République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université M'hamed Bougara de Boumerdès

Faculté des Sciences

Département d'Informatique

Thèse Pour l'obtention du diplôme de Magister en Informatique

Option :"Informatique Fondamentale"

Présenté et soutenu publiquement par **Mourad Bouache**

▷ **Thème**

Amélioration de la performance des processeurs généralistes par une exécution distribuée des instructions

▷ ***Jury***

M. S Niar, Professeur Université Valenciennes, Président
M. B Goossens, Professeur Université de Perpignan, Encadreur
M. M Mezghiche, Professeur Université de Boumerdès, Co-Encadreur
M. M Koudil, Docteur Institut National d'Informatique, Examinateur

Remerciements

Je remercie le Professeur Bernard Goossens d'avoir dirigé mon travail et de m'avoir intégré dans son équipe 'DALI' du laboratoire LP2A. Travailler avec cette équipe a été pour moi un grand plaisir.

Je remercie de même le Professeur Mohamed Mezghiche, d'avoir acceptés d'être co-encadreur de ma thèse.

Je remercie aussi David Parello de m'avoir aidé pendant la phase de simulation.

Je tiens à remercier tout particulièrement M. Smail Niar pour ses nombreux conseils et je suis content qu'il ait accepté de faire partie de mon jury de thèse.

Je remercie tous les membres de mon jury de thèse, M.Niar et M.Koudil de me faire l'honneur d'assister à ma soutenance.

Je remercie de même Madame ZERARI, Directrice de la bibliothèque Universitaire de l'UMBB pour son aide et support.

Mes remerciements vont aussi à M. Belahcene pour son aide.

Dédicaces

Je dédie ce mémoire aux êtres qui me sont les plus chers,

A la mémoire de mon père.

A ma mère, de m'avoir supporté et aidé, sans oublier mes frères Mohamed et Farid, ma soeur Meriem, mes camarades au département informatique et tout mes collègues à la bibliothèque universitaire de Boumerdès.

A Madame ZERARI, de m'avoir beaucoup aidé.

Et à mes amis de l'UMBB et de ma ville Cherchell qui ont tous été de sympathiques compagons de travail.

Table des matières

Table des figures

Liste des tableaux

Résumé

Le banc de registres avec ses ports d'accès et le mécanisme de renvoi sont les ressources critiques parmi d'autres du chemin de données du processeur. Notre étude évalue l'impact de l'augmentation du degré superscalaire sur le flux des données manipulées par le processeur. Nous montrons que la réduction du nombre de ports d'accès au banc de registres, sans réorganisation de la lecture des registres sources et de l'écriture des destinations, est peu efficace. Les registres sources prêtes au renommage, critiques pour la performance, doivent être transmises par le chemin le plus direct possible c'est-à-dire le réseau de renvoi. Les autres registres sources sont moins critiques pour la performance. Leur chemin peut transiter par le banc de registres.

Mots-clés : *Micro-architecture, processeur superscalaire, exécution en désordre, banc de registres, simulation*

Abstract

The register file with its wearing of access and the mechanism of reference are the cretical resources of processor's data way.

This thesis evalutes the impact of the increase in the superscalar degree on the dat flow handled by the processor.

We show that, the reduction of the access's wearing number to the register file, without reorganization of the reading sources and results writing, is not very effective , it's necessary to preserve at least 2 ports per instruction not to degrade the performance.

We show that the number of ready sources with the renaming can decrease hen the superscalar degree increase ; making the access early to the sources as much less productive than the degree is high.

These sources, critical for the performance, must be transmitted by the way most direct possible is with to say the reference network.

Chapitre 1

Introduction

Le microprocesseur est le cœur des ordinateurs. Fruit de plus 50 années de recherche et de développement, le processeur détermine la performance des machines numériques actuelles. À l'origine outils de calcul scientifique, les ordinateurs et donc les processeurs sont devenus des produits « grand public » avec l'explosion des jeux vidéo et des applications multimédia.

L'ordinateur est constitué d'une partie matériel et logiciel, cette dernière peut être composé d'un seul ou une suite de programmes. Un programme est constitué d'instructions qui agissent sur des données. Ces instructions sont assemblées, par le compilateur, suivant un schéma séquentiel et leur exécution s'effectue à travers le processeur qui les traite, en apparence, l'une après l'autre. En réalité, la méthode d'exécution a considérablement évolué. À l'origine, les processeurs étaient d'une architecture simple et le procédé de fabrication limitait la densité d'intégration. En 1971, Intel introduit le 4004 comme première solution intégrant toutes les fonctions d'un processeur sur une seule puce.

Le 4004 intégrait 2300 transistors et fonctionnait à 740 KHz. Trente ans plus tard, les processeurs de la dernière génération intègrent 120 millions de transistors et fonctionnent à 3 GHz. La "loi de Moore" qui prédisait dès 1965 un doublement du nombre de transistors tous les 18 mois s'est vérifiée jusqu'à présent.

En d'autres termes, le gain en vitesse de cadencement sur 30 ans est d'un facteur supérieur à 3000 (de 1Mhz à 3Ghz). On pourrait croire, à tort, que cela traduit directement l'amélioration de la performance. En réalité, outre la vitesse de cadencement, la performance d'un processeur est aussi proportionnelle au travail qu'il effectue dans un cycle : en moyenne un dixième d'instruction exécutée par le 4004, contre près de deux instructions pour les récents Pentium 4[24]. Le facteur 3000 doit donc être multiplié par un facteur supérieur à 10, ce qui donne une progression de la performance des processeurs sur 30 ans de l'ordre de 50000.

Cette amélioration constante des performances est due d'une part à l'évolution des procédés de fabrication (en gros, le facteur 3000), et d'autre part à l'amélioration de l'organisation interne du processeur -sa micro-architecture- (en gros le facteur 10). Au vu de ces deux facteurs, on pourrait penser que le progrès technologique a été bien plus important que le progrès architectural. Il n'en est rien. D'une part, les améliorations architecturales se sont constamment ajoutées aux améliorations technologiques malgré la difficulté croissante de faire tenir dans un cycle de plus en plus court des fonctions de plus en plus complexes. D'autre part, une partie non négligeable du facteur 3000 vient d'améliorations architecturales, qui ont réduit le cycle au-delà de ce qu'auraient permis les seules avancées technologiques : au cours des deux dernières décennies, les processeurs ont été pipelinés, avec des profondeurs de pipelines qui n'ont cessé de s'accroître, passant de deux étages en 1980 à plus de vingt sur un Pentium 4 du début de la décennie 2000.

La performance d'un processeur s'exprime par la formule $n/i * c$ qui donne le temps de calcul d'un programme. Dans cette formule, n est le nombre total d'instructions exécutées pour le programme. La variable i est l'IPC (nombre d'instructions exécutées par cycle). Le rapport n/i est le nombre de cycles pour exécuter le programme. Enfin, la variable c est la durée du cycle. On améliore la performance en réduisant le temps de calcul, c'est-à-dire en réduisant n, en augmentant i et en réduisant c.

Le facteur i a été l'objet de toutes les attentions des architectes depuis les années 80. Mesuré aux alentours de 0,2 pour les processeurs CISC[1], il a fait un bond spectaculaire à 0,8 avec l'introduction des architectures RISC et des caches. Le jeu d'instructions réduit a aussi eu pour effet de dégrader le facteur n qui a été augmenté de 30%.

Dans les années 90, les architectures superscalaires de degré 2 ont permis à i d'atteindre 1. Ensuite, l'exécution en désordre et la spéculation dans les architectures superscalaires de degré 4 ont encore fait progresser i jusqu'à une valeur voisine de 2 aujourd'hui.

Parallèlement, le facteur c s'est aussi amélioré, au gré du découpage de plus en plus fin du pipeline et des améliorations technologiques (diminution de la finesse de gravure, qui se traduit par une diminution des distances à parcourir par les signaux, donc de leur délai).

Néanmoins, on constate depuis maintenant trois ou quatre ans que d'une part, le cycle des processeurs a cessé de se réduire (on est scotché entre 3Ghz et 4Ghz depuis 2004) et d'autre part, le degré superscalaire a lui aussi cessé d'évoluer (il est de 6 micro-instructions RISC par cycle depuis l'avènement du Pentium 4 en 2001). Le cycle n'évolue plus principalement parce qu'il est désormais devenu impossible de découper le pipeline plus finement. Seule la réduction de la finesse de gravure des composants (le pro-

[1] Complex Instruction Set Computer

cédé de fabrication) peut améliorer la vitesse des circuits. Le degré superscalaire n'évolue pas non plus parce que passer à huit instructions par cycle nécessite d'opérer de profonds aménagements micro-architecturaux.

L'objet du travail rapporté dans ce mémoire est de mesurer l'impact de l'augmentation du degré superscalaire sur le flux des données manipulées par le processeur. Les micro-architectures d'aujourd'hui sont bâties autour d'un traitement uniforme des instructions perçues comme des unités atomiques regroupant deux sources, une destination et une opération. En multipliant le nombre d'instructions traitées simultanément, on s'oblige à disposer de ressources suffisantes pour à chaque cycle lire les sources des instructions lancées et écrire les résultats des instructions terminées. Le modèle micro-architectural actuel conduit à lire deux sources par instruction et à écrire un résultat par opérateur, ce qui requiert $3o$ ports d'accès sur le banc de registres, o étant le nombre d'opérateurs du processeur. Notre étude doit permettre de quantifier les ressources réellement utilisées lors d'une exécution.

La première partie de ce mémoire expose l'état de l'art des processeurs généralistes.

La deuxième partie est consacrée à la description de la démarche expérimentale des architectes utilisée pour quantifier leurs propositions, basée sur une simulation appliquée à un jeu de programmes tests (benchmarks). Conjointement, nous y présentons le modèle de processeur dont nous nous sommes servi pour effectuer nos expériences qui est une adaptation du processeur PISA (Portable Instruction Set Architecture) implémenté dans le simulateur simplescalar.

La troisième partie est constituée d'un article dont nous sommes coauteur, accepté à la conférence Sympa'06 [25]. Cet article rapporte les résultats d'une étude que nous avons menée au sein de l'équipe DALI de l'université de Perpignan au cours du mois de février 2006.

Chapitre 2

Micro-architecture des processeurs généralistes

2.1 Le chemin de données d'un processeur à exécution en désordre

L'architecture superscalaire, pipelinée, comprend plusieurs unités de calculs spécialisées. Il s'agit d'exécuter l'ensemble des instructions du langage machine, à savoir des opérations sur les entiers et les flottants, mais aussi des sauts et des accès à la mémoire pour charger et ranger les registres. Une architecture superscalaire permet d'employer ces opérateurs en parallèle en exécutant simultanément plusieurs instructions. Le nombre d'instructions que peut exécuter le processeur en parallèle est son degré superscalaire.

Le nombre moyen d'instructions qu'on peut exécuter en parallèle dans un programme est son degré de parallélisme d'instructions ou ILP[1]. Si le degré superscalaire est supérieur à l'ILP, des ressources du processeur sont inutilisées. Si le degré superscalaire est inférieur à l'ILP, du parallélisme n'est pas exploité par le processeur par manque de ressources.

On fait apparaître beaucoup d'ILP en autorisant une exécution des instructions en désordre. Un calcul se compose d'instructions qui forment de petites chaînes d'opérations. Prise dans une telle chaîne, une instruction est le plus souvent dépendante de celle qui la précède. Pour autant, plusieurs chaînes indépendantes peuvent être exécutées en parallèle à condition d'organiser le processeur pour qu'il exécute le programme selon le flot des données plutôt que selon le flot des instructions. Une exécution en séquence limite l'ILP des programmes à moins de deux instructions par cycle, ce qui en pratique borne le degré superscalaire des processeurs à deux. Les processeurs d'aujourd'hui, qui sont de degré superscalaire voisin de quatre,

[1] ILP : Instruction Level Parallelism

s'appuient tous sur une micro-architecture à exécution en désordre. L'ILP des programmes est dans ce cas de plusieurs dizaines d'instructions par cycle[16], ce qui laisse une bonne marge de progression au degré superscalaire des processeurs.

