Multiplicação de matrizes utilizando *multithreading*

Willian Bandeira Alves <willian.alves.001@acad.pucrs.br>

Roland Teodorowitsch[[1]](#footnote-1) <roland.teodorowitsch@pucrs.br> – Professor

Pontifícia Universidade Católica do Rio Grande do Sul – Faculdade de Informática – Curso de Sistemas de Informação

Av. Ipiranga, 6681 Prédio 32 – Bairro Partenon – CEP 90619-900 – Porto Alegre – RS

25 de setembro de 2018

Resumo

Este artigo descreve a realização do trabalho da cadeira de Sistemas Operacionais do curso de Sistemas de Informação da PUCRS, onde foram aplicados os conceitos de *multithreading* na multiplicação de matrizes, comparando o tempo de execução do mesmo com a multiplicação sequencial – disponibilizada pelo professor – que utiliza apenas uma *thread* do processador, a fim de identificar qual dessas execuções obtém o melhor desempenho com o *hardware* utilizado.

**Palavras-chave:** Sistemas Operacionais; *Threads;* Multiplicação de Matrizes.

Abstract

**Title:** “Matrices multiplication utilizing multithreading”

This paper describes the work of the Operational Systems subject of PUCRS’s Information Systems course, where the concepts of multithreading were applied into matrices multiplication, comparing execution times of this application with a sequential multiplication – provided by the professor – which utilizes just one thread, looking for the execution which has the best performance utilizing the selected hardware.

**Key-words:** Operational Systems; Threads; Matrices Multiplication.

1. Introdução

Atualmente os computadores modernos possuem uma grande capacidade de processamento que vem sendo explorada por diversas empresas e desenvolvedores. Torna-se cada vez mais comum identificar dispositivos, como computadores e celulares, com processadores *dual-core, quad-core,* ou até mesmo *octa-core,* visto que eles vêm ficando mais baratos e amplamente disponíveis (OAKS, 2004, p. 7)*.* Com esse aumento na quantidade de núcleos por processador, surge também a necessidade de utilização dos mesmos, explorando diferentes abordagens para adquirir o melhor desempenho possível.

Como proposta de trabalho para a cadeira de Sistemas Operacionais do curso de Sistemas de Informação, foi disponibilizado o cenário de multiplicação de matrizes sequencial, podendo variar o tamanho das mesmas. Nesse contexto, deve-se explorar o uso de processamento concorrente – ou *multithreading* – para fazer melhor uso de mais núcleos do processador e comparar os resultados dos dois tipos de processamento.

Dessa maneira, seguindo os requisitos propostos, deve-se variar o tamanho das matrizes de 100 em 100, iniciando em 100 até 2000 e, alternando também, a quantidade de *threads* utilizadas, que deve variar de 2 até o dobro de núcleos do processador utilizado para realizar a multiplicação. Essas tarefas devem ser executadas algumas vezes a fim de adquirir dados concisos e suficientes para a comparação entre os dois métodos: sequencial e *multithreading.*

A Seção 2 descreve como foi o desenvolvimento do trabalho, divido em subseções que descrevem a multiplicação sequencial, a paralela e a comparação entre os dois métodos. Por fim, na Seção 3 são apresentadas as conclusões.

