Introdução à Arquitetura VLIW: Fundamentos e Perspectivas*

Alternativas para o Paralelismo a Nível de Instrução

André Augusto Ciré (015463)
Instituto de Computação - UNICAMP
Caixa Postal 6176
Campinas, Brasil - 13084-971
andre.cire@ic.unicamp.br

ABSTRACT

Este artigo apresenta os principais conceitos envolvendo a arquitetura *Very Long Instruction Word*, ou *VLIW*, como uma abordagem alternativa às máquinas superescalares para paralelismo a nível de instrução. Será feita uma análise histórica e levantadas algumas técnicas significativas de escalonamento de instruções, além da apresentação de exemplos de máquinas que implementam conceitos de VLIW como um pilar fundamental de suas arquiteturas. O artigo também lista vantagens, desvantagens e pesquisas em andamento, finalizando com uma conclusão em termos de comentários e perspectivas futuras da filosofia VLIW.

Keywords

Abordagem VLIW, Sistemas Superescalares, Paralelismo a Nível de Instrução

1. INTRODUÇÃO

Arquitetos e projetistas de computadores se colocam em uma constante busca pelo aumento do desempenho de máquinas monoprocessadas, motivados pelas crescentes exigências das aplicações e, principalmente, pela competição acirrada inerente ao mercado. Tais fatores justificam os volumosos investimentos para a pesquisa de novas arquiteturas e tecnologias de silício, tanto em universidades como em empresas de grande porte.

Neste contexto, a disseminação das arquiteturas RISC (*Reduced Instruction Set Computer*), ocorrida na década de 1980, foi de essencial importância para a elevação do desempenho dos microprocessadores ao definir instruções simples e de tamanho fixo [16]. Além de necessitar de menos transistores por processador, sua simplicidade favorece muito mais otimizações no projeto de hardware quando comparado ao seu modelo anterior, CISC (*Complex Instruction Set Computer*).

Contudo, a evolução da arquitetura RISC acabou limitada pelo próprio paradigma das máquinas von Neumann [17], no qual os programas são implementados com a suposição tácita de que as instruções devem ser executadas seqüencialmente, na mesma ordem em que aparecem no código. Prosseguiu-se, então, na direção de explorar ao máximo o palarelismo inerente entre as próprias instruções, ou ILP (*Instruction Level Parallelism*), executando múltiplas de uma vez ou simplesmente reordenando-as para a obtenção de um maior desempenho.

Dentre as formas de explorar o paralelismo de baixa granularidade próprio do ILP, três se destacam [16]. A primeira, *pipelining*, já está presente há décadas no mercado e hoje é universalmente utilizada nos processadores de alto desempenho. Entretanto, poucas otimizações ainda são realmente efetivas em pipelines simples, caracterizados por possuir várias instruções em diferentes estágios de execução concorrentemente, mas apenas uma única instrução iniciada por vez.

Como forma de superar este gargalo prático, surge a conclamada panacéia dos processadores modernos: arquiteturas *superescalares*, no qual mais de uma instrução é lida e decodificada em um mesmo ciclo de *clock* [17]. Apesar de considerado como a evolução das arquiteturas RISC e de realmente prover ganhos de desempenho consideráveis, processadores superescalares impõe uma complexidade crescente no projeto do hardware [15], e obter altos IPCs (número de instruções por ciclo) sem sacrificar o desempenho torna-se cada vez mais difícil e custoso.

Temos então as arquiteturas *VLIW* (*Very Long Instruction Word*), uma das principais abordagens alternativas que objetiva explorar o ILP sem transferir demasiada carga à complexidade do processador. Para tanto, arquiteturas VLIW clássicas têm sua estrutura de controle extremamente simplificada, não possuindo suporte algum para o escalonamento dinâmico de instruções e tratamento de dependências [13].

*Artigo produzido como trabalho 2 da disciplina *MO401 - Arquitetura de Computadores I*, ministrada pelo Prof. Rodolfo Jardim Azevedo, 2º semestre de 2005

Portanto, ao invés de explorar recursos em hardware (por exemplo, o algoritmo de Tomasulo [18]), a filosofia VLIW propõe a utilização de técnicas de escalonamento estático efetuadas exclusivamente pelo compilador, onde múltiplas operações independentes são compactadas em apenas uma palavra longa, conseguindo-se o despacho simultâneo de mais de uma instrução [9].

