Introdução aos Conceitos de Computação Paralela através da estimativa de Pi

Diego da Silva Pereira[1]

[1]Professor de Redes de Computadores – IFRN Câmpus Currais Novos. e-mail: diego.pereira@ifrn.edu.br

Resumo: O presente trabalho retratou dos conceitos inicias que envolvem a computação paralela e das motivações para a área. É mostrado a API OpenMP que trabalha com o modelo Fork-Join, ilustrando os detalhes e o comportamento do sistema. Além disso é exposto o comportamento do processador quando é executado o algoritmo para cálculo de pi na forma serial e também na forma paralela. Fica comprovado a eficiência do paralelismo com quase 60% de diminuição no tempo total de realização do experimento. Além disso, fui ilustrado como calcular o *speedup* e comprovado que o algoritmo paralelizado realmente obteve um melhor desempenho em uma arquitetura multicore.

Palavras–chave: computação paralela, desempenho, paralelismo, processador

1. INTRODUÇÃO

A busca por poder processamento é motivo de pesquisa em grandes centros acadêmicos, como as universidades de *Rice*, *Stanford* e de *Illinois*, ambas nos Estados Unidos, e empresas como Intel, AMD, IBM e Microsoft. Conseguir o melhor desempenho é uma ambição real do atual momento tecnológico, por isso, há algumas décadas que computação paralela se faz cada vez mais presente no cotidiano de todos.

A *parallel computing*, como costuma ser chamada a computação paralela no cenário internacional, vem sendo utilizada em larga escala motivada principalmente por dois fatores, a limitação física do *clock* das UCPs (Unidade Central de Processamento) e o consumo de energia elétrica, este último levou a Intel a cancelar o lançamento de processadores em 2004, afinal, uma das grandes tendências mundiais é reduzir os impactos dos recursos tecnológicos no meio ambiente, conforme SILVA *et al.* esta prática está sendo denominada de TI Verde ou *Green* TI, e não se restringe apenas a isso, também inclui-se o descarte responsável e a reciclagem dos componentes eletrônicos, além de outros pontos importantes como a virtualização de servidores e criação de grandes parques computacionais.

A rápida evolução dos requisitos para processamentos computacionais das aplicações científicas, dos ambientes distribuídos e da própria internet, aliados a mudanças na arquitetura do hardware são grandes catalisadores para a propulsão desta forma de computação, o paralelismo é uma realidade, inclusive para os computadores *desktops* com a popularização dos processadores multicores, exemplos disso são os produtos da Intel chamados de i3, i5 e i7, todos eles fazem uso dessa tecnologia de múltiplos núcleos para computadores de uso doméstico.

O principal objetivo da computação paralela é fracionar o trabalho em partes menores e distribuí-los entre os processadores disponíveis no sistema, conseguindo assim uma redução no tempo gasto para o processamento do algoritmo, caso este fosse processado sequencialmente. Essas frações recebem o nome de *tasks*, ou no português, tarefas. De acordo com PENHA, CORRÊA e MARTINS, os sistemas paralelos são uma coleção de elementos que comunicam e cooperam entre si para resolver problemas com alta performance.

Fazer uso do paralelismo ofertado por estes sistemas é algo que requer habilidades dos programadores, pois identificar que trechos do código podem ser paralelizados é uma tarefa que, inicialmente, pode trazer dificuldades ao programador, pois o trecho que será particionado em *tasks* deve permitir que estas tarefas sejam executas em qualquer ordem sem promover alterações ao resultado final do algoritmo, esse tipo de paralelismo é dito explícito. Também é possível trabalhar com paralelismo implícito, ou seja, o compilador ou sistema de execução irá identificar os trechos de potencial paralelismo e atribuir as tarefas para execução em paralelo.

ISBN 978-85-62830-10-5

Para PEDRONETTE, o *software* representa muitas vezes um grande obstáculo para a adoção da computação paralela, tendo em vista que ela está intrinsecamente ligada ao desempenho e que, nesse sentido, a interação *software* e *hardware* deve ser estreita com o objetivo de conseguir extrair o máximo de desempenho do sistema.

O presente trabalho tem como objetivo principal realizar uma abordagem inicial a computação paralela, destacando os principais conceitos da área e ilustrando alguns dos benefícios encontrados na utilização adequada de um sistema paralelo. Para isso será utilizado um algoritmo com a finalidade de realizar a estimativa do número π (Pi) através do método de Leibniz de forma sequencial e paralela, analisando os resultados obtidos.

