Corrigé Examen Juin 2011 - Architectures Avancées

3H – Tous documents autorisés

Parties indépendantes

OPTIMISATION DE BOUCLES

On utilise le processeur superscalaire défini dans l'annexe 1

Soit la boucle suivante écrite en assembleur, qui travaille sur des vecteurs de « floats » X[N], A[N], B[N], C[N] et D[N] ; R1 contient la valeur 0001 0000H et R2 contient la valeur N

Les adresses des vecteurs sont

$0001\ 0000_H$	X(0)
$0001\ 1000_H$	A(0)
$0001\ 2000_H$	B(0)
$0001\ 3000_H$	C(0)
$0001\ 4000_H$	D(0)

Soit la boucle assembleur

```
Boucle :  LF F1, 1000H (R1)
          LF F2, 2000H(R1)
          LF F3, 3000H (R1)
          LF F4, 4000H(R1)
          FMUL F1,F1,F2
          FMUL F3,F3,F4
          FADD F1,F1,F3
          SF F1, 0(R1)
          ADDI R1,R1,4
          ADDI R2,R2,-1
          BNE R2, Boucle
```

Question 1) Donner le programme C équivalent.

```
float X[N], A[N], B[N], C[N], D[N];

main ()
{
int i ;
for (i=0;i<N;i++)
      X[i] = A[i]*B[i] + C[i]*D[i];
```

Question 2) On suppose d'abord que l'on utilise un processeur scalaire (une seule instruction démarrée par cycle). Optimisez la suite d'instructions ? Quel est, en nombre de cycles, le temps d'exécution par itération de la boucle originale ?

Boucle 1	LF F1, 1000_H (R1)
2	LF F2, 2000_H(R1)
3	LF F3, 3000_H (R1)
4	LF F4, 4000_H(R1)
5	FMUL F1,F1,F2

6	FMUL F3,F3,F4
7	ADDI R1,R1,4
8	ADDI R2,R2,-1
9	FADD F1,F1,F3
10	
11	
12	SF F1, -4(R1)
13	BNE R2, Boucle

13 cycles

Question 3) Avec le processeur superscalaire, donner l'exécution cycle par cycle de la boucle en plaçant les instructions dans les différents pipelines E0, E1, FA et FM. Quel est, en nombre de cycles, le temps d'exécution par itération de la boucle originale ?

	E0	E1	FA	FM
Boucle	LF F1, 1000_H (R1)	LF F2, 2000_H(R1)		
2	LF F3, 3000_H (R1)	LF F4, 4000_H(R1)		
3	ADDI R1,R1,4	ADDI R2,R2,-1		FMUL F1,F1,F2
4				FMUL F3,F3,F4
5				
6				
7			FADD F1,F1,F3	
8				
9				
10	SF F1, -4(R1)	BNE R2, Boucle		

10 cycles

Question 4) Refaire la question 3) avec un déroulage de boucle d'ordre 4.

	E0	E1	FA	FM
Boucle	LF F1, 1000_H (R1)	LF F2, 2000_H(R1)		
2	LF F3, 3000_H (R1)	LF F4, 4000_H(R1)		
3	LF F5, 1004_H (R1)	LF F6, 2004_H(R1)		FMUL F1,F1,F2
4	LF F7, 3004_H (R1)	LF F8, 4004_H(R1)		FMUL F3,F3,F4
5	LF F9, 1000_H (R1)	LF F10, 2000_H(R1)		FMUL F5,F6,F5
6	LF F11, 3000_H (R1)	LF F12, 4000_H(R1)		FMUL F7,F8,F7
7	LF F13, 1004_H (R1)	LF F14, 2004_H(R1)	FADD F1,F1,F3	FMUL F9,F1,F2
8	LF F15, 3004_H (R1)	LF F16, 4004_H(R1)		FMUL F11,F11,F12
9	ADDI R1,R1,16	ADDI R2,R2,-4	FADD F5,F5,F7	FMUL F13,F13,F14
10	SF F1,-16(R1)			FMUL F15,F15,F16
11			FADD F9,F9,F11	
12	SF F5, -12(R1)			
13			FADD F13,F13,F15	

