Avaliando eficiência energética em padrões de algoritmos para computação científica e de alto desempenho

Paulo N. M. dos Anjos[1], **Alvaro L. Fazenda**[1]

[1]Campus São José dos Campos – Universidade Federal de São Paulo (UNIFESP)
São José dos Campos – SP – Brazil

{paulo.anjos,alvaro.fazenda}@unifesp.br

Abstract. *The development of new algorithms usually focuses on performance improvement, with little attention given to environmental impact and energy cost in terms of their execution. However, this topic has been gaining greater attention recently. This work aims to objectively demonstrate the energy consumption of programming patterns commonly found in high-performance scientific programs. The measurement of energy consumption was performed using software through the RAPL interface, and other performance metrics will utilize the quantity of executed operations and elapsed time. Tests are conducted by varying the number of threads used, compilation options, amount of memory utilized, and the processors used, aiming to identify the impacts these changes have on energy efficiency. The results demonstrate that energy efficiency is directly affected by the scalability of the algorithm, and that more aggressive compilation optimizations generally increase it.*

Resumo. *O desenvolvimento de novos algoritmos costuma concentrar-se na melhoria do desempenho, muitas vezes sem dar devida atenção ao impacto ambiental e ao custo energético decorrentes de sua execução. No entanto, esse tema tem recebido crescente atenção recentemente. Este trabalho tem como propósito demonstrar de maneira objetiva o consumo de energia de padrões de programação frequentemente encontrados em programas científicos que exigem alto poder de processamento. A medição do consumo de energia foi realizada por meio de software utilizando a interface RAPL, enquanto outras métricas de desempenho empregaram a contagem de operações executadas e o tempo decorrido. Foram conduzidos testes variando a quantidade de threads utilizadas, as opções de compilação, a quantidade de memória empregada e os processadores usados, com o objetivo de identificar os impactos que essas mudanças causam na eficiência energética. Os resultados evidenciam que a eficiência energética, geralmente, beneficia-se ao utilizar otimizações mais agressivas na compilação, sendo também diretamente influenciada pela escalabilidade do algoritmo.*

1. Introdução

O contínuo aumento de custo energético relacionado ao uso de equipamentos eletrônicos de forma cada vez mais frequente e pervasiva vem causando preocupação mundial. Acredita-se que, em 2015, foram despendidos cerca de 250 bilhões de dólares globalmente para manter os computadores atuais operacionais. Surpreendentemente, somente 15% desse montante foi direcionado para atividades de computação efetiva, de acordo

com [Anwar et al. 2013], enquanto o restante foi destinado a manter os dispositivos em modo de espera ou inativos [Salama 2020]. De acordo com um relatório recente da IEA (*International Energy Agency*) [1] atualmente a demanda de energia de *data centers* e transmissão de dados em rede, corresponde à 1-1.5% da demanda total de energia no mundo, sendo também responsável por 1% da geração de gases do efeito estufa no mundo.

A computação verde (*Green Computing*), também conhecido como tecnologia da informação verde (*Green IT*), representam o conjunto de hardware, software, técnicas e metodologias que visam reduzir o impacto energético dos sistemas digitais, consequentemente melhorando a eficiência energética e diminuindo a emissão de carbono oriunda de todas as etapas de um produto, incluindo o desenvolvimento, uso e descarte [Murugesan and Gangadharan 2012]. A busca por melhor eficiência energética atinge também a área de Processamento de Alto Desempenho, área da computação que geralmente demanda muito poder de computação e, consequentemente, muita energia no processamento. É possível encontrar publicações recentes de trabalhos relacionados a soluções que promovem eficiência energética para computação de alto desempenho [D'Agostino et al. 2021] [Czarnul et al. 2019]. A lista Green500 [2] também demonstra a preocupação da comunidade científica com o tema. Na edição de junho de 2022, o supercomputador denominado Frontier (*Hewlett Packard Enterprise Frontier* ou OLCF-5, localizado no *Oak Ridge Leadership Computing Facility* nos EUA) ocupava tanto a primeira posição em desempenho no Top500 quanto a segunda posição no Green5002[2].

