# O multiprocessador Cell: uma arquitetura para explorar o paralelismo

Taísa Cristina Costa dos Santos

Instituto de Computação – Unicamp
RA 036065
taisa.santos@gmail.com

## Resumo

Conforme cresce a demanda por *chips* cada vez menores e que consumam menos energia, e, em contrapartida, os métodos baseados na microarquitetura tradicional para a entrega de alto desempenho possuem o ônus do preço elevado e do alto consumo de energia, os *chips* multiprocessados (CMPs) apresentam-se como um caminho promissor aos projetistas de sistemas para a obtenção de melhor desempenho.

Explorar o paralelismo em um sistema multiprocessado é a chave para o ganho de desempenho do sistema como um todo, incluindo um uso mais eficiente da largura de banda da memória.

Este artigo provê informações gerais sobre o multiprocessador Cell, uma extensão revolucionária da arquitetura e organização de microprocessadores convencionais. Descreve, ainda, como este explora o paralelismo em múltiplos níveis: nível de dados, nível de instrução, nível de thread, nível de memória e ainda em nível de computação e transferência de dados.

## Palavras-chave

Multiprocessamento em chip heterogêneo, DMA, memória compartilhada, Cell Broadband Engine, paralelismo.

## 1. Introdução

Quando IBM, Sony e Toshiba lançaram o projeto Cell em 2000, o objetivo deste era prover desempenho uma ordem de magnitude melhor que os *desktops* produzidos em 2005 [1]. Para tal, os projetistas precisariam otimizar o desempenho sem comprometer área, consumo de energia, volume e custo, e arquiteturas *single-core* apresentavam retorno cada vez menor pelo investimento.

Posto isto, a estratégia adotada foi explorar uma inovação arquitetural direcionada no aumento da eficiência do sistema, de modo a entregar o maior desempenho por área investida. O modelo ainda deveria explorar o paralelismo de aplicações suportando, também, modelos de aplicação já estabelecidos, além de assegurar boa programabilidade.

O resultado foi a *Cell Broadband Engine Architecture* (CBEA), baseada num *chip* multiprocessado heterogêneo. Sua primeira implementação foi o *Cell Broadband Engine* (Cell BE ou somente Cell), que suporta execução escalar e *single-instruction, multiple-data* (SIMD) igualmente bem, e provê execução *multithreaded* de alto desempenho para todas as aplicações. A seção a seguir contém maiores detalhes sobre esta arquitetura.

## 2. Arquitetura do Cell BE

O processador Cell é a primeira implementação da CBEA, que é uma extensão totalmente compatível da arquitetura PowerPC 64-bits. Seu alvo inicial foi o jogo Playstation 3, porém o Cell se encaixa muito bem na execução de outras aplicações como visualização, processamento de imagens e sinais e vários *workloads* científicos e técnicos.

A figura 1 mostra a arquitetura do sistema Cell. Este é um processador *multicore* heterogêneo, capacitado para o processamento massivo de operações de ponto-flutuante otimizado para *workloads* de computação intensiva e aplicações de mídia de grande largura de banda. Consiste de um *Power Processor Element* (PPE) de 64 bits, oito coprocessadores especializados chamados *Synergistic Processor Elements* (SPEs), um controlador de memória veloz e uma interface de barramento de alta largura de banda, tudo integrado *on-chip*. O PPE e os SPEs se comunicam através de um barramento interno de alta velocidade chamado *Element Interconnect Bus* (EIB) [2].

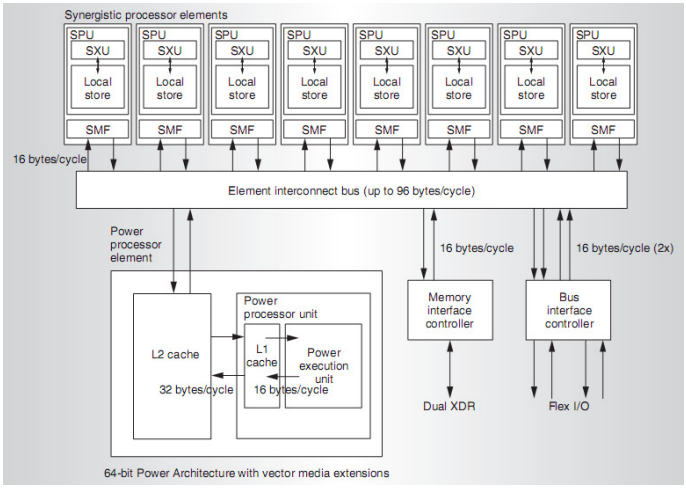


**Figura 1: Arquitetura do sistema Cell.**

### 2.1. *Power Processor Element* (PPE)

O PPE é o principal núcleo de processamento do Cell, provê funções de controle comuns, executa o sistema operacional

(32 ou 64 bits) e fornece controle às aplicações. Ele entrega serviços gerais de sistema, como gerenciamento de memória virtual, tratamento de exceções, escalonamento de *threads* e outros serviços do sistema operacional.

