Arquiteturas Multicore *

Giovani Chiachia
RA: 098362
Instituto de Computação
Universidade Estadual de Campinas
giovanichiachia@gmail.com

RESUMO

O ritmo de evolução das máquinas paralelas mudou drasticamente com a adoção por parte das grandes companhias ao modelo de computação de múltiplos núcleos. Previsões indicam que no ano de 2022 serão necessários processadores com um desempenho 300 vezes maior que os atuais. Até mesmo em contextos mais restritos os processadores multicore estão sendo exigidos. Isso faz com que eles sejam objeto de grande interesse comercial e científico. Esse trabalho discorre sobre as muitas possibilidades de implementação das arquiteturas multicore, ponderando as particularidades de cada uma delas e oferecendo um panorama sobre os modelos comerciais as concebe.

Palavras-chave

Multicore, Arquitetura de Computadores, Computação Paralela

1. INTRODUÇÃO

A história da computação paralela pode ser contada a partir de meados da década de 1960, com o surgimento do computador Solomon [1]. Porém, a dificuldade em programá-lo restringiu inicialmente seu uso a cientistas e engenheiros que entendiam o domínio de suas aplicações e tinham recursos e conhecimentos para tal. Nas décadas subsequentes, esse mesmo motivo fez com que iniciativas de muitas companhias na criação de máquinas paralelas fossem frustradas. Uma das exceções foi a máquina vetorial Cray. Porém, essas máquinas também possuíam um processador escalar muito rápido que podia ser programado de maneira convencional. Além disso, o paradigma de programação vetorial não era tão intimidador quanto a programação paralela de propósito geral [1].

Recentemente, o ritmo de evolução das máquinas paralelas mudou drasticamente com a adoção por parte das grandes companhias ao modelo de computação de múltiplos núcleos. Esta foi a forma encontrada por seus arquitetos para manter a taxa de aumento de desempenho estabelecida pela lei de Moore e observada pelos consumidores. O paradigma de evolução até então baseava-se principalmente no aumento da frequência de operação dos processadores de núcleo único. No entanto, o aumento no consumo de energia resultante desta abordagem é proporcionalmente maior que o ganho de desempenho. Em função disto, o superaquecimento destes processadores tornou-se uma de suas principais limitações [2].

Na abordagem multicore, o aumento do desempenho está relacionado com o aumento do número de núcleos de processamento, e não com a frequência de operação dos núcleos. Embora isso seja uma grande vantagem, esta solução traz consigo as complicações inerentes à programação paralela. Em grande parte, estas complicações são resultantes do gerenciamento hierárquico das memórias, nas quais os dados armazenados precisam ser compartilhados por diferentes processadores [2].

De acordo com o *roadmap* ITRS [3], para atender as demandas crescentes, no ano de 2022 serão necessários processadores com um desempenho 300 vezes maior que os atuais. Até mesmo em contextos mais restritos os processadores multicore de propósito geral já estão sendo, e serão ainda mais, exigidos. Em dispositivos como os *smart phones*, por exemplo, tais processadores vêm de encontro com a necessidade de processar sinais digitais de forma simultânea com aplicações de outras naturezas. Essa perspectiva, aliada aos desafios que ela representa, faz com que as arquiteturas multicore sejam objeto de grande interesse comercial e científico.

Conforme veremos ao longo das próximas seções, o objetivo deste trabalho é apresentar uma visão geral sobre as arquiteturas multicore. Para isto, cobriremos na Seção 2 os principais atributos que as caracterizam. Em seguida, na Seção 3, apresentaremos uma série de implementações comerciais de arquiteturas multicore. Ao final dessa mesma seção, três dessas arquiteturas, cada qual com suas particularidades, serão apresentadas com maior refinamento. Por fim, na Seção 4 faremos algumas considerações finais sobre o tema.

*Trabalho 2 da disciplina de Arquitetura de Computadores do Programa de Pós-Graduação do IC-UNICAMP `http://www.ic.unicamp.br/~ducatte/mo401/1s2010/trabalho2.html`

2. PRINCIPAIS ATRIBUTOS

As arquiteturas multicore podem ser classificadas de muitas maneiras. De acordo com Blake et al. [1], a classe de aplicação, a relação consumo de energia/desempenho, os elementos de processamento e o sistema de memória estão entre os atributos mais importantes.

2.1 Classe de Aplicação

Se uma arquitetura é concebida para atender a um domínio específico de aplicação, ela pode ser desenhada para responder melhor às características desse domínio. Como resultado, espera-se que esta arquitetura seja eficiente para esse domínio, mas provavelmente ela será ineficiente para outros. O ajuste de uma solução para um domínio específico de aplicação pode trazer muitas consequências positivas. A melhor delas talvez seja a economia de energia. Processadores de Sinais Digitais (DSPs) são um bom exemplo disso. Os domínios de aplicação podem ser divididos em duas grandes classses: Os dominados pelo processamento de dados e os em que o processamento de controles prevalece.

2.1.1 Processamento de Dados

Muitas aplicações são dominadas pelo processamento de dados. Entre as mais conhecidas, podemos citar a rasterização de gráficos, os processamentos de imagens e de áudio e o processamento de comunicações *wireless*. Muitos algoritmos para processamento de sinais fazem parte deste grupo. Nessas aplicações, a computação baseia-se tipicamente em uma sequência de operações sobre um *stream* de dados onde pouca ou nenhuma informação é reusada e um alto *throughput* é necessário. Estas operações podem frequentemente ser feitas em paralelo. Por isso, elas favorecem uma arquitetura que tenha tantos elementos de processamento quanto possível, mas sem exceder a relação entre consumo de energia e desempenho almejada.

