Implementação de Decodificadores de Código em Fluxo ASIC Digital

Aluno: Felipe Prado Yonehara

Orientador: Prof. Dr. Ricardo Ribeiro dos Santos

21 de Março de 2011

1 Apresentação e Justificativa

Um grande número de técnicas foram desenvolvidas na tentativa de minimizar o problema de Memory Wall [5][12][15]. Muitas delas focaram em novos esquemas de codificação enquanto outras utilizavam técnicas para redução do tamanho dos programas, como as técnicas de compressão. Todas possuem um unico objetivo: reduzir a quantidade de dados armazenados na memória principal. Especificamente, estas técnicas têm sido aplicadas em arquiteturas que buscam grandes instruções da memória e em arquiteturas voltadas para domínios específicos de aplicações, como os sistemas embarcados [6][7][14][8].

A técnica de codificação de instruções PBIW (*Pattern Based Instruction Word*) [1] é voltada para arquiteturas que buscam instruções longas na memória como processadores EPIC, arquiteturas reconfiguráveis com várias unidades funcionais e arquiteturas de alto desempenho baseada em matriz de unidades funcionais. A técnica é composta por um algoritmo baseado em fatoração de operandos [9][4][3] e por uma memória cache chamada de *Pattern Cache* (P-cache). Parte da solução da proposta consiste em utilizar uma cache de padrões cuja estrutura foi definida a partir de estudos sobre desempenho em caches.

Este plano de trabalho volta-se para a implementação em hardware de técnicas de codificação de instruções visando otimizar o uso de memória por programas. Em particular, o trabalho focará a implementação dos módulos de cache de padrões (P-*cache*) e decodificação de instruções em uma plataforma de hardware com tecnologia ASIC (Application Specific Integrated Circuits) e a avaliação dessa implementação. A escolha da tecnologia ASIC como plataforma para implementação e síntese em hardware se dá pela necessidade de se obter um sistema que possa ser reproduzido em larga escala com tecnologia CMOS (Complementary metal-oxide Semiconductor), baseando-se no fluxo de desenvolvimento digital cuja infraestrutura do CTEI (Centro de Tecnologia em Eletrônica e Informática) oferece. O intuito é verificar o comportamento da componente de cache (P-cache) de PBIW sob um hardware específico em plataforma ASIC a fim de determinar taxas de *hit*, taxas de *miss*, tempo de acesso, análise de power e área ocupada. Este projeto esta relacionado com o desenvolvimento de um projeto maior voltado para o desenvolvimento de técnicas, algoritmos e implementação em hardware em arquiteturas avançadas de computadores. Além disso, esta proposta estende um plano de trabalho anterior desenvolvido no âmbito do GPEC/UCDB e que focou o estudo e comportamento estatístico da técnica PBIW

e seus algoritmos.

2 Objetivos

Como objetivo geral, deseja-se estender a técnica PBIW para a codificação de instruções 2D-VLIW de maneira a melhorar a qualidade da solução e o desempenho.

Os objetivos específicos são:

1. Estudar a literatura da área envolvendo codificacção de instruções e envolvendo fluxo ASIC digital.
2. Compreender o funcionamento da técnica PBIW e seus algoritmos.
3. Implementar o fluxo ASIC digital inicial (frontend) a partir de uma componente de cache de padrões (P-cache) e um hardware para decodificação de instruções codificadas implementada em linguagem de descrição de Hardware, devidamente verificado em nível de simulação.
4. Analisar o desempenho do hardware implementado.
5. Preparar, organizar e escrever artigos e documentos científicos para serem submetidos em eventos das áreas relacionadas com este trabalho.

3 Revisão da Literatura

Esta Seção apresenta os conceitos básicos para o entendimento e desenvolvimento deste projeto. Esses conceitos estão organizados em duas subseções: Codificação de Instruções e Técnica PBIW.

3.1 Codificação de Instruções

Trabalhos não muito recentes já mostravam que a taxa de melhora na velocidade dos microprocessadores supera a taxa de melhora na velocidade das memórias DRAM (Dynamic Random Access Memory) [11]. A velocidade dos processadores aumentava aproximadamente 80% ao ano, enquanto que a velocidade das DRAMs aumentava apenas 7% ao ano. Entretanto, boa parte da comunidade de arquitetura de computadores ainda está focada em aumentar o desempenho dos

processadores. Como resultado, a diferença entre a velocidade do processador e da memória tem crescido exponencialmente levando a um fenômeno conhecido como Memory Wall [5][12][15].

Para sobrepor o problema de Memory Wall, diversas técnicas focaram em novos esquemas de codificação enquanto outras utilizavam técnicas para redução do tamanho dos programas, como as técnicas de compressão. Todas possuem um unico objetivo: reduzir a quantidade de dados armazenados na memória principal. Especificamente, estas técnicas têm sido aplicadas em arquiteturas que buscam grandes instruções da memória e em arquiteturas voltadas para domínios específicos de aplicações, como os sistemas embarcados [6][7][14][8].

