Rt_{7-0} ← 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	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