2.1.2 Processamento de Controles

As aplicações aonde o processamento de controles é dominante incluem a compressão/descompressão de arquivos, os processamentos em rede e a execução de operações transacionais. Nesses tipos de aplicação, os programas tendem a possuir muitos desvios condicionais, dificultando o paralelismo. Além disso, eles muitas vezes precisam manter uma série de informações sobre seu próprio estado e normalmente fazem reuso de uma grande quantidade de dados. Por isso, essas aplicações favorecem um número modesto de elementos de processamento de propósito geral para lidar com a natureza desestruturada do seu código devido aos controles.

Na maioria dos casos, nenhuma aplicação se ajusta exatamente a uma dessas duas divisões. Em um *codec* de vídeo, por exemplo, o processamento pode ser dominado por dados ao se processar algum filtro sobre os quadros, mas pode ser dominados por estruturas de controle quando o vídeo é comprimido ou descomprimido. No entanto, é interessante pensar nessa divisão para entender como diferentes arquiteturas multicore podem afetar o desempenho de aplicações com características específicas.

2.2 Relação Consumo de Energia/Desempenho

Muitos dispositivos possuem requisitos de consumo de energia e desempenho restritos. Um telefone celular com suporte a execução de vídeos, por exemplo, deve gastar pouca energia para isso. Mesmo assim, a execução do vídeo deve satisfazer certas características de qualidade.

O desempenho dos processadores sempre foi o principal objetivo dos projetistas. Porém, na última década, o baixo consumo de energia também passou a ser tratado como um requisito de grande importância. Em grande parte, isto se deve ao crescimento do mercado de telefones celulares e de computadores portáteis, onde o tamanho e a vida útil das baterias são restrições críticas. Mais recentemente, o crescimento dos *data centers* como suporte à *computação em nuvem* fez com que o consumo de energia também se tornasse um atributo importante no desenho de computadores não-portáteis. O modelo de processamento de propósito geral com múltiplos núcleos se encaixa muito bem às demandas desses *data centers*, que já consomem mais energia que as industrias de base nos Estados Unidos [1]. Esses *data centers* são tão grandes que o consumo de energia dos componentes multicore são críticos para os seus custos e sua operação.

2.3 Elementos de Processamento

Sobre os elementos de processamento, podemos abordá-los em dois aspectos, sua arquitetura e sua microarquitetura. A arquitetura está relacionada ao conjunto de instruções (em inglês, ISA) e define a interface entre o hardware e o software. A microarquitetura, por sua vez, tem a ver com a implementação do conjunto de instruções.

2.3.1 Arquitetura

Em processadores multicore convencionais, o ISA de cada núcleo é tipicamente um ISA legado dos processadores mais antigos com pequenas modificações para suportar paralelismos, tal como a adição de instruções atômicas para sincronização. A vantagem desses ISAs que mantêm o legado das instruções é a sua compatibilidade tanto com as implementações quanto com as ferramentas de programação já existentes.

Os ISAs podem ser classificados como reduzidos (RISC, em inglês) ou complexos (CISC, em inglês). No entanto, muitas máquinas CISC se parecem com suas correspondentes RISC uma vez que a decodificação das instruções complexas é feita. Os códigos em uma máquina CISC são menores em função da riqueza semântica das instruções. Por outro lado, as máquinas RISC facilitam o trabalho dos compiladores e sua microarquitetura é mais fácil de ser implementada. Além das definições básicas do ISA, os fabricantes têm continuadamente melhorado o desempenho de operações mais comuns através de extensões adicionadas ao conjunto. A Intel, por exemplo, adicionou os subconjuntos MMX, MMX2 e SSE1-4 [4] para melhorar o desempenho dos processamentos multimídia. De forma similar, outros fabricantes seguiram o mesmo caminho. Essas instruções permitiram melhorar a relação entre consumo de energia e desempenho por empregarem partes especializadas do hardware em sua execução (e.g., obter a transposta de um vetor).

2.3.2 Microarquitetura

A microarquitetura dos elementos de processamento é responsável, em muitos aspectos, pelo desempenho e pelo consumo de energia que podem ser esperados de um multicore.

Essa microarquitetura é normalmente associada ao domínio de aplicação para o qual o processador multicore é desenhado. Apesar de grandes fabricantes como a Intel oferecem ao mercado processadores multicore com núcleos homogêneos, muitas vezes é vantajoso combinar diferentes tipos de elementos de processamento em uma mesma arquitetura. Uma forma de organização heterogênea consiste no emprego de um processador de controle para comandar as atividades de um grupo de núcleos mais simples dedicados ao processamento de dados. Um dos benefícios dessa organização é reduzir o consumo de energia sem perder desempenho. Porém, o modelo de programação desse tipo de arquitetura é normalmente mais complicado.

