Computer Architecture under Energy Consumption Vision

[1]Edward David Moreno, [1]Franklin M. R. Junior, [1]Felipe dos Anjos Lima, [2]Wanderson Roger A. Dias
[1]Departamento de Ciência da Computação da Universidade Federal de Sergipe, Aracaju, Brasil
[2]Departamento de Informática do Instituto Federal de Sergipe, Aracaju, Brasil
{edwdavid, franklin.mr3, felipes1474, wradias}@gmail.com

Abstract—This paper presents a new methodology for teaching computer architecture under energy consumption vision. The educational goals of this methodology are fivefold, know and understand the integration about: (i) computer architecture of traditional platforms (for example SPARC, MIPS) (ii) computer architecture of embedded systems (ARM) (iii) modern and popular applications (embedded systems), (iv) optimizations from compilers and (v) there is a synergy among of them for performance and energy consumption point of view.

Keywords: embedded systems, benchmarks, compiler optimizations; energy and power consumption.

I. INTRODUÇÃO

Um computador pode ser um *desktop* comumente visto nas empresas e domicílios, ou qualquer outro dispositivo com capacidade de processamento usado para gerar, armazenar ou transmitir dados. Estes dispositivos são em sua maioria sistemas embarcados, que são sistemas digitais de propósitos especiais, usados dentro de outro dispositivo com funções específicas.

Com a evolução dos processadores e a redução nos custos de produção, os sistemas embarcados tornaram-se presentes em praticamente tudo que utiliza energia elétrica, desde o controlador do LCD de um DVD player até o controle de injeção de combustível em automóveis, além de terem seu potencial de processamento aumentado, o que acarreta em um aumento de consumo de energia.

Sistemas embarcados estão sendo cada vez mais usados em aplicações modernas, com ênfase em telecomunicações com dispositivos móveis e no uso de aplicativos de propósito geral [8]. Aspectos críticos no desenvolvimento e projeto desses sistemas é o tempo de execução, tamanho físico e consumo de energia uma vez que as plataformas computacionais são limitadas [8].

A problemática se inicia devido ao fato de o desempenho das baterias não ter tido um aumento significativo nos últimos anos, já que as tecnologias das mesmas já estarem próximas de seus limites, e o fato de não existir alternativas comercialmente viáveis para os próximos anos. Diante de tais limitações, existe uma grande queda de autonomia nos sistemas embarcados móveis que utilizam um poder de processamento maior.

Para suprir essa deficiência, vêm sendo estudadas formas de diminuir o consumo de energia nos sistemas embarcados, usando-se de arquiteturas que apresentam baixo consumo de energia [7], de algoritmos mais eficientes [6], uso de técnicas de compilador [5], de compressão de código [7] entre outros.

Por essa razão, neste artigo se apresenta uma metodologia de ensino que adiciona atividades aos métodos normais de ensino de arquitetura de computadores, apresentando além da tradicional medida de desempenho o tempo de execução de algoritmos comuns em sistemas embarcados, dando um início a um estudo mais abrangente sobre a correlação que existe entre o consumo de energia, o tempo de execução e os componentes arquiteturais de uma determinada plataforma.

O artigo está organizado em quatro seções, sendo que a Seção 2 apresenta os conceitos teóricos enfatizando o simulador *Sim-Panalyzer*; o *benchmark MiBench* e os tipos de otimizações comuns no compilador *gcc*, a Seção 3 detalha as novas fases de nossa metodologia, e a Seção 4 apresenta as conclusões.

II. AMBIENTE DE SIMULAÇÃO

Neste artigo foram analisados o consumo de energia com as otimizações de compilação para aplicações da categoria *security* do *MiBench*, sendo o algoritmo criptográfico simétrico AES o programa selecionado para exemplo neste artigo. Usamos duas plataformas diferentes, uma convencional (*desktop*), e outra, sistema embarcado que usa o processador ARM, e usamos simulação com o *SimpleScalar* e o *Sim-Panalyzer*, além da medição experimental das variações de Tensão e Corrente.

