Implementação Prática de Algoritmo de Estimativa de Movimento H.264 Paralelo em um Processador Cell

Ricardo Kintschner, Ronaldo Husemann
Departamento de Engenharia Elétrica – DELET – UFRGS
Av. Osvaldo Aranha, 103 – Porto Alegre – RS – Brasil
{rkintschner, rhusemann}@inf.ufrgs.br

José Valdeni de Lima, Valter Roesler
Instituto de Informática – II – UFRGS
Av. Bento Gonçalves, 9500 Bloco IV – Porto Alegre – Brasil
{valdeni, roesler}@inf.ufrgs.br

ABSTRACT
Inside a video encoder, the technique of motion estimation (or motion prediction) is one of the most relevant algorithms, which is particularly responsible by the consumption of most expensive portion of the global video encoder computational processing costs. The current paper explores the use of a Cell microprocessor as platform to the implementation of a dedicated parallel H.264 motion estimation solution. The proposed algorithm has been designed in order to operate simultaneously over six distinct regions of the same reference video frame. The obtained results points to significant improvements.

Categories and Subject Descriptors
I.4.2 [**Computing Methodologies**] I.4 Image Processing and Computer Vision - I.4.2 *Compression (Coding)*

General Terms
Algorithms, Performance and Design.

Keywords
H.264 encoder, Motion Estimation, Cell processor , CBEA.

1. INTRODUÇÃO

Dentro de um codificador de vídeo, um dos módulos mais relevantes e que responsável pelas maiores demandas de processamento. O algoritmo de estimativa de movimento tenta reduzir a redundância temporal nas sequências de vídeo compensando os movimentos realizados durante a sucessão de quadros. Sua implementação prática demanda uma elevada carga computacional, pois vídeos digitais tipicamente são compostos por grandes quantidades de informação (conjuntos de pixels que compõem as sequências de imagens) [1].

WebMedia'11: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web. Full Papers.
October 3 -6, 2011, Florianópolis, SC, Brazil.
ISSN 2175-9642.
SBC - Brazilian Computer Society

A necessidade de se trabalhar com aplicações multimídia faz com que engenheiros e programadores explorem o uso de arquiteturas dedicadas para DSP (*Digital Signal Processing*) e MPC (*Multi-Processing Chip*).

Um exemplo atual de solução que consegue explorar em uma mesma arquitetura praticamente todas essas tecnologias (DSP, SIMD e MPC) é o processador *Cell Broadband Engine* (CBE) [2].

Esse processador possui uma poderosa arquitetura de multiprocessamento heterogêneo (distintas estruturas computacionais inclusas em uma mesma arquitetura), com nove núcleos de processamento: um microprocessador central, denominado *PowerPC Processor Element* (PPE) e oito processadores de DSP, denominados *Synergistic Processor Elements* (SPEs). Apesar de pode ser utilizado em aplicações genéricas, deve-se considerar que um CBE opera com maior eficiência sobre aplicações que trabalham com operações matemáticas (somas, subtrações, multiplicações, rotações entre outras) e que utilizam grandes volumes de dados, como, por exemplo, aplicações de multimídia e de processamento vetorial.

Estas características se adéquam bem ao algoritmo de estimativa de movimento de um codificador de vídeo. As diversas etapas de cálculo envolvidas nesse algoritmo podem ser consideradas como operações independentes, quando se consideram as análises que são feitas em regiões geometricamente distintas da imagem, o que levanta a possibilidade de realizá-las de forma independente e paralela.

Baseado nisto o presente artigo explora o uso de um CBE para a implementação de um algoritmo paralelizado de estimativa de movimento do codificador H.264. O algoritmo foi desenvolvido de forma a operar de forma paralela sobre seis distintas regiões de um mesmo quadro de referência de vídeo, cada qual rodando em um SPE distinto. Os resultados obtidos comprovam o aumento significativo de eficiência ao se utilizar uma plataforma distribuída como é o caso da CBE para este tipo de algoritmo.

O artigo se divide da seguinte forma: Seção 2 apresenta uma visão geral sobre o codificador de vídeo H.264 com especial destaque para o algoritmo de estimativa de movimento; a Seção 3 descreve a arquitetura interna de um CBE; a Seção 4 apresenta o algoritmo proposto, bem como os detalhes de sua implementação; a Seção 5 traz os resultados práticos obtidos enquanto que a Seção 6 apresenta as considerações finais dos autores.

2. ALGORITMOS DE UM CODIFICADOR PADRÃO H.264

Vídeos digitais tipicamente demandam uma grande taxa de bits para transmissão. Como exemplo, basta considerar que uma sequência de HDTV (1920x1080 pixels), utilizando um formato 4:2:0 a 30fps, ocupa aproximadamente 746 Mbits por segundo. A fim de reduzir esta quantidade de informações algoritmos de compressão de dados se baseiam em encontrar e remover redundâncias do vídeo, reduzindo assim o número de informações necessárias para poder representá-lo.

De forma simplificada, pode-se dizer que a maioria dos métodos de compressão de vídeo explora dois tipos de redundância: (i) espacial, que existe entre pixels próximos em uma imagem, e (ii) temporal, que ocorre entre quadros adjacentes no vídeo. Para reduzir a redundância espacial, são utilizados métodos de compressão de imagens estáticas. Já para a redundância temporal, empregam-se algoritmos de estimativa de movimento sobre um modelo temporal [3].