O PPE é um processador PowerPC de 64 bits tradicional, com uma unidade de extensão multimídia vetorial (VMX) de 128 bits, duas *caches* L1 de 32KB cada, uma de dados e outra de instruções, e uma *cache* L2 de 512KB. Possui despacho duplo, execução em ordem [2] e *two-way simultaneous multithreading* (o que permite que instruções de duas *threads* possam estar em qualquer estágio do *pipeline* num dado momento, dando ao *software* a impressão de estar efetivamente executando em duas unidades de processamento independentes).

O processador é composto por três unidades (figura 2). A unidade de instrução (IU) é responsável pelo *fetch*, *decode*, *branch*, despacho e conclusão de instruções. A unidade de execução de ponto-fixo (XU) é responsável por todas as instruções de ponto-fixo e pelas instruções de *load*/*store*. Por fim, a unidade escalar-vetorial (VSU) é responsável por todas as instruções de ponto-flutuante.

A IU faz o *fetch* de quatro instruções por ciclo por *thread* para um *buffer* de instruções, e despacha as instruções a partir desse buffer. Depois da decodificação e da verificação de dependências, as instrução são despachadas em duplas para uma unidade de execução. Uma tabela de histórico de *branches* (BHT) com um histórico global por *threads* é usada para prevê os resultados dos *branches*. A UI pode despachar até duas instruções por ciclo. Todas as combinações de duplo despacho são possíveis a menos que as duas sejam destinadas à mesma unidade funcional ou que as duas estejam no seguinte conjunto de instruções aritméticas: vetor simples, vetor complexo, ponto-flutuante vetorial e ponto-flutuante escalar. Além disso, uma fila de despacho na VSU desacopla os *pipelines* vetoriais e de ponto-flutuante dos outros, permitindo que estes tipos de instrução sejam despachados fora de ordem em relação aos demais.

## 2.2. *Synergistic Processor Element* (SPE)

Os SPEs entregam a maior parte do desempenho do sistema Cell. São núcleos aceleradores que implementam uma inovadora arquitetura de computação predominantemente paralela em nível de dados, baseada em computação RISC SIMD e explícito gerenciamento de transferência de dados.

Consistem de uma unidade de processamento sinérgico (SPU) e de um controlador de fluxo de memória sinérgico (na figura 1, SMF, porém é mais conhecido por MFC, *memory flow controller*), que juntos fornecem a cada *thread* de SPE a capacidade de executar sequências de computação e transferência independentes.

A SPU é um processador ao estilo RISC com conjunto de instruções e microarquitetura projetados para a computação intensiva e o *streaming* de dados de alto desempenho. Inclui uma memória de armazenamento local de 256KB para manter instruções e dados do programa da SPU. Esta não pode acessar diretamente a memória principal, mas pode lançar comandos DMA para seu MFC trazer dados para a memória local ou para escrever resultados na memória principal. A SPU pode continuar a execução do programa enquanto o MFC realiza as transações DMA de modo independente. Não existem estruturas de *hardware* para a predição do carregamento de dados no gerenciamento da memória local, que, por sua vez, deve ser feito via *software* [2].