O tipo mais simples de elemento de processamento faz o despacho das instruções na ordem em que elas estão no programa. Nele, os *data forwarding* e os *hazards* de controle são tratados dinamicamente. A partir desse modelo, há duas frentes que podem ser trabalhadas para obter um desempenho superior. Primeiramente, múltiplos *pipelines* podem ser adicionados para ler e despachar mais de uma instrução em paralelo, criando um elemento de processamento *superscalar*. No entanto, o aumento no número de instruções despachadas requer uma lógica extra para lidar com os *data forwarding* e as detecções de *hazards* necessárias para assegurar a corretude da execução. A complexidade dessa lógica cresce mais do que o quadrado do número de *pipelines* e o ponto em que os ganhos não são compensadores é rapidamente atingido. Experimentos com processadores de propósito geral sugerem que esse ponto seja em torno de três a quatro *pipelines*, mas isso depende muito da aplicação [1]. A segunda frente de melhorias baseia-se no aumento do número de estágios do pipeline, reduzindo a lógica contida em cada estágio. Isso permite que o período do *clock* seja reduzido, mas também faz com que a penalidade diante de um desvio seja maior. Os elementos de processamento com despacho em ordem são fisicamente menores, consomem menos energia e podem ser facilmente combinados em grandes quantidades para atender aplicações que possuem alto paralelismo em nível de *thread* (TLP, em inglês) e pequenos trechos de processamento sequencial. Como exemplo, a arquitetura G200 da NVIDIA [5] implementa 240 núcleos em ordem em função do processamento gráfico ser altamente paralelo, com poucas seções de processamento sequencial.

Para explorar o tanto quanto possível de paralelimo na execução de códigos sequenciais em núcleos superescalares, a execução *fora de ordem* é empregada. Seu objetivo é encontrar dinamicamente uma sequencia em que as operações possam ser executadas de modo a manter o *pipeline* o mais "cheio" possível. Todavia, este agendamento dinâmico requer o emprego de circuitos complexos e que consomem muito energia. Os núcleos com execução fora de ordem são mais adequados a aplicações que possuem muitos tipos de comportamento e que demandam por alto desempenho. A maioria dos processadores fora de ordem fazem o despacho de múltiplas instruções simultaneamente. Pelo fato deles serem maiores e consumirem mais energia, na prática, poucos núcleos fora de ordem podem ser combinados. Mesmo assim, eles são preferíveis para execução de aplicações que são dominadas por extruturas de controle com grandes porções de código sequencial e que possuem um TLP moderado. Esse é o caso do processador ARM Cortex A9 [6], dirigido ao uso em netbooks e que emprega um conjunto de núcleos fora-de-ordem.

Ainda no intuito de melhorar o desempenho das arquiteturas superescalares, mas desconsiderando a necessidade de lidar com a lógica complexa necessária para execução apropriada de uma sequencia de instruções, os conceitos de *única instrução sobre múltiplos dados* (SIMD, em inglês) ou de *instruções com palavras muito longas* (VLIW, em inglês) podem ser empregados. A arquitetura SIMD faz uso de registradores maiores que são divididos em porções de informação menores a serem processadas simultaneamente pela mesma instrução. Um exemplo simples de utilização do conceito é no cálculo do produto escalar entre dois vetores, onde cada par de elementos é processado em paralelo. Este tipo de arquitetura é muito aderente a aplicações onde o processamento de dados é intensivo e independente. O processador IBM Cell [7] faz uso de múltiplos núcleos SIMD talhados para execução de aplicações dominadas pelo processamento de dados.

Para evitar as limitações da arquitetura SIMD, onde apenas uma instrução pode operar sobre os dados, as instruções VLIW podem ser usadas. Nela, múlitplos *pipelines* são usados. Porém, ao contrário de um núcleo superescalar típico, não há necessidade de se fazer *forwarding*, de agendar as instruções, ou ainda de detectar *hazards*, pois espera-se que o compilador faça isso através do agrupamento das instruções em pacotes que possam ser executados em paralelo sem ferir restrições de dados ou de controle. Como se pode observar, a complexidade nessa arquitetura é transferida para o compilador e ela pode ser muito afetada caso o compilador não consiga encontrar um nível satisfatório de paralelismo. As arquiteturas VLIW e SIMD resultam em boas relações entre desempenho e consumo de energia, mas normalmente são adequadas somente para nichos específicos de aplicação com um grande número de operações independentes que podem ser exploradas pelo compilador ou pelo programador.

2.4 Sistema de Memória

Em arquiteturas monoprocessadas, o sistema de memória é relativamente simples, consistindo em alguns poucos níveis de memória cache para alimentar o processador com dados e instruções. Nas arquiteturas multicore, as memórias cache são apenas parte do sistema de memória. Os outros componentes incluem o modelo de consistência dos dados, os mecanismos para coerência entre as caches e a forma de interconexão entre elas. Esses componentes determinam como os núcleos se comunicam e impactam na programação, no desempenho das aplicações paralelas e no número de núcleos que o sistema comporta.

