Extração do Paralelismo em Arquiteturas com Capacidade de Execução Condicional

Anna Dolejsi Santos* e Edil S. T. Fernandes†
Deptº de Computação - UFF e COPPE Sistemas - UFRJ

Sumário

O artigo apresenta o modelo CONDEX: uma arquitetura do tipo VLIW com capacidade de execução condicional. A arquitetura básica é formada por múltiplas unidades funcionais independentes que podem operar concorrentemente, processando diversas instruções procedendo do mesmo programa de aplicação. Além da especificação do modelo de arquitetura, o artigo descreve algumas técnicas para geração de código paralelo para configurações derivadas do modelo CONDEX e os experimentos para avaliar a qualidade desse código.

Abstract

In this paper we present the CONDEX machine model: A VLIW architecture supporting the conditional execution concept. The basic machine includes multiple functional units that can operate in parallel, executing several machine instructions proceeding from the same application program. In addition to the machine model specification, the paper describes some techniques for the generation of parallel code, and the experiments to evaluate the quality of this code.

1 Introdução

Arquiteturas Super Escalares são capazes de executar simultaneamente, durante cada ciclo de máquina, diversas instruções procedentes de um mesmo programa de aplicação ([TABA91], [JOHN91] e [FERN92]). Para tanto são equipadas com múltiplas unidades funcionais independentes que podem ser ativadas em paralelo.

Para que o paralelismo potencial de um processador Super Escalar possa ser efetivamente explorado, torna-se necessário detectar e selecionar as instruções do

*Universidade Federal Fluminense, Praça do Valonguinho s/n, Instº de Matemática 4.º andar, Deptº de Computação, Centro, Niterói, RJ, CEP 24210-130.
†Universidade Federal do Rio de Janeiro C.P. 68.511, CEP 21.945-970 - Rio de Janeiro, RJ.

programa de aplicação que podem ser executadas durante cada ciclo de máquina. Essa tarefa é levada a cabo pelo algoritmo de escalonamento que modifica a ordem em que as instruções do programa serão executadas e determina quais as unidades que devem ser ativadas durante cada ciclo de máquina.

Dependendo do nível de implementação do algoritmo responsável pelo escalonamento das instruções, arquiteturas Super Escalares podem ser classificadas em:

- processadores com escalonamento dinâmico, e
- processadores com escalonamento estático.

No primeiro tipo, o algoritmo de escalonamento é implementado diretamente no *hardware*. No segundo tipo, o escalonamento das instruções é levado a cabo por um algoritmo implementado pelo *software* de suporte da arquitetura. Desse modo, o algoritmo de escalonamento é realizado previamente, durante a fase de geração de código. É importante mencionar que é possível também combinar esses dois tipos de algoritmos escalonamento.

Máquinas *Very Large Instruction Word* (VLIW) são exemplos de processadores com algoritmo de escalonamento estático. Essas arquiteturas ([FISH84] e [COLW88]) surgiram como conseqüência do desenvolvimento de técnicas de compactação global de micro-programas ([TOKO78], [FISH81] e [NICO85]).

A unidade de controle de uma arquitetura VLIW, faz a busca do grupo de instruções que serão executadas durante cada ciclo do processador conforme especificado em tempo de geração de código.

Nesse texto utilizamos o termo "Instrução Multifuncional" (IMF ou instrução longa) para denotar o conjunto daquelas instruções de máquina que serão buscadas e executadas a cada ciclo de processador. Portanto cada instrução longa (ou IMF) é formada pela coleção de instruções que serão iniciadas no mesmo ciclo de máquina.

A unidade de controle dos processadores VLIW é muito simples: para cada dispositivo funcional existe um campo na IMF indicando a função que deverá ser executada. A seguir ilustramos a organização das instruções longas de uma arquitetura VLIW hipotética.

