Architecture et optimisation de codes pour microprocesseur hautes performances

TD 2 – Ordonnancement et contraintes de ressources

Partie 1 – Contraintes de ressources

On se propose d'étudier la fonction suivante :

```
void my_func( double * X, double * Y,
          double * Z, double a, int N) {
     int i ;
     for ( i = 0 ; i < N ; i++ ) {
          Y[i] = a * X[i] + Y[i] ;
     }
     for ( i = 0 ; i < N ; i++ ){
          Z[i] = X[i] + Y[i] ;
     }
}
```

Cette fonction se compose de deux boucles basées sur les opérations BLAS : la première effectue une opération appelée DAXPY tandis que la seconde additionne deux vecteurs.

Question 1 :

- Les 2 boucles de cette fonction sont-elles indépendantes ?
- Quelles transformations proposez-vous pour élargir le parallélisme d'instructions dans ces boucles ? Quelles sont les conditions à respecter pour pouvoir appliquer ces transformations

Question 2 :

Considérons notre fonction après fusion des deux boucles.

- Ecrire le code C résultant de cette fusion.
- Ecrire le DDG (graphe de dépendance de données) du corps de boucle en ne considérant que les types d'instruction suivants :
    - LD : chargement mémoire (latence de 3 cycles),
    - ADD : addition flottante (latence de 1 cycle),
    - MUL : multiplication flottante (latence de 2 cycles)
    - ST : rangement mémoire (latence 5 cycles).

  On pourra négliger les incréments de l'itérateur `i` en admettant que les chargements mémoire effectuent également une addition entière.

- Est-il nécessaire d'effectuer 2 chargements mémoire pour Y ?

Question 3 :

Dans la suite de ce TD, nous allons viser une architecture VLIW composée de trois ressources : U0, U1 et U2. Nous décrivons les caractéristiques de ces ressources :

1. Sur l'unité U0, on peut exécuter une addition ou une multiplication.
2. Sur l'unité U1, on peut exécuter une addition ou un chargement mémoire.
3. Enfin sur l'unité U2, on peut exécuter un chargement ou un rangement mémoire.

Attention : toutes les unités fonctionnelles sont parfaitement pipelinée sauf U0.

- Construire les tables de réservations pour l'ISA de notre architecture.

Question 4 :

- Construire l'automate de gestion des ressources pour l'architecture décrite dans la question précédente.
  - Pour chaque état, combien de bits doit-on utiliser (fonction des ressources et de la latence des instructions) ?
  - Quelles étiquettes doit-on considérer pour les arcs ?
- L'automate est-il déterministe ? Si non, déterminiser le sous-automate correspondant à l'état initial et les états atteignables à distance de 1.

Question 5 :

- Calculer un ordonnancement du corps de la boucle en utilisant un algorithme d'ordonnancement par liste.
  - Calculer les performances en cycles de cet ordonnancement
- Effectuer un déroulage de boucle d'un facteur 2.
- Réappliquer cet algorithme d'ordonnancement sur le corps de boucle déroulé.
  - Calculer les performances en cycles de cet ordonnancement
- Que pouvez-vous en conclure ?

Partie 2 – Ordonnancement pipeliné

Dans cet exercice, nous allons générer un ordonnancement pipeliné d'une boucle. Cette partie se focalise sur l'algorithme *Iterative Modulo Scheduling* (IMS) publié par Bob Rau. Le cours présente cet algorithme ainsi que les formules pour les calculs de hauteur et d'intervalle d'ordonnancement. Néanmoins, il peut être nécessaire de jeter un coup d'œil à la publication d'origine pour obtenir plus de détails. Le calcul du MII est détaillé en section 3 tandis que l'algorithme lui-même est détaillé en section 4 (notamment, les figures 7, 8 et 11 peuvent être utiles).

