Impacto das Interfaces de Programação Paralela e do Grau de Paralelismo no Consumo Energético de uma Aplicação*

Thayson R. Karlinski[1], Arthur F. Lorenzon[2], Antonio Carlos S. Beck[2], Márcia C. Cera[1]

[1]Universidade Federal do Pampa (UNIPAMPA) – Campus Alegrete
Av. Tiarajú, 810, CEP: 97546- 550 – Alegrete – RS – Brazil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

thaysonr.karlinski@gmail.com, {aflorenzon,caco}@inf.ufrgs.br, marciacera@unipampa.edu.br

Resumo. *O processamento paralelo é amplamente utilizado para obter ganho de desempenho, porém com uma maior quantidade threads/processos o consumo energético pode tornar-se uma preocupação. Este trabalho investiga o impacto, da IPP utilizada e da variação do número de processos/threads no consumo de energia. Os resultados mostram que, dependendo da configuração utilizada, pode-se consumir até 59% menos energia explorando o paralelismo.*

1. Introdução

Com a popularização dos processadores *multicore*, cada vez mais a programação paralela vem sendo utilizada no desenvolvimento de softwares. Entretanto, se por um lado usar processadores *multicore* reduz o tempo de execução, por outro lado isto poderia aumentar o consumo de energia devido a plena utilização dos núcleos de processamento. Adicionalmente, a comunicação ocorre em regiões de memória mais distantes do processador aumentando assim o consumo de energia [Korthikanti and Agha 2010].

[Lorenzon et al. 2014] mostraram que o consumo de energia de aplicações paralelas sofre influência tanto da Interface de Programação Paralela (IPP) utilizada, quanto da quantidade de *threads*/processos. Identificou-se que o consumo energético dos *benchmarks* utilizados foi mais eficiente a medida que aumentou-se o grau de paralelismo. Adicionalmente, foi possível identificar que grande parte do consumo concentra-se nos acessos a memória. Entretanto, naquele escopo, onde o consumo de energia estava sendo analisado para sistemas embarcados, o grau máximo de paralelismo testado foi 4.

Este trabalho visa analisar o impacto do aumento do grau de paralelismo no consumo de energia. Para isso, serão testadas as IPPs *OpenMP*[1], *Pthread*[2] e MPI[3] para a aplicação do Cálculo do Número PI, estendendo o trabalho de [Karlinski et al. 2014]. A organização do restante do texto será: A Seção 2 descreve a aplicação utilizada e sua implementação; A Seção 3 apresenta o ambiente de execução, bem como, a ferramenta utilizada na coleta de dados; A seção 4 expõe os resultados obtidos e, por fim, a Seção 5 conclui e apresenta os trabalhos futuros.

*Agradeço o auxílio recebido através Edital PBDA/UNIPAMPA 2014.

[1]http://www.openmp.org/ - último acesso em Novembro de 2014

[2]https://computing.llnl.gov/tutorials/pthreads/ - último acesso em Novembro de 2014

[3]https://www.open-mpi.org/ - último acesso em Novembro de 2014

2. Cálculo do Número PI

O Cálculo do Número PI repete iterações de um laço `for`, independentes umas das outras, onde o valor de PI é incrementado e decrementado de acordo com o modelo de Gregory-Leibniz. Esta aplicação é classificada do tipo CPU-Bound, pois existe pouca, ou nenhuma comunicação entre as *threads*/processos e o uso do processador é intenso. A paralelização com *OpenMP* baseou-se na distribuição de iterações do laço `for` entre as *threads*, sendo analisadas as políticas *Static* e *Guided*, uma vez que estas apresentam melhor desempenho [Karlinski et al. 2014]. Já para *Pthreads* e MPI, a carga de trabalho representa o número de iterações do laço divido pelo número de *threads*/processos.

