Analyse statique des erreurs de prédiction de branchement

RAPPORT INTERNE IRIT / 2004-27-R

Claire Burguière et Christine Rochange
Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier, CNRS
Toulouse, France
{burguier,rochange}@irit.fr

Novembre 2004

Résumé : L'utilisation de processeurs haute-performance dans les systèmes temps-réel augmente la difficulté de garantir que les programmes respecteront leurs échéances. Tandis que le calcul de temps d'exécution pire-cas (WCET) repose sur une analyse statique du code, modéliser avec suffisamment de fiabilité, mais aussi de précision, le comportement dynamique des processeurs actuels reste un défi. Dans ce rapport, nous nous intéressons plus particulièrement au prédicteur de branchement. Plusieurs modèles ont été proposés par le passé pour le prendre en compte dans le calcul de WCET. Certains présentent une complexité importante tandis que d'autres nécessitent de connaître la structure du code. Nous proposons ici un modèle qui analyse le code binaire à un coût modéré. Nous calculons une borne du nombre de mauvaises prédictions qui nous permet de vérifier la précision des résultats obtenus. Nous montrons également comment intégrer, de manière réaliste, les pénalités d'erreur de prédiction dans le calcul de WCET.

Abstract : The wider and wider use of high-performance processors as part of real-time systems makes it more and more difficult to guarantee that programs will respect their strict deadlines. While the computation of Worst-Case Execution Times relies on static analysis of the code, the challenge is to model with enough safety and accuracy the behaviour of intrisically dynamic components. In this report, we focus on the dynamic branch predictor. Several models to bound the number of branch mispredictions have been previously published. Some of them exhibit a high complexity while other ones have shown that taking into account semantic information from the source code makes things more tractable. We propose a model which analyse binary code with a moderate cost. We calculate an upper bound on the number of mispredictions which allows us to verify the accuracy of our results. We show how to integrate misprediction penalties in WCET analysis.

Chapitre 1

Introduction

La connaissance du temps d'exécution pire-cas de programmes (ou *WCET : Worst-Case Execution Time*) est nécessaire, lors de la conception de systèmes temps-réel stricts, pour définir un ordonnancement des tâches qui garantit le respect des échéances. Les méthodes d'évaluation du WCET orientées "mesures" ne permettent pas d'obtenir des résultats fiables parce qu'on ne peut pas, en général, garantir que tous les chemins possibles (ou en tout cas les plus longs) ont été explorés. Des méthodes basées sur une analyse statique du code permettent, elles, de calculer une borne supérieure sûre du WCET.

Certains composants ont un comportement fortement dynamique qui dépend de l'historique passé de l'état du processeur, ce qui rend l'analyse statique compliquée. Parmi ces composants, nous nous intéressons, dans ce rapport, au prédicteur de branchement.

La *prévisibilité temporelle* d'un processeur repose sur l'existence de techniques qui puissent modéliser son architecture et calculer une borne supérieure du temps d'exécution. La complexité de ces techniques doit entrer en ligne de compte et un processeur peut être déclaré "non prévisible" si les besoins en temps de calcul et/ou en mémoire pour analyser le WCET sont prohibitifs. La précision de la borne de WCET obtenue peut aussi modérer la notion de prévisibilité : si l'analyse repose sur des hypothèses très pessimistes sur le fonctionnement du matériel, le système ne peut pas être considéré comme "prévisible". Aussi, on dira qu'un processeur est *prévisible* s'il peut être analysé à un coût raisonnable et si le WCET estimé n'est pas trop éloigné du WCET réel. Par extension, un prédicteur de branchement est prévisible s'il ne remet pas en question la prévisibilité du processeur.

Prendre en compte la prédiction de branchement dans le calcul de WCET, c'est : (a) estimer le nombre de mauvaises prédictions, et (b) intégrer dans le calcul du WCET les pénalités (en termes de temps d'exécution) induites par les mauvaises prédictions. Comme nous le verrons, plusieurs modèles ont été proposés ces dernières années pour borner le nombre d'erreurs de prédiction, et certains prennent en compte des mécanismes de prédiction évolués. Cependant, ces modèles sont complexes et on peut se demander s'ils suffisent à déclarer les prédicteurs de branchement "prévisibles". Dans ce rapport, nous restreignons l'étude à des processeurs qui permettent de choisir le mode de prédiction (statique ou dynamique), branchement par branchement. Ceci permet de mettre en œuvre des stratégies qui empêchent les conflits dans les tables de prédiction, conflits à l'origine d'une grande partie de la complexité des modèles existants. Nous montrons qu'il est possible, en l'absence de conflits, de modéliser de manière précise le comportement du prédicteur, et ce sans avoir à connaître la structure algorithmique du code (ce qui permet de traiter des codes non structurés, ou dont on ne sait pas retrouver la structure, comme c'est le cas lorsque on ne dispose que du code binaire). De plus, nous étendons le calcul de bornes maximales du nombre de mauvaises prédictions proposé dans [1] pour le rendre applicable à tout branchement "de boucle" ce qui nous permet de vérifier la précision des résultats obtenus.

Par ailleurs, les travaux de la littérature adoptent une représentation simplifiée des pénalités de mauvaise prédiction : pénalité unique et fixe pour l'ensemble des branchements, ou pénalités différentes entre branchements mais identiques pour les deux branches. Nous montrerons que cette représentation est assez éloignée de la réalité et nous proposerons une prise en compte plus précise des diverses pénalités.

Chapitre 2

Prédiction de branchement et calcul du WCET

Nous allons rappeler les principaux mécanismes de prédiction de branchement et montrer en quoi il est parfois délicat de les modéliser dans le cadre d'une analyse statique du WCET.

2.1 Techniques de prédiction de branchement

Parmi les techniques de prédiction, on distingue les techniques statiques, qui imposent des schémas de prédiction figés, des techniques dynamiques, implémentées dans le matériel, qui analysent l'historique des branchements pour anticiper le comportement des branchements.

2.1.1 Prédiction statique

La prédiction statique [5] est élaborée au préalable et n'évolue pas au cours de l'exécution. Les mécanismes les plus simples fixent une même prédiction pour tous les branchements (toujours *pris* ou toujours *non pris*). Certains modulent la prédiction en fonction du type de branchement (par exemple, les branchements en arrière sont prédits *pris* et tous les autres sont prédits *non pris*). Ces stratégies simples peuvent être mises en œuvre par le matériel. D'autres stratégies, plus élaborées, sont mises en œuvre à la compilation et prédisent individuellement chaque branchement. Elles basent leurs prédictions sur une classification des branchements en fonction de la structure du code, ou parfois sur l'étude de traces d'exécution.

Les diverses techniques de prédiction statique ont toutes pour inconvénient de figer la prédiction pour un branchement donné. Or la plupart des branchements conditionnels ont un comportement qui évolue au cours de l'exécution. Les mécanismes de prédiction dynamique que nous allons présenter ci-dessous permettent d'adapter, dans une certaine mesure, les prédictions à cette évolution.

2.1.2 Prédiction dynamique

La prédiction dynamique de la direction d'un branchement est construite au cours de l'exécution sur la base des comportements passés des branchements. L'historique d'un branchement est généralement représenté par un compteur 2-bits à saturation, mis à jour au gré des directions suivies par le branchement, comme présenté sur la figure 2.1. Le compteur évolue selon les transitions étiquetées “1” (respectivement “0”) quand le branchement est exécuté *pris* (resp. *non pris*). La prédiction d'un branchement se fait sur la base de son compteur : lorsque le bit de poids fort du compteur est à 1, le branchement est prédit *pris* ; lorsqu'il est à 0, le branchement est prédit *non pris*.

