Arquitetura Raw

Ricardo M Nishihara
RA 936161 - UNICAMP
ricardo.nishihara@terra.com.br

RESUMO

Este trabalho descreve a arquitetura Raw desenvolvida pelo Laboratório de Ciência da Computação do Massachusetts Institute of Technology - MIT. A arquitetura proposta tem como desafio viabilizar um processador de propósito geral que tenha bom desempenho tanto em aplicações computacionais que envolvam *streaming de dados*, quanto em programas sequenciais típicos. No presente trabalho são apresentadas além da descrição dos principais aspectos desta arquitetura também algumas considerações sobre os desafios e características software de suporte (run-time e compilador) requerido para o bom desempenho do sistema. Também são fornecidas informações sobre testes comparativos envolvendo esta arquitetura e microprocessadores comerciais de complexidade equivalente.

1. INTRODUÇÃO [1,2]

O grande avanço da tecnologia de circuitos integrados tem expandido de maneira considerável o número de aplicações implementadas em VLSI. Tais aplicações incluem tanto programas sequenciais executados sobre microprocessadores de propósito geral (os quais utilizam técnicas para exploração de paralelismo no nível de instrução - ILP), quanto aplicações "streaming" com alto grau de paralelismo, implementadas até o momento, principalmente em hardware dedicado através de circuitos integrados específicos à aplicação - ASICs. Muitos destes circuitos ASIC "altamente paralelos" implementam algoritmos cujas demandas computacionais estão ainda muito acima das capacidades disponibilizadas pelos microprocessadores de propósito geral atuais. Exemplos de tais circuitos são decodificadores / processadores de áudio e vídeo, aceleradores gráficos, controladores de rede, codificadores para criptografia, etc.

O objetivo principal do projeto Raw foi fazer um melhor uso de tal avanço tecnológico propondo uma arquitetura que viabilizasse novos microprocessadores de propósito geral, capazes de executar com bom desempenho tanto aplicações sequenciais típicas explorando ILP, quanto um maior número de aplicações "*streaming*" com alto grau de paralelismo, e que até aqui vem sendo implementadas principalmente através de hardware dedicado.

Com o intuito de viabilizar esta inovação, os idealizadores da arquitetura Raw identificaram quatro fatores como responsáveis principais para o sucesso dos ASICs na implementação de aplicações "streaming" altamente paralelas:

1. Especialização: As implementações baseadas em ASIC podem especializar as operações requeridas pela aplicação no nível de *gate*, ao passo que estas mesmas operações implementadas em um microprocessador tem de ser feitas através de combinações de "sub-operações" definidas pelo conjunto de instruções do microprocessador. Esta implementação de operadores especializados viabilizada pelos ASICs, resulta em ganhos expressivos de desempenho. Por exemplo, a implementação de um operador específico como uma operação de ponto flutuante incompatível implementada em hardware dedicado tem o potencial de ser executada em um ou alguns ciclos de relógio, a passo que uma implementação equivalente em microprocessador pode requerer várias instruções e demandar vários ciclos de relógio para sua execução.

2. Maior utilização de recursos em paralelo: Abordagens baseadas em ASICs são capazes de viabilizar um grande número de operadores e canais de comunicações operando em paralelo, e tal capacidade costuma ser um requisito de aplicações "streaming". Embora existam microprocessadores superescalares e VLIW capazes de executar mais de uma instrução em um único ciclo de relógio (como por exemplo Itanium II capaz de executar até 6 instruções por ciclo), circuitos dedicados implementados em ASIC como por exemplo aceleradores gráficos costumam executar centenas ou milhares de operações paralelas no nível de *word* por ciclo.

3. Gerenciamento de condutores e dos atrasos de propagação associados: No projeto de ASICs o gerenciamento dos atrasos dos condutores é feito através do *placement/routing* adequado das várias unidades funcionais (ou operadores) requeridas pela aplicação. Com exemplos de boas práticas de *placement/routing* podemos citar: a colocação de mais próxima de unidades que se comunicar mais freqüentemente; a implementação de canais de comunicação dedicados nos pontos que requerem maior largura de banda; e a colocação de registradores *pipeline* entre operadores distantes, convertendo atrasos de propagação em ciclos de latência associados ao acesso destes registradores. Através destas estratégias, obtem-se um compromisso entre paralelismo e latência, de modo a maximizar o número de recursos usufruindo de um maior número de sinais para comunicação.

4. Gerenciamento de pinos: Implementações ASICs em geral não são limitadas por gargalos como a hierarquia do sistema de memória, pois procuram utilizar os pinos do encapsulamento de modo a melhor se adequar às necessidades da aplicação. Tal abordagem visa minimizar latências tanto no acesso a memórias DRAMs externas e maximizar a largura de banda de E/S, facilitando a interface com dispositivos E/S que demandam mais eficiência como CCDs, conversores A/D, matriz de sensores, etc. Os atuais mecanismos de E/S disponibilizados por microprocessadores de propósito geral ainda são ineficientes, principalmente devido ao uso do sistema de memória (baseado em DRAMs) como *buffer* intermediário.