O objetivo deste trabalho é medir desempenho e custo energético de códigos-fonte que representam padrões comuns em soluções numéricas aplicadas à Computação Científica, aos quais normalmente se aplicam técnicas de computação de alto desempenho, promovendo concorrência em diversos níveis (instruções - vetorização, *threads* e processos), além de otimizações diversas, tais as que promovem melhoria na latência de acesso aos diversos níveis de memória, execuções especulativas e fora de ordem, entre outras. Os estudos de caso desse trabalho terão como variações dois diferentes níveis de otimização do compilador, o incremento contínuo na quantidade de *threads* paralelas utilizada, a variação no uso de memória, utilizando-se de duas diferentes arquiteturas computacionais. Buscar-se-á identificar quais os efeitos dessas variações para a eficiência energética partir de métricas comuns à área, tais como o tempo de processamento, o custo energético total e relativo por unidade de tempo, além da métrica *MFLOPs/Watt*.

2. Metodologia

Foram selecionados dois diferentes padrões de programação comumente utilizadas em programas de computação científica, adaptados ao uso de recursos de processamento paralelo em *multithread*, sendo eles:

- *Reduction*: programa padrão para reduzir um conjunto de entradas, normalmente utilizando-se de operadores associativos e comutativos, onde a ordem de execução das operações geralmente não mudam o resultado [Voss et al. 2019];
- *Stencil*: programa padrão na qual uma função mapeia o conjunto de dados de entrada para o conjunto de dados de saída[Voss et al. 2019], geralmente de mesma

[1] https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks
[2] https://www.top500.org/

dimensionalidade e tamanho, porém levando em consideração a vizinhança da entrada [Malony 2014].

As métricas serão baseadas na quantidade de operações realizadas e a energia necessária para executá-las, o qual é também utilizado pela bem conhecida lista *Green500*. Assim, a principal métrica utilizada será *MFLOPs/watt*, onde o desempenho computacional será medido em FLOPS (*floating point operations per second*, em português, quantidade de operações com dados de ponto flutuante por segundo), e *watt* como a unidade padrão de energia em joules por segundo.

A medida do custo energético para executar programas referentes a cada um dos dois padrões será obtido através de uma funcionalidade chamada de RAPL (*Running Average Power Limit*, em português, Limite de Energia Médio de Execução), sendo ele uma interface desenvolvida pela Intel que utiliza MSRs (*Model-Specific Registers*, em português, registradores de modelo específico) para armazenar o uso e gasto energético de vários domínios relacionados ao processador [Pandruvada 2014], como pacote total *cores* (pp0), *igpu* (pp1), memórias. O RAPL não é um medidor analógico de energia, mas um modelo matemático por *software*, onde se estima o consumo energético do *hardware* por meio de contadores de desempenho e modelos de *I/O*, porém, as medidas obtidas são muito próximas dos valores reais [Rotem et al. 2012].

Vários trabalhos semelhantes são encontrados na literatura realizando testes e análises de eficiência energética com o uso do RAPL [Hähnel et al. 2012], aplicando em métodos de Aprendizado de máquina [Brownlee et al. 2021], medindo custo energético em operações do MPI [Venkatesh et al. 2013], Multiplicação de matrizes [Paniego et al. 2018].

A partir da fundamentação apresentada foram implementados programas em linguagem C++ para os dois citados padrões, utilizando programação aderente ao bem conhecido padrão OpenMP para promover paralelismo em múltiplas *threads*. O código-fonte desenvolvido foi também instrumentado para medir o tempo de execução do trecho de interesse através da função própria do padrão OpenMP denominada *omp_get_wtime*, e para adquirir os valores dos registradores do RAPL que permitam estimar o custo energético do sistema durante a execução. Para minimizar a interferência de outros processos em execução simultânea, os testes foram realizados com o sistema executando apenas o algoritmo desenvolvido ou em nós isolados quando possível. Para obtenção das métricas, são guardadas em variáveis específicas os valores do MSR do RAPL e o horário imediatamente antes e depois da execução da função específica do algoritmo selecionado. Todos os códigos-fonte desenvolvidos e seus resultados podem ser obtidos a partir do repositório disponível no GitHub em https://github.com/TCC-Green/TCC-Green/.

Cada algoritmo com suas respectivas opções de otimização de compilação e de variação de *threads* foi executado cinco vezes para cada um dos dois sistemas descritos:

- Sistema 1: Computador com um processador i7-9750H (6 núcleos, 12 *threads*), 2x8GB KF432S20IB/8, Linux Mint 21.1, g++ 11.3.0.
- Sistema 2: Servidor com dois processadores E5-2660 V4 (14 núcleos, 28 *threads*), 8x16GB M393A2G40EB1-CRC, CentOS 7, g++ 11.2.0.