O modelo temporal permite a análise de similaridade entre quadros temporalmente distintos a fim de se aproveitar de informações já transmitidas. O quadro que pode ser montado a partir de informações previamente transmitidas é chamado de quadro predito ou quadro estimado. O quadro predito pode ser formado a partir de um ou mais quadros passados ou futuros. O processo de predição pode ser melhorado significativamente compensando o movimento entre os quadros de referência e o quadro atual. Para que isso seja possível, é necessário estimar o movimento de cada bloco de um dado tamanho (M x N pixels), considerando as relações de posicionamento de blocos similares em um quadro de referência, gerando os chamados vetores de movimento. Um vetor de movimento representa a informação de diferença relativa entre a posição atual e a posição onde um bloco muito similar foi localizado em um quadro já transmitido. Muitos padrões importantes de codificação (incluindo MPEG-1, MPEG-2, MPEG-4 Visual, H.263 e H.264) utilizam como unidade básica para a predição de movimento o macrobloco, que corresponde a uma região de 16 x 16 pixels [3].

Para encontrar os vetores de movimento, é necessário fazer para cada bloco da imagem atual uma busca na imagem de referência pelo bloco correspondente, o que exige um grande número de comparações. Essas comparações consomem muito tempo, e por isso vários algoritmos de busca têm sido propostos, nos últimos anos, visando diminuir o número de comparações necessárias sem que isso implique em um aumento significativo na taxa de bits final [4].

Uma técnica bastante utilizada é a de restringir a área de busca a uma região de um determinado tamanho, chamada janela de busca, que está centrada na posição atual do bloco, pois se assume que, na maioria das vezes, o movimento não é rápido o suficiente para exceder os limites daquela região

Na maioria das vezes, o bloco encontrado não é exatamente igual ao que se procura. Os algoritmos devem assim tentar encontrar um bloco na imagem de referência que possua a maior similaridade possível com o bloco atual.

Para isso, é necessária uma medida de similaridade que seja eficaz e computacionalmente eficiente, pois seu cálculo será realizado para comparações feitas entre blocos em uma grande quantidade de posições possíveis. Dentre os mecanismos de cálculo de similaridade mais utilizados estão o SAD (*Sum of Absolute Differences*), que é a soma das diferenças absolutas dos valores de cada pixel, o MSE (*Mean Squared Error*), que é o erro quadrático médio, o MAD (*Mean Absolute Differences*), que é a diferença absoluta média, entre outros.

Um esquema geral do processo de cálculo de vetor de movimento para um bloco pode ser visto na figura 1. Na figura, um bloco similar ao do quadro atual está sendo procurado ao redor de uma janela de busca que se estende de [-p, p], centrada no próprio bloco. Quando o bloco de maior similaridade é encontrado, a diferença de posição relativa (deslocamento) do bloco encontrado em relação ao original é considerada como o vetor de movimento detectado.

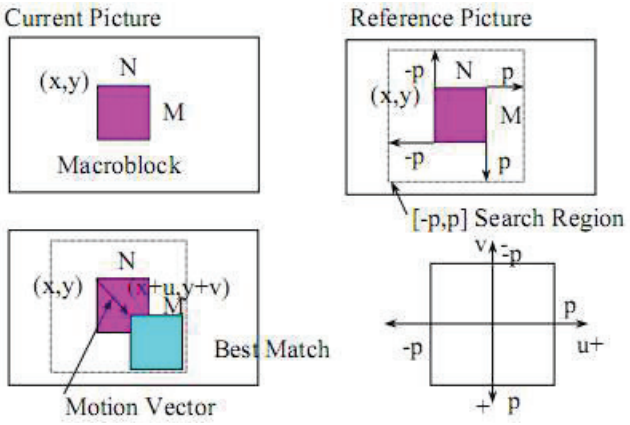

Figure 1. Mecanismo de cálculo do vetor de movimento.

Subtraindo-se o quadro predito do quadro atual, produz-se o que se chama de quadro residual. O quadro residual é codificado e enviado para o decodificador, de forma a corrigir os erros produzidos pela reconstrução do vídeo utilizando-se apenas o quadro predito. Quanto melhor for a predição, menos bits serão necessários para o quadro residual, pois menos erros serão detectados [5].

3. SOLUÇÃO CELL BROADBAND ENGINE

A Cell Broadband Engine Architecture (CBEA) foi apresentada para a comunidade em 2005, como uma solução que estende a estrutura PowerPC de 64 bits da IBM, para um uma arquitetura de multi-processamento paralelo (AMORIM, 2005). O projeto gerou uma inédita solução multiprocessada com nove processadores independentes operando sobre uma memória compartilhada e coerente. A idéia principal foi utilizar um processador central PowerPC, denominado PPE, com funções comuns de controle, que coordena oito processadores vetoriais chamados SPEs, que são otimizados para processamento intensivo de dados. Uma visão geral da arquitetura CBEA pode ser vista na figura 2.

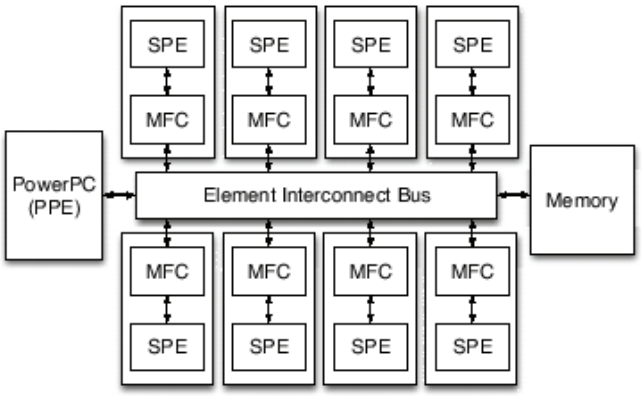

Figure 2. Arquitetura Cell.