Exécuter huit, seize ou même plus d'instructions par cycle est théoriquement possible si l'on s'en réfère à l'ILP mesuré dans les programmes de la suite SPEC[2]. Cependant, les défis à relever sont nombreux. Cela commence par la lecture même des instructions : comment récupérer de la mémoire où le programme est conservé, huit, seize ou plus d'instructions chaque cycle, sachant que le code exécuté est truffé d'instructions de sauts conditionnels (en moyenne une instruction sur huit) ? Ces ruptures de séquence font de la lecture du code un parcours très chaotique de la mémoire d'instructions. Le cache de traces, proposé en 1996 [28], et introduit dans le Pentium 4 en 2001, est une avancée majeure dans ce domaine : en plus de cacher le code statique, on cache aussi le code exécuté par le processeur. Un prédicteur de sauts prédit le chemin qui sera suivi à l'exécution. Le processeur lit le cache de traces et en retient ce que lui conseille son prédicteur de sauts. En moyenne, un tel dispositif (cache de traces [6]et prédicteur[7]) qui lit des traces de seize instructions fournit au processeur une douzaine d'instructions sur le chemin d'exécution tous les cycles.

Une fois les instructions dans le processeur, il faut en déterminer l'ordre partiel d'exécution. Cet ordre est fixé par les dépendances de données : une instruction *i* doit s'exécuter avant une instruction *j* si le résultat de *i* est une source de l'instruction *j*. Le modèle RISC[3] des jeux d'instructions des processeurs depuis les années 80 est basé sur les registres. Les registres sont un intermédiaire entre les unités de calcul du processeur et sa mémoire. On calcule en lisant les sources de l'opération en registres et en écrivant son résultat en registre. Si les sources ne sont pas en registres, il convient de les y charger depuis la mémoire.

Au besoin, il faut faire de la place, le nombre de registres étant limité (on fait de la place en rangeant un registre en mémoire, le libérant ainsi pour lui permettre d'héberger une source manquante).

Le modèle à registres a ses avantages : le code machine des instructions est compact (un numéro de registre tient sur quelques bits alors qu'une adresse mémoire prend quelques octets). La dépendance des instructions est apparente dans le code lui-même : l'instruction *j* dépend de *i* si et seulement si le registre résultat de *i* est un des registres sources de *j*. Il a aussi ses inconvénients : le compilateur mappe l'ensemble des variables du programme sur l'ensemble (très réduit) des registres de l'architecture. Ainsi, un registre est réemployé en permanence pour des données successives tout au long de l'exécution du programme. Par exemple, au sein d'une

[2]SPEC : Standard Performance Evaluation Corporation

[3]RISC : Reduced Instruction Set Computer

boucle, chaque registre de destination du corps de boucle reçoit successivement autant de valeurs que d'itérations. Cela crée de fausses dépendances entre les instructions : l'instruction *j* peut dépendre de l'instruction *i* parce que les deux instructions écrivent leur résultat dans le même registre. Cette dépendance empêche une exécution en parallèle des deux instructions : l'instruction j doit attendre que toute instruction *k* entre *i* et *j*, dépendant de *i* (*dépendance de donnée*s), ait lu le résultat de *i*.

Le renommage des registres permet au processeur d'étendre l'ensemble des registres architecturaux, éliminant ainsi les fausses dépendances : deux instructions *i* et *j* écrivant dans le même registre architectural, se voient attribuer par le processeur deux registres distincts de destination. Ainsi, les instructions *i* et *j*, indépendantes par ailleurs, peuvent être exécutées en parallèle.

L'algorithme d'exécution en désordre est le suivant :

- Renommer chaque instruction l'une après l'autre. Pour chaque destination, on alloue un registre de renommage libre. Pour chaque source, on détermine la destination dont elle dépend, donc le registre de renommage à lire.
- Parmi les instructions renommées, certaines attendent une source encore en calcul. Les autres sont en compétition pour l'obtention d'une unité de calcul. Toutes celles qui obtiennent une unité démarrent leur exécution.
- De chaque unité de calcul est issu, à chaque cycle, un résultat. Ce résultat est reporté dans son registre de destination et propagé aux sources dépendantes.

Ainsi, le mécanisme de renommage emmagasine dans la fenêtre de lancement des instructions renommées qui attendent leurs données pour pouvoir s'exécuter. Les instructions dont les données sont disponibles s'exécutent. Leurs résultats fournissent les données aux instructions suivantes. L'ordre d'exécution des instructions dans le processeur est contrôlé par le flot des données.

2.2 Les registres

Le banc de registres a augmenté en surface au fur et à mesure des améliorations micro-architecturales. Cet accroissement tient à trois facteurs. Tout d'abord, le mécanisme de renommage, introduit dans le Pentium III, implique une extension du nombre de registres du processeur. Par exemple, un Pentium 4 définit huit registres architecturaux étendus à 128 registres de renommage. Ensuite, le chemin de données des processeurs généralistes est en train de migrer de 32 à 64 bits, ce qui entraîne un doublement de la taille des registres. Enfin, 18 ports d'accès sont nécessaires sur le banc de registres pour autoriser le lancement de six micro-instructions RISC

par cycle (les ports sont les voies d'accès pour lire les sources -deux par micro-instruction- et écrire les résultats -un par micro-instruction-). Ainsi, le banc de huit registres architecturaux d'un processeur 8086 des années 80 contenait-il 16 fois moins de registres que le banc de 128 registres de renommage du Pentium 4 d'aujourd'hui. Chaque registre regroupait huit bits de données, contre 64 aujourd'hui, soit pour le banc, 128 fois plus de bits conservés. Enfin, on accédait à ces registres par deux ports contre 18 pour le pentium 4. Ainsi, la surface d'un bit a-t-elle été multipliée par 81 (elle est proportionnelle au carré du nombre de ports, ce qui est expliqué deux paragraphes plus loin).

Au total, le banc de registres du 8086 occupait une surface 10000 fois ($128 * 81$) moins grande (à finesse de gravure constante) que celle du banc du pentium 4. Or, le temps d'accès au banc est proportionnel à sa surface (le délai d'un signal est proportionnel au carré de la distance qu'il parcourt). Si bien qu'avec un cycle très court, il faut deux cycles au Pentium 4 pour accéder à ses registres.

L'augmentation du degré superscalaire du processeur passe par un accroissement proportionnel du nombre de ports : pour exécuter *n* instructions par cycle, il faut disposer de *3n* ports pour lire *2n* sources et écrire *n* résultats par cycle. Chaque port, c'est-à-dire un transistor d'accès, emploie une ligne et une colonne de la grille d'un composant. L'ajout d'un port se traduit par l'occupation d'une ligne et d'une colonne supplémentaire. Le schéma de la figure 2.1 fait apparaître que la surface des voies d'accès à chaque bit du banc de registres évolue en fonction du carré du nombre de ports (*n* lignes x *n* colonnes pour *n* ports), lui-même fonction linéaire du degré superscalaire (*3n* ports pour un degré *n*).

Egalement, pour maintenir un IPC à *n*, il faut chercher chaque cycle *n* instructions indépendantes. Plus on se projette en avant dans le code à exécuter, plus on récolte de telles instructions (il faut enjamber les instructions formant une chaîne pour atteindre le début de la chaîne suivante). Or, à chacune des instructions rencontrées, il faut attribuer, par renommage, un registre unique de destination. Par conséquent, le nombre de tels registres augmente avec le degré superscalaire. Si une instruction conserve le registre de renommage qui lui a été attribué en moyenne c cycles, et qu'on renomme en moyenne n instructions chaque cycle, il faut au moins $n * c$ registres de renommage.

Au total, la surface du banc de registres est proportionnelle à $c*n^3$ pour un degré superscalaire *n*. Par conséquent, doubler le degré superscalaire multiplie la surface du banc de registres au moins par 8. On en déduit par exemple qu'un Pentium de degré superscalaire 8 accèderait à chacun des 24 ports de son banc de 256 registres en six ou sept cycles.

L'objet de notre étude est de déterminer quelles sont les ressources réellement nécessaires pour que le banc de registres puisse délivrer leurs sources aux *n* instructions lancées à chaque cycle et pour qu'il puisse réceptionner

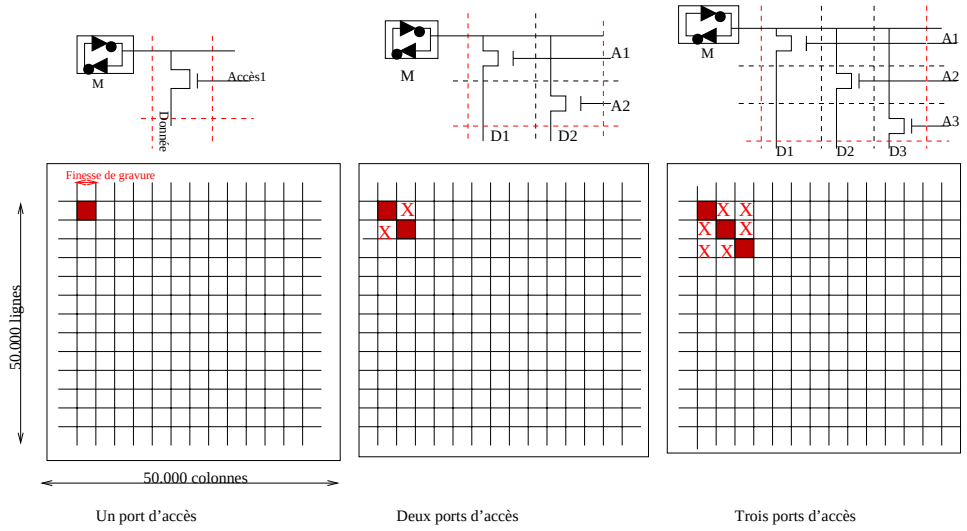


FIG. 2.1 – Structure d'un registre

leurs résultats. On peut remarquer que les instructions du langage machine n'emploient pas toutes deux sources et une destination. Par exemple, un saut conditionnel est une instruction sans destination. Le résultat du calcul n'est pas rangé dans un registre, mais sert uniquement à déterminer la direction du saut. De même, un calcul qui fait intervenir une constante, comme une incrémentation, ne lit pas deux sources en registre mais une seule. On voit que, loin de nécessiter *3n* ports d'accès au banc de registre pour exécuter *n* instructions par cycle, un processeur peut se contenter du nombre de ports correspondant aux seules lectures de sources et écriture de résultat en registres.

2.3 Le mécanisme de renvoi et le réveil des instructions

Parmi d'autres, les ressources critiques du chemin de données sont le banc de registres avec ses ports d'accès, le mécanisme de renvoi et la logique de réveil avec ses comparateurs.

Le mécanisme de renvoi consiste à transmettre un résultat directement de la sortie de l'unité de calcul qui l'établit aux sources des instructions qui en dépendent. Ce renvoi permet d'éviter, pour les instructions concernées, qu'elles lisent leur source dans le banc de registres après que celle-ci y ait été écrite. Dans un processeur scalaire (avec un seul opérateur), le chemin de la donnée, au lieu de passer par le banc de registres, prend un raccourci en reliant directement la sortie de l'opérateur à son entrée. Une instruction i, ayant calculé une valeur v à destination du registre d, renvoi cette valeur

vers l'entrée de l'opérateur. Supposons que l'instruction j, lancée lors du même cycle, ait pour source ce même registre d. Une fois lancée, elle lit une valeur v' périmée dans le registre d, le résultat v n'ayant pas encore pu y être écrit. L'instruction j reçoit la valeur à jour v émise par le renvoi. La valeur v se substitue à la valeur v' puisqu'elles concernent un seul et même registre d.

