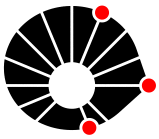
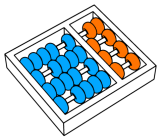




Análise de execução de aplicações em ambientes de computação aproximada

Rodrigo de S. S. da Silva, Lucas F. Wanner

Relatório Técnico - IC-PFG-19-56
Projeto Final de Graduação
2019 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Análise de execução de aplicações em ambiente de Computação Aproximada

Rodrigo de S. S. da Silva[1], Lucas F. Wanner[2]

[1,2] Instituto de Computação Universidade Estadual de Campinas (UNICAMP), Caixa Postal 6176
13083-970 Campinas-SP, Brasil

Resumo. Computação Aproximada é um paradigma que busca reduzir o custo de energia do processamento ao comprometer a precisão dos resultados. A técnica abordada nesse trabalho é o uso de multiplicadores inexatos inteiros, que executam operações de forma mais rápida ou mais eficiente, porém geram produtos imprecisos. Nosso estudo utiliza a arquitetura ArchC em conjunto com a linguagem de descrição ADeLe para avaliar a eficiência de diversas aplicações, quando submetidas a 6 tipos multiplicadores inexatos inteiros diferentes.

Enquanto algumas aplicações de uso geral não se beneficiam da aproximação, outras geram erros fatais e não produzem nenhuma saída, devido aos produtos inexatos. Aplicações de processamento de imagens foram as únicas que apresentaram a queda em qualidade e energia esperada, porém a queda de qualidade é grande demais para justificar seu uso em atividades práticas. Espera-se que futuros pesquisas possam gerar multiplicadores inexatos com uma queda de qualidade aceitável para uso geral, além de serem mais resilientes que os utilizados para a pesquisa.

Palavras-Chave: Computação Aproximada; Computação Energética Eficiente

1. Introdução

Um dos fatores mais importantes no desempenho computacional moderno é o consumo e dissipação de energia. Com o fim da escalabilidade multicore próximo de ser atingido, existe um limite no poder de processamento de processadores convencionais [1]. Com a crescente demanda por capacidade de processamento, através de aplicações de larga escala, em um fator superior à capacidade de produção de processadores, esta não é uma solução eficiente para o desafio da indústria de computação nos próximos anos [2].

A Computação Aproximada é um paradigma que busca reduzir o custo de energia do processamento ao comprometer a precisão dos resultados. Entre as diversas técnicas exploradas para esse fim estão a redução na tensão de alimentação, o uso de arquiteturas imprecisas, [precision scaling], e aleatoriamente ignorar leituras a memória [3] [4]. Ao economizar energia realizando as mesmas operações, é possível acelerar a frequência de processamento e, assim, processar muito mais com a mesma energia de antes.

Entre os desafios atuais da Computação Aproximada estão a necessidade de futuros designs de processadores e linguagens de programação acomodarem operações aproximadas, que sempre assumiram operações e escritas exatas [1]. Além disso, um melhor desenvolvimento de técnicas para avaliar em tempo real se os resultados imprecisos podem comprometer as aplicações, evitando catástrofes como falhas de segmentação.

2. Trabalhos Relacionados

Kulkarni, Gupta e Ercegovac apresentam um multiplicador aproximado de 2x2 bits que economiza energia e área através de uma modificação no mapa de Karnaugh [5]. Ao se modificar o resultado de 3x3 de 9 para 7, mostra-se uma redução na área dos circuitos, uma vez que a saída passa a ser representada apenas por 3 bits. A economia de energia vem da quantidade signicativamente menor de portas, e um caminho crítico menor. A aproximação apresenta uma probabilidade de erro de 1/16, com magnitude de 22.2%. Porém, os estudos práticos dos autores mostram uma redução de energia de (31.8% - 45.6%) em multiplicações e valores de SNR de 2 a 8 vezes melhores do que em multiplicações aproximadas.

EvoApprox8B [6] é uma suíte de modelos de arquitetura que implementa operações matemáticas inexatas, com o objetivo de auxiliar estudos de Computação Aproximada. Nossa pesquisa utiliza cinco multiplicadores inexatos da suite para comparação com o multiplicador de Kulkarni, totalizando 6 diferentes multiplicadores.

ADeLe [7] é uma linguagem de modelagem em alto nível que simplifica o processo de injeção de aproximações em um modelo de CPU. Graças a essa simplificação, as alterações

necessárias ao código original são mínimas. Assim, um design de aproximação pode ser validado simplesmente com o arquivo de descrição e um modelo. Além disso, a estrutura da linguagem gera resultados de energia que podem ser facilmente utilizados para se medir qualidade.