O artigo, desta forma, objetiva levantar os principais aspectos, vantagens e desvantagens das abordagens VLIW, e está organizado como se segue. Na seção 2, será apresentada uma perspectiva história da tecnologia, além de alguns conceitos básicos envolvidos. Já na seção 3 serão explicados os fundamentos conceituais da arquitetura VLIW, e na seção 4 será feito um detalhamento maior do escalonamento de instruções, fundamental para um compilador VLIW. Em seguida, a seção 5 mostra exemplos de máquinas VLIW já produzidas, e a seção 6 relaciona os atuais projetos de pesquisa e possíveis passos para o futuro. Por fim, a seção 7 faz uma conclusão do trabalho, e são mostradas as referências.

2. PERSPECTIVA HISTÓRICA

As idéias iniciais da arquitetura VLIW surgiram nos estudos de computação paralela feitas por Alan Turing, em 1946, e no trabalho de microprogramação de Maurice Wilkes, 1951 [20]. Uma *CPU microprogramada* está relacionada a *macroinstruções*, as quais representam as instruções do programa em si. Cada uma dessas macroinstruções, por sua vez, é composta por uma seqüência de *microinstruções*, cada qual associada diretamente a alguma unidade funcional do processador (por exemplo, a ALU).

Esta organização é denominada *microprogramação horizontal* [3] e é a base das arquiteturas VLIW. Ela se contrapõe à microprogramação *vertical*, onde os campos das instruções devem ser decodificados para produzir os respectivos sinais de controle. Assim, a microprogramação horizontal permite que todas as possíveis combinações de sinais (e, portanto, *operações*) sejam expressas diretamente como instruções, ao invés da lógica de sinais concentrada em apenas uma única unidade de decodificação. Uma conseqüência natural da microprogramação horizontal, desta forma, é a necessidade de maiores tamanhos de palavra para cada instrução.

O problema de gerar palavras longas a partir de seqüências de instruções curtas foi tratado com sucesso por Joseph Fisher, em 1979, durante a criação de um microcódigo horizontal para o emulador CDC-6600 [5]. A técnica desenvolvida foi a de *Escalonamento por Traces* (veja seção 4.1), posteriormente utilizada para a compilação de programas escritos em linguagens convencionais para máquinas VLIW.

Motivados com os resultados, Fisher e colegas da Universidade de Yale fundaram em 1984 a *Multiflow*, para a produção de supercomputadores VLIW. No mesmo ano, Bob Rau fundara a *Cydrome*, para iniciar a construção de minisupercomputadores utilizando a filosofia VLIW e com suporte arquitetural para loops escalonados pela técnica de *software pipeline*

Apesar de ambos terem lançados produtos finais (como o *Cydra 5*, da Cydrome, com palavras de instruções de 256 bits e despacho de 7 operações por ciclo), o mercado não estava pronto e a tecnologia ainda era relativamente prematura, fornecendo poucas alternativas viáveis dentre as soluções da época como conseqüência do alto custo de memória. Devido às dificuldades financeiras, Cydrome encerra suas atividades em 1988 e a Multiflow, em 1990.

Com o desaparecimento de ambas as empresas do mercado, o desenvolvimento da tecnologia VLIW seguiu-se lentamente, até que na década de 1990 estudos identificaram VLIW como a arquitetura ideal para a computação de algoritmos complexos e altamente repetitivos [11, 6]. Isto levou ao surgimento de uma nova série de chips para processamento de mídias, liderados pelas empresas *Chromatics's MPact* e *Philips's TriMedia*, e finalmente ao primeiro sucesso comercial de uma implementação VLIW: a série *Texas Instruments C6X*.

Este sucesso pavimentou o caminho para projetos de microprocessadores VLIW de larga escala. Em 2000, a *Transmeta* lançou a família de processadores VLIW *Crusoe*, com baixo consumo de energia para sistemas dedicados. A Intel seguiu a mesma filosofia com sua especificação de arquiteturas IA-64 e com a linha *Itanium* de chips.

Atualmente, centros de pesquisa, como o da IBM, têm dado continuidade às pesquisas em tecnologias VLIW, o que demonstra o interesse sobre sua filosofia e os benefícios que esse tipo de arquitetura pode trazer [9]. Fisher e Rau também dão continuidade às suas pesquisas, liderando a área de compiladores na *Hewlett-Packard*.

