UNIVERSIDADE FEDERAL DE MINAS GERAIS
Documentação
Trabalho Prático 1 de Programação Paralela

André Gomes Heringer • Marco Túlio Motta de Jesus
andregomesh@gmail.com • marcotmotta@gmail.com

Resumo. *Este trabalho tem como objetivo desenvolver um estudo sobre princípios de paralelismo para otimização de algoritmos. A análise consiste em diversas execuções considerando variações de parâmetros de entrada buscando identificar mudanças de desempenho e eficiência na resolução do problema proposto.*

Palavras Chave. *Paralelismo, Threads, OpenMP*

1. Introdução

Os fundamentos de Programação Paralela quando relacionados com diversos outros aspectos da área de conhecimento de otimização de algoritmos são fortemente impactantes no universo da Ciência da Computação como um todo.

Esse trabalho trata de um problema clássico onde dado o número inteiro e positivo n se busca encontrar todos os números primos de 2 até n. Inicialmente será implementado um algoritmo que resolve o problema proposto de maneira sequencial e, eventualmente, uma versão paralela será feita para que, fazendo uso de um número maior de *threads*, possamos otimizar o tempo de execução.

2. Metodologia

A implementação do programa sequencial transcorreu sem grandes problemas. Seguindo o algoritmo sugerido na especificação do trabalho os resultados foram obtidos como esperado. Alguns testes foram feitos com o algoritmo sequencial e estes serão detalhados logo a frente na *sessão 3*.

Logo em seguida a implementação do algoritmo paralelo foi iniciada. Para tal fizemos uso da interface de programação de aplicativo (API) OpenMP, que é responsável por gerenciar grande parte do processo de criação e uso de threads ao longo da execução.

A metodologia usada foi relativamente simples: a primeira thread realiza a enumeração que marca como não primos todos os múltiplos de 2. A próxima thread faz o mesmo para o primo subsequente (no caso 3), já que essa ação não depende da ação

realizada pela thread anterior. Esse processo é repetido enquanto houver threads livres até que o ciclo se complete, associando a primeira thread ao próximo primo não visitado e assim por diante. Essa decisão foi tomada pois, em etapas iniciais do desenvolvimento o escalonamento estava prejudicando consideravelmente a execução. Como da forma descrita acima é assinalado apenas um número por vez para cada thread nós otimizamos esse processo para um funcionamento mais balanceado e dinâmico. Ao final, teremos o mesmo resultado obtido pelo algoritmo sequencial mas, possivelmente, em um tempo suficientemente menor.

3. Testes

Utilizando das técnicas descritas na seção anterior foram realizados diversos testes para verificar a eficiência do algoritmo. A fase inicial de testes foi focada em configurar o ambiente para que fosse possível rodar o programa no Microsoft Azure, o serviço de nuvem da Microsoft. Essa tarefa, entretanto, acarretou alguns problemas de modo que a execução do código na nuvem foi prejudicada. Utilizamos, então, como plano alternativo, as máquinas disponíveis nos laboratórios do DCC, e os resultados estão descritos abaixo.

Os primeiros testes feitos foram com o programa sequencial. Variamos os valores de entrada de modo que pudemos perceber claramente a diferença de desempenho do algoritmo para números muito grandes, como segue nas *figuras 1-5*. Os gráficos representam o tempo de execução em segundos das respectivas execuções.