ArchC [8] [9] é uma linguagem de descrição de arquitetura que, em conjunto com SystemC, permite que qualquer sistema seja representado, com a possibilidade de expansão para unidades e ferramentas customizadas. Em conjunto com a ADeLe, é possível que todo o sistema seja controlado para injeção de aproximações e simulações.

Este estudo pretende explorar melhor o impacto prático de multiplicadores aproximados, através de mais testes em múltiplos benchmarks e algoritmos de processamento de imagens, através da composição ArchC e ADeLe. De acordo com a literatura, experimentos com multiplicadores inexatos foram realizados com sucesso apenas com algoritmos relacionados a processamento de imagens.

3. Condução dos experimentos

O modelo de arquitetura considerado no trabalho substitui multiplicadores de inteiros por multiplicadores inexatos, que consomem menos energia, porém geram resultados inexatos.

Como as modificações e os algoritmos não possuem elementos aleatórios, apenas uma execução é necessária para cada combinação de aplicação, multiplicador e entrada, resultando em um total de 175 execuções.

Configuração das ferramentas

Para a instalação do simulador, é necessário a instalação e configuração das ferramentas ArchC e SystemC e de variáveis de ambiente. Uma versão modificada do ArchC, denominada VArchC, foi utilizada para a geração de arquivos .csv para leitura facilitada dos resultados.

As instruções para obter e compilar o ArchC estão disponíveis na pagina http://www.archc.org/doc.quickstart.html [10].

Para a obtenção do SystemC, é necessário obter a fonte da página https://www.accellera.org/downloads/standards/systemc. ArchC requer a versão antiga 2.3.1 seja instalada. Após isso, as instruções para compilação estão em [10].

Para a compilação de aplicações em C e C++ para a arquitetura MIPS, utilizamos o compilador ELLCC. As instruções para sua obtenção e compilação estão disponíveis em http://ellcc.org/?page_id=20295.

Modificação das aplicações

Após a instalação do VArchC, a pasta raíz do simulador gerará o arquivo mips_iface.h. Esse arquivo deve ser copiado para a pasta do código fonte da aplicação a ser aproximada e simulada. Além disso, o código fonte deve declarar a biblioteca mips adicionando-se a linha

#include "mips_iface.h"

A implementação das funções declaradas na função é feita pelo simulador MIPS. Assim, a compilação e execução nativa da aplicação após a declaração não gerará diferenças. Para a ativação de uma aproximação, a função *vac_activate()* deve ser chamada antes da seção que se deseja tornar aproximada. O único parâmetro é um inteiro de 2 bytes correspondente à aproximação a ser escolhida. A lista de aproximações disponíveis é gerada na compilação do simulador e seus códigos de identificação descritos no arquivo *mips_iface.h*. Para o nosso trabalho, a lista de multiplicadores está descrita na seção 3.4.

É importante ativar a aproximação apenas antes de iniciar o kernel da aplicação, para que ela não influencie a leitura da entrada e utilize dados inválidos para a computação.

Para a desativação de uma aproximação, a função *vac_deactivate()* deve ser chamada após a seção que de deseja tornar aproximada. Assim como *vac_activate()*, o único parâmetro é um inteiro de 2 bytes correspondente à aproximação escolhida e a lista é idêntica.

É importante desativar a aproximação após toda computação necessária seja executada, para que ela não influencie a escrita dos dados de saída e torne-os ilegíveis.

A função *vac_newSection()* é utilizada para separar o trecho de código a ser aproximado em seções diferentes. Cada seção recebe seus próprios contadores de energia, o que permite análises mais detalhadas da execução.

Para a alteração do código fonte, adicionamos a sequência de linhas

vac_activate(VAC_APPROX_A);

vac_newSection();

antes de chamada a função ou loop principais da aplicação, e a sequência de linhas

vac_newSection();

vac_deactivate(VAC_APPROX_A);

após a chamada. Assim, os gráficos de saída dividem os resultados em três seções, sendo a primeira e ultima seções de 0 consumo de energia. Isso facilita a leitura manual dos dados obtidos.

