Arquiteturas VLIW*

Uma Alternativa Para a Exploração de ILP

Rafael Augusto Scaraficci (RA: 009649)
Instituto de Computação - UNICAMP
Caixa Postal 6176
Campinas - SP, Brasil
ra009649@students.ic.unicamp.br

RESUMO

Os processadores VLIW possuem múltiplas unidades funcionais, permitindo-lhes executarem várias operações por ciclo de clock. No entanto, diferentemente dos processadores superescalares, o hardware do VLIW é bem simples e o escalonamento das instruções é de total responsabilidade do compilador. Neste trabalho é descrito os conceitos fundamentais da arquitetura VLIW assim como as suas vantagens e desvantagens. Também focamos nas principais técnicas de exploração estática de ILP (*Instruction Level Paralellism*), uma vez que o desempenho da arquitetura é totalmente depende do compilador. As técnicas abordadas são: *Loop Unrolling*, *Software Pipelining*, *Trace Scheduling*, *SuperBlock Scheduling* e *HyperBlock Scheduling*. O trabalho é finalizado com uma perspectiva futura sobre o desenvolvimento e a comercialização dos processadores VLIW.

Termos Gerais

Arquitetura de Computadores

PALAVRAS CHAVES

VLIW, Paralelismo em Nível de Instrução, Escalonamento Estático

1. INTRODUÇÃO

Very Long Instruction Word, mais conhecida como VLIW é um paradigma que propõe uma implementação em hardware simples e uma alta capacidade de processamento. As arquiteturas VLIW tentam alcançar um alto nível de ILP (*Instruction Level Parallelism* [1]) através da execução de instruções longas, que são compostas de múltiplas operações independentes pré-escalonadas pelo compilador. A CPU VLIW, diferentemente da superescalar, não contém nenhum hardware especializado para escalonar ou verificar dependências entre as instruções, toda essa complexidade lógica é transferida ao compilador.

A arquitetura VLIW surgiu no final da década de 70 como uma evolução do microcódigo horizontal. Mas foi o trabalho de Joseph Fisher sobre *trace scheduling* [2] que impulsionou o desenvolvimento deste novo conceito de arquitetura.

Motivados com os resultados, Fisher e alguns colegas de Yale fundaram em 1984 a Multiflow, com o objetivo de criar supercomputadores VLIW. No mesmo ano, Bob Rau fundou a Cydrome que também tinha o mesmo objetivo. Apesar de ambas empresas lançarem produtos finais, o mercado ainda não estava pronto para absorvê-la, além disso a tecnologia proposta era muito prematura na época. Isto fez com que as empresas entrassem em crise financeira e encerrassem as suas atividades anos mais tardes.

Com o fechamento dessas empresas, o desenvolvimento da tecnologia VLIW procedeu-se de maneira lenta, até que em meados da década de 90, descobriu-se que a arquiteturad VLIW eram ideais para o processamento de algoritmos complexos e repetitivos. Isto re-impulsionou a produção de processadores VLIW, principalmente voltados para o processamento digital de sinais e multimídia. O primeiro grande sucesso de um processador VLIW, foi a série C6X da Texas Instruments.

O grande sucesso dos DSPs da Texas Instruments, alavancou o projeto de uma nova geração de processadores VLIW. Em 2000, a Transmeta lançou os processadores Crusoe, sendo o foco deste projeto o mercado de processadores para dispositivo embarcados. Ademais, a Intel manteve a mesma perpectiva, dando continuidade a linha Intanium.

O ressurgimento da tecnologia VLIW é uma conseqüência da tecnologia certa disponível no tempo certo. Em meados da década de 70, a arquitetura VLIW era praticamente implementável, devido ao preço da memória naquela época. Enquanto uma instrução VLIW tinha centenas de bits, uma instrução CISC (tecnologia adotada na época) podia ser muito pequena. No entanto, nos dias de hoje, a memória deixou de ser um problema e a arquitetura VLIW passou novamente a ser uma solução viável.

*Trabalho da disciplina MO401 - Arquiteturas de Computadores I, ministrada pelo professor Paulo Cesar Centoducatte, 1^o semestre de 2006

[1]Uma tradução deste termo é Paralelismo em Nível de Instrução. Todavia, optou-se por manter o termo em inglês para se fazer uso da sigla ILP que é amplamente conhecida.