3. FUNDAMENTOS DO VLIW

Há basicamente três fatores principais que determinam o tempo de execução de um programa [9]: o *número de instruções da aplicação*, o *o número médio de ciclos necessários para executar uma instrução* e o *tempo de um ciclo do processador*.

Cada uma das diferentes técnicas de ILP explora diferentemente esses três fatores. Nos casos citados, processadores superescalares buscam reduzir o *número de ciclos* para executar uma instrução (fator 2) e a arquitetura VLIW, em contrapartida, foca reduzir o *número de instruções* da aplicação (fator 1) [19].

Como é típico no microcódigo horizontal, as palavras de instruções de uma arquitetura VLIW clássica possuem centenas de bits, contendo os *opcodes* que especificam as operações a serem executadas nas diferentes unidades funcionais, conforme mostrado na figura 1. Estas unidades, por razões de eficiência e simplicidade, compartilham um grande banco de registradores comum, por sua vez mostrado na figura 2 (ambas as figuras extraídas de [9]).

Desta forma, as operações paralelizadas estão encapsuladas em uma única instrução, bastando o processador VLIW acessar um único endereço (a da instrução) para acessá-las. Devido à essas características, em [14] a arquitetura VLIW também recebe a denominação de SIMOMD (*Single Instruction, Multiple Operation, Multiple Data*) ou, simplificadamente, *MultiOp*.

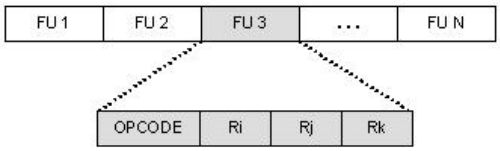

Figure 1: Palavra VLIW, com diferentes *opcodes* para cada unidade funcional

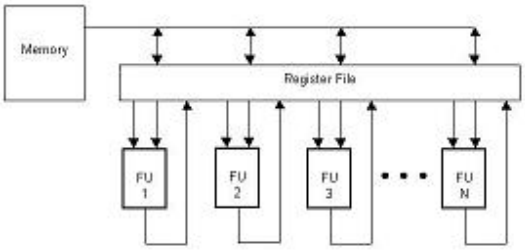

Figure 2: Arquitetura VLIW teórica, onde diferentes unidades funcionais compartilham um grande registrador.

Conseqüentemente, o processador deve receber as operações já devidamente agendados nas instruções do programa, o que implica na transferência da complexidade do escalonamento do hardware para os compiladores, um compromisso fundamental da arquitetura VLIW. Este agendamento é, portanto, realizado **estaticamente**, de forma a não gerar conflitos estruturais e de manter a semântica da seqüencialidade das instruções [17].

O ganho de desempenho de programas VLIW depende então diretamente de compiladores capazes de gerar bons agendamentos das operações. Este pré-agendamento traz benefícios e desvantagens, descritos a seguir [9, 15]:

- o compilador tem conhecimento e acesso completo da arquitetura do processador, tal como a latência de cada unidade funcional. Isto dá margem a uma vasta gama de otimizações, além de dispensar mecanismos de sincronização em tempo de execução para impedir inconsistência de dados;
- apesar de necessitar de um número elevado de portas para acesso à *cache* de dados pelas unidades funcionais, o hardware de controle é bem mais simples de um modo geral, reduzindo teoricamente o tempo de um ciclo de *clock*;
- como a análise de paralelismo é estática, ela deve sempre considerar o pior caso da execução do código. Isto implica que informações obtidas apenas em tempo de execução não podem ser aproveitados para otimização de *branches*, por exemplo. Nos trabalhos de [9, 7], propõe-se utilizar informações de *profiling* como uma forma de sanar este problema;
- a *densidade de compactação* (taxa de operações por instrução) depende do paralelismo inerente das instruções. Se tal densidade for baixa e o escalonador não for suficientemente robusto para identificá-las, haverá uma má utilização da memória devido ao longo tamanho das instruções.
- exige-se uma elevada largura de banda de memória que, se não tratada corretamente, pode levar à uma queda no desempenho das máquinas. É importante que compiladores tenham algum tipo de conhecimento prévio em relação à organização do banco de memória, a fim de otimizar o acesso utilizando, por exemplo, técnicas de entrelaçamento de memória.