Supondo que a máquina VLIW possui três ALUs (unidades responsáveis pelas operações aritméticas e lógicas), e uma FPU (responsável pelas operações em ponto flutuante), então cada IMF deve incluir quatro campos distintos especificando a instrução que será realizada por cada unidade funcional durante a execução da instrução longa. A Figura 1.1 mostra parte de uma IMF da máquina VLIW contendo os campos que controlam as quatro unidades.

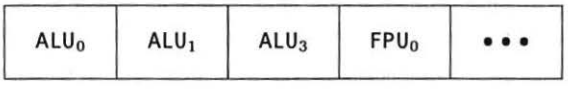

Figura 1.1: A Instrução Multifuncional (IMF) de uma Arquitetura VLIW

Na Figura 1.1 os três primeiros campos da IMF controlam as ALUs e o quarto campo a FPU da máquina VLIW em questão. Examinando um campo dessa instrução (Figura 1.2), notamos que ele contém o código de operação e os operandos necessários para a execução da instrução especificada.

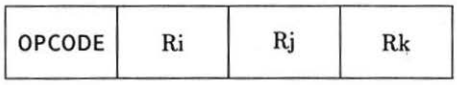

Figura 1.2: Um Campo da Instrução Multifuncional de uma VLIW

Na Figura 1.2, OPCODE indica a instrução que deverá ser executada pela unidade funcional correspondente, Ri e Rj são os registradores fonte e Rk o destino da instrução.

Em tempo de execução, as instruções indicadas nos quatro campos da Figura 1.1 serão realizadas em paralelo. Operações do tipo NOP (*no operate*) deverão ser introduzidos se desejarmos que as unidades funcionais correspondentes não iniciem uma nova instrução quando a IMF for executada.

As técnicas de escalonamento de instruções reorganizam o código objeto para manter as unidades funcionais das arquiteturas Super Escalares bem utilizadas (promovendo assim a redução no tempo de execução de programas do usuário). Independente da técnica empregada, o escalonamento deve considerar a interdependência das instruções (i.e., as dependências de dados) e a limitação de recursos existentes na arquitetura [KELL75], [AUHT85], [ACOS86], [FERN92] e [SANT94].

Durante a compactação (ou escalonamento) de instruções duas estratégias básicas podem ser usadas:

- a compactação local, e
- a compactação global.

Na compactação local, o alcance das atividades do algoritmo restringe-se aos comandos do bloco básico. Somente após o tratamento de um bloco básico, é que o algoritmo de compactação local examina um outro bloco [LAND80].

As técnicas de compactação global examinam o programa integralmente. Elas não se restringem as fronteiras impostas pelos blocos básicos, e por esse motivo é possível migrar instruções de um bloco básico para um outro ([TOKO81], [FISH81] e [NICO85]). Em outras palavras, algoritmos de compactação global permitem empacotar um número maior de instruções nas instruções multifuncionais. Por esse motivo, diversas instruções do programa de aplicação podem ser executadas a cada ciclo de máquina.

Apesar do avanço verificado na área de geração de código paralelo para arquiteturas VLIW, ainda hoje observa-se que o nível de ociosidade de suas unidades funcionais é bastante considerável: na prática podemos constatar que a média de

empacotamento é de duas a três instruções compartilhando a mesma IMF. Essa média depende basicamente do programa de aplicação e do algoritmo de escalonamento. A ocorrência de freqüentes desvios no programa de aplicação é um dos principais fatores limitando o desempenho do algoritmo de escalonamento. Por exemplo, a técnica de compactação global *Trace Scheduling* é capaz de empacotar instruções de diferentes blocos básicos numa mesma IMF. Contudo, esse algoritmo de escalonamento introduz código de reparo para neutralizar os efeitos produzidos por instruções que foram indevidamente executadas (i.e., executadas especulativamente antes da avaliação da condição do desvio).