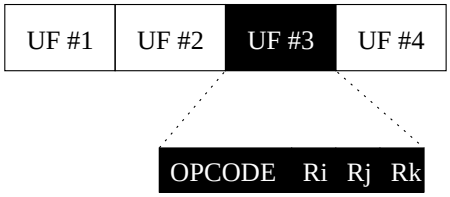

Figure 2: Esquema de uma Instrução VLIW

CARACTERÍSTICAS	VLIW
Tamanho da Instrução	Tamanho único
Formato da Instrução	Regular, posição consistente dos campos
Semântica da Instrução	Várias operações simples e independentes
Registradores	Vários, propósito geral
Acesso à Memória	Arquitetura do tipo *load-store*
Foco do Projeto de Hardware	Simples, explora múltiplas unidades funcionais, lógica de despacho de baixa complexidade

Table 1: Características da Arquitetura VLIW

O objetivo deste trabalho é introduzir os conceitos básicos das arquiteturas VLIW juntamente com as principais técnicas de compilação para a exploração de ILP. Lista-se também as vantagens e desvantagens desta arquitetura e a sua tendência futura. Este trabalho está organizado da seguinte forma. Na seção 2 é apresentada as características fundamentais das máquinas VLIW. Na seção seguinte é introduzido várias técnicas para a exploração estática de ILP. Na seção 4 são listadas as principais vantagens e desvantagens dos processadores VLIW. Na seção 5 é apresentado a perpectiva futura para a arquitetura. E por fim, a seção 6 encerra o trabalho com uma breve conclusão.

2. ARQUITETURA VLIW

Uma máquina VLIW genérica [4] pode ser esquematizada como uma máquina do tipo *load-store* [7] com múltiplas unidades funcionais que são conectadas a um banco central de registradores de propósito geral (vide Figura 1). Todas as operações a serem executadas simultaneamente nas diversas unidades funcionais são encapsuladas em uma única instrução, que pode ser imaginada como um conjunto de instruções RISC (vide Figura 2). Este escalonamento de várias operações em uma única instrução é feita estaticamente pelo compilador o que simplifica o hardware, que não precisa escalonar as instruções para explorar ILP como nos casos das máquinas superescalares.

As principais características da arquitetura VLIW estão sintetizadas na Tabela 1. Observando-a, podemos notar que a arquitetura VLIW tende a ser parecida com a arquitetura RISC [7], diferindo apenas pelo fato das instruções conterem múltiplas operações pré-escalonadas e o hardware explorar várias unidades funcionais utilizando uma lógica simples.

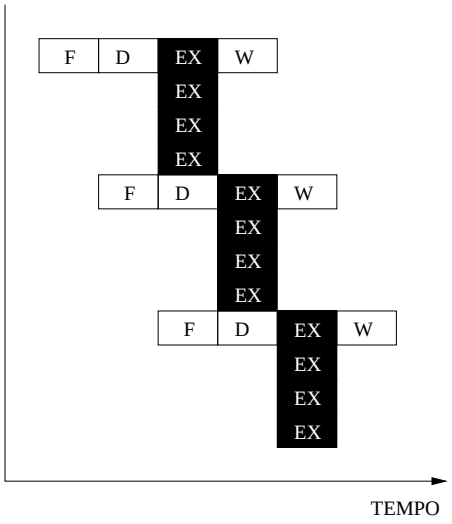

Figure 3: Modelo de Pipeline - VLIW

Um exemplo de pipeline com quatro unidades funcionais e quatro estágios: *fetch* (F), decodificação (D), execução (EX) e escrita (W), para uma máquina VLIW genérica é apresentado na Figura 3. Neste pipeline, faz-se o *fetch* de uma única instrução por vez, porém esta instrução longa especifica várias operações a serem executadas simultaneamente. Como a instrução VLIW é regular e contém os *opcodes* de todas as operações encapsuladas, o processo de decodificação fica simples. No estágio de execução, as operações são despachadas para as unidades funcionais, não havendo nenhum hardware extra para a detecção e tratamento de dependências, sendo responsabilidade do processador gerenciar tudo isso.

A eficiência deste tipo de pipeline fica limitada a capacidade do compilador em encapsular as operações em instruções longas, pois uma baixa densidade de operações por instrução irá resultar em unidades funcionais ociosas e conseqüentemente num baixo número de operações completadas por vez. No entanto, apesar das diversas técnicas aplicadas em compiladores, encapsular às instruções com operações que preencham todas as unidades funcionais é uma tarefa difícil e às vezes impossível. Por exemplo, numa máquina VLIW com unidades funcionais de inteiros e ponto flutuantes, não é possível gerar trabalho para as unidades de ponto flutuantes durante a execução de uma aplicação de inteiros.