14	SF F9, -8(R1)			
15				
16	SF F13, -4(R1)	BNE R2, Boucle		

16 cycles/4 = 4 cycles/itération

PIPELINE LOGICIEL AVEC TMS 320C62

Le code assembleur TMS320C62 ci-dessous donne l'itération du pipeline logiciel pour un programme C.
A4 contient au départ l'adresse d'un tableau d'entiers 16 bits X[N] et B4 contient l'adresse d'un tableau d'entiers 16 bits Y[N].
L'instruction MPY effectue une multiplication 16 bits par 16 bits sur les parties basses des registres et MPYH effectue une multiplication 16 bits par 16 bits sur les parties hautes des registres.

```
LOOP:
            LDW   .D1   *A4++,A2          ; load ai and ai+1
||          LDW   .D2   *B4++,B2          ; load bi and bi+1
||[A0]      ADD   .L1   A6,A7,A7          ; sum0+=(ai*bi)si A0≠0
||[B0]      ADD   .L2   B6,B7,B7          ; sum1+=(ai+1*bi+1) si B0≠0
||[A1]      B     .S2   LOOP              ; branch to loop si A1≠0
            MPY   .M1X  A2,B2,A6          ; ai*bi
||          MPYH  .M2X  A2,B2,B6          ; ai+1*bi+1
||          CMPLT .L1    0,A6,A0          ; A0 = 1 si A6 >0
||          CMPLT .L2 0,B6,B0       ; B0 = 1 si B6 >0
||[A1]      SUB   .S1   A1,1,A1     ; decrement loop counter si A1≠0
            ADD   .L1X  A7,B7,A4          ; sum=sum0+sum1
```

Question 5 : Donner le code C correspondant au code assembleur.

```
short  X[N], Y[N], S,S1,S2, tmp1, tmp2 ;
S1=0; S2=0;
for (i = 0 ; i<N ; i+=2){
      tmp1 = X[i]*Y[i];
      tmp2 = X[i+1]*Y[i+1];
      if (tmp1 >0) S1+=tmp1 ;
      if (tmp2 >0) S2+=tmp2 ;
}
S=S1+S2
```

Question 6 : Quel est l'intervalle inter-itération (II) ? Justifier la valeur.

L'intervalle inter-itération est de 2. Il y a deux instructions utilisant L1 et deux instructions utilisant L2, d'où la valeur 2.

PREDICTION DE BRANCHEMENT

Un branchement a le comportement suivant, où ¨P signifie pris et N signifie non pris.

PPNPPPPNPPPPNPPPPNPPPPNPPPPNPPPPN….

Question 7 : après la phase d'initialisation, quelle est le pourcentage de bonne prédiction

a) avec un prédicteur 1 bit
PPNPP PPNPPPPNPPPPNPPPPNPPPPN
xPPNP PPPNPPPPNPP PPNPPPPNPPPPNPPPPN
3 bonnes prédictions sur 5 soit 60%

b) avec un prédicteur 2 bits.
PPNPP PPNPPPPNPPPPNPPPPNPPPPNPPPPN
PPPpPP
4 bonnes prédictions sur 5 soit 80%

SIMD IA-32

Soit le programme C suivant utilisant des instructions SIMD. Il travaille sur des tableaux d'octets non signés X[N], Y[N] et A[N]. XS, YS et AS sont des tableaux de mots de 128 bits. Les trois tableaux d'octets ont la même adresse de début que les trois tableaux de mots de 128 bits correspondants et tous les tableaux sont alignés sur des frontières de mots de 128 bits (les « loads » sont alignés).

```
_m128i a, b, c, d
   for(i=0; i<32; i++) {
      a = ld16(&XS[i]);
      b = ld16(&YS[i]);
      c = addu (a,b);
      st16(&AS[i], c)}
```

Question 8) Donner la version C scalaire correspondant à la version SIMD ? Que fait ce programme ?

unsigned char A[512], X[512], Z[512];
for (i=0; i<512;i++)
A[i] = X[i]+ Y[i];

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];

Question 9 : 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] ;

Question 10 : 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 %

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

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

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

Annexe 1

Soit un processeur superscalaire à ordonnancement statique qui a les caractéristiques suivantes :
- les instructions sont de longueur fixe (32 bits)
- Il a 32 registres entiers (dont R0=0) de 32 bits et 32 registres flottants (de F0 à F31) de 32 bits.
- Il peut lire et exécuter 4 instructions par cycle.
- L'unité entière contient deux pipelines d'exécution entière sur 32 bits, soit deux additionneurs, deux décaleurs. Tous les bypass possibles sont implantés.
- L'unité flottante contient un pipeline flottant pour l'addition et un pipeline flottant pour la multiplication. - L'unité Load/Store peut exécuter jusqu'à deux chargements par cycle, mais ne peut effectuer qu'un load et un store simultanément. Elle ne peut effectuer qu'un seul store par cycle.
- Il dispose d'un mécanisme de prédiction de branchement qui permet de "brancher" en 1 cycle si la prédiction est correcte. Les sauts et branchements ne sont pas retardés.

La Table 1 donne les instructions disponibles et le pipeline qu'elles utilisent : E0 et E1 sont les deux pipelines entiers, FA est le pipeline flottant de l'addition et FM le pipeline flottant de la multiplication. Les instructions peuvent être exécutées simultanément si elles utilisent chacune un pipeline séparé. La table 1 donne également les latences des instructions.
L'addition et la multiplication flottante sont pipelinées. La division flottante n'est pas pipelinée (une division ne peut commencer que lorsque la division précédente est terminée).
L'ordonnancement est statique.

