Estimativa de Consumo de Energia em Nível de Instrução para Processadores Modelados em ArchC

Josué Tzan Hsin Ma, Rodolfo Azevedo
Instituto de Computação
Universidade Estadual de Campinas
Campinas, SP, Brasil
josuema@gmail.com, rodolfo@ic.unicamp.br

Resumo

A constante redução do tamanho e o conseqüente aumento do número de transistores em um mesmo chip faz com que a potência dissipada pelos circuitos digitais aumente exponencialmente. Esse fato, combinado com a crescente demanda por dispositivos portáteis, têm levado à uma crescente preocupação quanto ao consumo de energia. Quanto mais potência é dissipada mais calor é gerado e mais energia é gasta com o seu resfriamento. Como resultado, projetistas estão considerando cada vez mais o impacto do consumo em suas decisões. Atualmente, Linguagens de Descrição de Arquiteturas (ADLs) têm sido utilizadas para projetar novos processadores. Essas linguagens descrevem o comportamento da arquitetura para cada ação ou instrução. ADLs, além de diminuirem o tempo de projeto, são úteis para descobrir problemas arquiteturais em um nível mais elevado. Nesse trabalho, foi desenvolvida uma ferramenta de estimativa de consumo de energia em nível de instrução utilizando-se como base a ADL ArchC e, como estudo de caso, um processador (ISA) SPARCv8. Como resultado do uso da ferramenta desenvolvida, uma simulação de um programa com estimativa de consumo de energia pode ser realizada 100 vezes mais rápida, na média, em relação ao fluxo tradicional.

1 Introdução

O projeto de sistemas computacionais complexos envolve diversas fases e uma delas, a análise de consumo de energia, está se tornando cada vez mais crítica. No modelo tradicional, esta análise é postergada até a primeira versão do sistema sintetizável, quando resultados insatisfatórios de consumo podem ocasionar um grande retrabalho no sistema. Após o surgimento de simuladores completos de sistemas, foi possível realizar avaliações de desempenho bem mais cedo no fluxo de projeto, mas ainda era difícil a simulação para estimativa de consumo de energia, uma vez que o consumo só era estimado após demorados processos de síntese lógica e simulação.

A evolução do desenvolvimento de simuladores levou a sua segmentação em áreas de especificidades. Assim, é comum hoje em dia encontrarmos processadores descritos através de Linguagens de Descrição de Arquiteturas (ADLs). Estas linguagens permitem descrever com variável nível de detalhe, o funcionamento de um processador. Um dos resultados possíveis desta descrição é a geração de um simulador para o processador que pode ser integrado a um simulador completo do sistema. Assim, o projetista não terá que gastar tanto tempo preparando um simulador para cada processador necessário.

Modelos de estimativa de consumo de energia também tiveram seu nível de abstração elevado. Ao invés de exigirem caras simulações em nível de transistores, hoje em dia são desenvolvidos modelos para cada um dos possíveis componentes do sistema: memórias, barramentos, *IPs* e processadores. No caso específico de processadores, um dos métodos mais utilizados é a análise de consumo por instrução executada.

Este artigo apresenta uma ferramenta para estimativa de consumo de energia de processadores, baseada na Linguagem para Descrição de Arquitetura ArchC e no modelo de estimativa de consumo por instruções. Um estudo de caso foi realizado com o processador Leon (SPARCv8) com resultados obtidos até 100 vezes mais rápido que no método convencional.

O restante deste artigo está organizado da seguinte forma: A seção 2 descreve os conceitos básicos sobre estimativas de consumo baseadas em instruções, a seção 3 ilustra conceitos básicos da linguagem ArchC. O corpo principal deste trabalho é descrito nas seções 4 e 5. Os resultados são mostrados na seção 6 e a seção 7 conclui o trabalho.

2. Estimativa de Energia em Nível de Instrução

Os microprocessadores atuais são sistemas extremamente complexos constituídos por vários blocos funcionais internos. No entanto, essa complexidade está escondida por detrás de uma abstração muito mais simples, o conjunto de instruções do processador. Tendo essa idéia como base, é bastante lógico considerar o consumo de energia destas instruções, já que cada instrução envolve processos e atividades específicas por entre os vários blocos internos do processador.

O problema de estimativa de consumo de energia em nível de instrução começou a ser estudado em [9]. Desde a sua proposta até os dias atuais, o problema tem sido abordado de muitas maneiras e propósitos diferentes.

A maneira mais utilizada é a execução de cada uma das instruções repetidamente, os chamados *stimuli*, do qual se obtém o consumo médio de cada instrução. Durante esse processo, a corrente elétrica média é medida e guardada em uma tabela de custos base e pode ser estimada também por classes de instruções.