Além das vantagens e desvantagens acima, um dos maiores obstáculos para a evolução da tecnologia é a *incompatibilidade binária* entre os códigos-objeto de diferentes gerações de processadores VLIW, decorrente das especificidades arquiteturais que os compiladores devem considerar para realizar os escalonamentos e otimizações necessárias. Diversas soluções já foram propostas para este problema, como descrito no ótimo artigo de [2], incluindo reescalonamento dinâmico das operações ou apoio do sistema operacional na compilação.

4. ESCALONAMENTO ESTÁTICO

A chave para se obter bons desempenhos em arquiteturas VLIW, como visto nas seções anteriores, está em compiladores capazes de fazer com que as unidades funcionais permaneçam o mínimo possível de tempo ociosas. Assim, o *escalonamento* (ou *agendamento*) das instruções caracteriza-se como um processo fundamental em compiladores VLIW, e cujas características de alguns métodos veremos a seguir.

Há basicamente duas classes principais de escalonamento estático [1]: *Escalonamento por Blocos-Básicos* e *Escalonamento por Blocos-Básicos Estendido*. A primeira classe compreende os métodos mais tradicionais de agendamento, com a subdivisão do programa em *blocos básicos* e verificação de conflitos entre tais blocos. São muito pouco eficazes no caso do VLIW, já que habitualmente 14 a 16% dos códigos representam desvios condicionais ([12]).

Já a segunda classe compreende os métodos de escalonamento mais significativos, e uma breve descrição de alguns bastante utilizados será feita nas subseções seguintes.

4.1 Escalonamento por Traces

O algoritmo de *Escalonamento por Traces* foi originalmente proposto por [5] para a compactação (i.e., agrupamento das operações em palavras longas) de microprogramas em microinstruções horizontais, e se tornou um dos principais métodos de agendamento de código para máquinas VLIW [1].

Neste tipo de escalonamento, uma determinada função é dividida em um conjunto de *traces*, que representam os caminhos com maior probabilidade de execução. Em um trace, podem existir branches condicionais que levam a outros traces (*portas de saída*) ou transições de outros *traces* para ele (*portas de entrada*). As instruções são então escalonadas a partir dos traces e ignorando os fluxos de controle, sendo necessário armazenar as informações de todas as portas de entrada e saída de cada trace para manter a semântica das aplicações.

A utilização do *Escalonamento por Traces* possibilita a exploração de paralelismo muito além daquela contida em blocos básicos, já que não há restrições associadas a blocos na escolha dos traces, o que faz com que o método seja conveniente para processadores com um grande número de despachos por ciclo.

4.2 Escalonamento por Superblocos

O algoritmo de *Escalonamento por Superblocos* [7] é semelhante ao método da seção anterior, diferenciando-se por evitar a necessidade de armazenar todos os fluxos de controle ao definir traces que não possuem pontos de entrada: são os *superblocos*. O controle entra apenas a partir do topo do trace, e pode sair por um ou mais pontos de saída.

Os superblocos são formados em basicamente dois passos: inicialmente, identificam-se os traces a partir de informações de *profiling*. Em seguida, um método denominado *tail duplication* (duplicação de cauda) é utilizado a fim de eliminar todos os pontos de entrada do trace, movendo-os para blocos básicos duplicados.

Uma desvantagem do *Escalonamento por Traces* e *Superblocos* é que ambos os algoritmos consideram apenas um único possível caminho de execução por vez. Portanto, se as informações do profiling levarem à escolha errada deste caminho, haverá um grande desperdício de ciclos do processador. Para contornar este problema, surgem os algoritmos de *Escalonamento por Hiperblocos* e *Escalonamento por Árvore de Decisão*, descritos a seguir,

4.3 Escalonamento por Hiperblocos

Neste algoritmo, múltiplos caminhos de execução são agendados em uma unidade simples, utilizando-se *predicação* pra definir os escopos de cada agendamento. Uma *predicação* equivale à execução condicional de instruções baseadas no valor de operandos booleanos, os *predicados*.

Um *hiperbloco* pode conter diversos caminhos combinados por um predicado do tipo *if* e por duplicação de caudas, mas nunca conter blocos básicos com chamadas de procedimentos ou acessos à memória não resolvidos. Tal como no superbloco, há apenas uma entrada com múltiplas saídas.