Os PPEs e SPEs possuem acesso a um espaço comum de endereçamento, que inclui a memória principal, e uma extensão de endereços que correspondem à memória local de cada SPE, registradores de controle e dispositivos de entrada e saída. O PPE é um processador RISC (*Reduced Instruction Set Computer*) de 64 bits de propósito geral, com suporte para até duas threads. Sua arquitetura está em conformidade com a versão 2.02 da arquitetura PowerPC, criada em 1991 por uma aliança entre Apple, IBM e Motorola. O PPE é responsável pelo controle geral do sistema. É nessa unidade que executa o sistema operacional, que gerencia todas as aplicações a serem executadas no próprio PPE e nos SPEs. O PPE é constituído de duas unidades principais:

- PPU (*Power Processor Unit*), que executa propriamente as instruções de programa;
- PPSS (*Power Processor Storage Subsystem*), que trata as requisições de memória da PPU e as requisições externas das SPEs e dispositivos de I/O.

Os SPEs, por sua vez, são processadores independentes, cada um executando uma thread independente. Cada SPE inclui uma memória local (*Local Store* – LS) privada para acesso eficiente a dados e instruções. Além disso, cada SPE também possui acesso completo à memória compartilhada coerente, incluindo a área mapeada para I/O (GSCHWIND, 2006). Cada SPE é um processador RISC de 128 bits especializado para aplicações escalares e SIMD, quando for exigida a computação intensiva de grandes volumes de dados. De forma geral, um SPE é constituído de dois blocos:

- SPU (*Synergistic Processor Unit*), basicamente voltado para fazer uso de instruções SIMD sobre vetores;
- MFC (*Memory Flow Controller*), que serve como interface entre a SPU e outros elementos de processamento, tais como módulo de cálculo de ponto fixo, ponto flutuante, canais de DMA, etc.

A comunicação entre os elementos de processamento e as estruturas externas de I/O é feita através de um barramento circular de alto desempenho, chamado EIB (*Element Interconnect Bus*), que permite um acesso à memória totalmente coerente com a memória cache. O EIB consiste de quatro anéis de dados de largura igual a 16 bytes. Cada anel transfere 128 bytes (uma linha de cache PPE) por vez. Múltiplas transferências podem ocorrer concorrentemente em cada anel, incluindo mais de 100 requisições DMA entre a memória principal e os SPEs. A banda interna máxima do EIB é de 96 bytes por ciclo de relógio [6].

O software que roda no PPE, outros SPEs e dispositivos podem acessar funções das interfaces de canais do SPE através dos registradores MMIO (*Memory-Mapped I/O*) no espaço da memória principal. Para tanto *mailboxes* são usados para comunicação entre processos. Cada interface de canal em cada SPE suporta dois *mailboxes* independentes: um para mensagens de um SPE para o PPE (*outbound mailbox*) e uma para mensagens no caminho inverso (*inbound mailbox*). SPEs podem trocar mensagens desse tipo entre si, de forma a permitir o acesso aos registradores MMIO um do outro.

Um *inbound mailbox* é capaz de armazenar até quatro mensagens em um buffer do tipo FIFO. A leitura em um *mailbox* vazio bloqueia SPU até uma mensagem ser recebida. Já um *outbound mailbox* somente suporta uma mensagem ao PPE por vez, sendo a SPU bloqueada em uma tentativa de escrever em um mailbox cheio. O PPE precisa requisitar a mensagem a um determinado SPE para que ela seja lida. Para evitar *polling* e, consequentemente, tráfego desnecessário no EIB, existe um *outbound mailbox* especial, chamado *interrupt outbound mailbox*.

4. SOLUÇÃO PROPOSTA

A maioria dos algoritmos de busca começa a procurar pelo bloco em alguma posição inicial e, a partir daí, realizam a procura pelas proximidades de acordo com regras e padrões de busca próprios até encontrar um erro mínimo. A posição inicial pode ser a própria posição do bloco na imagem atual ou pode ser determinada por um ou mais vetores preditores, que podem ser escolhidos de diversas formas. Em [8] é proposto um algoritmo de estimativa de movimento que tem como objetivo acelerar o processo sem que se tenha uma perda significativa na qualidade da imagem final.

4.1 Algoritmo Proposto

O algoritmo escolhido utiliza um padrão de busca simples, conhecido como padrão de diamante pequeno. Para cada estágio do padrão de diamante pequeno, cinco candidatos são analisados em uma topologia em forma de cruz (centro, acima, abaixo, esquerda e direita) com resolução de um pixel, como mostra a figura 3. Após a realização do cálculo de similaridade para todos os cinco possíveis blocos, o candidato com a maior similaridade (menor SAD) é escolhido, tornando-se o centro do diamante para uma nova busca. A busca termina quando o a maior similaridade for a do bloco na posição central.

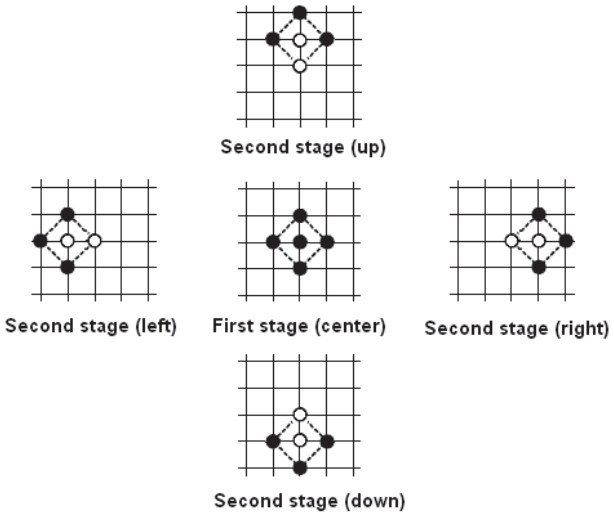

Figure 3. Padrão de diamante pequeno

É importante notar que, neste algoritmo, apenas no primeiro estágio é preciso calcular o SAD das cinco posições de bloco. Para os estágios seguintes, apenas três são necessários (indicados como quadrados pretos na figura), uma vez que dois deles já foram calculados no estágio anterior. Isso contribui para a redução do número de iterações necessárias para se localizar o bloco de maior similaridade, aumentando o desempenho do algoritmo de forma global.