2.4.1 Modelo de Consistência

O modelo de consistência define como as operações em memória podem ser reordenadas enquanto o código está sendo executado. Ele determina o quanto de esforço é requerido do programador ao escrever códigos paralelos. Modelos mais fracos exigem que o programador defina explicitamente como o código deve ser agendado no processador e possui protocolos de sincronização mais complexos. Por outro lado, os modelos mais robustos requerem menos esforço do programador com relação às sincronizações. O uso de um ou de outro também causa um impacto direto no desempenho. Por

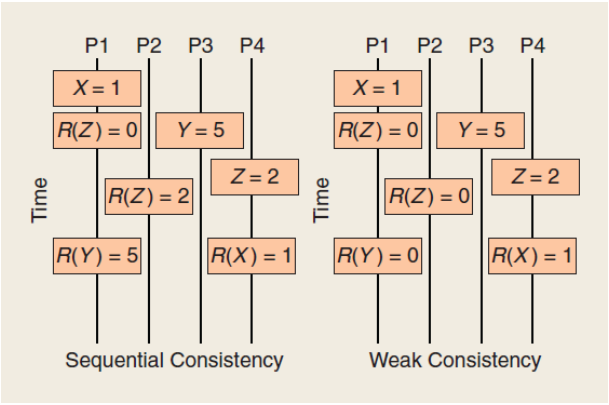

Figura 1: Ilustração dos modelos de consistência [1].

restringirem a ordem como o sistema de memória é autorizado a propagar leituras e escritas aos processadores, os modelos mais robustos impactam consideravelmente o desempenho da aplicação. No modelo de consistência sequencial, por exemplo, todos os processadores do sistema devem enxergar todas as leituras e escritas tal como elas ocorrem globalmente e no programa. Já nos modelos mais fracos, os processadores não precisam necessariamente enxergam as leituras e escritas feitas por outros processadores na ordem em que elas acontecem. Em função dessa baixa consistência, os modelos mais fracos requerem que os programadores façam uso de primitivas conhecidas como *barriers* e *fences* para forçar a consistência quando ela é necessária. Os modelos de consistências estão ilustrados na Figura 1. Nela, cada um dos processadores P1-P4 estão despachando escritas (e.g., $X = 1$) e leituras (e.g., $R(Z) = 0$). O modelo de consistência sequencial determina que todas as leituras e escritas para todos os endereços sejam vistas na mesma ordem por todos os processadores. Isso é o que ocorre, por exemplo, quando P2 lê Z e o valor retornado é o que foi escrito previamente por P4. Essa comunicação é tipicamente forçada por uma mediação estabelecida através da rede de interconexão. No caso do modelo de consistência fraca, P2 lê Z e o resultado retornado é zero. Neste caso, o modelo de consistência permite que núcleos diferentes enxerguem diferentes ordens globais de execução. Por relaxarem a sincronização, esses modelos são mais fáceis de serem implementados, deixando o esforço por conta do programador. Já os modelos mais robustos, além de complicados, também são lentos pelo fato de não obterem as vantagens do acesso fora de ordem à memória.

2.4.2 Configuração da Cache

As memórias caches são muito importantes nas arquiteturas multicore, onde muitos elementos de processamento tentam acessar a memória principal, que é lenta por definição. O tamanho que os diferentes níveis devem possuir depende muito da aplicação. Caches maiores oferecem melhor desempenho até certo ponto. A partir de um determinado tamanho, seu desempenho começa a decair. Memórias cache grandes também podem não contribuir para o desempenho de aplicações que usam os dados apenas uma vez, tal como a decodificação de vídeo. Neste caso, é desejável que a cache trabalhe com modos de operações diferentes. Ou seja, ora para acessos do tipo *streaming* e hora para acessos com comportamento normal. Para os acessos do tipo *streaming*, as caches podem dar lugar às *memórias locais*, que são tipos não gerenciados de cache onde não há *tags*, o tempo de acesso é determinístico e o controle é feito por software. A arquitetura Microsoft Xenon [8] implementa este tipo de comportamento.

Normalmente, o tamanho das caches está associado ao tamanho do circuito do qual ela faz parte e ao consumo de energia a ela permitido. Esta tem sido uma tendência na elaboração de arquiteturas de propósito geral que deve lidar com grande reuso de dados, como é o caso dos processadores Intel Core i7 [9]. O número de níveis de cache tem crescido a medida que os elementos de processamento aumentam tanto em quantidade quanto em velocidade. A principal consideração para determiná-lo é a distância, em ciclos, entre os núcleos e a memória principal. Quanto maior ela for, mais níveis de cache são desejáveis [1]. O primeiro nível de cache é normalmente pequeno, rápido e particular de cada elemento de processamento. Os níveis subsequentes vão crescendo em tamanho e latência e passam a ser compartilhados por diferentes núcleos. Em arquiteturas multicore embarcadas, pelo fato da distância entre ciclos ser normalmente menor, um nível de cache é muitas vezes suficiente, o que também contribui na redução do tamanho e do consumo de energia da arquitetura.

2.4.3 Interconexão e Coerência

A interconexão entre os diferentes níveis na hierarquia da memória é responsável pela comunicação geral entre os elementos de processamento e também pela coerência (se presente) entre as caches. Há muitas formas de se fazer esta interconexão. Podemos fazê-la por meio de um barramento, de um anél, de um *crossbar* ou mesmo através do conceito de *network-on-chip* (NoC). Cada um dos tipos de interconexão tem suas vantagens e desvantagens. O barramento, por exemplo, é simples de implementar, mas quando empregado entre um grande número de núcleos, rapidamente se torna proibitivo tanto em termos de largura de banda quanto de latência. O NoC, por outro lado, tem boa escalabilidade, mas é mais difícil de conceber.

Os mecanismos para manter a coerência dos dados são fundamentais na definição do modelo de programação que deve ser usado na arquitetura. Através deles, uma única imagem da memória é mantida automaticamente para o acesso de todos os processadores. Portanto, esses mecanismos são essenciais entre aplicações que compartilham informação. Eles são muito comuns em processadores de propósito geral, tal como o ARM Cortex A9 [6]. Dois modelos de coerência podem ser implementados: o baseado em *broadcast* e o baseado em *diretório*.

