Efeito do Escalonamento Baseado no Perfil de Programas em Arquiteturas VLIW com Capacidade de Execução Condicional

Anna Dolejsi Santos
Departamento de Ciência da Computação
Universidade Federal Fluminense
Niterói – RJ – Brasil
annads@dcc.uff.br

Edil Severiano Tavares Fernandes
Programa de Engenharia de Sistemas e Computação, COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro – RJ – Brasil
edil@cos.ufrj.br

Sumário

Buscando encontrar métodos alternativos para explorar os recursos existentes no modelo CONDEX, uma arquitetura VLIW (*Very Large Instruction Words*) suportando o conceito da execução condicional, desenvolvemos e avaliamos dois algoritmos de compactação global de instruções. Através deles, operações oriundas de diferentes blocos básicos, podem compartilhar a mesma instrução longa. A seleção dos blocos básicos que serão compactados em conjunto leva em conta o perfil de execução dos programas de aplicação. Visando avaliar os dois métodos de escalonamento para diversas configurações do modelo CONDEX, realizamos experimentos com um conjunto de programas de teste. Para cada configuração, avaliamos inicialmente o efeito estático provocado por esses algoritmos de compactação, i.e., verificamos a variação do número de instruções longas de cada programa de teste. Posteriormente, medimos o efeito dinâmico dessas duas técnicas durante a interpretação dos programas de teste nas configurações usadas.

Abstract

In order to find alternative strategies to increase the use of the the CONDEX machine resources, a VLIW architecture supporting the conditional execution concept, we developed and evaluated two global scheduling techniques. These two schemes allow the placement of instructions proceeding from different basic blocks within the same very large instruction. The selection of the basic blocks which will share the same large instruction takes into account the execution profile of the application program. In order to assess these scheduling methods, several experiments involving a suite of test programs were carried out in different CONDEX configurations. For or each experimental machine we observed the static effect of the compaction algorithms, i.e., we evaluated the variation in the number of very long instruction of each test program. By interpreting the suite of test programs in each machine configuration, we evaluated the impact of the algorithms on the performance of the processors.

1 Introdução

Durante a especificação de uma arquitetura VLIW (*Very Large Instruction Word*), projetistas incluem diversas unidades funcionais que podem operar concorrentemente. Nessas máquinas, o escalonamento das instruções que serão ativadas durante cada ciclo de máquina, é realizado por um algoritmo implementado pelo *software* de suporte da arquitetura. Isto é, o escalonamento (ou compactação) de instruções é realizado durante a fase de geração de código.

Embora novas técnicas de otimização venham sendo desenvolvidas, a geração eficiente de código para máquinas VLIW continua sendo um grande desafio: as técnicas precisam ser melhor avaliadas, e dependendo do resultado dessa avaliação, devem ser incorporadas nos compiladores.

O maior problema encontrado na geração de código eficiente para máquinas VLIW, diz respeito a baixa utilização das suas unidades funcionais, ou seja, é muito difícil preencher todos os campos da instrução longa, e desse modo os recursos do *hardware* não são bem utilizados. Um exemplo dessa baixa utilização foi constatado nos experimentos realizados com o processador Multiflow TRACE 14/300 [COLW88]: uma máquina VLIW com capacidade de iniciar até 13 operações no mesmo ciclo. De acordo com os experimentos descritos por Schuette e Shen [SCHU91], durante a execução de 10 aplicações numéricas, o número médio de operações iniciadas em um ciclo foi menor que 1,4 (dependendo do programa de teste, as médias variaram de 0,85 até 2,41 operações por ciclo). Apesar dessa baixa utilização de recursos, tem ocorrido um incremento no número de projetos de pesquisa envolvendo processadores VLIW. A simplicidade dessa classe de arquiteturas, permite que elas operem com relógios de freqüência muito alta.

Este trabalho trata da geração de código para máquinas VLIW que incorporam o conceito de execução condicional. Nesse modelo, o modelo CONDEX, cada operação da instrução longa tem campos de controle, que indicam as condições em que a unidade funcional correspondente deve ser ativada.

Contribuições importantes no desenvolvimento de máquinas VLIW, incluem: técnicas de compactação local [LAND80] e global [TOKO78], [FISH81] e [NICO85] de código; especificação e implementação de processadores VLIW [FISH81]; a inclusão da capacidade de execução condicional em processadores VLIW [LABR90], [GRAY94], [STEV94], [FERN94], [SANT94] e [WOLF97].