Outros problemas inerentes a essas instruções esparsas são: o desperdício de espaço na memória e na cache de instruções, além de banda de transmissão. Uma das soluções para esse problema é o uso de técnicas de compressão de dados. No entanto, o fato de haver ou não compressão tem impacto direto na complexidade da unidade de *fetch*. Segundo [4] pode-se classificar a arquitetura VLIW em duas categorias de acordo com o tipo de codificação da instrução, conforme descrito a seguir:

- *Uncompressed encoding* - formato de instrução com número fixo de bits, os NOPs [2] são codificados ex-

[2]Cosidera-se como NOP qualquer operação que não faz nada

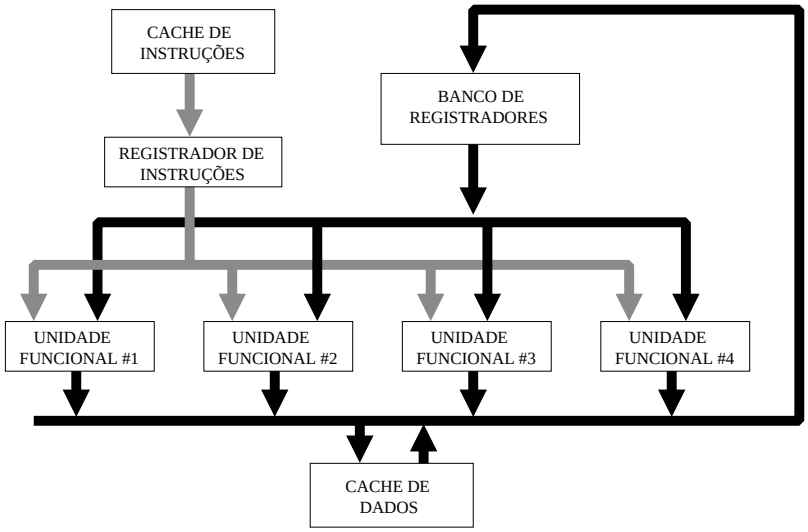

Figure 1: Esquema de uma Arquitetura VLIW Genérica

plicitamente quando uma operação não pode ser escalonada para a instrução.

- *Compressed encoding* - formato de instrução com um número variável de bits, os NOPs não são codificados.

3. ESCALONAMENTO ESTÁTICO

Embora as arquiteturas VLIW reduzam a complexidade do seu hardware com relação a outras arquiteturas como as superescalares, elaa necessitam de um compilador muito mais complexo, que seja capaz de explorar um alto grau de paralelismo em nível de instrução a fim de garantir um bom desempenho do processador. Como o compilador é responsável pelo escalonamento das operações a serem executadas em paralelo (encapsulamento das operações nas instruções longas) e pelo tratamento das dependências, a arquitetura é exposta ao compilador, permitindo-o um alto grau de liberdade para explorar os recursos da arquitetura.

Para garantir um bom desempenho do processador VLIW, várias técnicas de otimização e escalonamento de código são utilizadas. Para o escalonamento de operações tem-se basicamente dois métodos: *basic block scheduling* e *extended basic block scheduling* [1]. O primeiro método explorar ILP dentro de um bloco básico. Esta técnica acaba sendo muito limitada uma vez que um bloco básico tem em média 4–5 operações interdependentes. O segundo método estende os blocos básicos de forma a aumentar o número de operações interdependentes. A seguir são descritas algumas destas técnicas.

3.1 Loop Unrolling

O *loop unrolling* [3, 8, 5] é uma técnica para se aumentar o tamanho do corpo do loop através da diminuição do seu número de iterações, permitindo desta forma uma exploração de paralelismo entre instruções que encontram-se em diferentes iterações. Esta técnica basicamente reorganiza o corpo do loop, seguindo os seguintes passos:

de útil à execução do programa.

1. Replica-se o corpo do loop n vezes.
2. Exceto para o último bloco, remove-se as instruções de incremento/decremento do contador do loop.
3. Multiplica-se o incremento/decremento por n e ajusta os *offsets*.
4. Exceto para o último bloco, remove-se as instruções de teste/desvio do loop.

Na Figura 4(b) é apresentado o código para o desenrolamento de uma iteração do loop da Figura 4(a). Como pode ser observado, após ser aplicado o *loop unrolling* o corpo do loop passa a ser composto por instruções de duas iterações adjacentes, além disso o número de iterações foi reduzido pela metade. Conforme mostrado em [9], esta técnica provê os seguintes benefícios:

1. Elimina um número significante de instruções de desvios.
2. Reduz o número total de instruções despachadas, pois elimina-se instruções de incremento/decremento que são desnecessárias entre blocos adjacentes.
3. Permite um melhor escalonamento das instruções, pois aumenta o tamanho do bloco básico.