Trabalhos anteriores sobre redução de código utilizaram conceitos e técnicas da área de compressão de código. Existem várias propostas, como em [10], descrevendo técnicas de compressão com foco em arquiteturas VLIW (Very Long Instruction Word).

Trabalhos como em [2] mostraram que é possível obter redução no tamanho do código e no consumo de energia através de uma compressão de código eficiente e um descompressor bastante simples.

No entanto, boa parte das propostas que tentam reduzir os programas causam queda no desempenho do processador devido ao overhead da descompressão.

Em [10] é apresentada uma técnica de compressão baseada em dicionário que utiliza isomorfismo de instrução. Sua abordagem consiste em selecionar as instruçõess mais frequentes e quebrá-las em opcodes e operandos, armazenando-os separadamente em dois dicionários. A lógica de decodificação acrescenta um novo estágio no datapath do processador, ao contrário da técnica PBIW que permite que a decodificação ocorra em paralelo a outras atividades do datapath.

A abordagem utilizada em PBIW difere de estratégias que reduzem o número de instruções como a técnica Instruction Collapsing [13], onde as instruções dependentes são agrupadas em uma única instrução e cujos experimentos mostram aumentos de até 20% no IPC (Instruction Per Cycle). A técnica PBIW explora a sobrejeção entre instruções e seus padrões (reuso de padrões) com o objetivo de reduzir o tamanho da instrução em memória e o tamanho da cache de padrões.

3.2 Técnica PBIW

A técnica PBIW fatora as operações de um programa em instruções (conjunto de operações) codificadas e padrões. Diferente da abordagem em [9], a técnica PBIW armazena padrões fatorados em uma cache que pode ser acessada em paralelo à execução de outras atividades no datapath. A preparação da instrução utilizada nos estágios de execução é mais simples do que alguns mecanismos tradicionais de descompressão pois a instrução na memória possui uma referência para seu respectivo padrão na P-cache.

PBIW é indicada para arquiteturas que buscam instruções longas na memória. A técnica é composta por um algoritmo baseado em fatoração de operandos [9][4][3] e por uma memória cache chamada de Pattern Cache (P-cache).

Depois que as atividades de escalonamento de instruções e alocação dos registradores são desenvolvidas pelo back-end do compilador, o algoritmo de codificação PBIW extrai operandos redundantes entre as operações, criando uma instrução codificada sem redundância de operandos. Essa instrução é armazenada em uma I-cache. O algoritmo mantem o registro da posição original dos operandos criando um padrão de instrução que é armazenado na P-cache. Esse padrão é uma estrutura de dados que contem informações necessárias para compor a instrução utilizada nos estágios de execução.

A codificação de uma instrução para o esquema PBIW ocorre durante a fase final de compilação do programa. O compilador percorre todas as operações da instrução e, baseado no conjunto de operandos utilizados, cria a instrução codificada e o respectivo padrão. O algoritmo não verifica dependências de dados entre as instruções pois elas estão diretamente associadas a arquitetura e, por isso, devem ser resolvidas estaticamente (pelo compilador) ou dinamicamente (pelo hardware).

Após encerrar o algoritmo de codificação, obtem-se um conjunto de instruções codificadas e um conjunto de padrões, sendo que cada um desses padrões pode ser compartilhado por mais de uma instrução. Porém, alguns destes padrõs podem possuir muitos espaços vazios dependendo das restrições arquiteturais aplicadas durante o escalonamento das operações. Com o objetivo de obter um conjunto de padrões menor e mais denso e, consequentemente, reduzir o tamanho do programa, utiliza-se uma técnica para unir padrões [1]. A junção de padrões e uma otimização que une dois padrões quando a soma das operações de ambos é menor ou igual a N (número de

unidades funcionais da arquitetura). Após a junção de dois padrões, as instruções que apontavam para os padrões antigos devem apontar para o novo padrão originado a partir da junção. Mais importante, cada instrução deve anular operações no padrão que não serão utilizadas durante a execução. Assim, deve-se indicar em cada instrução (campo "Bits Anulados"), quais operações serão executadas e quais nao serão executadas (anuladas).

No estágio de decodificação, enquanto os registradores globais são lidos, busca-se um padrão de acordo com o campo de endereço presente na instrução codificada. Este campo indica a posição do padrão dentro da cache de padrões. O padrão é buscado na memória caso não se encontre na P-cache. A Figura 1 apresenta os passos básicos para decodificação de uma instrução PBIW.

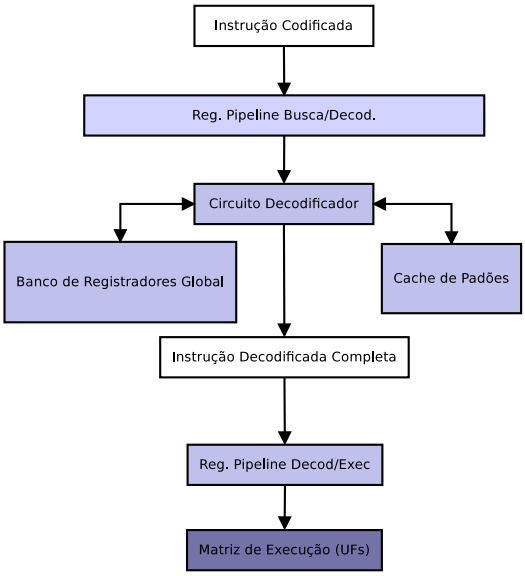

Figura 1: Fluxo de decodificação de uma instrução PBIW.

