Explorando o paralelismo de dados e de thread para atingir a eficiência energética do ponto de vista do desenvolvedor de software

Rebeka G. de Oliveira[1], Sérgio V. Cavalcante[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 50740-540– Recife – PE – Brasil

{rgo2,svc}@cin.ufpe.br

Abstract *Environmental issues and the demand for mobility in mobile devices has generated a growing demand for reduction in energy consumption. Parallelism is a bet on energy efficiency in computing devices. This study aims to examine the use of thread parallelism and data parallelism for the purpose of improving energy efficiency from the point of view of the software developer. A case study was conducted with the benchmark ALPBench, which was analyzed the performance and power consumption of applications. The reduction in energy consumption reached 77.1% and the gain in performance reached 75.5%. The results provide further evidence that through parallelism is possible to achieve the energy efficiency of software applications.*

Resumo. *As questões ambientais e a demanda por mobilidade nos dispositivos móveis vem gerando uma crescente demanda pela redução no consumo de energia. O paralelismo é uma aposta para a eficiência energética em dispositivos computacionais. Esse trabalho tem como objetivo examinar o uso do paralelismo de thread e o de dados, com o propósito de melhorar a eficiência energética do ponto de vista do desenvolvedor de software. Um estudo de caso foi realizado com o benchmark ALPBench, no qual foram analisados a performance e o consumo de energia das aplicações. A redução no consumo de energia chegou a 77,1% e o ganho em desempenho a 75,5%. Os resultados reforçam as evidencias de que através do paralelismo é possível atingir a eficiência energética de aplicações de software.*

1. Introdução

1.1. Problema de pesquisa

Por muito tempo o foco principal no desenvolvimento de sistemas computacionais foi o desempenho. Com o crescimento da complexidade das aplicações atuais, com interfaces gráficas cada vez mais elaboradas e trabalhando sobre volumes de dados cada vez maiores, para atender a demanda por desempenho os sistemas precisam cada vez mais de poder de processamento, o que conseqüentemente leva a um maior consumo de energia.

No entanto, nos últimos anos, a demanda pelo uso eficiente de energia nos sistemas computacionais tem crescido. Essa demanda tem duas motivações principais. A primeira é por questões ambientais, devido aos problemas causados ao meio-ambiente, como as recentes mudanças climáticas, em decorrência do aumento crescente

no consumo de energia mundial. E a segunda motivação trata-se, particularmente, dos dispositivos móveis. Nesses casos o desafio principal é prolongar o tempo de vida da bateria e garantir que os requisitos de desempenho sejam atendidos.

Idealmente, o problema da eficiência energética deve ser abordado em todos os níveis do sistema, do hardware às aplicações de software. Entretanto, a maioria das pesquisas nessa área estão no nível de hardware e do sistema operacional (SO)[1]. Esse trabalho traz uma abordagem no nível de aplicação de software e procura mostrar evidencias de que o software, por reger a execução do hardware e do sistema operacional, tem grande potencial para o uso eficiente de energia.

A abordagem proposta neste trabalho é a paralelização de aplicações, explorando a capacidade do paralelismo, de thread e de dados, em reduzir o consumo de energia e aumentar o desempenho. O paralelismo de thread é o tipo de paralelismo mais comum. No entanto, o paralelismo de dados também pode ser usado para atingir a eficiência energética. Com o paralelismo de dados é possível executar uma mesma instrução sobre vários elementos de dados ao mesmo tempo. Isso faz com que haja a redução no número de instruções de *load/store* e conseqüentemente redução no consumo de energia e no tempo de execução.

Acreditamos que explorar os dois tipos de paralelismo pode trazer melhores resultados de desempenho e de consumo de energia, do que quando explorado apenas um tipo de paralelismo.

1.2. Objetivo da pesquisa

O objetivo principal do trabalho é analisar o uso do paralelismo de thread e de dados, através de instruções SIMD, com o propósito de melhorar a eficiência energética de aplicações de software do ponto de vista do desenvolvedor de software.

1.3. Contexto

Foram realizados experimentos com 3 aplicações multimídia do benchmark ALPBench [2]: MPEGEnc, MPEGDec e FaceRec. A escolha desse tipo de aplicação deve-se ao fato de que estas aplicações tipicamente possuem uma grande quantidade de paralelismo e são caracterizadas por executar muitas operações sobre dados, o que permite explorar também o paralelismo SIMD.

As aplicações foram analisadas em uma plataforma Intel T5500. Foram examinados o consumo de energia e o desempenho das aplicações em 4 situações: (1) explorando apenas o paralelismo de thread, (2) quando explorado apenas o paralelismo de dados, (3) utilizando os dois tipos de paralelismo e (4) quando nenhum tipo de paralelismo foi explorado.

1.4. Estrutura do artigo

O restante deste artigo está dividido em: Na próxima seção serão apresentadas as tecnologias investigadas nesta pesquisa e outras pesquisas relacionadas. Na seção III

será apresentado o experimento realizado. Os resultados do experimento serão discutidos na seção IV e finalmente, concluímos o artigo na seção V.