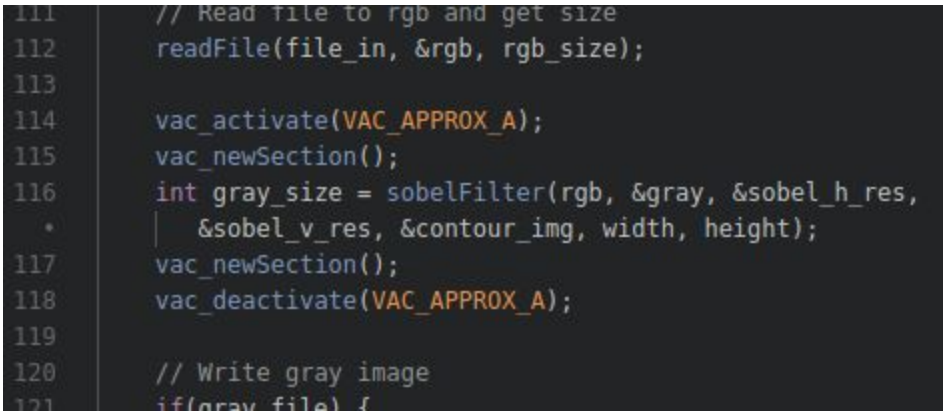


Figura 1: trecho de código mostrando como ativar e desativar aproximações para o experimento.

Uso das ferramentas

Todas as aplicações são compiladas através compiladores ECC, para aplicações em C, e ECC++, para aplicações em C++. Ambos são parte da geração do compilador ELLCC.

O executável MIPS deve ser gerado através do comando

${PATH_TO_ECC} -target mips32r2-linux -O3 ${FILES} -o ${EXEC} ${ARGS} ${FLAGS}

para aplicações em C, ou

${PATH_TO_ECC++} -target mips32r2-linux -O3 ${FILES} -o ${EXEC} ${ARGS} ${FLAGS}

para aplicações em C++.

${PATH_TO_ECC} é o caminho para o compilador de C gerado por ELLCC;

${PATH_TO_ECC++} é o caminho para o compilador de C++ gerado por ELLCC;

${FILES} são os arquivos do código fonte;

${EXEC} é o pátio do executável MIPS;

${ARGS} corresponde aos argumentos requisitados pela aplicação;

${FLAGS} são parâmetros de compilação adicionais que a aplicação pode necessitar, como *-lm* para incluir bibliotecas de matemática ou *-D_VARCHC_SUPPRESS_* para ignorar aproximações e analisar o consumo de energia com multiplicadores exatos.

O executável MIPS pode ser usado através do comando

${PATH TO SIMULATOR} -- ${PATH TO EXECUTABLE} ${ARGS}

${PATH TO SIMULATOR} é o caminho do simulador MIPS gerado pela ferramenta VArchC;

${PATH TO EXECUTABLE} é o caminho do executável gerado por ELLCC;

${ARGS} são os argumentos da aplicação.

Após o término da execução, os arquivos

${EXEC}_varchc_${TYPE}_${PID}.csv

são produzidos, descrevendo as estatísticas de gasto de energia.

${EXEC} é o nome do executável;

${TYPE} corresponde a "counters" ou "energy". "counters" descreve o total de instruções executadas para cada tipo, e "energy" descreve a quantidade de energia consumida por elas.

${PID} é o ID do processo do simulador, durante a execução.

Aplicações

As aplicações escolhidas para o trabalho são:

1. JPEG, algoritmo de compressão de imagens, disponível na suíte AxBench [11]. É uma aplicação utilizada em diversos pesquisas relacionadas à Computação Aproximada [3]. A análise dos resultados gerados nesse trabalho podem ser comparados com a de outros trabalhos como uma forma de validação dos resultados obtidos.
2. FFT (Fast Fourier Transform) é um algoritmo de conversão de sinais digitais utilizado para processamento multimídia, disponível na suíte MiBench [12]. O benchmark também implementa a função inversa IFFT (Inverse Fast Fourier Transform).
3. ADPCM (Adaptative Differential Pulse Code Modulation) é uma variação do algoritmo PCM (Pulse Code Modulation), usada para converter amostras PCM de 16 bits para 4 bits, resultando em uma taxa de conversão 4:1. A aplicação está disponível na suíte MiBench [12].
4. Basicmath, aplicação que calcula soluções de equações quadráticas e cúbicas, realiza conversões de graus para radianos e vice-versa, e calcula raízes quadradas de números inteiros. Tal aplicação está disponível na suíte MiBench [12]. O algoritmo original foi modificado para que todas as impressões de saída ocorressem ao final, para evitar que a impressão da saída fosse aproximada pelos multiplicadores, gerando resultados inválidos.
5. Lame é um codificador de áudio MP3 com suporte a diversos tipos de variação de taxa de bits. A aplicação está disponível na suíte MiBench [12].