Ao especificar uma arquitetura VLIW com a capacidade de execução condicional, estávamos interessados em reduzir o custo imposto pelas instruções de desvio. Mais especificamente, ao invés de preenchermos os campos vazios da IMF com NOPs, a capacidade de execução condicional também permite o escalonamento de instruções provenientes de blocos básicos distintos numa mesma IMF, aumentando a taxa de empacotamento das instruções longas. Tendo em vista que o início da execução das instruções constituindo a IMF depende do conteúdo dos *flags* do processador, torna-se desnecessário introduzir código de reparo: somente as instruções pertencentes ao caminho determinado pela instrução de desvio é que serão iniciadas. Adicionalmente, a nova arquitetura poderia atuar como uma importante plataforma para o desenvolvimento e avaliação de alternativas técnicas de geração de código paralelo.

A execução condicional é um tópico bastante atual, e tem sido investigado por diversas universidades e fabricantes de processadores. Recentemente, a equipe da Universidade de Hertfordshire apresentou resultados sobre o efeito da execução condicional em uma arquitetura VLIW ([STEV94] e [GRAY94]). A execução guardada nesse caso, proporcionou uma taxa de aceleração (*speedup*) de até 1.76 durante a execução de programas não numéricos. Na Universidade Wisconsin-Madison, Sohi e Pnevmatikatos [PNEV94] desenvolveram um estudo para avaliar o efeito provocado pela associação do conceito de execução condicional com técnicas de predição de desvios em arquiteturas Super Escalares com algoritmo de escalonamento dinâmico.

Esse trabalho está organizada em cinco seções. A próxima seção descreve a arquitetura CONDEX. A Seção 3 trata da geração de código para nosso modelo de arquitetura. Na Seção 4 descrevemos os experimentos realizados para avaliar a qualidade do código paralelo, e finalmente destinamos a Seção 5 às conclusões.

2 O Modelo CONDEX

Nosso modelo de arquitetura é um processador do tipo VLIW com capacidade de EXecução CONDicional. Por esse motivo, o modelo é denominado CONDEX ([FERN94] e [SANT94]).

O modelo CONDEX, ilustrado na Figura 2.3, representa uma família de máquinas VLIW. Cada componente dessa família consiste de múltiplas unidades funcionais que podem operar em paralelo. O modelo básico é dotado de quatro tipos de uni-

dades funcionais:

- ALU – unidade aritmética e lógica para inteiros;
- FPU – unidade para aritmética em ponto flutuante;
- MEM – unidade de acesso à memória;
- BR – uma única unidade de processamento de desvios.

O modelo CONDEX possui um banco de registradores e diversos conjuntos de indicadores de condição. A partir desse modelo é possível configurar com várias unidades funcionais de cada tipo (exceto a unidade de processamento de desvios, que é unica), máquinas derivadas do modelo básico.

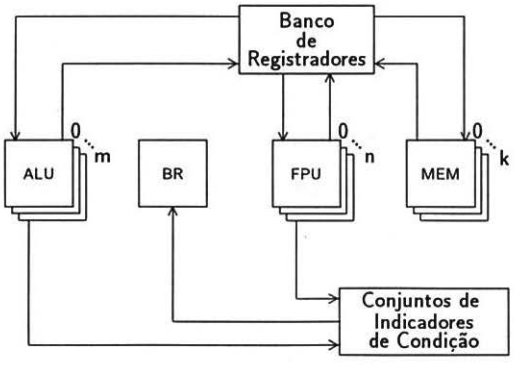

Figura 2.3: Esquema da Arquitetura Modelo – CONDEX

As unidades funcionais do modelo CONDEX, são capazes de executar um conjunto de instruções que se enquadram nas seguintes categorias:

- transferência com memória;
- aritmética com ponto fixo;
- aritmética com ponto flutuante;
- lógicas;
- de desvio;
- transferência entre registradores; e
- as operações NOP e HALT.

Para não limitar o desempenho do modelo, alongando o ciclo de processador para igualar a latência da instrução mais complexa, definimos latências diferenciadas, que vão de 1 até 4 ciclos de máquina, para as diferentes instruções executadas pelas unidades funcionais.