2. Trabalhos Relacionados

2.1. Paralelismo no nível de thread

O paralelismo de thread consiste da execução em paralelo de independentes conjuntos de operações. Cada conjunto de operação é chamado de thread. Para ser eficiente uma thread precisa ser o mais independente possível uma da outra, ou seja, devem existir poucos pontos de interação entre as threads, pois cada ponto de interação entre operações de threads diferentes exige mecanismos de sincronização para evitar problemas como *race condition* [3].

Como as operações de um mesmo conjunto podem ser dependentes ou independentes entre si, com relação a dados e a controle, caso haja independência entre as operações do conjunto e essas operações possuam o mesmo *op*-code, então o paralelismo de dados também pode ser explorado no conjunto.

A arquitetura de computador que dá suporte ao paralelismo de thread é, segundo a taxonomia criada por *Michael Flynn* [4], a *Multiple Instruction Multiple Data* (MIMD).

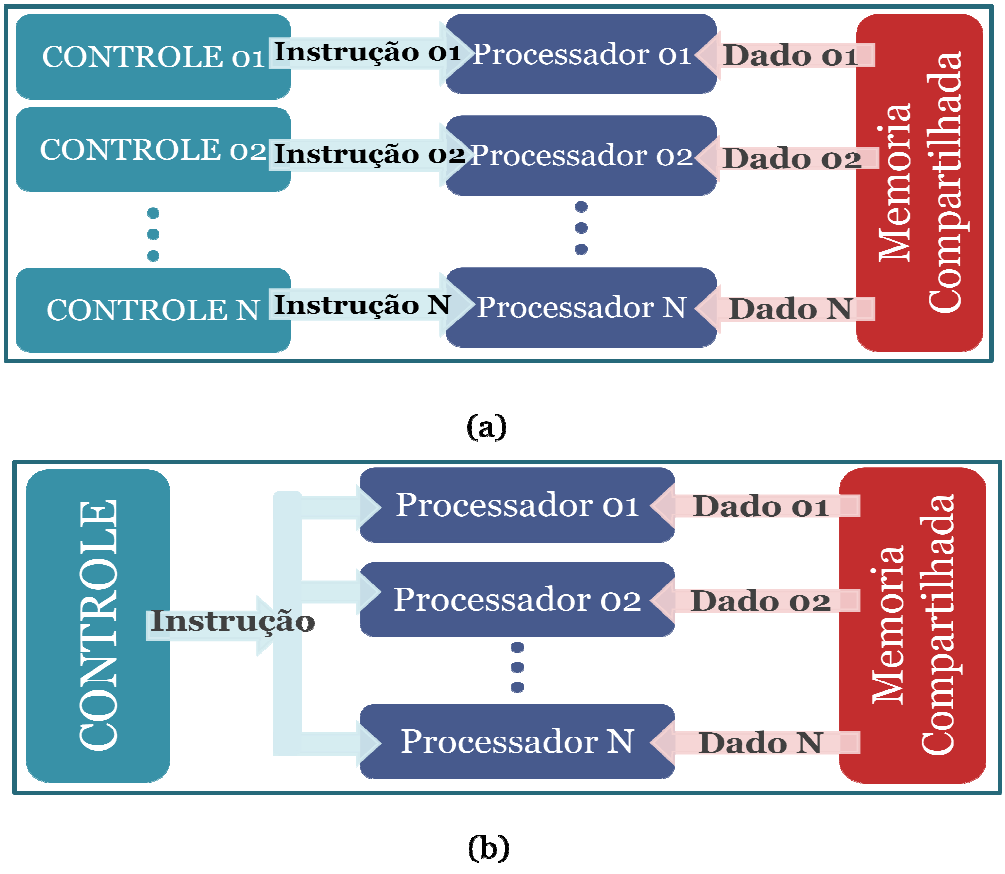

Figura 1. Computador MIMD e SIMD, respectivamente.

Os computadores dessa classe possuem N processadores, N streams de instruções e N streams de dados. A arquitetura de um computador MIMD é exiba na figura 1 (a). Cada processador tem a sua própria unidade de controle, além de uma memória local, uma ULA e seu próprio *streaming* de dados. Cada processador executa a instrução enviada por sua própria unidade de controle, porém sobre dados diferentes.

2. 2. Paralelismo no nível de dados

O paralelismo de dados trata-se da execução de uma mesma operação sobre vários elementos de dados ao mesmo tempo. Se as operações possuem o mesmo *op-code* e são independentes quanto aos dados e ao controle, então o paralelismo de dados pode ser explorado. A arquitetura de computador que dá suporte a esse tipo de paralelismo, de acordo com a taxonomia de *Flynn,* é a *Single Instruction Multiple Data* (SIMD).