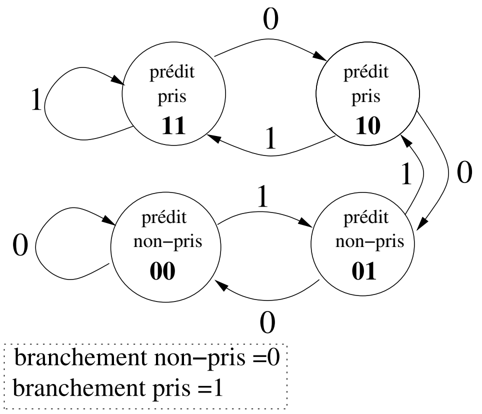

FIG. 2.1 – Compteur 2 bits à saturation

Lorsqu'un branchement est détecté dans le flot d'instructions, il faut prédire sa direction et son adresse cible. Les adresses cibles calculées par le passé sont rangées dans une table appelée BTB (*Branch Target Buffer*), tandis que les compteurs permettant de prédire la direction sont rangés dans une seconde table appelée BHT (*Branch History Table*). La table des adresses cibles est indexée par l'adresse du branchement. C'est parfois aussi le cas de la table des compteurs : on parle alors de "prédicteur local" [8]. D'autres prédicteurs exploitent les corrélations éventuelles entre branchements : la direction d'un branchement donné peut être liée aux directions suivies par les branchements précédents. Un registre à décalage enregistre les directions suivies par les derniers branchements exécutés, ce qui constitue un historique global. La table de compteurs peut alors être indexée par cet historique [9]. Ainsi, chaque branchement peut accéder à plusieurs compteurs utilisés en fonction du comportement des branchements précédents.

Quelle que soit la manière d'indexer la table de prédiction, il y a risque de conflits entre branchements pour une même entrée de la table. La table des adresses cibles est étiquetée (chaque entrée contient l'adresse du branchement, en plus de son adresse cible), ce qui permet de détecter les conflits éventuels. Par contre, par soucis d'économie, la table des compteurs n'est généralement pas étiquetée et plusieurs branchements peuvent se partager une même entrée. C'est particulièrement vrai dans le cas où la table est indexée par l'historique global. Les conflits peuvent être limités en mettant en œuvre des fonction d'indexage plus évoluées (par exemple, un ou-exclusif entre l'historique global et certains bits de l'adresse du branchement), mais ne peuvent pas être complètement éliminés (sauf intervention spécifique du compilateur, comme on le verra plus loin). Les interférences entre branchements peuvent dans certains cas avoir un effet positif, mais elles sont le plus souvent destructrices.

2.2 Analyse statique d'un prédicteur de branchement

Prendre en compte la prédiction de branchement dans le calcul de WCET, c'est évaluer statiquement le nombre d'erreurs de prédiction. Ces erreurs sont directement liées à l'évolution des compteurs de prédiction et aux conflits dans les tables qui peuvent perturber cette évolution.

Faisons, dans un premier temps, abstraction des conflits. Dans le cas où la table des compteurs est indexée par l'adresse des branchements, l'étude du compteur d'un branchement consiste à considérer des exécutions successives de ce branchement (les seules arêtes à considérer étant celles qui sortent du bloc de base : arête vers le bloc en séquence et arête vers le bloc cible). Dans le cas où la table est indexée par l'historique global, et si celui-ci comporte n bits, chaque branchement fait évoluer 2^n compteurs différents. Pour modéliser le comportement de ces compteurs, il faut prendre en compte tous les chemins permettant d'atteindre le branchement, ce qui complique grandement la modélisation (la description du modèle est plus longue et le temps de résolution est également plus important).

La prise en compte des conflits éventuels nécessite de les identifier, mais aussi d'évaluer leurs effets (qui ne sont pas forcément négatifs). Il s'agit donc, dans un premier temps, de déterminer les branchements qui accèdent à une même entrée de la table de prédiction. Ceci est plus ou moins facile, selon la manière dont cette table est indexée : dans le cas où l'index est l'adresse du branchement, les conflits sont faciles à prévoir ; si l'index est un historique global, une analyse complexe des chemins d'exécution est nécessaire (il faut trouver tous les branchements qui peuvent être atteints avec un même historique). Une fois les conflits détectés, il faut analyser tous les branchements concernés de manière conjointe pour appréhender l'evolution de leur compteur commun.

Dans le paragraphe suivant, nous allons faire le point sur ce qui a été modélisé à ce jour.

2.3 État de l'art

Les premiers travaux concernant la modélisation d'un prédicteur de branchement dans le calcul de WCET ont été menés par Colin et Puaut [2]. Ils considèrent des tables indexées par l'adresse des branchements, et proposent une première analyse des tables de prédiction par simulation statique, qui permet de prévoir si les branchements sont dans la table ou non lorsqu'ils doivent être prédits : en cas d'absence, le branchement est considéré comme mal prédit par défaut, ce qui revient à ignorer les effets éventuellement constructifs des conflits. Le résultat de cette première analyse est ensuite couplé à des informations sur la structure algorithmique du code (ce travail s'inscrit dans un calcul du WCET à partir de l'arbre syntaxique) pour évaluer le nombre d'erreurs pour un prédicteur mettant en œuvre des compteurs 2 bits.

Mitra et Roychoudhury [6] proposent un modèle de prédicteur dans le cadre d'un calcul de WCET par la méthode IPET [4] (cette méthode sera développée dans la section 4). Ils considèrent des tables indexées par un historique global, et une prédiction construite à partir de compteurs 1 bit (il n'est pas clair que leur modèle puisse être étendu à des compteurs 2 bits). Les relations entre chemins suivis et valeurs du compteur sont exprimées en termes de contraintes sur les nombres d'exécutions des blocs de base et les valeurs possibles du compteur en entrée d'un bloc donné. Le nombre de contraintes est très important et il n'est pas sûr que le modèle soit calculable si on considère des programmes importants. C'est ce qui a mené Engblom [3] à la conclusion que les prédicteurs de branchements à historique global ne pouvaient pas être pris en compte dans le calcul de WCET.

Enfin, Bate et Reutemann [1] étudient un prédicteur supposé sans conflits et exploitent la connaissance de la structure algoritmique du code pour calculer, de manière analytique, des bornes supérieures des nombres de mauvaises prédiction dans le cas d'un prédicteur local (table indexée par l'adresse du branchement) et de compteurs 2 bits.

2.4 Vers un prédicteur de branchement prévisible

Notre sentiment, après étude des modèles de la littérature, est qu'il est difficile de prendre en compte, de manière suffisamment simple et précise, les conflits dans les tables de prédiction et leurs effets sur les heuristiques de prédiction. C'est pourquoi, nous étudions actuellement des solutions pour éviter ces conflits. Bate et Reutemann [1] suggèrent de travailler sur le positionnement du code en mémoire pour placer les branchements à des adresses associées à des entrées distinctes de la table. Notre approche est différente.