O artigo apresenta as duas técnicas de compactação global de instruções. Usando um conjunto de programas de teste as técnicas foram avaliadas segundo dois diferentes aspectos: a explosão no tamanho do código objeto e o impacto no tempo de execução. Na próxima seção fazemos uma breve descrição das arquiteturas arquiteturas VLIW e do modelo CONDEX. Na Seção 3 mostramos as técnicas de escalonamento desenvolvidas para a geração de código paralelo. Na Seção 4 descrevemos os experimentos realizados em três diferentes configurações do modelo, e destinamos a Seção 5 às conclusões.

2 Modelo CONDEX e Arquiteturas VLIW

Máquinas com instruções longas [FERN92] utilizam algoritmos de escalonamento estático de instruções e surgiram como conseqüência do desenvolvimento de técnicas

de compactação global de micro-programas [TOKO78], [FISH81] e [NICO85].

Durante cada ciclo de processador, a unidade de controle de uma arquitetura VLIW realiza a busca do grupo de instruções que serão executadas, conforme especificado em tempo de geração de código pelo algoritmo de escalonamento de instruções.

Utilizaremos os termos "instrução longa," "Instrução Multifuncional," ou simplesmente IMF para denotar o grupo de instruções que serão iniciadas durante cada ciclo de máquina.

Para cada unidade funcional do processador VLIW, existe um campo na IMF indicando a operação que deverá ser executada pela unidade. Desse modo, as operações indicadas nos vários campos da IMF serão executadas em paralelo. Operações NOPs (*no operate*) precisam ser introduzidas nos campos das unidades funcionais que não devem iniciar uma nova operação quando da execução da instrução longa.

A Figura 2.1 ilustra uma instrução longa. Cada campo desta IMF contém o código de operação (OPCODE) e os operandos da instrução especificada (Ri, Rj e Rk).

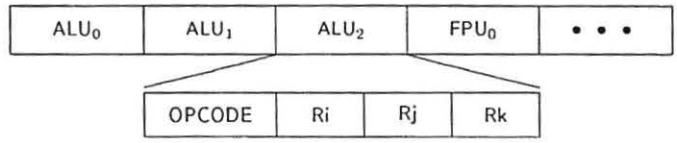

Figura 2.1: Uma Instrução Longa e o Detalhamento de um dos seus Campos

CONDEX é o termo usado para designar nosso modelo de processador "VLIW com capacidade de EXecução CONDicional."

Ao propor um modelo de processador VLIW com capacidade de execução condicional, visávamos eliminar as fronteiras de blocos básicos ([LAND80]) que limitam a detecção e extração do paralelismo a nível de instrução. Em outras palavras, eliminando as limitações impostas pelos blocos básicos, é possível realizar um escalonamento global de instruções por meio de técnicas de compactação local.

O modelo CONDEX ([FERN94], [SANT94], [SANT95], [SANT96], [SANT97] e [SANT97a]), ilustrado na Figura 2.2, representa uma família de máquinas VLIW. Ele consiste de múltiplas unidades funcionais que podem operar em paralelo. O modelo básico inclui:

- ALUs – unidades aritmética e lógica para inteiros;
- FPUs – unidades para aritmética em ponto flutuante;
- MEMs – unidades de acesso à memória;
- BR – uma única unidade de processamento de desvios;
- Reg – um banco de registradores;
- CCCs – conjuntos de indicadores de condição.

Variando o número de unidades funcionais e de conjuntos de códigos de condição, é possível especificar diversas máquinas experimentais derivadas do modelo básico.

As unidades funcionais do modelo executam os seguintes tipos de instruções: acesso à memória, aritmética com ponto fixo, aritmética com ponto flutuante, lógicas, desvio, transferência entre registradores, NOP e HALT.

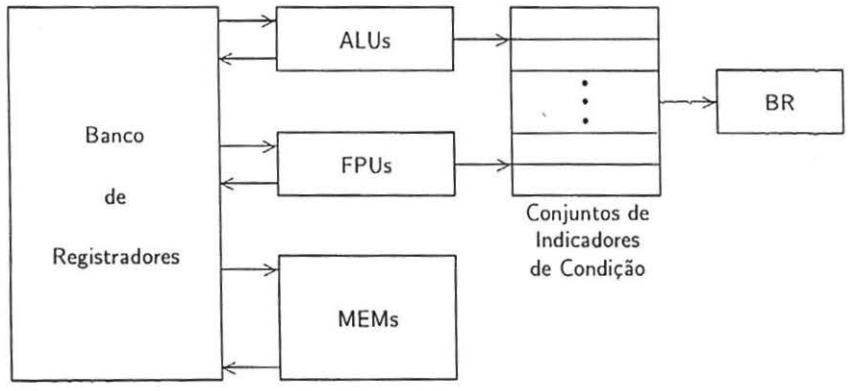

Figura 2.2: Arquitetura do Modelo CONDEX

Para não alongar o ciclo de processador, definimos operações com tempos de latência variando de 1 até 4 ciclos de máquina, dependendo da complexidade da instrução [SANT94].

