

Examen Décembre 2011

Architectures Avancées

3H – Tous documents autorisés

OPTIMISATION 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
```

Question 1) Donner le code C correspondant aux programme P1 et P2

Les latences des instructions flottantes sont

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

Question 2) Quel le temps d'exécution, en nombre de cycles d'horloge par itération de la boucle de P1 optimisée sur le processeur superscalaire

Question 3) Quel est le temps d'exécution de la boucle de P2 optimisée sur le processeur superscalaire en nombre de cycles par itération de la boucle initiale ?

SIMD IA-32 (première partie)

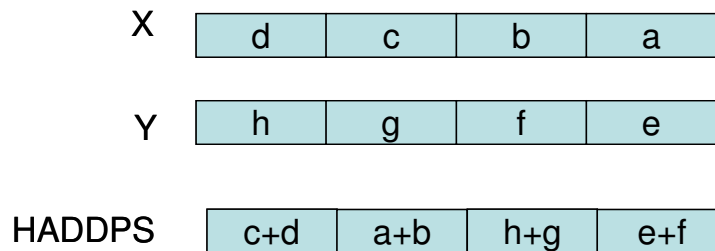
Soit le programme suivant, qui utilise les intrinsics IA-32 fournis en annexe

```
_m128 XS[1025] YS[1025] ,A, B, C ; // mots de 128 bits (4 floats)
for (i=0 ; i<1024 ; i++) {
    A= lf4 (&XS[i]) ;
    B= lf4u (&XS[i] +4) ;
    C= lf4u (&XS[i] + 8) ;
    A = addps (A,B) ;
    A = addps (A,C) ;
    sf4u (&YS[i] +4 , A) ;
}
```

Question 4) Donner le code C scalaire correspondant au code SIMD

SIMD IA 32 (Deuxième partie)

L'effet de l'instruction HADDPS sur des mots de 4 floats est défini par la figure suivante :



Question 5 : Comment peut on effectuer l'addition horizontale (d+c+b+a) à l'aide de l'instruction HADDPS ?

Question 6 : En utilisant les intrinsics définis en annexe, écrire la version SIMD du programme C suivant.

```
float A[1024][1024], Y[1024], X[1024], SF ;  
  
for (i=0 ; i<1024 ; i++) {  
    S=0.0 ;  
    for (j=0 ; j<1024 ; j++)  
        SF+= A[i][j] * X[j];  
    Y[i] = SF; }
```

On utilisera les variables

`_mm128 **AS, *X, *Y, SS;` // sur des mots de 4 floats.

INSTRUCTIONS SPECIALISEES POUR NIOS II

Question 7 : Définir l'instruction spécialisée NIOS II correspondant au code VHDL ci-dessous :

```
use IEEE.std_logic_unsigned.all;  
  
entity XX is  
  
    port  
    (  
        dataa :    in    std_logic_vector(31 downto 0);  
        datab :   in    std_logic_vector(31 downto 0);  
        result:    out   std_logic_vector(31 downto 0)  
    );  
end XX;  
  
