D. Etiemble
Université Paris Sud

Corrigé Examen Décembre 2010
Architectures Avancées
3H – Tous documents autorisés

OPTIMISATION DE BOUCLES

Soit le code C pour la multiplication matrice vecteur

Le programme assembleur (figure 1) travaille sur les flottants définis ci-dessous

```
#define N 128
float A[N][N], X[N] Y[N], sum;
```

//Au démarrage, R1 contient l'adresse de X[0], R2 contient l'adresse de Y[0] et R3 contient l'adresse de A[0][0];

```
      ADDI R4,R0,N
OUT : FSUB F0,F0,F0
      ADDI R5, R0, N
IN :  LF F1, (R1)
      LF F3, (R3)
      FMUL F1,F1,F3
      FADD F0,F0,F1
      ADDI R1,R1,4
      ADDI R3,R3, 4
      ADDI R5,R5, -1
      BNE R5, IN
      SF F0,(R2)
      ADDI R2,R2,4
      ADDI R4,R4,-1
      ADDI R1,R1, -4*N
      BNE R4, OUT
```

Figure 1 : Programme assembleur

Question 1 : Donner le code C correspondant au programme de la figure 1

```
float A[N][N], X[N] Y[N], sum;

for (i=0 ; i<N ; i++){
      sum=0.0 ;
      For (j=0 ; j<N ;j++)
            sum += A[i][j]* X[j] ;
            Y[i ] =sum }
```

On utilise le processeur superscalaire statique décrit en annexe.

Question 2 : Donner les dépendances de données dans la boucle interne. Donner les dépendances de données entre la boucle interne et la boucle externe.

Dans la boucle interne, les dépendances de données sont

- des loads F1 et F3 vers la multiplication
- de la multiplication vers l'addition (F1)
- Il n'y a pas de dépendance de données à partir de FADD, car F0 destination est utilisé comme source à l'itération suivante.

Dans la boucle externe, il y a une dépendance entre le dernier FADD et le SF (F0).

Question 3 : Après optimisation, donner le nombre de cycles par itération de la boucle interne (en supposant qu'il n'y a ni pénalité de branchement, ni défauts de cache).

	E0	E1	FA	FM
	ADDI R4,R0,N			
OUT	ADDI R5, R0, N		FSUB F0,F0,F0	
IN 1	LF F1,(R1)	LF F3,(R3)		
2	ADDI R1,R1,4	ADDI R3,R3, 4		
3	ADDI R5,R5, -1			FMUL F1,F1,F3
4				
5				
6				
7		BNE R5, IN	FADD F0,F0,F1	
OUT2	ADDI R2,R2,4	ADDI R4,R4,-1		
3		ADDI R1,R1,-4*N		
4				
5	SF F0,(R2)	BNE R4,IN		

Boucle interne : 7 cycles par itération

Question 4 : Après optimisation, donner le temps d'exécution total du programme (sans oublier l'instruction initiale, et toujours sans pénalité de branchement ou défauts de cache).

1 itération de la boucle externe = 5 + 128 * 7 = 901 cycles
Total : 1 + 128 *901 = 115 329 cycles

Question 5 : En supposant que l'on utilise pour chacun des branchements des prédicteurs 1 bit initialisé à « pris », quel est le nombre total de mauvaises prédictions ?

Prédicteur branchement IN : 2 mauvaises prédiction/boucle externe sauf première (1)
Prédicteur branchement OUT : 1 mauvaise prédiction (sortie)
Total : 128*2+1-1 = 256

Question 6 : Quel serait le temps total d'exécution avec un déroulage d'ordre 4 de la boucle interne ?

	E0	E1	FA	FM
	ADDI R4,R0,N			
OUT	ADDI R5, R0, N		FSUB F0,F0,F0	
2			FSUB	
3			FSUB	
4			FSUB	
IN 1	LF F1,(R1)	LF F3,(R3)		
2	LF	LF		
3	LF	LF		FMUL F1,F1,F3
4	LF	LF		FMUL
5	ADDI R1,R1,16	ADDI R3,R3, 16		FMUL
6	ADDI R5,R5, -4			FMUL
7			FADD F0,F0,F1	
8			FADD	
9			FADD	
10		BNE R5, IN	FADD	
OUT5	SF F0,(R2)	ADDI R4,R4,-1		
6	SF	ADDI R2,R2,4		
7	SF			
8	SF	BNE R4,IN		

Boucle interne : 10 cycles pour 4 itérations soit 2,5 cycles/itération
Boucle externe : 8 cycles
Temps total : 1 + 128 *(8 + 128*2,5) = 41985 cycles

PIPELINE LOGICIEL AVEC TMS 320C64

Soit le code

```
  MVKL .S1 01010101h, A5
  MVKH .S1 01010101h, A5
||MVK .S2 64, A1
||ZERO .D1 A7
// le prologue n'est pas fourni.
Loop :
   LDW .D1 *A4++, A3
|| LDW .D2 *B4++, B3
|| SUBABS4 .L2X A3,B3,B6
|| DOTPU4 .M1X A5,B6,A2
|| ADD .L1 A7,A2,A7
|| [A1] .S1 A1,1,A1
|| [A1] .S2 B Loop
```

Le code travaille sur deux tableaux X[128] et Y[128] d'octets non signés. A4 est le pointeur vers X[i] et B4 est le pointeur vers Y[i].