Il y a quelques années, Patil et Emer [7] ont proposé de réduire les risques de conflits en combinant prédiction statique et prédiction dynamique : seuls certains branchements sont pris en compte par le prédicteur dynamique, et se voient allouer une entrée dans la table de prédiction. Les autres branchements sont prédits de manière statique, la direction prédite étant calculée par le compilateur pour chaque branchement. Ainsi, on peut contrôler le contenu de la table de prédiction et empêcher les conflits. Cette solution repose sur la possibilité de maîtriser le mode de prédiction pour chaque branchement. Alors que la plupart des processeurs imposent un mode de prédiction commun à tous les branchements, le récent processeur IA-64 d'Intel offre cette possibilité, grâce à des extensions (*branch hints*) des instructions de contrôle :

`.spnt`	prédiction statique : *non pris*
`.sptk`	prédiction statique : *pris*
`.dp`	prédiction dynamique

Avec ce type d'extension, le compilateur peut choisir les branchements qui seront prédits dynamiquement et ainsi empêcher les conflits. Patil et Emer proposent une analyse de traces d'exécution pour choisir les branchements à prédire dynamiquement. Ils examinent le comportement du prédicteur dynamique pour chaque branchement et choisissent de prédire statiquement : (a) les branchements très fortement biaisés, facile à prédire dynamiquement mais aussi statiquement, et (b) les branchements difficiles à prédire dynamiquement pour lesquels il y aura de toute façon un nombre d'erreurs important. Nous recherchons actuellement des heuristiques de choix adaptées à notre objectif d'améliorer la *prévisibilité* du processeur du point de vue du WCET. Un des critères sera de ne laisser prédire dynamiquement que les branchements faciles à analyser.

Dans le reste de cet article, nous nous plaçons dans le cas d'un prédicteur sans conflits, avec indexage de la table de prédiction par l'adresse du branchement. Ceci nous permet de modéliser de manière plus précise les erreurs de prédiction.

Chapitre 3

Impact de la prédiction de branchement sur le temps d'exécution

Une erreur de prédiction de branchement se traduit par un temps d'exécution plus élevé car le pipeline doit être vidé des instructions du chemin suivi par erreur, et rechargé avec les instructions du chemin correct. Le temps ainsi perdu constitue ce que l'on appelle une *pénalité* de mauvaise prédiction.

Les modèles de prédicteurs cités dans le paragraphe 2.3 intègrent les pénalités de manière simplifiée : soit ils considèrent une valeur unique de pénalité pour tous les branchements du programme [2], soit ils attribuent une pénalité différente à chaque branchement, mais cette pénalité est identique sur chacune des deux branches [6]. Nous avons réalisé un certain nombre de mesures qui montrent qu'il est souhaitable de représenter les choses de manière plus précise.

Dans la pratique, la pénalité d'erreur de prédiction d'un branchement donné suivant une des deux directions peut être calculée en mesurant la séquence des deux blocs liés par ce branchement en prédisant ce dernier correctement puis de manière erronée : la pénalité est la différence entre les deux temps obtenus.

Nous avons utilisé un simulateur au niveau cycle, capable d'extraire d'un graphe de flot de contrôle toutes les séquences de deux blocs et de les exécuter pour obtenir leur temps d'exécution. Ce simulateur modélise un processeur générique, superscalaire de degré 4, à ordonnancement dynamique des instructions. Nous avons alimenté ce simulateur par différents codes binaires de benchmarks (`fibcall, matmul, jfdctint, crc, ludcmp, lms, fft1`) issus de la suite SNU (`www.archi.snu.ac.kr/realtime/benchmark/`) compilés pour PowerPC 603.

La figure 3.1 montre la distribution des valeurs de pénalité (en nombre de cycles) sur l'ensemble des branchements de tous les benchmarks. Les valeurs varient de 1 à 12, et si l'on devait fixer une pénalité unique pour tous les branchements, il faudrait choisir la valeur maximale, ce qui engendrerait sans aucun doute (nous le vérifierons sur un exemple, dans le paragraphe 6.2) une surestimation conséquente du WCET.

Nous avons également mesuré, pour chacun des branchements, la différence de pénalité entre chacune des deux branches. La figure 3.2 montre que cette différence n'est pas négligeable. Elle est du même ordre de grandeur que le temps d'exécution d'un bloc de base de quelques instructions. Là encore, on voit que prendre une valeur de pénalité unique pour les deux branches d'une instruction de flot ne permet pas d'obtenir une valeur précise du temps d'exécution global.

Nous étendons la représentation du programme par graphe de flot de contrôle pour expri-

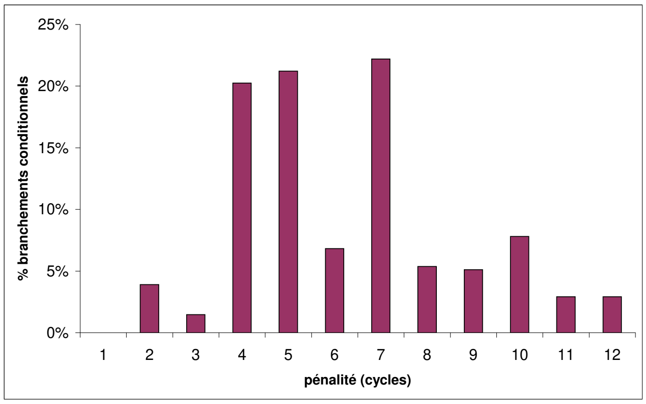

FIG. 3.1 – Distribution des pénalités de mauvaises prédictions

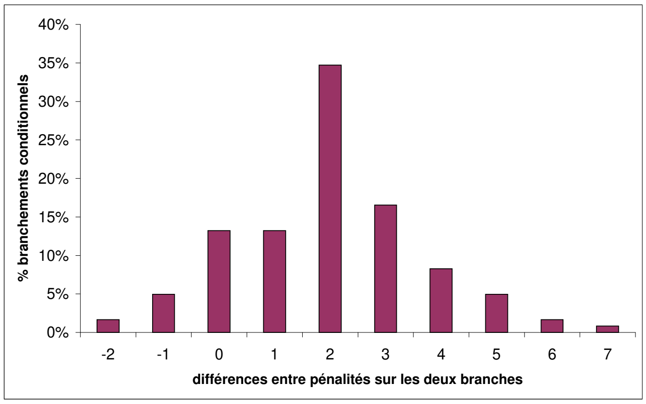

FIG. 3.2 – Distribution des différences entre les pénalités mesurées sur les deux branches

mer ces différences. Chaque bloc terminé par un branchement conditionnel est lié à chacun de ses deux successeurs par deux arêtes : l'une est associée à une prédiction correcte du branchement (bp) et l'autre à une erreur de prédiction (mp). Ceci est illustré sur la figure 3.3. Ces deux transitions doivent être considérées séparément dans le calcul du WCET.

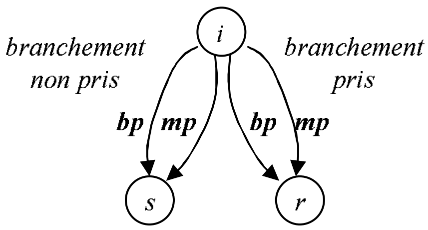

FIG. 3.3 – Extension du graphe de flot de contrôle pour une modélisation précise des pénalités

Chapitre 4

Analyse du compteur de prédiction

Comme indiqué précédemment, il nous paraît très important de pouvoir effectuer une évaluation du WCET sur le code binaire, car ceci permet de prendre en compte toute transformation de la structure du code réalisée par le compilateur. Par ailleurs, cela permet de traiter des codes dont on ne possède pas le code source, comme par exemple des fonctions issues de bibliothèques. La méthode IPET [4], qui se base sur une représentation du code sous forme de graphe de flot de contrôle, permet d'analyser des codes binaires. C'est cette méthode que nous avons retenue pour notre étude.