O modelo CONDEX difere de um processador VLIW pois permite a execução condicional das operações contidas na IMF. Além do código de operação e dos operandos mostrados na Figura 2.1, cada campo da instrução longa possui três campos adicionais que especificam as condições que devem ser satisfeitas para que a instrução seja executada. A Figura 2.3 ilustra o campo de uma IMF do modelo.

OPCODE	Ri	Rj	Rk	Conj.	Flags	Valor

Figura 2.3: Um Campo da Instrução Longa do Modelo CONDEX

Na Figura 2.3, os campos "Conj.", "Flags" e "Valor" podem habilitar a execução da operação e especificam respectivamente:

- um dos conjuntos de indicadores de condição (i.e., conjunto de *flags*);
- os valores que os *flags* devem conter; e
- quais os *flags* que devem ser testados.

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

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

Na Seção 3 examinamos as técnicas empregadas para geração do código objeto paralelo, interpretado posteriormente no simulador do modelo CONDEX.

3 As Técnicas de Escalonamento

Nossos algoritmos de escalonamento global descendem da técnica *Trace Scheduling* [FISH81] e [ELLI86]: técnica que realiza a compactação global de instruções para processadores VLIW e baseia-se no perfil de execução dos programas de aplicação.

O *Trace Scheduling* escalona as instruções ao longo do fluxo de controle do programa com maior probabilidade de execução (ou seja, a técnica explora o paralelismo em trechos, ou *traces*, com maior probabilidade de execução). A escolha dos trechos do programa que serão compactados, baseia-se na probabilidade com que eles são executados: quanto maior a freqüência de execução, maior a prioridade para movimentação de suas operações.

A probabilidade de execução de cada trecho do programa, pode ser fornecida pelo programador, ou determinada por uma heurística. A movimentação de instruções ao longo dos *traces*, obriga a inclusão de código de reparo na última fase do processo de compactação, (i.e., código para anular os efeitos da movimentação de instruções para outros blocos básicos).

Contudo o *Trace Scheduling* não foi projetado para explorar o conceito de execução condicional. Assim tornou-se necessário, desenvolver técnicas de escalonamento alternativas que levassem em consideração essa importante característica.

Inicialmente produzimos código executável para configurações CONDEX, empregando a técnica descrita em [SANT94] e [SANT96], denominada *Compactação Condicional*. Posteriormente, buscamos encontrar métodos alternativos para explorar de forma mais eficiente o paralelismo potencial do modelo. Geramos então código executável, empregando nossas duas novas modalidades de algoritmo de compactação global, baseadas no perfil de execução dos programas de teste.

Chamamos de *Técnica baseada no Perfil de Execução* (ou TPE) ao processo de escalonamento de instruções em IMFs pertencentes a um trecho de programa previamente selecionado. A exemplo do *Trace Scheduling*, o nosso trecho também inclui os blocos básicos mais freqüentemente executados. Em nossos experimentos, a freqüência de execução de cada trecho dos programas, foi determinada interpretando-se os programas de teste seqüencialmente.

A geração de código por meio do processo de compactação TPE, obedece a seguinte seqüência:

- o programa de aplicação escrito em uma linguagem Pascal–*like* é traduzidos para uma forma intermediária;
- os blocos básicos da forma intermediária são identificados;
- através da execução seqüencial, obtem-se a freqüência de execução de cada bloco básico;
- os trechos de programa que incluem os blocos básicos mais freqüentemente executados são selecionados;
- cada bloco básico da forma intermediária é compactado pelo algoritmo (local) *List Scheduling* [LAND80];
- a técnica *List Scheduling* é empregada para mover instruções de cada trecho além das fronteiras dos blocos básicos, respeitando as dependências de dados e restrições no uso de recursos. Durante essa etapa os campos "Conj.", "Flags" e "Valor" são preenchidos convenientemente, e diversas instruções de desvio podem ser eliminadas e outras acrescidas; e

- código de reparo é introduzido para garantir a equivalência semântica do programa, quando o fluxo de controle for diferente daquele determinado pelo *trace*.