O objetivo da arquitetura Raw é portanto, viabilizar um microprocessador que contemple os fatores mencionados, sem deixar de lado funcionalidades típicas de um processador de propósito geral. Neste sentido a arquitetura Raw adota a seguinte abordagem:

1. Implementação através de mecanismos de hardware especializado de operadores requeridos para a exploração de ILP

em aplicações seqüênciais, e/ou para o aumento de eficiência em aplicações "*streaming*". A maioria destes mecanismos é exposta ao software através da arquitetura do conjunto de instruções – ISA. Dentre estes mecanismos incluem-se: operações inteiras e de ponto flutuante usuais, operações especializadas para manipulação multigranulares (nos níveis de *bit*, *byte* e *word*), operações de roteamento de operandos entre unidades funcionais adjacentes, e bypass de operandos entre unidades funcionais, registradores, filas de E/S, e cache de dados.

2. A arquitetura Raw tem como paradigma fundamental replicar em grande número os operadores mencionados no item 1, e expô-los ao software através do ISA. Deste modo, o usuário (compilador ou programador) pode via software fazer uso destes recursos para explorar melhor tanto o ILP existente nas aplicações sequênciais quanto o alto grau de paralelismo das aplicações "*streaming*".

3. A arquitetura Raw gerencia o impacto dos atrasos de propagação nos condutores que inteligam unidades funcionais do processador expondo ao software operadores relativos aos canais de comunicação interligando estas unidades. Deste modo o software de suporte pode considerar as latências associadas ao realizar a organização do transporte de dados escalares e *streaming* entre estas unidades, e também criar novos padrões de comunicação dedicados a uma dada aplicação. Estes operadores relativos aos canais de comunicação quando considerados em conjunto provêm uma abstração para uma rede de operandos escalares de baixa latência que também pode ser usada na exploração de ILP.

4. As abstrações expostas pelo ISA referentes aos "pinos", permitem o gerenciamento via software de sistemas de memória cache e de interfaces E/S de alto desempenho.

O presente texto visa apresentar a arquitetura Raw desenvolvida pelo Laboratório de Ciência da Computação do Massachusetts Institute of Technology – MIT, e para tanto possui a seguinte organização: na seção 1 são apresentadas as motivações do projeto Raw; na seção 2 são considerados os principais os componentes definidos por esta arquitetura; na seção 3, são feitas algumas algumas considerações sobre características e desafios associados ao software de suporte (run-time e compilador) requeridos para um bom desempenho de um sistema Raw; na seção 4 são fornecidas informações sobre testes comparativos envolvendo esta arquitetura e microprocessadores comerciais de complexidade equivalente; e finalmente na seção 5 são apresentados os comentários finais.

2. ARQUITETURA RAW [3,4]

O paradigma fundamental da arquitetura Raw baseia-se na obtenção de um processador a partir da replicação de elementos de processamento mais simples e idênticos denominados *tiles*, sendo que cada *tile* é responsável pelo processamento de seu próprio fluxo de instruções.

Em sua implemetação inicial, conforme mostra a Figura 1, um processador RAW foi construído a partir de um conjunto de 16 *tiles* interconectados em um arranjo *mesh-2D*. Cada *tile* é dotado de um processador principal (dotado de um pipeline baseado no MIPS 32 bits e de memórias SRAM de instruções e de dados), de um roteador programável dedicado roteamento estático e dois roteadores dinâmicos.

2. ARQUITETURA RAW

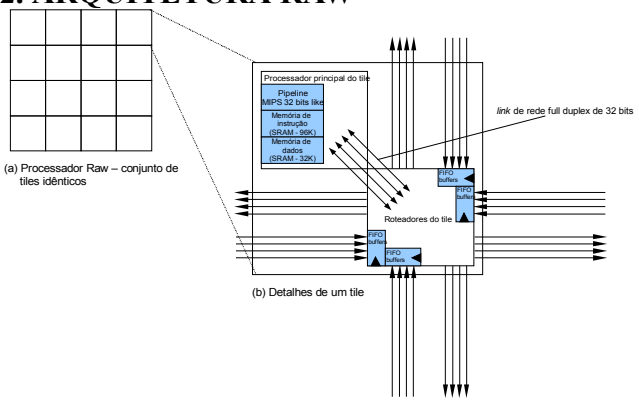

Figura 1 – Processador Raw – Um conjunto de *tiles* idênticos conectados através de um arranjo *mesh-2D*

Ainda conforme mostrado na Figura 1, os *tiles* são interconectados através de quatro redes *full duplex* de 32 bits integradas ao *chip*. Duas dessas redes são ditas estáticas, ou seja, com as rotas são especificadas em tempo de compilação, e as outras são ditas dinâmicas, ou seja, cujas rotas são especificadas em tempo de execução. Cada *tile* é conectado apenas a seus quatro vizinhos mais próximos (nas direções norte, sul, leste e oeste), ou seja o condutor mais longo do sistema tem no máximo a largura de um *tile*. Esta uma propriedade é muito importatnte para a escalabilidade da arquitetura. As referidas redes são expostas pelo ISA da arquitetura Raw ao software, permitindo que o programador ou compilador programe diretamente os canais de comunicação para controlar a transferência de dados entre os processadores principais dos *tiles,* numa abordagem análoga ao *placement/routing* realizado no projeto de circuitos ASICs para operadores especializados.

Ao contrário dos processadores superescalares, a arquitetura Raw não incorpora ao hardware estruturas lógicas para *register renaming*, ou para *scheduling* dinâmico de instruções. Ao invés disso, a orientação é mover estes mecanismos para o software e manter o *tile* o mais simples e pequeno possível, com intuito de: maximizar o número de *tiles* integrado no chip, aumentar a freqüência de relógio do sistema e aumentar a oferta de recursos computacionais que podem ser usados em paralelo.