Le matériel utilisé pour ce chemin raccourci est un multiplexeur (pour choisir entre v et v') et un comparateur (pour comparer la destination de i à la source de j et contrôler le multiplexeur). Puisque les deux sources de j peuvent être concernées par le renvoi de v, le matériel nécessaire comprend deux multiplexeurs à deux voies et deux comparateurs de numéros de registres. Si l'exécution en désordre doit être permise, il faut en outre programmer le lancement de l'instruction j pour le cycle de sortie du résultat de l'instruction i. Le lancement de i vers un opérateur de délai constant c implique celui de j, si son autre source est disponible, c cycles plus tard.

En passant à une micro-architecture superscalaire, on multiplie les opérateurs, donc les résultats émis par le renvoi et les instructions lancées, donc les comparateurs et les multiplexeurs. Pour d opérateurs, soit d instructions terminées et d instructions lancées, donc d résultats et $2d$ sources, il faut $2 * d^2$ comparateurs et $2d$ multiplexeurs à $d + 1$ voies. Le réseau de comparateurs croît très vite et la taille des multiplexeurs rallonge sérieusement le chemin de données.

Le mécanisme de renvoi n'est pas indispensable à l'algorithme d'exécution en désordre. On peut simplement notifier qu'une donnée est prête en diffusant les numéros des registres de destination des résultats aux instructions en attente. Dans ce cas, toutes les sources sont lues en registre, après que les résultats dont elles dépendent y aient été écrits. Néanmoins, cette façon de faire rallonge le chemin des données d'instructions dépendantes de deux fois le nombre de cycles pour accéder aux registres (sur le pentium 4, cela fait 4 cycles de surcoût, ce qui est considérable).

Notre étude doit aussi quantifier ce qui est nécessaire en matière de renvoi. On peut par exemple séparer les renvois critiques de ceux qui ne le sont pas. Un renvoi est critique s'il transmet à une instruction son ultime source avant son lancement. Les renvois ne concernent que les sources en registres. Enfin, certains opérateurs ne renvoient pas de résultat (par exemple, les opérateurs de traitement des sauts).

Chapitre 3

Simulation

3.1 Le simulateur *Simplescalar*

3.1.1 Introduction

Les processeurs actuels sont de plus en plus compliqués car ils utilisent des techniques très évoluées afin d'avoir de meilleures performances. Cependant, cette complexité fait qu'il est plus difficile de les tester et de comprendre leur fonctionnement. De plus, il est impossible d'effectuer des modifications sur le processeur afin de tester des concepts nouveaux. C'est pour cela que des simulateurs ont été créés. (tels que SPIM [1], SimOS [2] ou SimpleScalar [3]). Ces simulateurs permettent d'implémenter de nouveaux concepts et de les tester. Le simulateur SimpleScalar[4] a été créé également dans cette optique. L'architecture utilisée dans SimpleScalar 3.2 est proche de celle utilisée par MIPS.

Il est de plus possible d'utiliser le compilateur GCC5, ainsi que les utilitaires qui lui sont associés, afin de générer des programmes pour SimpleScalar.

Les principaux avantages de SimpleScalar sont sa flexibilité, sa portabilité, son extensibilité et ses performances. Le simulateur se décline en plusieurs versions effectuant une simulation de plus en plus détaillée. La version la plus complexe est *sim-outorder* qui effectue une exécution dans le désordre du programme simulé.

3.1.2 Architecture logicielle de *Simplescalar*

Simplescalar permet de concevoir un simulateur fonctionnel de processeur, c'est-à-dire pouvant exécuter un programme instruction après ins-

[1] http :/ / www.es.wisc.edu / larus / spim.html
[2] http :/ / simos.standford.edu /
[3] http :/ / www.simplescalar.com /
[4] Le simulateur peut être téléchargé à l'adresse http :/ / www.simplescalar.com /

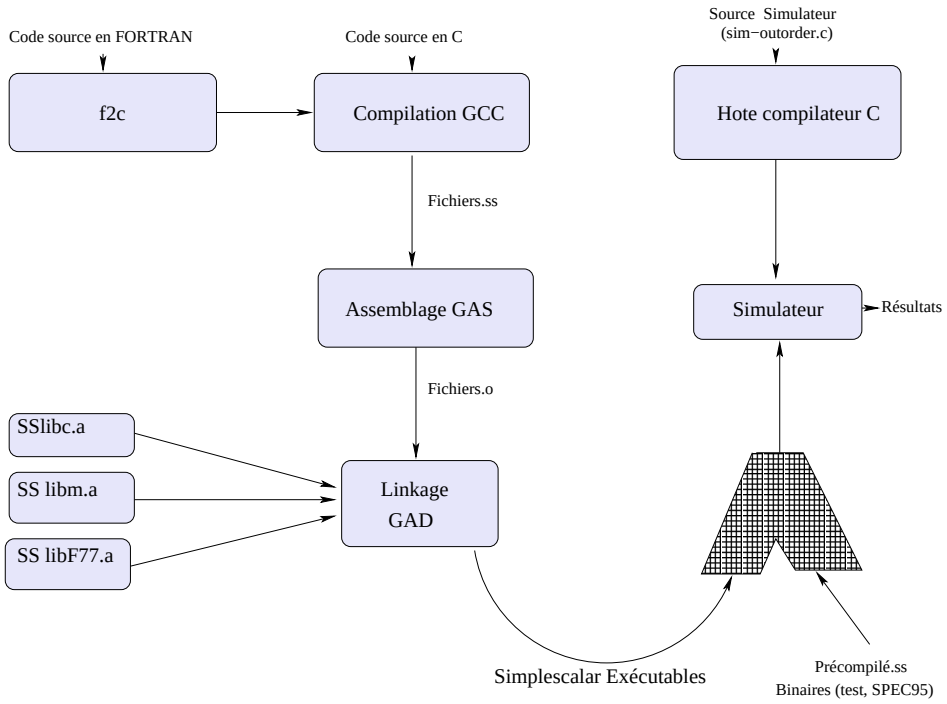


FIG. 3.1 – Vue synoptique de Simplescalar

truction en réagissant de la même manière qu'une version « réelle ». En d'autres termes, il permet de simuler le comportement interne du processeur et non seulement la partie interprétation d'un jeu d'instruction donné. Ainsi, il est possible de construire des architectures en spécifiant avec précision leur fonctionnement.

La figure 3.1 montre comment une application (source en Fortran ou en C) est compilée en binaire PISA et liée avec les librairies SimpleScalar avant d'être exécutée pas à pas par le simulateur.

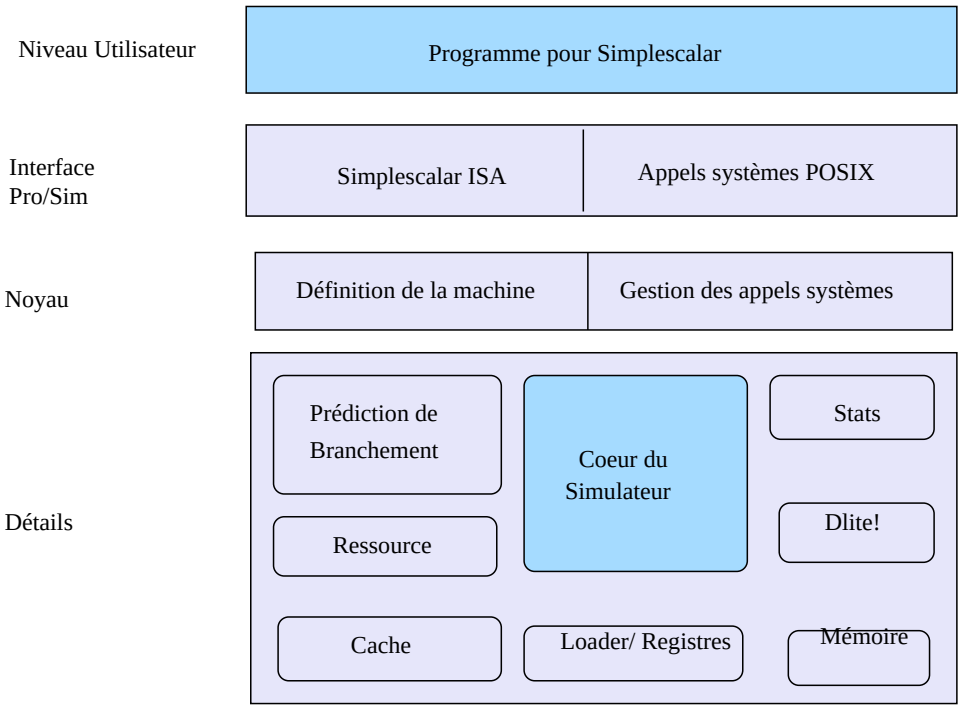


FIG. 3.2 – Architecture de *Simplescalar*

La figure 3.2 présente l'architecture des différents simulateurs SimpleScalar. Les appels systèmes POSIX sont exécutés mais non simulés (l'exécution permet d'obtenir l'effet souhaité pour le programme simulé mais le code des appels systèmes ne fait pas partie de la simulation).

Le simulateur décode les instructions PISA à partir de la définition de la machine PISA. La simulation de chaque instruction décodée par le cœur du simulateur met en œuvre, selon le simulateur choisi, diverses ressources parmi lesquelles les caches, la prédiction de sauts, le débogage avec Dlite !, la mémoire ou l'affichage de statistiques de la simulation.

3.1.3 Le fonctionnement des différents simulateurs

Simplescalar se compose de 8 simulateurs allant du plus simple *sim-fast* (et le plus rapide) au plus élaboré qui simule une exécution en désordre et spéculative *sim-outorder*.

3.1.3.1 Emulation : *sim-fast et sim-safe*

Ces deux simulateurs implémentent le minimum de fonctions. Le premier et le plus rapide, *sim-fast*, est implémenté dans *sim-fast.c*. Il possède très peu de fonctions. Il exécute les instructions séquentiellement. Sim-fast

est optimisé pour exécuter les instructions rapidement et ne possède pas de cache et ne supporte pas le débugger *DLite*.

Une autre version de sim-fast se nomme *sim-safe*. La différence entre ces deux versions est que sim-safe vérifie que les instructions sont alignées et pour chaque accès à la mémoire, la permission est vérifiée.

3.1.3.2 Simulation de cache : *sim-cache* et *sim-cheetah*

SimpleScalar est distribué avec deux versions gérant les caches : *sim-cache* et sim-cheetah. Ces deux simulateurs utilisent le fichier *cache.c* et ils utilisent respectivement les fichiers *sim-cache.c* et sim-cheetah.

Ils permettent d'obtenir des statistiques sur l'utilisation des hiérarchies mémoires, sans pour autant nécessiter une simulation fine du programme exécuté.

3.1.3.3 Profilage : *sim-profile*

Le simulateur sim-profile permet de produire des informations concernant l'exécution d'un programme. Il peut générer des statistiques détaillées sur les adresses des instructions, les accès mémoires, les branchements, etc.

3.1.3.4 Exécution dans le désordre : *sim-outorder*

C'est le simulateur le plus complet et le plus compliqué (il fait plus de 4000 lignes). Ce simulateur est capable de lancer et d'exécuter des instructions dans le désordre et d'effectuer de la prédiction de sauts et de l'exécution spéculative.