La méthode IPET exprime le temps d'exécution du programme à partir des nombres d'exécution des blocs de base (nœuds du graphe de flot de contrôle) et de leurs temps d'exécution individuels. Le recouvrement de deux blocs de base dans le pipeline produit un gain qui est modélisé par un temps d'exécution négatif associé à l'arête liant les deux blocs. Le WCET est calculé en maximisant le temps d'exécution global tout en respectant un certain nombre de contraintes qui expriment la structure du graphe de flot de contrôle (contraintes liant les nombres d'exécution des blocs et des arêtes) ainsi que des informations de flot (par exemple, des bornes de boucles). La formulation de ce problème d'optimisation sera développée dans la partie 4.2.

On peut prendre en compte la prédiction de branchement dans le calcul de WCET par cette méthode en ajoutant des contraintes qui expriment les nombres de mauvaises prédictions des branchements et les lient aux nombres d'exécution des blocs et des arêtes. C'est ce qui est proposé dans le paragraphe suivant.

4.1 Modélisation de la prédiction de branchement

On cherche ici à évaluer le nombre de mauvaises prédictions subies par le branchement conditionnel qui termine un bloc de base i. Comme nous l'avons expliqué précédemment, modéliser la prédiction de branchement dans un contexte sans conflits dans les tables de prédiction revient à décrire l'évolution du compteur de prédiction. L'état du compteur, noté c, peut prendre une valeur dans l'ensemble $\mathcal{E} = \{00, 01, 10, 11\}$.

Si b_i est le nombre d'occurences du bloc i sur le chemin d'exécution, et si b_i^c est le nombre d'exécutions de i pour lesquelles le compteur de prédiction est dans l'état c à l'entrée du bloc, on peut écrire :

$$b_i = \sum_{c \in \mathcal{E}} b_i^c$$

De la même manière, pour le nombre de mauvaises prédictions du branchement terminant le bloc i :

$$m_i = \sum_{c \in \mathcal{E}} m_i^c$$

De plus, le nombre de mauvaises prédictions est forcément inférieur au nombre total d'exécutions du bloc :

$$\forall c \in \mathcal{E}, m_i^c \leq b_i^c$$

Soient *start* et *end* le premier et le dernier bloc de base du programme (pour simplifier l'explication, on suppose qu'il y a un seul point de sortie, mais le modèle peut facilement être étendu pour en prendre en compte plusieurs). Notons $p_{i \rightsquigarrow i}^c$ le nombre de fois où, après avoir exécuté le bloc i rencontré avec le compteur dans l'état c, le chemin d'exécution atteint de nouveau le bloc i. La variable $p_{start \rightsquigarrow i}^c$ (respectivement $p_{i \rightsquigarrow end}^c$) représente le nombre de fois où, partant du bloc *start*, on atteint le bloc i pour la première fois (respectivement, partant du bloc i pour la dernière fois, on atteint le bloc *end*) avec le compteur dans l'état c en entrée (respectivement en sortie) du bloc i. Sachant que, pour une exécution donnée, il y a un seul chemin menant de *start* à la première exécution de i (l'état du compteur c en entrée de i est alors égal à la valeur d'initialisation des compteurs), et qu'il y a un seul chemin menant de la dernière exécution de i à *end*, on peut écrire :

$$\sum_{c \in \mathcal{E}} p_{start \rightsquigarrow i}^c = 1$$

$$\sum_{c \in \mathcal{E}} p_{i \rightsquigarrow end}^c = 1$$

Le nombre d'exécutions de i avec le compteur de prédiction en entrée dans l'état c est donné par :

$$\forall c \in \mathcal{E}, \quad b_i^c = p_{start \rightsquigarrow i}^c + p_{i \rightsquigarrow i}^c + p_{i \rightsquigarrow end}^c$$

Nous allons maintenant décrire l'évolution du compteur. Pour cela, on note $p_{i \rightsquigarrow i}^{c,d}$ le nombre de fois où, à partir de i avec le compteur dans l'état c, on suit la direction d ($d = 0$ pour une exécution en séquence, $d = 1$ pour un branchement pris) et on atteint de nouveau i. On utilisera le même type de notation pour le bloc *end*, et on écrira :

$$\forall c \in \mathcal{E}, \quad p_{i \rightsquigarrow end}^c = \sum_{d \in \{0,1\}} p_{i \rightsquigarrow end}^{c,d} = 1$$

Le nombre d'exécutions du bloc i avec le compteur en entrée dans un état donné est exprimé par les équations suivantes qui modélisent l'évolution du compteur en fonction des directions suivies par le branchement :

$$b_i^{00} = p_{start \rightsquigarrow i}^{00} + p_{i \rightsquigarrow i}^{00,0} + p_{i \rightsquigarrow i}^{01,0}$$

$$b_i^{01} = p_{start \rightsquigarrow i}^{01} + p_{i \rightsquigarrow i}^{10,0} + p_{i \rightsquigarrow i}^{00,1}$$

$$b_i^{10} = p_{start \rightsquigarrow i}^{10} + p_{i \rightsquigarrow i}^{11,0} + p_{i \rightsquigarrow i}^{01,1}$$

$$b_i^{11} = p_{start \rightsquigarrow i}^{11} + p_{i \rightsquigarrow i}^{10,1} + p_{i \rightsquigarrow i}^{11,1}$$

Les arêtes en sortie du bloc i donnent également :

$$\forall c \in \mathcal{E}, \quad b_i^c = p_{i \rightsquigarrow i}^{c,0} + p_{i \rightsquigarrow i}^{c,1} + p_{i \rightsquigarrow end}^{c}$$

Si le bloc i est suivi du bloc s en séquence et du bloc r en cas de redirection (branchement pris), on a :

$$\sum_{c \in \mathcal{E}} (p_{i \rightsquigarrow i}^{c,0} + p_{i \rightsquigarrow end}^{c,0}) = a_{i,s}$$

$$\sum_{c \in \mathcal{E}} (p_{i \rightsquigarrow i}^{c,1} + p_{i \rightsquigarrow end}^{c,1}) = a_{i,r}$$

Il est maintenant simple d'exprimer le nombre de mauvaises prédictions. C'est le nombre de fois où l'on emprunte la branche qui ne correspond pas à la prédiction, c'est-à-dire lorsque le branchement est prédit *pris* (resp. *non pris*) et est en réalité *non pris* (resp. *pris*). Ces deux situations sont décrites par les expressions suivantes :

$$m_i^{00} = p_{i \rightsquigarrow i}^{00,1} + p_{i \rightsquigarrow end}^{00,1}$$

$$m_i^{01} = p_{i \rightsquigarrow i}^{01,1} + p_{i \rightsquigarrow end}^{01,1}$$

$$m_i^{10} = p_{i \rightsquigarrow i}^{10,0} + p_{i \rightsquigarrow end}^{10,0}$$

$$m_i^{11} = p_{i \rightsquigarrow i}^{11,0} + p_{i \rightsquigarrow end}^{11,0}$$