3.2 Sotware Pipelining

O *software pipelining* [3, 5] é uma técnica que reorganiza o loop de forma que cada iteração no loop reestruturado é formado por instruções escolhidas de diferentes interações do loop original (vide Figura 4(c)). O objetivo desta técnica é eliminar as dependências que ocorrem entre as instruções de uma iteração do loop. O *software pipelining* também é conhecido como *symbolic loop unrolling*, devido a sua capacidade de intercalar instruções de diferentes iterações, sem explicitamente desenrolar o loop.

for($i = 1, i \leq 1000, i = i + 1$){
$I_1(i)$
$I_2(i)$
$I_3(i)$
}

(a) Loop Original

for($i = 1, i \leq 999, i = i + 1$){
$I_1(i)$
$I_2(i)$
$I_3(i)$
$I_1(i+1)$
$I_2(i+1)$
$I_3(i+1)$
}

(b) ***Loop Unrolling***

$I_1(1)$
$I_2(1)$
$I_1(2)$
for($i = 1, i \leq 1000, i = i + 1$){
$I_1(i)$
$I_2(i-1)$
$I_3(i-2)$
}
$I_3(999)$
$I_2(1000)$
$I_3(1000)$

(c)*Software Pipelining*

Figure 4: *Loop Unrolling* e *Sotware Pipelining*

No exemplo apresentado na Figura 4(c), observa-se que a aplicação do *software pipelining* resultou na intercalação de instruções de três iterações diferentes do loop orignal (vide Figura 4(a)). O código adicionado no início e no fim do loop, faz-se necessário para manter a semântica do loop original.

Como as técnicas de *loop unrolling* e *software pipelining* eliminam diferentes tipos de *overhead*. Sendo que a primeira elimina o overhead das instruções de controle do loop (instruções de desvio e incremento/decremento do contador do loop) e a segunda reduz o número de vezes em que o loop não está com o overlap máximo [3]. Elas podem ser combinadas para produzir um melhor escalonamento do código.

3.3 Trace Scheduling

Apesar do *trace scheduling* [3] ter sido inicialmente proposto como um algoritmo para compactação de microcódigo [2], esta técnica se tornou um dos principais métodos para escalonamento de código para máquinas VLIW. O que motiva a sua utilização é a sua capacidade de otimização além das fronteiras de um bloco básico.

A idéia desta técnica é transformar o caminho mais comum de execução em um conjunto de instruções sequenciais, livres de instruções de desvios. O algoritmo de *trace scheduling* funciona da seguinte maneira. Inicialmente, seleciona-se um *trace* [3] com alta probabilidade de ser executado. Em seguida, compacta-se o *trace*, ou seja, tenta-se encapsular as suas operações no menor número de palavras longas (instruções VLIW) possíveis. Durante esta fase, pode ser necessário a geração de código de compensação, uma vez que outros *traces* de execução podem entrar ou sair do meio do *trace* que está sendo compactado.

3.4 Superblock Scheduling

O *superblock scheduling* [3, 1] é uma técnica similar ao *trace scheduling*, exceto pelo fato de não permitir pontos de entrada no meio do *trace*. A compactação é feita em cima de um *trace* com apenas uma entrada e uma ou mais saídas. Esta estrutura simplifica o processo de compactação das operações em instruções longas, pois a ausência de pontos de entrada elimina a movimentação de código para *traces* entrantes.

Para se formar os superblocos, inicialmente encontra-se os *traces* através de informações de *profiling*. Em seguida, elimina-se os pontos de entrada que estão no meio do *trace* através da utilização de um método denominado de *tail duplicatication*, que move os pontos de entrada para blocos básicos duplicados.

Uma desvantagem do *superblock scheduling* e do *trace scheduling* é que ambos os esquemas de escalonamento consideram um único caminho de execução. Logo a escolha de um caminho errado de execução baseado em informações de *profiling* levará a um desperdício de ciclos de execução do processador.

3.5 Hyperblock Scheduling

No *Hyperblock Scheduling* [1], múltiplos caminhos de execução são escalonados em uma única unidade. Esta técnica

[3]Um *trace* é uma seqüência linear de blocos básicos

utiliza predicação para formar escopos de escalonamento. A predicação envolve a execução de instruções baseadas num predicado (valor booleano de um operador). Um *hyperblock* pode conter vários caminhos de execução que são combinados por *if-conversion* e *tail duplication*. Blocos básico que contenham chamadas de procedimento e acessos a memória não resolvidos, não são incluídos no *hyperblock*.