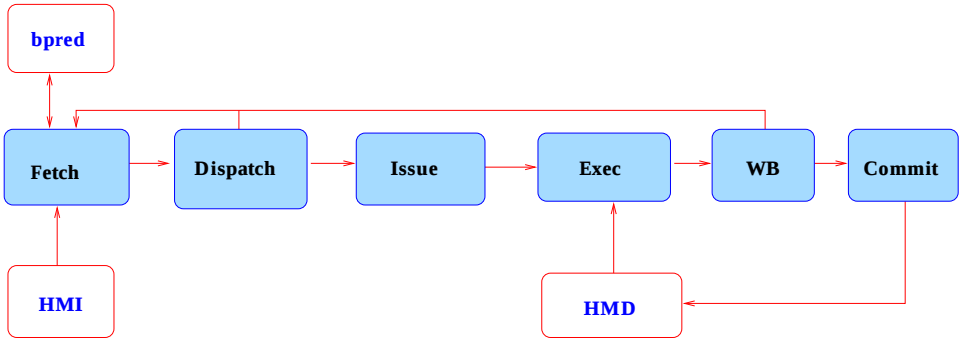


FIG. 3.3 – Pipeline Simplescalar

La boucle principale du simulateur (fonction sim_main()) est définie ainsi :

```
ruuinit();
```

```
for (;;){

ruucommit();

ruureleasefu();

ruuwriteback();

lsqrefresh();

ruuissue();

ruudispatch();

ruufetch(); }
```

Cette boucle est exécutée une fois par cycle d'horloge. On pourra remarquer que le pipeline, par l'appel des fonctions, progresse au cours du cycle du dernier étage vers le premier. Cela force l'attente du cycle suivant pour passer une donnée d'un étage à son successeur, ce qui correspond au fonctionnement réel du pipeline.

3.1.4 Pipeline simplescalar

La figure 3.3 nous montre le pipeline Simplescalar, chaque opération est décomposée en étapes :

- L'étape *ruu_fetch()* est la partie du pipeline qui charge les instructions. Cette fonction extrait les instructions pointées par le compteur de programme. Elle prédit les sauts et fixe le compteur de programme suivant. À chaque cycle, les instructions sont lues à partir d'une seule ligne du cache d'instructions. Après que les instructions ont été lues, elles sont placées dans une liste qui sera utilisée par *ruu_dispatch()*.
- La partie du simulateur qui se charge de répartir les instructions se situe dans la fonction *ruu_dispatch()*. C'est ici que le décodage et le renommage des registres est effectué. Les instructions décodées sont celles figurant en tête de la liste alimentée par l'étage de *fetch*. Une fois décodées, les instructions sont placées dans une file d'ordonnancement. Les dépendances sont établies (les sources de l'instruction s'ajoutent aux listes de successeurs des registres qu'elles référencent et la destination crée une liste de successeurs vide).
- Une file des instructions prêtes est alimentée par l'étage *ruu_writeback()* (voir ci-dessous). Le lancement des instructions prêtes est effectué dans les fonctions *ruu_issue()* (instructions générales) et *lsq_refresh()*

(instructions de chargement). Ces fonctions s'occupent de lancer les instructions dans les unités fonctionnelles. Le simulateur se contente de calculer, pour chaque instruction, le délai de son opération. Ce délai est établi à partir de la latence des unités fonctionnelles. Une fois lancée, une instruction s'insère, selon son délai, dans la liste triée des instructions lancées.

– La fonction *ruu_writeback()* parcours la tête de la liste des instructions lancées en terminant les instructions dont le délai est écoulé. Le résultat de chaque instruction terminée est propagé aux sources qui en dépendent. Les instructions ayant toutes leurs sources disponibles sont placées dans la file des instructions prêtes.
– La fonction *ruu_commit* reçoit les instructions terminées et provenant de l'étage *ruu_writeback()*. Elle traite dans l'ordre les instructions de sauts et de rangements. Pour les sauts, la prédiction faite à l'extraction est vérifiée et le cas échéant, corrigée (un compteur de programme corrigé est transmis à l'étage d'extraction).

3.2 Présentation des benchmarks

Tous les résultats présentés dans cette étude sont issus des benchmarks MediaBench[8] et MiBench.

Les *benchmarks* sont des programmes qui permettent de mesurer de manière standard les performances d'un système ou d'un processeur. Du point de vue du micro-architecte, les benchmarks (*SPEC CPU 2000 , MediaBench, MiBench*) sont utilisés comme référence et le critère de performance reste le taux d'instructions par cycle ou IPC.

3.2.1 Les benchmarks de la SPEC CPU 2000

La suite SPEC CPU 2000 est un ensemble de programmes tests basés sur des applications existantes et qui ont déjà été portées sur une grande variété de plates-formes. Ces programmes font office de référence pour comparer les performances des processeurs.

La suite des Benchmarks est partagée en deux catégories : *SPECInt2000* du tableau 3.1(a) regroupant des programmes effectuant des calculs entiers, et *SPECFp2000* du tableau 3.1(b) qui est orientée vers les calculs flottants. La figure ci-dessous présente les programmes de la suite SPEC 2000.

Programme	**Description**
164.gzip	Compression / décompression
256.bzip	Compression
176.gcc	Compilateur gcc2.7.2.2
181.mcf	Programme utilisant des opérations entières
175.vpr	Positionnement et routage

(a) *SPECInt 2000*

Programme	**Description**
188.ammp	Résolution équation de Newton (Déplacement des atomes)
179.art	Reconnaissance d'image et des réseaux neurones
183.equake	Simulation de programme d'ondes sismiques
186.wupwise	Programme de physique / mécanique quantique

(b) *SPECFp 2000*

TAB. 3.1 – Les programmes de la suite *SPEC*

Bien que ces benchmarks soient de loin les plus utilisés, ils ont le défaut d'être payants.

3.2.2 MediaBench

MediaBench[8] se compose d'applications multimédia et du domaine des communications.

Le tableau 3.2 donne une courte description des application de la suite MediaBench :

Programme	**Description**
JPEG	Compression d'imaes (Encode / Decode)
MPEG	Compression vidéo (Encode / Decode)
GMM	Norme européenne pour le transcodage de la parole (Encode / Decode)
G721	Compression CCITT (Encode / Decode)
PGP	Chiffrage à clef publique (RSA)
PEGWIT	Chiffrage à clefs publique (courbe elliptique)
GhostScript	Interpréteur Post-Script
MESA	Clone libre d'OpenGL
EPIC	Compression d'image
ADPCM	Codage audio

TAB. 3.2 – programmes de la suite MediaBench

3.2.3 MiBench

La suite Mibench[5] se compose de 35 applications de tests. Ces benchmarks sont divisés en six suites avec chaque suite visant un secteur spécifique comme la bureautique, les réseaux, la sécurité ou les télécommunications. Tous les programmes sont disponibles en code source C.

La suite MiBench [15] est dédiée à l'évaluation d'architecture embarquées et présente donc des caractéristiques différentes des programmes de tests des processeurs généralistes comme SPEC CPU 2000. La principale différence se situe sur le comportement des programmes vis-à-vis de la hiérarchie mémoire.

Les programmes de la suite MiBench 3.3 présentent un faible taux d'échec des caches. De plus, les codes flottants ont un faible taux d'opérations flottants.

Auto/ Industriel.	**Consumer**	**Office**	**Network**	**Security**	**Télécom.**
basicmath	jpeg	GhostScript	Djikstra	blowfish	CRC32
bitcount	lame	ispell	patricia	blowfish dec.	FFT
qsort	mad	rsynth	(CRC32)	pgp sign	IFFT
susan (edges)	tiff2bw	sphinx	(sha)	pgp verify	ADPCM enc.
susan (comers)	tiff2rgba	stringserach	(blowfish)	rijndael enc.	GSM enc.
susan (smoothing)	tiffdither			rijndael.dec	GSM enc.
	tiffmedian			sha	GSM
	typest				

TAB. 3.3 – Les programmes de la suite MiBench

[5] http :/ / www.eecs.umich.edu / mibench

Chapitre 4

Simulations et résultats

4.1 L'environnement de simulation

Le but des expériences réalisées est d'étudier la performance du cœur d'exécution dans le désordre d'un processeur superscalaire de degré élevé (16). Afin que les mesures du nombre d'instructions par cycle du processeur reflètent la performance de son cœur, nous avons opté pour les choix suivants :

- une prédiction de branchement parfaite,
- un cycle d'extraction des instructions qui se poursuit jusqu'au remplissage de la file d'entrée,
- une hiérarchie mémoire d'instructions délivrant toute instruction en un cycle,
- des programmes de tests ayant un faible taux d'échecs au premier niveau de cache de données.

D'autre part, nous nous intéressons plus particulièrement à l'impact sur la performance de la limitation des ressources sur le chemin de données. Par conséquent, nous avons opté pour une grande fenêtre d'instructions (512 entrées) et une file des lectures et des écritures en mémoire en proportion (256 entrées).

Les expériences ont été réalisées avec le simulateur de processeur cycle à cycle SIMPLESCALAR3.0 [1]. Le processeur de référence exécute du code PISA. Il possède 5 étages de *pipeline* (*extraction*, *renommage*, *lancement*, *terminaison*, *retirement*) et il est simulé par une version du simulateur modifiée afin de garantir l'extraction continue des instructions. En particulier, l'extraction se poursuit jusqu'au remplissage de la file d'extraction, dont la taille fixe le degré superscalaire. On franchit ainsi les sauts en cours de cycle. L'extraction peut être stoppée par le remplissage de la fenêtre d'instructions.

[1] http://www.simplescalar.com/

Le tableau 4.1 résume les paramètres principaux d'un processeur de référence en fonction de son degré superscalaire. On notera que l'étage de lancement lance toutes les instructions prêtes et que l'étage de retirement retire toutes les instructions terminées jusqu'à la première non terminée. De même, les unités de calculs sont suffisamment nombreuses pour ne pas représenter un frein au lancement.

Paramètres	**Degré 4**	**Degré 8**	**Degré 16**
Largeur de chargement	4	8	16
Prédiction de branchement	parfaite		
Extraction des instructions	parfaite		
Cache d'instructions	désactivé		
Cache de données L1	16Ko / 1-cycle / 4-voies		
Cache de données L2	256Ko / 6-cycles / 4-voies		
Largeur de lancement	illimitée		
Largeur de retirement	illimitée		
Unités de calculs (ad / int / fp)	4 / 4 / 4	8 / 8 / 8	16 / 16 / 16
Largeur de la fenêtre d'instructions	512		
File de lectures et d'écritures	256		

TAB. 4.1 – Paramètres du processeur de référence

Les programmes de tests utilisés pour évaluer les performances sont présentés dans la table 4.2. Ils sont extraits de la suite de programmes de tests MIBENCH [2]. Chaque simulation est limitée à un préfixe de 500 millions d'instructions. Chaque programme a été compilé avec le compilateur GNU `gcc` version 2.7.2.3 pour SIMPLESCALAR avec l'option `-O3`.

La suite MIBENCH est dédiée à l'évaluation d'architectures embarquées. Elle présente donc des caractéristiques différentes des programmes de tests des processeurs généralistes comme les SPEC2000 [3]. La principale différence se situe sur le comportement des programmes vis-à-vis de la hiérarchie mémoire [15]. Les programmes de la suite MIBENCH présentent un faible taux d'échecs des caches. De plus, les codes flottants ont un faible taux d'opérations flottantes.

Ainsi, les accès mémoire se faisant presque tous en un cycle et les opérations longues étant peu nombreuses, la latence moyenne des opérations est très voisine d'un seul cycle. Elle n'est plus significative et les deux seuls freins résiduels à la performance sont la largeur d'extraction et le parallélisme d'instruction (ILP) du programme. Autrement dit, les expériences que nous rapportons sont placées dans un cadre maximisant le nombre d'instructions exécutées par cycle (IPC) pour un ILP et un degré superscalaire donnés.