Les nombres de mauvaises prédictions sur les branches en séquence (branchement non pris) et avec redirection (branchement pris) sont donnés par :

$$m_{i,s} = m_i^{10} + m_i^{11}$$

$$m_{i,r} = m_i^{00} + m_i^{01}$$

4.2 Système de contraintes pour un calcul de WCET par la méthode IPET

Soient b_i et $a_{i,j}$ les nombres d'exécution respectifs du bloc i et de l'arête reliant le bloc i au bloc j. Pour chaque bloc (sauf le premier et le dernier), le nombre d'exécutions des arêtes entrantes est égal à celui des arêtes sortantes. On peut alors écrire :

$$b_i = \sum_j a_{i,j} = \sum_j a_{j,i}$$

De plus, comme les blocs de début et de fin sont exécutés une seule fois, on a :

$$b_{start} = b_{end} = 1$$

A ces contraintes structurelles, on peut ajouter des contraintes de flot. Par exemple, pour un bloc de base i situé à l'intérieur d'une boucle dont le nombre maximum d'itérations est M, on peut écrire :

$$b_i \leq M$$

Le temps d'exécution total T est exprimé par :

$$T = \sum_i b_i \times t_{b_i} + \sum_{i,j} a_{i,j} \times t_{a_{i,j}}$$

où t_{b_i} et $t_{a_{i,j}}$ sont les temps d'exécution du bloc i et de l'arête $i \to j$ (gain apporté par le recouvrement des blocs i et j dans le pipeline).

Pour intégrer la description du prédicteur de branchement dans ce modèle, il faut lier les nombres de mauvaises prédictions des branchements aux nombres d'exécution des arêtes correspondantes. Pour ce faire, on écrit :

$$a_{i,j} = m_{i,j} + a'_{i,j}, \quad j \in \{r, s\}$$

où $a'_{i,j}$ représente le nombre de fois où l'arête $i \to j$ est exécutée avec le branchement bien prédit.

Les pénalités $(p_{i,j})$ sont intégrées dans l'expression du temps total d'exécution de la manière suivante :

$$T = \sum_i b_i \times t_{b_i} + \sum_{i,j} a_{i,j} \times t_{a_{i,j}} + \sum_{i,j} m_{i,j} \times p_{i,j}$$

Notons que, pour tout bloc i qui ne se termine pas par un branchement conditionnel, on a $m_{i,j} = 0$.

Le WCET est obtenu en maximisant T sous les contraintes définies ci-dessus : contraintes structurelles, contraintes de flot, et contraintes modélisant le comportement du prédicteur de branchement. La résolution de ce problème d'optimisation peut se faire en utilisant, par exemple, des techniques de programmation linéaire en nombre entiers.

Nous avons ainsi proposé une modélisation simple permettant la description précise de l'évolution des compteurs de prédiction pour chaque branchement conditionnel. Nous allons montrer par un exemple la facilité de synthèse du système et la précision des résultats en utilisant la borne maximale du nombre de mauvaises prédictions calculée dans le cas général.

Chapitre 5

Évaluation du nombre de mauvaises prédictions maximum

Le modèle présenté dans la section précédente permet de calculer le nombre de mauvaises prédictions de chaque branchement conditionnel sur le chemin le plus long, en prenant en compte les temps d'exécution et les pénalités sur chacune des branches. Nous montrons ici comment on peut calculer une borne supérieure de ce nombre de mauvaises prédictions. Nous pourrons par la suite les comparer avec les résultats obtenus par notre modèle.

Nous nous intéresserons aux branchements "de boucle" en nous basant sur le schéma de compilation de la figure 5.1. Le branchement est pris pour sortir de la boucle.

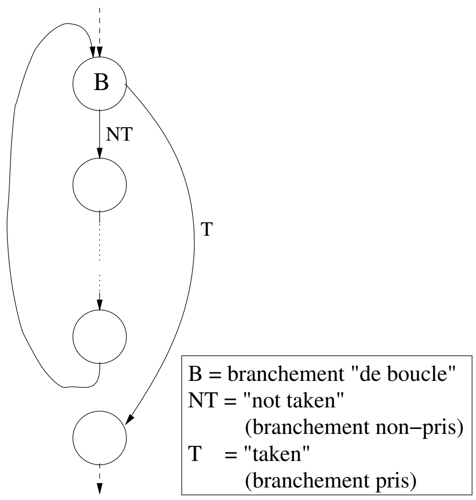

Fig. 5.1 – Structure de boucle

Bate et Reutemann [1] ont analysé le nombre de mauvaises prédictions pour des branchements "de boucle". Cependant, ils ont calculé le nombre total de mauvaises prédictions pour un branchement. Nous proposons une étude des bornes pour chacune des branches séparément.

Pour une boucle simple, similaire à celle de la figure 5.1, le tableau 5.1 donne le nombre de mauvaises prédictions sur chaque branche pour tout état initial du compteur et pour un nombre n d'itérations de la boucle. Notons que dans chaque cas le nombre de mauvaises prédictions total est le même que celui calculé dans [1].

Init	nombre d'itérations (évolution du compteur 2-bit | nombre de mauvaises prédictions (NT/T))							
	$n = 0$		$n = 1$		$n = 2$		$n \geq 3$	
00	$00 \rightsquigarrow 01$	0/1	$00 \rightarrow 00 \rightsquigarrow 01$	0/1	$00 \rightarrow 00 \rightarrow 00 \rightsquigarrow 01$	0/1	$00 \cdots \rightsquigarrow 01$	0/1
01	$01 \rightsquigarrow 10$	0/1	$01 \rightarrow 00 \rightsquigarrow 01$	0/1	$01 \rightarrow 00 \rightarrow 00 \rightsquigarrow 01$	0/1	$01 \rightarrow 00 \cdots \rightsquigarrow 01$	0/1
10	$10 \rightarrow 11$	0/0	$10 \rightsquigarrow 01 \rightsquigarrow 10$	1/1	$10 \rightsquigarrow 01 \rightarrow 00 \rightsquigarrow 01$	1/1	$10 \rightsquigarrow 01 \rightarrow 00 \cdots \rightsquigarrow 01$	1/1
11	$11 \rightarrow 11$	0/0	$11 \rightsquigarrow 10 \rightarrow 11$	1/0	$11 \rightsquigarrow 10 \rightsquigarrow 01 \rightsquigarrow 10$	2/1	$11 \rightsquigarrow 10 \rightsquigarrow 01 \cdots \rightsquigarrow 01$	2/1

TAB. 5.1 – Nombres de mauvaises prédictions rencontrées lors de l'exécution de n itérations d'une boucle simple ($\rightsquigarrow$ représente une transition causant une mauvaise prédiction)

Bate et Reutemann ont aussi modélisé les boucles imbriquées (la boucle analysée est imbriquée dans une autre et de ce fait est exécutée plusieurs fois). Ils fixent le nombre d'itérations de la boucle interne. Nous considérons un cas plus général où la boucle interne a un nombre d'itérations variable d'une exécution à l'autre. Nous avons calculé l'évolution du compteur 2-bits pour M exécutions de la boucle interne. Cette évolution peut être représentée sous forme d'un automate donné figure 5.2 : chaque transition représente une exécution de la boucle interne et chaque état (à part les états *start* et *end*) représente une valeur possible du compteur 2-bits. Chaque transition est annotée par un triplet $< w_i, m_{NT}, m_T >$: w_i est le nombre de fois où l'arête est exécutée au cours de l'exécution du programme, m_{NT} et m_T sont les nombres de mauvaises prédictions pour chacune des deux branches. Les transitions et les annotations sont dérivées du tableau 5.1 : chaque transition représente l'évolution du compteur d'un état initial à un état final pour n itérations de la boucle. Par exemple, deux transitions sortent du sommet représentant l'état "01" car : aprés l'exécution de la boucle, soit le compteur se trouve dans l'état "01" ($n \geq 1$), soit dans l'état "10" ($n = 0$).