Um ponto importante a ser notado é que o processo de conversão de predicados *if* transforma as dependências de controle em dependência de dados e, portanto, mais otimizações podem ser aplicadas no hiperbloco quando comparado ao processo de Escalonamento por Traces regular.

4.4 Escalonamento por Árvore de Decisão

Este método é similar ao Escalonamento por Hiperblocos, com o fluxo de controle dos blocos básicos representados como uma *árvore* [21]. Os escopos de agendamento também compreendem uma simples entrada e múltiplas saídas, e cada *folha* da árvore de decisão ou termina em uma chamada de procedimento, ou pula para uma árvore diferente.

Não há pontos de saída do interior de blocos básicos de um árvore de decisão, o ponto de entrada é denominado de *raiz* da árvore. Predicação também pode ser empregada neste algoritmo, a fim de permitir mais otimizações e explorar mais o ILP, na forma descrita a seguir.

Todas as operações de controle em todos os escopos de agendamento são adiados ou se tornam predicados. Em caso de branches adiados, o escalonador busca operações apropriadas para preencher os "slots vazio" e, caso não for possível encontrar tais operações, simplesmente insere *nops*. Já no caso de se tornarem predicados, as dependências de controle se tornam dependências de dados (tal como no escalonamento por hiperblocos), melhor otimizáveis e, portanto, preferíveis.

5. MÁQUINAS VLIW

O objetivo desta seção é traçar algumas características básicas das máquinas que usam tecnologia VLIW, a fim de identificarmos suas características principais e aspectos comuns. Contudo, apenas duas empresas produziram comercialmente máquinas puramente VLIW: a *Multiflow Computer, Inc.* e a *Cydrome, Inc.*. Pelo fato de não atingirem sucesso de mercado, como visto na seção 2, há pouquíssimos dados sobre a eficiência real dessas máquinas, que ficaram restritas a poucas unidades entregues. Os dados das duas respectivas subseções foram retiradas de [9].

Também serão apresentados alguns aspectos arquiteturais de duas tecnologias bastante relevantes para a abordagem VLIW, citadas na seção 2: o processador *Itanium*, da Intel, e o *Crosue*, da Transmeta. As informações do Intel Itanium foram obtidas de [8] e, do Transmeta Crosue, de [10].

5.1 Multiflow TRACE

Os principais objetivos perseguidos no projeto do processador Multiflow TRACE foram:

- projeto modular, com um número de unidades funcionais expansível;
- uso de grande quantidade de componentes eletrônicos padrão e de baixo custo;
- uso de memórias DRAM convencionais para obter uma memória principal de alta capacidade à um baixo custo;
- allcançar alta performance para computações de ponto flutuante de 64 bits;
- boa adequação ao ambiente multi-usuário.

O núcleo do processador é dividido em uma unidade de inteiros e uma de ponto flutuante. Existem bancos de registradores separados para cada uma das unidades, pois dificilmente é realizado uma operação de ponto flutuante sobre dados inteiros (e vice-versa), enquanto a idéia é realizar operações sobre dados inteiros e cálculos de endereços em paralelo com operações de ponto flutuante.

O conjunto de instruções RISC-like sobre inteiros contém opcodes dedicados à operações aritméticas, lógicas e de comparação. Existem primitivas de alto desempenho para multiplicação de 32 bits e operações de shift, merge e extract para a manipulação de campos de bits e bytes e o acesso à memória é realizado por operações de load e store em pipelines. Já as operações de adição de ponto flutuante têm sobre as unidades funcionais dessa máquina uma latência de

6 ciclos, em contraste com a latência simples das unidades inteiras.

O processador Trace mais poderoso poderia incorporar palavras de instrução com até 1024 bits, que poderiam iniciar 28 operações por instrução, com um desempenho de 215 "VLIW MIPS" e 60 MFLOPS. A política do barramento é arbitrada pelo compilador, obtendo barramentos simples, rápidos e de baixo custo.

A alta largura de banda da memória dessa máquina era alcançada utilizando entrelaçamento de memória, 0Kb de cache de dados e nenhum hardware para escalonar referência à memória. Os projetistas argumentaram na época que, para grandes aplicações científicas que possuam algum padrão de referência à memória, o entrelaçamento consegue sustentar um desempenho melhor do que o uso de caches de dados.