Outra consideração feita na definição deste algoritmo foi de que macroblocos espacialmente próximos geralmente possuem uma grande correlação. Sendo assim, valores já calculados de posições vizinhas podem ser usados como preditores para a posição inicial de cada procedimento de busca, reduzindo o número de iterações necessárias para encontrar o bloco de similaridade. Além disso, o uso desses preditores ajuda a reduzir o erro causado por SADs mínimos locais. Na prática, como a procura por vetores de movimento é feita linha a linha, da esquerda para a direita e de cima para baixo para cada macrobloco da imagem atual, três valores que podem ser considerados como "bons candidatos". Estes representam os vetores encontrados para o bloco à esquerda, para o bloco acima e para o bloco acima e à direita.

No algoritmo de [8] a posição inicial de busca é calculada pela média dos vetores já calculados dentre esses três (desde que já tenham sido determinados). Na prática, a definição real do número de preditores a serem considerados vai depender da respectiva localização do bloco alvo dentro da janela de busca, uma vez que blocos localizados nas bordas superior, à direita e à esquerda não têm todos os seus vizinhos.

4.2 Cálculo de Similaridade

No trabalho em questão foi utilizado o método de similaridade SAD. O SAD resultante da comparação de um bloco A de tamanho NxN localizado na posição (x,y) dentro do quadro atual com um bloco B localizado a um deslocamento (vx,vy) em relação a A no quadro de referência é definido como [9]:

$$SAD(v_x, v_y) = \sum_{m,n=0}^{N-1} |I_t(x+m, y+n) - I_{t-i}(x+v_x+m, y+v_y+n)| \quad (1)$$

Por sua simetria, este algoritmo pode ser facilmente implementado utilizando operações SIMD.

No trabalho de [8] para facilitar a utilização de operações vetoriais, propõe-se que todas as estruturas utilizadas no algoritmo utilizem uma granularidade de 64 bits, o que permite a leitura de um grupo de 8 bytes (8 x 8 = 64 bits), sendo obtidas assim com um único acesso à memória.

A busca é realizada com blocos de tamanho 16 x 16, e o padrão de diamante utilizado tem resolução de um pixel. Outra estratégia utilizada para redução do número de operações é a de utilizar o cálculo da similaridade utilizando uma versão subamostrada do bloco atual (no caso com tamanho 8 x 8). Isso, além de diminuir o número de subtrações em quatro vezes, ainda possibilita que cada linha ou coluna do bloco seja representada exatamente por um valor de 64 bits.

A fim de se evitar a realização do procedimento de subamostragem e remontagem dos vetores de 64 bits a cada novo bloco pesquisado na imagem de referência, essa imagem é inicialmente dividida em quatro imagens subamostradas.

Na prática, esse processo de subamostragem classifica os pixels de acordo com suas linhas e colunas, dependendo se forem pares ou ímpares. O bloco do quadro de referência é implementado com quatro vetores de 64 bits, chamados de vetores lineares, cada um contendo os pixels do bloco que estiverem em uma das subamostragens da imagem de referência. Dois desses vetores são alinhados com a direção horizontal, sendo os outros dois alinhados com a direção vertical, o que também favorece o uso de instruções SIMD.

Para possibilitar o cálculo dos cinco SADs correspondentes ao padrão de diamante, devido às paridades dos pixels, são necessários três vetores diferentes (um para a posição central, um para os deslocamentos verticais e um para os deslocamentos horizontais).

4.3 Algoritmo no PPE

Na primeira vez que o módulo de estimativa de movimento é chamado, o PPE inicialmente aloca memória para todas as estruturas necessárias para subquadros e vetores de movimento e inicializa os SPEs. Essa inicialização inclui a carga do programa, que é o mesmo para todos os SPEs, e o comando de início de execução (cada um em uma *thread*).

A figura 4 mostra um fluxograma em passos gerais do algoritmo executado pelo PPE para cada novo quadro a ser calculado.

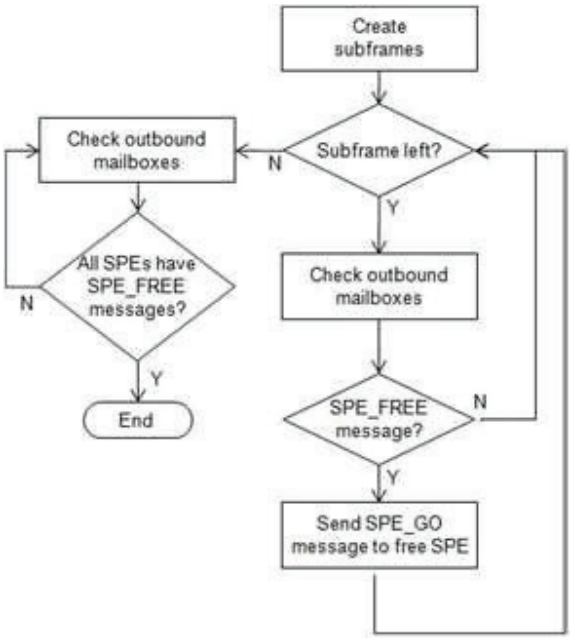

Figure 4. Algoritmo de controle que roda na PPE

4.4 Algoritmo no SPE

Um SPE inicia a execução de seu código com o pedido para o PPE de uma estrutura que descreve alguns parâmetros do vídeo, como altura e largura.