As seções estão organizadas para permitir um rápido entendimento do assunto dando ênfase a realização do experimento e aos resultados obtidos. Na seção 2 é feito a descrição do método de Leibniz, a API OpenMP e detalhada as etapas da realização do experimento. Em seguida, na seção 3 é exposto os resultados obtidos e sua discussão. Por fim, na seção 4 é feita a conclusão do trabalho, além das expectativas para trabalhos futuros.

2. MATERIAL E MÉTODOS

Com intuito de observar os benefícios do recurso de paralelismo que podem ser oferecidos por um *hard*ware de múltiplos núcleos, optou-se por realizar neste equipamento um cálculo bastante comum na área acadêmica, a estimativa do número π (Pi) através do método de séries infinitas. O mecanismo é bastante popular, pois é um dos algoritmos que apresenta melhor desempenho para tal função. Ele foi elaborado por Gottfried Leibniz em 1682 utilizando-se a série de Taylor conforme a Figura 1.

$$\pi = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots$$

Figura 1 – Cálculo de π utilizando o método de Leibniz

O hardware utilizado para a execução do código na linguagem de programação C foi um notebook Dell Inspiron N5010 com processador Intel Core i5 M450 2.4GHz 64 bits com o sistema operacional Linux distribuição Ubuntu 11.10.

O experimento consistiu em demonstrar como é possível obter ganho de desempenho através da utilização adequada das técnicas de paralelismo, sendo assim foi observado o comportamento do processador e seus núcleos, além do tempo gasto para realização do cálculo proposto.

O código paralelizado foi desenvolvido em OpenMP, isto é, uma interface de programação (API), portável, baseada no modelo de programação paralela de memória compartilhada para arquiteturas de múltiplos processadores. Conforme descrito por PENHA, CORRÊA e MARTINS, a API OpenMP provê diretivas de compilador e rotinas de bibliotecas e variáveis, embutidas no código fonte C/C++, para escopo de dados, especificação e compartilhamento de carga de trabalho, e ainda, criação e sincronização de *threads*.

Além da portabilidade já citada, a API destaca-se também pelo ambiente de programação amigável, pois o programados não necessita de lidar diretamente com *threads*, isto se deve principalmente a geração automática do código de gerenciamento e disparo das mesmas pelo compilador, a partir da especificação das regiões paralelas no código fonte através de diretivas OPenMP, conforme ilustrado na Figura 2.

ISBN 978-85-62830-10-5

```
#pragma omp parallel for reduction (+:pi)
for(i = 0; i < NUM_PASSOS; i++){
   pi += 4.0 / (4.0*i + 1.0);
   pi -= 4.0 / (4.0*i + 3.0);
}
```

Figura 2 – Especificação de região paralela no algoritmo para estimativa de π(Pi) em OpenMP

Todo programa OpenMP utiliza paralelismo explícito e é baseado no modelo Fork-Join, ou seja, ele inicia como um processo simples, com uma única *thread* chamada *thread master*. Por usa vez, este executa todo trecho sequencial, entretanto ao chegar a uma região paralelizada, ela cria um conjunto de *threads* chamados de *team* que irão executar as tarefes simultaneamente conforme disponibilidade dos núcleos do processador. Esse conjunto de *threads* é sincronizado e finalizado ou deparar-se com um *join*, permanecendo assim apenas a *thread master* ativa até a conclusão do algoritmo ou depara-se com nova região paralelizada com a repetição do processo descrito anteriormente. A Figura 3 ilustra todo o comportamento do modelo Fork-Join utilizado pelo OPenMP.