Na Figura 3.4 apresentamos o grafo de fluxo de controle (GFC) do programa *Livermore Loop*, usado em nossos experimentos. Cada nó do GFC representa um bloco básico. No interior de cada nó existe a indicação das instruções que fazem parte de cada um deles. Na parte externa de cada bloco, temos o número de vezes que ele foi executado durante a interpretação seqüencial do código. Os seis blocos básicos que fazem parte do *trace* utilizado em nossos experimentos, aparecem no interior da região tracejada.

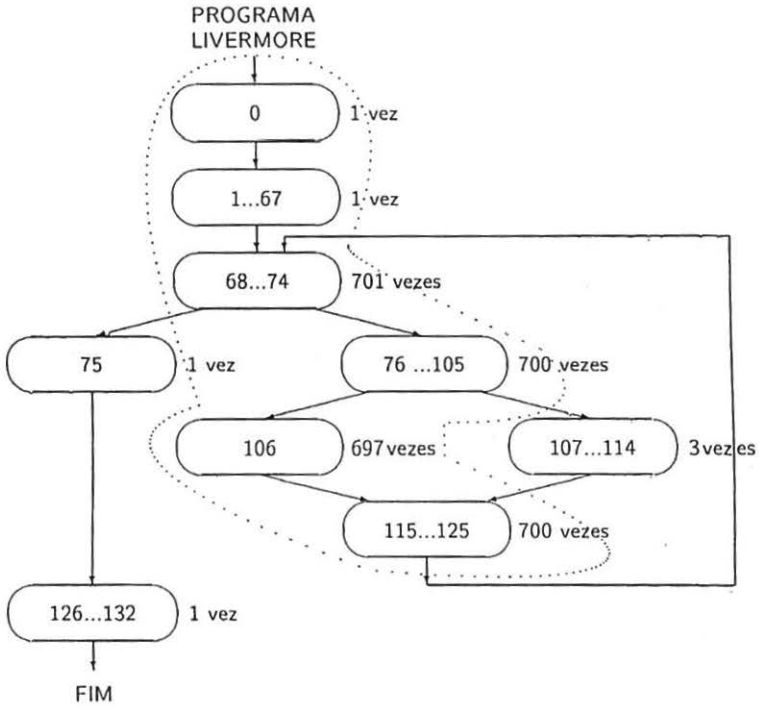

Figura 3.4: *Trace* do Programa *Livermore* Usado na Técnica TPE

A segunda técnica de escalonamento é denominada *Técnica baseada no Perfil de Execução* +, (ou TPE+). O desenvolvimento dessa segunda estratégia teve por objetivo aprimorar a técnica de Compactação Condicional [SANT94], [SANT96] e [SANT97a]. Usamos como entrada para o processo de compactação TPE+, o código previamente submetido à Compactação Condicional.

Para facilitar a descrição do processo TPE+, fazemos uma breve apresentação do algoritmo de Compactação Condicional.

No processo de Compactação Condicional, denominamos "bloco sucessor" (ou "bs"), o bloco básico para onde o fluxo de controle é dirigido quando na linguagem

de alto nível temos uma estrutura do tipo IF *condição* THEN... (isto é, não existe a cláusula ELSE), e em tempo de execução, a *condição* é falsa.

A idéia fundamental da Compactação Condicional consiste em empacotar na mesma instrução longa, instruções oriundas de blocos básicos correspondentes as estruturas em linguagem de alto nível, do tipo "THEN e ELSE," ou "THEN e bloco sucessor". Durante a ativação da instrução longa, são executadas apenas as instruções cujas condições forem verdadeiras.

Para facilitar a apresentação, utilizaremos os termos "bt" e "be" ao invés de "bloco THEN" e "bloco ELSE" respectivamente.

No trecho de programa mostrado na Figura 3.5(a), o bloco básico "bt" é composto pelo comando *odd := odd + 1*, e o bloco "be" é composto pelo comando *even := even + 1*. Nesse caso, ao compactarmos as instruções provenientes dos dois blocos básicos examinados, em tempo de execução é possível ativar somente aquelas instruções cuja condição for satisfeita. Em outras palavras, se *(i and 1) = 1*, as instruções provenientes do "bt" são executadas. Contudo se a condição for falsa então são executadas as instruções do "be."

```
if (i and 1) = 1
  then odd := odd + 1
  else even := even + 1;
```

(a)

```
if x[k] < x[m]
  then m := k;
k := k + 1;
```

(b)

Figura 3.5: Trechos dos Programas *Branch* e *Livermore*

Na Figura 3.5(b) o bloco básico "bt" é formado pelo comando $m := k$, e "bs" pelo comando $k := k + 1$. Dependendo da avaliação da condição, em tempo de execução poderemos ter instruções dos dois blocos sendo executadas simultaneamente, ou somente as do bloco sucessor. Isto é, se x[k] < x[m], então todas as instruções contidas na instrução longa serão executadas. Se a condição for falsa, então apenas as instruções oriundas do "bs" serão executadas. Podemos então concluir que as instruções do "bs" sempre serão executadas, ou seja, serão executadas incondicionalmente.