A. Simulador Sim-Panalyzer

Conforme AUSTIN *et al* [2, 3], o *Sim-Panalyzer* é um simulador de consumo de energia baseado no simulador arquitetural *SimpleScalar* [1]. O *SimpleScalar* é um simulador de arquitetura computacional que modela um computador virtual com CPU, memórias *caches* (de dados e instruções), memória RAM (*Random Access Memory*) e toda a hierarquia das memórias. Assim, com base neste modelo os simuladores (*SimpleScalar* e *Sim-Panalyzer*) conseguem simular programas reais executando sobre tais plataformas.

Sendo uma ferramenta baseada em um simulador de arquitetura detalhada, o *Sim-Panalyzer* consegue modelar de forma detalhada tanto a potência dinâmica quanto a potência de fuga. Para gerar os resultados detalhados o simulador é dividido em componentes que simulam partes distintas do computador, no qual simula os seguintes modelos de energia: *cache*, caminho de dados, unidade de execução, circuitos de *clock* e unidades de entrada e saída (I/O).

Os parâmetros para a geração de tais componentes do computador são configurados como entrada para o simulador *Sim-Panalyzer* que juntamente com o simulador *SimpleScalar* geram os padrões do consumo de energia da arquitetura modela. O *Sim-Panalyzer* possui vários componentes que em conjunto geram o consumo total de energia da arquitetura [3].

B. Benchmark MiBench

O *MiBench* é um *benchmark* livre, sendo basicamente implementado utilizando a linguagem de programação C [4]. Todos os programas que compõem o *MiBench* estão disponíveis em código fonte, sendo possível executar em qualquer arquitetura de microprocessador que suporte um compilador C. No *MiBench* são consideradas as seguintes classes de instrução: controle, aritmética de inteiro, aritmética de ponto flutuante e acesso à memória.

Conforme [7], o *MiBench* é dividido em seis categorias, sendo que nas categorias de *Telecommunications* e *Security* todas as aplicações têm mais de 50% de operações inteiras na ULA (*Arithmetic Logic Unit*). A categoria *Consumer Devices* tem mais operações de acesso à memória por causa do processamento de grandes arquivos de imagem e áudio. A categoria *Office Automation* faz uso intenso de operações de controle e acesso à memória. Os programas dessa categoria usam muitas chamadas de funções para bibliotecas de *string* para manipular os textos, o que causa o uso de várias instruções de desvio. Além disso, os textos ocupam bastante espaço na memória, o que requer muitas instruções de acesso à memória. A categoria *Automotive and Industrial Control* faz o uso intenso de todas as classes de instrução devido a sua aplicabilidade e por fim, a categoria *Networking* faz muito uso de operações aritméticas de inteiros, pontos flutuantes e acesso à memória, devido às tabelas de IPs para roteamento em rede, ordenação de dados e outros.

C. Otimizações de Compilador

As otimizações de compilação são processos de melhoria do código final, que dá origem ao código executável, conforme [5]. Por meio de trocas, substituição e até exclusão de estruturas, o compilador pode tornar a execução do programa mais rápido, consumindo menos memória. Os códigos executáveis gerados através das otimizações são semanticamente equivalentes aos códigos originais, isto é, eles comportam-se exatamente iguais para as mesmas entradas [5]. Neste trabalho, utilizamos o compilador *gcc* versão 4.4.5 aplicando as seguintes otimizações: -O0 (base para comparações, sem otimização nenhuma), -O1,-O2, -O3 e -Ip na compilação dos *MiBenchs*. Os efeitos de cada tipo de otimização de compilação podem ser observados em mais detalhes em [5, 6].

III. METODOLOGIA COM CONSUMO DE ENERGIA

Os objetivos educacionais desta metodologia são quatro, a saber: (i) entender detalhes da arquitetura de um sistema computacional, do ponto de vista de desempenho e de consumo de energia; (ii) visualizar impacto de algumas otimizações oferecidas pelo compilador *gcc*, (iii) visualizar o comportamento de sistemas embarcados em aplicações modernas, e (iv) entender a sinergia que existe em todos esses aspectos com o desempenho e o consumo de energia.

Para isso, apresentamos uma metodologia com as seis fases seguintes:

Fase I. Foco no Tempo de Execução e Simulação. Inicialmente, nós focamos no conteúdo da disciplina de "arquitetura de computadores" em temas tradicionais (como processador, memória, *cache*, elementos de I/O, rede de comunicação, arquiteturas de conjunto de instruções, RISC, CISC, *pipeline* e outros aspectos da microarquitetura, com foco na avaliação de desempenho usando o tempo processamento, uso de memória e etc.).

