Examen Juin 10 - 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 :

```
float X[128], Y[128], Z[128] ;
for (i = 0 ; i<128 ; i++)
      Z[i]= (X[i]+Y[i])*0.5 ; /*optimisation par rapport à
                          la division par 2.0*/
```

On supposera que l'adresse de X[0] est initialement dans R1, que l'adresse de Y[0] est initialement dans R2 et que l'adresse de Z[0] est initialement dans R3. R4 contient initialement le nombre d'itérations de la boucle. Lorsqu'on rentre dans la boucle, F0 contient 0.5.

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. ?

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

PIPELINE LOGICIEL AVEC TMS 320C62

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

Initialisation

```
            MVK .S1 100,A1
      ||    MVK .S2 0xFFFE, B1
            SHL .S2 B1,16,B1  //Décalage arithmétique gauche
```

Prologue // non donné

```
Pipeline logiciel

LOOP : LDW .D1 *A4++, A2
      || LDW .D2 *B4++, B2
      || SHR   .S1  A3,1,A3       // Décalage droite
      || [A1] B.S2  LOOP

         ADD2 A2,B2, A3 // ADD2 est l'addition SIMD de mots de 16 bits
      || STW.D1 *A5++, A3
      ||[A1] SUB .S2 A1,2,A1
```

Question 3 : Donner le code C correspondant au code assembleur.

Question 4 : Quel est l'intervalle inter-itération (II) ? Justifier la valeur.

Question 5 : Quel est le nombre de cycles par itération de la boucle initiale ?

SIMD IA-32

Les filtres de Robert sont des filtres (2,2) qui ont la définition suivante :
Gx (i,j) = I(i+1, j) – I(i,j)
Gy(i,j) = I(i,j+1) – (i,j)
G(i,j) = abs (Gx) + abs (Gy)

Question 6 : Donner la version scalaire optimisée du code C pour calculer l'image gradient G à partir d'une image I 128 x 128 en niveau de gris.

Question 7 : En utilisant les intrinsics du jeu d'instructions SIMD IA-32, écrire la version SIMD du code de la question précédente.

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

INSTRUCTIONS SPECIALISEES POUR NIOS II

Question 8 : Donner le code VHDL pour ajouter au jeu d'instructions NIOS les instructions spécialisées suivantes

- addition saturée de deux entiers non signés (ADDSU)
- addition saturée SIMD de deux fois deux mots de 16 bits (ADDHSU)

On donnera l'entité commune aux deux instructions, et l'architecture de chacune des 2 instructions.

PROGRAMMATION OPENMP

On reprend le code C de la question 1

```
float X[128], Y[128], Z[128] ;
for (i = 0 ; i<128 ; i++)
      Z[i]= (X[i]+Y[i])*0.5 ;
```

Question 9 : Donner une version OpenMP de ce programme pour 4 processeurs qui minimise les défauts de cache.

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*** / ***Destination***		UAL	LF/SF (données)	FADD/ FSUBF MAX/F MIN	FMUL	FDIV	FSQRT
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

	_mm_set_ps1 (float w)	Quatre fois le flottant 32 bits w dans un mot de 128 bits
ADDUS	_mm_adds_epu8(a,b)	Addition saturée de 16 octets non signés
CVTDQ2PS	_mm_cvtepi32_ps (a)	Convertit 4 entiers 32 bits signés en 32 bits flottants
DIVPS	_mm_div_ps (a,b)	Quatre divisions flottantes 32 bits
MAXUB	_mm_max_epu8(a,b)	Maximum de 16 octets non signés
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
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
SUBUS	_mm_subs_epu8 (a, b)	Soustraction saturée de 16 octets non signés