3. Ambiente de execução

Para a realização dos testes foi utilizada uma *Workstation* com dois processadores Intel Xeon E5-2650 com frequência de 2.80 GHz, cada um com 8 *cores* e suporte a tecnologia *Hyper-Threading*. Cada *core* possui uma *cache* L1 de 32 KB e uma *cache* L2 de 256 KB; e ainda existe uma *cache* L3 compartilhada de 20 MB. A versão do compilador GCC utilizada foi a 4.8.2 (sem *flags* de otimização) e a distribuição OpenMPI versão 1.6.5.

Na execução dos testes, foram coletados as seguintes métricas: número total de instruções executadas; o número total de ciclos de *clock*; número total de acessos a memória; e o número total de *misses* da memória *cache* L1. Para coletá-los, foi utilizada a ferramenta PAPI (*Performance Application Programming Interface*), versão 5.3.0, a qual permite acessar os contadores de desempenho disponíveis na maioria dos microprocessadores atuais[4]. Repetiu-se 16 vezes a execução da aplicação para cada número de *threads*/processos: 2, 4, 8, 16, 32 e 64 e IPP, variando-se o evento do PAPI. O desvio padrão permaneceu dentro de 1% eliminando-se os 3 melhores e os 3 piores resultados.

O consumo de energia da aplicação foi calculado através da equação $E_{Total} = E_{inst} + E_{mem} + E_{est}$, definida em [Lorenzon et al. 2014] onde, E_{inst} é o consumo total das instruções executadas; E_{mem} corresponde ao consumo dos acessos (*hits* e *misses*) a memória *cache* L1; e a energia estática E_{est} do processador e do sistema de memória (*caches* e principal). Os dados de energia foram obtidos do manual do processador[5]; do McPAT (*Multicore Power*, *Area and Timing*) [Li et al. 2013] e CACTI Tool [6].