Além das variações citadas anteriormente, com relação às opções de otimização do compilador, a quantidade de *threads* utilizada e dos diferentes hardwares, variaram-se os

tamanhos dos *arrays* (tipo *double*) de forma a provocar mudanças no acesso à memória cache em seus diversos níveis, observando se tais fatos poderiam causar diferenças na eficiência energética. Dessa forma, os programas desenvolvidos foram testados nas seguintes configurações:

- Configuração 1: tamanho 100×100 e 5 milhões de iterações.
- Configuração 2: tamanho 40000×40000 e 31 iterações.
- Configuração 3: tamanho 28000×28000 e 63 iterações.

3. Resultados

Nas Tabelas 1, 2, 3 e 4, encontram-se as métricas para o padrão *reduction* referente a energia total consumida em joules, tempo decorrido de processamento, eficiência energética em *MFlops/Watt* e *speedup*. Em quase todos os casos, exceto com o sistema 2 na configuração 1, houveram ganhos de desempenho com o aumento da quantidade de *threads*, mesmo ao utilizar as *threads* lógicas/virtuais. Com a redução do tempo de execução, também houve um aumento na eficiência energética na maioria dos testes.

Tabela 1. Sistema 1, configuração 1, algoritmo *reduction*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	5886,93	227,17	8,49	1,00
O0	2	3844,07	118,31	13,01	1,92
O0	4	2866,32	63,89	17,44	3,56
O0	8	1900,78	39,52	26,30	5,75
O0	12	1512,71	29,37	33,05	7,73
O3	1	4426,52	193,25	11,30	1,00
O3	2	2846,41	100,98	17,57	1,91
O3	4	2060,82	54,02	24,26	3,58
O3	8	1506,95	31,12	33,18	6,21
O3	12	1182,28	22,91	42,29	8,44

Tabela 2. Sistema 1, configuração 2, algoritmo *reduction*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	5630,82	229,59	8,81	1,00
O0	2	3958,41	117,45	12,53	1,95
O0	4	2871,59	62,07	11,27	3,70
O0	8	1728,98	36,07	28,69	6,37
O0	12	1296,02	24,96	38,27	9,20
O3	1	4291,82	192,63	11,56	1,00
O3	2	2858,5	99,19	17,35	1,94
O3	4	2054,31	51,79	24,14	3,72
O3	8	1374,01	27,98	36,10	6,88
O3	12	1020,27	19,14	48,61	10,06

As métricas para o algoritmo *stencil* podem ser encontradas nas Tabelas 5, 6, 7 e 8. Para o sistema 1 não foi possível utilizar a configuração 2, visto que ultrapassa

Tabela 3. Sistema 2, configuração 1, algoritmo *reduction*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	25330,72	335,23	1,97	1,00
O0	2	13756,59	170,92	3,63	1,96
O0	4	7390,48	88,11	6,77	3,80
O0	8	4402,31	48,93	11,36	6,85
O0	16	3725,77	35,47	13,42	9,45
O0	32	5214,66	41,11	9,59	8,15
O3	1	19211,72	230,68	2,60	1,00
O3	2	10341,81	117,48	4,83	1,96
O3	4	5608,73	61,59	8,91	3,75
O3	8	3458,74	34,99	14,46	6,59
O3	16	2864,92	25,31	17,45	9,11
O3	32	3361,65	25,82	14,87	8,93

Tabela 4. Sistema 2, configuração 2, algoritmo *reduction*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	31194,42	331,88	1,59	1,00
O0	2	16359,67	166,27	3,03	2,00
O0	4	8617,64	83,20	5,76	3,99
O0	8	4790,21	42,01	10,35	7,90
O0	16	2735,30	21,15	18,13	15,69
O0	32	1751,97	11,67	28,31	28,44
O3	1	18959,37	228,49	2,62	1,00
O3	2	9948,14	114,43	4,99	2,00
O3	4	5267,85	57,29	9,42	3,99
O3	8	2902,52	28,97	17,09	7,89
O3	16	1675,83	14,47	29,60	15,79
O3	32	1027,07	7,42	48,29	30,79

a quantidade de memória disponível, dessa forma, foi utilizada a configuração 3 para explorar o desempenho computacional e energético ao se utilizar mais memória que a configuração 1. Para o sistema 1, o padrão *stencil* exibe baixa escalabilidade em ambas as configurações. Para o sistema 2, a escalabilidade continua abaixo do desejado, especialmente com a configuração 1 e O3, onde o tempo de execução com mais de 16 *threads* fica maior que a versão serial. Para a configuração 2 no sistema 2, a escalabilidade continua baixa, porém, não repete o comportamento citado acima, para a otimização máxima.

