Corrigé Examen Juin 07 - Architectures Avancées

3H – Tous documents autorisés

OPTIMISATION DE BOUCLES

On utilise le processeur superscalaire défini dans l'annexe 1
Soit la boucle suivante :

```
float X[4096], Y[2048];
for (i=0 ; i<2048 ; i++)
      Y[i] = X[i+2048] – X[i] ;
```

On supposera que l'adresse de X[0] est initialement dans R1, que l'adresse de Y[0] est initialement dans R2. R3 contient initialement le nombre d'itérations de la boucle.

Question 1 : Quel est en nombre de cycles, le temps d'exécution par itération de la boucle originale sans et avec utilisation des instructions SIMD. ?

Sans SIMD : 6 cycles/itération
Avec SIMD : 7/4 = 1,75 cycles/itération

Sans déroulage, sans SIMD

	E0	E1	FA	FM
1 Boucle	LF F1, 0(R1)	LF F2, 2048(R1)		
2	ADDI R1,R1,4	ADDI R2,R2,4		
3	ADDI R3,R3,-1		FSUB F1,F1,F2	
4				
5				
6	SF -4(R2)	BNE R3, Boucle		

Sans déroulage, avec SIMD

	E0	E1	FA	FM
1 Boucle	PLF S1, 0(R1)	ADDI R3,R3,-4		
2	PLF S2, 2048(R1)	ADDI R1,R1,16		
3	ADDI R2,R2,16			
4			PFSUB S1,S1,S2	
5				
6				
7	PSF -16(R1)	BNE R3, Boucle		

Question 2

Donner par itération de la boucle initiale
- le nombre de cycles sans instructions SIMD avec un déroulage d'ordre 4
- le nombre de cycles avec instructions SIMD avec un déroulage d'ordre 4

Avec déroulage, sans SIMD : 9/4 = 2,25 cycles/itération
Avec déroulage, avec SIMD : 13/ 16 = 0,8125 cycles/itération

Avec déroulage, sans SIMD

	E0	E1	FA	FM
1 Boucle	LF F1, 0(R1)	LF F2, 2048(R1)		
2	LF F3, 4(R1)	LF F4, 2052(R1)		
3	LF F5, 8(R1)	LF F6, 2056(R1)	FSUB F1,F1,F2	
4	LF F7, 12(R1)	LF F8, 2060(R1)	FSUB F3,F3,F4	
5	ADDI R1,R1,16	ADDI R2,R2,16	FSUB F5,F5,F6	
6	SF F1, -16(R2)	ADDI R3,R3,-4	FSUB F7,F7,F8	
7	SF F3, -12(R2)			
8	SF F5, -8(R2)			
9	SF F7, -4(R2)	BNE R3, Boucle		

Avec déroulage, SIMD

	E0	E1	FA	FM
1 Boucle	PLF S1, 0(R1)	ADDI R3,R3,-16		
2	PLF S2, 2048(R1)	ADDI R2,R2,64		
3	PLF S3, 4(R1)			
4	PLF S4, 2064(R1)		PFSUB S1,S1,S2	
5	PLF S5, 32(R1)			
6	PLF S6, 2080(R1)		PFSUB S3,S3,S4	
7	PLF S7, 48(R1)			
8	PLF S8, 2096(R1)		PFSUB S5,S5,S6	
9	PSF S1, -64(R2)	ADDI R1,R1,64		
10	PSF S3, -48(R2)		PFSUB S7,S7,S8	
11	PSF S5, -32(R2)			
12				
13	PSF S7, -16(R2)	BNE R3, Boucle		

Résumé :
Sans SIMD : 6 cycles/itération
Avec SIMD : 7/4 = 1,75 cycles/itération
Avec déroulage, sans SIMD : 9/4 = 2,25 cycles/itération
Avec déroulage, avec SIMD : 13/ 16 = 0,8125 cycles/itération

Question 3 : Pour la boucle ci-dessous, quelle est la condition pour pouvoir utiliser les instructions SIMD ?

```
float X[2*N], Y[N];
for (i=0 ; i<N ; i++)
      Y[i] = X[i+N] - X[i] ;
```

Il faut que N >=4 ;

PREDICTEURS DE BRANCHEMENT

Soit le programme C suivant :

```
int a=0, b=0, n=0, p=0;
main()
{
int i;
for (i=0; i<24; i++){
      a = (a+1)%3
      b= (b+1)%4
```

```
      if (a>=b)
            n++;
            else p++;}
}
```

On considère le branchement conditionnel correspondant au if(a>=b). Le branchement est pris (P) si (a<b) et non pris (NP) autrement.
On associe un prédicteur à ce branchement. On utilise soit une prédiction toujours pris, soit un prédicteur 1 bit, soit un compteur 2 bits. Le compteur 2 bits a 4 états : fortement non pris (FNP), faiblement non pris (fNP), faiblement pris (fP) et fortement pris (FP) auxquels on peut associer les valeurs 0, 1, 2 et 3.
Dans tous les cas, le prédicteur 1 bit est initialisé à NP et le prédicteur 2 bits à FNP.