6. Blackscholes é um algoritmo que modela a variação de preços sobre tempo, utilizado para auxílio em decisões de investimentos. Ela está disponível na suíte AxBench. [10].
7. CannyEdge é o algoritmo mais popular de detecção de bordas em imagens. A aplicação está disponível em [13].
8. SobelFilter é outro algoritmo de detecção bordas em imagens, e se diferencia do algoritmo CannyEdge separando-as em bordas verticais e horizontais. A aplicação está disponível em [14].

Para cada aplicação, foram gerados 7 executáveis listados abaixo.

1. Aplicação sem multiplicadores inexatos, que são identicas à aplicação original. Para isso, o parâmetro *-D_VARCHC_SUPPRESS_* é fornecido ao compilador ECC durate a geração dos executáveis MIPS, e as funções *vac_activate()*, *vac_deactivate()* e *vac_newSection()* são definidas vazias, fazendo com que nenhuma aproximação seja utilizada. Logo, os resultados de energia e qualidade são idênticos aos das aplicações compiladas nativamente.
2. Aplicação com o multiplicador inexato de Kulkarni [5]. Para isso, o arquivo gerado pela constução do simulador *mips_iface.h* foi modificado para especificar o multiplicador de escolha. O simulador permite que múltiplas aproximações estejam ativas simultâneamente, porém o trabalho apenas utiliza uma aproximação por aplicação.
3. Cinco aplicações com multiplicadores inexatos gerados pela biblioteca EvoApprox8b. [6] Os multiplicadores foram criados através de técnicas como aproximação da árvore dos produtos parciais, e o uso de contadores aproximados. Os multiplicadores escolhidos foram *mul8-303*, *mul8-469*, *mul8-479*, *mul8-423* e *mul8-279*, com base em suas recompensas teóricas na troca de fator energia-qualidade [7].

Métrica de energia

Para cada execução, foram analisados a qualidade final do resultado gerado e o ganho de energia final. Tal ganho foi calculado ao comparar-se o total de energia gasto reportado pelo simulador com o total de energia que a aplicação com multiplicadores precisos e mesma entrada gastou. O ganho relativo de energia pode ser calculado ao dividir-se o total de energia da aplicação aproximada (itens 2 a 7 acima) pelo total de energia da aplicação precisa (item 1 acima).

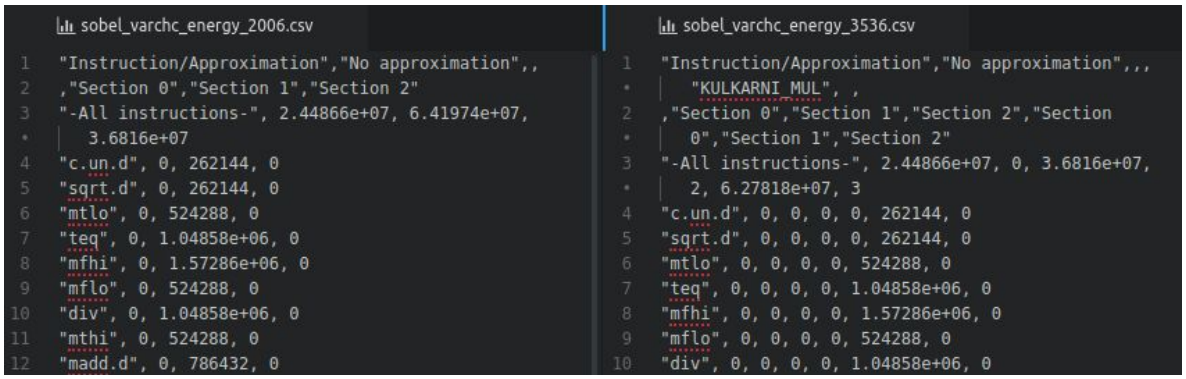


Figura 2: Comparação das métricas de energia da mesma aplicação para diferentes multiplicadores

Métrica de qualidade

Para as aplicações que geram uma imagem como saída, que são JPEG e SobelFilter, a qualidade do resultado pode ser medida através de um algoritmo de métrica de qualidade específico para imagens JPEG, originado de [15] e [16]. Para as demais aplicações, foi utilizada, quando possível, a técnica dos mínimos quadrados, utilizando-se como entrada todos os números resultantes de cada saída.