A geração de código através da Compactação Condicional é realizada do seguinte modo:

- o programa de aplicação escrito em uma linguagem Pascal-*like* é traduzido para uma forma intermediária;
- os blocos básicos e os pares "bt, be" e "bt, bs" são identificados na forma intermediária;
- os blocos básicos da forma intermediária são submetidos individualmente, ao processo de compactação local que emprega o algoritmo *List Scheduling*; e
- é realizada a fusão dos pares "bt, be" e "bt, bs" previamente submetidos à compactação local (essa etapa também utiliza *List Scheduling* para movimentar operações entre blocos básicos).

No processo de Compactação Condicional, instruções de desvio são eliminadas, provocando a fusão de blocos básicos.

Na Figura 3.6 mostramos uma vez mais o GFC do programa *Livermore* e no interior da região tracejada, temos os quatro blocos básicos envolvidos na Compactação

Condicional. O par "bt, bs" (bt inclui instruções 107...114 e bs as instruções 115...125) da região é fundido e devido a eliminação de instruções de desvio, os quatro blocos produzem um único bloco básico de IMFs.

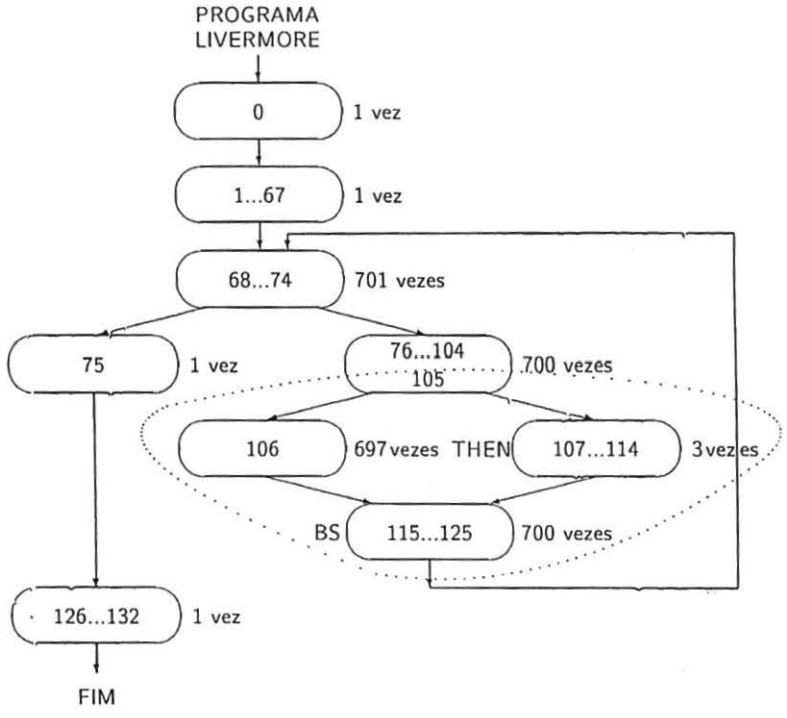

Figura 3.6: Blocos do Programa *Livermore* Envolvidos na Compactação Condicional

Uma vez apresentado o processo da Compactação Condicional, podemos examinar a segunda técnica de compactação que usa o perfil de execução de programas, a TPE+.

Durante a geração de código para o modelo CONDEX através da técnica TPE+, trechos de código com pares de blocos "bt, be" e "bt, bs", escalonados previamente por meio da Compactação Condicional, são submetidos ao nosso novo processo de escalonamento.

O processo TPE+ produz código paralelo do seguinte modo:

- o programa de aplicação escrito em uma linguagem Pascal-*like* é submetido à Compactação Condicional;
- o programa compactado é executado para obtenção da freqüência de execução de cada bloco básico;
- os trechos do programa que incluem os blocos básicos mais freqüentemente executados são selecionados; e
- a técnica *List Scheduling* compacta instruções de cada trecho além das fronteiras dos blocos básicos. Nessa fase, diversas instruções de desvio são elimi-

nadas e os campos "Conj.", "Flags" e "Valor" são preenchidos convenientemente.