Tiwari et al. [9, 10] explorou a estimativa de consumo de energia em nível de instruções utilizando modelos simples como a média de energia consumida por instrução, derivada de medidas experimentais. Em seus trabalhos também foi considerada a energia consumida entre as instruções, o consumo intra-instruções. Eles também propuseram um método simples e prático para medir a corrente. Ao testar um programa contendo um conjunto repetido de instruções em um processador cuja alimentação foi isolada do resto do sistema, a forma da corrente é captada e, sendo periódica, sua média pode ser facilmente calculada. Essa técnica muito simples foi utilizada para obter dados sobre o consumo de um processador 486DX2. Variações dessa técnica foram utilizadas para medir o consumo de energia de outros processadores como o ARM7TDMI e o Motorola DSP 56100 em outros projetos.

Russel [8] considerou uma média de consumo de energia igual para todas as instruções do ISA baseado na observação de que, para certas classes de processadores, a diferença de consumo de energia entre as instruções era mínima.

Em [4] os autores propuseram estimar o consumo de energia considerando apenas as instruções que são executadas após um NOP.

O custo base de cada instrução, no entanto, pode ser camuflado por vários fatores da arquitetura do processador. Cada um deles gera um valor que difere do custo base, podendo ser o efeito de um *cache-miss* ou um efeito de um *stall* de sistema ou interrupção, entre outros. Além desses, são discutidos em [6] o efeito causado por diferentes registradores fontes e destinos, o efeito dos valores de operandos, o efeito das instruções condicionais e o efeito dos diferentes tipos de endereçamentos de uma mesma instrução.

A maior dificuldade para ser implementada em cada tipo de processador é o tamanho do conjunto de instruções de cada um deles. Isto decorre do fato de estar em um nível mais alto de abstração, pois normalmente um processador comum possui centenas de instruções. No nível mais baixo, seria possível isolar cada um dos blocos funcionais utilizados pelo núcleo do processador, medir o consumo em cada um deles e saber quais deles são utilizados por cada instrução.

Em suma, uma vez construída a tabela de custos base, o cálculo da estimativa de consumo de energia em nível de instrução pode ser simplificado e dado pela fórmula 1

$$P = \sum (n_i.X_i) \tag{1}$$

onde n_i representa o número de vezes que a instrução foi executada, X_i o consumo médio (custo-base) dessa instrução e N o número total de instruções executadas.

Em [5] é descrita uma série de propriedades necessárias para que o modelo de consumo de energia possa ser utilizado como base para otimização em código de programa. São elas:

- **Precisão (Accuracy):** O modelo deve ser capaz de estimar o consumo de energia precisamente;
- **Simplista (Simplicity):** O modelo deve ser construído utilizando propriedades simples, visíveis ao nível de instruções;
- **Responsável (Accountability):** O modelo deve ser capaz de identificar a significância de cada fator que afeta o consumo de energia;
- **Reutilizável (Retargetability):** O modelo deve ser generalista o suficiente para poder ser utilizado em vários processadores.

Além destas características, é necessário observar certos fatores para minimizar o consumo de energia. Esses fatores podem ser garantidos utilizando um compilador especializado ou otimizado para esse fim. São eles:

- **Instruções utilizadas:** Durante a fase de seleção de código, o modelo precisa escolher seqüências de instruções de uma série de alternativas. Além disso, para o modelo ser altamente preciso, é necessário que todas as instruções sejam incluídas nos testes.
- **Fluxo das instruções:** Quando duas instruções subseqüentes requerem o uso de unidades funcionais diferentes, estas são ativadas ou desativadas de acordo com a necessidade. A energia consumida por essas mudanças de estado podem ser minimizadas otimizando-se o agendamento, ou seqüência das instruções de maneira a utilizar uma mesma unidade

funcional pelo período mais longo possível. Esse tipo de melhoramento precisa levar em conta as dependências dessas instruções.

- **Hierarquia de memória:** Em sistemas com duas ou mais memórias, o compilador precisa decidir a alocação dos objetos na memória. Essas diferenças devem ser consideradas quando o consumo de energia for medido.
- **Parâmetros do modelo:** Para que os requerimentos acima sejam alcançados, não é possível apenas utilizar dados de planilhas. É necessário que se façam medições. Estas medições devem ser simples o suficiente para serem feitas sem que conhecimentos da estrutura interna sejam requeridos.

3. ArchC

ArchC [7] é uma ADL, linguagem de descrição de arquiteturas, desenvolvida no Laboratório de Sistemas de Computação do Instituto da Computação da Universidade Estadual de Campinas.

ArchC é baseado em uma linguagem de descrição de sistemas, SystemC. Com esta ferramenta é possível aos projetistas avaliarem rapidamente novas idéias na área de arquitetura de computadores como: projeto de processadores, hierarquia de memória e outros aspectos. Dentre os aspectos que podem ser descritos em ArchC está o comportamento de um conjunto de instruções.