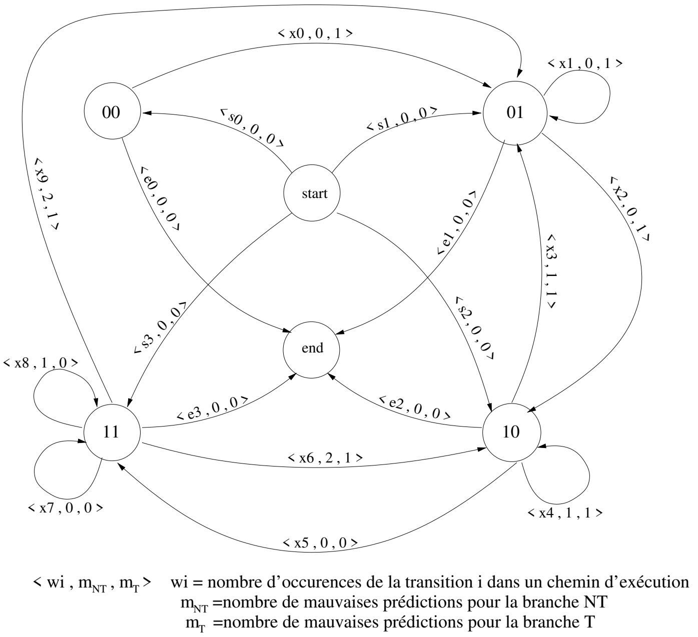

FIG. 5.2 – Évolution du compteur 2-bit pour une boucle imbriquée

Cet automate constitue un support pour calculer les bornes supérieures du nombre de mauvaises prédictions. En considérant M exécutions de la boucle externe, nous pouvons écrire :

$$\sum_{i=0}^{9} x_i = M \qquad (1)$$

$$\forall i,\ 0 \leq x_i \leq M \qquad (2)$$

$$\sum_{i=0}^{3} e_i = 1 \text{ et } \sum_{i=0}^{3} s_i = 1 \qquad (3)$$

Nous allons tout d'abord calculer le nombre de mauvaises prédictions pour la branche NT. Il est donné par :

$$m_{NT} = x_0 + x_3 + x_4 + 2x_6 + x_8 + 2x_9$$

A partir de l'équation (1) nous pouvons écrire :

$$m_{NT} = M - x_1 - x_2 - x_5 + x_6 - x_7 + x_9 \qquad (4)$$

Comme l'état "11" n'est ni final ni initial, nous pouvons écrire :

$$x_9 + x_6 + e_3 = s_3 + x_5$$

qui est équivalent à :

$$x_9 + x_6 - x_5 = s_3 - e_3 \qquad (5)$$

L'équation (3) nous donne :

$$0 \leq e_3 \leq 1 \text{ et } 0 \leq s_3 \leq 1$$

Donc, nous obtenons :

$$-1 \leq s_3 - e_3 \leq 1 \qquad (6)$$

Des équations (2) et (6) nous pouvons déduire :

$$x_9 + x_6 - x_5 - x_1 - x_2 - x_7 \leq 1 \qquad (7)$$

Les équations (4) et (7), nous permettent de construire une borne supérieure de m_{NT} :

$$m_{NT} \leq M + 1$$

Cette borne peut être atteinte : elle l'est par exemple pour la séquence s_3, x_8, x_6, e_2.

Nous pouvons borner le nombre de mauvaises prédictions pour la branche T de la même manière. A partir de la figure 5.2, nous pouvons écrire :

$$m_T = x_0 + x_1 + x_2 + x_3 + x_4 + x_6 + x_9$$

et

$$m_T = M - x_5 - x_7 - x_8 \qquad (8)$$

L'équation (2) nous permet de déduire :

$$-x_5 - x_7 - x_8 \leq 0$$

Et donc :

$$m_T \leq M$$

Comme dans le cas précédent, cette borne peut être atteinte (par exemple pour la séquence s_3, x_6, e_2).

Les bornes maximales du nombre de mauvaises prédictions de chacune des branches d'un branchement conditionnel d'une boucle imbriquée s'exécutant M fois sont :

$$\boxed{\begin{array}{c} m_T \leq M \\ m_{NT} \leq M + 1 \end{array}}$$

Chapitre 6

Application du modèle sur un exemple

6.1 Exemple

Pour montrer l'utilité de la répartition des pénalités et l'efficacité de notre modèle, considérons un exemple contenant plusieurs structures algorithmiques différentes : deux boucles imbriquées et une structure fonctionnelle situé dans le corps de la boucle externe. De plus, le nombre d'itérations de la boucle interne varie d'une itération à l'autre de la boucle externe. Le code de l'exemple et son graphe de flot de contrôle sont détaillés sur les figures 6.1 et 6.2.

```
#define N 20

int main()
{
    int i, j, fact, somme3=0, sommef=0;

    for (i=0; i<N ; i++)
    {
      fact=1;

      for(j=2;j<tab[i];j++)
        fact=fact*j;

      if(tab[i] % 3 == 0)
      {
        tab[i]=tab[i]/3;
        somme3=somme3+tab[i];
      }
      else
        sommef=sommef+fact;
    }
}
```

Fig. 6.1 – Exemple de programme

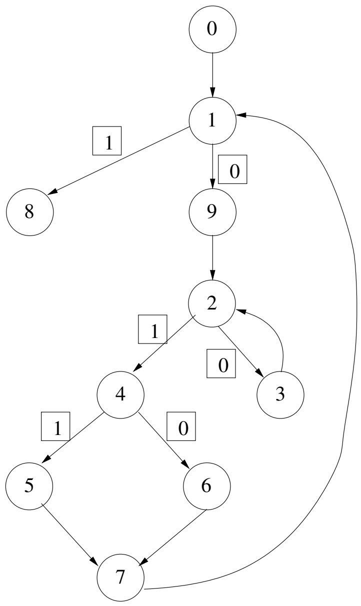

FIG. 6.2 – Graphe de flot de contrôle du programme exemple

Le calcul du WCET de ce programme selon la méthode IPET nécessite de générer un certain nombre de contraintes, énumérées ci-dessous.