O modelo de coerência baseado em *broadcast* é simples. Ele a obtém através do aviso a todos os elementos de processamento de que um determinado núcleo precisa fazer uso de uma informação de uso global. Esses avisos são tratados um por vez e os conflitos são resolvidos através de sua sequencialização. De uma forma geral, essa abordagem é aplicável para um pequeno número de processadores, como é o caso do Intel Core i7 [9].

A coerência baseada diretório, por outro lado, permite que um grande número de elementos de processamento sejam in-

Tabela 1: Resumo dos prós e contras considerados na concepção de um sistema de memória.

MEMÓRIA NO CHIP	PRÓS	CONTRAS
Caches	Provê acesso à memória principal com baixa latência, pode ser facilmente configurado em múltiplos níveis	Não é possível garantir respostas em tempo real, uma parte do chip é dedicada às TAGS
Armaz. local	Pode armazenar mais dados por área que a cache, é possível garantir respostas em tempo real	Deve ser controlado por software
COERÊNCIA	PRÓS	CONTRAS
Sim	Provê o comportilhamento de dados entre os processadores, suporta todos os modelos de programação	Difíl de implementar
Não	Fácil de implementar	Restringe os modelos de programação
INTERCONEXÃO	PRÓS	CONTRAS
Barramento	Fácil de implementar, todos os processadores têm a mesma latência ao comunicar-se com os outros e com a memória	Baixa escalabilidade, suporte um pequeno número de processadores
Anel	Suporta um número maior de processadores em comparação ao barramento	Latências não-uniformes, requer lógica de roteamento
NoC	Suporta muitos processadores, e as latências não-uniformes são menos variantes do que no modelo Anel	Requer lógica de roteamento sofisticado
Crossbar	É o mais escalável, suporte um número muito grande de núcleos, latências uniformes	Requer uma lógica de arbitragem sofisticada e requer uma grande área no chip

terligados pelo fato dela possibilitar que múltiplas ações de coerência sejam tomadas simultaneamente. Esse diretório é normalmente distribuído em nós e possui informações sobre quais caches contêm quais endereços de memória. Cada endereço é atribuído a um nó do diretório. Quando um acesso é requerido, o processador consulta o nó em que o endereço está armazenado para saber quais processadores estão fazendo uso daquela informação. Dessa forma, esse processador pode negociar com os demais para obter as permissões de acesso que ele deseja. A coerência baseado em diretório é mais apropriada para modelos de consistência fracos e sistema com muitos núcleos, tal como o Tilera TILE 64 [10].

É comum encontrar arquiteturas multicore que omitem os mecanismos de coerência entre as caches para reduzir sua complexidade. Muitos processadores multicore atuais não possuem esse mecanismos, incluindo o IBM Cell [7]. Essa ausência implica que a coerência dos dados deve ser garantida pelo software durante sua execução. Isto limita o modelo de programação a variantes do modelo de passagem de mensagens. Para domínios de aplicação que compartilham pouca informação, essa pode ser uma boa opção. As decisões a serem tomadas ao definir o sistema de memória estão resumidas na Tabela 1.

3. MODELOS COMERCIAIS

Nos dias de hoje, há uma grande quantidade de arquiteturas multicore sendo oferecidas no mercado e direcionadas a todos os nichos, desde o segmento de sistemas embarcados, passando pelo de desktops de propósito geral até o segmento de servidores. Conforme já mencionado, isto se deve à necessidade de aumentar o desempenho sem aumentar muito o consumo de energia. Para oferecer um panorama geral dos modelos oferecidos, nós fizemos um recorte do recente trabalho de Blake et al. [1], onde alguns dos principais modelos são destacados. As quatro primeiras linhas da Figura 2 representam uma seleção feita pelos autores de multicores de propósito geral. Todos esses modelos empregam um número limitado de processadores idênticos com caches grandes. Essas arquiteturas são direcionadas aos mercados de desktops e de servidores onde o baixo consumo de energia não é um fator prioritário. As demais linhas da Figura 2 também apresentam arquiteturas multicore, porém destinadas ao mercado de computação móvel e embarcada. Elas também apresentam núcleos de propósito geral bastante apropriados para aplicações dominadas por estruturas de controle. Nesse contexto, o consumo de energia é muito importante pelo fato de que eles serão alimentados por baterias.

Na Figura 3, as arquiteturas são mais especializadas, e dirigidas a computação de alto desempenho. Neste sentido, elas empregam um número expressivo de núcleos. Como podemos observar, nas arquiteturas AMR R700 e NVIDIA G200, esse número é na casa das centenas. A arquitetura IBM Cell é heterogênea, com um número modesto de núcleos altamente especializados. Normalmente, essas soluções consomem bastante energia, variando entre 100W e 180W. Na Figura 4, podemos observar arquiteturas específicas para determinados domínios de aplicação. A maioria dessas aplicações são dominadas pelo processamento de dados, como é o caso das comunicações *wireless* e dos *codecs* de áudio e vídeo, onde paralelismos intrínsecos são explorados. As características das redes de interconexão também costumam ser ajustadas diante das especificidades da aplicação. A combinação desses fatores resulta em arquiteturas com altas taxas de operações por ciclo e sem um consumo excessivo de energia. A seguir, abordaremos três desses modelos com maiores detalhes.

