# Corrigé Examen Juin 10 - 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[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. ?**

Sans déroulage, sans SIMD

| | E0 | E1 | FA | FM |
|---|---|---|---|---|
| 1 Boucle | LF F1, 0(R1) | LF F2, 0(R2) | | |
| 2 | ADDI R1,R1,4 | ADDI R2,R2,4 | | |
| 3 | ADDI R3,R3,4 | ADDI R4,R4,-1 | FADD F1,F1,F2 | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | FMUL F1,F1,F0 |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | SF F1, -4(R3) | BNE R3, Boucle | | |

| | E0 | E1 | FA | FM |
|---|---|---|---|---|
| 1 Boucle | PLF S1, 0(R1) | ADDI R4,R4,-4 | | |
| 2 | PLF F2, 0(R2) | ADDI R3,R3,16 | | |
| 3 | ADDI R1,R1,16 | ADDI R2,R2,16 | | |
| 4 | | | PADD S1,S1,S2 | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | PMULS S1,S1,F0 |
| 8 | | | | |
| 9 | | | | |
| 10 | PSF F1, -16(R3) | BNE R4, 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

| | E0 | E1 | FA | FM |
|---|---|---|---|---|
| 1 Boucle | LF F1, 0(R1) | LF F2, 0(R2) | | |
| 2 | LF F3, 0(R1) | LF F4, 0(R2) | | |
| 3 | LF F5, 0(R1) | LF F6, 0(R2) | FADD F1,F1,F2 | |
| 4 | LF F7, 0(R1) | LF F8, 0(R2) | FADD F3,F3,F4 | |
| 5 | ADDI R1,R1,16 | ADDI R2,R2,16 | FADD F5,F5,F6 | |
| 6 | ADDI R3,R3,16 | ADDI R4,R4,-4 | FADD F7,F7,F8 | FMUL F1,F1,F0 |
| 7 | | | | FMUL F3,F3,F0 |
| 8 | | | | FMUL F5,F5,F0 |
| 9 | SF F1 -16(R3) | | | FMUL F7,F7,F0 |
| 10 | SF F3, -12(R3) | | | |
| 11 | SF F5, -8(R3) | | | |
| 12 | SF F7, -4(R3) | BNE R3, Boucle | | |

Avec déroulage, SIMD

| | E0 | E1 | FA | FM |
|---|---|---|---|---|
| 1 Boucle | PLF S1, 0(R1) | ADDI R3,R3,64 | | |
| 2 | PLF S2, 0(R2) | ADDI R1,R1,64 | | |
| 3 | PLF S3, -48(R1) | ADDI R2,R2,64 | | |
| 4 | PLF S4, -48(R2) | ADDI R4,R4,-16 | PADD S1,S1,22 | |
| 5 | PLF S5, -32(R1) | | | |
| 6 | PLF S6, -32(R2) | | PADD S3,S3,S4 | |
| 7 | PLF S7, -16(R1) | | | PMULS S1,S1,F0 |
| 8 | PLF S0, -16(R2) | | PADD S5,S5,S6 | |
| 9 | | | | PMULS S3,S3,F0 |
| 10 | PSF S1 -64(R3) | | PADD S7,S7,S0 | |
| 11 | | | | PMULS S5,S5,F0 |
| 12 | PSF S3 -48(R3) | | | |
| 13 | | | | PMULS S7,S7,F0 |
| 14 | PSF S5 -32(R3) | | | |
| 15 | | | | |
| 16 | PSF S7 -16(R3) | BNE R3, Boucle | | |

Résumé :
Sans déroulage sans SIMD : 9 cycles
Sans déroulage avec SIMD : 10/4 = 2,5 cycles
Avec déroulage sans SIMD : 12/4 = 3 cycles
Avec déroulage avec SIMD : 16/16 = 1 cycles

## 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
```

```
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.**

Short X[100], Y[100], Z[100]

For (i=0 ; i<100 ; i+=2){
Z[i] = (X[i] + Y[i] )/2 ;
Z[i+1] = (X[i+1] + Y[i+1] )/2 ;
}

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

II = 2 car il y a 3 instructions mémoire par itération.

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

1 cycle (2 cycles pour 2 itérations)

## 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.**

```
#define vabs(a) ((a>0)?a:(0-a)) ;
/* ------------------------------------------------------------- */
void G(byte **X, long i0, long i1, long j0, long j1, byte **Y)
/* ------------------------------------------------------------- */
{
  int i, j, tmp;
  int Gx, Gy;

  for(i=i0; i<=i1-1; i++) {

    for(j=j0; j<=j1-1; j++) {

            Gx = X[i+1][j]- X[i][j];
            Gy = X[i][j+1]- X[i][j];
            Gx = vabs(Gx); Gy = vabs(Gy);
            Y[i][j] = (Gx+Gy);
    }
  }
}
```

**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.

```
#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))
#define subus(va,vb) _mm_subs_epu8 (va, vb)
#define maxub(va,vb) _mm_max_epu8(va,vb)
#define addus(va,vb) _mm_adds_epu8(va,vb)

/* ------------------------------------------------------------------ */
void G_simd(byte **X, long i0, long i1, long j0, long j1, byte **Y)
/* ------------------------------------------------------------------ */
{
  int i, j;
  __m128i t1, t2, tij, tijp, tipj, GX1, GX2, GY1, GY2, GX, GY;
      __declspec( align(16) )  __m128i **XS, **YS;
  XS=X; YS=Y;

  for(i=i0+1; i<=(i1-1); i++) {

    for(j=j0+1; j<=(j1-1)>>4; j++) {
            tij=_mm_load_si128(&XS[i][j]); // XS[i][j]
            tipj=_mm_load_si128(&XS[i+1][j]); //XS[i+1][j]
            t2 =_mm_load_si128(&XS[i][j+1]);
            tijp= decg(tij,t2);
            GX1= subus (tipj,tij); GX2 = subus(tij,tipj);
            GY1= subus (tijp,tij); GY2 = subus(tij,tijp);
            GX= maxub(GX1,GX2) ; GY = maxub(GY1,GY2);
            GX=addus(GX,GY);
            _mm_store_si128(&YS[i][j],GX);
    }
  }
}
```

## 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.

```
entity commun_aux_deux_instructions 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 commun_aux_deux_instructions;

architecture comp of ADDSU is
signal tmp: std_logic_vector(31 downto 0);

begin
tmp (31 downto 0) <= dataa (31 downto 0)+ datab (31 downto 0)
result (31 downto 0) <= tmp (31 downto 0) when (tmp (31 downto 0)>
dataa (31 downto 0))else (others => '1');
```

```
end comp; -- end of architecture

architecture comp of ADDSS is
signal tmp: std_logic_vector(31 downto 0);

begin
tmp (31 downto 16) <= dataa (31 downto 16)+ datab (31 downto 16)
result (31 downto 16) <= tmp (31 downto 16) when (tmp (31 downto 16)>
dataa (31 downto 16))else (others => '1');
tmp (15 downto 0) <= dataa (15 downto 0)+ datab (15 downto 0)
result (15 downto 0) <= tmp (15 downto 0) when (tmp (15 downto 0)>
dataa (15 downto 0))else (others => '1');
end comp; -- end of architecture
```

## 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.**

```
float X[128], Y[128], Z[128] ;
omp_set_num_thread (4);
#pragma omp parallel
{int id;
id = omp_get_thread_num();
for (i =id*32 ; sum[id]=0.0; i<(id+1)*32 ; i++)
      Z[i]= (X[i]+Y[i])*0.5 ;
}
```

## 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

| | | |
|---|---|---|
| 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 |