Depois que o padrão e recuperado da cache de padrões, o hardware de decodificação utiliza os ponteiros que estão armazenados no padrão e monta a instruções utilizada nos estágios de execução. Durante a decodificação, a instrução codificada funciona como um dicionário de operandos para os ponteiros do padrão. Deve-se observar que a leitura dos registradores globais é realizada concomitante com a decodificação da instrução PBIW.

4 Metodologia

A metodologia para desenvolvimento deste plano de trabalho de Iniciação Científica toma como base os objetivos estabelecidos na Seção 2.

1. Estudo da literatura da área e relacionar PBIW com outras alternativas para codificação de instruções.
   (a) Estudo teórico envolvendo arquitetura de computadores e técnicas de codificação de instruções.
   (b) Estudo envolvendo projeto de hardware, a linguagem VHDL e ASICs.
2. Utilização da infraestrutura PBIW de forma familiarizar-se com as ferramentas atualmente disponíveis e visualizando os resultados através da simulação em software.
   (a) Experimentação das características já implementadas.
3. Projeto e implementação da cache de padrões e mecanismo de decodificação PBIW em uma parte do fluxo digital ASIC (frontend).
   (a) Simulação da cache de padrões em hardware.
   (b) Síntese do código RTL (Register Transfer Level) escrito em VHDL.
   (c) Experimentação e validação da implementação.
4. Preparar artigos e documentos científicos para serem submetidos em eventos da área.
   (a) Organização e escrita de artigos científicos para submissão para revistas e eventos cientifícos da área.

5 Cronograma

A metodologia para desenvolvimento deste plano de trabalho de Iniciação Científica toma como base os objetivos estabelecidos na Seção 2.

Atividade	2011				
	Mar	Abr	Mai	Jun	Jul
1.a	x	x			
1.b	x	x			
2.a			x	x	
3.a			x	x	
3.b				x	x
3.c				x	x
4.a					x

Tabela 1: Cronograma para execução do plano de trabalho.

Referências

[1] R. Battistela. Pbiw: Um esquema de codificação baseado em padrões de instrução. Master's thesis, Instituto de Computação - Universidade Estadual de Campinas, 2008.

[2] E. Billo; R. Azevedo; G. Araujo; P. Centoducatte; and E. Wanderley Netto. Design of a decompressor engine on a sparc processor. In *In Procs. of the 18th SBCCI*, 2005.

[3] M. Franz and K. Thomas. Slim binaries. *Communications of the ACM*, 40(2):"87–94", 1997.

[4] J. Ernst; W. Evans; C. W. Fraser; S. Lucco; and T.A. Proebsting. Code compression. In *Proceedings of the SIGPLAN Programming Language Design and Implementation*, June 1997.

[5] S. A. McKee. Reflections on the memory wall. In *Proceedings of the 1st Conference on Computing Frontiers*, pages 162–167. ACM, April 2004.

[6] S. K. Menon and P. Shankar. Space/time tradeoffs in code compression for the tms320c62x processor. Technical report, Indian Institute of Science, 2004. IISc-CSA-TR-2004-4.

[7] R. Montserrat and P. Sutton. Compiler optimization and ordering effects on vliw code compression. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES 2003)*. ACM, 2003.

[8] R. Montserrat and P. Sutton. Code compression based on operand-factorization for vliw processors. In *Proceedings of the Data Compression Conference (DCC 2004)*, 2004.

[9] G. Araujo; M. Cortes; R. Pannain. Code compression based on operand factorization. In *Proceedings 31st International Symposium on Microarchitecture*. ACM, 1998.

[10] S-J. Nam; I-C. Park; and C-H. Kyuong. Improving dictionary-based code compression in vliw architectures. *IEICE Transactions on Fundamentals*, E82-A(11):2318–2324, november 1999.

[11] John L. Hennessy; David A. Patterson. *Computer Organization Design: The Hardware/Software*. Morgan Kauffman Publishers Inc., third edition, 2007.

[12] A. Saulsbury; F. Pong; and A. Nowatzyk. Missing the memory wall: the case for procesor/memory integration. In *Proceedings of the 23rd International Symposium on Computer Architecture*, pages 90–101, October 1996.

[13] Peter G. Sassone and D. Scott Wills. Dynamic strands: Collapsing speculative dependence chains for reducing pipeline communication. In *Procs. of the 37th IEEE/ACM MICRO*, pages 7–17, Washington, DC, USA, 2004. IEEE.

[14] J. Prakash; C. Sandeep; P. Shankar; and Y.N. Srikant. Experiments with a new dictionaty base code-compression tool on a vliw processor. Technical report, Indian Institute of Science, 2004.

[15] W.A. Wulf and S.A. McKee. Hitting the memory wall: Implications of the obvious. *ACM SigArch*, 1995.