Na Figura 3.7 temos o GFC do programa *Livermore* resultante da Compactação Condicional e os blocos básicos que fazem parte do *trace* utilizado pela técnica TPE+. A convenção empregada nessa ilustração, é a mesma da Figura 3.4.

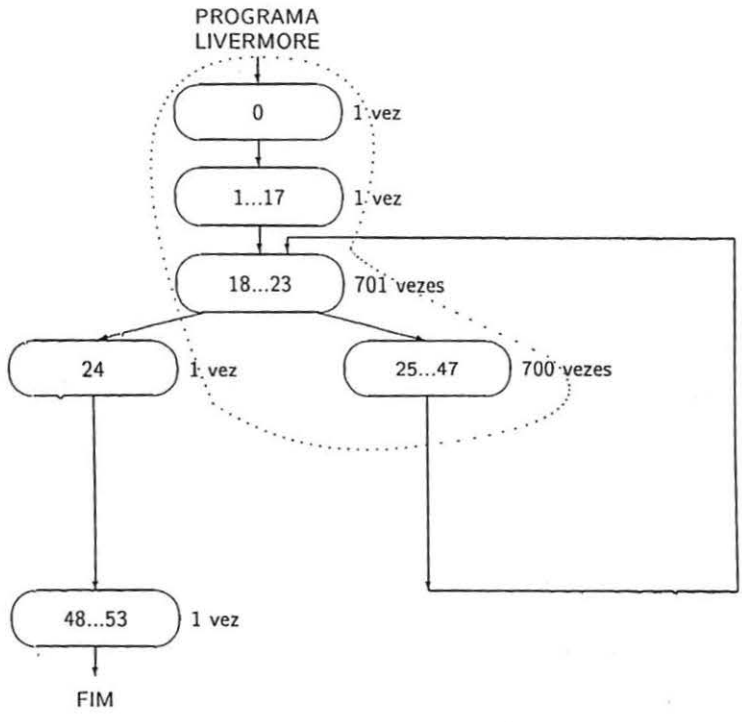

Figura 3.7: *Trace* do Programa *Livermore* Usado na Técnica TPE+

Na Figura 3.7 observamos que os quatro blocos básicos, (mostrados na Figura 3.6), ficaram reduzidos a um único bloco e desse modo as instruções que constituem o bloco THEN também passam a fazer parte *trace* selecionado.

Na Figura 3.7 vemos que a numeração no interior de cada nó difere da mostrada na Figura 3.6. Isso se deve ao fato de que na Figura 3.7 mostramos as instruções longas, que resultaram do processo de Compactação Condicional (correspondente a máquina M_3 descrita na Seção 4), ao passo que na Figura 3.6 a numeração das instruções refere-se ao código seqüencial.

Concluímos então que as técnicas TPE e TPE+, diferem entre si pelo código fornecido para a compactação, isto é, TPE utiliza os blocos básicos compactados localmente, ao passo que a TPE+ gera código a partir de programas que foram previamente processados pela Compactação Condicional.

A seguir descrevemos os experimentos realizados para avaliar as técnicas de escalonamento apresentadas.

4 Os Experimentos

Para avaliar o impacto produzido pelos algoritmos TPE e TPE+ no tamanho de código e no tempo de processamento de uma bateria de programas de teste, especificamos uma máquina de referência (R_1) e três configurações (M_1, M_2 e M_3) do nosso modelo. A Tabela 4.1 apresenta o número de unidades funcionais de cada configuração.

Durante os experimentos, usamos a configuração R_1 como máquina de referência. Essa máquina possui uma unidade funcional de cada tipo, e executa as instruções seqüencialmente: uma instrução é iniciada somente depois do término da anterior.

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

Tabela 4.1: As Configurações da Arquitetura do Modelo CONDEX

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

Os experimentos foram realizados com os seguintes programas de teste: *Livermore Loop número 24* [MCMA83], *Branch* [DITZ87], *Simpson* [CONT65], *BCD.bin* [UHT85] e *Árvore* [WIRT76].

Inicialmente cada programa de teste foi interpretado seqüencialmente na máquina de referência (R_1) e os trechos mais freqüentemente executados foram identificados. Posteriormente através das técnicas TPE e TPE+, obtivemos as formas interpretáveis para as três configurações CONDEX mostradas na Tabela 4.1. Isto é, produzimos para cada programa de teste e para cada uma máquina, três formas executáveis geradas por meio das técnicas: Compactação Condicional, TPE+ e TPE.