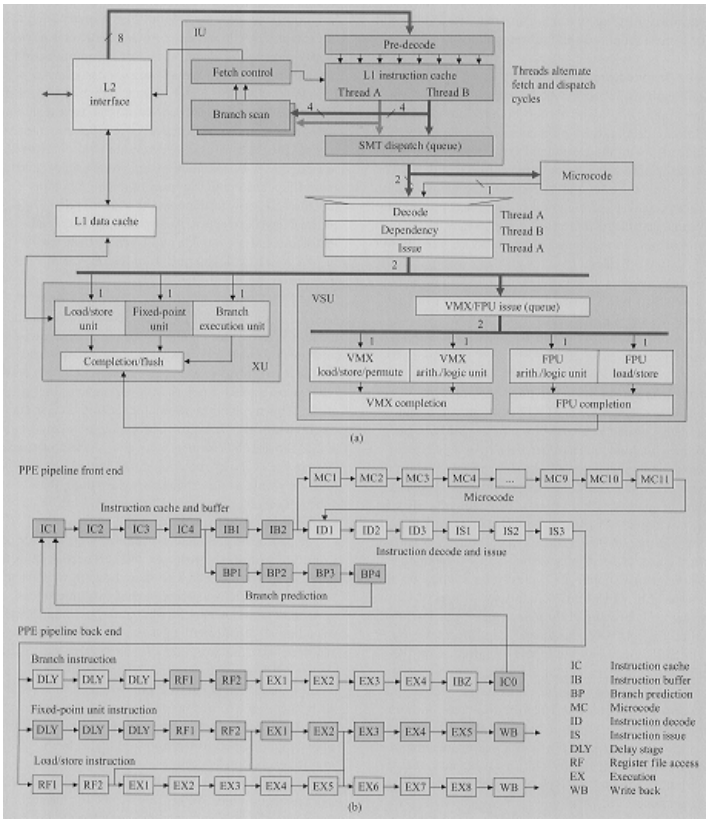


**Figura 2: Diagrama esquemático do PPE: (a) unidades principais e (b) diagrama do *pipeline*.**

O MFC realiza operações DMA para transferir dados entre o armazenamento local e o sistema de memória. Operações DMA especificam locais de memória usando endereços virtuais totalmente compatíveis com PowerPC. Podem transferir dados entre o armazenamento local e qualquer outro recurso conectado via a interconexão *on-chip* do Cell (memória principal, memória local de outro SPE ou um dispositivo de I/O). Transferências paralelas entre SPEs são mantidas a uma taxa de 16 bytes por *clock* de SPE, enquanto que a largura de banda da memória agregada é de 25.6GB/s para o processador inteiro.

Cada SPU tem 128 registradores SIMD de 128 bits cada, o que facilita um escalonamento de instrução altamente eficiente por parte do compilador e possibilita o uso de importantes técnicas de otimização, como *loop unrolling*. Todas as instruções da SPU são inerentemente SIMD, e o *pipeline* pode executar em quatro granularidades: 16 inteiros de 8 bits, 8 inteiros de 16 bits, 4 inteiros ou pontos-flutuantes de precisão simples de 32 bits ou 2 pontos-flutuantes de precisão dupla de 64 bits.

A SPU é um processador em ordem com dois *pipelines* de instrução (figura 3), denominados par e ímpar. As unidades funcionais de ponto-fixo e ponto-flutuante pertencem ao *pipeline* par, enquanto que todas as demais pertencem ao ímpar. Cada SPU pode completar até duas instruções por ciclo, uma por *pipeline*. Operações de ponto-fixo simples levam 2 ciclos e de ponto-flutuante com precisão simples levam 6 ciclos. Operações SIMD

de ponto-flutuante de precisão dupla são suportadas, porém apresentam taxa de despacho de uma a cada 7 ciclos.

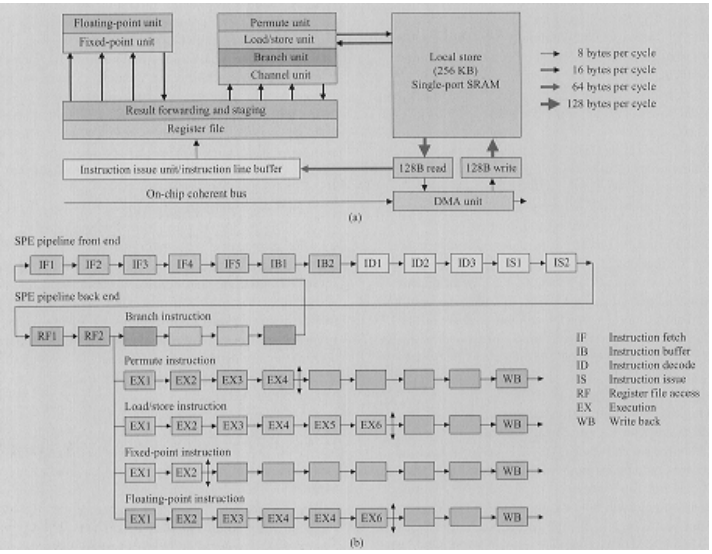


**Figura 3: Diagrama esquemático do SPE: (a) organização e (b) diagrama de *pipeline*.**

## 2.3. Elementos de comunicação

Para aproveitar todo o poder de processamento disponível no processador Cell, o trabalho precisa ser distribuído e coordenado entre o PPE e os SPEs. Os mecanismos de comunicação especializada dos processadores permitem a coleta e a distribuição de dados tão eficientes quanto à coordenação de atividades concorrentes entre os elementos de computação.

O PPE e os SPEs compartilham a arquitetura de tradução de endereço e de memória virtual, provendo suporte à virtualização e ao particionamento dinâmico do sistema. Compartilham também tabelas de páginas e funções de sistema, como apresentação de interrupção. Por fim, compartilham formatos de tipos de dados e semântica de operações, de modo a tornar mais eficiente o compartilhamento de dados entre eles.