O *hyperblock* corresponde a uma estrutura com uma única entrada e múltiplas saídas laterais. O método *if-conversion* transforma dependência de controle em dependência de dado e consequentemente novas otimizações podem ser aplicadas.

4. VANTAGENS E DESVANTAGENS

Assim como as arquiteturas escalares e superescalares, as arquiteturas VLIW também tem os seus prós e contras. Algumas das vantagens dos VLIW são:

- Hardware de controle simples, uma vez que o escalonamento das operações e a verificação das dependências é feita pelo compilador.
- O fato do hardware ser simples implica num menor consumo de energia.
- Capacidade de despacho de múltiplas operações através do uso de instruções longas.
- O fato da arquitetura ser exposta ao compilador, permite a aplicação de uma vasta gama de otimizações.
- O acesso ao código fonte permite ao compilador analisar janelas maiores de código, aumentando o nível de otimização sem aumentar a complexidade do hardware.

Alguns dos pontos negativos que podem desistimular a produção comercial destes processadores são:

- A construção de um compilador VLIW não é uma tarefa nada trivial, uma vez que o desempenho desta arquitetura depende da capacidade do compilador em explorar o paralelismo entre as instruções.
- Não tem vantagem de rodar o mesmo código que outras plataformas, como, por exemplo, os superescalares que rodam código de escalares.
- Existe uma dependência evidente da arquitetura, sendo díficil existir compatibilidade entre diferentes máquinas VLIW.
- Se a taxa de operações por intrução for baixa, ocorrerá um mal uso da memória (desperdício de banda de transmissão e espaço na cache), uma vez que as instruções longas serão completadas com Nops.

5. PERSPECTIVAS FUTURAS

De maneira geral, pode-se dizer que o succeso comercial das arquiteturas VLIW ainda é moderado, como é exemplificado pelos processadores Philips Trimedia, C6X da Texas Instruments, Intel Intanium e Transmeta Crusoe [6]. No entanto o papel desta arquitetura vem mudando ao longo dos anos, deixando de ser usada para o desenvolvimento de processadores destinados a supercomputadores científicos e passando a ser utilizada em processadores destinados ao processamento digital de sinais, imagem, multimídia e dispositivos móveis, onde a relação desempenho/energia é importante.

A crescente demanda por aplicações multimídias provavelmente irá continuar incentivando o desenvolvimento da tecnologia VLIW, pois para estas aplicações o comportamento das instruções de desvios são bem regulares, ou seja, os *traces* de execução são bem definidos, além disso, a quantidade de instruções paralelizáveis é muito grande, o que faz destas aplicações ideais ao estilo VLIW de execução. Todavia, a curto prazo, os processadores superescalares provavelmente irão continuar dominando o mercado de processadores de propósito geral.

Atualmente, a pesquisa relacionada a tecnologia VLIW concentra-se nas seguintes áreas:

- Compiladores - desenvolvimento de novas técnicas de otimização e escalonamento que aumentem a capacidade de exploração de ILP.
- Arquitetura - desenvolvimento de uma arquitetura mais portável.

6. CONCLUSÕES

Este trabalho apresentou os principais conceitos relacionados as arquiteturas VLIW, focando principalmente na questão de exploração de paralelismo em nível de instrução. Apesar do desenvolvimento desta tecnologia ter se sucedido de maneira lenta ao longo da última década, nos últimos anos o crescente mercado de DSPs e sistemas dedicados, vem alavancando novamente um novo interesse por essa arquitetura.

7. REFERÊNCIAS

[1] M. Agarwal. Speculative trace scheduling in vliw processors, 2002.

[2] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Trans. Computers*, 30(7):478–490, 1981.

[3] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[4] S. A. Ito. Uma alternativa para a exploração de paralelismo a nível de instrução. Relatório técnico, Universidade Federal do Rio Grande do Sul, 2000.

[5] N. Kondo, A. Koseki, H. Komatsu, and Y. Fukazawa. A method for applying loop unrolling and software pipelining to instruction-level parallel architectures. *Systems and Computers in Japan*, 29(9):62–73, 1998.

[6] B. Mathew. *The Computer Engineering Handbook*, chapter Very Large Instruction Word Architectures. CRC Press, Dec. 2001.

[7] P. W. Papers. An introduction to very-long instruction word (vliw) computer architecture. Technical Report 9397-750-01759, Philips Groups, 2002.

[8] S. Sair and D. Kaeli. A study of loop unrolling for vliw based dsp processor, 1998.

[9] S. Weiss and J. E. Smith. A study of scalar compilation techniques for pipelined supercomputers. In *ASPLOS-II: Proceedings of the second international conference on Architectual support for programming languages and operating systems*, pages 105–109, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.