Université Paris Sud D. Etiemble
M1 Informatique 2005-06

Examen Janvier 06 - Architectures Avancées

3H – Tous documents autorisés

PIPELINES

Un processeur a les pipelines suivants pour les trois types d'instructions*

- Instructions entières

  LI1 LI2 DI LR EX1 EX2 ER
- Instructions mémoire

  LI1 LI2 DI LR EX1 AM1 AM2 ER
- Instructions de multiplication-accumulation

  LI1 LI2 DI LR M1 M2 M3 ER

Les différents étages ont la signification suivante :

- LI1 et LI2 correspondent à l'accès au cache instructions
- DI : décodage
- LR : Lecture des opérandes dans le banc de registres
- EX1 : Calculs UAL pour les opérations arithmétiques et logiques et calculs d'adresse mémoire et de branchement. Si un branchement a été mal prédit, toutes les instructions dans les étages précédents du pipeline sont annulées et l'on réinitialise le pipeline à partir de la bonne adresse
- EX2 Cet étage choisit ce qui va être écrit dans le banc de registres dans le cycle ER
- AM1 et AM2 : accès au cache données
- M1, M2, M3 : multiplication entière pipelinée
- ER : Ecrire du résultat dans le banc de registres.

Tous les circuits d'anticipation (bypass) existent.

Question 1

a)Donner les latences entre instruction producteur et instruction consommateur en nombre de cycles.

NB : la valeur n signifie que deux instructions dépendantes peuvent se suivre aux cycles i et i+n, que ce soit pour un processeur scalaire ou superscalaire. Une valeur 0 signifie que les deux instructions peuvent démarrer au même cycle (pour un superscalaire)

	Instruction producteur	Instruction consommateur	Latence
a	UAL Ri, -, -	UAL -, Ri, -	
b	UAL Ri, -, -	ST Ri, ()	
c	UAL Ri, -, -	LW Rj, (Ri)	
d	LW Ri, ()	UAL -, Ri, -	
e	UAL Ri	MUL -,Ri,-	
f	LW Ri, ()	MUL -,Ri,-	
g	MUL Ri,	ST Ri, ()	
h	LW Ri, ()	ST Ri, ()	

b) Quelle est la pénalité de branchement conditionnel mal prédit ?

GRAPHIQUE ET SIMD

Soit le code suivant pour un filtre 3x3 (EEMBC High-Pass Gray Filter)

```
int i, j;
unsigned char X[size][size], X[size][size];
short temp;
 for(i=1; i<size-1; i++) {
   for(j=1; j<size1; j++) {
    tmp = (short) (F11* X[i-1][j-1]+F21* X[i-1][j]+F31* X[i-1][j+1]
                  + F12* X[i][j-1]+F22* X[i][j]+F32* X[i][j+1]
                   + F13* X[i+1][j-1]+F23* X[i+1][j]+F33* X[i+1][j+1];
        Y[i][j]= (unsigned char) (tmp>>8);
       }
```

Figure 1 : Filtre 3 x 3

Le coefficient F22 a la valeur 255 et tous les autres coefficients ont la valeur -28.

Question 2 :

Ecrire une version C scalaire optimisée du programme de la figure 1

Question 3 :

On utilise maintenant le processeur NIOS II pour lequel on veut définir des instructions SIMD 4x8 ou 2x16 spéciales.

a) donner la liste des instructions SIMD ajoutées, en précisant (informellement) leur action et en leur attribuant un mnémonique
b) Ecrire la version C SIMD optimisée utilisant les instructions définies en a)

PREDICTION DE BRANCHEMENTS

Dans cette partie, P signifie qu'un branchement est Pris et N qu'il est non pris. On a le choix entre

- un prédicteur statique
- un prédicteur dynamique 1 bit
- un prédicteur dynamique 2 bits

Question 4 : pour chacun des branchements suivants (considérés chacun séparement), proposer un type de prédicteur en justifiant. Si plusieurs prédicteurs conviennent, on choisira celui qui utilise le moins de matériel

a) Branchement B1,
P,P,P,P,P,*N,N,N,N,N,N,N,N*,P,P,P,P,P,P,P,P,P,*N,N,N,N,N*,P,P,P,P,P,P
b) Branchement B2
P,P,P,P,*N*,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P,*N,N,N,N,N*,P,*N,N,N,N,N*,P,*N,N,N,N,N*
c) Branchement B3
P,P,P,P,P,*N*,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P,P,*N*,P,P,P,P,P,P,P

OPTIMISATION DE BOUCLES

On utilise le processeur superscalaire défini dans l'annexe 1 et les programmes P1 et P2 suivants

P1	P2

float X[N], Y[N], Z[N] ;
int i ;
 	
for (i=0 ; i<N ; i++)
Z[i]= X[i] + Y[i] ;
	float X[N], Y[N], Z[N], a , un=1.0;
	int i ;
	
	for (i=0 ; i<N ; i++)
	Z[i]= X[i] /a + Y[i] ;

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.