Componentes do *tile*

Processador principal

O processador principal do *tile* é composto basicamente de um processador RISC *pipeline* implementando um conjunto de instruções MIPS de 32 bits com algumas modificações (tais como implementação de operações de manipulação multigranular, ou seja, nos níveis de *bit*, *byte* e *word; e* roteamento de operandos escalares entre unidades funcionais adjacentes).

Considerando o protótipo da implementação inicial, o processador principal conta com 32K de SRAM para memoria de dados, e 96K de SRAM para memoria de instrução. A memória de instrução não faz uso de cache, sendo sua virtualização implementada via software. A memória de dados pode fazer ou não uso de cache. Nos casos em que não se faz de caching em hardware a virtualização da memória de dados também fica a cargo do software.

A Figura 2 mostra o diagrama de blocos do pipeline RISC implementado no processador principal de um *tile*. Ele é bastante similar a um *single issue in order pipeline*, a menos da presença de FIFO buffers que constituem a interface do processador principal com as redes para comunicação entre *tiles*. É importante destacar o projeto bastante agressivo usado na integração destas interfaces de rede dentro pipeline do processador principal do *tile*. Tal abordagem foi adotada visando minimizar as latências da comunicação entre *tiles*. As interfaces de rede não só foram mapeadas como registradores, como foram também integradas diretamente ao *path* de *bypass* do *pipeline*, ou seja, este pipeline é capaz de ler dois valores direto da rede, executar uma operação sobre eles e enviar o resultado de volta para a rede sem acessar o banco de registradores.

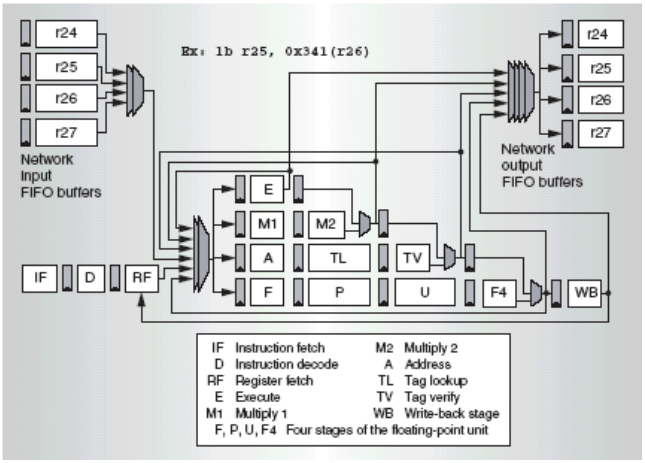

Figura 2 – Diagrama de blocos do pipeline RISC usado no processador principal de um *tile* da arquitetura Raw

Conforme ilustra a Figura 2, os registradores de 24 a 27 são mapeados para os quatro *ports* de rede. Assim por exemplo, uma leitura a partir do registrador 24 removerá um elemento do FIFO buffer associado, enquanto uma escrita enviará um dado para rede. Se nenhum dado está disponível no FIFO buffer de entrada, ou se não há espaço no FIFO buffer de saída para armazenar este dado, o pipeline do processador principal do tile sofrerá um *stall*.

Cada FIFO buffer de saída é conectado à saída de cada um dos estágio do pipeline. Assim os FIFO buffers podem retirar o valor "mais antigo" do pipeline assim que ele estiver pronto, ao invés de esperar atpe o final do estágio de *write-back*. Essa lógica é similar a lógica tradicional de bypass exceto pelo fato de que é dada prioridade às instruções mais antigas ao invés das instruções mais novas.

Ainda como uma pequena amostra de modo como estes mecanismos são expostos ao sofware ou programador, a Tabela 1 e a Figura 3 mostram respectivamente a definição dos *aliases* dos registradores especiais 24 a 27 associados aos FIFO buffers da interface de rede, e um pequeno trecho de código em linguagem assembly ilustrando sua utilização dos *ports* de rede.

Registrador	Alias	Descrição
$24	$csti	Port de entrada para a rede estática
$25	$csgn[i/o]	Port de E/S para a rede dinâmica de uso geral
$26	$csti2	Segundo port de entrada para a rede estática
$27	$cmn[i/o]	Port de E/S para a rede dinâmica reservada à memória

Tabela 1 – Mapeamento de registradores aos *ports* das redes estáticas e dinâmicas

```
# XOR register 2 with 15,
# and put result in register 31
xori $31,$2,15
# get two values from switch,
# add to register 3, and put
# result in register 9
addu $9,$csti2,$csti
# an ! indicates that the result
# of the operation should also
# be written to $csto
and! $0,$3,$2
# load from address at $csti+25
# put value in register 9 AND
# send it through $csto port
# to static switch
ld! $9,25($csti)
# jump through value specified
# by $csti2
jr $csti2
```

Figura 3 – Trecho de código assembly ilustrando o acesso às interfaces com as redes para comunicação entre tiles

Roteador estático

O roteador estático é um processador pipeline de 5-estágios que controla dois *crossbar switchers* e duas redes físicas. Cada crossbar rotea valores entre sete entidades: o processador pipeline do roteador estático, os quatro *tiles* vizinhos (norte, sul, leste e oeste), o processador principal do *tile*, e o outro crossbar.