Nesta primeira fase dedicamos às 60 horas tradicionais designadas a disciplina de arquitetura de computadores e concentramos na análise de desempenho, focando na medição do tempo de execução das aplicações. Nesta fase, costuma-se usar o simulador *SimpleScalar* [1], que permite a mudança de parâmetros arquiteturais de processadores de 32 *bits*, como o SPARC, MIPS, PowerPC e ARM. Este simulador permite que os alunos consigam executar aplicativos escritos usando a linguagem C, e eles podem ver como o tempo de execução das aplicações é medido em ciclos de processador, é gasto nos diferentes componentes da arquitetura alvo.

Nesta primeira fase focamos em arquiteturas SPARC e/ou MIPS. Durante esta fase, nós simulamos situações diferentes, tais como a mudança do tamanho do barramento, o tamanho do *pipeline*, a frequência de operação do processador, a latência da memória, e principalmente, observamos as diferentes configurações da memória *cache* e verificamos a importância deste componente na execução das aplicações. Nesta fase analisamos aplicações científicas e de engenharia, usando programas do *benchmark* SPEC, por exemplo, e focamos tanto no tempo de execução quanto na energia.

Em nossa Universidade esta disciplina foi mudada para ter 90 horas. Por esse motivo, às 30 horas restantes são dedicadas às outras fases da metodologia apresentadas neste artigo.

Fase II. Uso do Osciloscópio – Foco no Tempo e Energia. A medição real do consumo energético foi aplicada em um *desktop* com o processador Intel Celeron de 2.2 GHz, 512 *MBytes* de memória RAM, HD de 40 *GBytes* e placa mãe Foxconn 650M02-G-6, e sistema operacional Linux Ubuntu 9.10, usando um osciloscópio da marca Owon modelo PDS5022S.

Para a medição do consumo de energia e do desempenho foi usado o osciloscópio para medir a corrente consumida pelo processador ao executar um algoritmo desejado. Para medir a corrente, foi inserido um resistor de 0.333 *Ohms* conectado em série com o cabo de alimentação ATX12V [6] da placa mãe e foi medida a diferença de tensão da entrada e saída do resistor de *shunt* (Vr) que foi inserido em série no circuito da placa principal, conforme se observa na Figura 1. O correto seria colocar seria colocar o resistor em serie a cada componente, para medir o consumo desse componente, mas não foi possível pois em um PC ou em uma placa pronta não é possível isolar cada componente. Por esse motivo, nesta seção a medida corresponde ao consumo da placa.

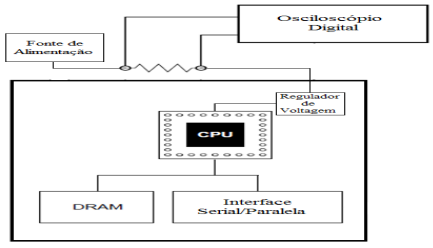


Figura 1. Uso de osciloscópio para medir consumo

Com esses dados pode-se calcular a corrente e potência consumida pelo processador. Lembrando que: (i) Corrente (I) = Variação de Tensão (Vr) dividida pela resistência (R) colocada no circuito; (ii) Potência (*Watts*) = Corrente (I) vezes

a tensão de saída (Vplaca) em (*Volts*). O cálculo do consumo de energia (E) do dispositivo (*desktop*) usado durante a execução do algoritmo pode ser feito ao analisar o gráfico gerado pelo osciloscópio e multiplicar o tempo gasto na execução do algoritmo (T) pela potência média (P), calculada com as variáveis: tensão (V) e corrente (I) reais consumidas pela plataforma. Lembrarmos que **E = P * T**.

No uso desta metodologia fizemos também testes com o algoritmo criptográfico simétrico AES, melhor conhecido como *Rijndael*, e foi possível perceber as mudanças no consumo de energia para diferentes tamanhos de arquivos e chave.