1. Desenvolvimento

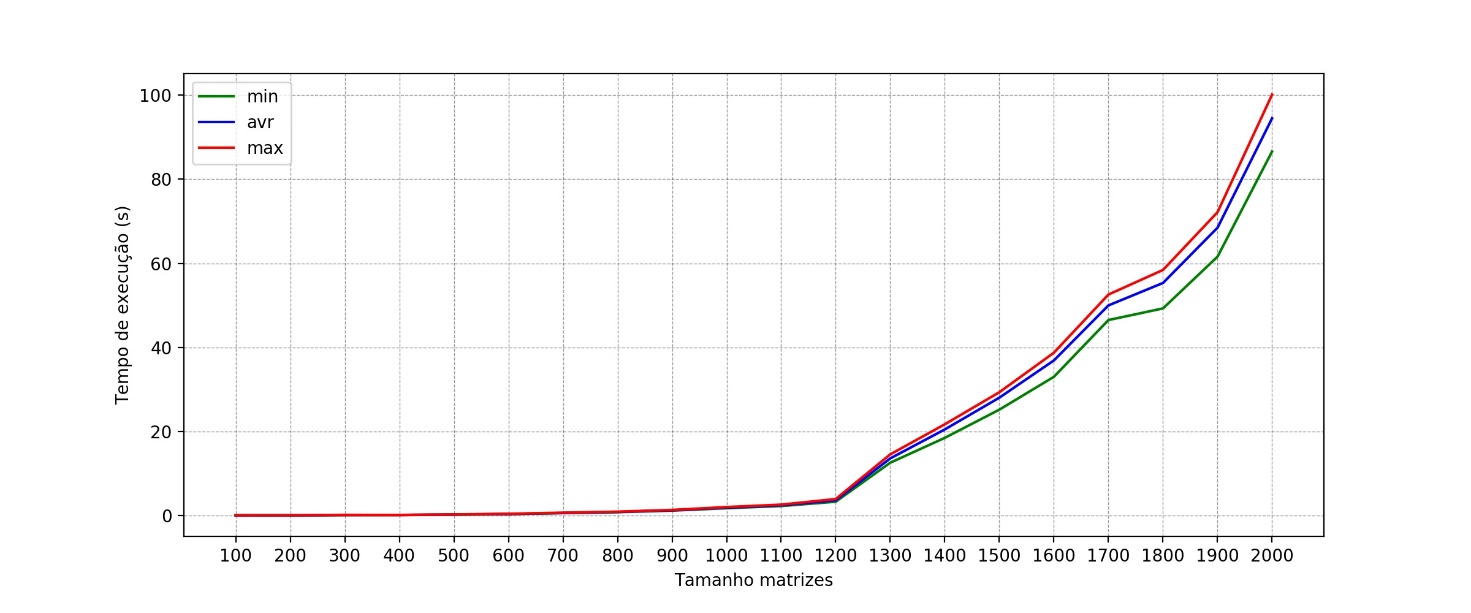
Para iniciar o desenvolvimento do trabalho, foi disponibilizado na plataforma *Moodle* da PUCRS o código em Java demonstrando a multiplicação de matrizes no método sequencial. Esse foi o modelo utilizado para o desenvolvimento do método paralelo, também em Java, utilizando o conceito de *threads* e *multithreading*.

A nível de desenvolvimento, ambos os métodos possuem alguns requisitos a serem seguidos. Na execução sequencial o tamanho das matrizes deve variar de 100 em 100, começando em 100 até 2000. Na execução paralela, além desse requisito, deve-se variar a quantidade de *threads* de 2 até o dobro da quantidade de núcleos do processador em que os testes estão sendo realizados.

As execuções de ambos os métodos foram realizadas no mesmo dispositivo, que possui um processador Intel Core i7-7700k, que possui 8 núcleos que operam a 4.20GHz no modo normal e 4.50GHz no modo turbo. Durante a obtenção dos tempos de cada método, ele operou no modo normal, não havendo nenhum processo extra executando, apenas o sistema operacional, que é um Windows 10 Pro.