A partir desses parâmetros, é possível realizar todas as alocações de memória necessárias. Após a alocação de memória, inicia-se o laço principal do algoritmo do SPE. A figura 5 mostra um fluxograma em passos gerais do algoritmo executado por cada um dos SPEs.

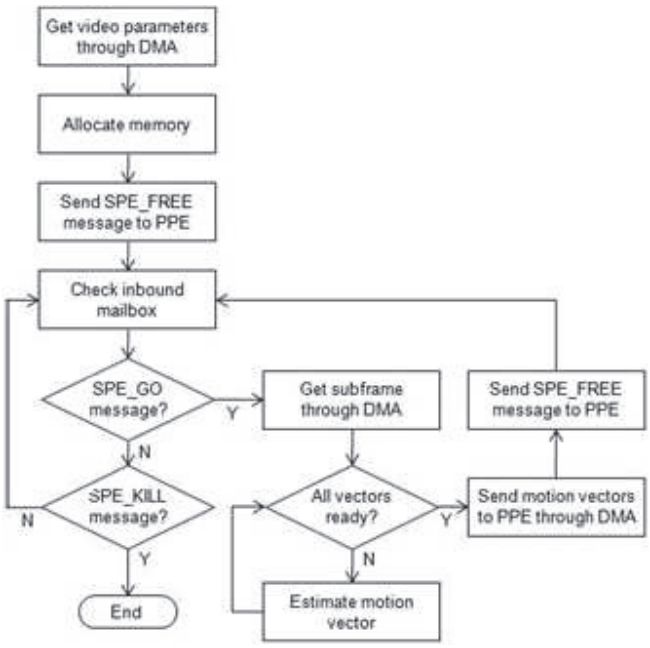

Figure 5. Algoritmo que roda na SPE

Além da gerência do algoritmo em si os procedimentos de cálculos foram implementados utilizando-se a arquitetura de SIMD de cada SPE, visando aumentar a eficiência do mesmo.

Conforme apresentado na equação (1), o SAD entre dois blocos é calculado pela soma das diferenças absolutas dos valores dos pixels desses blocos. Analisando-se o conjunto de instruções do SPE, encontrou-se uma instrução, denominada spu_absd, que realiza, de uma única vez, o cálculo das diferenças absolutas entre os bytes de dois vetores de 128 bits (16 bytes em paralelo).

Os cálculos de SAD no algoritmo proposto são realizados sobre macroblocos subamostrados, de tamanho 8 x 8, que são armazenados em vetores alinhados em 64 bits (8 bytes).

Considerando-se a arquitetura do SPE, um bloco inteiro pode ser considerado como quatro vetores de 16 bytes (16 x 8 bits = 128 bits).

A partir dessa manipulação dos dados, pode-se então utilizar a instrução spu_absd para calcular as diferenças absolutas entre os pixels de dois blocos com apenas quatro instruções.

Entretanto, na prática, a realização da soma dessas diferenças absolutas não ocorre de maneira tão direta, pois a instrução spu_absd gera uma saída vetorial, e não escalar.

Para o cálculo total do SAD, de fato, são utilizadas as seguintes instruções SIMD do SPU ISA:

- spu_absd(a,b) – Calcula, a diferença absoluta entre os bytes correspondentes dos vetores a e b, armazenando o resultado no vetor r;
- spu_rlqwbyte(a,n) – Rotaciona a palavra de 128 bits a de n bytes para a esquerda, armazenando o resultado no vetor r;
- spu_add(a,b) – Soma os elementos dos vetores a e b, armazenando o resultado no vetor r;
- spu_sumb(a,b) – Realiza somas de 4 em 4 bytes com os vetores a e b, armazenando o resultado em valores de 16 bits no vetor r;
- spu_extract(a,n) – Armazena na variável escalar x o valor do elemento n do vetor a.

Tomar como exemplo a instrução spu_sumb. É importante perceber que, apesar de essa instrução receber como entrada dois vetores de 16 bytes, a sua saída é um vetor composto por oito valores de 16 bits.

Na prática, a execução dessa instrução totaliza em oito valores, cada um sendo o resultado da soma de 4 bytes dentro de um dos vetores.

O cálculo de SAD entre dois blocos, considerando que cada bloco é composto de quatro vetores SIMD de 128 bits, é realizado separadamente dois a dois por vez, somando-se os resultados obtidos em cada operação de soma para se obter o SAD total.

Cada soma individual de vetores é calculada, inicialmente, através de uma mesma operação spu_absd, retornando em um vetor as diferenças absolutas relativas (16 valores de diferenças com resolução de byte).

Em seguida, são tomados dois desses vetores resultantes, que são somados com a instrução spu_sumb, resultando em um vetor com oito elementos de 16 bits.

Finalmente, para somar os elementos desse vetor, utiliza-se uma técnica que consiste em realizar somas sucessivas do vetor com cópias dele mesmo, porém rotacionadas. A cada iteração, o passo de rotação é aumentado, até que se obtenha a soma total de todos os elementos.

A figura 6 mostra o funcionamento dessa técnica. Vale lembrar que o vetor a ser somado é composto de valores de 16 bits, porém a instrução utilizada para as rotações, spu_rlqwbyte, recebe como parâmetro um número de bytes (8 bits) a serem rotacionados.

A instrução spu_extract, por fim, é utilizada para extrair uma das somas totais no vetor resultante.

