A utilização de paralelismo em um programa de cálculo de números primos utilizando a biblioteca OpenMP

João Paulo Cunha Ávila  
Departamento de Ciência da Computação   
Centro Universitário de Brasília   
Brasília, Brasil

Hiago Vitor  
Departamento de Ciência da Computação   
Centro Universitário de Brasília

Brasília, Brasil

*Resumo*— Este artigo apresenta o desenvolvimento de uma aplicação de calculo de números primos utilizando a API (Application Programming Interface) OpenMP para realizar a paralelização deste processo. Foram utilizadas três funções fornecidas pela API e medidos o speed up e eficiência de cada um com um quantidade n de threads. Os resultados demonstram o real ganho de desempenho da aplicação.

Palavras-chaves—Paralelismo, OpenMP, Threads

# INTRODUÇÃO

O software proposto engloba o cálculo dos números primos, até um determinado valor, números estes que não são divisíveis por nenhum número menor do que ele mesmo, com exceção do 1 (COUTINHO, 2003).

Devido ao alto processamento presente na aplicação, o uso de paralelismo se torna uma forma de melhorar o desempenho da mesma. Para isto, foi utilizada a API OpenMP.

A API OpenMP é composta por uma interface na qual permite gerenciar o paralelismo da aplicação de modo simples, segundo Silva et al. (2008).

Os sistemas paralelos, ou programação paralela, são elementos de processamentos que fazem uma comunicação e uma cooperação, juntos, afim de resolver problemas com alta performance (ALMASI; GOTTLIEB, 1994).

Este artigo tem como objetivo demonstrar o uso de três funções de paralelismo presentes na API em questão aplicado ao problema já mencionado acima. Será demonstrado a eficiência dos métodos utilizados, tal como a comparação entre os mesmos para a identificação do mais adequado na evolução do problema. O artigo conta como forma de organização 5 sessões, sendo essas, base teórica, desenvolvimento, resultados e conclusão.

# BASE TEÓRICA

Antes de iniciar o desenvolvimento do programa, é necessário conceituar os termos que serão apresentados no decorrer deste artigo. Sendo assim, a partir de pesquisas selecionadas na literatura cientifica brasileira e estrangeira, a programação paralela é a programação concorrente orientada para computadores paralelos, incorporando, assim, os seus requisitos de sincronização e comunicação, a fim de buscar uma utilização adequada dos recursos de processamento com o objetivo de otimizar o seu desempenho (SATO; et al., 1996).

Algo importante de se salientar é a diferença entre a computação distribuída e paralela. Enquanto a paralela possui diversas pequenas tarefas trabalhando juntas para resolver um problema, a distribuída vai possuir aplicações trabalhando juntas para resolver um problema (PACHECO, 2011).

A paralelização ocorre basicamente quando um processo vai ter suas tarefas divididas em blocos, as threads, que vão ser processados separadamente e de uma maneira que dá impressão de simultaneidade. Logo, a execução desse vai se tornar mais rápida e menos custosa, pois a execução das threads são divididas para cada núcleo do processador. Desta maneira, se houverem 4 threads e 4 núcleos, cada um destes executa uma thread.

Para que isto seja possível, é necessário que uma aplicação informe as informações necessárias para o paralelismo. O OpenMP entra como um facilitador destas operações, por fornecer uma boa quantidade de funções que ajudam a realizar este processo.

Sua principal característica é permitir que um programa serial seja paralelizado ao invés de se codificar um paralelo desde o princípio (PACHECO, 2011).

# DESENVOLVIMENTO

Pode ser aplicado a utilização de threads em diversos cenários, mas vale ressaltar que não é sempre que se tem benefício com utilização da mesma.

Foi utilizado cálculo dos primeiros um milhão e meio de números primos, para desse modo obter resultados de tempo significativos. Realizou-se a comparação usando “x” a “x elevado ao cubo” threads, sendo x o número de cores do processador da máquina utilizada no experimento. Vale ressaltar que foi utilizado raiz de y, na implementação dos números primos, sendo que y é a quantidade de números primos.

Reduction consiste em cada thread ter uma variável privada de uma variável que é incrementada (como por exemplo um contanto dentro de um for), para que desse modo ao ser finalizado o processo, seja somado todos os valores das threads obtendo o valor do contador.

“Simples” que é a implementação básica do OpenMP de paralização, em que cada thread recebe uma parte do programa para execução, mesmo que essa parte seja muito menor que a de outro thread. Portanto, pode existir uma thread Z que leva 3x mais tempo de executando que a Y.

Scheduling consiste basicamente na divisão do programa (ou parte dele) em grandes pedaços de tempo de execução bem próximos, ou seja, se for utilizado quatro threads cada uma delas vai obter aproximadamente 25%. Logo, nesse caso as threads vão ter tempos de execução (CPI) próximos.

Portanto, o esperado no caso da implementação dos números primos é que a Scheduling seja a que vai ter o com os melhores resultados, devido ao fato de teoricamente (conceito) ela utilizar melhor as threads, por não deixar as mesmas expostas a um período ocioso muito grande (se comparado com as outras implementações testadas).