Nesse modelo existem N processadores, cada um com uma memória local onde é possível armazenar dados e instruções, e uma única unidade de controle, que envia a mesma instrução para os N processadores, porém cada processador recebe seu próprio *streaming* de dado, essa arquitetura está demonstrada na figura 1 (b). Todos os processadores executam a mesma instrução, porém sobre dados diferentes. Quando não é necessário utilizar todos os processadores é possível tornar alguns inativos. Estes devem ficar no modo *idle* até receber a próxima instrução.

2.3. *Streaming SIMD Extension* (SSE)

O paralelismo de dados pode ser explorado em arquiteturas Intel através das instruções SSE [5]. Essas instruções operam sobre quatro pontos flutuantes de 32 bits ao mesmo tempo.

Elas são classificadas em dois grandes conjuntos: instruções que operam sobre *packed data* e as que operam sobre *scalar data*. Assim, temos *packed instructions* e *scalar instructions*. O novo tipo de dado definido para as instruções SSE permite armazenar quatro pontos flutuantes de 32 bits. As *packed instructions* operam sobre os quatro elementos do novo tipo de dado, como mostrado na figura 2. As instruções escalares operam somente no elemento menos significativo do tipo de dado, sem modificar os demais elementos.

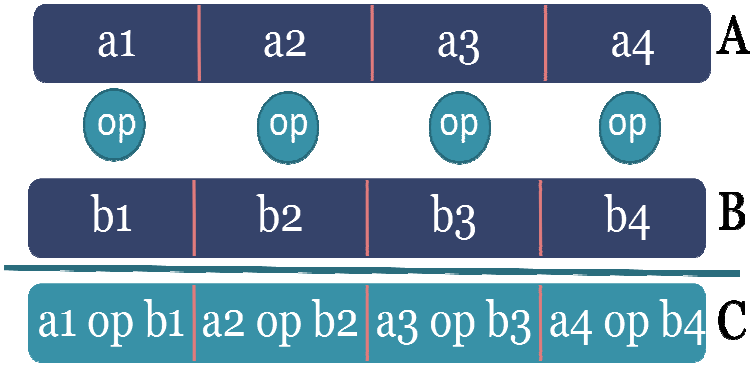

Figura 2. *Packed Instructions*

Um dos benefícios da utilização de instruções SSE é a redução no número de instruções executadas para um determinado conjunto de dados. Sem o suporte a SIMD a multiplicação de 400 pontos flutuantes por um número requer um loop sobre o conjunto de dados, executando a operação de multiplicação 400 vezes. Com SIMD, 100 multiplicações seriam necessárias para a realização da mesma tarefa, uma vez que uma multiplicação pode operar sobre quatro pontos flutuantes ao mesmo tempo. Atualmente as instruções SSE já estão na sua quarta versão, que são as instruções SSE4.

As instruções SSE podem ser exploradas de três formas. Através de código assembly, usando a biblioteca de *intrinsics* da Intel ou usando bibliotecas de C++ que definem o novo tipo de dado definido pelo SSE. As duas últimas alternativas são mais fáceis de usar, porém diminuem a performance. O código desenvolvido utilizando esse mecanismo é rápido mas não tão rápido quanto um escrito em *assembly*.

Nesse trabalho foi utilizada a biblioteca de C++. O suporte para SSE em C++ é através da classe F32vec4, que define uma abstração para o novo tipo de dado de 128 bits. Todas as operações sobre o tipo de dado estão encapsuladas na classe. Internamente a classe usa as funções da biblioteca de *intrinsics*. A figura 3 demonstra a utilização da biblioteca em C ++.

```
#define VECTOR_SIZE 4
__ declspec(align(16)) float xa[ARRAY_SIZE]
xb[ARRAY_SIZE], xc[ARRAY_SIZE];
float q;

void do_fvec_triad() {
   F32vec4 q_xmm = {q, q, q, q};
   F32vec4 * xa_mm = (F32vec4 *) & xa;
   F32vec4 * xb_mm = (F32vec4 *) & xb;
   F32vec4 * xc_mm = (F32vec4 *) & xc;
   for (int j = 0;j < (ARRAY_SIZE/VECTOR_SIZE); j++) {
        xa_xmm[j] = xb_mm[j] + q_xmm * xc_xmm[j];
   }
}
```

Figura 3. Loop utilizando a biblioteca F32vec4.

2.3. Benefícios do paralelismo

A exploração do paralelismo de dados e o paralelismo de thread permitem que haja redução no tempo de execução do programa sem que haja, no entanto, um aumento muito grande da potência consumida, trazendo assim benefícios de desempenho e a redução no consumo de energia. Entretanto, existem algumas situações em que esse ganho não é alcançado, podendo ocorrer até mesmo a degradação da performance em algumas situações. Esses problemas são causados geralmente devido a problemas de sincronização no uso do paralelismo de thread.

A exploração do paralelismo de dados ainda traz outros benefícios. O uso desse tipo de paralelismo faz com que o número de instruções de *load*/*store* sejam reduzido. Tratando-se de uma arquitetura com N processadores, ao explorar o paralelismo de dados, ao invés de se processar N operações, apenas 1 operação será processada. Essa redução no número de instruções de load/store contribui também para a redução no consumo de energia.