Os SPEs possuem, ainda, um conjunto de operações atômicas que interoperam com as operações atômicas do PPE, de modo a construir *locks* e outros mecanismos de sincronização que funcionam entre SPEs e PPE. Além disso, o sistema Cell permite acesso por memória mapeada a quase todos os recursos do SPE, incluindo o espaço de armazenamento local inteiro, o que fornece um mecanismo conveniente e consistente para necessidades especiais de comunicação não atendidas por outras técnicas.

### 2.3.1. *Element Interconnect Bus (EIB)*

O EIB é o coração da arquitetura de comunicação do processador Cell, possibilitando a comunicação entre o PPE, os SPEs, o sistema de memória principal e I/O externo. Possui diferentes caminhos de comunicação para comandos e dados. Cada elemento do barramento é conectado através de um *link* ponto-a-ponto para o concentrador de endereço, que recebe e pede comandos aos elementos do barramento, realiza *broadcast* de comandos (para *snooping*) e então agrega e propaga respostas de comandos. A resposta do comando é um sinal para os elementos apropriados para iniciar a transferência de dados [3].

A rede de dados do EIB consiste em quatro anéis de 16 bytes (figura 4), dois executando no sentido horário e os outros dois no sentido anti-horário. Cada anel permite potencialmente três transferências de dados concorrentes, desde que seus caminhos não se sobreponham.

Para iniciar uma transferência, os elementos devem pedir permissão de acesso ao barramento. O árbitro do barramento processa os pedidos e decide que anel deve tratar cada pedido. O árbitro sempre escolhe um dos dois anéis de transferência mais curta, assegurando que o dado nunca viaja mais que meio-anel para chegar ao seu destino. O árbitro também escalona as transferências de modo que elas não interfiram outras já em andamento naquele anel. Para minimizar *stalls* em leituras, o árbitro dá prioridade aos pedidos do controlador da memória. Todos os outros pedidos são tratados igualmente, seguindo o modelo *round-robin* de escalonamento.

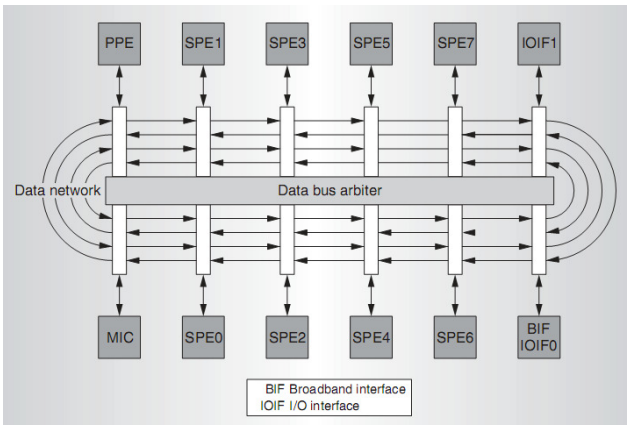


**Figura 4: O EIB e seus anéis.**

O EIB opera à metade da velocidade do *clock* do processador. Cada unidade do EIB pode receber ou enviar simultaneamente 16 bytes de dados a cada ciclo do barramento. A largura de banda máxima do EIB é limitada pela taxa com que endereços são observados (*snooped*) por todas as unidades do sistema, que é um endereço por ciclo de barramento. Cada pedido de endereço observado pode transferir até 128 bytes de dados, o que num processador Cell de 3.2GHz representaria 128 bytes × 1.6GHz = 204.8 Gbytes/s.

A interface de I/O *on-chip* permite que dois processadores Cell se conectem utilizando um protocolo coerente chamado *broadband interface* (BIF), que efetivamente estende a rede do multiprocessador para conectar ambos os PPEs e também todos os 16 SPEs em uma única e coerente rede.

A largura de banda de dados real do EIB depende de vários fatores: a localização relativa da origem e do destino, a chance de uma nova transferência interferir numa já existente, o número de *chips* Cell no sistema, se as transferências são da/para a memória ou entre áreas de armazenamento local de SPEs e a eficiência do árbitro de dados do barramento.

Em casos de largura de banda reduzida, podem ocorrer problemas, tais como todas as transferências ocorrerem numa mesma direção, deixando os outros anéis ociosos, um grande número de transferências de linhas parciais de *cache*, mais de um requerente acessando um mesmo destino ao mesmo tempo etc.

### 2.3.2. *Memory Flow Controller (MFC)*

Cada SPE contém um MFC que o conecta ao EIB e gerencia vários caminhos de comunicação entre o SPE e os outros elementos do Cell, executando na mesma frequência do EIB. A SPU interage com o MFC através de canais unidirecionais de comunicação que agem como filas FIFO. Isto significa que cada canal é definido como *read-only* ou *write-only* do ponto de vista da SPU. Além disso, alguns canais são definidos como bloqueantes, ou seja, se o canal for *read-only* e estiver vazio ou se ele for *write-only* e estiver cheio, a SPU ficará bloqueada até a operação terminar.