meansquare.py

```python
from skimage import data, img_as_float
from skimage.io import imread
from skimage.measure import compare_ssim as ssim
from skimage.measure import compare_mse

def mse(x, y):
    return np.linalg.norm(x - y)

def main():
    try:
        f1 = img_as_float(imread(sys.argv[1]))
        f2 = img_as_float(imread(sys.argv[2]))
    except:
        exit(1)
    v_mse = mse(f1,f2)
    v_ssim = ssim(f1,f2)

    print("comparing images " + sys.argv[1] + " and " + sys.argv[2] + ":")
    print("mean error is " + str(v_mse))
    print("ssim is " + str(v_ssim))
```

```
rodrigo@rodrigo-Aspire-E1-571:~/varchc/results$ python3 meansquare.py jp
eg/bin/v_o/10_2.jpeg jpeg/bin/v4/10_2.jpeg
comparing images jpeg/bin/v_o/10_2.jpeg and jpeg/bin/v4/10_2.jpeg:
mean error is 52.647965669335676
ssim is 0.8348635199522064
rodrigo@rodrigo-Aspire-E1-571:~/varchc/results$ python3 meansquare.py jp
eg/bin/v_o/11_2.jpeg jpeg/bin/v4/11_2.jpeg
comparing images jpeg/bin/v_o/11_2.jpeg and jpeg/bin/v4/11_2.jpeg:
mean error is 63.53256213442698
ssim is 0.8642006409065532
rodrigo@rodrigo-Aspire-E1-571:~/varchc/results$ python3 meansquare.py jp
eg/bin/v_o/12_2.jpeg jpeg/bin/v4/12_2.jpeg
comparing images jpeg/bin/v_o/12_2.jpeg and jpeg/bin/v4/12_2.jpeg:
mean error is 58.889446726845776
ssim is 0.7902393632638384
```

Figura 3: Script Python utilizado para métrica de qualidade, e exemplo de uso.

4. Resultados

As aplicações podem ser separadas em três grupos, de acordo com o tipo de resultado obtido.

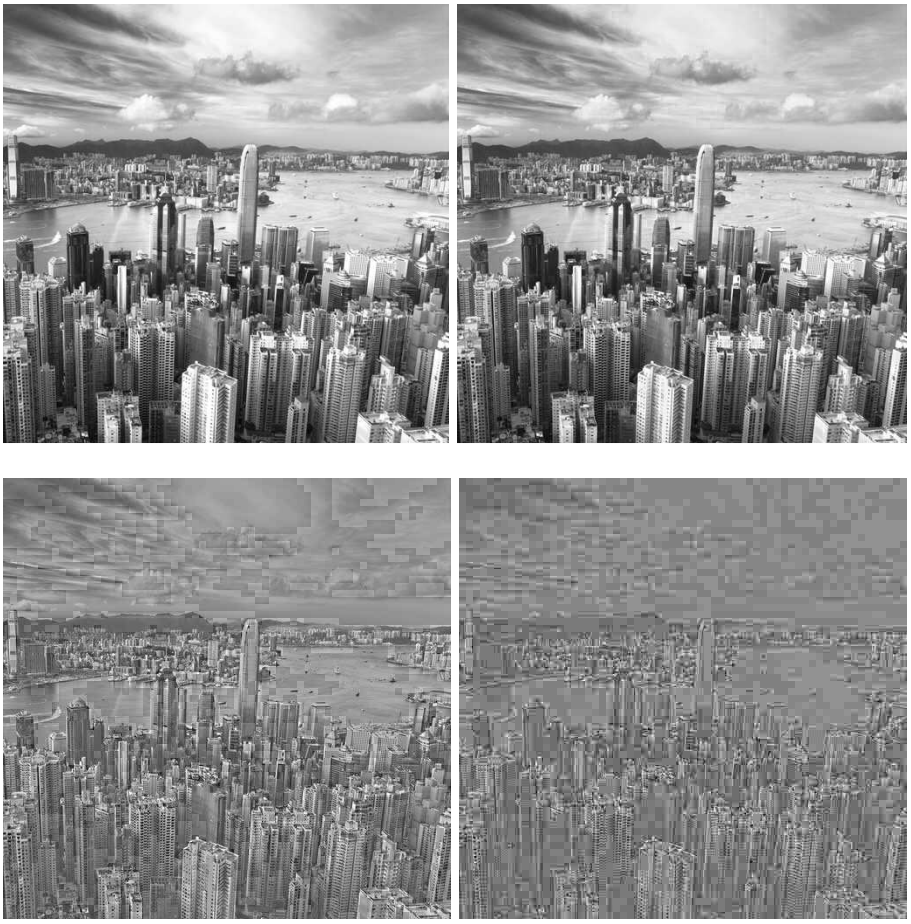
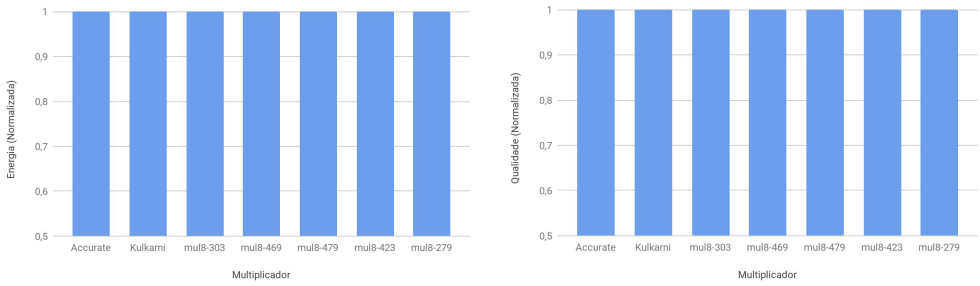