3.1 Silicon Hive HiveFlex CSP2x00 Series - Processamento de Sinais Digitais

O HiveFlex CSP2x00 Series [11] é uma arquitetura com núcleos *soft* poderosos e que operam na faixa de um quarto de watt. Seu diagrama de blocos pode ser visto na Figura 5.

Para obter baixo consumo, uma coleção heterogênea, e apropriada para o desempenho almejado, de núcleos é empregada. A arquitetura consiste de um núcleo de controle, destinado ao processamento de propósito geral. Este núcleo baseia-se no conceito VLIW e pode fazer despachos simultâneos de duas instruções. Quando submetido a operações com dados, ele repassa o processamento aos chamados "núcleos complexos". Esses núcleos complexos também são VLIW, fazem até cinco despachos simultâneos e contam com ALUs específicas conectadas a uma grande memória RAM local. Os núcleos complexos não suportam desvios, ou seja, eles devem ser alimentados com trechos lineares de código pelo núcleo de controle. Essa característica faz com que os núcleos complexos sejam simplificados, o que permite econo-

Figura 2: Arquiteturas multicore de propósito geral para servidores, desktops e aplicações embarcadas/móveis. Tabela e referências de [1].

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMD PHENOM [11], [15]	X86	THREE-WAY OUT-OF-ORDER SUPERSCALAR, 128-B SIMD	FOUR	64 KB IL1 AND DL1/ CORE, 256 KB L2/CORE, 2-6 MB L3	DIRECTORY	POINT TO POINT	PROCESSOR	140 W	2.5 GHZ–3.0 GHZ	12–48 OPS/ CLOCK
INTEL CORE I7 [2], [5]	X86	FOUR-WAY OUT-OF-ORDER, TWO-WAY SMT, 128-B SIMD	TWO TO EIGHT	32 KB IL1 AND DL1/ CORE, 256 KB L2/CORE, 8 MB L3	BROADCAST	POINT TO POINT	PROCESSOR	130 W	2.66 GHZ–3.33 GHZ	8–128 OPS/ CLOCK
SUN NIAGARA T2 [16], [17]	SPARC	TWO-WAY IN-ORDER, EIGHT-WAY SMT	EIGHT	16 KB IL1 AND 8 KB DL1/ CORE, 4 MB L2	DIRECTORY	CROSSBAR	TOTAL STORE ORDERING	60–123 W	900 MHZ–1.4 GHZ	16 OPS/CLOCK
INTEL ATOM [18], [5]	X86	TWO-WAY IN-ORDER, TWO-WAY SMT, 128-B SIMD	ONE TO TWO	32 KB IL1 AND DL1/ CORE, 512 KB L2/CORE	BROADCAST	BUS	PROCESSOR	2–8 W	800 MHZ–1.6 GHZ	2–16 OPS/ CLOCK
ARM CORTEX-A9‡ [6]	ARM	THREE-WAY OUT-OF-ORDER	ONE TO FOUR	(16,32,64) KB IL1 AND DL1/CORE, UP TO 2 MB L2	BROADCAST	BUS	WEAKLY ORDERED	1 W (NO CACHE)	N/A	3–12 OPS/ CLOCK
XMOS XS1-G4 [19]	XCORE	ONE-WAY IN-ORDER, EIGHT-WAY SMT	FOUR	64 KB LCL STORE/CORE	NONE	CROSSBAR	NONE	0.2 W	400 MHZ	4 OPS/CLOCK

‡Numbers are estimates because design is offered only as a customizable soft core.

Figura 3: Arquiteturas multicore de alto desempenho. Para obtenção das referência dos modelos, consulte [1].

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMD RADEON R700 [20]	N/A	FIVE-WAY VLIW	160 CORES, 16 CORES PER SIMD BLOCK, TEN BLOCKS	16 KB LCL STORE/SIMD BLOCK	NONE	N/A	NONE	150 W	750 MHZ	800-1,600 OPS/CLOCK
NVIDIA G200 [8], [21]	N/A	ONE-WAY IN-ORDER	240, EIGHT CORES PER SIMD UNIT, 30 SIMD UNITS	16 KB LCL STORE/EIGHT CORES	NONE	N/A	NONE	183 W	1.2 GHZ	240-720 OPS/ CLOCK
INTEL LARRABEE† [22]	X86	TWO-WAY IN-ORDER, 4-WAY SMT, 512-B SIMD	UP TO 48†	32 KB IL1 AND 32 KB DL1/ CORE, 4 MB L2	BROADCAST	BIDIRECTIONAL RING	PROCESSOR	N/A	N/A	96-1,536 OPS/ CLOCK
IBM CELL [9], [23]	POWER	TWO-WAY IN-ORDER, TWO-WAY SMT PPU, 2-WAY IN-ORDER 128-B SIMD SPU	1 PPU, EIGHT SPUs	PPU: 32 KB IL1 AND 32 KB DL1, 512 KB L2; SPU: 256 KB LCL STORE	NONE	BIDIRECTIONAL RING	WEAK (PPU), NONE (SPU)	100 W	3.2 GHZ	72 OPS/CLOCK
MICROSOFT XENON [10]	POWER	TWO-WAY IN-ORDER, TWO-WAY SMT, 128-B SIMD	THREE	32 KB IL1 AND 32 KB DL1/ CORE, 1 MB L2	BROADCAST	CROSSBAR	WEAKLY ORDERED	60 W	3.2 GHZ	6-24 OPS/ CLOCK

†All values are estimates as processor is not yet in production.

