M2R - NSI - Optimisation et compilation

14 février 2011

Barême indicatif: problème 1: 60%, problème 2: 60%, problème 3: 40%. Les questions n'ont pas forcément une réponse unique. Les analyses critiques seront appréciées. La note finale prendra en compte cet examen (coef 4) et le résumé d'article demandé par M. Bachir (coef 1).

1 Problème d'architecture

1.1 Exercice 1 −− Embarqué

Question 1. Décrivez les différentes contraintes à observer pendant le développement d'un processeur/produit pour l'embarqué. Décrivez aussi leurs interactions et comment peut-on essayer de les garantir.

Question 2. Positionnez les types de processeurs suivants par rapport aux contraintes définies dans la question précédente (justifiez brièvement) :

- processeur in-order (1 instruction par cycle)
- processeur in-order (2-4 instructions par cycle, super scalaire)
- processeur out-of-order (1 instruction par cycle)
- processeur out-of-order (super scalaire)
- processeur VLIW (in-order)
- CMP (2-4 processeurs in-order, 1 instruction par cycle)
- SMT (2-4 threads in-order, 1 instruction par cycle)

1.2 Exercice 2 − SMT

Vous devez considérer que les tableaux A, B et C ne se chevauchent pas en mémoire.

```
Code C
======
for (i=0; i<256; i++) {
  if (A[i] > 0)
    C[i] = A[i] * B[i];
}

Code assembleur
```

```
==============
; initial values
; r1 = &A[0]
; r2 = &B[0]
; r3 = &C[0]
; r4 = 0
loop:
1. l.s f1, 0(r1) ; f1 = A[i]
2. blez f1, skip ; if (A[i]<=0) goto skip
3. l.s f2, 0(r2) ; f2 = B[i]
4. mul.s f3, f1, f2 ; f3 = f1 * f2
5. s.s f3, 0(r3) ; C[i] = f3
skip:
6. addi r1, r1, 4 ; i++
7. addi r2, r2, 4
8. addi r3, r3, 4
9. addi r4, r4, 1
10. slti r5, r4, 256
11. bnez r5, loop ; if (i < 256) loop
```

Dans ce problème, nous allons analyser la performance du programme ci-dessus sur une architecture multithread. Voici un sommaire de l'architecture considérée :

- 1 instruction par cycle et pipeline in-order
- chaque instruction prend 3 cycles (Fetch, Decode, et Lecture de registre) avant son exécution.
- pour son exécution, chaque instruction prend :
    1. 1 cycle pour les opérations entières;
    2. 1 cycle pour la résolution de branchement;
    3. 4 cycles pour les opérations mémoire;
    4. 3 cycles pour les opérations avec flottants;
- après exécution chaque instruction prend 1 cycle pour écrire son résultat (pas de bypassing).

Une instance du programme tourne sur cette machine. La machine change de thread à chaque cycle avec une politique de scheduling round robin. Chacun des N threads exécute une instruction tous les N cycles. Nous allouons le code pour les threads de telle sorte que chaque thread s'exécute chaque N-ème itération du code C original (chaque thread incrémente la variable 'i' par N).

Question 1. Combien de threads sont nécessaires pour garantir qu'il n'y a pas de bulles dans le pipeline ?

Question 2.
(1) Pour une politique stricte de scheduling round robin avec 16 threads (ça peut être et ça peut ne pas être la réponse pour la question précédente) et supposant que 25% des valeurs dans le tableau A sont plus petites ou égales à zéro, combien d'instructions peut exécuter chaque thread en moyenne ?
(2) Pour le cas avec la meilleure distribution de valeurs inférieures ou égales à zéro dans le tableau A (encore supposant que 25% des valeurs sont $<= 0$), combien de temps (en cycles) cela prendra-t-il pour que tous les

threads finissent ? (Vous pouvez calculer le temps de finition à partir de l'insertion de la première instruction dans le pipeline jusqu'à l'insertion de la dernière instruction dans le pipeline.)
(3) Pour le cas avec la pire distribution de valeurs inférieures ou égales à zéro dans le tableau A (supposant toujours que 25% des valeurs sont $<= 0$), combien de temps (en cycles) cela prendra-t-il pour que tous les threads finissent ?

Question 3. Un utilisateur a du code multithread à faire tourner. Le code a une section séquentielle de 100.000 instructions et une section parall'ele (4 ways) de 100.000 instructions par thread qu'il veut faire tourner dans son processor SMT multithread 4-way. Pendant que la section séquentielle tourne il trouve que le processeur exécute 2 instructions par cycle (IPC). Et quand la section parallèle s'exécute l'IPC au total est de 3. Quel est le temps d'exécution du programme (en cycles) ?

Question 4. Comparez la performance que vous avez trouvée dans la question précédente avec la performance que l'utilisateur aurait observé sur un 2-processeur CMP (chip multiprocesseur) où chaque processeur tourne avec un IPC de 1.5.

2 Modèle polyédrique

Exercice 1.
On considère la boucle suivante:

```
for(i=0; i<n; i++)
  for(j=i; j<n-i; j++)
    S(i,j) = 3*S(i,j-i) + S(j,i) + j + 2*j;
```

1) Écrire l'espace d'itérations sous forme polyédrique (i.e. $Ax \leq b$).
Expliciter mathématiquement A et b, et écrire les 3 fonctions d'accès.
2) On pose $n = 2p$. Réécrire la boucle sur i en se limitant strictement à l'intervalle utile (*prendre en compte la variation de j*).
3) Calculer le volume de l'espace d'itération.
4) Peut-on directement intervertir les deux boucles sur i et j ? Pourquoi ?
5) En prenant $n = 5$ et $S = 0$ (matrice 5×5 initialisée à zéro), écrire le contenu de la matrice S à la fin des itérations.

Exercice 2.
On considère le calcul suivant à effectuer sur tous les points intérieurs d'une matrices $n \times n$

```
V(i,j)=f(V(i-1,j),V(i,j-1),V(i-1,j+1));
```

1) Écrire les différents vecteurs de dépendance.
2) Dessiner le diagramme de dépendances pour $n = 5$.
3) Trouver une fonction de temps de calcul $t(i, j) = \alpha i + \beta j + \gamma$ valide.
4) La fonction de temps trouvée au (3) implique-t-elle un calcul en parallèle ? Si oui, combien de processeurs faut-il pour une matrice de taille n ?
5) Écrire une boucle séquentielle permettant d'effectuer le calcul sur tous les points $1 \leq i, j \leq n$ de la matrice.
6) Pouvait-on effectuer les calculs dans un ordre différent ? Si oui, lequel ?.