[2] http://www.eecs.umich.edu/mibench/

[3] http://www.spec.org/

Programme	accès mémoire	Calculs entiers	Sauts conditionnel	Sauts incond.	Calculs flottants
Rawcaudio	8%	64%	26%	2%	0%
Basic math	24%	55%	16%	3%	2%
Qsort	25%	51%	17%	4%	4%
Crc	35%	45%	19%	1%	0%
Fft	21%	60%	12%	2%	5%
Gsm encode	22%	73%	5%	0%	0%
Gsm decode	12%	70%	13%	5%	0%
Ispell	41%	39%	17%	3%	0%
Patricia	27%	53%	15%	3%	2%

TAB. 4.2 – Programmes de tests extraits de la suite MiBench

La figure 4.1 montre l'IPC de chaque programme de test simulé sur les processeurs superscalaires de degrés 4, 8 et 16. Les résultats pour le degré 16 font apparaître un IPC très variable d'un programme de test à l'autre, allant du plus fort (15,3 IPC) pour RAWCAUDIO au plus faible (2,9 IPC) pour CRC.

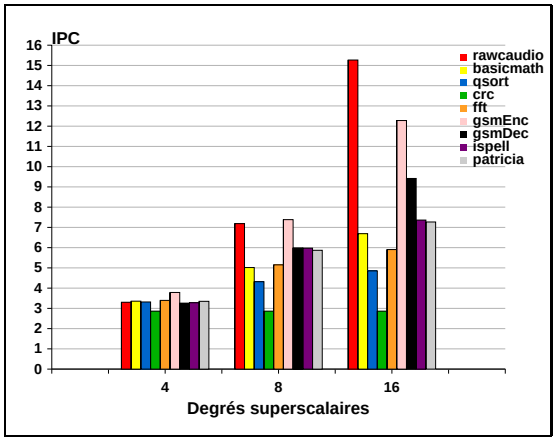


FIG. 4.1 – Performance en nombre d'instructions par cycle

La figure 4.1 est aussi indicatrice de l'ILP des divers programmes de tests. On peut constater que CRC atteint son IPC maximum dès le degré 4, ce qui provient de son faible taux d'ILP. BASICMATH, QSORT et FFT ont une progression infra-linéaire de leur IPC, qui s'approche de l'ILP. GSM, ISPELL et PATRICIA ont un accroissement linéaire de l'IPC. Enfin, RAWCAUDIO présente l'ILP le plus élevé avec un accroissement supra-linéaire de son IPC (ce qui peut sembler surprenant, mais que nous expliquons plus loin).

Pour ne retenir que l'exemple de CRC, son manque d'ILP vient du code source lui-même, organisé en une boucle calculant linéairement une somme

de contrôle. La récurrence crée une chaîne de dépendances. Le parallélisme exploité ne provient que du calcul du contrôle simultanément à l'amorce du calcul de la prochaine somme. En supposant qu'on conçoive à l'avenir des processeurs de degré superscalaire élevé, cette évolution devra avoir un impact sur l'organisation des codes, au même titre que la hiérarchie mémoire ou la prédiction de sauts. Pour CRC par exemple, on devrait remplacer le calcul linéaire de sa récurrence par un calcul arborescent pour augmenter l'ILP.

4.2 L'utilisation effective des ressources

4.2.1 Le chemin de données

Parmi les ressources présentes à l'intérieur du cœur du processeur, notre intérêt se porte principalement sur celles donnant accès aux données. La figure 4.2 montre le diagramme du cœur d'un processeur superscalaire de degré `d` sur lequel sont représentés les chemins de données.

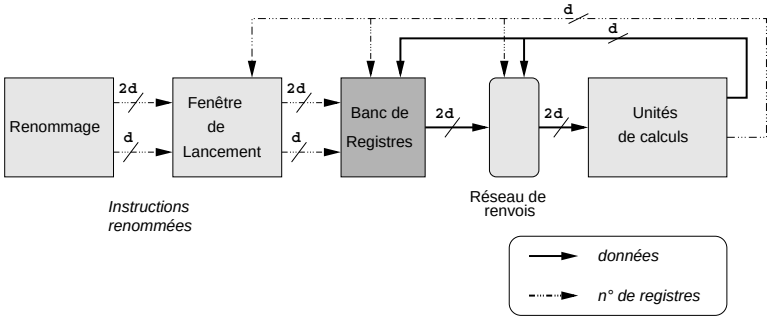


FIG. 4.2 – Diagramme du cœur d'un processeur à exécution dans le désordre de degré `d`

Le banc de registres et le réseau de renvois sont les ressources critiques pour un processeur superscalaire de degré élevé. En effet, concernant le banc de registres, le nombre de ses ports d'accès augmente avec le degré superscalaire du processeur. Concernant le réseau de renvois et la logique de réveil, le nombre de comparateurs est fonction du nombre d'unités de calcul u multiplié par la longueur de la fenêtre d'instructions f. Pour garantir des performances maximales, les processeurs actuels de degré `d` possèdent un banc de registres avec `3d` ports d'accès et un réseau de renvoi comprenant jusqu'à `2fu` comparateurs. Malheureusement, les futurs processeurs de degré élevé ne pourront pas conserver un jeu complet de ports et de comparateurs sans augmenter considérablement soit le temps de cycle du processeur, soit la latence des instructions.

Les expériences que nous proposons dans cette section mettent en évi-

dence les ressources du chemin de données effectivement utilisées lors de l'exécution des instructions.

4.2.2 Les compteurs de ressources

L'analyse du passage d'une instruction dans le cœur d'exécution du processeur permet de caractériser chacun de ses opérandes ainsi que les ressources effectivement consommées par ceux-ci.

Les différents compteurs que nous avons associés aux opérandes sont décrits précisément ci-dessous (on ne considère que les sources faisant référence à un registre et ne comptabilisons pas les sources constantes) :

SRC-PRenom : nombre d'opérandes prêts lorsque les instructions traversent l'étage de renommage.

`SRC-NPRenom` : nombre d'opérandes non prêts lorsque les instructions traversent l'étage de *renommage*.

`SRC-Prem` : nombre d'opérandes reçus par renvoi alors qu'un second opérande est encore en attente.

`SRC-Der` : nombre d'opérandes reçus par renvoi alors que plus aucun opérande n'est attendu.

`RES-TERM` : nombre de résultats dont la destination n'a pas encore été renommée.

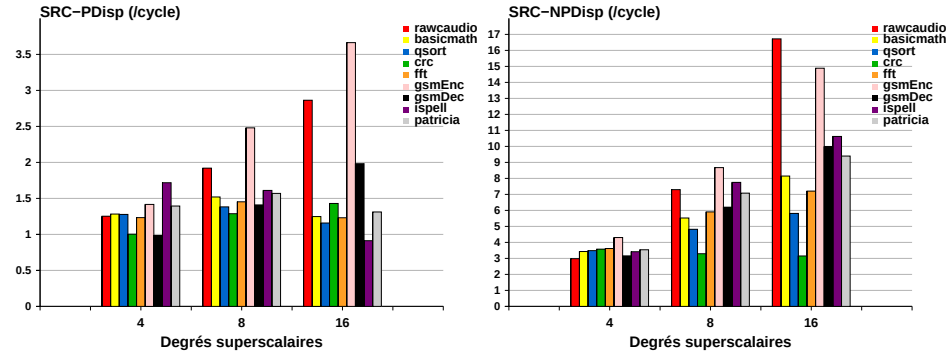
`RES-RETIR` : nombre de résultats dont la destination n'est toujours pas renommée au retirement.

4.2.3 Les compteurs de l'étage de *renommage*

Les compteurs placés dans l'étage de *renommage* séparent les sources qui peuvent être lues dans le banc de registres (`SRC-PRenom`) de façon anticipée à cette étape, de celles qui sont captées par le réseau de renvoi au niveau de l'étage de *lancement* (`SRC-NPRenom`).

Chaque compteur est une indication du nombre de ressources réellement utilisées par un programme au cours de son exécution. Le compteur `SRC-PRenom`, ramené à un nombre d'opérandes par cycle (voir la figure 4.3(a)), donne le nombre moyen de ports de lecture utilisés si l'on place le banc de registres dans l'étage de *renommage*. Le compteur `SRC-NPRenom` (voir la figure 4.3(b)) donne le nombre moyen de comparaisons positives par cycle lors du renvoi des résultats vers l'étage de *lancement*.

La figure 4.3(a) fait apparaître, de façon surprenante mais compréhensible, que le nombre de sources prêtes au *renommage* n'augmente pas forcément avec le degré superscalaire. Pour BASICMATH et QSORT, il y a moins de sources prêtes au *renommage* quand on traite 16 instructions par cycle que quand on en traite 4 ou 8. Cela s'explique parce que dans ces programmes tests, les dépendances concernent des instructions proches les



(a) Opérandes Prêts (b) Opérandes non Prêts

FIG. 4.3 – Compteurs de l'étage de renommage

unes des autres. Supposons que l'instruction *i+10* dépende de l'instruction *i*. Dans un processeur de degré superscalaire 4, quand l'instruction *i+10* passe dans l'étage de *renommage*, l'instruction *i* est déjà retirée. La source de *i+10* qui dépend du résultat de *i* est donc prête. Avec un processeur de degré 16, l'instruction *i* est encore dans l'étage de *lancement*. La source de *i+10* n'est donc pas prête.

Pour les programmes de tests de la figure 4.3(a), le nombre de sources prêtes ne dépasse jamais 4 par cycle. On en déduit que plus le degré superscalaire augmente, moins il est payant de mettre des ressources globales d'accès aux opérandes dans l'étage de *renommage* telles qu'un banc de registres à `2d` ports de lecture.

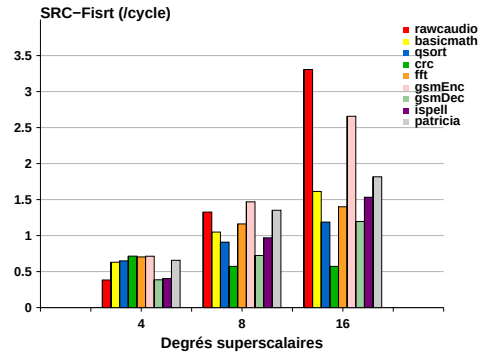
La figure 4.3(b) montre des valeurs de `SRC-NPRenom` pour les programmes de tests qui sont assez homogène pour le degré 4. Des variations apparaissent avec un degré 8 et se creusent avec un degré 16. Ces variations sont corrélées à l'IPC, ce qu'on peut constater en comparant les figures 4.3(b) et 4.1.

4.2.4 Les compteurs de l'étage de *lancement*

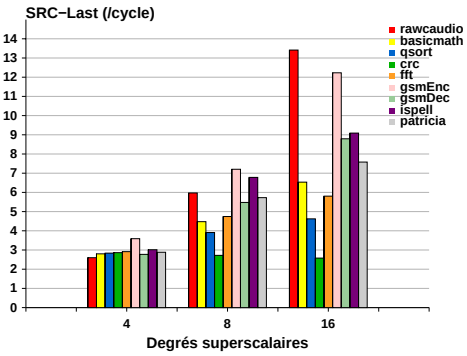
La figure 4.4(a) montre le nombre de sources `SRC-Prem` par cycle, la figure 4.4(b) montre le nombre de sources `SRC-Der` par cycle et la figure 4.4(c) montre la somme des sources `SRC-Prem` et `SRC-PRenom`.

Le compteur `SRC-Prem` est un indicateur de la part des sources en provenance d'instructions en vol qu'il n'est pas indispensable de récupérer par le réseau de renvoi. Ces sources sont celles d'instructions à deux sources en registre dont la seconde source est aussi en attente. On peut lire la première source dans une mémoire adressable après que le résultat dont elle dépend y ait été écrit.