Figura 4: Arquiteturas multicore para processamento de sinais e específicos. As referências dos modelos são as de [1].

	ISA	MICROARCHITECTURE	NUMBER OF CORES	CACHE	COHERENCE	INTERCONNECT	CONSISTENCY MODEL	MAX. POWER	FREQUENCY	OPS/CLOCK
AMBRIC AM2045 [24], [25]	N/A	ONE-WAY IN-ORDER SR, THREE-WAY IN-ORDER SRD	168 SR, 168 SRD	21 KB LCL STORE/EIGHT CORES	NONE	NoC	NONE	6–16 W	350 MHZ	672 OPS/ CLOCK
ELEMENT CXI ECA–64 [26], [3]	N/A	ONE-WAY IN-ORDER, DATAFLOW CONNECTIONS TO 15 RECONFIGURABLE ALUs	FOUR CLUSTERS OF ONE CORE+ALUs	32 KB OF LCL STORE/ CLUSTER	NONE	HIERARCHIAL NoC	NONE	1 W	200 MHZ	64 OPS/CLOCK (16-B)
TI TMS320-DM6467 [14]	ARM, C64X	ONE ARM9 ONE-WAY IN-ORDER, ONE C64X EIGHT-WAY VLIW	TWO	ARM9: 16 KB IL1, 8 KB DL1; C64X: 32 KB IL1 AND DL1, 128 KB L2	NONE	BUS	WEAKLY ORDERED	3–5 W	ARM: 297–364 MHZ, C64X: 594–729 MHZ	1–9 OPS/ CLOCK
TI OMAP 4430 [12]	ARM, C64X	TWO ARM THREE-WAY OUT-OF-ORDER, ONE C64X EIGHT-WAY VLIW	THREE	N/A	BROADCAST AMONG ARM CORES	BUS	WEAKLY ORDERED	1 W	1GHZ	6–14O PS/ CLOCK
TILERA TILE64 [27], [28]	N/A	THREE-WAY VLIW	36–64	8 KB IL1 AND DL1/CORE, 64 KB L2/CORE	DIRECTORY	NoC	N/A	15–22 W	500–866 MHZ	108–192 OPS/ CLOCK
HIVEFLEX CSP2X00† [29]	N/A	TWO-WAY VLIW CONTROL CORE, FIVE-WAY VLIW COMPLEX CORE	TWO TO FIVE	2X CONFIGURABLE L1 FOR BASE CORE, LCL STORE FOR COMPLEX CORE	NONE	BUS	NONE	0.25 W	200 MHZ	2–22 OPS/ CLOCK

†Numbers are estimates because design is offered only as a customizable soft core.

Figura 5: Concepção em blocos da arquitetura HiveFlex CSP2x00 Series [1].

mia tanto em área física quanto em consumo de energia. Além disso, o CSP2x00 possui uma hierarquia de memória bastante simples. A coerência e a consistência dos dados devem ser controladas pelo software em execução no núcleo de controle. O barramento que provê a interconexão é usado basicamente para transferir comandos entre o núcleo de controle e os núcleos complexos, que por sua vez comunicam-se entrei si através através da memória RAM local. Todas essas características fazem com que essa arquitetura consuma muito pouca energia, porém, que obtenha alto desempenho. No entanto, criar softwares eficientes para ela não deixa de ser um desafio.

3.2 ARM Cortex A9 - Propósito geral móvel

A arquitetura ARM Cortex A9 [6] foi concebida para computação móvel de propósito geral e pode ser customizada antes de ser produzida. Na Figura 6 podemos observar seu diagrama de blocos. As configurações mais usuais consomem pouco energia, em torno de 1W ou menos. Seu projeto foi concebido para atender desde *smart phones* até *netbooks* cheio de recursos. Isso implica em uma arquitetura que consiga processar bem aplicações dominadas pelo processamento de estruturas de controle. O processamento nos núcleos do A9 são fora de ordem podendo ser despachadas até três instruções simultaneamente, garantindo bom desempenho de propósito geral. A interconexão do sistema de memória é feita através de um barramento e há coerência entre os elementos de memória. Pelo fato do número de processadores ser pequeno, a coerência é baseada em *broadcast*. As caches são relativamente grandes para um processador voltado ao mercado de aplicações móveis ou embarcadas. Isso é necessário para proporcionar clocks mais rápidos e explorar bem processamentos em *threads* únicas. Em função dessas características, aplicações onde o processamento de dados é dominante não seriam muito bem exploradas por essa arquitetura.

3.3 Intel Core i7 - Propósito geral

O Intel Core i7 é um processador de propósito geral em todos os aspectos. Sua concepção tem por objetivo fazer tudo bem feito. Isto é obtido ao preço de um grande consumo de energia, podendo chegar aos 140W. Ele pode ser implementado com até oito núcleos, cada qual despachando fora de

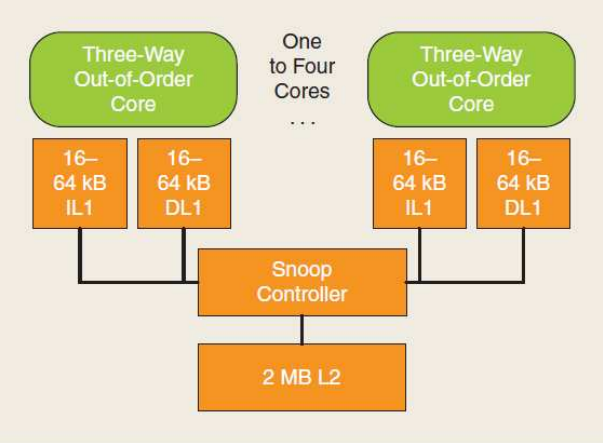

Figura 6: Diagrama de blocos da arquitetura ARM Cortex-A9 [1].