2.4. Soluções Alternativas

A abordagem mais comum para a redução de consumo de potência através de software é executar uma determinada tarefa utilizando o menor conjunto de instruções possível. Seguindo essa linha de raciocínio, Šimunić et al em [6] mostra que softwares mais eficientes podem ser utilizados, implicando em menos instruções sendo executadas no processador, menos instruções sendo carregadas na memória de instruções e assim redução no consumo de energia. Os resultados do experimento com uma aplicação de MP3 mostraram uma redução de 147% do consumo de energia, porém houve uma degradação de 26% no desempenho da aplicação.

O trabalho de pesquisa realizado por Sansaka et al em [7] usa o paralelismo com foco na eficiência energética. Sansaka propõe uma arquitetura chamada All Levels of Parallelism (ALP). Essa arquitetura dá suporte a várias formas de paralelismo, lnstruction Level Parallelism (ILP), Thread Level Parallelism (TLP) e Data Level Parallelism (DLP).

São propostas duas técnicas de paralelismo de dados chamadas SIMD Vector (SVector) e SIMD Stream (SStream). As técnicas SVector e SStream utilizam conceitos tanto do modelo SIMD quanto do modelo vetorial. Os resultados dos experimentos de *Sansaka* mostraram que o desempenho da ALP quando comparado ao de uma arquitetura single-thread sem SIMD, é 5 a 56 vezes melhor e que o consumo de energia é 1,7 a 17,2 vezes menor.

A principal diferença do nosso trabalho para o de *Sansaka* é que ele aborda a eficiência energética no nível de hardware enquanto nós abordamos no nível de software. Outra diferença é que ele propõe a construção de uma nova arquitetura de processadores, e o nosso trabalho explora arquiteturas já existentes.

Markus at al [8] examina a influencia do código gerado por um compilador, para processadores digitais de sinais, contendo instruções SIMD, no consumo de energia. Neste trabalho é desenvolvido um modelo de custo de energia por instrução para a arquitetura em questão, para medir o consumo de energia do programa.

Os experimentos realizados com alguns benchmark para DSP mostraram uma redução no tempo de execução entre 31%-95% e no consumo de energia de 29%-93%. Esse trabalho difere deste em diversos aspectos, arquitetura abordada, nível de abstração, aplicações examinadas, mas reforça as evidências de que o paralelismo e o

uso de instruções SIMD podem melhorar o desempenho e o consumo de energia das aplicações de software.

3. Experimento

Os experimentos realizados nesse trabalho visam analisar o impacto no desempenho e no consumo de energia do sistema, quando as aplicações exploram o paralelismo de dados e de thread. As aplicações analisadas nos experimentos fazem parte do benchmark ALPBench [2], que foram: MPEGEnc, MPEGDec e o FaceRec.

3.1 Objetivos

O objetivo do experimento foi analisar o consumo de energia e o tempo de execução das aplicações em quatro situações:

(1) Explorando apenas o paralelismo de thread

(2) Explorado apenas o paralelismo de dados

(3) Utilizando os dois tipos de paralelismo

(4) Quando nenhum tipo de paralelismo foi explorado.

3.2. Materiais do Experimento

Nesta Seção serão descritas com detalhes as três aplicações do ALPBench, como também a plataforma Intel utilizada nos experimentos.

3.2.1. MPEGEnc

A aplicação MPEGEnc converte frames de vídeo em uma stream de bits compactada, ela está dividida em sete etapas, como mostrado na figura 4, *motion estimation*, *form prediction*, *quantization*, *discrete cosine transform* (DCT), *variable length coding* (VLC), *inverse quantization* e *inverse discrete cosine transform* (IDCT).

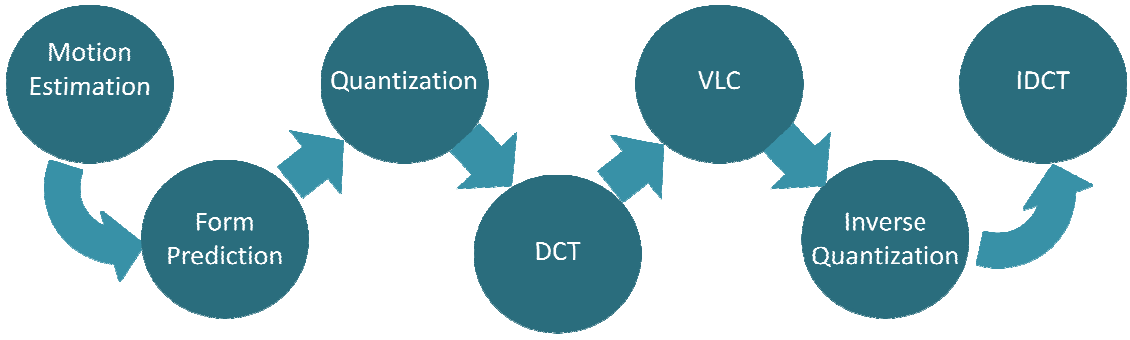

Figura 4. Fases do MPEGEnc

Nessa aplicação o paralelismo de dados é explorado através de instruções SSE em todas as fases exceto a fase de VLC. O paralelismo de thread é explorado dividindo os frames em macro-blocos que são então codificados em paralelo, no fim os resultados das threads são sincronizados.