5.2 Cydra 5

O Cydra 5 era um sistema multiprocessador heterogêneo que utilizava a arquitetura *directed-dataflow*, destinado ao uso de pequenos grupos de cientistas ou engenheiros e que pretendia ganhar o mercado da categoria de minisupercomputadores. Produzida pela Cydrome, Inc., essa máquina tinha como objetivos principais:

- alto desempenho principalmente em aplicações numéricas;
- manter a transparência ao usuário, para que não necessitasse fazer modificar seus programas;
- baixo custo.

Embora a arquitetura seja denominada *directed-dataflow*, diversos conceitos empregados para suportá-la são VLIW. Essa máquina tem capacidade de despacho de múltiplas operações por ciclo e palavras de instrução de 32 bytes. A parte destinada ao processamento numérico podia despachar até seis operações por ciclo mais uma adicional para controlar a unidade de instruções e outras operações diversas.

Entretanto, um recurso permitia a especificação de uma única instrução por vez, quando as seis operações da instrução são despachadas individualmente. Foi criado como uma forma de evitar desperdício na codificação de *nops* quando havia pouco paralelismo para ser explorado.

O desempenho dessa máquina ficava em torno de 15.4 e 5.8 MFlops para os benchmarks *Linpack* e *Livermore Fortran Kernel*, respectivamente. Era um dos mini-supercomputadores mais performáticos da época e ficava próximo de um terço do desempenho de um supercomputador Cray X-MP. Para bechmarks menos vetorizáveis, como o *ITPack*, o Cydra 5 chegava a atingir metade do desempenho do Cray X-MP.

5.3 Intel Itanium

O processador *Itanium* é a primeira implementação da arquitetura **IA-64** da Intel, com um estilo de VLIW denominado *EPIC* (*Explicit Parallel Instruction Computing*) criado em conjunto pela Intel e *Hewlett-Packard*. Possui *clock* de 800 Mhz, com um processo de fabricação de 0,18 micron e pipeline de 10 estágios.

Basicamente, é uma arquitetura VLIW de 64 bits, capaz de executar até 6 operações por ciclo por meio de 4 unidades inteiras, 4 multimídias, 2 para loads/stores, 2 de ponto flutuante com precisão estendida e, por fim, mais 2 de ponto flutuante com precisão simples. Cada palavra VLIW consiste de um ou mais pacotes de 128 bits, contendo 3 operações e um *template*, isto é, a codificação das combinações mais freqüentes de tipos de operação. Desta forma, duas instruções são lidas da memória a cada ciclo, totalizando as 6 operações citadas previamente.

Ele também comporta 128 registradores de uso geral de 64 bits, 128 registradores de ponto flutuante de 82 bits e mais 64 registradores de predicados, para a execução condicional de operações (as não confirmadas são sumariamente descartadas). O compilador, por sua vez, pode escalonar os loops a partir da técnica de *software pipeline* [9].

O IA-64 também suporta especulação de dados e controle por software. Ele é compatível com a família Pentium (a partir de uma unidade auxiliar para lidar com instruções IA-32) e, em decorrência do número significativo de recursos existentes para aumentar seu desempenho, ele é considerado como a arquitetura VLIW mais complexa já existente.

Portanto, o Itanium caracteriza-se, de certa forma, como um grande paradoxo desta tecnologia, já que a arquitetura VLIW tem justamente como objetivo a simplificação do hardware e transferência de toda a complexidade para os compiladores.

5.4 Transmeta Crusoe

A arquitetura *Crusoe* representa um ponto distinto na história dos processadores VLIW, pois foram projetados para aplicações móveis, possuindo baixo consumo de energia e capazes de emular a arquitetura de outros processadores, como o ISA do 80x86 e a máquina virtual Java. Diferentemente dos modelos anteriores de VLIW, portanto, o foco não era estritamente o ganho de desempenho.

Ele possui 64 registradores de uso geral, com duas unidades funcionais para inteiros, uma para ponto flutuante, uma unidade de load/store e uma unidade de desvio. Além disso, o Crusoe inclui uma cache de instruções L1 associativa de 64KB com associtividade 8 e de associatividade 16 para dados, além da cache L2 unificada de 512KB.