Le résultat des deux instructions MVKL et MVKH est de mettre 0x 01010101 dans le registre A5.
L'instruction SUBABS4 est une instruction SIMD qui calcule la valeur absolue de la différence des octets (non signés) sur les 4 * 8 bits des registres.
L'instruction DOTPU4 est une instruction SIMD qui calcule le produit scalaire sur les octets non signés des 4*8bits des registres et délivre un résultat sur 32 bits.
DOTPU4 (a, b) = a3.b3 + a2.b2+a1.b1.+ a0b0 (résultat sur 32 bits)

Question 7) Donner le code C correspondant à la version scalaire initiale du programme.

Unsigned char X[256], Y[256] ;
Int i, S=0;
For (i=0; i<256; i++)
S+= abs (X[i] –Y[i];

Question 8) Quel est le nombre de cycles par itération de la boucle scalaire initiale ?

1 cycle de pipeline logiciel pour 4 itérations, soit 0,25 cycles /itération.

SIMD IA-32

Première partie

Soit la fonction PMV4 utilisant des instructions SIMD (intrinsics)

```
void PMV4 ( float **A, float *X, float *Y, int N)
{
_m128 **AS, *X, *Y, SS, X1,X2;
int i,j;
AS=A; XS=X;YS=Y;
For (i=0; i<N; i++){
      SS=PXOR (SS,SS);
```

```
      For (j=0; j<N/4; j++) {
            X1= LF4 (&X[j]];
            X2 = LF4 (&A[i][j]);
            X1=MULPS (X1,X2) ;
            SS=ADDPS (SS,X1); }
      SF4 (&Y[I ], SS); }
}
```

Question 9) Donner le code C correspondant au code SIMD

```
float A[N][N], X[N] Y[N], sum;

for (i=0 ; i<N ; i++){
      sum=0.0 ;
      For (j=0 ; j<N ;j++){
            sum += A[i][j]* X[j] ;
            Y[i ] =sum }}
```

Deuxième partie

Le jeu d'instruction IA-32 contient des instructions SIMD de conversion d'entiers 32 bits (int) en flottants 32 bits simple précision (float) : CVTDQ2PS définie W2F
On considère des variables 128 bits (_m128i) v0, v1, v2, v3, v4, v5, v6, v7 pour les entiers et des variables 128 bits (_m128) f0, f1, f2 et f3 pour les flottants simple précision.

Question 10) En supposant que la variable v0 contient 16 octets (unsigned char) correspondant à des pixels (niveaux de gris), donner la suite des instructions SIMD (intrinsics) nécessaires pour convertir les 16 données 8 bits en 16 données de type float dans les variables f0 à f3. Quelle opération faut-il alors effectuer pour avoir des données flottantes comprises entre 0 et 1 ? Quel est le nombre total d'instructions nécessaires pour faire la conversion ?

```
zero = pxor (zero,zero);
v1= b2hl (v0,zero) ; // Huit pixels bas de v0 étendus sur 16 bits
v2 = b2hh(v0,zero) ; // Huit pixels hauts de v0 étendus sur 16 bits
v4 = h2wl (v1, zero) ; // Quatre pixels bas de v1 étendus sur 32 bits
v5 = h2wh (v1, zero) ; // Quatre pixels haut de v1 étendus sur 32 bits
v6 = h2wl (v2, zero) ;// Quatre pixels bas de v2 étendus sur 32 bits
v7 = h2wh (v2, zero) ; // Quatre pixels haut de v2 étendus sur 32 bits
f0= w2f(v4) ;          // int converti en float
f1= w2f (v5) ;         // int converti en float
f2= w2f (v6)   ;       // int converti en float
f3= w2f (v7)   ;       // int converti en float
w = 1/255.0   ;                        // constante 1/255.0
c = setf(w)     ;              // constante 4 * (1/255.0)
f0 = mulps (f0, c) ;           // valeurs des pixels entre 0.0 et 1.0
f1 = mulps (f1, c) ;           // valeurs des pixels entre 0.0 et 1.0
f2 = mulps (f2, c) ;           // valeurs des pixels entre 0.0 et 1.0
f3 = mulps (f3, c) ;           // valeurs des pixels entre 0.0 et 1.0
```