* 1. Execução sequencial

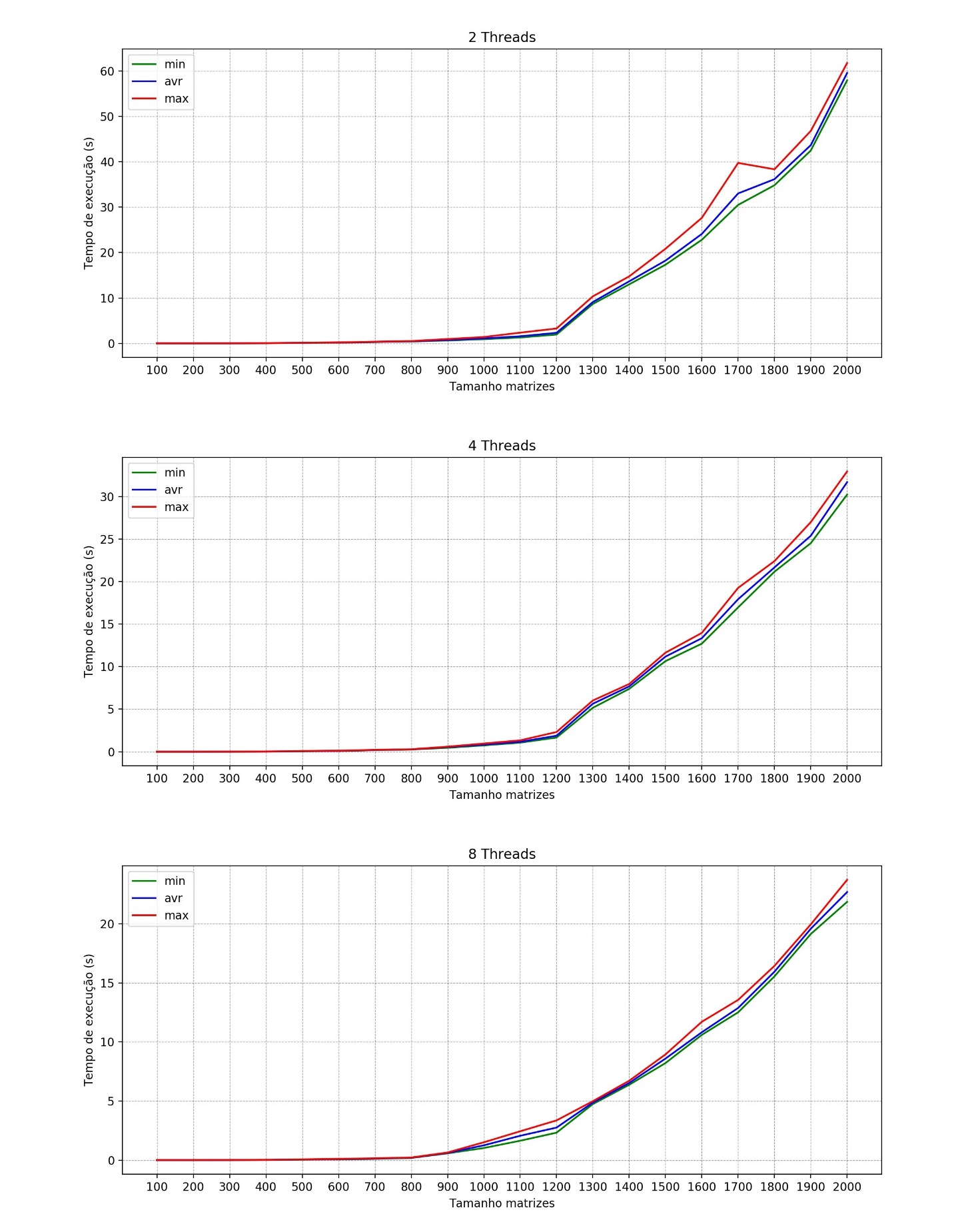
Para obter uma concisão entre os tempos de execução, o código foi executado 10 vezes e, a partir de todos esses valores, foi criado o gráfico ilustrado na Figura 1, que mostra os tempos máximos, médios e mínimos adquiridos. Analisando o mesmo, pode-se ver que houve uma linearidade conforme o tamanho das matrizes aumenta, mantendo uma certa proporção entre aumento de matrizes e aumento do tempo de execução, o que era o esperado.