Figura 4: Imagens de saída da aplicação jpeg, com os multiplicadores: preciso, Kulkarni, *mul8-469* e *mul8-279*

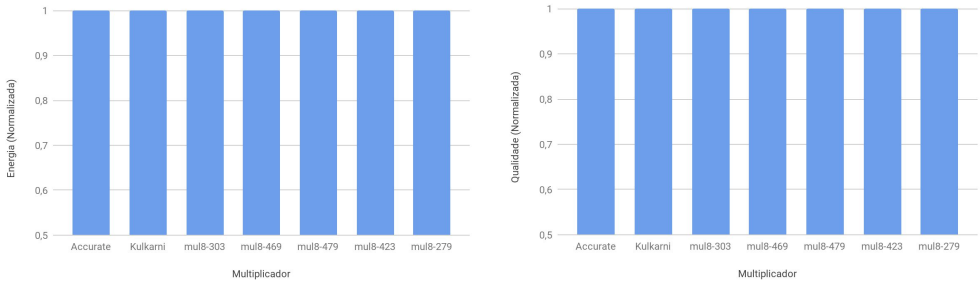
Aplicações que não se beneficiam de multiplicadores inexatos inteiros

As aplicações Basicmath, Blackscholes e FFT não apresentaram nenhuma alteração em qualidade e energia após a injeção dos multiplicadores inexatos. Observa-se que todas as operações de multiplicação realizadas no kernel das aplicações são multiplicações de ponto flutuante para ponto flutuante, que não são influenciadas pelos multiplicadores. Conclui-se que essa técnica de aproximação não é benéfica para tais aplicações.