3.2.2. MPEGDec

A aplicação MPEGDec descompacta uma stream de bits compactada. A figura 5 mostra as fases da aplicação: variable length decoding (VLD), inverse quantization, IDCT e motion compensation.

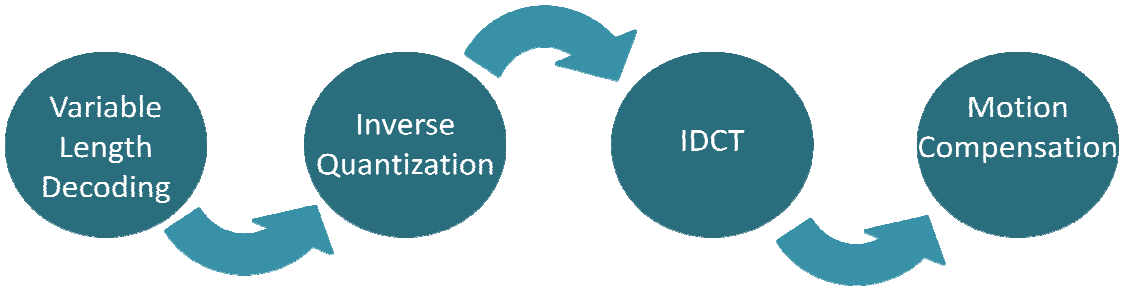

Figura 5. Fases do MPEGDec.

Para explorar o paralelismo de thread, primeiramente são identificadas as linhas contínuas que representam blocos, então essas linhas são agrupadas e distribuídas em threads para serem decodificadas paralelamente. O paralelismo de dados foi explorado nas fases de motion compensation e na de IDCT.

3.2.3. FaceRec

A aplicação FaceRec é capaz de reconhecer imagens de faces através da comparação da imagem de entrada com imagens de uma base de dados. Essa aplicação possui apenas uma fase, que é a fase de reconhecimento da imagem.

As instruções SSE foram utilizadas nos loops com operações de multiplicação, adição e subtração. O paralelismo de thread foi utilizado para calcular a proximidade da imagem de entrada às imagens da base de dados. Cada thread é responsável por computar a distância da imagem de entrada até um subconjunto da base de dados.

3.2.4. Arquitetura Intel

A arquitetura Intel utilizada foi o processador T5500 com 1.66 GHz de freqüência e 1.49 GB de memória.

3.3. Procedimento de coleta de dados

O desempenho da aplicação foi medido pelo tempo de execução da aplicação, através do comando *gprof* disponível nos sistemas operacionais Linux. O consumo de energia da aplicação foi medido através do comando *watch -n1 'cat /proc/acpi/battery/BAT0/*'* Esse comando fornece a energia consumida pelo sistema naquele determinado instante para dispositivos equipados com bateria.

O processo de coleta dos dados de consumo de energia ocorreu em duas etapas. Primeiramente foi medido o consumo de energia do sistema sem a execução da aplicação. Em seguida a aplicação foi executada em um loop infinito e então foi medido novamente o consumo de energia. A diferença entre o consumo de energia do sistema,

antes e durante a execução da aplicação foi considerada como a energia consumida devido a execução da aplicação.

As aplicações foram executadas sobre cada fonte de dados disponíveis no benchmark, e o consumo de energia e o tempo de execução das aplicações foi considerado como sendo a média das medições em cada execução.

4. Resultados

4.1. MPEGEnc

Os resultados desse experimento mostraram que o ganho de performance que o MPEGEnc pode atingir utilizando apenas o paralelismo de thread é de no máximo 31% da performance da versão seqüencial, como demonstrado no gráfico da figura 6. Explorando apenas o paralelismo de dados, através das instruções SSE, o ganho de performance alcançado foi de 69,2%, e o uso de SSE combinado com o paralelismo de threads, utilizando 2 threads, resultou em um ganho de 75,4%. Em todas as situações analisadas o paralelismo de dados obteve melhores resultados do que o paralelismo de threads para esse aplicação.

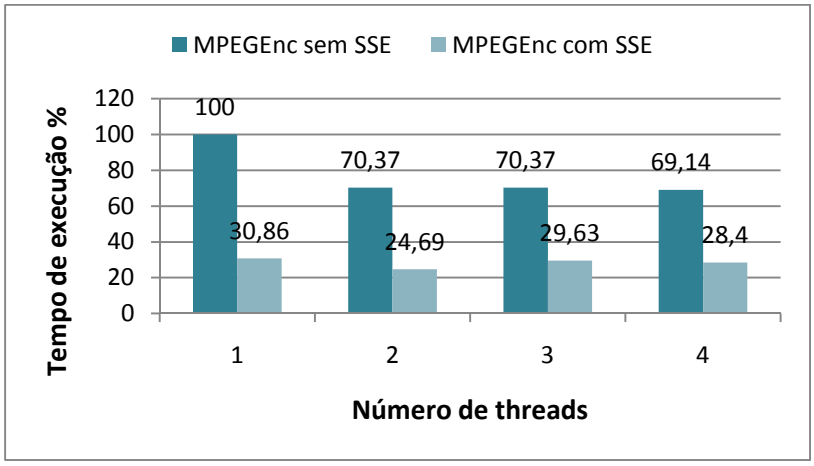

Figura 6. Resultados da performance do MPEGEnc.