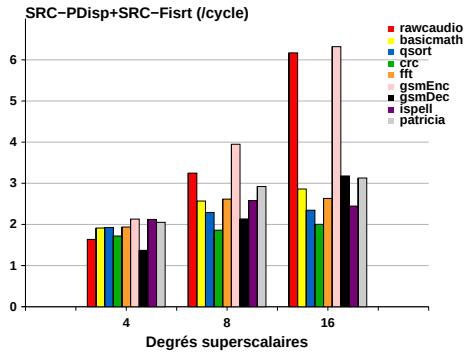
Le compteur `SRC-Der` est la part des sources qu'il faut récupérer aussi



(a) Premiers



(b) Derniers



(c) Prêts + Premiers

FIG. 4.4 – Compteurs de l'étage de lancement

vite que possible. Ce sont les dernières sources permettant le lancement immédiat de leur instruction (après attribution d'une unité de calcul). Un réseau de renvoi permet de récupérer ces sources au vol, plus rapidement qu'en les écrivant puis les lisant en registre. Faute d'un nombre suffisant de comparateurs, l'envoi de ces sources est retardé, ce qui ralentit le débit du chemin de données.

On constate que, comme pour le compteur `SRC-NPRenom`, les variations s'amplifient avec l'augmentation du degré superscalaire. Le programme de test CRC a un compteur stagnant alors que celui de RAWCAUDIO est plus que doublé. Cette progression du compteur `SRC-Der` peut paraître surprenante. Néanmoins, comme noté précédemment, plus le degré est élevé, moins il y a de données immédiatement prêtes au *renommage*. Il peut y avoir plus de sources reçues par envoi, et parmi elles dans certains cas, une forte proportion de sources `SRC-Der`. En quelque sorte, l'augmentation du degré superscalaire libère l'ILP potentiel. Le compteur `SRC-Der` est un bon indicateur de l'ILP. S'il stagne quand le degré augmente, c'est que l'IPC a atteint l'ILP du programme. S'il augmente plus que linéairement par rapport au degré, c'est qu'il reste un fort potentiel d'ILP non exploité.

En comparant les figures 4.4(b) et 4.1, on constate aussi que la valeur de `SRC-Der` est corrélée à celle de l'IPC. Cela montre que le dimensionnement du réseau de renvoi doit être privilégié par rapport au nombre de voies d'accès aux registres. Il est plus important pour l'IPC de transmettre le plus vite possible les sources de type `SRC-Der` que de récupérer les autres sources, moins critiques.

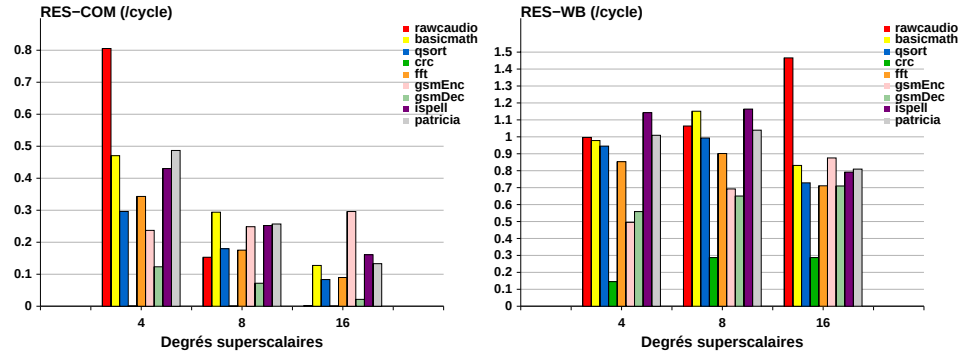
La figure 4.4(c) présente le nombre moyen d'accès aux sources `SRC-Prem` et `SRC-PRenom` par cycle, qui est une bonne indication du nombre de ports de lecture nécessaires par registre. Pour le degré 16, ce nombre moyen est inférieur à 7, ce qui laisse penser qu'un banc de registres actuel (12 ports de lecture pour celui du Pentium 4) aurait assez de voies d'accès.

4.2.5 Les compteurs de résultats

La figure 4.5(b) présente les valeurs du compteur `RES-RETIR`. Ce compteur est indicatif du nombre d'écritures inévitables quand elles sont effectuées dans l'étage de *retirement*. Notamment, il n'est pas nécessaire d'écrire un résultat qui ne peut plus être consommé que par des instructions déjà en attente de leurs sources. Ceci est le cas quand le registre architectural du résultat a fait l'objet d'un nouveau renommage, la nouvelle copie se substituant à l'ancienne pour les futures références. En quelque sorte, le résultat est conservé dans le réseau de renvoi pour alimenter les successeurs en attente, mais n'est pas écrit dans le banc de registres puisqu'aucune nouvelle instruction ne peut plus y faire référence.

On constate que le nombre d'écritures est très faible (inférieur à 1) et qu'il décroit avec le degré superscalaire. Par conséquent, l'accroissement

du nombre d'instructions traitées par cycle n'impose pas un accroissement du nombre de ports d'écriture. C'est même tout le contraire.



(a) au moment de la terminaison (b)au moment du retirement

FIG. 4.5 – Ecritures inévitables

La figure 4.5(a) présente les valeurs du compteur `RES-TERM` correspondant aux destinations non renommées des instructions atteignant l'étage de *terminaison*. Nous supposons, ne l'oublions pas, que la prédiction des sauts est parfaite. Quand la prédiction n'est pas parfaite, les renommages sont spéculatifs. Soient i et j deux instructions de destination d, i précédant j. Si les deux instructions i et j font partie du même groupe spéculatif (il n'y a pas de nouvelle spéculation entre i et j) et si, quand i se termine, j est renommée, alors la destination de i est renommée (et l'écriture du résultat n'est pas indispensable). Sans spéculation, on peut éviter toute écriture de résultat dont la destination est renommée.

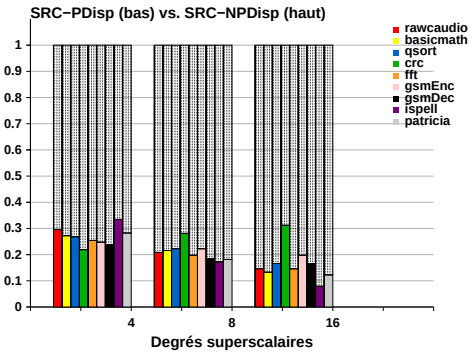
Cette fois encore, on note que le nombre d'écritures est très faible (inférieur à 2). Il est plus élevé que dans l'étage de *retirement*, ce qui n'est pas surprenant.

Ces résultats motivent fortement l'exploration des mécanismes favorisant la réduction du nombre de ports d'écriture. Les techniques de libération anticipée de registres [11, 22] sont généralement proposées pour réduire le nombre de registres. Couplées à une écriture tardive (au *retirement*), on ne vise alors plus une réduction des registres mais une réduction du nombre de ports d'écriture.

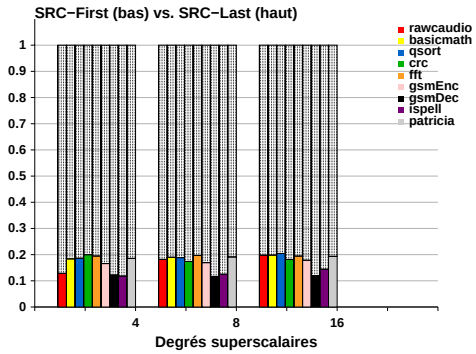
4.2.6 La répartition des trois catégories de registres sources

La figure 4.6(a) montre le rapport entre le nombre de sources prêtes (`SRC-PRenom`) et le nombre de sources non prêtes (`SRC-NPRenom`) au moment du *renommage*. La proportion des opérandes prêts est inférieure à 30% des opérandes sources. Cette proportion a tendance à diminuer quand le degré superscalaire augmente (27% pour le degré 4, 21% pour le degré 8

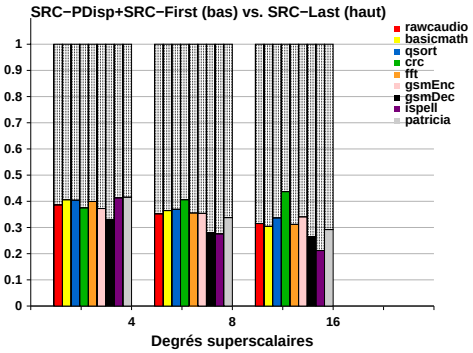
et 16% pour le degré 16). Cela suggère que la lecture anticipée des sources est d'autant moins profitable que le degré est élevé. Le passage d'un processeur actuel à un processeur futur ayant une meilleure prédiction de branchement et une fenêtre d'instructions plus grande nécessiterait donc une nouvelle évaluation de mécanismes tel que le préchargement des opérandes [33].



(a) au moment du renommage



(b) au moment du lancement



(c) au moment du lancement

FIG. 4.6 – Répartition des opérandes sources

La figure 4.6(b) montre le rapport entre le nombre d'opérandes reçus par renvoi alors qu'un second opérande est encore en attente (`SRC-Prem`) et le nombre d'opérandes reçus par renvoi alors que plus aucun opérande n'est attendu (`SRC-Der`). Les opérandes *Prem* représentent en moyenne entre 16% (degré 4) et 18% (degré 16) de tous les envois. La proportion varie très peu avec le degré superscalaire. La majeure partie des renvois (plus de 80%) sont dans le chemin critique de la latence des instructions.

La figure 4.6(c) présente le rapport entre les sources qui sont dans le chemin critique (`SRC-Der`) et celles qui n'y sont pas (somme des compteurs `SRC-Prem` et `SRC-PRenom`). Cette comparaison fait ressortir qu'un peu plus de 60% des sources en registre appartiennent à la première caté-

gorie. Cette proportion augmente avec le degré (61% pour le degré 4, 66% pour le degré 8 et 69% pour le degré 16).

Si l'on désire équilibrer les ressources de renvoi et les ressources d'accès, on peut recevoir les opérandes `SRC-Der` par le réseau de renvoi et lire les opérandes `SRC-Prem` et `SRC-PRenom` en registre. Néanmoins, l'IPC final en dépendant, il faut disposer d'un réseau de renvoi suffisant pour renvoyer, à chaque cycle, tous les résultats (autant que d'unités de calculs) à toutes les sources dépendantes de type `SRC-Der`.

L'exemple de RAWCAUDIO, qui fait apparaître une augmentation plus que linéaire de `SRC-Der`, semble indiquer que le réseau de renvoi doit aussi s'accroître plus que linéairement par rapport au degré superscalaire. Deux facteurs limitent le nombre de successeurs qu'il est nécessaire de fournir par cycle. Tout d'abord, il y a le nombre maximum de sources de type *Der* reçues au cours d'un même cycle. Nos simulations ont établies que cela varie de 36 à 47 selon les programmes de tests pour le degré 4, de 58 à 81 pour le degré 8 et de 62 à 96 pour le degré 16. Ensuite, on peut se contenter d'envoyer les résultats à suffisamment de sources pour, au cycle suivant, alimenter chacune des unités de calcul d'une opération (il vaut mieux envoyer un résultat à deux sources d'instructions employant deux unités distinctes plutôt qu'à deux instructions employant la même unité). En appliquant cette stratégie, on peut limiter les renvois à `u` par cycle.

4.3 Réduction du nombre de ports d'accès

Les expériences de la section précédente font apparaître que le nombre de ressources réellement utilisées demeure faible malgré l'augmentation du degré superscalaire. Concernant les ports d'accès au banc de registres, nous avons fait ressortir qu'un processeur proche de l'idéal pouvait, pour un degré superscalaire 16, limiter ses écritures à moins d'une par cycle et ses lectures à moins de sept par cycle. S'agissant des comparateurs du réseau de renvoi, un processeur doit propager les résultats issus de ses u opérateurs à autant de sources, soit u propagations par cycle.