Question 4 : Pour les 24 itérations de la boucle, quel est le nombre de prédictions correctes dans les cas suivants :

a) on utilise la prédiction toujours pris
b) on utilise un prédicteur 1 bit par branchement
c) on utilise un prédicteur 2 bits par branchement

a	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
b	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
B	N	N	N	P	N	N	P	P	N	P	P	P	N	N	N	P	N	N	P	P	N	P	P	P
a)	**N**	**N**	**N**	N	P	**N**	N	**P**	P	N	**P**	**P**	P	**N**	**N**	N	P	**N**	N	**P**	P	N	**P**	**P**
b)	**N**	**N**	**N**	N	**n**	**N**	N	n	p	n	**p**	**P**	P	p	**n**	N	**n**	**N**	N	n	p	n	**p**	**P**

Toujours pris : 12/24
Prédicteur 1 bit : 13/24
Prédicteur 2 bits : 12/24

CACHES

Un processeur utilise un cache de données de 8 Ko, avec des blocs de 16 octets, à correspondance directe. Le cache utilise la réécriture avec écriture allouée (il y a des défauts de cache en écriture). Le processeur a des adresses sur 32 bits.
On considère l'extrait de programme C suivant, pour lequel les tableaux X et Y sont rangés successivement en mémoire à partir de l'adresse 1000 0000_H (adresse de X[0].)

```
float X[4096], Y[2048];
for (i=0 ; i<2048 ; i++)
      Y[i] = X[i+2048] – X[i] ;
```

Avec la correspondance directe, ce programme a 3 défauts de cache par itération.

Question 5 : Utiliser une technique logicielle (sans changer la nature du cache) pour réduire le nombre de défauts de cache par itération et donner le nombre de défauts de cache par itération obtenue par cette méthode.

Les blocs ont 16 octets, soit 4 floats.
Avec un déroulage de boucle d'ordre 4 et en accédant d'abord à X[i], X[i+1], X[i+2], X[i+3] avant d'accéder à X[i+2048], X[i+2049], X[i+2050], X[i+2051], alors il y a 3 défauts de cache pour 4 itérations (0.75 défaut par itération) au lieu de 3 défauts.

PIPELINE LOGICIEL AVEC TMS 320C62

Le code assembleur TMS320C62 ci-dessous donne l'itération du pipeline logiciel pour un programme C.

```
LOOP:
          LDW  .D1  *A4++,A2       ; load ai and ai+1
||        ADD  .L1  A6,A7,A7       ; sum0+=(ai*ai)
||        ADD  .L2  B6,B7,B7       ; sum1+=(ai+1*ai+1)
||[A1]    B    .S2  LOOP           ; branch to loop
||        MPY  .M1X A2,A2,A6       ; ai*ai
||        MPYH .M2X A2,A2,B6       ; ai+1*ai+1
||[A1]    SUB  .S1  A1,1,A1   ; decrement loop counter

          ADD  .L1X A7,B7,A4       ; sum=sum0+sum1
```

Question 6 : 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]*X[i];
      tmp2 = X[i+1]*X[i+1];}
S=S1+S2 ;
```

Question 7 : Quel est l'intervalle inter-itération (II) ? Justifier la valeur.

L'intervalle inter-itération est de 1. Les 7 instructions se répartissent sur 7 unités fonctionnelles (D1, L1,L2, S1, S2, M1, M2).

SIMD IA-32

Le filtre conservatif 3 x 3 conserve la valeur d'un pixel si celle-ci est comprise entre le max et le min des huit pixels voisins. Sinon, il remplace la valeur du pixel par le max (resp. le min) des huit pixels voisins (les pixels sont des octets non signés)**.**

Question 8 : En utilisant les intrinsics du jeu d'instructions SIMD IA-32, écrire la version SIMD du filtrage conservatif 3x3 sur une image 128 x 128 en niveaux de gris

On pourra utiliser les « define » vus en cours et en TP.
On appellera X[128][128] l'image source et Y[128][128] l'image destination.