O processador pipeline do roteador estático dispõe de uma memória de instruções com 8096 palavras e implementa um conjunto de instruções restrito com palavras de largura de 64 bits. Cada instrução codifica um pequeno comando (branches com ou sem decremento e acessos a um pequeno banco de registradores) e 13 rotas (uma para cada saída de crossbar).

Para cada palavra de dado transmitida entre *tiles* sobre a rede estática, deve existir uma instrução na memória de instrução de cada roteador estático, que faz parte do caminho percorrido pela palavra de dado. Estas instruções são programadas em tempo de compilação e são virtualizadas via software da mesma maneira qua as instruções do processador principal do *tile*. Portanto, os roteadores estáticos coletivamente reconfiguram inteiramente o

padrão de comunicação da rede em uma base ciclo a ciclo. Deve-se ainda ressaltar que em razão da memória de instrução do roteador ser virtualizada tem-se uma enorme flexibilidade na criação de novos padrões de comunicação.

Em razão do roteador estático saber qual rota será executada muito antes da chegada de uma palavra, preparações relativas a rota a ser seguida podem ser executadas em *pipeline.* Isso permite que a palavra seja roteada imediatamente após sua chegada ao roteador. Como veremos mais adiante a obtenção de baixa latência no roteamento estático é crítica para a exploração de ILP em aplicações seqüênciais.

Outra característica crítica para exploração de ILP, é o controle de fluxo do roteador estático. O roteador estático prossegue para a próxima instrução somente depois que todas as rotas de uma dada instrução são completadas. Isso assegura que os *tiles* destino recebam as palavras sempre em uma ordem conhecida, mesmo que ocorram situações como erro na predição de desvios, interrupções, *cache misses*, ou outros eventos não preditos.

Roteadores e redes dinâmicas

Conforme mencionado anteriormente, cada *tile* conta também com 2 roteadores dinâmicos. Quando consideramos o conjunto interligado de todos roteadores dinâmicos de todos os *tiles*, tem-se um par de redes dinâmicas para comunicação entre *tiles.* Para enviar uma mensagem em uma dessas redes, o usuário envia uma única palavra de cabeçalho especificando: o *tile* destino (ou *port* de E/S), um campo de usuário e o comprimento da mensagem. O usuário pode enviar até 31 palavras de dados em uma mensagem.

Uma das principais preocupações do uso de redes dinâmica é a ocorrência de *deadlocks* relativos a negociação de acesso aos *buffers* da rede (localizados nos roteadores dinâmicos distribuidos). Existem duas soluções clássicas para este problema: eliminar a ocorrência de *deadlock*, ou prover mecanismos que permitam recuperação a partir de sua ocorrência. Eliminação de *deadlock* requer que os usuários restrinjam seu uso a um conjunto de práticas comprovadamente livres de *deadlock*. Recuperação a partir de deadlock não coloca restrições à utilização da rede, mas requer que dados armazenado nos *buffers* da rede sejam movido para algum outro recurso externo de memória, quando for detectada a condição de *deadlock*.

A solução proposta pela arquitetura Raw utiliza duas redes dinâmicas fisicamente idênticas: Uma rede para acesso à memória com um modelo de uso restrito e eliminação de *deadlock*, e uma rede de acesso geral com modelo de uso irrestrito com um esquema de recuperação de *deadlock.*

Somente os clientes de acesso a rede privilegiado (como sistema operacional, cache de dados, interrupções, dispositivos de hardware, DMA, e *ports* E/S) podem usar a rede dinâmica de acesso à memória. O protocolo de acesso desta rede limita modo de uso permitido aos clientes de modo a eliminar *deadlocks*. Por exemplo o cliente não pode "bloquear" no envio de uma mensagem a menos que ele possa garantir de recursos em seus buffers de entrada para receber todas as mensagens a ele endereçadas.

Os demais clientes da rede dinâmica só podem utilizar a rede de propósito geral contam com o mecanismo de recuperação de *deadlock* implementado no sistema, para garantir a continuidade de sua interação. Por este mecanismo o sistema operacional programa um contador configurável no processador principal do *tile* para detectar se as palavras estão esperando muito tempo para serem enviadas. Este contator informa a ocorrência de *deadlock*, disparando uma interrupção que remove dados dos buffers da rede para memórias DRAM externas através da rede de acesso à memória. Uma segunda interrupção virtualiza o *port* de entrada da rede de proposito geral e permite a recuperação dos dados a partir das memórias DRAM externas.

3. SUPORTE DE SOFTWARE [1,5]

Considerações sobre o *run-time* software

Atingir bom desempenho em programas com padrões de desvios dependentes de dados e com operações de memória baseados em ponteiros requer mecanismos capazes de analisar e reagir ao comportamento do programa em tempo de execução. Em processadores superescalares, tais mecanismos são providos por unidades de hardware tais como caches, buffers para predição de desvios, lógica para *register renaming*, e unidades para escalonamento de instrução.