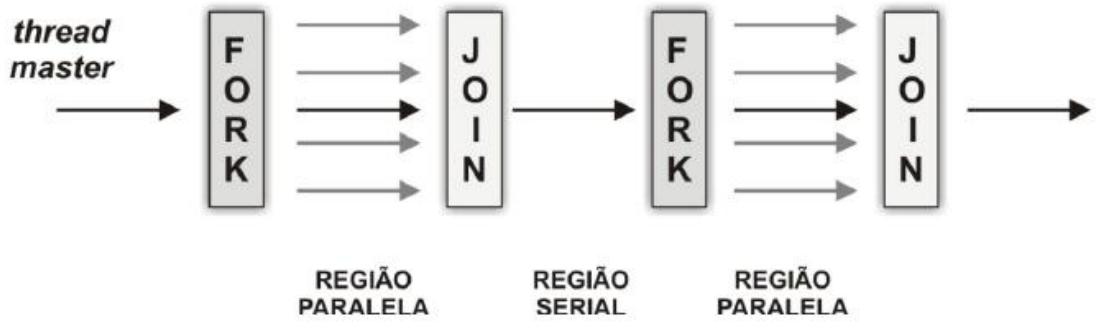

Figura 3 – Modelo Fork-Join adotado pelo OpenMP

3. RESULTADOS E DISCUSSÃO

Para capturar as informações sobre o estado da utilização do processador durante a execução dos algoritmos serial e em paralelo, foi utilizado o próprio monitoramento do sistema operacional Ubuntu 11.10. A Figura 4 retrata o algoritmo sequencial sendo executado, o tempo de execução total foi de 28,270236 segundos, entretanto fica muito claro que apenas um dos quatro núcleos disponíveis, neste caso o CPU1, foi utilizado durante o processo.

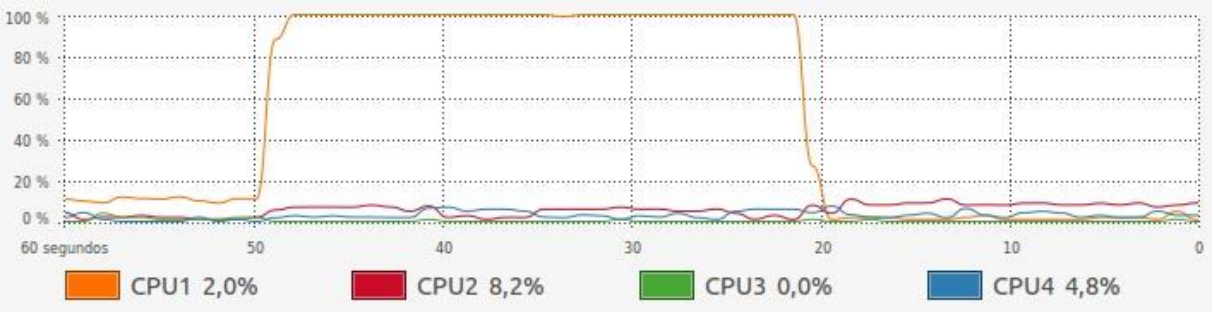

Figura 4 – Comportamento do processador durante execução do código sequencial

A Figura 4 retrata claramente a ineficiência do código, visto que durante a execução do algoritmo ocorreu ociosidade por grande parte dos recursos de *hardware* disponíveis. Entretanto, a Figura 5 mostra a execução do algoritmo em paralelo utilizado a API OpenMP. Neste segundo momento o gasto de tempo diminui consideravelmente, foi de 8,594432 segundos, ou seja, melhora de 69,59% no tempo gasto, isso é proporcionado pela utilização completa do recurso disponível, conforme Figura 5 os quatros núcleos tem atividade em 100% durante a execução.

ISBN 978-85-62830-10-5

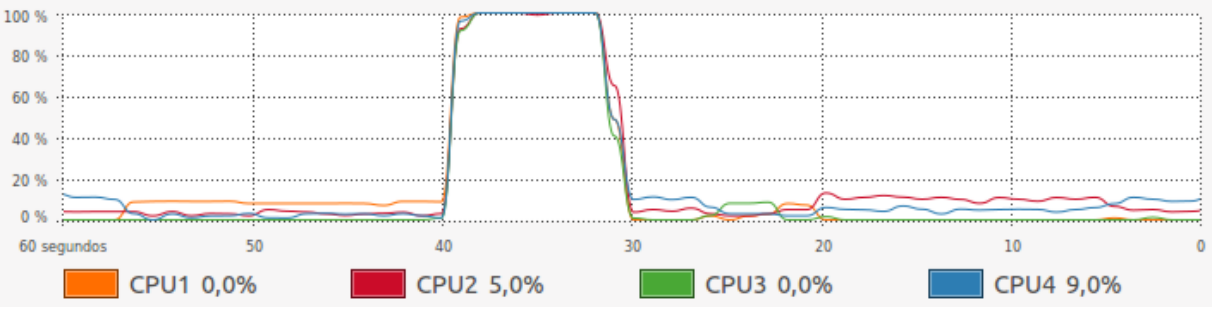

Figura 5 – Comportamento do processador durante a execução do código paralelizado

Para observar o comportamento do processador diante de uma carga maior de dados, foi elaborado um *script shell* que exigia a execução do código 100 vezes consecutivas, isso se fez necessário também para obter um valor mais expressivo do tempo gasto com cada algoritmo, pois diante de uma amostra maior é possível se obter uma maior precisão nos resultados obtidos, afinal uma única execução muitas vezes não retrata a realidade de um sistema. Para execução do *script* com o algoritmo serial foi gasto 47 minutos e 51,0620 segundos, enquanto no paralelo 19 minutos e 45,0360 segundos.