```
#define decg(va,vb)
_mm_or_si128(_mm_srli_si128(va,1),_mm_slli_si128(vb,15))
#define decd(va,vb)
_mm_or_si128(_mm_slli_si128(va,1),_mm_srli_si128(vb,15))

void ConservatifSIMD(byte **X, long i0, long i1, long j0, long j1,
byte **Y){
 __m128i **XS,**YS;
XS=X; YS=Y;
__m128i aimjm,aimj,aimjp,aijm,aij,aijp,aipjm,aipj,aipjp,m,res,mins,
      maxs;
int i, j;
for(i = i0+1; i <= i1-1; i++)
for(j = j0+1; j <= (j1-1)/16; j++){
// aij= _mm_load_si128(&XS[i][j]);
aimjm=decd(_mm_load_si128(&XS[i-1][j]),_mm_load_si128(&XS[i-1][j-1]));
aimj=_mm_load_si128(&XS[i-1][j]);
aimjp=decg(_mm_load_si128(&XS[i-1][j]),_mm_load_si128(&XS[i-1][j+1]));
aijm=decd(_mm_load_si128(&XS[i][j]),_mm_load_si128(&XS[i][j-1]));
aij=_mm_load_si128(&XS[i][j]);

aijp=decg(_mm_load_si128(&XS[i][j]),_mm_load_si128(&XS[i][j+1]));
aipjm=decd(_mm_load_si128(&XS[i+1][j]),_mm_load_si128(&XS[i+1][j-1]));
aipj=_mm_load_si128(&XS[i+1][j]);
aipjp=decg(_mm_load_si128(&XS[i+1][j]),_mm_load_si128(&XS[i+1][j+1]));
```