Question 5 : On considère le programme P1

a) Quel est en nombre de cycles, le temps d'exécution par itération pour le programme P1 (optimisée, mais sans déroulage de boucle) ?
b) Quel est en nombre de cycles, le temps d'exécution par itération de la boucle initiale, avec un déroulage d'ordre 4 (4 itérations initiale par itération de boucle) ?

Question 6 : On considère le programme P2

a) Quel est en nombre de cycles, le temps d'exécution par itération pour le programme P2 (après optimisation, mais sans déroulage de boucle) ? Quel est le temps d'exécution total si N=100 ?
b) Peut-on utiliser le déroulage de boucle ? Si non, pourquoi ? Si oui, donner le temps d'exécution du programme pour N=100 avec un déroulage de boucle d'ordre 4.

Optimisations logicielles

On considère un processeur scalaire dont les instructions et les latences sont données dans les tables 1 et 2 .
Soit le programme suivant, dans lequel on suppose que R3 contient au départ l'adresse de X[0], R4 l'adresse de Y[0], R5 l'adresse de Z[0] et R2 contient N-2.
X[i], Y[i] et Z[i] sont des flottants simple précision

Question 7 : Que fait le programme suivant. Quel est son temps d'exécution si N=100 ? Quelle technique est employée ?

```
        LF F3, (R3)
        LF F4,(R4)
        FADD F5,F3,F4
        LF F3, 4(R3)
        LF F4, 4(R4)
        ADDI R3, R3,8
        ADDI R4, R4,8
L1 :    SF F5, (R5)
         FADD F5,F3,F4
        LF F3, 0(R3)
        LF F4, 0(R4)
        ADDI R2,R2,-1
        ADDI R3,R3,4
        ADDI R4,R4,4
        ADDI R5,R5,4
        BNEQ R2, L1
        SF F5, (R5)
        FADD F5,F3,F4
        SF F5, 4(R5)
```

Pipeline logiciel avec IA-64

Dans IA-64, les registres flottants f0 et f1 sont câblés : f0=0.0 et f1=1.0.
L'instruction fma a l'action suivante :

fr= fa,fb,fc correspond à fr = fa*fb+fc

Question 8 : Que font les boucles des programmes 1 et 2 ?

IA-64 – Programme 1	IA-64 – Programme 2
L1 : { (p16)ld4 r36=[r8],4 (p16)ld4 r37=[r3],4 nop.i} { (p17)st4 [r2]=r39,4 ;; (p16)lfetch.nt1 [r35] (p16)add r38=r36,r37 } { nop.m 0 (p16)add r32=12,r35 br.ctop.sptk .L1 ;; }	L2 : { (p16)ldfs f32=[r8],4 nop.i 0 nop.i 0} { (p16)ldfs f38=[r3],4 (p16)lfetch.nt1 [r35] nop.b 0 ;;} { (p23)stfs [r2]=f46,4 (p21)fma.s f44=f37,f1,f43 (p16)add r32=12,r35 } { nop.m 0 nop.m br.ctop.sptk .L2 ;; }

Figure 2 : Code IA-64

CACHES

Soit le code naïf de la transposition d'une matrice [N][N] :

```
void transpose (unsigned char **X, int N ,unsigned char **Y){
int i,j ;
 for(i=0; i<N; i++)
        for(j=0; j<N; j++)
                Y[j][i]=X[i][j];}
```

Figure 3 : code naïf de la transposition

On utilise un processeur Pentium 4 dont les caches ont les caractéristiques suivantes :

- Cache données L1 de 8 Ko, associatif 4 voies avec des blocs de 64 octets. L'écriture simultanée dans le cache L2 est utilisée lors des défauts de cache L1 en écriture
- Cache L2 de 512 Ko, associatif 8 voies avec des blocs de 128 octets. La réécriture est utilisée lors des défauts de cache L2 en écriture.

L'exécution de la transposition (code de la figure 3) donne les résultats suivants :

Lena512 : 14,2 CPP
Lena1024 : 153,1 CPP

Question 9: Est que l'augmentation spectaculaire du temps d'exécution lorsqu'on passe de N=512 à N=1024 provient de défauts de capacité ou de défauts de conflit ? (Justifier). Quelle expérience simple permettrait de vérifier ?

Annexe 1 : superscalaire à ordonnancement statique

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

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
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

Latences	***Source***	UAL	LF/SF (données)	FADD	FMUL	FDIV	FSQRT
Destination							
UAL		1	2				
LF/ST (adresses)		1					
SF (données)		1	2	3	5	20 (NP)	20 (NP)
Opération flottante			2	3	5	20 (NP)	20 (NP)

Table 2 : latences