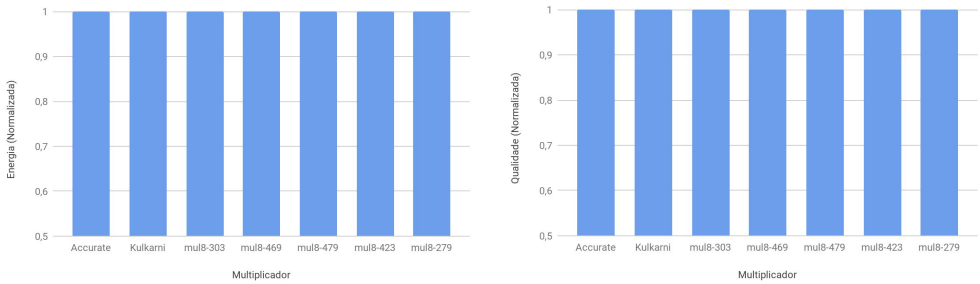
Energia Qualidade



a) BasicMath



b) Blackscholes

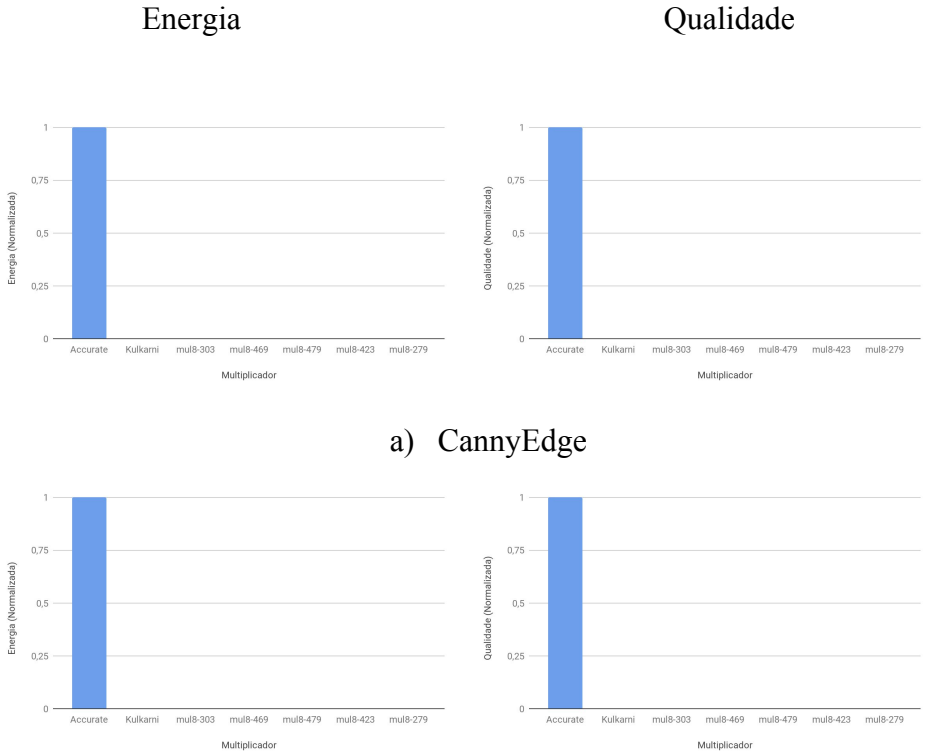


c) FFT

Figura 5: Métricas de energia e qualidade (normalizadas para o multiplicador preciso) para as aplicações do Grupo 1.

Aplicações cuja combinação com aproximações geram resultados invalidos

As aplicações Lame e CannyEdge apresentaram erros fatais na compilação e não geraram resultados válidos. Logo, o quoeficiente de energia e qualidade para qualquer resultado de qualquer multiplicador é zero. O principal motivo para esse resultado são falhas de segmentação causadas pelo uso de ponteiros calculados incorretamente.



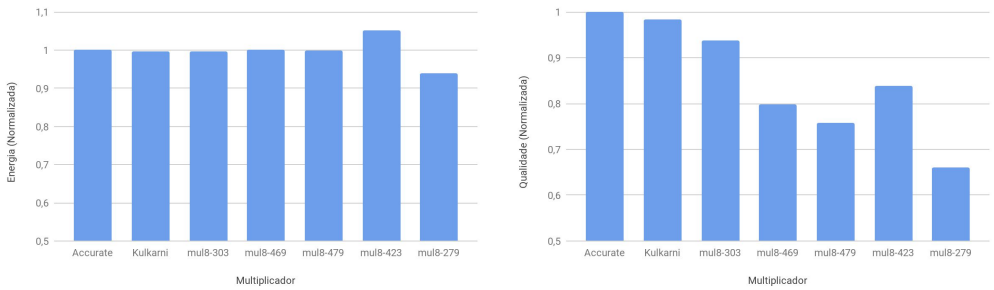
a) CannyEdge

b) Lame

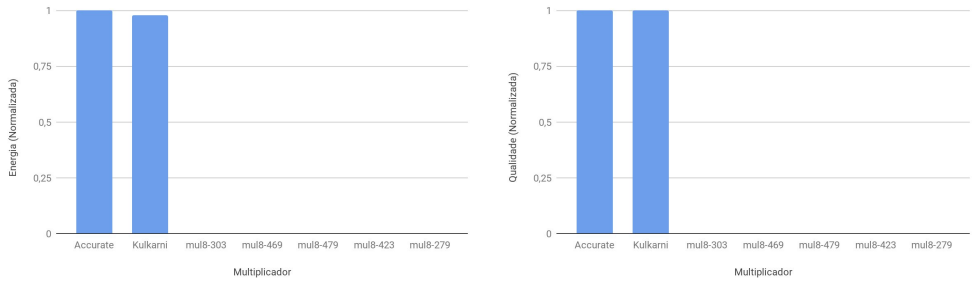
Figura 6: Métricas de energia e qualidade (normalizadas para o multiplicador preciso) para as aplicações do Grupo 2