Desta forma, neste artigo apresentamos alguns resultados, obtidos para chaves de 128, 192 e 256 *bits* (relacionadas na Figura 2 como Chave1, Chave2 e Chave3) e arquivos de entrada de tamanhos diferentes, sendo que Entrada1 e a Entrada2 são arquivos de puro texto, de tamanhos 100 *KBytes* e 500 *KBytes*, respectivamente.

Os alunos conseguiram observar que para as configurações do algoritmo (entrada do arquivo e tamanho da chave) se obtém diferentes consumos de energia. Os alunos também observaram que o processo de cifrar o arquivo demanda mais energia do que o processo de decifrar o arquivo.

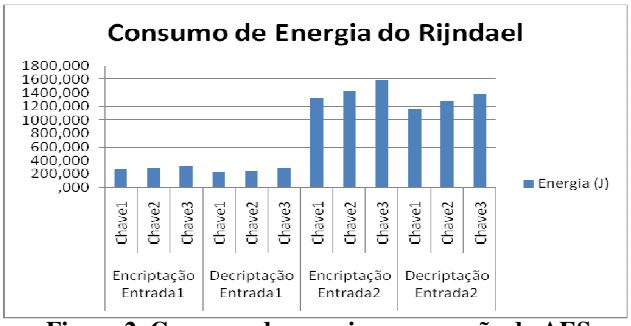
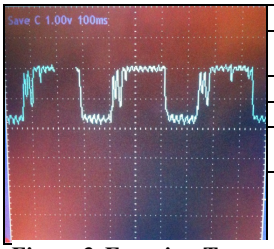


Figura 2. Consumo de energia na execução do AES

Descobrimos que os alunos se sentiram mais atraídos pela área (arquitetura de computadores e sistemas embarcados), quando conseguem visualizar e verificar que qualquer movimento do mouse, ou qualquer mudança na aplicação, ou em quaisquer dos componentes da arquitetura, pode influenciar não somente o tempo de processamento, mas também o consumo de energia (ver Figura 3, para arquivo de entrada *Money.mp3* de 4,5 *MBytes*). Os alunos quiseram descobrir a causa desse impacto. Os mesmos apresentaram-se motivados e estimulados às investigações relacionadas à área, e desejam adquirir mais conhecimentos, aumentando a curiosidade científica.



Arquivo	Medição
Tempo entre execuções	0,2 s
Tempo por execução	0,13 s
Variação de Tensão	1,6 V
Potência média durante a execução	48,43 W
Consumo	6,30 J

Figura 3. Energia e Tempo do AES medido no osciloscópio

Fase III. Selecionar uma Plataforma Embarcada. A terceira fase da metodologia, associada à mudança estratégica apontada como contribuição deste artigo consiste em escolher uma plataforma embarcada, e estudar seu ISA (*Instruction Set Achitecture*) e analisarmos o consumo de energia das aplicações alvo (fase IV) quando executada nessa plataforma.

Com o aumento do uso de sistemas móveis e sistemas embarcados, e o uso cada vez maior de processadores ARM nestes sistemas, bem como a possibilidade oferecida pelo simulador *Sim-Panalyzer*, decidimos então visualizar como acontece o consumo total de energia da aplicação quando se executa na arquitetura selecionada, e também qual o consumo de energia em cada componente específico da arquitetura.

O simulador *Sim-Panalyzer*, além de fornecer o consumo total, também oferece consumo médio de energia, o consumo de pico, tanto para a arquitetura em geral quanto para cada um dos componentes da arquitetura. Ele também fornece o tempo de execução em ciclos do processador. Assim, temos a energia e a potência consumida.

Fase IV. Selecionar Benchmark de Interesse. A quarta fase da metodologia consiste em selecionar um grupo de aplicações, as quais sejam de interesse para os alunos. Nesta fase, temos várias opções sempre usando programas bem conhecidos e utilizados na comunidade.

Para isso, recomendamos o uso de *benchmarks*. Existem várias opções, como por exemplo, *SPEC* (para aplicações científicas e de engenharia), *Linpack* que é para cálculos matriciais e computação com vetores, *NAS* (aplicações paralelas e de alto desempenho), *EEMBC* (para sistemas embarcados), *MediaBench* que é para aplicações multimídia, o *MiBench* (para sistemas embarcados) o *BioBench* (aplicações de genética). Outros *benchmarks* estão disponíveis para estressar a rede usando a pilha TCP/IP, entrada, saída de dados e outras tarefas específicas, entre outros.