O modelo CONDEX difere de um processador VLIW pela sua capacidade de executar condicionalmente as instruções contidas na IMF. Além do código de operação e dos operandos, cada campo da intrução longa do nosso modelo, possui três campos adicionais que especificam quais as condições que devem ser satisfeitas para que a instrução seja executada. A Figura 2.4 ilustra um campo de uma IMF do modelo CONDEX.

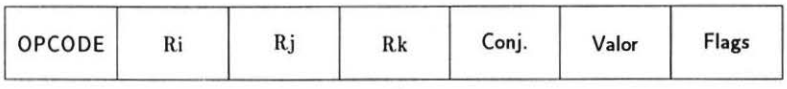

Figura 2.4: Um Campo da IMF do Modelo CONDEX

Na Figura 2.4, os campos "Conj.", "Valor" e "Flags" habilitam a execução da instrução e especificam respectivamente:

- um dos conjuntos de indicadores de condição;
- quais os bits do conjunto devem ser testados;
- os valores que os bits testados devem conter.

Em tempo de execução, os bits especificados no campo "Flags," do conjunto de indicadores de condição selecionado, é comparado com os bits correspondentes no campo "Valor." Se o teste for verdadeiro a instrução é executada pela unidade funcional correspondente ao campo. Caso contrário, a instrução é ignorada. Esse teste é realizado para cada instrução contida na IMF.

O conjunto de indicadores de condição é modificado pelas instruções de comparação (executadas pelas unidades funcionais dos tipos ALU e FPU). As instruções de comparação possuem um campo indicando qual desses conjuntos será afetado.

Na Seção 3 descrevemos a estratégia empregada para geração do código objeto paralelo.

3 A Geração do Código Paralelo

Programas de aplicação escritos em uma linguagem Pascal-*like* são traduzidos para uma forma intermediária pelo compilador. Em seguida, os blocos básicos da forma intermediária, são identificados e submetidos individualmente, ao processo de compactação local que emprega o algoritmo *list scheduling* [LAND80]. Durante esse processo, o algoritmo leva em conta as dependências de dados, a latência de cada operação e o número de unidades funcionais presente na configuração da máquina.

A última etapa que realiza a compactação global, é responsável pela movimentação de instruções além das fronteiras dos blocos básicos.

Os algoritmos *Trace Scheduling* ([FISH81] e [ELLI86]) e *Percolation Scheduling* ([NICO85] e [NICO90]), são exemplos clássicos de algoritmos de compactação global para processadores VLIW. Eles porém não implementam o conceito de execução condicional e por esse motivo não são apropriados para a paralelização do código do nosso modelo de processador.

Para superar essa dificuldade, desenvolvemos uma técnica alternativa de compactação global, denominada conpactação condicional, cuja atuação pode ser assim resumida.

A técnica de compactação condicional emprega o conceito de "bloco sucessor," isto é, o bloco básico para qual o fluxo de controle é dirigido, quando na linguagem de alto nível temos uma estrutura do tipo IF *condição* THEN... (isto é, não existe ELSE correspondente ao THEN), e em tempo de execução, *condição* é verificada como sendo falsa.

O processo de compactação condicional (que atualmente é feita interativamente) consiste em empacotar em uma mesma instrução longa, instruções oriundas de blocos básicos correspondentes as estruturas do tipo "THEN e ELSE," ou "THEN e o bloco sucessor." Durante a ativação de cada IMF serão executadas apenas as instruções cujas condições forem satisfeitas.

Quando aplicado aos blocos THEN ELSE, o algoritmo de compactação condicional produz um único bloco de instruções multifuncionais (i.e., um bloco formado por instruções longas). Essas instruções multifuncionais contêm instruções dos dois blocos originais (i.e., dos blocos THEN e ELSE). Adicionalmente, o algoritmo explora a capacidade de execução condicional do modelo, especificando (em cada instrução multifuncional) quais as instruções que devem ser executadas. O algoritmo introduz nos campos da intrução longa, indicadores de condição que, em conjunto com o conteúdo dos *flags* da máquina, irão habilitar (ou não) a execução da instrução especificada pelo campo. Nesse caso, embora na IMF possam existir instruções pertencentes aos blocos THEN e ELSE, em uma dada instrução multifuncional, serão executadas instruções pertencentes a um dos dois blocos.