Un premier ensemble est consitué des contraintes structurelles qui expriment les liens entre les nœuds et les arêtes dans le graphe de flot de contrôle :

$$\begin{array}{ccc}
b_0 = 1; & b_4 = a_{2,4}; & b_9 = a_{1,9}; \\
b_0 = a_{0,1}; & b_4 = a_{4,5} + a_{4,6}; & b_9 = a_{9,2}; \\
b_1 = a_{0,1} + a_{7,1}; & b_5 = a_{4,5}; & b_8 = a_{1,8}; \\
b_1 = a_{1,9} + a_{1,8}; & b_5 = a_{5,7}; & b_8 = 1; \\
b_2 = a_{9,2} + a_{3,2}; & b_6 = a_{4,6}; & \\
b_2 = a_{2,3} + a_{2,4}; & b_6 = a_{6,7}; & \\
b_3 = a_{2,3}; & b_7 = a_{5,7} + a_{6,7}; & \\
b_3 = a_{3,2}; & b_7 = a_{7,1}\,; &
\end{array}$$

Les contraintes de flot expriment les bornes de boucles. Nous avons fixé ici la taille maximale du tableau à 20 éléments, et la valeur maximale d'un élément du tableau à 5, ce qui implique que la boucle interne ne peut être exécutée que 100 fois au maximum (20×5) :

$$\begin{array}{c}
b_7 \leq 20; \\
b_3 \leq 100;
\end{array}$$

Pour terminer, il faut ajouter les contraintes qui modélisent le prédicteur de branchement. La figure 6.3 présente celles générées pour le branchement conditionnel qui termine le bloc 2.

$$
\begin{array}{ll}
b_2 = b_2^{00} + b_2^{01} + b_2^{10} + b_2^{11}; & p_{0\leadsto 2}^{00} + p_{0\leadsto 2}^{01} + p_{0\leadsto 2}^{10} + p_{0\leadsto 2}^{11} = 1; \\
m_2 = m_2^{00} + m_2^{01} + m_2^{10} + m_2^{11}; & p_{2\leadsto 8}^{00,1} + p_{2\leadsto 8}^{01,1} + p_{2\leadsto 8}^{10,1} + p_{2\leadsto 8}^{11,1} = 1; \\
m_2^{00} <= b_2^{00}; & \\
m_2^{01} <= b_2^{01}; & p_{2\leadsto 8}^{00} = p_{2\leadsto 8}^{00,0} + p_{2\leadsto 8}^{00,1}; \\
m_2^{10} <= b_2^{10}; & p_{2\leadsto 8}^{01} = p_{2\leadsto 8}^{01,0} + p_{2\leadsto 8}^{01,1}; \\
m_2^{11} <= b_2^{11}; & p_{2\leadsto 8}^{10} = p_{2\leadsto 8}^{10,0} + p_{2\leadsto 8}^{10,1}; \\
 & p_{2\leadsto 8}^{11} = p_{2\leadsto 8}^{11,0} + p_{2\leadsto 8}^{11,1};
\end{array}
$$

$$
\begin{array}{ll}
b_2^{00} = p_{0\leadsto 2}^{00} + p_{2\leadsto 2}^{00} + p_{2\leadsto 8}^{00}; & b_2^{00} = p_{2\leadsto 2}^{00,0} + p_{2\leadsto 2}^{01,0} + p_{0\leadsto 2}^{00}; \\
b_2^{01} = p_{0\leadsto 2}^{01} + p_{2\leadsto 2}^{01} + p_{2\leadsto 8}^{01}; & b_2^{00} = p_{2\leadsto 2}^{00,0} + p_{2\leadsto 2}^{00,1} + p_{2\leadsto 8}^{00}; \\
b_2^{10} = p_{0\leadsto 2}^{10} + p_{2\leadsto 2}^{10} + p_{2\leadsto 8}^{10}; & b_2^{01} = p_{2\leadsto 2}^{00,1} + p_{2\leadsto 2}^{10,0} + p_{0\leadsto 2}^{01}; \\
b_2^{11} = p_{0\leadsto 2}^{11} + p_{2\leadsto 2}^{11} + p_{2\leadsto 8}^{11}; & b_2^{01} = p_{2\leadsto 2}^{01,0} + p_{2\leadsto 2}^{01,1} + p_{2\leadsto 8}^{01}; \\
m_2^{00} = p_{2\leadsto 2}^{00,1} + p_{2\leadsto 8}^{00,1}; & b_2^{10} = p_{2\leadsto 2}^{01,1} + p_{2\leadsto 2}^{11,0} + p_{0\leadsto 2}^{10}; \\
m_2^{01} = p_{2\leadsto 2}^{01,1} + p_{2\leadsto 8}^{01,1}; & b_2^{10} = p_{2\leadsto 2}^{10,0} + p_{2\leadsto 2}^{10,1} + p_{2\leadsto 8}^{10}; \\
m_2^{10} = p_{2\leadsto 2}^{10,0}; & b_2^{11} = p_{2\leadsto 2}^{10,1} + p_{2\leadsto 2}^{11,1} + p_{0\leadsto 2}^{11}; \\
m_2^{11} = p_{2\leadsto 2}^{11,0}; & b_2^{11} = p_{2\leadsto 2}^{11,0} + p_{2\leadsto 2}^{11,1} + p_{2\leadsto 8}^{11};
\end{array}
$$

$$
\begin{array}{ll}
m_{2,3} = m_2^{10} + m_2^{11}; & a_{2,3} = m_{2,3} + a'_{2,3}; \\
m_{2,4} = m_2^{00} + m_2^{01}; & a_{2,4} = m_{2,4} + a'_{2,4};
\end{array}
$$

$$
p_{2\leadsto 2}^{00,0} + p_{2\leadsto 8}^{00,0} + p_{2\leadsto 2}^{01,0} + p_{2\leadsto 8}^{01,0} + p_{2\leadsto 2}^{10,0} + p_{2\leadsto 8}^{10,0} + p_{2\leadsto 2}^{11,0} + p_{2\leadsto 8}^{11,0} = a_{2,3};
$$
$$
p_{2\leadsto 2}^{00,1} + p_{2\leadsto 8}^{00,1} + p_{2\leadsto 2}^{01,1} + p_{2\leadsto 8}^{01,1} + p_{2\leadsto 2}^{10,1} + p_{2\leadsto 8}^{10,1} + p_{2\leadsto 2}^{11,1} + p_{2\leadsto 8}^{11,1} = a_{2,4};
$$

FIG. 6.3 – Extrait du système de contraintes généré à partir de l'exemple

Enfin, si $\mathcal{A}$ est l'ensemble de toutes les arêtes dans le graphe de contrôle de flot et si $\mathcal{C}$ est l'ensemble des arêtes associées à un branchement conditionnel, le temps total d'exécution est donné par :

$$\boxed{T = \sum_{0 \leq i \leq 9} b_i \times t_{b_i} + \sum_{(i,j) \in \mathcal{A}} a_{i,j} \times t_{a_{i,j}} + \sum_{(i,j) \in \mathcal{C}} m_{i,j} \times p_{i,j}}$$

6.2 Résultats

Nous avons mesuré les temps d'exécution des blocs et des arêtes du graphe de flot de contrôle de notre exemple (avec branchements bien et mal prédits), en utilisant un simulateur au niveau cycle d'un processeur superscalaire de degré 2, avec exécution des instructions dans le désordre. Les résultats sont donnés ci-dessous :

$t_{b_0} = 11$	$t_{a_{0,1}} = -5$	$t_{a_{6,7}} = -7$
$t_{b_1} = 9$	$t_{a_{1,9}} = -6$	$t_{a_{7,1}} = -5$
$t_{b_2} = 13$	$t_{a_{1,8}} = -7$	
$t_{b_3} = 17$	$t_{a_{9,2}} = -5$	
$t_{b_4} = 24$	$t_{a_{2,3}} = -8$	$p_{1,9} = 4$
$t_{b_5} = 28$	$t_{a_{2,4}} = -8$	$p_{1,8} = 5$
$t_{b_6} = 10$	$t_{a_{3,2}} = -9$	$p_{2,3} = 6$
$t_{b_7} = 9$	$t_{a_{4,5}} = -13$	$p_{2,4} = 6$
$t_{b_8} = 10$	$t_{a_{4,6}} = -8$	$p_{4,6} = 6$
$t_{b_9} = 8$	$t_{a_{5,7}} = -7$	$p_{4,5} = 11$

Pour calculer la valeur maximale de T qui respecte les contraintes, nous avons utilisé l'outil `lp_solve` (`ftp.es.ele.tue.nl/pub/lpsolve/`) qui met en œuvre des techniques de programmation linéaire en nombres entiers.