Question 1 :

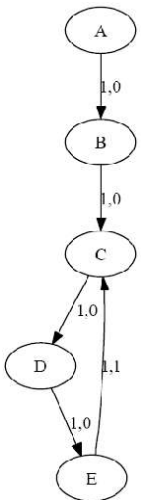
- Calculer la valeur minimal de l'intervalle d'initiation possible pour la boucle de la question 2 (partie 1).
  (On rappelle que cette valeur MII dépend à la fois des contraintes de dépendances cycles, RecMII, et des contraintes de ressources, ResMII.)
- Appliquer l'heuristique Iterative Modulo Scheduling sur notre exemple avec la boucle fusionnée (partie 1, question 2) pour obtenir un ordonnancement pipeliné.
  (L'heuristique se décompose en plusieurs étapes : (i) trie des nœuds grâce au calcul de la hauteur de chacune des instructions et (ii) ordonnancement d'une itération avec respect de la répétition tous les II cycles)
- Quelle la valeur de l'intervalle d'initiation II, de la profondeur D ainsi que du *makespan* M
  - Calculer les performances en cycles de cet ordonnancement

Question 2 :

- Comparer les ordonnancements pipelinés (question 1, partie 2) et non pipelinés de notre boucle (question 5, partie 1). Que peut-on en conclure ?

Question 3 :

Considérons à présent le graphe de dépendances ainsi que les tables de réservations suivantes :



A	r0	r1	r2
0	X		

B	r0	r1	r2
0	X		

C	r0	r1	r2
0		X	

D	r0	r1	r2
0	X		

E	r0	r1	r2
0			X

- Calculer l'ordonnancement pipeliné de cet ensemble d'instructions en appliquant l'heuristique IMS
- Quel est l'intérêt de la variable *Budget* décrit dans l'algorithme de la figure 8 ?
- Discuter de l'ordonnancement final en fonction de la valeur de cette variable.

Partie 3 – Swing Modulo Scheduling

Dans cette partie, nous allons étudier l'article *Swing Modulo Scheduling* publié par Llosa et al à la conférence PACT 1996.

Question 1 :

La section 1 (Introduction) de cet article présente un état de l'art sur l'ordonnancement pipeliné ainsi que les motivations de leur travail.

- Quel est le message de l'article ?
- Comment se compare principalement les auteurs aux travaux précédents ?

Question 2 :

La section 4 présente le cœur de l'algorithme de *Swing Modulo Scheduling*.

- Quels sont les attributs portés par les arcs dans le graphe de dépendance ?
- Combien d'étapes contient cet algorithme (on ne comptera pas la construction du graphe de dépendance comme une étape) ?
- Décrire brièvement ces étapes.

Question 3 :

Les auteurs introduisent plusieurs calculs préliminaires sur le graphe de dépendances. On retrouve le calcul de la hauteur de l'instruction qui se nomme H (ceci reprend le même principe que pour l'algorithme IMS).

- Lister ces grandeurs (en excluant H) en décrivant brièvement leur impact.

Question 4 :

L'algorithme de tri des nœuds est un peu plus sophistiqué que pour l'IMS. C'est d'ailleurs la contribution principale de l'article. Le pseudo-code de ce tri est présenté dans la figure 6.

- L'algorithme impose, en entrée, des ensembles de nœuds. Comment obtient-on ces ensembles ?
- Cet algorithme parcourt ces ensembles un par un (nommé S dans la figure 6). Pour un ensemble S fixé, quel est la principale différence avec l'algorithme de tri par priorité de l'IMS ?
- Une fois ce tri terminé, est-on sur que tous les nœuds ont été visités ? Décrire une ébauche de preuve.

Question 5 :

La section 4.3 décrit de façon littérale l'algorithme d'ordonnancement une fois que les nœuds du graphe ont été triés.

- Ecrire cet algorithme dans le pseudo-code de votre choix
- Quelles sont les principales différences avec l'algorithme d'ordonnancement IMS ?
- Comment les auteurs gèrent-ils les conflits de ressources ? Proposent-ils une nouvelle solution ?
- Parmi les tables de réservation et les automates, quelle représentation est la plus adéquate pour ce genre d'algorithme ?

Question 6 :

- Appliquer l'algorithme SMS au graphe de dépendance de la question 3, partie 2.
- Que peut-on en conclure par rapport à l'ordonnancement trouvé avec l'IMS (partie 2, question 1) ?