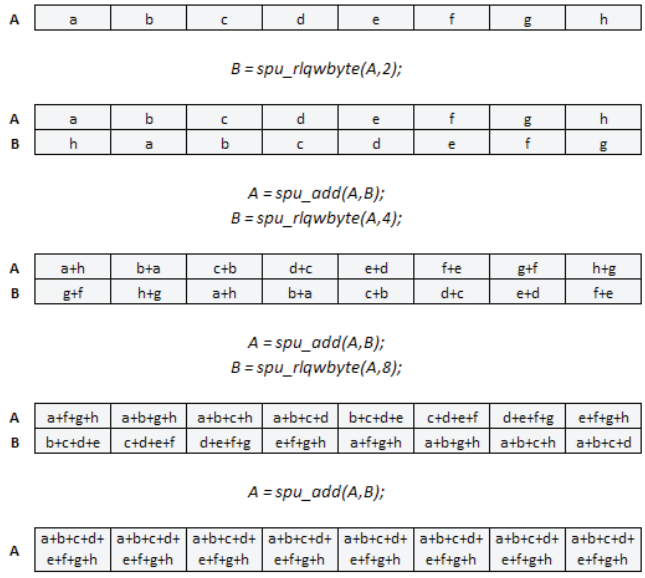

Figure 6. Implementação do cálculo de SAD utilizando instruções SIMD.

Como se pode perceber, o uso de instruções SIMD no SPE exige que os dados estejam alinhados com palavras de 128 bits. Assim sendo, os vetores do algoritmo proposto, que operava inicialmente com dados alinhados em 64 bits, teve de ser adaptado para agrupar dois vetores em uma estrutura nova de 128 bits antes do cálculo do SAD ser realizado.

5. RESULTADOS

Os algoritmos citados foram validados sobre uma plataforma real que utilizava o CBE como microprocessador principal.

A plataforma utilizada nestes experimentos práticos foi um console Sony PlayStation 3 (PS3), onde foi instalado o sistema Ubuntu 9.04.

A plataforma PS3 não foi projetada para desenvolvimento de aplicativos externos e por isso apresenta algumas limitações de uso. Nesta plataforma, por exemplo, apenas seis SPEs do CBE estão disponíveis para programas de usuário.

Os desenvolvimentos realizados nestes experimentos práticos utilizaram como base o software de referência oficial do codificador H.264, denominado de JM (JVT *Model*), versão 16.2, que é fornecido pela entidade JVT (*Joint Video Team*) mantenedora de software das normas H.26x [10].

Com isso pode-se contar com uma mesma base confiável para experimentos de comparação de qualidade de imagem e de desempenho.

O software JM é bastante modular, o que facilita a adaptação e testes de algoritmos individualmente.

Considerando-se as características da proposta em questão, apenas o módulo de estimativa de movimento do JM precisou ser adaptado [10].

Durante o desenvolvimento, quatro versões distintas de módulos de estimativa de movimento foram geradas, cada uma correspondendo a uma etapa diferente da implementação do algoritmo:

I. **Básica**: versão não paralela, onde o algoritmo executa inteiramente no PPE sobre a imagem inteira;

II. **Multi-quadro**: versão onde a imagem é dividida em 16 subquadros, todos estes manipulados no PPE;

III. **Paralela padrão**: versão paralelizada com imagem dividida em 16 subquadros, onde o PPE apenas prepara as estruturas de dados, e os SPEs processam os subquadros, sem o uso de instruções SIMD;

IV. **Paralela otimizada**: versão paralelizada com imagem dividida em 16 subquadros, onde o PPE apenas prepara as estruturas de dados, e os SPEs processam os subquadros, com o uso de instruções SIMD.

Para o levantamento de dados práticos comparativos, essas versões foram aplicadas sobre distintas sequências de vídeo. Foram utilizadas sequências de vídeo bem conhecidas da comunidade científica: City, Crew, Harbour e Ice, todas em tamanho 4CIF (704x576 pixels).

A escolha desses vídeos foi feita visando avaliar na prática o desempenho do algoritmo para vídeos com características diferentes de movimentação. Com exceção da sequência Crew, onde foram codificados 43 quadros, todas as sequências foram codificadas até o centésimo quadro.

Também para avaliar o desempenho do algoritmo em diferentes situações, foram definidas quatro taxas de bit de saída diferentes, que variam de 1500 a 3000 Kbps.

A resolução dos vídeos, a dimensão da janela de busca e o número de subquadros foram escolhidos considerando-se as limitações de capacidade de memória local (LS) de cada SPE, ou seja, o valor total não poderia ultrapassar o limite de 256KB. Considerando-se isso o tamanho utilizado para a janela de busca foi de 48x48 pixels com até 16 subquadros.

Os ensaios foram realizados em três etapas distintas. A primeira etapa teve como objetivo a medição prática dos ganhos percentuais, obtidos comparando-se o algoritmo em sua versão básica com as outras versões aprimoradas (multi-quadro, paralela normal e paralela otimizada).

Foram considerados nesta comparação os valores do tempo de execução específico da função de estimativa de movimento para cada uma das versões do algoritmo.

Na segunda etapa, os tempos de estimativa de movimento foram analisados de forma mais particularizada, de forma a segmentar o tempo gasto com preparação de estruturas e cálculo de movimento. A preparação de estruturas corresponde ao processo de subamostragem dos quadros e montagem dos subquadros, que é sempre executado no PPE.

O cálculo de movimento corresponde ao algoritmo de estimativa de movimento em si (busca e comparação do bloco de pixels, buscando-se o mais similar), que roda em paralelo nos SPEs nas versões paralelas (normal e otimizada).