Des travaux précédents ont étudié l'effet de la réduction des ressources sur la performance. Cependant, leur finalité était de faire apparaître les avantages de telle ou telle proposition d'aménagement du chemin des données d'un processeur actuel (degré 4) ou d'un futur proche (degré 8). Dans le cadre d'un processeur réaliste (prédiction réaliste des sauts, latence réelle des opérations), on ne perçoit pas tout l'effet de la réduction des ressources, l'IPC étant tout autant bridé par le processeur lui-même.

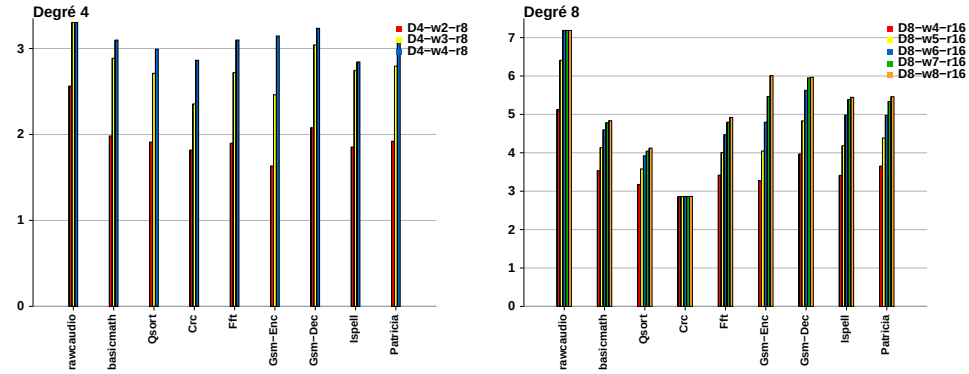
Dans cette section, nous évaluons les dégradations de performance induites par la seule limitation des ressources, sans mécanisme de compensation. Nous commençons par réduire le nombre de ports d'écriture puis nous ajoutons une réduction du nombre de ports de lecture. Notons qu'en

limitant les ports de lecture, on limite la capacité de lancement du processeur. De même, en limitant les ports d'écriture, on limite la capacité de terminaison du processeur. En effet, la lecture des sources en registres s'effectue conformément au schéma de la figure 4.2. Au lancement, toute instruction prête lit ses sources, à concurrence des ports de lecture disponibles. Une fois tous les ports consommés, le lancement s'arrête pour le cycle. De même, les résultats sont écrits à concurrence du nombre de ports d'écriture. Une instruction qui ne peut écrire son résultat faute de ports d'écriture disponibles tente son écriture au cycle suivant. Sa terminaison est retardée.

4.3.1 Réduction des ports d'écriture

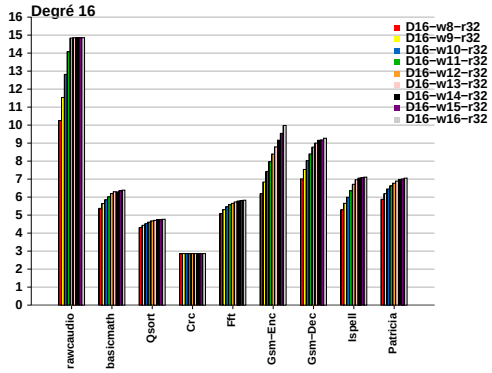
La figure 4.7 présente l' IPC pour des micro-architectures de degrés d ($d = 4, 8$ et 16) dont le nombre de ports d'écriture (`W`) varie de $d/2$ à d. Dans cette expérience, le nombre des ports de lecture est limité à $2d$.

Pour le degré 4, on constate une dégradation importante de l'IPC (jusqu'à 20%) dès le passage de quatre à trois ports d'écriture. Dans le cas du degré 8, sept ports garantissent de conserver 95% de l'IPC obtenu avec huit ports. La dégradation devient plus importante dès qu'on se limite à six ports. Pour le degré 16, 12 ports donnent au moins 95% de l'IPC obtenu avec 16 ports, à l'exception notable du programme test GSM-ENCODE pour lequel il faut disposer d'au moins 14 ports.



(a) Degré 4

(b) Degré 8



(C) Degré 16

FIG. 4.7 – Répartition du nombre de ports d'écriture

Ceci montre qu'il est très pénalisant de limiter les ports d'écriture et qu'on ne peut espérer qu'un gain marginal en procédant ainsi. Il faut disposer d'environ 0,8 ports d'écriture par instruction, ce qui correspond au nombre moyen de destinations par instruction.

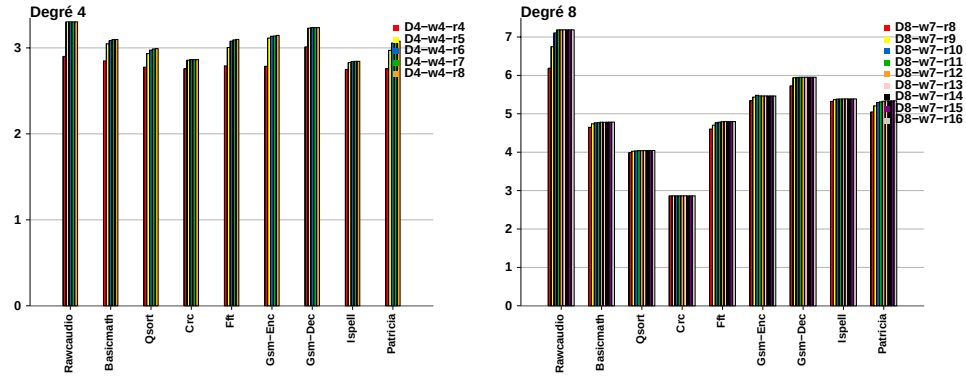
Les mesures des compteurs `RES-RETIR` et `RES-TERM` ont montré pourtant que peu d'écritures étaient indispensables. Nous suggérons de lever la contrainte d'écriture de tous les résultats. On peut imaginer de placer les résultats à leur sortie des opérateurs dans une file d'attente, à partir de laquelle l'étage de *retirement* procède aux seules écritures inévitables (i.e. destination non renommée). Le réseau de renvoi quant à lui peut n'effectuer aucune écriture s'il se limite à diffuser les résultats aux sources. En se basant sur des remarques comparables, Balkan *et. al.* ont proposé une technique limitant le nombre d'écritures.

4.3.2 Réduction des ports de lecture

Cette fois, nous fixons le nombre de ports d'écriture au minimum suggéré par la mesure précédente (soit quatre ports pour le degré 4, sept ports

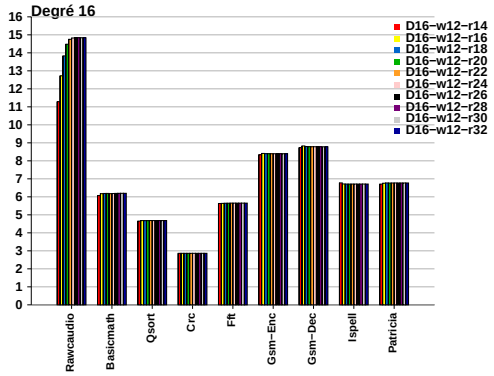
pour le degré 8 et douze pour le degré 16).

La figure 4.8(a) montre l'IPC pour la micro-architecture de degré 4 avec des ports de lecture variant de 4 à 8. On constate peu de dégradation de l'IPC quand on réduit les ports de lecture jusqu'à 5. La figure 4.8(b) montre l'IPC pour la micro-architecture de degré 8 avec des ports de lecture variant de 8 à 16. Elle fait apparaître une dégradation en deçà de 10 ports. Enfin, la figure 4.8(c) montre l'IPC pour la micro-architecture de degré 16 avec des ports de lecture variant de 14 à 32 par pas de 2. La dégradation se fait sentir dès qu'on dispose de moins de 20 ports.



(a) Degré 4

(b) Degré 8



(c) Degré 16

FIG. 4.8 – Réduction du nombre de ports de lecture

La figure 4.8 montre que, quels que soient les programmes, une économie plus substantielle peut être faite sur les ports de lecture que sur les ports d'écriture. Néanmoins, le nombre minimal de ports d'accès nécessaires reste élevé. Il faut en moyenne 1,2 ports de lecture par instruction, ce qui correspond au nombre moyen de sources en registres par instruction. Au total, il faut avoir 2 ports d'accès par instruction, soit 16 ports pour le degré 8 et 32 ports pour le degré 16 (un Pentium 4 a 18 ports d'accès sur le banc de registres entiers).

Les mesures précédentes des compteurs `SRC-PRenom`, `SRC-Prem` et `SRC-Der` montrent que la diminution des ports de lecture peut être nettement plus importante sans dégrader l'IPC en séparant les sources critiques (`SRC-Der`), transmises par le réseau de renvoi, des sources moins critiques (`SRC-Prem` et `SRC-PRenom`), lues en registres. Dans ces conditions, 8 ports d'accès (7 pour la lecture et 1 pour l'écriture) suffiraient.

4.4 Travaux relatifs

Le banc de registres, le réseau de renvois anticipés, la logique de sélection des instructions et la logique de réveil des instructions sont devenus les ressources critiques qui ont le plus grand mal à suivre l'extension du degré superscalaire des micro-processeurs [23]. De nombreux travaux se sont donc attaqués à réduire la complexité de ces différentes ressources.

4.4.1 Complexité du banc de registres

La notion de *registres physiques virtuels* introduite par González *et al.* [13] qui permet de découpler la gestion des dépendances de celle du stockage des résultats a inspiré de multiples travaux. D'abord, il y a ceux visant à réduire la pression sur les registres en retardant leur allocation [21], en anticipant leur libération [22, 11] ou bien en combinant les deux techniques [31].

Ensuite, la virtualisation a suscité des travaux exploitant la réutilisation de valeurs dans les registres physiques [18, 3]. En observant que de nombreux opérandes peuvent être stockés sur moins de 16 bits, la virtualisation partitionne les registres 64 bits en tranches de plus petites tailles (32 et 16 bits) [20, 11].

Les optimisations sur le banc de registres ont commencé par la réduction du nombre de ports. Les techniques de partitionnement (*clustering*) permettent de réduire le nombre de ports de lecture [12]. Un partitionnement permettant de réduire également le nombre de ports d'écriture a été proposé dans [32]. Le partitionnement introduit une difficulté supplémentaire : l'allocation des partitions. Dans [27], Rochecouste *et. al.* proposent une architecture partitionnée en chemins de données de largeurs différentes. Le partitionnement étend la réduction de la taille des registres à tout le chemin de données du cœur d'exécution.

Une réduction du nombre de ports a également été proposé, soit en hiérarchisant le banc de registres [4], soit en le découpant [10]. Ces techniques nécessitent un réseau de renvoi plus complexe.

Plus récemment, on a suggéré de supprimer des lectures inutiles dans le banc de registres soit en ajoutant un étage de pipeline avant la lecture, soit par une prédiction des renvois [10]. Dans [26], les auteurs réduisent le

nombre de ports en préchargeant des opérandes et en retardant l'écriture des résultats.

4.4.2 Complexité de l'étage de lancement

Le partitionnement (*clustering*) ne réduit pas uniquement la complexité du banc de registres mais également celle du réseau de renvoi et des files de lancement. La complexité de l'étage de lancement peut être réduite en limitant le nombre de comparateurs et en prédisant le dernier opérande prêt [34].

Pour réduire le nombre de comparateurs, Michaud *et. al.* [35] proposent de pré-ordonnancer dynamiquement les instructions suivant leurs dépendances afin de minimiser la taille de la fenêtre de lancement.