O número médio de instruções longas produzido pelos processos de Compactação Condicional, TPE+ e TPE é apresentado na Tabela 4.2. Na tabela, o termo Cond designa a técnica de Compactação Condicional.

Máquina	Nº de Médio de IMFs		
	Cond	TPE+	TPE
M_1	192,6	188,2	204,8
M_2	134,2	129,8	147,2
M_3	115,0	110,6	130,0

Nº Médio de Instruções para R_1: 251,4

Tabela 4.2: Nº Médio de IMFs

Examinando os dados da Tabela 4.2, vemos que o código produzido pela estratégia TPE sempre contém mais IMFs que as outras duas. A inclusão de código

de reparo para garantir a equivalência semântica dos programas, é a responsável por esse aumento. Para qualquer configuração, a técnica TPE+ produz código menor que a Compactação Condicional. Usando como critério a explosão de código, a técnica TPE+ é a mais eficiente das três.

A Tabela 4.3 mostra a variação média percentual do número de instruções longas. O percentual de referência corresponde ao tamanho do código produzido pela Compactação Condicional.

Podemos verificar na Tabela 4.3 que a redução no tamanho do código gerado pela técnica TPE+, no melhor caso (M_3) não chegou a 4%. O aumento no número de instruções provocado pelo emprego de TPE foi no pior caso de 13%. Podemos concluir que variação no número de IMFs não chega a ser relevante, visto que a capacidade das memórias vem aumentando ao longo do tempo, sem a proporcional elevação de custo.

Máquina	Variação Percentual (%)		
	Cond	TPE+	TPE
M_1	100,0	97,7	106,3
M_2	100,0	96,7	109,7
M_3	100,0	96,2	113,0

Tabela 4.3: Variação Média Percentual no Tamanho do Código Obtido

Para avaliar o tempo médio dispendido na execução do nosso conjunto de programas de teste, interpretamos cada um deles nas diversas configurações da arquitetura. Obtivemos assim o número médio de ciclos.

Na Tabela 4.4 listamos o número médio de ciclos requeridos durante a interpretação dos programas de teste em cada configuração CONDEX. Na parte inferior da tabela apresentamos o número médio de ciclos dispendidos na interpretação seqüencial em R_1.

Máquina	Nº Médio de Ciclos		
	Cond	TPE+	TPE
M_1	89.356,4	87.956,6	86.293,8
M_2	64.654,2	63.256,6	61.031,8
M_3	58.999,6	57.602,0	54.935,2

Nº Médio de Ciclos em R_1: 101.769,6

Tabela 4.4: Nº Médio de Ciclos Consumidos pela Bateria de Teste

Podemos observar na Tabela 4.4 que o tempo médio de interpretação em qualquer configuração CONDEX é sempre menor se o código for proveniente da técnica TPE. O código gerado pela técnica TPE+, embora não seja tão eficiente quanto o produzido por TPE, sempre é melhor que o obtido por meio da Compactação Condicional.

Se considerarmos que o *speedup* na interpretação seqüencial é 1, temos na Tabela 4.5 a aceleração resultante do emprego das três técnicas de geração de código

paralelo, desenvolvidas. Quando o código produzido pela técnica TPE é interpretado na máquina M_3, um *speedup* de 1,852 é atingido, ou seja o tempo de execução é reduzido a 54% do tempo necessário para a execução seqüencial. Na mesma máquina M_3, o tempo de execução é de cerca 57% e 58% do da execução seqüencial quando os códigos são produzidos respectivamente pelos algoritmos TPE+ e de Compactação Condicional.

Máquina	Seedup		
	Cond	TPE+	TPE
M_1	1,138	1,157	1,179
M_2	1,574	1,608	1,667
M_3	1,725	1,766	1,852

Tabela 4.5: *Seedups* Observados Durante a Interpretação

A redução no tempo de execução na máquina M_3, provocado pelo emprego da Compactação Condicional, TPE+ e TPE, pode ser melhor observada na Figura 4.8.

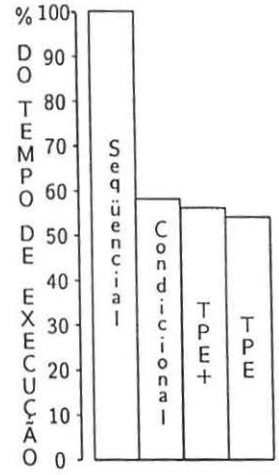

Figura 4.8: Redução no Tempo de Execução em M_3

5 Conclusões

Arquiteturas VLIW com capacidade de execução condicional são uma alternativa promissora para a exploração do paralelismo a nível de instrução. Nessas máquinas é possível preencher os campos das IMFs com operações oriundas de blocos básicos distintos. Por esse motivo, além de eliminar muitas instruções de desvio, podemos ter um número maior de operações sendo executadas concorrentemente.