Na terceira etapa, foi medido o tempo de execução médio utilizado para calcular todos os vetores de movimento de cada subquadro. Esse tempo, no caso das versões paralelas, inclui o tempo gasto com transferências de estruturas entre SPE e PPE. A versão básica do algoritmo não utiliza subquadros e, por isso, não é considerada nesta etapa.

Para a realização das medidas temporais foi tomada como base a unidade de "ticks" de relógio. Para a plataforma de testes utilizada, esses "ticks" são contados a uma freqüência base de 79,8MHz [11].

Na prática, para obter o tempo de execução no PPE, utilizou-se a instrução __mftb, que retorna o valor atual do registrador de base de tempo da PPU (Time Base Register), que é incrementado internamente segundo a frequência base citada.

Subtraindo-se o valor contido nesse registrador no início da execução de uma etapa do valor desse registrador no final da etapa, pode-se calcular o tempo total de execução de cada módulo do software.

Para os tempos de execução nos SPEs, o procedimento é análogo. Cada SPE possui um registrador (*SPU Decrementer*), que é decrementado à mesma frequência que o registrador de base de tempo do PPE.

Foi utilizada a instrução spu_write_decrementer para ajustar um valor inicial a esse registrador, lendo o valor final com a instrução spu_read_decrementer.

Como esse valor é decrementado, ao contrário do registrador de base de tempo, a subtração feita para se obter o tempo de execução precisa ser no sentido inverso.

Além das medidas de tempo de execução (desempenho), foi também realizada uma comparação dos resultados finais de qualidade sobre a imagem, em termos de PSNR, a fim de se analisar o impacto da omissão de preditores nas bordas dos subquadros. Medidas de PSNR sobre as imagens finais são calculadas e fornecidas pelo próprio software JM.

Nas seções seguintes, são apresentados e analisados os resultados obtidos em desempenho e qualidade.

5.1 Medida de Desempenho

As medidas de ganho de desempenho de cada uma das versões aprimoradas do algoritmo (multi-quadro, paralela normal e paralela otimizada) foram calculadas e resumidas na tabela 1.

O ganho foi calculado relacionando-se os tempos de execução da estimativa de movimento de cada versão aprimorada e o da versão básica.

TABELA 1 DESEMPENHO EM FUNÇÃO DO TEMPO DE ESTIMATIVA DE MOVIMENTO

Sequência	Ganho em relação à versão básica (%)			
	Taxa (Kbps)	*Multi-quadro*	*Paralela normal*	*Paralela otimizada*
CITY (704x576)	1500	69,96	73,80	86,70
	2000	27,31	37,74	45,65
	2500	25,44	35,34	46,16
	3000	23,31	34,00	43,99
CREW (704x576)	1500	20,64	24,81	38,79
	2000	18,03	20,83	32,80
	2500	19,59	22,10	35,37
	3000	19,67	22,71	36,15
HARBOUR (704x576)	1500	28,16	39,16	50,76
	2000	22,79	31,72	43,41
	2500	29,35	40,43	51,63
	3000	28,41	39,07	49,88
ICE (704x576)	1500	22,20	25,64	40,13
	2000	20,06	26,04	38,43
	2500	25,42	34,46	46,22
	3000	26,51	34,78	47,48

Os dados obtidos apontam para ganhos médios que variam de 26% a 45%, que correspondem aos ganhos reais obtidos com cada versão do algoritmo. Essa medição, no entanto, não reflete de forma individual o ganho obtido com o paralelismo nos SPEs. Isso, pois o tempo gasto com as preparações de estruturas, que são executadas no PPE para todas as versões do algoritmo, não pode ser desconsiderado. Além disso, por se tratarem de estruturas diferentes (visto que a versão básica trabalha com quadros, e as demais com subquadros), os tempos referentes à montagem de estruturas não é o mesmo em todas as versões. Como forma de melhor apresentar estes resultados a seguir (Figura 7) se apresentam estes na forma gráfica.

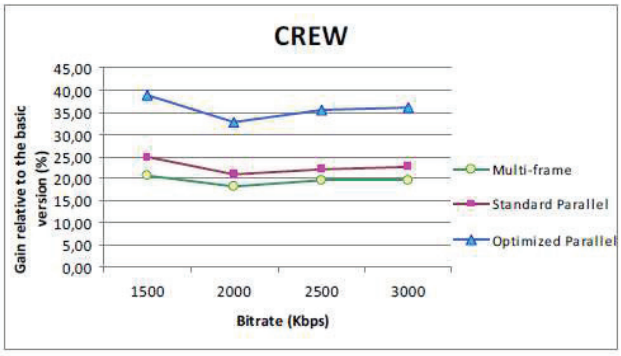

Figure 7. Ganhos de desempenho para a sequencia CREW.

5.2 Preparação e cálculo de movimento

Na prática, a fim de se obter uma medida real do ganho obtido com o paralelismo deste algoritmo, é preciso segmentar as medidas de tempo de execução em dois tempos diferentes: tempo com preparação de estruturas e tempo de cálculo de movimento, como na tabela 2.

Conforme foi dito na seção 4, a preparação de estruturas corresponde ao processo de subamostragem dos quadros e montagem dos subquadros (PPE), e o cálculo de movimento corresponde à estimativa de movimento propriamente dita (SPEs).