Em um processador Raw tais funcionalidades devem ser providas pelo *run-time* software do sistema. Por exemplo, o processador Raw implementa *caching* da memória de instruções em software. Neste caso o *run-time* software gerencia a hierarquia de memória executando a checagem de cada acesso de memória e provendo o mapeamento corrente do endereço requisitado. O compilador por sua vez tem neste situação duas atribuições: inserir código referentes aos estes testes (feitos a cada referência de memória), e eliminar os testes detectados como redundantes durante a compilação. Embora o *caching* via software seja mais custoso que a implementação em hardware, o primeiro pode valer-se de algoritmos mais sofisticados, e permitir se necessário a adequação de tais algoritmos a necessidades específicas da aplicação. Isto pode elevar a taxa de *hits* do sistema de *caching* compensando custos e overheads adicionais. Outro mecanismo que requer um suporte similar tanto do *run-time* software quanto do compilador é uso de execução com especulação. De maneira geral, os custos adicionais de se implementar tais serviços em software precisam ser reduzidos, e esta redução pode vir tanto através do compilador, a partir da eliminação de operações desnessárias, quanto a partir de algoritmos melhores de suporte. Como já mencionado o principal benefício de mover-se mecanismos de controle dinâmico para o software é a simplicação de hardware de controle, liberando mais espaço para maior disponibilização de recursos computacionais em paralelo, e permitindo maiores freqüência de relógio.

O desempenho geral de um sistema baseado na arquitetura Raw depende do compromisso de alguns fatores. Em razão do suporte a comportamentos dinâmicos dos programas ter sido movido do hardware para o software, um programa executará mais instruções que um sistema que implemente este suporte em hardware. Por outro lado, o hardware mais simples da arquitetura Raw poderá rodar a freqüências de clock maiores, e disporá de mais recursos computacionais para explorar o paralelismo da aplicação.

Considerações sobre o compilador

Como aspectos principais relacionados ao compilador dedicado a um sistema baseado na arquitetura Raw, devemos destacar: a alocação de recursos, a exploração de paralelismo de fina granularidade, e o escalonamento das comunicações entre *tiles.*

Alocação de recursos

Na execução de um programa no processador Raw, tanto o *run-time* software quanto o programa executado são mapeadas para uma coleção de *tiles* idênticos. Ao invés de uma alocação de uma alocação fixa pré-definida em hardware, a arquitetura Raw permite que os *tiles* sejam alocados de uma maneira específica definida pela aplicação. Deste modo, mais recursos podem ser alocados para as partes mais críticas do run-time software ou do programa em execução.

Além disso, o programa em execução pode ser dividido em regiões paralelas de maior granularidade. Cada uma destas regiões paralelas pode ser executada em múltiplos *tiles*, os quais em conjunto comportam-se como um único processador lógico. O número de *tiles* alocados para uma dada região depende da existência de paralelismo de granularidade mais fina dentro desta região.

Os *tiles* dentro de uma dada região comunicam-se entre si usando principalmente comunicações estáticas, enquanto as regiões comunicam-se entre si ou com run-time software usando mensagens dinâmicas.

O tamanho e o número de regiões alocadas pode ser decidido pelo compilador baseando-se na natureza do paralelismo existente na aplicação. Por exemplo, pode-se usar um pequeno número de regiões grandes quando um alto grau de paralelismo de granularidade fina for detectado pelo compilador, e um grande número de aplicações de regiões pequenas quando for detectado somente paralelismo de maior granularidade.

O compilador para um sistema Raw portanto, deve ser capaz de analisar um programa como um todo, com intuito de obter os requisitos demandados pelo run-time software, e detectar a disponibilidade de paralelismo de diferentes granularidades.

Exploração de paralelismo de granularidade fina

Uma oportunidade provida pela arquitetura Raw, devido às baixas latências associadas a comunicação entre *tiles* via rede estática, é a capacidade de explorar eficientemente o paralelismo de granularidade fina ao longo dos tiles. Nesta abordagem, múltiplos *tiles* podem trabalhar juntos para explorar o paralelismo no nível de instrução - ILP de um fluxo único de instruções. O compilador toma este fluxo único de instruções como entrada, particiona-o em multiplos fluxos de instruções, mapea cada fluxo particionado para um *tile*, e finalmente escalona a comunicação estática entre os fluxos particionados.

Esta abordagem da arquitetura Raw para exploração de ILP difere da utilizada em processadores superescalares e VLIW. Ao contrário dos processadores superescalares, um processador Raw explora ILP sem fazer uso de lógica complexa implementada em hardware. Os mecanismos de exploração são movidos para o software e são realizados de maneira estática pelo compilador. Em constraste com os processadores VLIW cada unidade de processamento da arquitetura Raw, ou seja o *tile*, tem seu próprio fluxo de instruções. Essa abordagem é mais flexível porque não requer que a execução seja feita em passos estanques determinados pelos blocos de múltiplas instruções contidas na palavra longa do VLIW.

A arquitetura Raw também oferece oportunidades para novas otimizações por parte do compilador, como por exemplo, novas formas de realizar o *spill* de registradores. Em processadores convencionais, o spill de registradores é feito para a memória. O processador Raw permite que o *spill* de registradores seja alternativamente feito para os *tiles* vizinhos.

Escalonamento de comunicação

Compiladores tradicionais realizam o escalonamento de um único tipo de evento (ou seja, instruções) ao longo de uma dimensão única (neste caso, o tempo). Devido a presença da rede estática, um compilador para arquitetura Raw enfrenta um problema mais complexo: ele tem de escalonar tanto instruções quanto eventos de comunicação, sendo que ambos tipos de eventos têm de ser escalonados temporalmente e espacialmente.