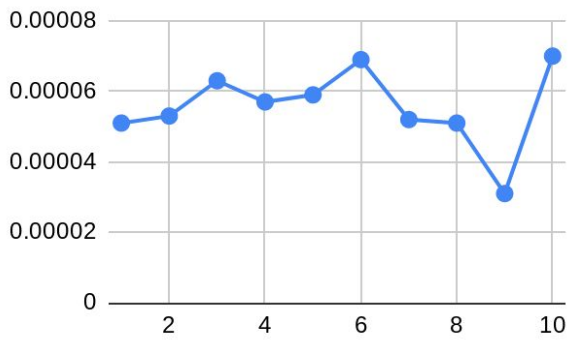

Figura 1. *Tempo de execução sequencial (10^2). Média = 0.0000556.*

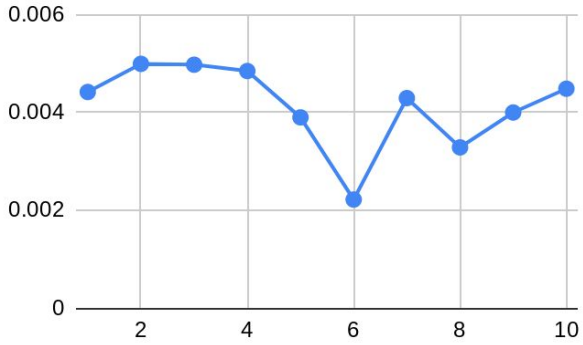

Figura 2. *Tempo de execução sequencial (10^3). Média = 0.0041436.*

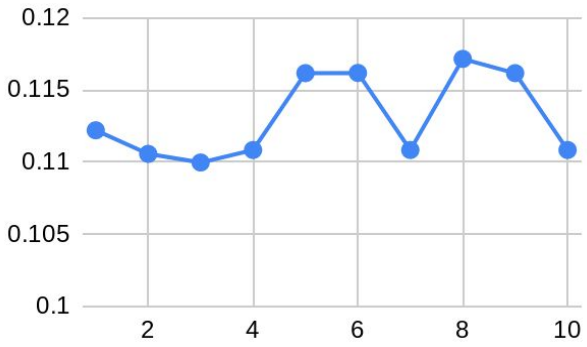

Figura 3. *Tempo de execução sequencial (10^4). Média =* 0.1130977.

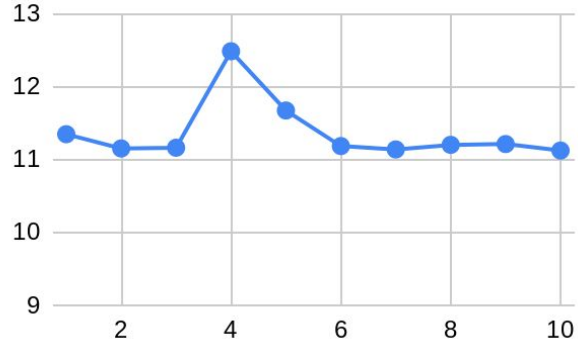

Figura 4. *Tempo de execução sequencial (10^5). Média =* 11.3729232.

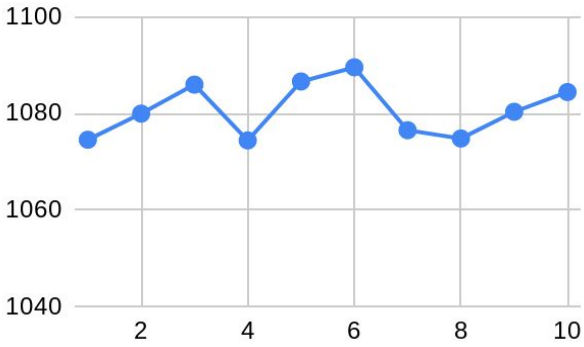

Figura 5. *Tempo de execução sequencial (10^6). Média =* 1080.75691.

Em seguida o programa em paralelo foi testado. Realizamos os mesmos testes descritos acima mas agora com números variados de threads, com o objetivo de verificar ganhos de desempenho. As comparações para as diferentes cardinalidades dos valores de entrada estão mostradas nas *figuras 6-10*. Os gráficos mostram o tempo de execução em segundos do algoritmo (eixo vertical) com as respectivas entradas e as respectivas quantidades de threads (eixo horizontal). O tempo sequencial está incluso nessas representações para podermos comparar diretamente.