TABELA 2 TEMPO DE PREPARAÇÃO DE ESTRUTURAS DO ALGORITMO

Sequência	Tempo (milhões de "ticks")					
	Taxa (Kbps)	*Tempo gasto com*	*Básica*	*Multi-quadro*	*Paralela normal*	*Paralela otimizada*
CITY (704x576)	1500	Preparação	76.76	40,669	51,922	52,074
		cálculo	109.7	69,066	55,385	47,821
	2000	Preparação	75.59	41,189	46,443	48,327
		cálculo	63.37	67,964	54,447	47,080
	2500	Preparação	72,45	40,645	46,543	46,491
		cálculo	62.97	67,317	53,521	46,165
	3000	Preparação	69.85	40,499	45,666	46,032
		cálculo	62.08	66,489	52,783	45,590
CREW (704x576)	1500	Preparação	30,86	17,725	20,156	20,062
		cálculo	43,66	44,050	39,555	33,634
	2000	Preparação	28,93	17,404	19,819	20,292
		cálculo	41,34	42,125	38,329	32,617
	2500	Preparação	29,63	17,435	19,742	19,702
		cálculo	40,15	40,915	37,411	31,847
	3000	Preparação	29,21	17,382	19,783	19,639
		cálculo	39,68	40,190	36,362	30,965
HARBOUR (704x576)	1500	Preparação	72,02	41,181	47,309	47,026
		cálculo	67,44	67,638	52,911	45,479
	2000	Preparação	68,30	40,730	47,701	46,940
		cálculo	61,99	65,378	51,207	43,907
	2500	Preparação	72,92	40,946	46,075	45,947
		cálculo	62,38	63,649	50,271	43,280
	3000	Preparação	72,87	40,904	46,127	46,286
		cálculo	60,53	62,981	49,792	42,714
ICE (704x576)	1500	Preparação	69,70	41,247	49,357	47,671
		cálculo	99,87	97,512	85,612	73,341
	2000	Preparação	69,53	40,868	48,897	48,710
		cálculo	85,76	88,471	74,307	63,472
	2500	Preparação	74,42	40,841	46,216	46,813
		cálculo	78,83	81,354	67,757	57,993
	3000	Preparação	74,05	40,044	46,044	45,777
		cálculo	73,80	76,828	63,656	54,479

Também neste caso os dados obtidos foram compilados na forma de tabela para facilitar a visualização e entendimento deste, considerando-se a sequência de vídeo ICE como referência.

A partir desses dados, podem-se calcular separadamente os ganhos referentes à preparação de estruturas e ao cálculo de movimento De forma similar à seção anterior, os ganhos foram calculados dividindo-se os tempos de cada versão (multi-quadro, paralela normal e paralela otimizada) pelo tempo da versão básica.

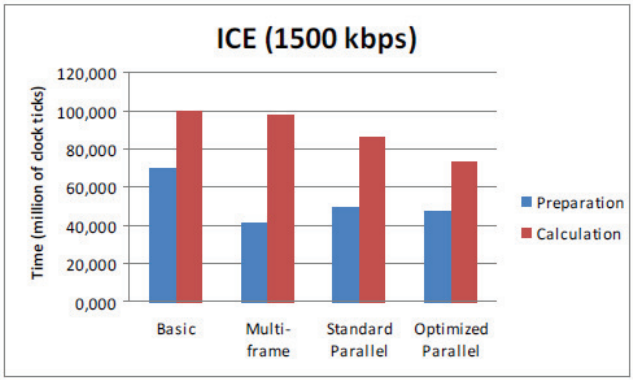

Figure 8. Tempos de execução individuais para a sequencia ICE

Os resultados apresentados até este ponto mostram que os ganhos médios relativos ao paralelismo obtido com os SPEs são de 17,6% e 37,2%, referindo-se respectivamente às versões paralela normal (sem o uso de instruções SIMD) e paralela otimizada (com o uso de instruções SIMD), em relação à versão básica.

6. CONCLUSÕES

Os resultados dos ensaios mostram que a etapa de cálculo de movimento em subquadros, na versão paralela, chegou a executar 37,2% mais rápida que a versão básica. Para alcançar esse ganho de desempenho, foram utilizadas seis unidades de execução paralelas (SPEs).

Experimentos práticos foram realizados a fim de avaliar métricas como ganho de desempenho e tempos de execução com até seis SPEs simultâneos. Os resultados mais significativos foram obtidos quando o algoritmo proposto foi otimizado para usar instruções dedicadas de CBE SIMD, obtendo-se ganhos relevantes na comparação da "versão básica" com a "paralela otimizada" para os vídeos testados.

REFERENCES

[1] Jack, K. Video Demystified: A Handbook for the Digital Engineer. 5th ed. [S.l]: Elsevier Science & Technology Books, 2007.

[2] GREHS, I. A. An LDPC Codes Simulator on the Cell Broadband Engine. 2009. 50 f., UFRGS, Porto Alegre.

[3] Richardson, I. E. G. H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia. [S.l]: Wiley & Sons, 2003.

[4] Husemann, R. et. al. Optimized SAD Calculation Algorithm for Cell Processor. In: Proced. of Webmedia, 2008. Vila Velha: [s.n.], 2008.

[5] IBM. Programming the Cell Broadband Engine Architecture: Examples and Best Practices. [S.l.: s.n.], 2008.

[6] Gschwind, M. et. al. Synergistic Processing in Cell's Multicore rchitecture. IEEE Micro. [S.l.], v.26, n.2, p. 10-24, mar. 2006.

[7] IBM. Cell Broadband Engine Programming Handbook: Including the PowerXCell 8i Processor. [S.l.: s.n.], 2008.

[8] Husemann, R. et. al. Proposal of an Improved Motion Estimation Module for SVC. In: Symposium On Applied Computing, SAC, 2010.

[9] Xuan Zhao, Ning Wan, Xiaofei Guo Parallel Motion Estimation of H.264 Decoder, Report, Spring 2010. 6 p.

[10] Suhring, K. H.264/AVC JM Reference Software. 2009.

[11] Wyganowsky, M. Classification Algorithms on the Cell Processor. 2008. 166 p. Kate Gleason College of Engineering, RIT, Rochester.