Algoritmo	Tempo
Serial	28,249440
Paralelo	11,848281

Tabela 01 – Tempo Médio do processamento dos algoritmos serial e paralelo

Como o principal objetivo de um processo de paralelização de um algoritmo é obter ganho no desempenho, ele foi plenamente alcançado. O tempo de execução foi reduzido em aproximadamente 58,05%. Dessa forma o *speedup* é 2,38. Isso apenas confirma os ganhos oriundos da paralelização, afinal *speedup* é razão entre o tempo gastos com o processamento linear, ou serial, e o paralelo. O cálculo é feito conforme a Figura 6.

$$Speedup\ s\ = \frac{tempo(linear)}{tempo\ (paralelo)} = \frac{28{,}249440}{11{,}848281} \cong 2{,}38$$

Figura 6 – Cálculo do *speedup* do algoritmo de cálculo de π(Pi)

4. CONCLUSÕES

O artigo procurou tratar dos benefícios da computação da paralela e os seus motivadores. Atualmente, a preocupação com o *hardware* do sistema comprova que cada vez mais os programadores terão que entender com que tipo de arquitetura estão lidando.

Com o exemplo utilizado, o cálculo de π(Pi) ficou comprovado que quando utilizado de maneira correta o paralelismo é capaz de oferecer benefícios consideráveis ao desempenho de um algoritmo. Portanto, nos dias de hoje, o paralelismo é algo indispensável e deve ser utilizado sempre que possível.

Pretende-se no futuro conseguir monitorar o comportamento de cada núcleo dos processadores e ainda paralelizar novos algoritmos. Também é de interesse desenvolver algoritmos para computação distribuída que trabalhem com *clusters* e *grids*.

REFERÊNCIAS

ISBN 978-85-62830-10-5

ALVES JÚNIOR, A. R. **O que é e como funciona o paralelismo.** Disponível em: <http://assemblando.wordpress.com/2010/09/24/o-que-e-e-como-funciona-o-paralelismo/> Acesso em: 11 ago 2012.

HILL, M. D., MARTY, M. R., **Amdahl's Law in the Multicore Era.** Disponível em: <http://research.cs.wisc.edu/multifacet/papers/tr1593_amdahl_multicore.pdf> Acesso em: 01 ago 2012.

LOTOFU, R. A. **Sistemas Operacionais.** Disponível em: <http://www.dca.fee.unicamp.br/~lotufo/cursos/EA877/sistemas-operacionais/exemplos/tempo.c> Acesso em: 01 ago 2012.

PEDRONNETE, D. C. G. **Programação Paralela.** Disponível em: <http://www.ic.unicamp.br/~dcarlos/publicacoes.html> Acesso em: 10 ago 2012.

PENHA, D. O. da, CORRÊA, J. B. T., MARTINS, C. A. P. S. **Análise Comparativa do Uso de Multi-Thread e OpenMP Aplicados a Operações de Convolução de Imagem.** Disponível em: <http://www.ppgee.pucminas.br/gsdc/papers/penha_wscad02.pdf> Acesso em: 30 jul 2012.

ROSA, F. H. F. P., COSTA, M. M., TORTELLA, T. L., APARECIDO JUNIOR, V. **Métodos de Monte Carlo e Aproximações de π.** Disponível em: <http://www.feferraz.net/files_/lista/montecarlopi.pdf> Acesso: 30 jul 2012

SILVA, M. R. P. da, ZANETI, G. B., ZAGO, M. G., et al. **TI Verde – Princípios e Práticas Sustentáveis para Aplicação em Universidades.** Disponível em: <http://www.labplan.ufsc.br/congressos/III%20SBSE%20-%202010/PDF/SBSE2010-0085.PDF> Acesso em: 10 ago 2012.

ISBN 978-85-62830-10-5