O algoritmo 3.1 mostra um exemplo da descrição comportamental de uma instrução modelada em ArchC. Para o processador (ISA) SPARCv8 a instrução *add* escreve no registrador *rd* a soma dos registradores *rs1* e *rs2* e atualiza o PC.

Algoritmo 3.1 Comportamento da instrução *add* em ArchC

```
void ac_behavior(add_reg)
{
  writeReg(rd, readReg(rs1) +
               readReg(rs2));
  update_pc(0,0,0,0,0);
}
```

Após descrever todas as instruções do processador, o projetista pode utilizar uma das ferramentas de ArchC para gerar um simulador em SystemC para o processador em questão e, com este simulador, executar programas diversos para a arquitetura.

A escolha de uso da ArchC neste projeto se deve a vários fatores como, por exemplo, o fato de ela ser desenvolvida nessa instituição e estar disponível para implementações de novos recursos, o que a torna diferente de seus atuais concorrentes e, como trabalha em um nível mais alto que simuladores RTL ou gate-level, as simulações em ArchC também são mais rápidas, economizando horas de trabalho. Além disto, cabe destacar que o simulador gerado por ArchC será utilizado apenas na fase final da metodologia, não afetando em nada o consumo das instruções que serão medidos do modelo RTL.

4. Integração de Estimativa de Consumo de Energia ao ArchC

Nesta seção são apresentados os aspectos práticos desse trabalho como o processador e as ferramentas utilizadas, os meios de captação de dados, a metodologia utilizada e a ferramenta desenvolvida. O processador escolhido para o estudo de caso foi o Leon, nas versões 2 e 3, que é o um processador (ISA) SPARCv8 já testado em diversas aplicações práticas. As duas versões do processador foram escritas em VHDL.

4.1. Processador

Para esse projeto foram utilizadas duas implementações de um processador SPARCv8[1][11]. O SPARC é um processador RISC[2] de 32 bits e em sua versão mínima possui 72 instruções implementadas.

Esse processador foi escolhido por já haver um modelo funcional em ArchC e possuir implementações em VHDL como o Leon2 e Leon3 sob licença LGPL.

Existe ainda um compilador específico para as arquiteturas Leon2 e Leon3 disponibilizado por Gaisler[3]. Esse suporte torna a escolha desses processadores bastante vantajosa pois não é necessário gastar tempo para modificar um compilador com as especificações dos processadores de teste. Ambos utilizam uma versão alterada do sparc-elf-gcc.

4.2. Ferramentas

Para a estimativa de energia, foram utilizadas ferramentas para FPGA e ASIC. Para FPGA, foi utilizada a ferramenta XPower da Xilinx, foca nas FPGAs Virtex4 e Spartan3. Para ASIC, foi utilizada a ferramenta PowerCompiler da Synopsys.

5. Metodologia

Nesse projeto foram utilizados os métodos propostos por [2, 6, 8, 9, 10] que utilizam programas alvo para gerar os estímulos necessários. Cada programa, criado especificamente para esse trabalho, contém um núcleo com 100

[1]SPARC: Scalable Processor ARChitecture Version 8.
[2]RISC: Reduced Instruction Set Computer.

instruções alvo, e esse núcleo é executado 1.000 vezes, totalizando 100.000 instruções alvo executadas por simulação. Um gráfico validando o número de instruções executadas por esses programas pode ser visto na figura 2 da próxima seção. Este número é grande o suficiente para que seja possível descartar o *overhead* do laço.

As instruções alvo para esse projeto foram escolhidas após simulação dos benchmarks Mediabench e MiBench. Após simular os programas com o ArchC 2.0, com suporte a estatísticas ligado e essas guardadas em arquivos, o número de instruções utilizadas foram colocadas em uma tabela, somadas e classificadas em ordem decrescente de uso. A partir daí foram escolhidas as instruções que mais foram executadas. Parte da tabela utilizada na escolha das instruções é mostrada no texto 5.1. Algumas instruções não são executadas nenhuma vez por nenhum dos programas e, portanto, não constam da tabela de consumo que será gerada neste trabalho. No entanto, basta seguir os procedimentos descritos anteriormente que o consumo destas instruções também pode ser obtido. Como a metodologia utilizada capta o consumo global, o *leakage power* também já está sendo considerado para cada instrução.