O MFC aceita e processa comandos DMA despachados pela SPU ou pelo PPE através da interface dos canais da SPU ou através de registradores de I/O mapeados em memória (ou registradores MMIO). Os comandos DMA se enfileiram no MFC, enquanto que a SPU ou o PPE podem prosseguir com sua execução em paralelo à transferência de dados. Esta execução autônoma dos comandos DMA pelo MFC permite o escalonamento conveniente das transferências de dados para esconder a latência da memória.

O MFC suporta transferências naturalmente alinhadas de 1, 2, 4 ou 8 bytes, ou um múltiplo de 16 bytes até um máximo de 16KB. Um comando de lista DMA pode pedir uma lista de até 2048 transferências DMA através deste único comando, porém apenas a SPU associada ao MFC pode despachar comandos de lista. A lista DMA é um *array* de endereços de origem/destino e comprimentos na memória local da SPU. Quando uma SPU despacha um comando de lista DMA, ela especifica o endereço e o comprimento da lista dentro do seu armazenamento local. O pico de desempenho em transferências ocorre quando ambos os endereços efetivos e da memória local são de 128 bytes alinhados, e o tamanho da transferência é um múltiplo par de 128 bytes.

Além das transferências DMA, os recursos de um SPE podem ser acessados diretamente, uma vez que todo o armazenamento local de um SPE pode ser mapeado no efetivo espaço de endereçamento. Isso permite que o PPE acesse recursos do SPE com operações simples de *load* e *store*, apesar de tal prática ser de longe menos eficiente que transferência DMA. Este tipo de acesso não é sincronizado com a execução da SPU, e portanto os programadores devem assegurar que o programa executado pela SPU é projetado para permitir acesso não sincronizado aos seus dados, o que pode ser feito, por exemplo, declarando a variável usando *volatile*.

Além da comunicação por DMA e por recursos de I/O mapeados em memória, o MFC tem suporte a operações atômicas (comandos DMA especiais) e a mecanismos de *mailbox* e notificação de sinal. Por meio de atualizações atômicas, a SPU pode participar junto com o PPE e com outras SPUs em protocolos de *locking*, barreiras e outros mecanismos de sincronização. Já a presença de *mailbox* e notificação de sinal permite comunicação um-a-um entre a SPU e o PPE ou outra SPU por meio de canais, que recebem escrita de outros elementos via endereço mapeado em memória.

### 2.3.3. *DMA flow*

O MFC possui uma unidade de gerenciamento de memória (MMU) que lida com a tradução de endereços e verificação de proteção de acesso ao armazenamento principal, usando, para isso, informações de tabelas de páginas e de segmentos definidos na arquitetura PowerPC. Possui um buffer de tradução (TLB) para fazer *caching* de resultados de traduções feitas recentemente.

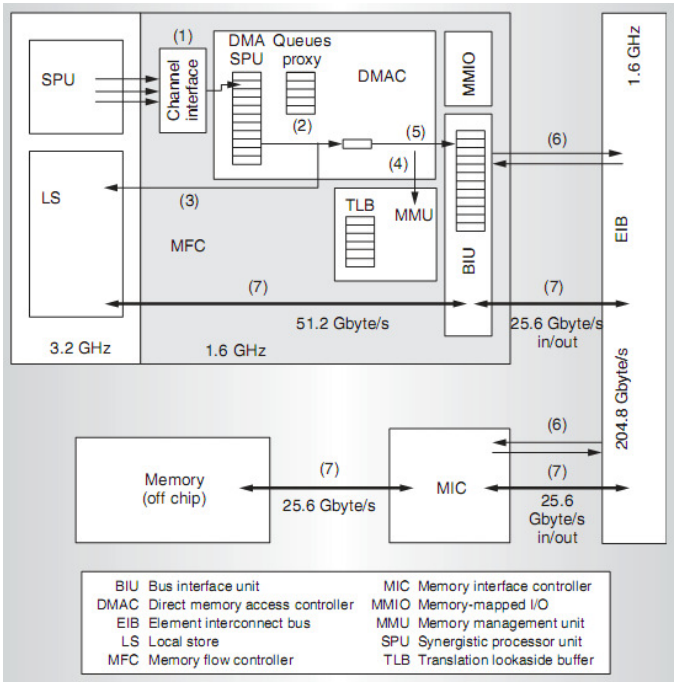


**Figura 5: Fluxo básico de uma transferência DMA.**

Já o controlador DMA do MFC (DMAC) processa comandos DMA enfileirados no MFC. Este, por sua vez, possui duas filas separadas de comandos: (a) fila de comando SPU, contendo os comandos despachados pela sua SPU associada e (b) fila de comandos *proxy*, contendo os comandos despachados pelo PPE ou por outros dispositivos via registradores MMIO.