A compilação de código para uso da rede estática resulta em desafios relacionados a correção e desempenho. Com relação à correção, o código de roteamento tem de ser livre de *deadlock*. Com relação ao desempenho, o compilador deve ser capaz de organizar cuidadosamente a comunicação de maneira a minimizar a ocorrência de *stalls* ao longo da rede. Estes dois aspectos tem ainda sua complexidade aumentada na presença de eventos dinâmicos como desvios, *cash misses*, e mensagens dinâmicas.

Quando uma aplicação faz uso tanto de redes estáticas quanto dinâmicas, aspectos relativos à interação destas duas rede também devem ser considerados. O compilador tem de garantir que os programas gerados livres de *deadlock* neste novo contexto, e provavelmente terá de valer-se de otimizações temporais para compensar as diferenças de desempenho e comportamento das redes estáticas e dinâmicas (a temporização nas redes estáticas é altamente previsível, enquanto os atrasos devido às mensagens dinâmicas podem ser grandes e altamente imprevisíveis).

4. IMPLEMENTAÇÃO E TESTES

Nesta seção apresenta-se algumas informações relativas à implementação do protótipo ASIC do processador Raw, e sobre os testes comparativo da arquitetura Raw e um microprocessador comercialmente existente de complexidade equivalente – Pentium 3.

Protótipo de validação implementado [2,3]

Os proponentes desta arquitetura Raw implementarão um chip com um array de *16 tiles* usando como tecnologia de implementação o processo ASIC da IBM SA-27E (0.15-micron, 6 níveis, cobre). Foi utilizado um die de tamanho 18.2 x 18.2 mm e um encapsulamento CCGA *(ceramic column grid array package)* de 1657 pinos, com 1080 deles, pinos de E/S HSTL (*high speed transceiver logic)*. O chip consome em média 18.2 watts operando à 425MHz, e opera em temperatura ambiente com freqüência de relógio nominal de 425MHz/500Mhz com alimentação de 1.8V/2.2V. Esses valores são da mesma ordem dos obtidos por outros processadores IBM usando o mesmo processo, como exemplo: o PowerPC 405GP que opera na faixa de 266-400 Mhz, e o PowerPC 440GP que opera na faixa de 400-500 Mhz. A Figura 4 mostra fotos do die do chip Raw e da motherboard construída a partir do chip.

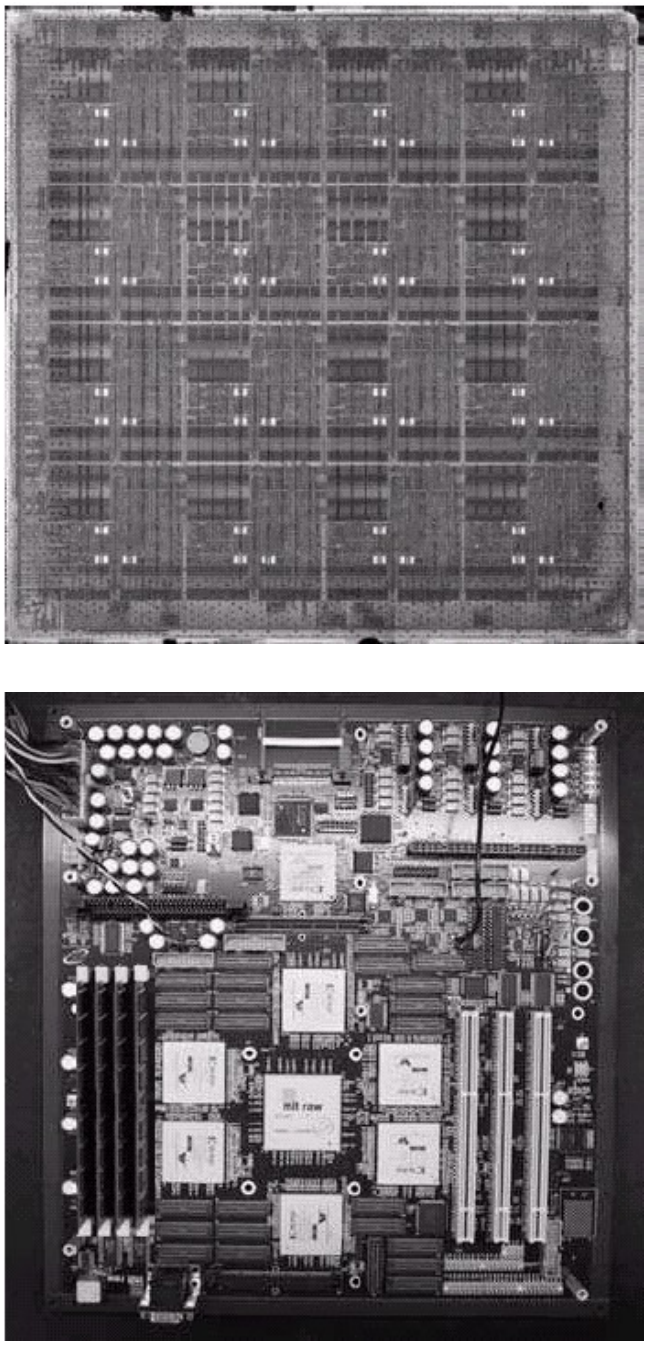

Figura 4 – Fotos do chip Raw e da motherboard construída em torno do chip Raw

Testes comparativos [2]

A referência [2] descreve em detalhes uma avaliação comparativa realizada com a implementação do processador Raw. Nesta subseção destacaremos alguns pontos da metodologia utilizada e comentaremos alguns resultados dos testes realizados.