architecture comp of XX is  
    signal tmp, a, b, c, d, e, f, g, h : std_logic_vector (15 downto 0);  
    signal ae, bf, cg, dh: std_logic_vector (31 downto 0);  
begin  
    a <= "00000000"& dataa (7 downto 0);  
    b <= "00000000"& dataa(15 downto 8);  
    c <= "00000000"&dataa(23 downto 16);  
    d <= "00000000"&dataa(31 downto 24);  
    e <= "00000000"& datab (7 downto 0);  
    f <= "00000000"&datab(15 downto 8);  
    g <= "00000000"&datab(23 downto 16);  
    h <= "00000000"&datab(31 downto 24);  
    ae <= a * e;  
    bf <= b * f;  
    cg <= c * g;
```

M1 Informatique
 Architectures avancées
 dh <= d * h;
 result <= ae + bf + cg + dh ;
 end comp; -- end of architecture

D. Etiemble
 Université Paris Sud

Question 8 : Donner le code VHDL (entité + architecture) pour ajouter au jeu d'instructions NIOS l'instruction Addition horizontale ADDHUB (octets non signés) et résultat sur 32 bits

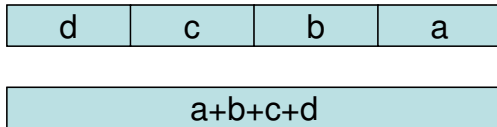


Figure 1 : Instruction ADDHUB

PIPELINE LOGICIEL TMS 320C6x

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
||      LDW    .D2    *B4++,B2           ; load bi and bi+1
|| [A0]  ADD    .L1    A6,A7,A7         ; sum0+=(ai*bi)
|| [B0]  ADD    .L2    B6,B7,B7         ; sum1+=(ai+1*bi+1)
|| [A1]  B      .S2    LOOP             ; branch to loop

        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

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

On rappelle le délai et la latence des instructions entières

Instruction Type	Delay Slots
NOP (no operation)	0
Store	0
Single cycle	0
Multiply (16 × 16)	1
Load	4
Branch	5

Tableau 1 : Délai des instructions entières (latence = 1+délai)

Question 9) Donner cycle par cycle le prologue correspondant au pipeline logiciel.

LOI D'AMDAHL

Un programme séquentiel a 5% de son temps d'exécution qui ne peut être parallélisé. On veut l'accélérer à taille constante.

Question 10) Quel est le nombre de processeurs nécessaire pour obtenir une accélération de 8 ? Quelle est alors l'efficacité parallèle (accélération /nombre de processeurs) ?

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 (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. L'ordonnancement est statique. Les chargements ne peuvent pas passer devant les rangements en attente.

JEU D'INSTRUCTIONS (extrait)

LF	LF Fi, dép.(Ra)	2	E0 ou E1	$F_i \leftarrow M(Ra + ES \text{dépl.16 bits})$
SF	SF Fi, dép.(Ra)		E0	$F_i \rightarrow M(Ra + ES \text{dépl.16 bits})$
ADD	ADD Rd, Ra, Rb	1	E0 ou E1	$Rd \leftarrow Ra + Rb$
ADDI	ADDI Rd, Ra, IMM	1	E0 ou E1	$Rd \leftarrow Ra + IMM - 16 \text{ bits avec ES}$
SUB	SUB Rd, Ra, Rb	1	E0 ou E1	$Rd \leftarrow Ra - Rb$
FADD	FADD Fd, Fa, Fb	4	FA	$Fd \leftarrow Fa + Fb$
FSUB	FSUB Fd, Fa, Fb	4	FA	$Fd \leftarrow Fa - Fb$
FMUL	FMUL Fd, Fa, Fb	4	FM	$Fd \leftarrow Fa \times Fb$
BEQ	BEQ Ri, dépl		E1	si $R_i=0$ alors $CP \leftarrow NCP + \text{depl}$
BNE	BNE Ri, dépl		E1	si $R_i \neq 0$ alors $CP \leftarrow NCP + \text{depl}$

Table 1 : instructions disponibles (avec latence et pipeline utilisé)

ANNEXE 2 : Instructions SIMD IA-32 utilisables

MULPS	<code>_mm_mul_ps(a, b)</code>	Quatre multiplications flottantes 32 bits
ADDPS	<code>_mm_add_ps(a, b)</code>	Quatre additions flottantes 32 bits
LF4	<code>_mm_load_m128(&x)</code>	Chargement aligné de 128 bits
LF4U	<code>_mm_loadu_m128(&x)</code>	Chargement non aligné de 128 bits
SF4	<code>_mm_store_m128(&x, a)</code>	Rangement aligné de 128 bits
SF4U	<code>_mm_storeu_m128(&x, a)</code>	Rangement non aligné de 128 bits
SF1	<code>_mm_store_ss(float * p, a)</code>	Rangement float de poids faible d'un mot de 128 bits.
SETF	<code>_mm_set_ps1(float w)</code>	Quatre fois le flottant 32 bits w dans un mot de 128 bits
HADDPS	<code>_mm_hadd_ps(a, b)</code>	Addition horizontale de 4 flottants