Texto 5.1 Parte da somatória dados da saída estatística do ArchC após simulação do MediaBench e MiBench em ordem decrescente.

```
[ArchC 2.0] Printing statistics from instruction or_reg:    3098659546
[ArchC 2.0] Printing statistics from instruction or_imm:    1694779916
[ArchC 2.0] Printing statistics from instruction ld_imm:    1510735531
[ArchC 2.0] Printing statistics from instruction add_reg:   1509612327
[ArchC 2.0] Printing statistics from instruction subcc_imm: 1361816541
[ArchC 2.0] Printing statistics from instruction srl_imm:   1351721320
[ArchC 2.0] Printing statistics from instruction add_imm:   1332784107

(...)
```

Após a escolha das instruções e confecção dos programas, esses foram então compilados utilizando o sparc-elf-gcc disponibilizado pela Gaisler. Nesse compilador foi utilizada a opção '-mv8' para gerar instruções da versão 8 do SPARC. Caso essa opção não seja utilizada, o compilador utiliza a versão 7 como padrão. A versão 8 do SPARC dá suporte às novas instruções diretamente. Um dos exemplos é o suporte à instrução de multiplicação ('mul') utilizada. Depois de compilados, foram simulados no Leon2 e Leon3 em tecnologias diferentes, como Virtex4, Spartan3 e em TSMC .25μm, utilizando o ModelSim, de onde são gerados os arquivos contendo as atividades de chaveamento nos formatos SAIFs e VCDs.

Com os arquivos de atividades de chaveamento em mãos, essas informações foram repassadas para o PowerCompiler e para o XPower para serem geradas as estimativas de consumo de energia. Essas estimativas foram então colocadas em uma tabela base, contendo cada uma das instruções e suas respectivas estimativas de consumo.

A figura 1 mostra em parte a metodologia utilizada para captação de dados no projeto e a tabela 1 da seção 6.1

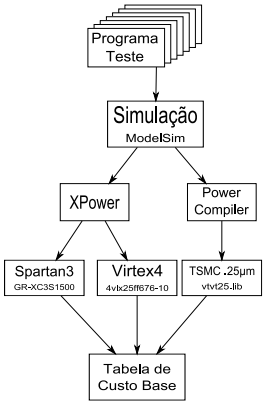


Figura 1. Metodologia utilizada para captação de dados

mostra a estimativa de consumo de energia por classes de instruções.

Essa metodologia segue as propostas de [5] descritas na seção 2. Segundo as características descritas pode-se dizer que a metodologia utilizada é responsável, pois estima o consumo de energia em nível de instruções com um erro médio de $\pm 6\%$ (em relação a resultados obtidos através das ferramentas PowerCompiler e XPower) mesmo não sendo capaz de identificar a significância de todos os fatores que afetam o consumo de energia do processador, resultados que podem ser visualizados na seção 6. É também simplista, pois todo o modelo é baseado em programas simples, com apenas um tipo de instrução em cada núcleo executado, sendo construído utilizando propriedades simples e visíveis ao nível de instruções e é reutilizável, podendo ser usado como base para testes em outros modelos de processadores.

6. Resultados Experimentais

A alimentação do PowerCompiler foi feita através da simulação e captação de atividade de chaveamento utilizando o ModelSim 6.1b [1]. Foi utilizado o mesmo conjunto de programas para validar os testes. Todas as medições de tempo foram feitas utilizando-se o mesmo computador, com as seguintes configurações: processador Pentium IV HT 2.8 GHz 1024 MB RAM. A máquina em questão foi isolada durante os testes de desempenho.

6.1. Validação dos programas utilizados

Para validar os programas de testes foi utilizado o recurso de estatística do ArchC. O núcleo dos programas tes-

tes é um laço que contém 100 instruções de um mesmo tipo. Este laço é executado 1.000 vezes, totalizando 100.000 execuções de uma mesma instrução, mostrado pelo algoritmo 1.

Nesse algoritmo é mostrado o programa base para a instrução ADD. Todos os outros programas base tem o mesmo cabeçalho e rodapé, onde .LL2 e .LL5 são responsáveis pela iteração e, main e .LL3 são as instruções de controle de início e fim de programa. Para os testes das demais instruções, a mudança se limita às instruções do núcleo do programa. Em .LL2, a instrução cmp %g1,999 é responsável pelas iterações.