Quando aplicada aos blocos THEN e sucessor, o algoritmo produz um único bloco com IMFs contendo instruções dos dois blocos. Como anteriormente, os componentes das instruções longas também possuem informações sobre as condições de execução. Nesse caso porém, as instruções do bloco THEN serão executadas se a condição especificada pelo comando "IF" for verdadeira. Tal não ocorre para as instruções do bloco sucessor. As instruções desse bloco serão executadas independentemente do resultado produzido pela avaliação do "IF."

Em ambos os casos, o processo de compactação condicional emprega o algoritmo *list scheduling* e considera as dependências de dados, disponibilidade dos recursos da máquina e as latências das operações.

4 Os Experimentos

Com o objetivo de avaliar o desempenho do modelo CONDEX, especificamos uma máquina referência e três configurações da máquina CONDEX. A Tabela 4.1 apresenta o número de unidades funcionais de cada configuração. Cada máquina possui quatro conjuntos de indicadores de condição (CIC).

MÁQUINA	ALUs	FPUs	MEMs	BR	CIC
R_1	1	1	1	1	4
M_1	2	1	1	1	4
M_2	4	2	2	1	4
M_3	8	4	4	1	4

Tabela 4.1: As Configurações da Arquitetura do Modelo CONDEX

Durante nossos experimentos, usamos a configuração R_1 (descrita na Tabela 4.1) como máquina referência. Essa máquina possui uma unidade funcional de cada tipo, e executa as instruções seqüencialmente, isto é: uma instrução é iniciada somente depois que a instrução anterior terminou sua execução.

M_1, M_2 e M_3 são máquinas CONDEX que podem executar respectivamente, até 5, 9 e 17 operações em paralelo.

A avaliação de desempenho foi realizada com os seguintes programas de teste:

- *Livermore Loop número 24* – determina o índice do menor componente de um vetor de números inteiros [MCMA83];
- *Branch* – descrito em [DITZ87], conta os números pares e ímpares de um intervalo;
- *BCD-bin* – converte um número inteiro codificado em decimal, para o correspondente em binário [AUHT85];
- *Simpson* – avalia o valor da integral de uma função pelo método de Simpson [CONT65]; e
- *Árvore* – gera uma árvore binária ordenada em um vetor [WIRT76].

Empregando essas configurações da máquina CONDEX e esse conjunto de programas de teste, realizamos dois tipos de experimentos: estáticos e dinâmicos.

4.1 Avaliação Estática do Código Paralelo

De posse do código seqüencial gerado pelo compilador (i.e., o código para a máquina referência), obtivemos código paralelo (produzido pela técnicas de compactação condicional) de cada configuração. A Tabela 4.2 contém o número de instruções longas do código executável de cada configuração.

Programa	R_1	M_1	M_2	M_2
Livermore	133	110	66	54
Branch	98	73	56	51
BCD	235	201	142	111
Simpson	401	308	210	188
Árvore	390	271	197	171

Tabela 4.2: Nº de IMFs do Código Executável

Programa	R_1	M_1	M_2	M_2
Livermore	62	60	60	60
Branch	47	44	44	44
BCD	107	105	105	105
Simpson	174	170	169	169
Árvore	194	184	184	184

Tabela 4.3: Nº de Instruções do Código Executável

Como podemos observar na Tabela 4.2, existe uma redução significativa no tamanho do código estático a medida que o número de unidades funcionais aumenta. Essa redução é proporcionada pela técnica de compactação condicional.

A Tabela 4.3 mostra o número de instruções que foram escalonadas pela técnicas de compactação condicional.