Hrishikesh *et. al.* [17] proposent de pipeliner la logique de réveil ainsi que la logique de sélection des instructions afin de permettre à ces structures de suivre l'augmentation de la fréquence des processeurs.

4.4.3 Complexité du réseau de renvoi

Comme pour le banc de registres et les files de lancement, le partitionnement réduit la complexité du réseau de renvoi. Réduire la complexité du réseau de renvoi par l'intermédiare du partitionnement est une solution qui dimimue la latence les transferts locaux (intra-partitions) [29]. Les travaux sur le réseau de renvoi divergent : alors qu'Aggarwal propose de limiter le renvoi des unités de calculs à elles mêmes pour minimiser le temps de cycle du renvoi [1], Sassone *et. al.* suggèrent de pipeliner le réseau de renvoi [30].

4.4.4 Généralisation et compositions des techniques

La comparaison des différentes techniques proposées afin de réduire la complexité de la micro-architecture est difficile car les environnements d'expérimentation sont toujours très différents. Majoritairement, les expériences se placent dans le cadre de processeurs actuels de degré superscalaire 4 et étendent parfois leurs expériences à un degré 8. L'extensibilité des techniques proposées est rarement vérifiée.

Les travaux sur l'extensibilité ne visent plus le degré superscalaire du processeur mais la fenêtre d'instructions. Ces travaux [2, 9] se basent principalement sur la sauvegarde de points de contrôle. Ces processeurs envisagent de contenir plus de mille instructions en vol afin d'amortir des latences mémoires de plus en plus importantes. La sauvegarde de points de contrôle leur permet de retirer les instructions dans le désordre et d'établir une gestion agressive des registres, notamment par une allocation retardée et une libération anticipée.

Une réduction générale de la complexité est présentée par une micro-architecture dans laquelle les ressources sont divisées par deux [19]. Récemment, Goossens *et. al.* ont proposé un algorithme d'ordonnancement distribué des instructions adapté aux micro-architectures de degré élevé [14].

Chapitre 5

Conclusion

Dans ce mémoire, nous avons mis en évidence que le nombre de ressources effectivement utilisées par le chemin de données du processeur n'augmente pas démesurément lorsque le degré superscalaire s'accroît, contrairement à ce que l'on aurait pu craindre. Cela laisse penser que l'algorithme d'exécution spéculative et en désordre est peut-être adapté aux degrés superscalaires élevés, ouvrant la voie vers des propositions micro-architecturales efficaces.

En particulier, le nombre de résultats qu'il faut écrire par cycle, correspondant aux données rémanentes, reste faible (inférieur à 2) et même décroît avec le *degré superscalaire*. Le nombre de sources non critiques (source prête au renommage ou dont l'instruction attend une autre source plus tardive) ne dépasse pas 7 par cycle pour le degré 16. Cela montre qu'un banc de registre à 8 ports d'accès peut suffire, ce qui est en decà de ce qui est implanté dans un pentium 4.

Enfin, notre étude a aussi montré que la seule diminution des ports d'accès au banc de registres sans réorganisation de l'attribution de ces ports dégrade l'IPC dès qu'on descend au-dessous de ports pour instructions traitées par cycle.

Bibliographie

[1] Aneesh Aggarwal. *Single fu bypass networks for high clock rate superscalar processors*. In High Performance Computing - HiPC 2004, 11th International Conference, Bangalore, India, December 19-22, 2004, Proceedings, pages 319-332, 2004.

[2] Haitham Akkary, Ravi Rajwar, and Srikanth T. Srinivasan. *Checkpoint processing and recovery : Towards scalable large instruction window processors*. In MICRO 36 : Proceedings of the 36th annual IEEE / ACM International Symposium on Microarchitecture, page 423, Washington, DC, USA, 2003. IEEE Computer Society.

[3] Saisanthosh Balakrishnan and Gurindar S. Sohi. *Exploiting value locality in physical register files*. In MICRO 36 : Proceedings of the 36th annual IEEE / ACM International Symposium on Microarchitecture, page 265, Washington, 2003. IEEE Computer Society.

[4] Rajeev Balasubramonian, Sandhya Dwarkadas, and David H. Albonesi. *Reducing the complexity of the register file in dynamic superscalar processors*. In MICRO 34 : Proceedings of the 34th annual ACM / IEEE international symposium on Microarchitecture, pages 237-248, Washington, DC, USA, 2001. IEEE Computer Society.

[5] Deniz Balkan, Oguz Ergin, Dmitry Ponomarev and Kanad Ghose. *Selective writeback : Improving processor performance and energy efficency*. In 1st IBM Waston Conference Interaction Architecture Circuits and Compilers (P=AC^2), 2004

[6] Daniel Holmes Friendely, Sanjay Jeram Patel, Yale N. Patt, Putting the fill unit to work : *Dynamic optimization for trace cache microprocessors*. In proceedings of the 31st annual ACM / IEEE International symposium on microarchitecture, pp. 173-181. IEEE Computer Society Press, 1998.

[7] Heihaw Chuang and Brad Calder : *Predicate prediction for efficient out-of-order execution*. In proceeding of the International Conference on super computing, 2003.

[8] Benjamin Bishop, Thomas P.Kelliher, Mary Jane Irwin. -A detailed analysis of MediaBench

[9] Adriàn Cristal, Oliverio J. Santana, Mateo Valero, and José F. Marténez. *Toward kilo-instruction processors*. ACM Trans. Archit. Code Optim., 1(4) :389-417, 2004.

[10] José-Lorenzo Cruz, Antonio Gonzàlez, Mateo Valero, and Nigel P. Topham. *Multiple-banked register file architectures*.In ISCA 00 : Proceedings of the 27th annual international symposium on Computer architecture, pages 316-325, NY, 2000. ACM Press.

[11] Oguz Ergin, Deniz Balkan, Kanad Ghose, and Dmitry Ponomarev. *Register packing : Exploiting narrow-width operands for reducing register file pressure*. In MICRO 37 : Proceedings of the 37th annual IEEE / ACM International Symposium on Microarchitecture, pages 304-315, Washington, DC, USA, 2004. IEEE Computer Society.

[12] Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko Vranesic. *The multicluster architecture : reducing cycle time through partitioning*. In MICRO 30 : Proceedings of the 30th annual ACM / IEEE international symposium on Microarchitecture, pages 149-159, Washington, DC, USA, 1997. IEEE Computer Society

[13] Antonio Gonzalez, J. Gonzalez, and M. Valero. Virtual-physical registers. In HPCA'98 : *Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, page 175, Washington, DC, USA, 1998. IEEE Computer Society.

[14] Bernard Goossens and David Defour, *Ordonnancement distribué d'instruction*s, In *SympA*'2005 : *Symposium en Architecture des Machines, le Croisic, France, 2005*

[15] Matthew R. Guthaus, Jeffrey S,Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, *MiBench : A free, commercially representative embedded benchmarks suite*. In *IEEE 4th Annual Workshop on Workload Characterization, Austin, TX, December 2001*.

[16] John L. Hennessy et David A. Patterson : *Computer Architecture ; A quantitative approach - Third edition*. Morgan Kaufman Publishers, 2003.

[17] M. S. Hrishikesh, Doug Burger, Norman P. Jouppi, Stephen W. Keckler, Keith I. Farkas, and Premkishore Shivakumar. *The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays*. In ISCA 02 : Proceedings of the 29th annual international symposium on Computer architecture, pages 14-24, Washington, DC, USA, 2002. IEEE Computer Society.

[18] Stephen Jourdan, Ronny Ronen, Michael Bekerman, Bishara Shomar, and Adi Yoaz. *A novel renaming scheme to exploit value temporal locality through physical register reuse and unification*. In MICRO 31 : Proceedings of the 31st annual ACM / IEEE international symposium on Microarchitecture, pages 216-225, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press

[19] Ilhyun Kim and Mikko H. Lipasti. *Half-price architecture.* In ISCA 03 : Proceedings of the 30th annual international symposium on Computer architecture, pages 28-38, New York, NY, USA, 2003. ACM Press.

[20] Masaaki Kondo and Hiroshi Nakamura. *A small, fast and low-power register file by bit-partitioning.* In HPCA 05 :Proceedings of the 11th International Symposium on High-Performance Computer Architecture, pages 40-49, Washington,DC, USA, 2005. IEEE Computer Society.

[21] Teresa Monreal, Antonio Gonzàlez, Mateo Valero, Jose Gonzalez, and Victor Vinals. Delaying physical register allocation through virtual-physical registers. In MICRO 32 : Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture, pages 186-192, Washington, DC, USA, 1999. IEEE Computer Society.

[22] Teresa Monreal, Victor Vinals, Antonio Gonzàlez, and Mateo Valero. *Hardware schemes for early register release. In ICPP 02* : Proceedings of the 2002 International Conference on Parallel Processing, page 5, Washington, 2002. IEEE Computer Society.

[23] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In ISCA 97 : *Proceedings of the 24th annual international symposium on Computer architecture*, pages 206-218, New York, NY, USA, 1997. ACM Press.

[24] Gellen Hinto, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, Patrice Roussel. The Microarchitecture of the Pentium 4 Processor. Intel Technologie Journal. Q1, 2001

[25] David Parello, Bernard Goossens, Mourad Bouache, Ali El Moussaoui. *Comment répartir les ressources du chemin de données* ? accepté à Sympa'06.

[26] Il Park, Michael D. Powell, and T. N. Vijaykumar. *Reducing register ports for higher speed and lower energy*. In MICRO 35 : Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture, pages 171-182, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[27] Olivier Rochecouste, Gilles Pokam, and André Seznec. *A case for a complexity-effective, width-partitioned microarchitecture*. In NRIA Research Report, RR-5677, 2005.

[28] E. Rotenberg, S. Bennett, and J. E. Smith. *Trace Cache : a Low Latency Approach to High Bandwidth Instruction Fetching.* 29th International Symposium on Microarchitecture, Dec. 1996.

[29] Pierre Salverda and Craig Zilles. *A criticality analysis of clustering in superscalar processors.* In MICRO 38 :Proceedings of the 38th annual IEEE/ACMInternational Symposium onMicroarchitecture, pages 55-66,Washington, 2005. IEEE Computer Society.

[30] Peter G. Sassone and D. Scott Wills. *Multicylcle broadcast bypass : Too readily overlooked. In Workshop onComplexity-Effective Design* (in conj. with ISCA), 2004.

[31] Josep Liosa Adrià n Cristal, Jose F. Martinez and Valero. *Ephemeral registers with multicheckpointing*. In Technical Report UPC-DAC-2003-51. Departament d'Arquitectura de Computadors, Universitat Politècnia de Catalunya, Bercelona, Spain. Also submitted to the 35th International Symposium on Microarchitecture, 2002.

[32] André Seznec, Eric Tollec, and Olivier Rochecouste. *Register write specialization register read specialization : a path to complexity-eeffective wide-issue superscalar processors*. In MICRO 35 : Proceeding of the 35th annual ACM/IEEE international symposium on Microarchitecture, pages 383-394, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[33] Nam Sung Kim and Trevor Mudge. *Reducting register ports using delayed write-back queues and operand prefetch*. In ICS'03 :Proceedings of the 17th annual international conference on supercomputing, pages 40-49, Washington, DC, USA, 2005. IEEE Computer Society.

[34] Piere Michaud and André, Seznec. *Data-flow prescheduling for large instruction windows in out-of-order procssors*. In HPCA '01 : Proceedings of the 7th International Symposium on High -Performance Computer Architecture, Pages 27, Washington, DC, USA, 2001. IEEE Computer Society.

[35] Dan Ernst and Todd Ausstin. *Efficient Dynamic Scheduling Through tag elimination*. In ISCA '02 : Proceedings of the 29th annual international symposium on Computer architecture, pages 37-46, Washington, DC, USA, 2002. IEEE Computer Society.

Index