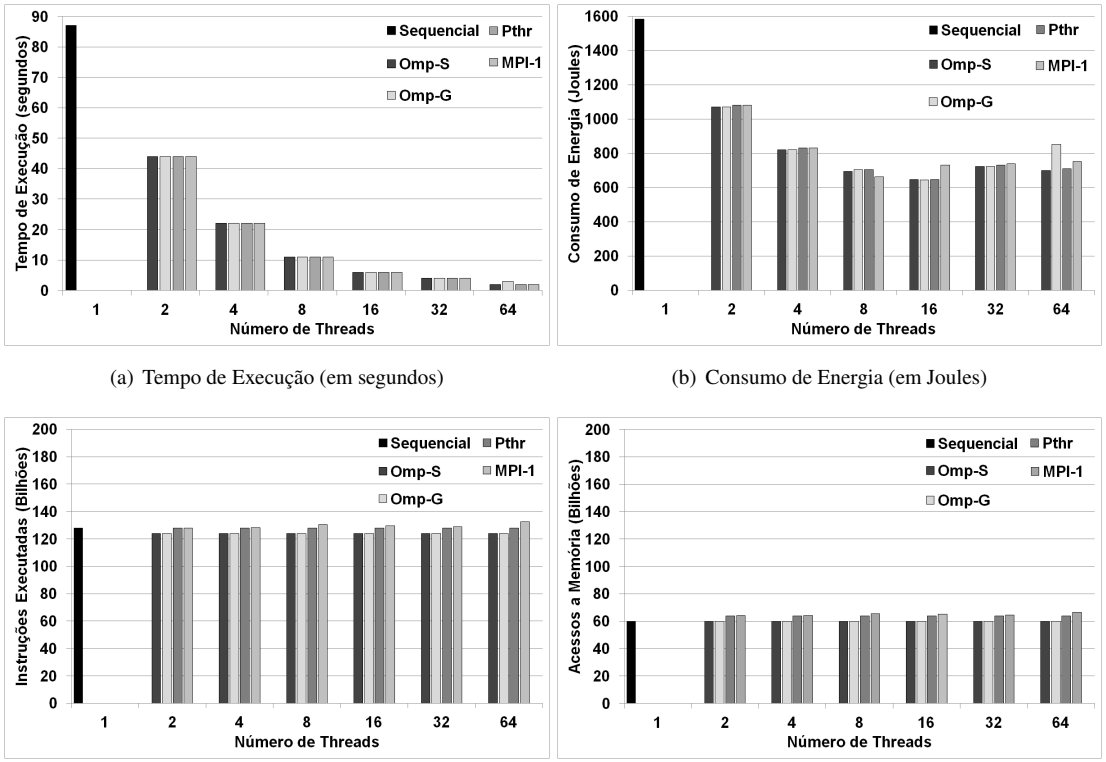
4. Análise dos Resultados

A Figura 1(a) apresenta o tempo de execução para cada IPP, variando o número de *threads*/processos, onde 1 representa o resultado sequencial. Observa-se que variando de 2 a 8 *threads*/processos o *Speedup* foi ideal. Para 16, 32 e 64 *threads*/processos o *Speedup* médio obtido foi de 15, 22 e 44, respectivamente. Isto ocorre, pois quanto maior o grau de paralelismo, menor é o número de iterações que cada *thread*/processo irá computar concorrentemente, entretanto, para 32 e 64 a utilização da tecnologia *Hyper-Threading* fez com que o *Speedup* não fosse o ideal. Na Figura 1(b) é apresentado o consumo de energia, onde pode-se notar que todas as versões paralelas tiveram um menor consumo de energia em relação ao sequencial. O melhor caso foi com 16 *threads* utilizando *Pthreads* (Pthr) onde o consumo de energia foi 59% menor em comparação ao sequencial. Isto deve-se

[4]http://icl.cs.utk.edu/papi/ - último acesso em Novembro de 2014

[5]http://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-1600-2600-vol-2-datasheet.html - acessado em Nov. de 2014

[6]http://www.cs.utah.edu/ rajeev/cacti6/ - último acesso em Novembro de 2014



(a) Tempo de Execução (em segundos)

(b) Consumo de Energia (em Joules)

(c) Número de Intruções Executadas (Em Bilhões)

(d) Acessos e Memória (Em Bilhões)

Figura 1. Eventos coletados para o Cálculo do Número PI.

ao *Speedup* próximo do ideal, diminuindo a energia estática do processador. O pior caso foi com 2 processos/*threads*, MPI e *Pthreads* (Pthr) onde o consumo foi 31% menor em comparação ao sequencial. Nota-se ainda que a utilização de 64 *threads*/processos teve um menor consumo (55% menor que o sequencial) que a utilização de apenas 2.

A Figura 1(c) e (d) mostram um comportamento muito parecido entre os gráficos de instruções executadas e acessos a memória. Para a computação da aplicação (PI) as *threads*/processos não dependem dos resultados uma das outras. Com isso, a variação no número de instruções executadas em relação ao sequencial, é consequência da maneira como cada IPP conduz a execução. Em *OpenMP* o número de instruções executadas foi 2% menor que em relação ao sequencial, devido a uma otimização padrão da *flag* `-fopenmp`, como a utilização de *Loop Unrolling*. A variação nos acessos a memória deve-se a maneira como as IPPs gerenciam as *threads*/processos e os acessos aos dados.