Nous avons lancé le calcul en considérant trois cas : (a) toutes les pénalités sont identiques, égales à la pénalité maximale, soit 11 cycles ; (b) à chaque branchement est affectée une pénalité propre, égale à la pénalité la plus longue sur les deux branches ; (c) les pénalités sont celles mesurées pour chaque branche de chaque branchement. Les résultats sont les suivants :

pénalité unique pour tous les branchements	$T = 2831$ cycles
pénalité unique pour chaque branchement	$T = 2609$ cycles
pénalité mesurée pour chaque arête	$T = 2557$ cycles

On observe qu'une modélisation réaliste des pénalités d'erreur de prédiction permet d'accroître la précision du WCET estimé de 10,7%, ce qui est considérable.

On peut également vérifier que notre modèle estime les nombres de mauvaises prédictions de manière précise. Les nombres calculés par l'outil `lp_solve` sont :

branchement de la boucle externe (bloc 1)	$m_{1,9} = 2$	$m_{1,8} = 1$
branchement de la boucle interne (bloc 2)	$m_{2,3} = 21$	$m_{2,4} = 20$
branchement de la conditionnelle (bloc 4)	$m_{4,6} = 10$	$m_{4,5} = 10$

Considérons tout d'abord la boucle externe. Les résultats que nous obtenons sont conformes à ceux exposés démontrés dans [1] : la branche de sortie de la boucle $(1, 8)$ est toujours mal prédite si le nombre d'itérations de la boucle est supérieur à 2, tandis que la branche $(1, 9)$ est mal prédite au maximum deux fois (celà peut être moins, en fonction de la valeur d'intialisation du compteur).

Nous avons démontré dans la section 5 que le nombre maximal de mauvaises prédictions pour le branchement d'une boucle à nombre d'itérations variable imbriquée dans une autre boucle à M itérations est égal à $M + 1$ sur la branche entrant dans le corps de boucle et à M sur la branche sortant de la boucle. Les valeurs obtenues ici pour le branchement du bloc 2 sont cohérentes avec ce résultat. La boucle externe a un nombre d'itérations égal à 20 ($M = 20$), les bornes sont donc atteintes pour les deux branches ($m_{2,3} = 21 = M + 1$ et $m_{2,4} = 20 = M$).

Enfin, Colin et Puaut [2] ont analysé le comportement du branchement d'une structure conditionnelle située dans un corps de boucle de la manière suivante. Si une des deux branches est sensiblement plus longue que l'autre, elle sera empruntée à chaque itération au sein du chemin le plus long : dans ce cas, le branchement ne peut être mal prédit que deux fois au maximum (selon la valeur initiale du compteur). Au contraire, si les deux branches sont de longueur équivalente, le pire cas est celui où elles sont empruntées en alternance : le branchement est alors mal prédit à chaque itération. Notons que, alors que Colin et Puaut ont établi une formule liant les temps d'exécution des deux branches permettant de trancher entre les deux cas, nous laissons le soin de l'analyse à l'outil d'optimisation. Les résultats de mesure que nous avons obtenus sur notre exemple nous situent dans le second cas (branches de longueur équivalente), et le pire cas correspond à une exécution de chacune des deux en alternance, avec erreur de prédiction à chaque fois. Nous avons cependant testé le premier cas en modifiant les temps d'exécution dans le modèle, et nous avons alors obtenu un total de deux erreurs de prédiction.

Chapitre 7

Conclusion et perspectives

Les prédicteurs de branchement dynamiques compliquent le calcul du temps d'exécution pire-cas de programmes basé sur une analyse statique du code. Quelques solutions ont toutefois été proposées dans la littérature pour les modéliser. Cependant, la prise en compte des conflits éventuels dans les tables de prédiction augmente de façon importante la complexité des modèles et le temps de calcul du WCET. Nous proposons de mettre en place des heuristiques permettant d'empêcher les conflits en contrôlant le mode de prédiction (statique ou dynamique) de chaque branchement. Le processeur IA-64 d'Intel fournit le support nécessaire à une telle approche et peut donc permettre d'activer la prédiction de branchement de manière sûre dans un contexte temps-réel.

Nous avons aussi mis en évidence qu'il est nécessaire d'exprimer le coût des erreurs de prédiction de manière précise. Nous avons présenté quelques mesures montrant que la pénalité de mauvaise prédiction peut sensiblement varier d'un branchement à l'autre, mais aussi, pour un branchement donné, d'une branche à l'autre. Il en découle que, pour calculer le WCET de manière précise, il est nécessaire de borner séparément les nombres de mauvaises prédictions sur les deux branches d'un branchement conditionnel.

Enfin, nous avons montré comment on peut modéliser la prédiction de branchement dans le cadre d'un calcul de WCET par la méthode IPET. L'intérêt de cette méthode est de pouvoir traiter des codes binaires optimisés et pas forcément bien structurés. Grâce à l'élimination des conflits, nous avons étendu des travaux récents de Mitra et Roychoudhury [6] à des algorithmes de prédiction basés sur des compteurs 2 bits, qui sont les plus courants. Nous avons aussi étendu l'étude de Bate et Reutemann [1] pour établir une borne du nombre de mauvaises prédictions sur chacune des branches d'un branchement conditionnel "de boucle" dans le cas général.

Nos perspectives de travail portent sur la synthèse d'un algorithme de répartition des branchements sur chacun des prédicteurs de branchement (statique et dynamique).

Bibliographie

[1] I. Bate and R. Reutemann. Worst-case execution time analysis for dynamic branch predictors. In *16th Euromicro Conference on Real Time Systems*, pages 215–222, june 2004.

[2] A. Colin and I. Puaut. Worst case execution time analysis for a processors with branch prediction. In *Real-Time Systems*, pages 249–274, may 2000.

[3] J. Engblom. Analysis of the execution time unpredictability caused by dynamic branch prediction. In *9th IEEE Real-Time and Embedded Technology and Applications Symposium*, may 2003.

[4] Y.-T. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *ACM SIGPLAN Notices*, volume 30, 1995.

[5] D. Lindsay. *Static Methods in Branch Prediction.* PhD thesis, Dept. of CS, University of Colorado, 1998.

[6] T. Mitra and A. Roychoudhury. A framework to model branch prediction for wcet analysis. In *WCET workshop held in conjunction with Euromicro Conference on Real-Time Systems*, june 2002.

[7] H. Patil and J. Emer. Combining static and dynamic branch prediction to reduce destructive aliasing. In *6th International Symposium on High-Performance Computer Architecture*, page 251, january 2000.

[8] J. Smith. A study of branch prediction strategies. In *8th annual ACM/IEEE International Symposium on Computer Architecture*, 1981.

[9] T.-Y.Yeh and Y. Patt. Alternatives implementations of two-level adaptative branch prediction. In *19th International Symposium on Computer Architecture*, 1992.