Neste artigo, nos concentramos no *MiBench*, uma vez que queremos motivar os alunos com as aplicações mais próximas à sua realidade, e que estão em várias outras aplicações, as quais os alunos estão em constante contato no cotidiano. Assim, temos selecionado programas usados em sistemas embarcados, como explicados na seção anterior. Este *benchmark* apresenta um conjunto de 30 programas que são divididos em seis categorias (telecomunicações, segurança, automação, setor automotivo e industrial, redes de computadores e dispositivos de consumo).

Temos observado que eles se sentiram mais motivados com este tipo de programas do que com programas científicos como *SPEC* ou *Linpack*. Eles sentem interesse com aplicações na área de segurança e criptografia, aplicativos multimídia, programas descritos na categoria de dispositivos de consumo do *MiBench* (com algoritmos de reconhecimento de voz, processamento de imagem e etc.).

A Figura 4, mostra em detalhes o consumo de energia de cada componente (que faz parte da arquitetura do microprocessador ARM) quando executado o *MiBench* AES.

Cada um desses componentes arquiteturais possui uma importante parcela e assim contribuem com o total de energia gasta pela execução de um algoritmo.

Neste trabalho além de verificar os benefícios gerais das otimizações presentes no compilador *gcc*, se apresentam quais os componentes arquiteturais da plataforma ARM que contribuíram para esses benefícios. Os alunos conseguiram visualizar que os componentes mais relevantes no consumo de energia são: *itlb* (tabela de instruções do TLB), *dtlb* (tabela de dados do TLB) e *bimod* (previsor de desvio), assim como

aqueles relacionados com o uso do *cache*, os componentes *il1 (cache* de instruções nível 1) e o *dl1 (cache* de dados nível 1).

Outro componente que também é altamente estimulado são *aio (*barramento de endereço da unidade de I/O) e o *dio (*barramento de dados da unidade de I/O). Além disso, componentes com alto uso e, portanto que afetam o desempenho são o *clock* (relógio gerador de *clock* do sistema) e o *uarch* (microarquitetura, a maneira com que o ISA é implementado em um processador).

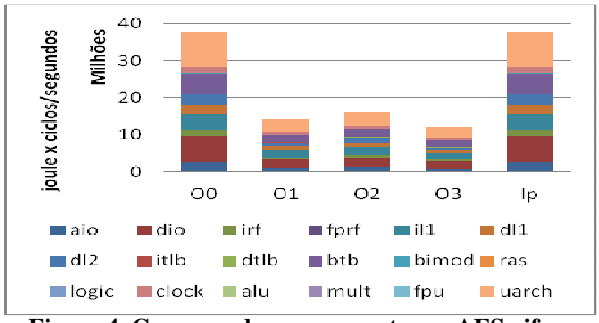


Figura 4. Consumo dos componentes no AES-cifragem com 5 KBytes

Após a realização de simulações da execução do algoritmo criptográfico AES para criptografar um arquivo com 5 *KBytes*, percebemos que as otimizações -O1 e -O3 apresentaram melhores desempenhos e um consumo de energia abaixo de 400J (ver Figura 5). Os alunos verificaram também que a otimização -O3 foi a que se destacou fazendo o algoritmo gastar menos energia, e um dos motivos para isso foi à utilização de mecanismos para gerenciar a memória durante a soma aritmética de valores.