A máquina utilizada nos testes foi um DELL com sistema operacional Ubuntu que possui as seguintes especificações:

\* i7-4500U CPU - 1.80GHz - 2 Cores

\* 8GB RAM

\* Sistema operacional 64 bits

# RESULTADOS

Foram realizados 10 testes em cada método utilizado para cada uma de 8 threads. Serão utilizadas as seguintes legendas para os métodos:

* M1: #pragma omp parallel num\_threads()
* M2: #pragma omp parallel for num\_threads()
* M3: #pragma omp parallel num\_threads() schedule(static,1)

Os resultados do tempo de execução em cada thread estão representados, em segundos, na tabela abaixo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **serial** | **m1** | **m2** | **m3** |
| **1** | 158.61 | 180.6 | 182.37 | 146.79 |
| **2** | 158.61 | 92.42 | 132.6 | 81.3 |
| **3** | 158.61 | 80.95 | 112.26 | 76.56 |
| **4** | 158.61 | 74.88 | 100.37 | 74.88 |
| **5** | 158.61 | 91.56 | 91.72 | 68.74 |
| **6** | 158.61 | 97.64 | 89.57 | 67.84 |
| **7** | 158.61 | 90.12 | 97.31 | 66.96 |
| **8** | 158.61 | 99.17 | 96.14 | 67.69 |

E a partir destes foram feitos o gráfico e tabela de SpeedUp:

O SpeedUp mede a velocidade de execução paralela do programa em relação à sua execução serial. É aparente a melhora neste quesito quando são utilizados parâmetros de paralelização, que chega a até quase 150%.

Entre os métodos o que possui maior velocidade é o método 3, que faz a distribuição igualitária dos números para cada thread, ou seja, em uma distribuição de 10 números utilizando 2 threads, por exemplo, cada thread executará o próximo número da sequencia. Isso faz com que todas trabalhem em uma velocidade muito maior do que simplesmente dividir a quantidade de números pela de threads e cada uma executar um conjunto desta separação.

Também é possível perceber que os métodos 1 e 2 começam a oscilar suas velocidades a partir da 5ª thread, sendo que neste momento o método 1 tem uma enorme perda de velocidade. Isso se dá pelo fato de que neste momento o processador está fazendo as trocas de contexto em cada core.

A partir da tabela de tempo de execução foi possível montar o gráfico de eficiência para cada método:

Neste gráfico, percebe-se que o programa perde eficiência em todos os métodos utilizados conforme é aumentado o número de threads. Este resultado se dá pelo motivo de que conforme mais threads são utilizadas os cores vão ter de revezar o uso do mesmo para cada uma das que estão o utilizando. Isso diminui muito a eficiência de execução.

Outro motivo é a divisão das tarefas para as threads, sendo que em alguns momento algumas vão ficar ociosos, sendo que podiam estar resolvendo mais processos.

# CONCLUSÃO

Foi possível concluir que a utilização de thread diminui consideravelmente o CPI, no entanto foi possível verificar que a eficiência vai abaixando consideravelmente, quando a quantidade de threads é maior que a quantidade de core vezes dois. Vale ressaltar que a perda foi considerável ao ser ultrapassado 4 cores, apesar de não ser tão grande quanto a de mais de dois cores, pois ao ser ultrapassado 4 cores (2 cores físicos e 2 virtuais) não existirá uma thread para cada núcleo, portanto realmente o esperado é que ocorra novamente uma queda considerável na eficiência.

Portanto, o ideal é utilizar duas vezes a quantidade de cores para número de threads, isto acontece devido aos cores virtuais, pois eles não vão possuir o mesmo desempenho que os físicos. Então, se existir dois cores “físicos”, terá dois cores virtuais.

Foi possível observar que quando foi utilizado o Scheduling o desempenho foi consideravelmente superior como pode ser observado na parte de resultados, logo é possível constatar que o problema das outras implementações o problema é a má utilização das threads.

Logo, é possível afirmar que devido ao tempo ocioso nas outras implementações, foi gerado resultados inferiores a 3 implementação, a qual obteve resultados bem acima na maior parte dos testes.

##### References

1. PACHECO, Peter. An introduction to parallel programming. Elsevier, 2011.
2. CAVALHEIRO, Gerson GH. Princípios da programação concorrente.
3. DA COSTA, Celso Maciel; STRINGHINI, Denise; CAVALHEIRO, Gerson Geraldo Homrich. Programação Concorrente: Threads, MPI e PVM. Escola Regional de Alto Desempenho, II ERAD, captulo, v. 2, 2002.bbs
4. SATO, Liria Matsumoto; MIDORIKAWA, Edson Toshimi; SENGER, Hermes. Introdução a programação paralela e distribuída. Anais do XV Jornada de Atualização em Informática, Recife, PE, p. 1-56, 1996.
5. ALMASI, George S.; GOTTLIEB, Allan. Highly parallel computing. 1994.
6. SILVA, Marcelo O. et al. O Sistema Integrado do Time de Futebol de Robôs USPDroids. Team Description Paper on Latin American Robots Competition, 2008.
7. COUTINHO, S. C. Primalidade em tempo polinomial: uma introduc ao ao algoritmo AKS. 2003.
8. LEANDRO, Avaliação de Desempenho em OpenMp em arquiteturas paralelas

9. INTEL, getting started with OpenMP <<https://software.intel.com/en-us/articles/getting-started-with-openmp>>