Algorithm 1 Exemplo de um dos núcleos dos programas base utilizados

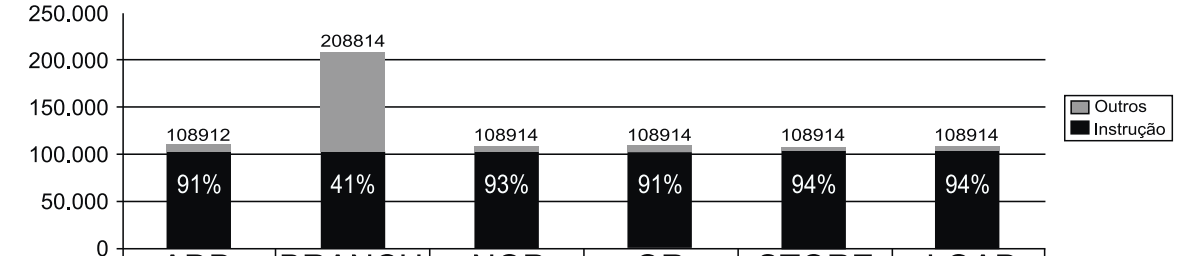
```
   .file "call.s"
   .section ".text"
   .align 4
   .global main
   .type main, #function
   .proc 04

main:
   !#PROLOGUE# 0
   save %sp, -120, %sp
   !#PROLOGUE# 1
   mov 1, %g1
   st %g1, [%fp-20]
   mov 1, %g1
   st %g1, [%fp-12]
.LL2: // Variáveis de controle
   ld [%fp-12], %g1
   cmp %g1, 999 // Comparação
   bg .LL3
   nop
.LL4: // Início do núcleo
   add %g0, %g1, %g2
   .
   .
   .
   add %l3, %l4, %l5 // Fim do núcleo
.LL5:
   ld [%fp-12], %g1
   add %g1, 1, %g1 // Iteração
   st %g1, [%fp-12]
   b .LL2
   nop
.LL3: // Saída
   mov 0, %g1
   mov %g1, %i0
   ret
   restore
   .size main, .-main
   .ident "GCC: (GNU) 3.4.4"
```

O laço tem o tamanho suficiente e é executado um número de vezes suficiente para que o *overhead* das instruções de controle seja mínimo. Como pode ser visto no gráfico da figura 2, os núcleos dos programas são responsáveis por $\pm$90% das instruções executadas pelo programa. Com exceção da instrução BRANCH que, para cada desvio, um NOP é utilizado para preencher o *delay slot*.

Com base nessas informações é possível afirmar que o responsável pelo consumo do programa é a instrução que está sendo executada no núcleo.

A tabela 1 mostra o consumo de energia por classe de instrução para cada um dos processadores, Leon2 e Leon3, em cada uma das diferentes tecnologias, com freqüências próximas à 50MHz (52 MHz em FPGA, Virtex4 e Spartan3; e 50 MHz em ASIC, TSMC .25μm) obtidas pelas ferramentas PowerCompiler e XPower. As instruções pertencentes à cada classe utilizada são:

- **ADD:** add_reg, addcc_reg, addx_reg, addxcc_reg, addxcc_imm, addx_imm, addcc_imm, add_imm.
- **AND:** and_reg, andcc_reg, andn_reg, andncc_reg, and_imm, andcc_imm, andn_imm, andncc_imm.
- **BRANCH:** ba, bn, bne, be, bg, ble, bg, bgu, bleu, bcc, bcs, bpos, bneg, bvc, bvs.
- **CALL:** call.
- **CMP:** cmp.
- **DIV:** udiv_reg, udivcc_reg, sdiv_reg, sdivcc_reg, udiv_imm, udivcc_imm, sdiv_imm, sdivcc_imm, stb_imm.
- **JMP:** jmpl_reg, jmpl_imm.
- **LOAD:** ldsb_reg, ldsh_reg, ldub_reg, lduh_reg, ld_reg, ldd_reg, ldsb_imm, ldsh_imm, ldub_imm, lduh_imm, ld_imm, ldd_imm, ldstub_imm, ldstub_reg.
- **MOV:** mov.
- **NOP:** nop.
- **OR:** or_reg, orcc_reg, orn_reg, orncc_reg, or_imm, orcc_imm, orn_imm, orncc_imm.
- **RESTORE:** restore_reg, restore_imm.
- **SAVE:** save_reg, save_imm.
- **SETHI:** sethi.
- **SHIFT:** sll_reg, srl_reg, sra_reg, sra_imm, srl_imm, sll_imm.
- **SMUL:** smul_reg, smulcc_reg, mulscc_reg, smul_imm, smulcc_imm, mulscc_imm.
- **STORE:** sth_imm, st_imm, std_imm, stb_reg, sth_reg, st_reg, std_reg.
- **SUB:** sub_reg, subcc_reg, subx_reg, subxcc_reg, subxcc_imm, subx_imm, subcc_imm, sub_imm.
- **UMUL:** umul_reg, umulcc_reg, umul_imm, umulcc_imm.
- **XOR:** xor_reg, xorcc_reg, xnor_reg, xnorcc_reg, xor_imm, xorcc_imm, xnor_imm, xnorcc_imm.



	ADD	BRANCH	NOP	OR	STORE	LOAD
Outros	9012	106915	7012	9013	6014	6014
Instrução	99900	101899	101902	99901	102900	102900

Figura 2. Validação do método utilizando estatísticas de alguns dos programas de teste

- **UNIMP:** trap_imm, trap_reg, unimplemented, wry_imm, rdy, swap_imm, wry_reg, swap_reg.