Examinando as Tabelas 4.2 e 4.3, podemos notar que os números de IMFs e de instruções são diferentes: como algumas instruções precisam de mais de um ciclo de máquina para serem completadas, é necessário introduzir NOPs (*No Operate*) para reservar as respectivas unidades funcionais nos ciclos de máquina subseqüentes. A Tabela 4.3 mostra que o código objeto da máquina referência apresenta mais instruções que o das outras máquinas. Isso deve-se ao escalonamento de diversas instruções em uma IMF das máquinas M_1, M_2 e M_3, e a eliminação de instruções de desvio. Essa mesma explicação é valida para o programa *Simpson* que apresenta 170 instruções no código objeto da máquina M_1 e 169 instruções nos das máquinas M_2 e M_3.

A próxima subseção trata da avaliação dinâmica do desempenho do processador CONDEX.

4.2 Avaliação Dinâmica

O código paralelo gerado para cada configuração do modelo CONDEX, foi posteriormente processado pelos respectivos interpretadores, permitindo dessa forma avaliar a variação no tempo de execução de cada programa de teste.

Na Tabela 4.4 mostra o percentual do tempo de execução consumido por cada programa de teste, nas diversas configurações do modelo. Essas percentagens foram calculadas em termos do tempo de execução requerido pela máquina referência. Para facilitar a visualização dos resultados da Tabela 4.4, apresentamos os histogramas nas Figuras 4.5, 4.7, 4.6, 4.8 e 4.9.

Programa	R_1	M_1	M_2	M_2
Livermore	100%	99,92%	59,13%	59,09%
Branch	100%	86,27%	68,62%	66,66%
BCD	100%	100,28%	69,97%	51,27%
Simpson	100%	83,65%	59,89%	55,45%
Árvore	100%	82,59%	61,24%	53,90%

Tabela 4.4: Percentual do Tempo de Execução Consumido

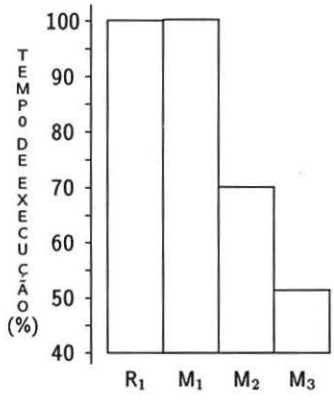

Figura 4.5: Percentuais de Tempo de Execução do Programa *BCD*

Na Tabela 4.4 e na Figura 4.5, observamos que a compactação condicional no caso do programa *BCD* não reduziu o tempo de execução no caso da máquina M_1. Essa anomalia ocorreu pois nesse caso específico foram compactadas instruções de um bloco sucessor com instruções de um bloco THEN. Como na Configuração 1, o número de unidades funcionais presentes é reduzido, o número de instruções do bloco sucessor que foram movimentadas para as IMFs contendo instruções do bloco THEN foi pequeno, não reduzindo o número de IMFs relacionadas com o bloco sucessor que ficaram mais vazias, mas não puderam ser eliminadas, resultando num maior número de instruções longas buscadas quando a condição que habilita as instruções do bloco THEN for falsa. Quando a condição que ativa as instruções do bloco THEN for verdadeira, o número de IMFs buscadas não aumenta (durante a execução desse programa de teste o bloco THEN raramente é executado). Nas

outras duas configurações, a situação é solucionada pela presença de um número maior de unidades funcionais.

Para o programa *Livermore* o problema também está presente, porém com menor gravidade. No caso da máquina M_1 o tempo de execução foi quase igual ao da máquina referência R_1. Na Tabela 4.4 e na Figura 4.6 podemos verificar mais claramente a ocorrência do problema.