O fluxo básico de uma transferência iniciada pela SPU para a memória principal (figura 5) consiste em: (1) a SPU utiliza a interface de canal para colocar o comando DMA na fila apropriada; (2) o DMAC seleciona o comando a ser processado seguindo algumas regras – comandos na fila da SPU tem prioridade sobre os da fila de *proxy*, alternância entre transferências do armazenamento local para a memória principal e vice versa, e o comando precisa estar pronto, e não esperando pela resolução de endereços ou *fetches* de elementos de lista; (3) se o comando for uma lista DMA, é feito o *fetch* dos elementos da lista e o DMAC volta a fazer uma seleção; (4) se o comando precisa de tradução de endereço, o DMAC já o coloca na fila da MMU, e quando a tradução já está pronta no TLB, segue-se para o próximo passo; se há um *miss* na TLB, a MMU realiza a tradução consultando a tabela de páginas na memória principal[3], e atualiza a TLB; neste caso, a entrada DMA é atualizada, e o DMAC volta a fazer a seleção; (5) o DMAC "desenrola" o comando e cria um pedido de transferência pelo barramento para o próximo bloco de dados do comando, e o enfileira na fila da unidade de interface do barramento (BIU); (6) A BIU seleciona o pedido da sua fila e o envia ao EIB, que faz o broadcast do comando para todos os elementos do barramento – se a transferência envolver a memória principal, o controlador de

interface de memória (MIC) precisará informar ao EIB que reconheceu o comando, e este por sua vez, avisar a BIU que o comando foi aceito e que a transferência irá começar; (7) a BIU faz as leituras da memória local necessárias para a transferência, o EIB transfere os dados entre a BIU e o MIC e o último transfere dados de/para a memória fora do *chip*.

O "desenrolar" de comandos gera uma sequência de pedidos ao barramento para a execução de um dado comando DMA, e este permanece na fila até que todos os seus pedidos ao barramento se completem e a SPU seja notificada disso. Entretanto o DMAC pode seguir no processamento de outros comandos.

## 3. Explorando o paralelismo da aplicação

Para entregar um aumento significativo no desempenho da aplicação em um ambiente com restrições de energia, o projeto do processador Cell explora o paralelismo em seus diversos níveis:

- paralelismo em nível de dados (DLP) com o suporte a instruções SIMD predominante;
- paralelismo em nível de instrução (ILP) por meio de uma microarquitetura estaticamente escalonada e *power-aware*;
- paralelismo em nível de *thread* (TLP) com um projeto *multicore* e suporte a *hardware multithreading* no PPE;
- paralelismo em nível de computação-transferência (CTP) graças ao uso de *engines* de transferência de dados programáveis;
- paralelismo em nível de memória (MLP) obtido através da sobreposição de transferências de múltiplos pedidos por núcleo e de vários núcleos.

O DLP eleva o montante de computação de modo eficiente a um custo muito baixo em relação à computação escalar. Isto é possível porque a complexidade de controle, que tipicamente é escalável com o número de instruções em execução, permanece inalterada, ou seja, o número de instruções carregadas, decodificadas, analisadas quanto às dependências, o número de acessos a arquivos de registradores e *write-backs* e o número de instruções *committed* permanecem os mesmos.

O compartilhamento de unidades de execução entre computações escalares e SIMD reduz o consumo de energia marginal da computação SIMD devido à eliminação da duplicação do controle e do *datapath*, restando apenas o adicional gerado pela área acrescentada para o suporte a SIMD e à execução das operações propriamente dita. Além disso, implementar múltiplas unidades de execução escalar aumenta a complexidade do processador para o gerenciamento de um número maior de instruções (já que ocorrem mais despachos ao mesmo tempo) e para se descobrir paralelismo em instruções seqüenciais [4].

O compartilhamento de unidades funcionais entre o processamento escalar e SIMD pode ser obtido tanto de modo arquitetural, como nos SPEs que possuem um único banco de registradores para guardar dados escalares e SIMD, como de modo microarquitetural para operações de ponto-flutuante e mídia, como no PPE, que implementa ambos os tipos de computação em uma mesma unidade de processamento. Além da eficiência nos recursos, o compartilhamento arquitetural aumenta a eficácia na exploração de *software* SIMD ao reduzir o custo do compartilhamento de dados.

O processador Cell também explora ILP por meio da sua microarquitetura estaticamente escalonada, de despacho múltiplo e *power-aware*. Provendo paralelismo estaticamente escalonado entre unidades de execução permite-se o duplo despacho de instruções tanto para o PPE quanto para os SPEs. Em ambos os elementos, o duplo despacho é limitado a sequências de instruções que combinam com as unidades de execução provisionadas como se fosse num processador de despacho simples. Ou seja, as instruções devem ser escalonadas de modo a combinar com o perfil de recursos como se não houvesse nenhum suporte a reordenação de instrução para um potencial despacho múltiplo e, além disso, as unidades de execução não são duplicadas.