```
mins=minbu(aimjm,aimj);
mins=minbu(aimjp,mins);
mins=minbu(aijm,mins);
mins=minbu(aijp,mins);
mins=minbu(aipjm,mins);
mins=minbu(aipj,mins);
mins=minbu(aipjp,mins);
maxs=maxbu(aimjm,aimj);
maxs=maxbu(aimjp,maxs);
maxs=maxbu(aijm,maxs);
maxs=maxbu(aijp,maxs);
maxs=maxbu(aipjm,maxs);
maxs=maxbu(aipj,maxs);
maxs=maxbu(aipjp,maxs);
res=maxbu(mins,minbu(aij,maxs));
_mm_store_si128(&YS[i][j],result);}}
```

INSTRUCTIONS SPECIALISEES POUR Nios II

Question 9 : Définir les instructions spécialisées SIMD nécessaires pour implanter le filtre conservatif (de la question 8) sur le processeur NIOS II. Quelle est l'accélération maximale que l'on peut espérer ?

Décalage gauche d'un octet de deux mots de 32 bits va et vb
Décalage droit d'un octet de deux mots de 32 bits va et vb
Min 4 x 8 sur octets non signés
Max 4 x 8 sur octets non signés
L'accélération maximale possible est 4.

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
- 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.

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. Les chargements ne peuvent pas passer devant les rangements en attente.

JEU D'INSTRUCTIONS (extrait)

LF	LF Fi, dép.(Ra)	E0 ou E1	Fi ← M (Ra + dépl.16 bits avec ES)
SF	SF Fi, dép.(Ra)	E0	Fi -> M (Ra + dépl.16 bits avec ES)
ADD	ADD Rd,Ra, Rb	E0 ou E1	Rd ←Ra + Rb
ADDI	ADDI Rd, Ra, IMM	E0 ou E1	Rd ←Ra + IMM-16 bits avec ES
SUB	SUB Rd,Ra, Rb	E0 ou E1	Rd ←Ra - Rb
FADD	FADD Fd, Fa, Fb	FA	Fd ←Fa + Fb
FSUB	FSUB Fd, Fa, Fb	FA	Fd ← Fa - Fb
FMAX	FMAX Fd, Fa, Fb	FA	Fd ← Max(Fa, Fb)
FMIN	FMIN Fd, Fa, Fb	FA	Fd ← Min(Fa, Fb)
FMUL	FMUL Fd, Fa, Fb	FM	Fd ←Fa x Fb
FDIV	FDIV Fd, Fa, Fb	FA	Fd ← Fa/Fb
BEQ	BEQ Ri, dépl	E1	si Ri=0 alors CP ← NCP + depl
BNE	BNE Ri, dépl	E1	si Ri≠0 alors CP ← NCP + depl

Table 1 : instructions disponibles

La Table 2 donne la latence entre une instruction source et une instruction destination, dans le cas de dépendances de données. La valeur 1 est le cas où les deux instructions peuvent se succéder normalement, d'un cycle i au cycle i+1.

Latences	***Source***	UAL	LF/SF (données)	FADD/ FSUBF MAX/F MIN	FMUL	FDIV	FSQRT
Destination							
UAL		1	2				
LF/ST (adresses)		1	3				
SF (données)		1	2	3	4	20 (NP)	20 (NP)
Opération flottante			2	3	4	20 (NP)	20 (NP)

Table 2 : latences

Le processeur a également 32 registres de 128 bits S0 à S7 pouvant contenir chacun 4 flottants simple précision et les instructions SIMD données dans la Table 3. Cette table donne également les latences des instructions SIMD et le pipeline utilisé dans le cas superscalaire.

PLF	PLF Si, dép.(Ra)	2	E0	Charge quatre flottants simple précision à partir de l'adresse (Ra + dépl.16 bits avec ES).
PSF	PSF Si, dép.(Ra)	2	E0	Range en mémoire à partir de l'adresse (Ra + dépl.16 bits avec ES) quatre flottants simple précision
PFADD	PFADD Sd,Sa, Sb	3	FA	Sd ← Sa + Sb sur quatre « floats »

PFSUB	PFSUB Sd,Sa, Sb	3	FA	Sd ← Sa – Sb sur quatre « floats »
PFMUL	PFMUL Sd, Sa, Sb	4	FM	Sd ← Sa x Sb sur quatre « floats »
PFMULS	PMULS Sd, Sa, Fb	4	FM	SF ← Sa x Fb (chaque élément de Sa est multiplié par Fb et rangé dans l'élément correspondant de SF) sur quatre « floats »
PLB	PLB Si, dép.(Ra)	2	E0	Charge seize octets à partir de l'adresse (Ra + dépl.16 bits avec ES).
PSB	PSB Si, dép.(Ra)	2	E0	Range en mémoire à partir de l'adresse (Ra + dépl.16 bits avec ES) seize octets
PFMAX	PMAX Sd,Sa, Sb	1	FA	Sd ← Sa max Sb : maximum sur 4 floats
PFMIN	PMIN Sd,Sa, Sb	1	FA	Sd ← Sa min Sb : minimum sur 4 floats

Table 3 : Instructions SIMD

ANNEXE 2 : Instructions SIMD IA-32 utilisables

CVTDQ2PS	_mm_cvtepi32_ps (a)	Convertit 4 entiers 32 bits signés en 32 bits flottants
MULPS	_mm_mul_ps(a,b)	Quatre multiplications flottantes 32 bits
DIVPS	_mm_div_ps (a,b)	Quatre divisions flottantes 32 bits
MOVDQA	_mm_load_si128 (*p)	Chargement aligné de 128 bits
MOVDQA	_mm_store_si128 (*p,a)	Rangement aligné de 128 bits
MULPS	_mm_mul_ps (a, b)	Quatre multiplications flottantes 32 bits
PACKUSWB	_mm_packs_epu16 (m1, m2)	Compacte avec saturation shorts en octets non signés
PMAXUB	_mm_max_epu8 (a, b)	Max des octets non signés source et destination
PMINUB	_mm_min_epu8 (a, b)	Min des octets non signés source et destination
POR	_mm_or_si128 (a, b)	Ou logique parallèle
PSRLDQ	_mm_srli_si128 (a, imm)	Décalage logique gauche de « imm » octets
PSSLDQ	_mm_slli_si128 (a, imm)	Décalage logique droite de « imm » octets
PUNPCKHBW	_mm_unpacklo_epi8 (m1, m2)	Entrelace les octets (haut) de la destination et la source
PUNPCKHWD	_mm_unpacklo_epi16(m1, m2)	Entrelace les shorts (haut) de la destination et la source
PUNPCKLBW	_mm_unpacklo_epi8 (m1, m2)	Entrelace les octets (bas) de la destination et la source
PUNPCKLWD	_mm_unpacklo_epi16 (m1, m2)	Entrelace les shorts (bas) de la destination et la source
PXOR	_mm_xor_si128 (a, b)	Ou exclusif parallèle
	_mm_set_ps1 (float w)	Quatre fois le flottant 32 bits w dans un mot de 128 bits