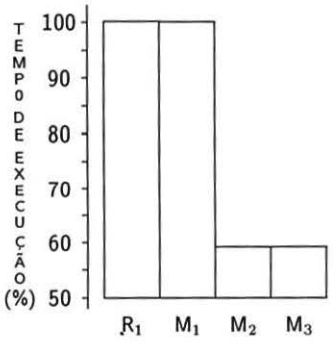

Figura 4.6: Percentuais de Tempo de Execução do Programa *Livermore*

Nos demais programas, obtivemos para todas as configurações do modelo CONDEX, uma redução no tempo de execução.

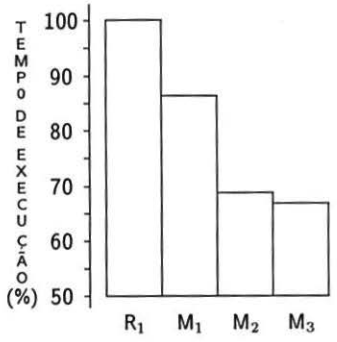

Figura 4.7: Percentuais de Tempo de Execução do Programa *Branch*

A Figura 4.10 apresenta o *speedup* médio obtido em cada configuração (em relação à máquina referência R_1). Na máquina M_1 temos o *speedup* médio de 1,11, na M_2 ele é igual a 1,57 e na M_3 atingimos a marca de 1,76.

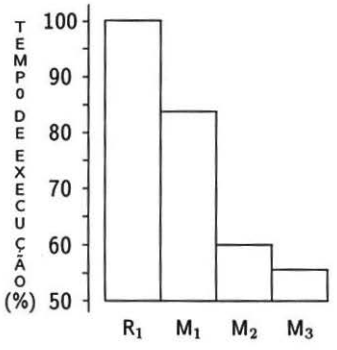

Figura 4.8: Percentuais de Tempo de Execução do Programa *Simpson*

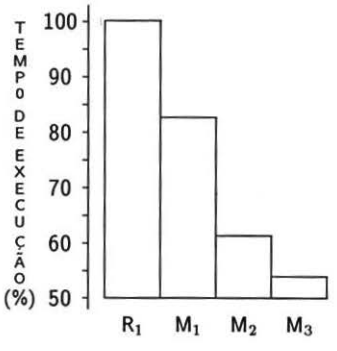

Figura 4.9: Percentuais de Tempo de Execução do Programa *Árvore*

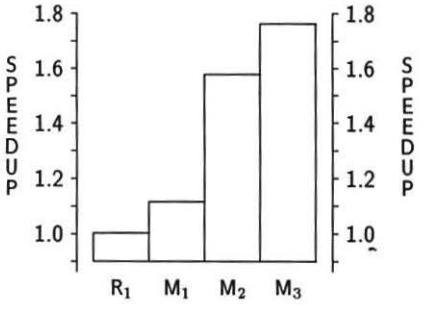

Figura 4.10: Efeito da Execução Condicional – *Speedups* Médios

5 Conclusões

Apresentamos o modelo CONDEX, uma arquitetura VLIW que possibilita a execução condicional. Essa característica do modelo permite que operações provenientes de blocos básicos distintos compartilhem a mesma instrução de máquina do CONDEX, reduzindo o número de NOPs do código objeto.

Variando o número de unidades funcionais, algumas configurações de máquina foram especificadas, e os respectivos códigos paralelos dos programas de teste para essas configurações foram gerados. Interpretando esse código paralelo, avaliamos o desempenho do modelo CONDEX. Durante esses experimentos obtivemos taxas de aceleração variando de 1,11 até 1,76.

Atualmente estamos automatizando o processo de compactação condicional e investigando alternativas de técnicas de compactação baseadas no perfil de execução dos programas.

Referências

[ACOS86] R. D. Acosta, J. Kjelstrup, and H. C. Torng, "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors," IEEE Transactions on Computers, Vol. 35, No. 9, September 1986, pp. 815–828.

[AUHT85] Augustus K. Uht, "Hardware Extraction of Low Level Cocurrency from Sequential Instruction Streams," Ph.D. Thesis, Carnegie-Mellon University, December 1985.