6.2. Validação das ferramentas

Para validar as ferramentas foi utilizado um mesmo programa teste, apenas variando a freqüência do processador. Essa validação utilizou a ferramenta PowerCompiler, tecnologia TSMC 0.25 e variação de freqüência de 25 a 200 MHz. O gráfico 3, valida a ferramenta pois, mantendo a capacitância do sistema, a tensão de funcionamento e o *toggle rate*, a freqüência de operação é a responsável pelo consumo de energia do sistema, como indica a equação a seguir:

$$P_{din} = C_L V_{dd}^2 f T_{Rate} \quad (2)$$

A figura 3 mostra o crescimento do consumo para um conjunto de programas tais como ADD, LOAD e Hello World, em comparação com uma linha base (tracejada), de crescimento linear

Caso o crescimento fosse linear, a equação 2 poderia ser reduzida à seguinte equação onde a constante K substituiria $C_L V_{dd}^2 T_{Rate}$.

$$P = \frac{f_{new}}{f_{old}} K \quad (3)$$

A equação 4 foi construída após os resultados experimentais e implementada no ***acpower*** para se calcular a estimativa de consumo de energia em uma freqüência diferente da estabelecida no arquivo de entrada.

Esses resultados foram obtidos utilizando como programas testes o programa ADD, LOAD e o Hello World! e todos utilizaram como ferramenta o PowerCompiler, como tecnologia o TSMC .25μm e como freqüência de testes 25, 50, 75, 100, 125, 150, 175 e 200 MHz.

Essa função pode ser visualizada pela reta pontilhada no gráfico da figura 3. Para a tecnologia TSMC .25μm o valor de α é 0,76689883.

$$P = \frac{f_{new}}{f_{old}} \alpha K \quad (4)$$

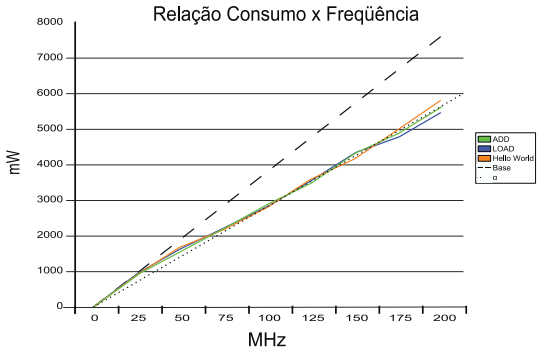


Figura 3. Comparativo do crescimento linear, real e utilização do fator α.

6.3. Avaliação do ArchC e acpower

A tabela 2 mostra a diferença entre os tempos de execução de alguns programas, utilizando ModelSim e ArchC, simulando um processador SPARCv8 e os mesmos programas quando compilados em C e executados em uma máquina qualquer. Os programas utilizados como teste são programas bem básicos como Hello World! e um programa que multiplica duas matrizes 20x20 com valores predefinidos. Esses dois programas serão utilizados como base para

Tabela 1. Consumo aferido por classe de instrução.

	Resultados - Consumo por Classes de Instruções				
	Leon2			Leon3	
	FPGA (mW)		ASIC (W)	FPGA (mW)	ASIC (W)
	Spartan3	Virtex4	TSMC .25μm	Spartan3	TSMC .25μm
add	182	507	1,60	204	1,51
and	182	507	1,74	204	1,48
branch	181	507	1,68	204	1,51
call	171	501	1,64	204	1,52
cmp	182	507	1,65	204	1,53
div	179	505	1,65	205	1,53
jump	195	514	1,71	216	1,48
load	242	539	1,67	225	1,52
mov	182	507	1,66	204	1,53
nop	182	507	1,66	204	1,51
or	182	507	1,68	204	1,53
restore	182	507	1,64	204	1,51
save	182	507	1,64	204	1,51
sethi	182	507	1,64	204	1,50
shift	182	507	1,63	204	1,50
smul	164	497	1,64	201	1,54
store	242	539	1,67	225	1,52
sub	182	507	1,59	204	1,57
umul	164	497	1,64	201	1,50
xor	182	507	1,63	204	1,52
unimpl	0	0	0	0	0

comparação de alguns outros resultados mostrados abaixo. Foram escolhidos por serem simples e de rápida simulação, mesmo assim a simulação do programa de matrizes demorou 1 hora para finalizar no ModelSim. Como teste final de eficiência, foi utilizado o programa qsort_small, presente no MediaBench ou no MiBench. Para a utilização desse programa foram necessárias algumas modificações como por exemplo a utilização de um vetor com apenas 128 palavras inclusas no código fonte, ao invés da leitura de arquivo utilizado pelo programa original. Para a captação de tempo foi utilizado o comando *time*. O campo SpeedUp(X) mostra quantas vezes o ArchC foi mais rápido que o ModelSim na simulação do processador.