A figura 7 mostra os resultados obtidos com relação ao consumo de energia. A redução no consumo de energia obtida para essa aplicação utilizando apenas o paralelismo de thread variou entre 30% a 35,6% com relação a versão seqüencial, que possui apenas 1 thread e não utiliza as instruções SSE. Utilizando as instruções SSE com apenas 1 thread a redução foi de 74,3%. O melhor resultado obtido foi com a associação do paralelismo de dados com o paralelismo de threads, utilizando 2 threads, nesta situação a redução no consumo de energia foi de 77,1%.

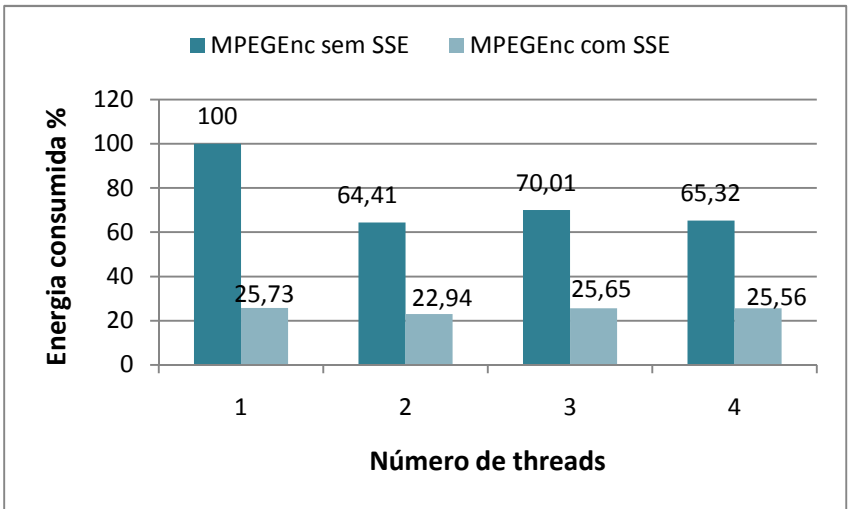

Figura 7. Resultados de consumo de energia do MPEGEnc.

4.2. MPEGDec

Analisando os resultados do MPEGDec com relação ao tempo de execução, através do gráfico da figura 8, nota-se que o ganho de performance obtido com o uso apenas do paralelismo de thread para essa aplicação foi muito pequeno em todas as situações analisadas, variando entre 1,6% e 2% em relação a versão sequencial, sem paralelismo algum. O mesmo não ocorreu com relação ao paralelismo de dados, na primeira situação onde ele é utilizado apenas com 1 thread o ganho de performance foi de 58,8% em relação a versão sequencial, e de 60,4% quando associado também ao paralelismo de thread, ao utilizar 2 threads.

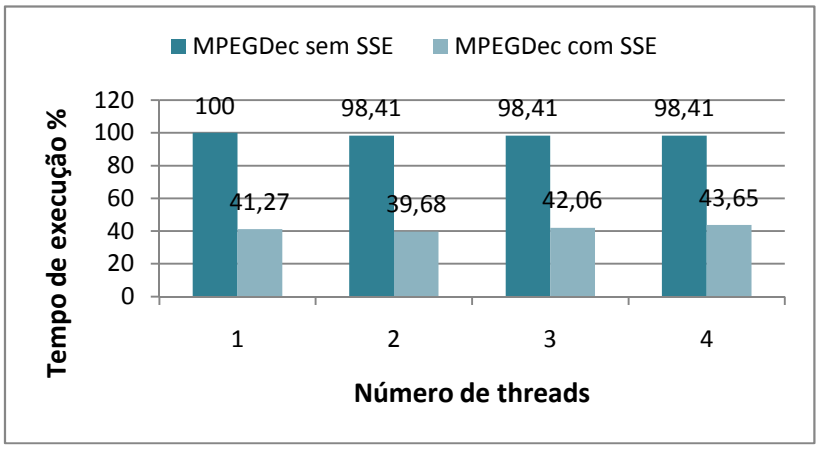

Figura 8. Resultados de performance obtidos para o MPEGDec.

Os resultados obtidos com relação ao consumo de energia podem ser vistos na figura 9. Para o MPEGDec a redução no consumo de energia utilizando apenas as instruções SSE foi de 73,4% em relação a versão sem SSE e com apenas 1 thread. Quando o uso de SSE foi associado ao uso de 2 threads a redução foi de 75%, para um número de thread maior do que 2 houve aumento no consumo de energia. Quando

utilizado apenas o paralelismo de thread, o melhor resultado obtido foi com o número de threads igual a 2, a redução nesse caso foi de 8,1%.