Nesse trabalho apresentamos duas técnicas de compactação global (TPE e TPE+), e mostramos como elas afetam o tamanho do código executável dos programas de aplicação e o desempenho de uma família de máquinas derivadas do modelo CONDEX: um processador VLIW que implementa o conceito de execução condicional.

Ao desenvolver essas duas técnicas de escalonamento global, procuramos obter métodos de geração de código (para o modelo CONDEX) mais eficientes que o produzido pela técnica de Compactação Condicional.

Considerando o tamanho do código produzido pela Compactação Condicional, observamos que o método TPE provoca uma explosão máxima de código de cerca 13%, enquanto a técnnica TPE+, reduz o código em até 3,8%. Essa variação no número de IMFs não chega a ser relevante se considerarmos que o tamanho das memórias vem aumentando expressivamente, sem a correspondente elevação no custo.

Certamente o aspecto mais relevante dos nossos experimentos, diz respeito à redução no tempo médio de execução dos programas de aplicação. Obtivemos taxas de aceleração de 1,852 na configuração M_3 do modelo CONDEX, com o código produzido pela técnica TPE.

Convém lembrar que para uma completa avaliação das técnicas de escalonamento apresentadas, é necessário realizar um maior número de testes, utilizando mais programas de aplicação e um maior número de configurarações CONDEX. Nosso trabalho prossegue nessa direção.

Referências

[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 and 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*, livro da VIII Escola de Computação, Gramado, RS, Agosto de 1992, 155 páginas.

[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. 668–692.

[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.

[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.

[LABR90] J. Labrousse and G. Slavenburg "A 50MHz microprocessor with a VLIW architecture," Proceedings of the International Solid State Cicuits Conference, San Francisco, 1990.

[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 Tçchnique," Technical Report TR–85–678, Department of Computer Science, Cornell University, May 1985.

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

[SANT95] Anna Dolejsi Santos e Edil S. T. Fernandes, "Extração do Paralelismo em Arquiteturas com Capacidade de Execução Condicional," VII Simpósio Brasileiro de Arquitetura de Computadores – Processamento de Alto Desempenho, (VII SBAC-PAD), 1995, pp. 77–99.

[SANT96] Anna Dolejsi Santos e João Francisco Pereira Neto, "Um Gerador Automático de Código para Arquiteturas VLIW com Capacidade de Execução Condicional," VIII Simpósio Brasileiro de Arquiteturas de Computadores e Processamento de Alto Desempenho (VIII SBAC-PAD), Agosto de 1996, pp. 77–86.

[SANT97] Anna Dolejsi Santos e Edil S. T. Fernandes, "A Ocupação das Unidades Funcionais de Arquiteturas VLIW com Capacidade de Execução Condicional," XXIV Seminário de Software e Hardware (XXIV SEMISH), Agosto de 1997, pp. 25–36.

[SANT97a] Anna Dolejsi Santos, Andrew Wolfe and Edil S. T. Fernandes, "Functional Units Utilization in a Multiple-Instruction Issue Architecture," Aceito como Short Note pela 23rd Euromicro Conference, Hungria, 1997. Será publicado no Journal of Systems Architecture, 11 pages.

[SCHU91] Michael A. Schuette and John P. Shen, "An Instruction-Level Performance Analysis of the Multiflow Trace 14/300," Proceedings of the 24th Annual International Symposium on Microarchitecture, November 1991, pp. 2–11.

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

[TOKO78] M. Tokoro, T. Takizuka, E. Tamura, and I. Yamaura, "A Technique of

Global Optimization of Microprograms," Proceedings of the 11th Annual Micro- programming Workshop, 1978, pp. 41–50.

[UHT85] Augustus K. Uht, "Hardware Extraction of Low Level Cocurrency from Sequential Instruction Streams," Ph.D. Thesis, Carnegie-Mellon University, December 1985.

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

[WOLF97] A. Wolfe, J. Fritts, S, Dutta and E. S. T. Fernandes, "Datapath Design for a VLIW Video Signal Processor," Proceedings of the Third High Performance Computer Architecture Conference, February 1997, USA, 12 pages.