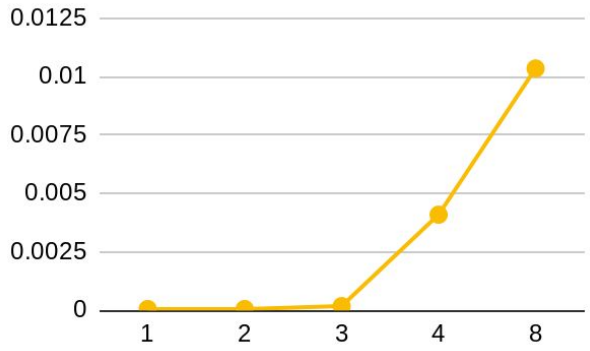

Figura 6. *Tempo de execução paralelo (10^2).*

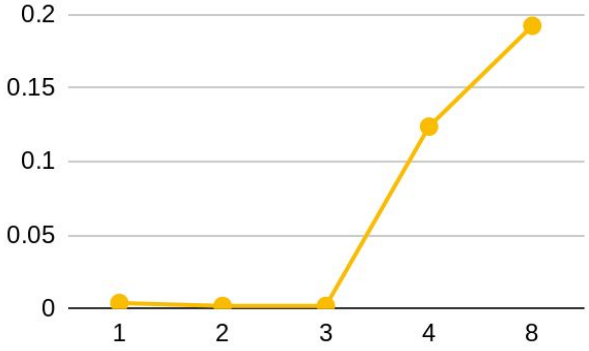

Figura 7. *Tempo de execução paralelo (10^3).*

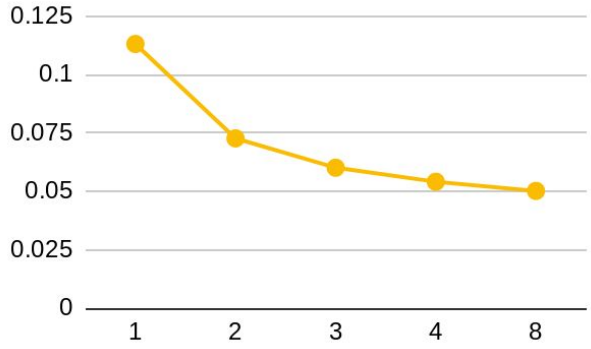

Figura 8. *Tempo de execução paralelo (10^4).*

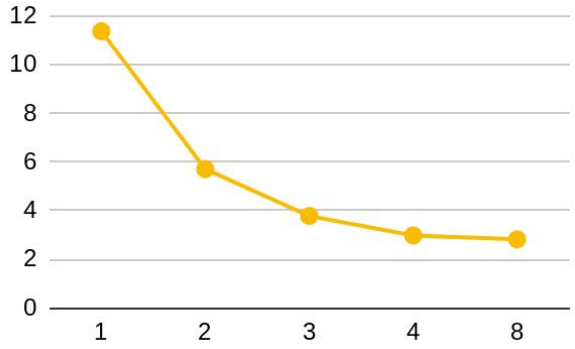

Figura 9. *Tempo de execução paralelo (10^5).*

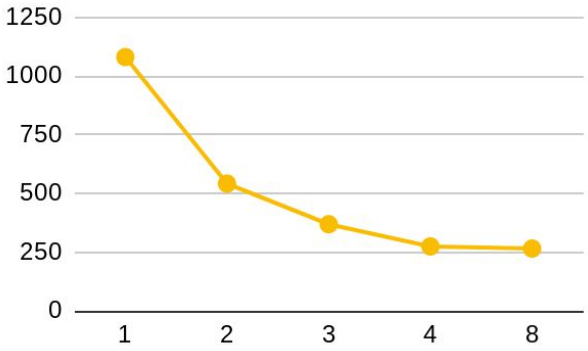

Figura 10. *Tempo de execução paralelo (10^6).*

Podemos perceber que, para uma entrada pequena, a paralelização prejudica a execução, visto que o custo para criar as threads é provavelmente maior do que o benefício gerado por elas. Para entradas suficientemente grandes, entretanto, a paralelização otimiza o algoritmo, fazendo com que ele rode em uma velocidade consideravelmente maior. Os respectivos *speedups* estão tabelados abaixo.

Speedup	$N = 10^2$	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$
2 Threads	**1.010909091**	**1.935266919**	**1.555569768**	**1.990819935**	**1.993168559**
3 Threads	**0.3088888889**	**1.917356902**	**1.880042223**	**3.004220995**	**2.925026012**
4 Threads	**0.013587487778**	**0.033443099227**	**2.088368786**	**3.809168257**	**3.930475674**
8 Threads	**0.005374577099**	**0.021543100766**	**2.251282919**	**4.024604004**	**4.064579606**

Para reproduzir os testes entre na pasta raiz do projeto (primes_counter) e execute o comando “make release”. Um executável será gerado na pasta “release” do projeto, em seguida execute o arquivo passando os parâmetros desejados. (e.g. “./release/primes 1000 a 2”)

4. Conclusão

Implementar e analisar esse trabalho foi uma tarefa muito interessante e repleta de desafios mas, apesar de tudo, foi muito gratificante chegar ao final com um algoritmo pronto.

Com base nos testes feitos, podemos concluir que a implementação cumpriu seu objetivo dado que obtivemos speedups consideravelmente bons durante as diversas execuções paralelas do algoritmo.

Resumidamente, toda a teoria aprendida com relação a Programação Paralela no decorrer desse trabalho será de grande valia para o futuro, tanto nessa área como em outras tangentes.

5. Bibliografia

1. OpenMP - Documentation https://www.sharcnet.ca/help/index.php/OpenMP
2. Moodle da disciplina Programação Paralela 2019/02