Tabela 2. Tempo de execução(em segundos).

	Tempo de execução do programa			
	C	ModelSim	ArchC	SpeedUp(X)
qsort (modificado)	0,016	2,5 dias	3,288	65693
Matriz	0,002	3540	0,507	6982
Hello World	0,002	28	0,255	109

Como pôde ser visto, a diferença de tempo de simulação é considerável e o crescimento é não-linear. Esse crescimento depende do nível de simulação utilizado. Quanto mais alto o nível, mais rápida é a simulação. Esse é um dos fatores positivos em se utilizar de uma ADL para simulação e do nível de instruções para o cálculo da estimativa de consumo de energia.

A tabela 3 compara os tempos gastos com a simulação para captação da atividade de chaveamento, utilizando ModelSim, com a estimativa de consumo de energia, utilizando o PowerCompiler, com a simulação no ArchC e o com a estimativa utilizando o ***acpower***. Observando a tabela é possível reafirmar que quanto mais alto o nível de abstração, mais rápida será sua simulação e resultado. Devido ao tempo gasto com o qsort modificado, este não foi re-simulado utilizando ModelSim para Leon2 nem para captação de VCDs.

Tabela 3. Tempo de simulação.

	Tempo de Simulação (em segundos)					
	Leon2		Leon3		ArchC	
	MSim	PCompiler	MSim	PCompiler	Simulação	acpower
qsort	-	-	2,5 dias	1857	3,288	0,031
Matriz	3540	630	1680	1860	0,507	0,031
Hello World	28	612	18	1860	0,255	0,031

A tabela 4 faz uma última comparação entre o tempo gasto com a estimativa no método convencional e o tempo gasto com o ArchC (simulação + ***acpower***). É possível afirmar que o tempo economizado utilizando o ArchC para estimar o consumo de energia compensa o erro médio de $\pm 5\%$. Como fator de comparação, o PowerCompiler utilizado como base tem erros médios intrínsecos de $\pm 20\%$ (isto significa que os valores de referência já estão com este erro incluídos). Como o qsort modificado não foi re-simulado para captação de VCDs, não há resultado comparativo utilizando XPower. Os resultados apresentados aqui são do Leon2, utilizando tecnologia TSMC .25μm, 50 MHz, no PowerCompiler e Virtex4, 52 MHz, no XPower. Para o qsort modificado, o único resultado disponível é do PowerCompiler, utilizando Leon3 a 50 Mhz na tecnologia TSMC .25μm.

Tabela 4. Comparativo entre tempos e resultados de estimativa de consumo de energia das ferramentas.

	Resultados Experimentais						
	Método Convencional				acpower		ERRO
	XPower		PCompiler				
	Cons(mW)	T(s)	Cons(mW)	T(s)	Cons(mW)	T(s)	%
qsort	-	-	1646	1857	1534	0,031	6,8
Matriz	-	-	1644	630	1606	0,031	2,3
Hello World	-	-	1666	612	1652	0,031	0,8
Matriz	507,3	90	-	-	497,6	0,031	1,9
Hello World	506,5	72	-	-	513,3	0,031	1,3

A tabela 5 mostra os consumos médios dos benchmarks Mediabench e Mibench. Os programas comuns aos dois

benchmarks não foram re-simulados.O consumo médio do Leon3 em Virtex4 não pôde ser calculado devido à versão do ISE utilizada. As bibliotecas de modelos Virtex4 disponíveis no WebPack não eram grandes o suficiente para sintetizá-lo. Para obter as estatísticas do lame large foram necessários 7 horas de simulação em ArchC.

Essa tabela apresenta apenas os resultados obtidos pelo uso do ***acpower***. A maior característica do ***acpower*** é que, uma vez tendo o modelo de consumo de energia e a saída estatística do ArchC, ele não necessita de uma re-simulação para calcular o consumo para um mesmo programa. Apenas se troca o arquivo que contém as novas informações ou a freqüência de operação do dispositivo, e se tem a nova estimativa. Assim, o tempo total para calcular toda a tabela utilizando ***acpower*** foi de apenas 70 segundos. Se o ***acpower*** estivesse atrelado internamente ao ArchC, seriam necessárias 35 horas de simulação apenas para calcular a estimativa do programa lame large. Analisando a tabela 2 é possível estimar que seriam necessários mais de 17 dias (ou mais de 420 horas) para obter a atividade de chaveamento do mesmo programa utilizando o Modelsim.

Uma característica interessante apontada pela tabela é o fato do consumo na FPGA ser menor do que o apresentado pela ASIC. Como a freqüência utilizada pelas FPGAs são compatíveis com a utilizada pela ASIC, $\pm$50 MHz, essa diferença se deve ao fato da tecnologia utilizada pelos fabricantes. Enquanto o TSMC é fabricado utilizando .25μm e as PFGAs Virtex4 e Spartan3 são fabricadas utilizando tecnologia 90nm.