Apesar destas características limitarem o despacho duplo, elas implicam numa execução paralela com consumo consciente de energia. Não são necessários *buffers* de reordenação, unidades de renomeação de registradores, *commit buffers* ou unidades similares. Como o perfil de recursos é conhecido, o compilador pode estaticamente escalonar instruções de modo eficiente. Ou seja, no Cell, ILP é explorado de maneira eficaz, de modo a não comprometer o consumo de energia para um ganho de desempenho marginal.

O TLP é suportado pelo PPE *multithreaded* e pelos múltiplos SPEs em um único processador, entregando um ganho de desempenho significativo ao fornecer dez contextos de execução a aplicações *multithreaded*, com um desempenho total excedendo a casa dos 200GFLOPs.

Já para aumentar o desempenho de uma *thread* explora-se o CTP, que desacopla e paraleliza computação e transferência de dados, resultando no uso mais eficiente da largura de banda da memória. O CTP considera operações de transferência de dados como ações explicitamente escalonadas que podem ser controladas pelo programa para melhorar a entrega de dados. Baseando-se no conhecimento em nível de aplicação, operações de movimento de dados explícitas são inseridas no *stream* de instruções a frente do seu uso, assegurando que o dado estará disponível quando da sua utilização, reduzindo o tempo ocioso do programa. Diferentemente do *prefetch* de dados orientado por *software*, que oferece acesso a um conjunto pequeno de dados por cada pedido de *prefetch*, o CTP é sequenciado de modo independente e tem como alvo transferências de blocos de até 16KB por pedido, além da transferência de comandos em lista. No Cell, as transferências em massa são realizadas pelos MFCs acoplados às SPUs.

Por fim, o MLP é uma técnica utilizada pelo Cell para explorar a largura de banda da memória aumentando o número de *cache misses* não resolvidos simultâneos. Isto tanto reduz o tempo de serviço médio da memória quanto aumenta a utilização da largura de banda como um todo, através da alternância de múltiplas transações de memória [5].

Enquanto *cache misses* isolados incorrem na latência total por acesso, com sobreposição limitada de computação/memória durante a fase inicial de acesso à memória, a sobreposição de *cache misses* pode reduzir significativamente o tempo médio de serviço de memória por transação. A figura 6 (a) mostra um cenário de *cache misses* em uma única *thread*. Com uma política comum de *stall-on-use*, execução e acessos à memória podem proceder em paralelo até que uma dependência seja encontrada,

expondo pouco CTP. O problema chave desses tipos de sequência é que acessos isolados à memória são seguidos de períodos curtos de computação, até que a próxima linha de *cache* precise ser buscada.

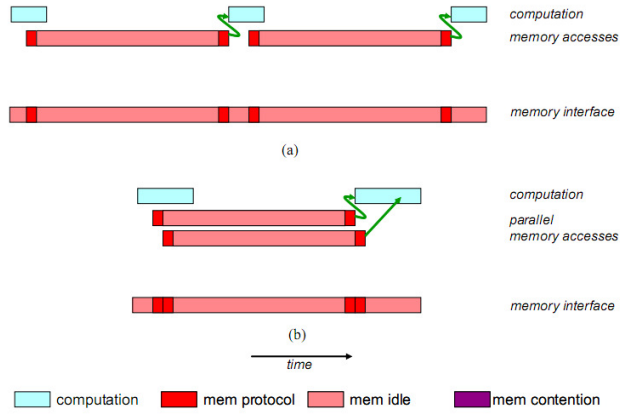


**Figura 6: Cenários de cache miss para *single threaded workloads*: (a) *cache misses* isolados e (b) *cache misses* agrupados expondo MLP.**

Já o cenário (b) da figura 6 adiciona paralelismo entre os acessos. Esse código pode ser encontrado quando o escalonamento de instruções é desenvolvido para iniciar acessos múltiplos à memória. A política *stall-on-use* é importante por permitir a descoberta de múltiplos *cache misses* e a inicialização de vários pedidos concorrentes à memória desde que não exista nenhuma dependência de dados. Esse paralelismo é também referenciado por MLP, e reduz o tempo de execução geral sobrepondo múltiplas operações de longa latência.

A figura 7 mostra como o CTP e o MLP são fornecidos pelo processador Cell. Como descrito na seção 2.3.2, cada SPE consiste de subunidades separadas e independentes, a SPU, direcionada para o processamento de dados, e o MFC para transferência de dados em massa entre o sistema de memória principal e o espaço de armazenamento local da SPU. Esta arquitetura dá aos programadores novos caminhos para atingir desempenho na aplicação explorando o CTP disponível nela.