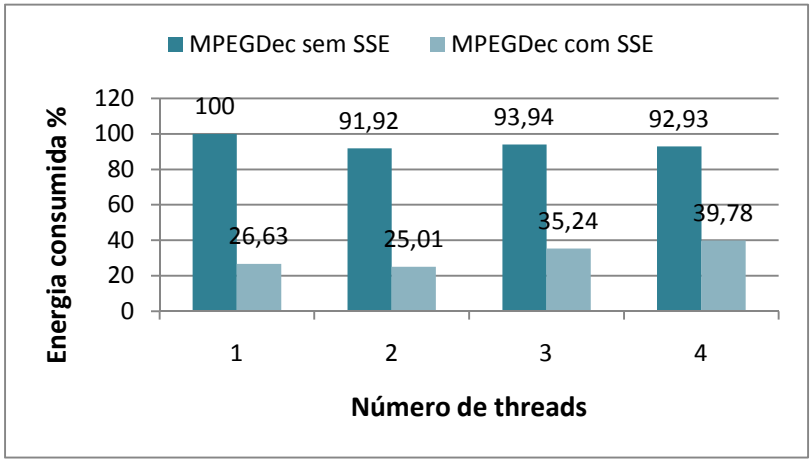

Figura 9. Resultados de consumo de energia para o MPEGDec.

4.3 FaceRec

O gráfico da figura 10 mostra os resultados obtidos com relação ao tempo de execução do FaceRec. Diferentemente das aplicações analisadas anteriormente o ganho de performance dessa aplicação com o uso de paralelismo não foi tão grande. Fazendo uso apenas do paralelismo de dados o ganho foi de 40,2% com relação a versão sequencial, esse ganho diminuiu quando ao paralelismo de dados foi associado o paralelismo de thread. Quando utilizado apenas as threads, o melhor resultado obtido foi com o número de threads igual a 2, com um ganho de 5%, com 3 threads o ganho foi de 2,5% e com 4 threads o tempo de execução foi igual ao da versão seqüencial.

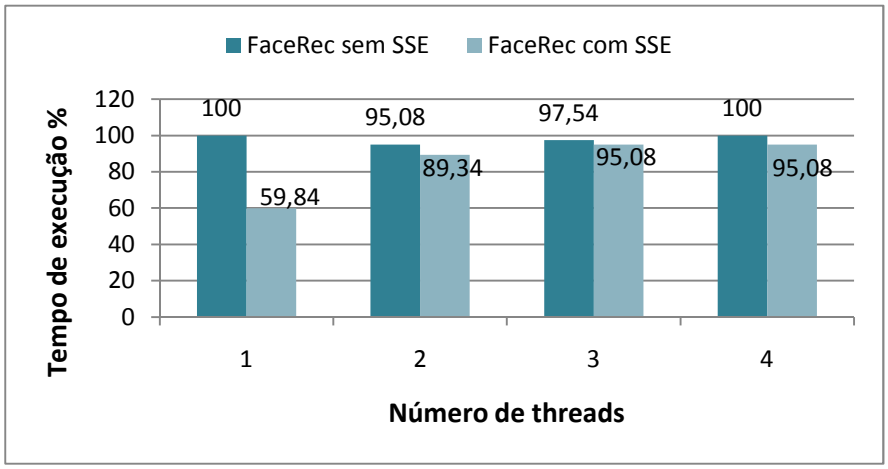

Figura 10. Resultados de performance do FaceRec.

O gráfico da figura 11 mostra os resultados obtidos com relação à energia consumida pelo FaceRec. Nessa aplicação o melhor resultado alcançado em relação à redução no consumo de energia foi utilizando apenas as instruções SSE, neste caso a

redução foi de 40,2%, quando associado ao uso de threads houve um aumento no consumo de energia. Ao utilizar o paralelismo de thread a redução no consumo de energia foi de 28% com 2 threads, 28,4% com 3 threads e de -3,7% com 4 threads, ou seja, nesse último caso o consumo de energia foi maior do que o da versão seqüencial.

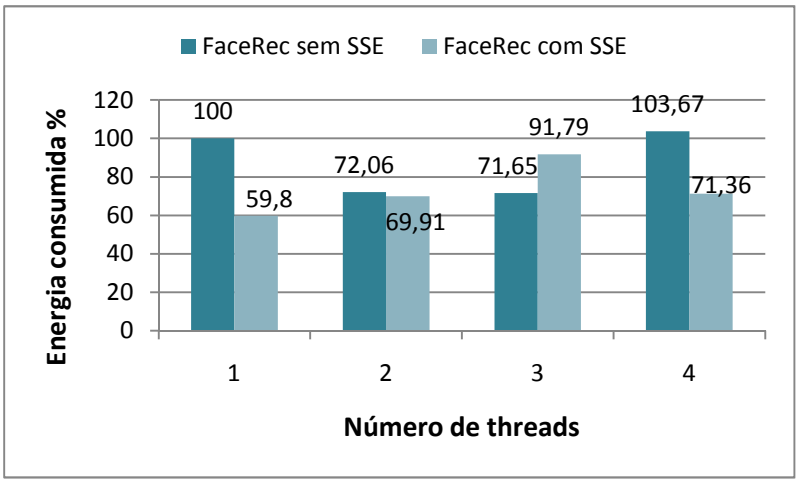

Figura 11. Resultados de consumo de energia do FaceRec.