[COLW88] R. P. Colwell, R. P. Nix, J. J. O' Donnell, D. B. Papworth and P. K. Rodman, "A VLIW Architecture for a Trace Scheduling Compiler," IEEE Transactions on Computers, Vol. 37, No. 8, August 1988, pp. 967–979.

[CONT65] S. D. Conte, "Elementary Numerical Analysis," McGraw-Hill Book Co., New York, NY, USA, 1965.

[DITZ87] David R. Ditzel, e Hubert R Mclellan, "Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero," Proceedings of the 14th Annual International Symposium on Computer Architecture, 1987, pp. 2–9.

[ELLI86] John R. Ellis, "Bulldog: A Compiler for VLIW Architectures," ACM Doctoral Dissertation Award 1985, The MIT Press, USA, 1986.

[FERN92] Edil S. T. Fernandes e Anna Dolejsi Santos, "Arquiteturas Super Escalares: Detecção e Exploração do Paralelismo de Baixo Nível," VIII Escola de Computação, 1992.

[FERN94] Edil S. T. Fernandes, Anna Dolejsi Santos and Claudio L. de Amorim, "Conditional Execution: an Approach for Eliminating the Basic Block Barriers," Microprocessing and Microprogramming The Euromicro Journal, North Holland, Vol. 40, Numbers 10–12, December 1994, pp. 689–692.

[FISH81] Joseph A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," IEEE Transactions on Computers, Vol. C-30, No. 7, July 1981, pp. 478–490.

[FISH84] Joseph A. Fisher, "VLIW Machine: A Multiprocessor for Compiling Scientific Code," Computer, July 1984, pp. 45–53.

[GRAY94] S. Gray and R. Adams, "Using Conditional Execution to Exploit Instruction Level Concurrency," Technical Report no. 181, School of Information Sciences, Division of Computer Science, University of Hertfordshire, March 1994.

[JOHN91] M. Johnson,"Superscalar Microprocessor Design," Prentice Hall, 1991.

[KELL75] R. M. Keller, "Look-Ahead Processors," Computing Surveys, Vol. 7, No. 4, December 1975, pp. 177–195.

[LAND80] David Landskov, Scott Davidson, Bruce Shriver, and Patrick W. Mallet, "Local Microcode Compaction Techniques," Computing Surveys, Vol. 12, No. 3, September 1980, pp. 261–294.

[MCMA83] F. H. McMahon, "Fortran Kernels: MFLOPS," Lawrence Livermore National Laboratory,1983.

[NICO85] A. Nicolau, "Percolation Scheduling: A Parallel Compilation Technique," Technical Report TR–85–678, Department of Computer Science, Cornell University, May 1985.

[NICO90] A. Nicolau and R. Potasman, "Realistic Scheduling: Compaction for Pipelined Architectures," Proceedings of the 23rd Annual International Workshop on Microprogramming and Microarchitecture MICRO-23, ACM and IEEE Computer Society, November 1990, pp. 69–79.

[PNEV94] D. N. Pnevmatikatos and G. S. Sohi, "Guarded Execution and Branch Prediction in Dynamic ILP Processors," Proceedings of the 21st Annual International Symposium on Computer Architecture, 1994, pp. 120–129.

[SANT94] Anna Dolejsi Santos, "Efeito da Execução Condicional em Arquiteturas Paralelas," Tese de Doutorado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, 1994.

[STEV94] F. L. Steven, G. B. Steven and L. Wang, "An Evaluation of the iHARP Multiple Instruction Issue Processor," Euromicro 94, September,1994.

[TABA91] Daniel Tabak, "Advanced Microprocessors," Mc Graw-Hill, Inc., USA, 1991.

[TOKO78] M. Tokoro, T. Takizuka, E. Tamura, and I. Yamaura, "A Technique of Global Optimization of Microprograms," Proceedings of the 11th Annual Microprogramming Workshop, 1978, pp. 41–50.

[WIRT76] N. Wirth, "Algorithms + Data Structures = Programs," Prentice-Hall, Inc., 1976.