Outras características da especificação são:

- Pipeline para inteiros de 7 estágios e de 10 estágios para ponto flutuante.
- Controlador de memória DDR SDRAM com 100-133 MHZ, 2.5V
- Controlador de memória SDR SDRAM com 100-133 MHZ, 3,3V
- Controlador PCI (PCI 2.1 compatível) com 33 MHZ, 3.3V
- Consumo médio de 0.4-1.0 W executando a 367-800MHZ, 0.9-1.3V executando aplicações multimídia.

A palavra longa no Crusoe tem ou 64 ou 128 bits de largura e, no último caso, é denominada *molécula*, codificando 4 operações chamadas de *átomos*. O formato da molécula determina a rota das operações para as unidades funcionais. Ao contrário do IA-64, usa códigos de condição idênticos aos utilizados na arquitetura X86 para facilitar a simulação.

Todos os átomos dentro de uma molécula são executadas em paralelo, e o formato da molécula diretamente determina como átomos são roteados para as unidades funcionais; isso simplifica bastante o hardware do despacho e decodificação.

Para a emulação, há um tradutor binário auxiliar denominado *code morphing*, projetado para traduzir dinamicamente as instruções x86 em VLIW, resolvendo eficientemente o problema da compatibilidade binária. O *code morphing* é um programa que reside em um Flash ROM e é a primeira aplicação a iniciar quando o Crusoe é ligado.

Apenas o código nativo do *code morphing* necessita ser modificado em caso de atualização da arquitetura x86. E, para otimizar seu desempenho, duas atitudes são tomadas: o hardware e o software mantém em conjunto uma cache de código traduzido, e há um *profiling* das traduções para coletar freqüência de execução e histórico de desvio, como forma de auxiliar uma tradução mais eficiente do código pelo *code morphing*.

Como o processador Crusoe foi projetado especificamente para diminuir o consumo de energia, a utilização do *code morphing* no lugar da lógica de transistores auxilia a gerar bem menos calor que os processadores convencionais.

6. O FUTURO DE VLIW

As pesquisas na área de VLIW atualmente se concentram em três grandes áreas:

- **compiladores**, no desenvolvimento de novas técnicas de escalonamento de instrução;
- **arquitetura**, na busca de novos mecanismos de busca e decodificação das instruções complexas;
- **incompatibilidade de código-objeto**, na tentativa de tornar o código VLIW mais portável entre as diferentes gerações de tecnologia.

Dentre os centros de pesquisas mais atuantes, pode-se citar a IBM com o desenvolvimento de uma arquitetura VLIW com representação das instruções em forma de *árvores* [4]. Pesquisadores da IBM argumentam que o uso deste tipo de representação pode também resolver o problema de compatibilidade de código objeto, por meio do suporte apropriado em hardware e software, e a tradução dinâmica da representação em árvore para palavras longas de instrução.

Outra técnica proposta pelos pesquisadores da IBM [9] é o conceito de *DIF* (*Dynamic Instruction Formatting*) que utiliza o palavras longas de instrução escalonadas dinamicamente e armazenadas em uma cache (*DIF cache*) para execução paralela das operações. O objetivo dessa proposta é simplificar o hardware de máquinas superescalares através do uso de alguns princípios da filosofia VLIW [15].

Contudo, ainda há um mercado diferenciado e muito pouco afetado pela inércia do conjunto de instruções e da falta de compatibilidade do código-objeto [13]: o de *sistemas dedicados*, caracterizado por produtos com softwares pré-instalados de fábrica e com processadores cujo conjunto de instruções não é evidente ao usuário final. Assim, há uma maior liberdade para migrar softwares dedicados para novos processadores, contanto que forneçam um melhor desempenho absoluto a um baixo custo.

Assim, fabricantes de microprocessadores estão sendo atraídos pelos potenciais benefícios que a arquitetura VLIW pode trazer a sistemas dedicados, mais flexíveis a mudanças dos conjuntos de instruções. Um exemplo pode ser visto pelo próprio projeto Crosue, descrito na seção 5.4.

7. CONCLUSÃO

Neste artigo foi apresentado uma série de aspectos da abordagem VLIW, incluindo conceitos, perspectiva histórica, vantagens e desvantagens de sua utilização. Como é possível verificar, transferir a complexidade de escalonamento de instruções do hardware para os compiladores pode ser extremamente vantajoso, principalmente ao considerarmos um possível limite físico para a evolução das máquinas superescalares.