4.4. Limitações do estudo

No entanto, não se pode concluir que o paralelismo reduz o cosumo de energia e melhora o desempenho em todos os casos. Diversos fatores podem influenciar no consumo de energia, como a arquitetura e o tipo de aplicação utilizados. Por esse motivo os resultados deste estudo estão limitados a arquitetura Intel T5500 e para aplicações multi-mídia.

Um outro limitante do estudo é o procedimento de coleta dos dados. Foram medidos os dados do sistema, como outros processos estão executando conjuntamente com as aplicações analisadas os valores captados não são precisos. Porém ainda não existe um modelo padrão para medir o consumo de energia devido ao software, apesar de varios esforços já estarem sendo feitos nesse sentido [9]. E desenvolver um modelo próprio limita ainda mais os resultados para comparação com outros estudos.

Um ponto negativo para o uso de instruções SIMD é o impacto causado na qualidade do código. A melhor forma de explorar as vantagens das instruções SSE é através do uso da programação em assembly, porém a programação de baixo nível diminui a produtividade, a detecção de erros e a manutenabilidade do código. Há impacto também na portabilidade do código, o código fica limitado a processadores que suportem instruções SSE.

5. Conclusões e Trabalhos Futuros

Neste artigo foi apresentada uma abordagem para o uso eficiencia de software do ponto de vista do desenvolvedor de software, através da paralelização de aplicações. Foram realizados experimentos com 3 aplicações do benchmark ALPBench: MPEGDec,

MPEGEnc e FaceRec. Nas três aplicações foram explorados tanto o paralelismo de thread, quanto o de dados.

Os resultados mostraram melhoras significativas no desempenho e no consumo de energia. No geral, os melhores resultados foram obtidos quando explorados os dois tipos de paralelismo. Utilizando apenas o paralelismo de thread o MPEGEnc obteve uma redução no consumo de energia de 35,6%, explorando apenas o paralelismo de dados a redução foi de 74,3% e quando explorado os dois tipos de paralelismo a redução chegou a 77,1%.

Existem duas sugestões para trabalhos futuros. A primeira é o estudo de novas bibliotecas para instruções SSE, com o intuito de torná-las mais legíveis, diminuindo assim o seu impacto na produtividade, detecção de erros e manutenabilidade do código. A segunda sugestão é a investigações de modelos para medir o consumo de energia devido ao software. Apesar de existirem alguns esforços nesta área [9], ainda são muito limitados.

Referencias

[1] Salapura V., Bickford R., Blumrich M., Bright A., Chen D., Coteus P., Gara A., Giampapa M., Gschwind M., Gupta M., Hall S., Haring R., Heidelberger P., Hoenicke D., Kopcsay G., Ohmacht M., Rand R., Takken T., Vranas P.; Power and performance optimization at the system level, 2nd conference on Computing frontiers. Italy, 2005.

[2] Li, M., Sasanka, R., Adve, S., Chen, Y., Debes, E. “The ALPBench benchmark suite for multimedia applications”. In IEEE Intl. Symp. on Workload Characterization, 2005.

[3] Tanenbaum, Andrew S., Modern operating systems – 2nd ed. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 2001.

[4] Aki, Selim G. The design and analysis of parallel algorithms. Prentice-Hall, Inc. Upper Saddle River, New Jersey, USA, 1989.

[5] Intel Corporation. The IA-32 Intel Architecture Optimization Reference Manual, 2004

[6] Šimunić, T.; Benini, L.; Micheli G.; Hans M. “Source code optimization and profiling of energy consumption in embedded systems”, Proceedings of the 13th international symposium on System synthesis, Setembro, 2000, Madrid, Espanha.

[7] Sasanka, R., Li, M., Adve, S., Chen, Y., Debes E. “ALP: Efficient support for all levels of parallelism for complex media applications,” ACM Transactions on Architecture and Code Optimization, Vol. 4, No. 1, Art. 3. Março, 2007.

[8] Markus Lorenz, Peter Marwedel, Thorsten Drager, Gehard Fettweis, Rainer Leupers. “Compiler based Exploration of DSP Energy Savings by SIMD Operations”, XXX, 2004.

[9] Júnior, Meuse Nogueira de Oliveira. “Estimativa do Consumo de Energia Devido ao Software: Uma abordagem baseada em redes de Petri coloridas”. 214p. Tese de

Doutorado, Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco. Recife, Outubro de 2006.