Aplicações que apresentam queda de energia e queda de qualidade sob influência das aproximações

As aplicações Jpeg e Sobel, quando executadas com os multiplicadores inexatos, apresentaram tanto resultados válidos, sem erros fatais de execução, quanto queda no consumo de energia da aplicações simuladas. Logo foi possível analisar os resultados de acordo com o objetivo da pesquisa.



a) jpeg



b) SobelFilter

Figura 7: Métricas de energia e qualidade (em comparação ao multiplicador preciso) para as aplicações do Grupo 3

Observa-se que os multiplicadores *mul8-469*, *mul8-479* e *mul8-423* apresentaram um quoeficiente de energia maior que 1, ou seja, consumiram mais energia, em média, que a aplicação com multiplicador preciso. Um possível motivo é a repetição adicional de loops causada pelo cálculo incorreto de suas quantidades.

Observa-se também que nem todos os multiplicadores geraram resultados válidos para a aplicação SobelFilter. Enquanto o multiplicador de Kulkarni gerou resultados válidos, os outros multiplicadores inexatos apresentaram erros fatais. Isso se deve a diferentes formas de multiplicação gerarem possivelmente resultados inesperados, como loops terminados inesperadamente ou escrita de dados inválidos.

5. Conclusão

Existem uma grande variedade de técnicas de computação aproximada, porém não há uma quantidade extensiva de resultados no impacto prático das aproximações em sistemas embarcados. Nesta pesquisa, procuramos explorar o impacto de multiplicadores inexatos em diversos tipos de aplicações através de simulações em aplicações MIPS com as ferramentas ADeLe e ArchC.

Nos nossos experimentos com computação aproximada, multiplicadores inexatos têm uso limitado a algoritmos de processamento de imagens, devido ao formato dos dados. Não foi possível obter resultados válidos ao se aplicar as aproximações estudadas em aplicações fora desse campo, para as aplicações pesquisadas.

Além disso, observa-se que o multiplicador de Kulkarni apresentou melhores resultados em qualidade e energia e resiliência, em comparação com os multiplicadores da suíte EvoApprox8B. Para os multiplicadores da suíte, a queda de qualidade é grande demais para justificar seu uso nessas aplicações. Para o multiplicador de Kulkarni, a queda de qualidade é muito menor, porém a queda de energia também é muito baixa.

6. Referências

[1] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., & Burger, D. Dark silicon and the end of multicore scaling. In ISCA, 2011.

[2] John Gantz and David Reinsel. 2011. Extracting Value from Chaos. http://www.emc.

com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf, 2011.

[3] Mittal, S. “A Survey of Techniques for Approximate Computing. CSUR. 2016

[4] Fursin G. Collective Benchmark. ctuning.org/cbench/, 2010.

[5] P. Kulkarni, P Gupta, M. D. Ercegovac. “Trading Accuracy for Power in a Multiplier Architecture”, Journal of Low Power Electronics, vol. 7 no. 4, 2011.

[6] V. Mrazek, R. Hrbacek, Z. Vasicek, e L. Sekanina, “EvoApprox8b: Library of Approximate adders and multipliers for circuit desin and benchmarking of approximation methods”, in *DATE*, 2017.

[7] I. Felzmann, M. Susin, L. Duena, R. Azevedo and L. Wanner, “ADeLe: Rapid Architectural Simulation for Approximate Hardware”, in *SBAC-PAD*, 2018.

[8] Rigo, S. Araujo, G., Bartholomeu, M. and Azevedo, R. “ArchC: a systemC- based Architecture Description Language. SBAC-PAD, 2004.

[9] M. Guedes, R. Auler, L. Duenha, E. Borin, and R. Azevedo, “An Automatic Energy Consumption Characterization of Processors using ArchC”, Journal of Systems Architecture, vol. 59, no 8, 2013.

[10] ArchC - Getting Started. http://www.archc.org/doc.quickstart.html

[11] A. Yazdanbakshs, D. Mahajan, P. Lotfi-Kamran, and H. Esmaeilzadeh. “AXBENCH: A Multi-Platform Benchmark Suite for Approximate Computing.” IEEE D&T, 2016.

[12] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., e Brown, R. B. Mibench, A free, commercially representatibe embedded benchmark suite. In IEEE WWC, 2001

[13] Repositório de petermlm. https://github.com/petermlm/SobelFilter

[14] Documentação de scikit-image. Structural similarity index. http://scikit-image.org/docs/dev/auto_examples/transform/plot_ssim.html

[15] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.