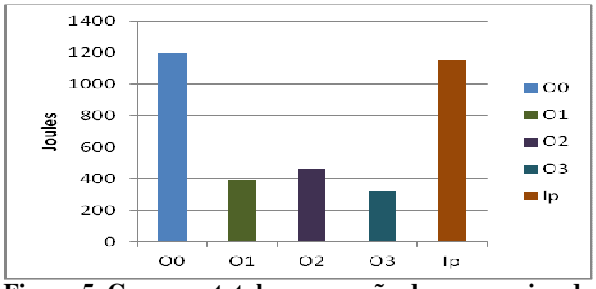


Figura 5. Consumo total na execução de um arquivo de 5 KBytes no AES

Fase V. Impacto das Otimizações de Compilador. Após a seleção de plataformas e aplicações, começamos a quinta fase, na qual mostramos que há otimizações prontas do compilador, com o uso das *flags* -O1, -O2, -O3 e -lp, e explicamos o que cada uma delas faz, e sua correlação com componentes arquiteturais descritos na fase IV, como por exemplo, menor uso de memória, menor uso do barramento, menor número de acessos de I/O e etc. E aplicamos essas otimizações nos programas, de modo automático pois é somente gerar um novo código executável, com as opções de compilação desejadas. Isso se faz usando as *flags* no momento da compilação. Imediatamente executamos novamente o aplicativo e verificamos seu impacto sobre o consumo total de energia e tempo de execução, visualizando o efeito, principalmente, o consumo de energia (ver Figura 5).

Fase VI. Análise da Integração e Sinergia. Após o uso das fases II a V, pedimos para os alunos realizarem o levantamento do consumo de energia, visualizando os principais componentes da arquitetura que ajudam na redução do consumo total. Solicitamos aos alunos para discutir esse efeito com otimizações do compilador e componentes de arquitetura que mais alteram o comportamento da energia.

É interessante que eles percebem que qualquer ação no código tem efeito não somente no tempo de execução, mas também na potência e energia consumida, e então, eles demonstraram interesse em entender o porquê de cada situação na programação, do ponto de vista da aplicação, da otimização do compilador e quais suas relações com os componentes arquiteturais.

IV. CONCLUSÕES

O trabalho apresentou uma nova metodologia de fácil implementação, que usa seis (6) fases: (i) Simulação com foco no tempo de execução; (ii) Experimentação com uso do osciloscópio – foco no tempo, potência e energia; (iii) Selecionar uma plataforma embarcada, (iv) Selecionar *benchmark* de interesse; (v) Impacto das otimizações de compilador e (vi) Análise da integração e sinergia.

No trabalho, usamos os simuladores *SimpleScalar* e *Sim-Panalyzer*, o *benchmark MiBench*, plataformas tradicionais de um *desktop*, e um sistema embarcado com o processador ARM. Além de usarmos simulação, incentivamos os alunos a medirem o tempo de execução, a potência (tensão e corrente) e o consumo de energia com auxilio de um osciloscópio.

Esta metodologia permite identificar e compreender que existe uma sinergia entre a arquitetura do computador, a plataforma alvo, o uso do compilador, a programação da aplicação, e a respectiva avaliação de desempenho (medido não só com o tempo de execução, mas também com o consumo de energia).

Percebemos que usando esta metodologia, os alunos mostraram maior interesse pela área de arquitetura de computadores e hardware, melhorando o desempenho acadêmico. Além disso, em nossa Universidade aumentou o número de alunos que anseiam fazer Iniciação Científica e Trabalhos de Conclusão de Curso com os docentes da área.

REFERÊNCIAS

[1] D. Burger, and T. M. Austin, "The SimpleScalar Tool Set, version 2.0". ACM SIGARCH Computer Architecture News, 25(3):13-25, June 1997.

[2] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling". IEEE Computer, 35(2):59-67, Feb. 2002.

[3] T. Austin, T. Mudge, and D. Grunwald, "Sim-Panalyzer: The SimpleScalar-ARM Power Modeling Project". Access in 07/2013.

[4] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite". In Proc. of the IEEE 4th Annual Workshop on Workload Characterization, USA, pages 3-14, December 2001.

[5] M. Kandemir, N. Vijaykrishnam, M. J. Irwin, and W. Ye, "Influence of compiler optimizations on system power". In IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 9(6):801-804, Dec. 2001.

[6] J.S. Seng, and D. M. Tullsen, "The Effect of Compiler Optimizations on Pentium 4 Power Consumption", In Proc. of the Seventh INTERACT'03, Anaheim, California, USA, p. 51-53, Feb. 2003.

[7] W. R. A. Dias, E. D. Moreno, and R. S. Barreto, "Architectural Characterization and Code Compression in Embedded Processors". IEEE Latin America Transactions, 10(4):1865-1873, June 2012.

[8] A. S. Oliveira, and F. S. Andrade, "Sistemas Embarcados - Hardware e Firmware na Prática". São Paulo, Brazil, Érica, 2006, 316p.