INSTRUCTIONS SPECIALISEES POUR NIOS II

Question 11 : Donner le code VHDL (entité + architecture)pour ajouter au jeu d'instructions NIOS l'instruction spécialisée suivante :

- UNPKLU4 : décompactage de deux octets bas non signés en mots de 16 bits

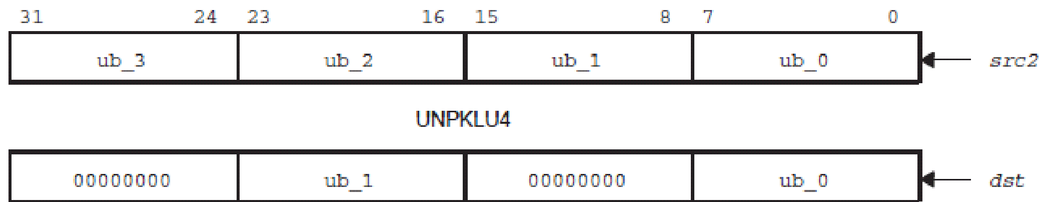

Figure 2 : Instruction UNPKLU

```
entity unpkl 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 unpkl;

architecture comp of unpkl is

begin
result (7 downto 0) <= dataa (7 downto 0);
result (15 downto 8)  = "00000000";
result (23 downto 16) <= dataa (15 downto 8);
result (31 downto 24) <= "00000000";
end comp; -- end of architecture
```

PROGRAMMATION OPENMP

Soit les tableaux X, Y et A définis ci-dessous
`float A[N][N], X[N] Y[N]`

Question 12 : Donner une version OpenMP pour 8 processeurs du programme calculant Y = A* X (produit matrice –vecteur).

```
float X[128], Y[128], Z[128] ;
omp_set_num_thread (8);
#pragma omp parallel private (j)
{int id;
id = omp_get_thread_num();
for (i =id*16 ; i<(id+1)*16 ; i++){
      sum=0.0 ;
      For (j=0 ; j<N ;j++){
            sum += A[i][j]* X[j] ;
            Y[i ] =sum }}
}
```

LOI D'AMDAHL

Un programme séquentiel a 10% de son temps d'exécution qui ne peut être parallélisé. On veut l'accélérer à taille constante.

Question 13) Quel est le nombre de processeurs nécessaire pour obtenir une accélération de 4 ? Quelle est alors l'efficacité parallèle (accélération /nombre de processeurs) ?

$$Acc = \frac{1}{0,1+\frac{0,9}{n}} = 4$$

0,1 + 0,9/n = 0,25
0,9/ n = 0,15
n/0,9 =1/0,15
n=0,9/0,15 = 6

Accélération parallèle = 4/6 = 66,66 %

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. 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)	2	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	1	E0 ou E1	Rd ←Ra + Rb
ADDI	ADDI Rd, Ra, IMM	1	E0 ou E1	Rd ←Ra + IMM-16 bits avec ES
SUB	SUB Rd,Ra, Rb	1	E0 ou E1	Rd ←Ra - Rb
FADD	FADD Fd, Fa, Fb	4	FA	Fd ←Fa + Fb
FSUB	FSUB Fd, Fa, Fb	4	FA	Fd ← Fa - Fb
FMUL	FMUL Fd, Fa, Fb	4	FM	Fd ←Fa x 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 (avec latence et pipeline utilisé)

ANNEXE 2 : Instructions SIMD IA-32 utilisables

W2F	_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
LF4	_mm_load_si128 (*p)	Chargement aligné de 128 bits
SF4	_mm_store_si128 (*p,a)	Rangement aligné de 128 bits
MULPS	_mm_mul_ps (a, b)	Quatre multiplications flottantes 32 bits
H2BS	_mm_packs_epu16 (m1, m2)	Compacte avec saturation shorts en octets non signés
B2HH	_mm_unpacklo_epi8 (m1, m2)	Entrelace les octets (haut) de la destination et la source
H2WH	_mm_unpacklo_epi16(m1, m2)	Entrelace les shorts (haut) de la destination et la source
B2HL	_mm_unpacklo_epi8 (m1, m2)	Entrelace les octets (bas) de la destination et la source
H2WL	_mm_unpacklo_epi16 (m1, m2)	Entrelace les shorts (bas) de la destination et la source
PXOR	_mm_xor_si128 (a, b)	Ou exclusif parallèle
SETF	_mm_set_ps1 (float w)	Quatre fois le flottant 32 bits w dans un mot de 128 bits