Tabela 5. Resultados do acpower para programas do Mediabench e Mibench.

	Resultados Experimentais - Benchmarks				
	Leon2			Leon3	
	FPGA(mW)		ASIC(W)	FPGA(mW)	ASIC(W)
	Spartan3	Virtex4	TSMC .25μm	Spartan3	TSMC .25μm
MediaBench					
adpcm coder	182,36	499,12	1,61	201,39	1,50
adpcm decoder	194,96	532,91	1,73	215,08	1,59
adpcm timing	186,14	509,40	1,64	205,55	1,52
cjpeg	190,50	503,21	1,60	204,22	1,49
djpeg	192,61	514,70	1,64	209,43	1,52
gsm toast	191,49	509,28	1,62	206,77	1,51
gsm untoast	187,04	509,63	1,63	205,88	1,52
mpeg encoder	192,83	508,76	1,64	206,54	1,50
mpeg decoder	190,40	509,47	1,65	206,52	1,51
pegwit decript	197,88	513,71	1,64	208,98	1,51
pegwit encript	196,72	513,74	1,64	208,79	1,51
pegwit generate	195,21	511,60	1,63	207,81	1,51
MiBench					
basicmath	188,45	507,16	1,64	205,28	1,51
bitcount large	183,50	503,55	1,63	203,34	1,50
bitcount small	183,57	503,58	1,63	203,37	1,50
quicksort large	201,82	525,07	1,68	213,75	1,55
quicksort small	202,80	528,49	1,69	215,10	1,56
susan corners	194,15	502,95	1,60	204,80	1,48
susan edges	189,95	506,53	1,62	205,33	1,50
susan smoothing	193,30	514,74	1,64	209,05	1,53
lame large	189,67	505,88	1,63	205,08	1,50
lame small	189,31	505,95	1,63	205,04	1,50
djikstra large	199,53	516,46	1,65	210,33	1,52
dijkstra small	199,13	516,14	1,65	210,16	1,52
patricia large	193,20	510,74	1,64	207,28	1,51
patricia small	193,11	510,75	1,64	207,27	1,51
stringsearch large	193,29	512,79	1,64	208,04	1,52
stringsearch small	193,28	512,76	1,64	208,02	1,52
crc32 large	195,46	495,66	1,58	202,39	1,45
crc32 small	195,46	495,67	1,58	202,39	1,45

7. Conclusões

Este artigo apresentou o ***acpower***, uma ferramenta de estimativa de consumo de energia em nível de instrução, adicional à saída estatística do ArchC, que utiliza-se de técnicas de estimativa de consumo de energia em um nível mais alto de abstração do que as ferramentas mais conhecidas no mercado e foram apresentados os resultados finais do projeto e comparações entre os tempos de execução dos métodos tradicionais e os da ferramenta desenvolvida, juntamente com seu erro.

O ***acpower***, apesar de ser uma ferramenta simples, é extremamente eficiente para re-estimar o consumo de energia de um determinado programa sem ter o trabalho de re-simulação, economizando tempo útil a algum projeto. Além de ser mais rápido, apresenta um erro consideravelmente baixo em relação ao método convencional.

Referências

[1] M. G. Corporation. *ModelSim SE 6.1b Quick Reference Guide*, 2005.

[2] J. Frenkil. Tools and methodologies for low power design. In *DAC '97: Proceedings of the 34th annual conference on Design automation*, pages 76–81, New York, NY, USA, 1997. ACM Press.

[3] Gaisler. *http://www.gaisler.com/*, 2007.

[4] B. Klass, D. E. Thomas, H. Smith, and D. F. Nagle. Modeling inter-instruction energy effects in a digital signal processor, 1998.

[5] S. Lee, A. Ermedahl, and S. L. Min. An accurate instruction-level energy consumption model for embedded risc processors. *SIGPLAN Not.*, 36(8):1–10, 2001.

[6] S. Nikolidis and T. Laopoulos. Instruction-level power consumption estimation on embedded processors for low-power applications, 1999.

[7] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. Archc: A systemc-based architecture description language. In *SBAC-PAD '04: Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pages 66–73, Washington, DC, USA, 2004. IEEE Computer Society.

[8] J. Russell and M. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *ICCD '98: Proceedings of the International Conference on Computer Design*, page 328, Washington, DC, USA, 1998. IEEE Computer Society.

[9] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. Syst.*, 2(4):437–445, 1994.

[10] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. Instruction level power analysis and optimization of software. *J. VLSI Signal Process. Syst.*, 13(2-3):223–238, 1996.

[11] S. V8. *www.sparc.org/standards/V8.pdf*, 2005.