4. Discussões e conclusões

Para os algoritmos testados, apenas o *reduction* apresentou ganhos com o aumento do número de *threads* em execução, ainda que algumas vezes pequenos, em todas as configurações, enquanto o *stencil* mostrou ganhos menores geralmente acima de 4 *threads*. Percebe-se que a eficiência energética é diretamente proporcional a eficiência paralela do algoritmo, implicando na importância de se ter um algoritmo paralelo eficiente.

Importante citar que a utilização de uma opção de otimização mais agressiva do

Tabela 5. Sistema 1, configuração 1, algoritmo *stencil*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	10181,74	343,88	19,64	1,00
O0	2	6819,43	183,17	29,33	1,88
O0	4	4651,17	104,69	43,00	3,28
O0	8	4716,46	103,78	42,40	3,31
O0	12	3790,96	80,23	52,76	4,29
O3	1	715,16	27,11	279,66	1,00
O3	2	588,87	16,50	339,63	1,64
O3	4	623,57	11,71	320,73	2,32
O3	8	718,36	13,64	278,41	1,99
O3	12	758,73	12,32	263,60	2,20

Tabela 6. Sistema 1, configuração 3, algoritmo *stencil*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	9677,40	353,03	20,42	1,00
O0	2	6072,24	188,28	32,54	1,88
O0	4	4353,81	104,77	45,38	3,37
O0	8	3668,15	84,44	53,86	4,18
O0	12	3282,07	69,86	60,20	5,05
O3	1	1561,46	47,86	126,53	1,00
O3	2	1696,86	42,60	116,43	1,12
O3	4	1939,57	40,98	101,86	1,17
O3	8	2050,64	42,81	96,34	1,12
O3	12	2425,71	50,01	81,45	0,96

Tabela 7. Sistema 2, configuração 1, algoritmo *stencil*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	45938,25	597,31	4,35	1,00
O0	2	25131,60	307,77	7,96	1,94
O0	4	14973,29	172,02	13,36	3,47
O0	8	10095,58	104,08	19,81	5,74
O0	16	9725,80	85,05	20,56	7,02
O0	32	22248,99	164,73	8,99	3,63
O3	1	4161,89	49,13	48,06	1,00
O3	2	2944,77	32,45	67,92	1,51
O3	4	2560,09	26,32	78,12	1,87
O3	8	3223,96	21,29	62,04	2,31
O3	16	6106,85	52,79	32,75	0,93
O3	32	11709,26	86,33	17,08	0,57

compilador sempre resultou em melhor eficiência energética, especialmente na versão serial, de encontro a expectativa criada pela redução no tempo de execução, porém, restava a dúvida se usar funcionalidades mais avançadas do processador ativadas por um código otimizado, tal como pipelines de execução vetorial, poderiam causar aumento significativo

Tabela 8. Sistema 2, configuração 2, algoritmo *stencil*.

Otim.	Threads	Energia(J)	Tempo(s)	E.e.(MFlops/W)	Speedup
O0	1	58760,44	616,05	3,36	1,00
O0	2	31520,78	310,95	6,27	1,98
O0	4	17014,75	158,40	11,67	3,89
O0	8	9574,36	79,49	20,64	7,75
O0	16	5885,77	40,41	33,57	15,24
O0	32	5468,30	37,36	36,13	16,49
O3	1	8854,74	104,26	22,31	1,00
O3	2	4950,82	52,70	39,91	1,98
O3	4	2972,50	28,75	66,47	3,63
O3	8	1889,19	15,14	104,03	6,89
O3	16	1596,19	10,79	123,77	9,66
O3	32	1747,89	12,25	113,03	8,51

no consumo energético, o que não se configurou. Cabe lembrar ainda que a otimização mais agressiva implicaram em menor escalabilidade paralela nos códigos testados, o que diminuiu os ganhos energéticos ao se utilizar maiores quantidades de *threads*.