Comparação com um microprocessador comercial existente

A metodologia usada na avaliação do processador baseou-se na sua comparação direta com um microprocessador existente comercialmente. Tal abordagem além de mais honesta por não esconder ineficiênciais do compilador e da arquitetura, facilita a comparação indireta da arquitetura Raw com outras alternativas desenvolvidas por outros grupos, uma vez que, a comparação de um dada alternativa com o mesmo sistema comercial de referência pode ser extentida para uma comparação com a arquitetura Raw.

O processador Pentium 3 foi o escolhido por ser a implementação de processador Intel mais próxima do protótipo Raw. O Pentium 3 foi implementado usado um processo com a mesma geração de litografia do protótipo Raw, e possui latências associadas às unidades funcionais muito próximas as do protótipo Raw.

A Tabela 2 a seguir mostra uma comparação dos parâmetros de implementação do protótipo Raw e do Pentium 3.

Parameter	Raw (IBM ASIC)	P3 (Intel)
Lithography Generation	180 nm	180 nm
Process Name	CMOS 7SF (SA-27E)	P858
Metal Layers	Cu 6	Al 6
Dielectric Material	SiO_2	SiOF
Oxide Thickness (T_{ox})	3.5 nm	3.0 nm
SRAM Cell Size	4.8 μm^2	5.6 μm^2
Dielectric k	4.1	3.55
Ring Oscillator Stage (FO1)	23 ps	11 ps
Dynamic Logic, Custom Macros (SRAMs, RFs)	no	yes
Speedpath Tuning since First Silicon	no	yes
Initial Frequency	425 MHz	500-733 MHz
Die Area[2]	331 mm^2	106 mm^2
Signal Pins	~ 1100	~ 190
Vdd used	1.8 V	1.65 V
Nominal Process Vdd	1.8 V	1.5 V

Tabela 2 – Parametros de implementação: Rw x Pentium 3

Validação usando simulador

Embora o protótipo em hardware do processador Raw estivesse disponível, optou-se por realizar a avaliação através de um simulador do processador Raw com precisão de ciclo de relógio. Segundo os proponentes da arquitetura Raw, tal simulador tem exatamente o mesmo comportamento RTL do chip Raw implementado pelo processo ASIC da IBM, e a razão para este procedimento foi a necessidade de normalização de alguns itens para viabilizar a comparação. Neste sentido menciona-se por exemplo a normalização de latências associadas à motherboard e às memórias DRAM, e a troca do sistema de caching por software do Raw (um algoritmo ainda em fase de pesquisa) por um sistema de caching convencional implementado em hardware para que uma comparação mais direta com o Pentium 3.

Ferramentas de SW utilizadas

A Tabela 3 a seguir lista as ferramentas de sofware utilizadas nos testes de comparação reportados em [2] pelos sistemas Raw e Pentium.

Pentium 3	Raw
> gcc 3.3 -O3 -march=pentium3 -mfpmath=sse > Intel Performance Primitives > LAPACK/BLAS com SSE para rotinas de algebra linear	> rawcc – compilador desenvolvido internamente > Streamit - compilador desenvolvido internamente > gcc 3.3

Tabela 3: Ferramentas de SW usadas nos testes de comparação

Sumário dos resultados dos testes

A Figura 5 apresenta um sumário dos resultados dos testes de desempenho envolvendo inúmeras aplicações. Os resultados são apresentados na forma de *speedup* relativos ao Pentium 3. As aplicações foram divididas em 4 classes:

1. ILP: Na qual foram colocadas aplicações sequenciais convencionais. Tipicamente, a única forma de paralelismo disponível nestas aplicações é o ILP. Para esta avaliação foram selecionados programas com diferentes graus de ILP.

2. Stream: Nesta classe foram colocadas aplicações "streaming" que lidam com grandes conjuntos de dados, ou podem manipular fluxos contínuos de dados em tempo-real.

3. Server: Para medir o desempenho do processador Raw em termos de workload de servidor, foi conduzido o seguinte experimento para se obter resultados SpecRate: para cada subconjunto de aplicações Spec 2000, executou-se uma cópia independente deste subconjunto em cada um dos 16 *tiles*, e mediu-se o throughput total deste workload em relação a uma única execução no Pentium 3.

4. Bit-Level: Inclui duas aplicações manipulação no nível de bit em geral implementadas em FPGA e ASIC: 802.11a ConvEnc, 8b/10b Encoder.

A partir do gráfico da Figura 5, pode-se fazer várias observações. Com relação as aplicações seqüenciais típicas observa-se que o Pentium 3 saiu-se melhor que o Raw para aplicações com baixo de grau de ILP, enquanto ocorre o oposto em aplicações com maior grau de ILP como Vpenta. Para aplicações "streaming" e vetoriais, o Raw supera o Pentium 3 por fatores de 10 a 100 vezes. Um servidor construido a partir de 16 processadores Pentium 3 foi escolhido como o melhor sistema servidor. Note que um sistema com um único chip Raw fica a um fator de apenas 3 abaixo deste melhor servidor para a maioria das aplicações. Com relação as aplicações da classe *bit-level*, o desempenho do Raw fica de um fator de 2 a 3 vezes abaixo das implementações em ASIC.