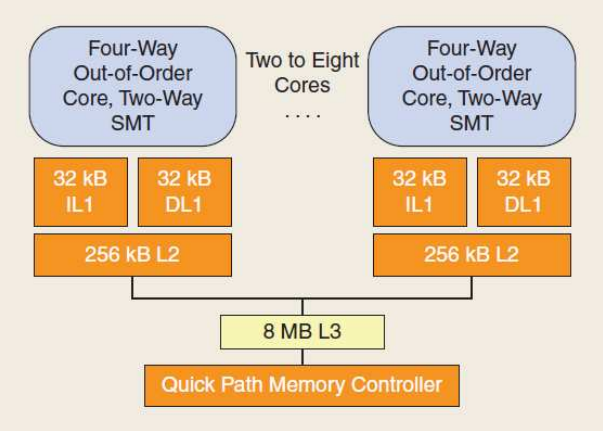

Figura 7: Diagrama de blocos do Intel Core i7 [1].

ordem até quatro instruções em simultâneo. Além disso, ele faz uso do conceito de *symmetric multithreading*, conforme pode ser observado na Figura 7. Esses núcleos contêm uma série de aprimoramentos complexos para que possam o extrair o máximo de paralelismo possível na execução de uma única *thread*. Eles também possuem uma unidade SIMD de 128-B para obter as vantagens de eventuais paralelismos sobre os dados. Para continuar compatível com a maioria dos processsadores Intel, seu conjunto de instruções é baseado no CISC x86. A concepção do Intel Core i7 permite que ele faça muitas coisas bem, mas arquiteturas especializadas em determinados domínios de aplicação podem obter desempenho equivalente ou melhor consumindo menos energia. O sistema de memória é típico de arquiteturas multicore de propósito geral com poucos núcleos. As caches são grandes e há total coerência entre elas. A coerência é do tipo *broadcast*, que é suficiente para o número de núcleos projetados. Esse conjunto de características reunidas dá origem a um chip que é bom para uma grande variedade de aplicações, desde que o consumo de energia não seja um problema.

4. CONCLUSÕES

Conforme pudemos observar neste trabalho, a arquitetura multicore é hoje um paradigma de projeto de processadores fundamental. É através dela que os projetistas das grandes empresas pretendem manter a lei de Moore sem esbarrar nas limitações físicas impostas pela abordagem de núcleo único. Muitas são as possibilidades ao implementar arquiteturas multicore, sendo determinante saber as características das aplicações para o qual ela é projetada, tais como o tipo de processamento demandado e as restrições no consumo de energia. Também vimos que o sistema de memória é parte fundamental do desenho de um processador multicore. É ele que irá ditar o modelo de programação e o número de núcleos comportados pela arquitetura. Essa talvez seja a frente que mereça mais atenção por parte dos projetistas para satisfazer as demandas crescentes pelo aumento de desempenho. À medida que mais aplicações conseguirem extrair as vantagens do processamento paralelo, novas configurações multicore deverão ser propostas para atendê-las.

5. REFERÊNCIAS

[1] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *Signal Processing Magazine, IEEE*, vol. 26, pp. 26–37, October 2009.

[2] J. E. Savage and M. Zubair, "A unified model for multicore architectures," in *IFMT '08*, (New York, NY, USA), pp. 1–12, ACM, 2008.

[3] "International technology roadmap for semiconductors-system drivers," tech. rep., ITRS. disponível em 11/06/2010 http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_SystemDrivers.pdf.

[4] "Intel 64 and ia-32 architectures software developer's manual," tech. rep., Intel, November 2008.

[5] "Nvidia's 1.4 billion transistor gpu: Gt200 arrives as the geforce gtx 280 & 260," tech. rep., A. L. Shimpi and D. Wilson, 2008. disponível em 11/06/2010 http://www.anandtech.com/video/showdoc.aspx?i=3334.

[6] "The arm cortex-a9 processors," tech. rep., ARM Ltd., September 2007. disponível em 11/06/2010 http://www.arm.com/pdfs/ARMCortexA-9Processors.pdf.

[7] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in cell's multicore architecture," *IEEE Micro*, vol. 26, no. 2, pp. 10–24, 2006.

[8] J. Andrews and N. Baker, "Xbox 360 system architecture," *IEEE Micro*, vol. 26, no. 2, pp. 25–37, 2006.

[9] "Intel core i7-940 processor," tech. rep., Intel Corp., 2009. disponível em 11/06/2010 http://ark.intel.com/cpu.aspx?groupId=37148.

[10] "Tilepro64 processor," tech. rep., Tilera Corp., 2008. disponível em 11/06/2010 http://www.tilera.com/pdf/ProductBrief_TILEPro-64_Web_v2.pdf.

[11] "Hiveflex csp2000 series: Programmable ofdm communication signal processor," tech. rep., Silicon Hive, 2007. disponível em 11/06/2010 http://www.siliconhive.com/Flex/Site/Page.aspx?PageID=8881.