Como relação aos diferentes sistemas utilizados, percebe-se que o sistema 1 apresenta melhor eficiência energética em praticamente todas as configurações testadas, incluindo a versão serial, quando comparada ao sistema 2. Isso se deve, possivelmente, a tecnologias construtivas mais modernas, uma vez que o processador do sistema 1 foi lançado em 2019, tendo frequência máxima de 4.50GHz (2.60GHz em modo padrão), enquanto o processador do sistema 2 é um lançamento de 2016, com frequência máxima de 3.20GHz (2.00GHz em modo padrão). Para as diferentes configurações de memória, percebem-se diferentes comportamentos. No sistema 1, tem-se uma melhor escalabilidade e eficiência energética de forma sutil para o *reduction* a medida que se usa mais memória. Para o algoritmo de *stencil* a configuração com mais uso de memória tem eficiência energética muito baixa ao se utilizar de otimização agressiva (O3). Para o sistema 2, a melhor escalabilidade da configuração 2 causa uma melhor eficiência energética para os dois algoritmos, entretanto, a eficiência energética é menor ao se usar poucas *threads* com mais memória. Ao se aumentar o número de *threads* a quantidade de memória por *thread* diminui devido à divisão do domínio, incrementando a eficiência energética.

4.1. Conclusões

Este trabalho demostra a realização de diversos testes que avaliam desempenho computacional e eficiência energética de códigos simples, porém comuns em aplicações de computação científica, altamente demandantes de computação paralela e de alto desempenho, e visa contribuir para demonstrar e justificar os esforços no sentido de se modificar o código-fonte para melhorar a eficiência paralela e escalabilidade forte do software para que se tenha melhor eficiência energética. Como trabalhos futuros, destaca-se a expansão dos testes para outros padrões e *benchmarks* conhecidos, testes com outros processadores e aceleradores e padrão de programação paralela para paralelismo com memória distribuída (MPI) e OpenAcc, bem como investigar as razões da baixa eficiência paralela encontrada em alguns testes e propor melhorias.

5. Agradecimentos

Este trabalho foi parcialmente financiado pelo Projeto #2019/26702-8, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Referências

Anwar, M., Qadri, S., and Sattar, A. (2013). Green computing and energy consumption issues in the modern age. *IOSR Journal of Computer Engineering*, 12(6):91–98.

Brownlee, A. E., Adair, J., Haraldsson, S. O., and Jabbo, J. (2021). Exploring the accuracy – energy trade-off in machine learning. In *2021 IEEE/ACM International Workshop on Genetic Improvement (GI)*, pages 11–18.

Czarnul, P., Proficz, J., and Krzywaniak, A. (2019). Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019:1–19.

D'Agostino, D., Merelli, I., Aldinucci, M., and Cesini, D. (2021). Hardware and software solutions for energy-efficient computing in scientific programming. *Scientific Programming*, 2021:1–9.

Hähnel, M., Döbel, B., Völp, M., and Härtig, H. (2012). Measuring energy consumption for short code paths using rapl. *SIGMETRICS Perform. Eval. Rev.*, 40(3):13–17.

Malony, A. D. (2014). Stencil pattern. Disponível: `https://ipcc.cs.uoregon.edu/lectures/lecture-8-stencil.pdf`. Acesso: 04/01/2022.

Murugesan, S. and Gangadharan, G. R. (2012). *Harnessing green it: Principles and practices*. John Wiley & Sons, Inc.

Pandruvada, S. (2014). Running average power limit. Disponível: `https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl`. Acesso: 06/01/2022.

Paniego, J. M., Gallo, S., Pi Puig, M., Chichizola, F., De Giusti, L., and Balladini, J. (2018). Analysis of rapl energy prediction accuracy in a matrix multiplication application on shared memory. In De Giusti, A. E., editor, *Computer Science – CACIC 2017*, pages 37–46, Cham. Springer International Publishing.

Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., and Rajwan, D. (2012). Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27.

Salama, M. (2020). Green computing, a contribution to save the environment. Disponível: `https://www.lancaster.ac.uk/data-science-of-the-natural-environment/blogs/green-computing-a-contribution-to-save-the-environment`. Acesso: 06/01/2022.

Venkatesh, A., Kandalla, K., and Panda, D. K. (2013). Evaluation of energy characteristics of mpi communication primitives with rapl. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 938–945.

Voss, M., Asenjo, R., and Reinders, J. (2019). *Mapping Parallel Patterns to TBB*, pages 233–248. Apress, Berkeley, CA.