Muitas aplicações ricas em dados podem predizer os acessos a blocos de dados baseadas na estrutura do programa e escalonar transferências de dados antes deles serem de fato requisitados pelo programa, evitando, assim, que o programa entre em *stall* enquanto espera pelo dado.

Em adição ao CTP disponível dentro do SPE, a arquitetura *multicore* do Cell também permite explorar MLP através das *threads* dos SPEs e do PPE. Explorar o paralelismo entre computação e transferência de dados (CTP) e entre múltiplas transferências simultâneas de memória (MLP) pode entregar um *speedup* superlinear em várias aplicações com relação ao crescimento do MIPS provido pelo Cell.

Ainda com relação ao sistema de memória, para aumentar sua eficiência, a tradução de endereços só é realizada durante as operações de transferência de blocos, tendo como vantagem o fato da tradução de um único endereço ser usado para traduzir operações correspondentes a uma página inteira de uma vez, ao invés de a cada acesso a operandos. Além disso, a implementação das áreas de armazenamento local dos SPEs baseada na semântica *copy-in copy-out* faz com que nenhuma coerência seja mantida com o sistema de memória principal, o que aumenta a densidade de armazenamento pela eliminação dos custosos *arrays* de *tags* para manter a correspondência entre a memória local e principal presentes nas *caches* tradicionais. Ademais, a abstração do espaço de memória local provê um armazenamento de dados de operandos denso, de uma única porta, com latência de acesso determinística, o que fornece a habilidade de realizar substituição de dados gerenciada por *software* em *workloads* com padrões de acesso previsíveis. Isto possibilita que a exploração de técnicas de tolerância de latência baseadas no compilador, como *software pipeline*, possam ser aplicadas para diminuir de forma eficiente a longa latência de operações de memória[6].

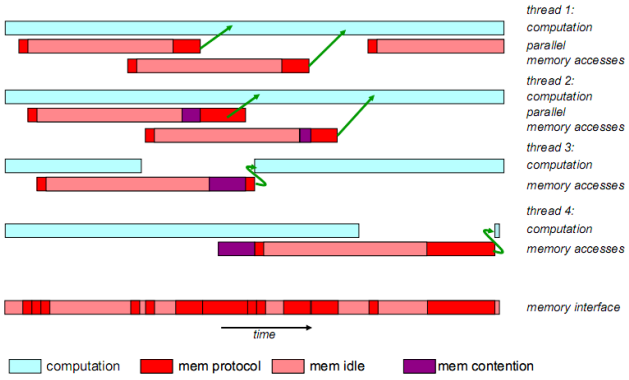


**Figura 7: CTP e MLP sendo explorados pelo Cell em *workloads multithreaded*.**

## 4. Conclusão

Os grandes obstáculos que surgiram no caminho dos processadores tradicionais rumo à melhoria de desempenho conduziram os fabricantes de *chips* para os projetos *multicore*. Neste contexto o processador Cell foi criado como um *chip* multiprocessado de arquitetura revolucionária, que estende a organização de microprocessadores convencionais.

Este artigo apresentou informações gerais sobre a arquitetura do processador Cell, estudando seus principais elementos e como estes se comunicam. Descreveu, ainda, como o Cell explora o paralelismo em múltiplos níveis[7], e como tal característica garante o sucesso no desempenho do multiprocessador.

## 5. Referências

[1] M. Gschwind et al., "Synergistic Processing in Cell´s Multicore Architecture", IEEE Micro, Mar./Apr. 2006, pp 10-24.

[2] J.A. Kahle ET AL., "Introduction to the Cell Multiprocessor", IBM J. Research and Development, vol.49, no. 4/5, July 2005, pp. 589-604.

[3] M. Kistler et al., "Cell Multiprocessor Communication Network: Built for Speed", IEEE Micro, May 2006, pp 10-23.

[4] M. Gschwind, “Chip Multiprocessing and the Cell Broadband Engine”, Proc. ACM Computing Frontiers 2006, ACM Press, May 2006, pp 1-8.

[5] M. Gschwind, “The Cell Broadband Engine: Exploiting multiple levels of parallelism in a chip multiprocessor”, IBM J. Research and Development, Oct. 2006.

[6] A novel SIMD architecture for the Cell heterogeneous chip multiprocessor:

http://www.iuma.ulpgc.es/~nunez/clases-micros-para-com/Noticias/Cell%20Processor%20CF%2006%20Key note .pdf accessed on Jun. 2010.

[7] IBM – The Cell project at IBM Research:

http://www.research.ibm.com/cell/ accessed on Jun. 2010.