A tabela 4 a seguir sumariza as funcionalidades primárias responsáveis pelos ganhos de performace do processador Raw.

Classe	Benchmark	S	R	W	P
ILP	Swim, Tomcatv, Btrix, Cholesky, Vpenta, Mxm, Life, Jacobi, Fpppp-kernel, SHA, AES, Encode, Unstructured, 172.mgrid, 173.applu, 177.mesa, 183.equake, 188.ammp, 301.apsi, 175.vpr, 181.mcf, 197.parser, 256.bzip2, 300.twolf	X	X	X	
Stream	Beamformer, Bitonic Sort, FFT, Filterbank, FIR, FMRadio, Mxm, LU fact., Triang. solver, QR fact., Conv., Copy, Scale, Add, Scale & Add, Acoustic Beamforming, FIR, FFT, Beam Steering, Corner Turn, CSLC	X	X	X	X
Server	172.mgrid, 173.applu, 177.mesa, 183.equake, 188.ammp, 301.apsi, 175.vpr, 181.mcf, 197.parser, 256.bzip2, 300.twolf		X		X
Bit-level	802.11a ConvEnc, 8b/10b Encoder	X	X	X	

Tabela 4: Utilização de funcionalidades Raw. S = Especialização R = Utilização de Recursor Paralelos. W = Gerenciamento dos atrasos dos condutores. P = Gerenciamento de Pinos.

5. CONCLUSÃO

Este trabalho apresenta a arquitetura Raw proposta com desafio viabilizar um processador de propósito geral que tenha um bom desempenho tanto em aplicações computacionais altamente paralelas que envolvam *streaming de dados*, quanto nos programas sequenciais tradicionais (através da exploração de ILP). A arquitetura Raw procura atingir este objetivo a partir da de uma arquitetura reconfigurável construida a partir da replicação de elementos de processamento conhecidos como tiles, e da implementação de redes integradas de comunicação entre tiles de baixíssima latência. Estes recursos são expostos pelo ISA ao software, e deste modo, o usuário (compilador ou programador) pode fazer uso deles para explorar melhor tanto o ILP existente nas aplicações sequênciais quanto o forte paralelismo das aplicações "*streaming*". Os resultados dos testes de avaliação reportam ganhos de desempenho moderados em relação a processadores convencionais de complexidade equivalente (Pentium 3) na exploração de ILP de aplicações seqüências (fatores de 2 a 6 vezes) e ganhos expressivos de desempenho quando aplicações "streaming" são consideradas (fatores de 10 a 100 vezes).

REFERENCES

[1] E.Waingold, et al. Baring It All to Software: Raw Machines. IEEE Computer 30, 9 (September 1997), pp. 86–93.

[2] M. B. Taylor, et al. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. Proceedings of International Symposium on Computer Architecture, June 2004

[3] M. B. Taylor, et al. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. IEEE Micro (Mar 2002), pp. 25–35.

[4] M. B. Taylor. Design Decisions in the Implementation of a Raw Architecture Workstation. MIT MS Thesis, Cambridge, MA, September, 1999

[5] A. Agarwal, et al. The Raw Compiler Project. Proceedings of the Second SUIF Compiler Workshop, Stanford, CA, August , 1997.

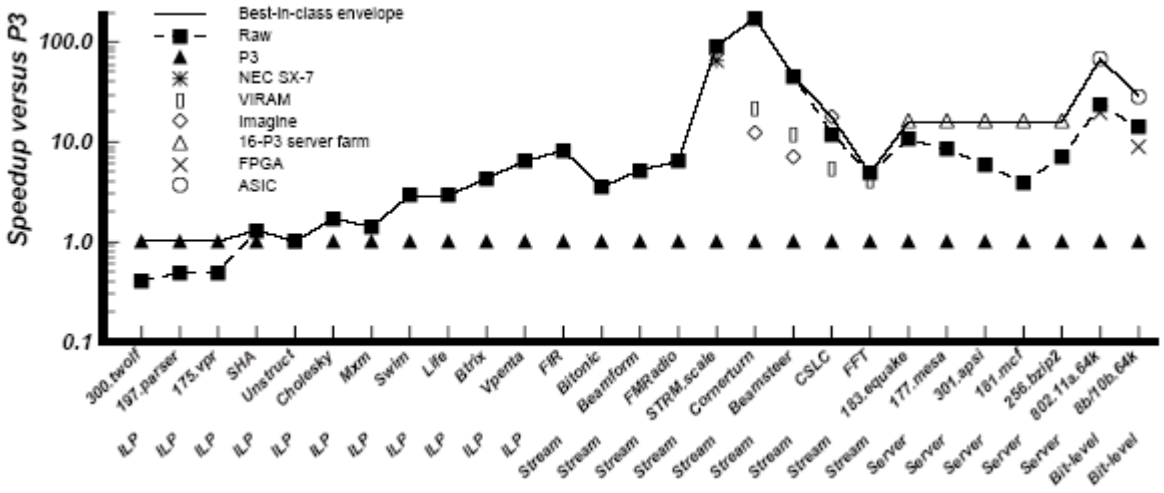

Figura 5 – Desempenho comparativo para várias aplicações