Apresenta-se na Figura 2 os *misses* da *cache* L1, os quais estão separados em dados e instruções. Nota-se que o comportamento de *Pthreads* é similar ao sequencial, onde o número de *misses* de dados é menor que o número de *misses* de instruções, se comparado as outras IPPs. Isto ocorre, por consequência de que a sincronização das *threads* em *Pthreads* depende da inserção de *mutexes* no código. Já para as IPPs *OpenMP* e MPI os *misses* de dados possuem maior quantidade, se comparado ao sequencial e *Pthreads*. Este comportamamento se dá em *OpenMP* em razão de que quando uma *thread* chega no ponto de sincronização, a mesma fica esperando o término da execução das outras *threads*, evento conhecido como *Busy-Waiting*. Para MPI o comportamento de maior número de *misses* em relação as outras IPPs, se dá pela primitiva `MPI_Init`, onde

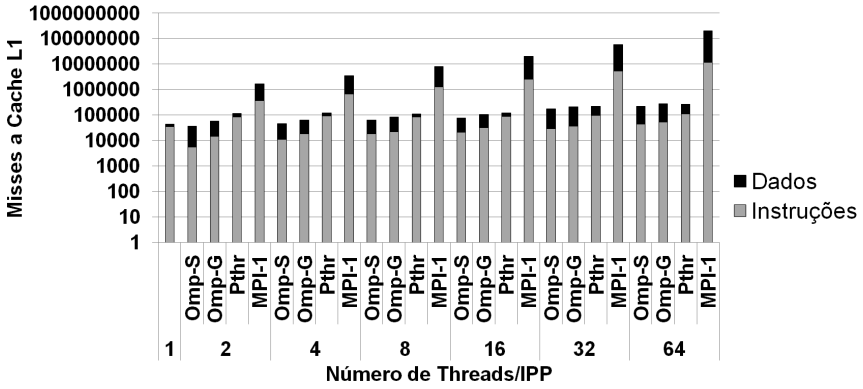


Figura 2. Misses a *cache* L1

são criados os processos e concentra-se o maior número de *misses* a memória *cache* L1. Para 32 e 64 *threads* os *misses* de dados são maiores em virtude da tecnologia *Hyper-Threading*, pois executou mais de uma *thread*/processo por núcleo de processamento.

5. Conclusão e Trabalhos Futuros

Este trabalho analisou a execução do Cálculo do Número PI com diferentes IPPs, variando o número de *threads*/processos, e comparando-as com a execução sequencial. Foram analisados o tempo de execução, os acessos e *misses* a memória *cache* L1, o número de instruções executadas e o impacto desses eventos no consumo total de energia. Nota-se que o aumento do número de *threads*/processos, levou a diminuição do tempo de execução. Logo, obteve-se um menor consumo de energia, que no melhor caso foi de 59%, se comparado ao sequencial, utilizando 16 *threads* com a IPP *Pthreads*. Onde *OpenMP* também é considerado o melhor resultado, devido seus valores estarem dentro do desvio padrão. Entretanto, para 32 e 64 *threads*/processos onde o tempo de execução foi ainda menor, o consumo de energia aumentou. Isto ocorreu, em consequência da utilização da tecnologia *Hyper-threading* onde tivemos um maior número de *misses* da memória *cache* L1. Como trabalhos futuros pretende-se executar outras aplicações do tipo CPU-Bound, além de aplicações I/O-Bound (com maior dependências de dados e comunicação). Assim, será possível aprofundar o estudo com um escopo mais amplo de aplicações.

Referências

Karlinski, T. R., Lorenzon, A., F., A. C. B., and Cera, M. (2014). Análise de desempenho e consumo energético de uma aplicação openmp. In *Workshop de Iniciação Científica do WSCAD 2014*, São José dos Campos/SP.

Korthikanti, V. A. and Agha, G. (2010). Towards optimizing energy costs of algorithms for shared memory architectures. In *22nd Annual ACM, Symposium on Parallelism in Algorithms and Architectures, Thira/Greece*, pages 157–165. ACM.

Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. (2013). The McPAT framework for multicore and manycore archit.: Simult. modeling power, area, and timing. *ACM Transactions on Architecture and Code Optimization*.

Lorenzon, A. F., Cera, M. C., and Beck, A. C. S. (2014). Performance and energy evaluation of different multi-threading interfaces in embedded and general purpose systems. *Journal of Signal Processing Systems*, pages 1–13.