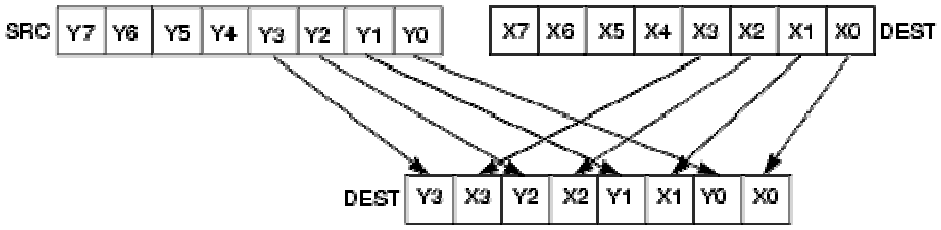
JEU D'INSTRUCTIONS (extrait)

Instruction	Latence	Syntaxe assembleur	Pipeline	Action
LF	2	LF Fi, dép.(Ra)	E0 ou E1	Fi ← M (Ra + dépl.16 bits avec ES)
SF	1	SF Fi, dép.(Ra)	E0	Fi -> M (Ra + dépl.16 bits avec ES)
ADD	1	ADD Rd,Ra, Rb	E0 ou E1	Rd ←Ra + Rb
ADDI	1	ADDI Rd, Ra, IMM	E0 ou E1	Rd ←Ra + IMM-16 bits avec ES
SUB	1	SUB Rd,Ra, Rb	E0 ou E1	Rd ←Ra - Rb
FADD	3	FADD Fd, Fa, Fb	FA	Fd ←Fa + Fb
FMUL	3	FMUL Fd, Fa, Fb	FM	Fd ←Fa x Fb
BEQ	1	BEQ Ri, dépl	E1	si Ri=0 alors CP ← NCP + depl
BNE	1	BNE Ri, dépl	E1	si Ri≠0 alors CP ← NCP + depl

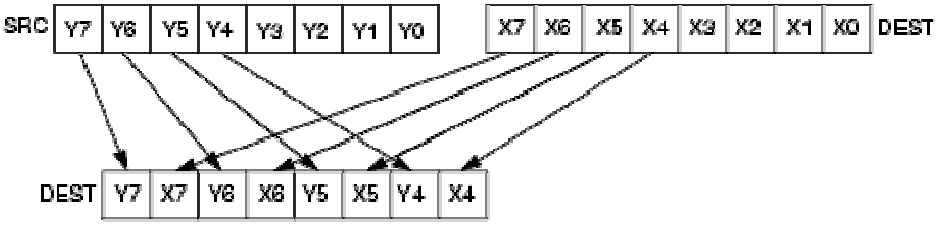
Table 1 : instructions disponibles

ANNEXE 2 : Instructions SIMD IA-32 utilisables

#define int2float(a)	_mm_cvtepi32_ps (a)	Convertit 4 entiers 32 bits signés en 32 bits flottants
#define ld16(a)	_mm_load_si128(&a)	chargement aligné
#define st16(a, b)	_mm_store_si128(&a, b)	rangement aligné
#define por(a,b)	_mm_or_si128(a,b)	ou logique
#define pxor(a,b)	_mm_xor_si128(a,b)	ou exclusif
#define maxbu(a,b)	_mm_max_epu8(a,b)	max 8 bits non signés
#define minbu(a,b)	_mm_min_epu8(a,b)	min 8 bits non signés
#define addh(a,b)	_mm_add_epi16(a,b)	addition 16 bits signée
#define addhu(a,b)	_mm_add_epu16(a,b)	addition 16 bits non signée
#define subh(a,b)	_mm_sub_epi16(a,b)	soustraction 16 bits signée
#define subbu (a,b)	_mm_subs_epu8(a,b)	Soustraction 8 bits non signés avec saturation
#define b2hl(a,b)	_mm_unpacklo_epi8 (a,b)	8 octets bas entrelacés vers 8 shorts
#define h2intl(a,b)	_mm_unpacklo_epi116 (a,b)	4 shorts bas entrelacés vers 4 int
#define b2hh(a,b)	_mm_unpackhi_epi8 (a,b)	8 octets haut entrelacés vers 8 shorts
#define h2inth(a,b)	_mm_unpackhi_epi16 (a,b)	4 shorts haut entrelacés vers 4 int
#define h2b(a,b)	_mm_packus_epi16(a,b)	8 shorts (a) dans 8 octets bas - 8 shorts (b) dans 8 octets haut
#define srli128(a,v)	_mm_srli_si128(a,v)	décalage droite des 128 bits de a de v octets
#define slli128(a,v)	_mm_slli_si128(a,v)	décalage gauche des 128 bits de a de v octets
#define srai16(a,v)	_mm_srai_epi16(a,v)	déc. droite des8x 16 bits de a de v bits
#define slli16(a,v)	_mm_slli_epi16(a,v)	déc. gauche des 8x16 bits de a de v bits
#define cmpgt8(a,b)	_mm_cmpgt_epi8(a,b)	comparaison octets signés. Si octet (a)> octet(b) octet résultat = FF_H sinon 00_H



PUNPCKLBW (B2HL)



PUNPCKHBW (B2HH)