Contudo, essa transferência de complexidade acarreta a incompatibilidade do código-objeto entre diferentes gerações de arquiteturas VLIW, principal impecílio para seu amadurecimento comercial. Tecnologias atuais buscam contornar este problema a partir de técnicas de escalonamento dinâmico ou de unidades extras para conversão entre arquiteturas, mas ainda não há uma solução definitiva para este problema.

Por fim, ainda há muitas pesquisas na área de VLIW na tentativa de aumentar sua viabilidade comercial como um computador de alto desempenho, além de um possível interesse que as empresas de sistemas dedicados podem ter em relação à esse tipo de arquitetura, por serem mais flexíveis no conjunto de instruções de seus processadores.

8. REFERENCES

[1] P. M. Agarwal and S. K. Nandy. On the benefits of speculative trace scheduling in VLIW. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '02, June 24 - 27, 2002, Las Vegas, Nevada, USA, Volume 2*. CSREA Press, 2002.

[2] T. M. Conte and S. W. Sathaye. Optimization of VLIW compatibility systems employing dynamic rescheduling. *International Journal of Parallel Programming*, 25(2):83–112, 1997.

[3] S. Dasgupta. The design of some language constructs for horizontal microprogramming. In *ISCA '77: Proceedings of the 4th annual symposium on Computer architecture*, pages 10–16, New York, NY, USA, 1977. ACM Press.

[4] K. Ebcioglu, E. R. Altman, S. W. Sathaye, and M. Gschwind. Execution-based scheduling for VLIW architectures. In *European Conference on Parallel Processing*, pages 1269–1280, 1999.

[5] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers, IEEE*, C-30(7):478–490, 1981.

[6] F. Gasperoni. *Scheduling for Horizontal Systems: The VLIW Paradigm in Persepctive*. PhD thesis, May 1991.

[7] W.-M. W. Hwu, S. A. Mahlke, W. Y. Chen, P. P. Chang, N. J. Warter, R. A. Bringmann, R. G. Ouellette, R. E. Hank, T. Kiyohara, G. E. Haab, J. G. Holm, and D. M. Lavery. The superblock: an effective technique for VLIW and superscalar compilation. *J. Supercomput.*, 7(1-2):229–248, 1993.

[8] Intel White Papers. Sustainable performance scaling for high-end enterprise and technical computing, 2005.

[9] S. A. Ito. Arquiteturas *VLIW*: Uma alternativa para exploração de paralelismo a nível de instrução. Technical report, Universidade Federal do Rio Grande do Sul, Junho 1998.

[10] A. Klaiber. The technology behind crusoe processors. Technical report, Transmeta Corporation, January 2000.

[11] W. R. Mark and K. Proudfoot. Compiling to a vliw fragment pipeline. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 47–56, New York, NY, USA, 2001. ACM Press.

[12] D. A. Patterson and H. J. L. *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1996.

[13] Philips White Papers. An introduction to very-long instruction word computer architecture, 2002. Pub#: 9397-750-01759.

[14] B. R. Rau. Dynamically scheduled vliw processors. In *MICRO 26: Proceedings of the 26th annual international symposium on Microarchitecture*, pages 80–92, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[15] K. W. Rudd. *VLIW Processors: Efficiently Exploiting Instruction Level Parallelism*. PhD thesis, Stanford University, December 1999.

[16] J. E. Smith and G. S. Sohi. The microarchitecture of superscalar processors. In *Proceedings of the IEEE*, volume 83, pages 1609–1624, Dec 1995.

[17] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar. Multiscalar processors. In *25 Years ISCA: Retrospectives and Reprints*, pages 521–532, 1998.

[18] R. M. Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development*, 11:25–33, Jan 1967.

[19] D. W. Wall. Limits of instruction-level parallelism. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, volume 26, pages 176–189, New York, NY, 1991. ACM Press.

[20] M. Wilkes. The genesis of microprogramming. *Ann. Hist. Comp.*, 8(2):115–126, 1986.

[21] H. Zhou, M. Jennings, and T. Conte. Tree traversal scheduling: A global scheduling technique for vliw/epic processors. In *Proceedings of the 14 th Annual Workshop on Languages and Compilers for Parallel Computing (LCPC'01)*. LNCS, Springer Verlag, 2001.