**Figura 1 - Tempos de execução sequencial**

* 1. Execução paralela

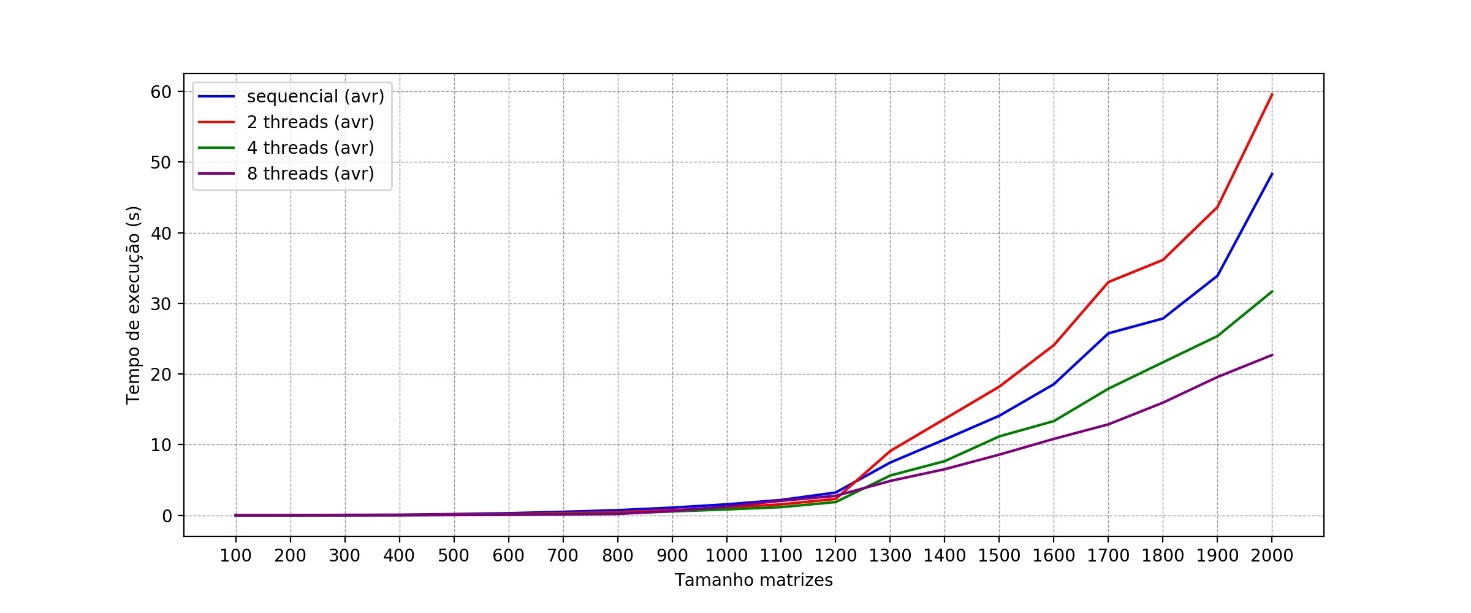
Assim como na execução sequencial, a execução paralela também foi realizada 10 vezes para obter valores concisos. A diferença é que esse método foi divido em duas, quatro e oito *threads*. Portanto, foram realizadas 30 execuções por vez. A Figura 2 mostra o gráfico gerado a partir dos tempos adquiridos para cada execução. Pode-se perceber que na execução paralela com duas *threads*, o tempo de execução é já é consideravelmente menor quando comparado à execução sequencia. Já na execução com quatro *threads*, pode-se notar que o tempo de execução diminuiu, chegando à metade do tempo de execução com duas *threads.* Por fim, na execução com oito *threads* aconteceu o que teoricamente deveria acontecer: ela obteve o melhor tempo de execução quando comparada com as outras execuções com menos *threads*. Aqui, o tempo de execução foi, em média, 30% mais rápido que a execução de quatro *threads.*



**Figura 2 - Execuções paralelas com duas, quatro e oito threads**

* 1. Comparação de execuções

Para identificar qual o melhor desempenho entre todas as execuções feitas, foi feito um gráfico comparativo entre o tempo médio de cada execução paralela e sequencial que pode ser visto na Figura 3. Nessa imagem fica mais nítido o que foi dito na subseção 2.2. Nela, todas as execuções são comparadas utilizando uma média de todos os tempos executados. Além disso, fica notável a diferença entre os tempos de execução, que são inversamente proporcionais à quantidade de *threads* utilizadas naquela execução.



**Figura 3 - Gráfico comparativo entre execução paralela e sequencial**

1. Conclusão

Este artigo apresentou o desenvolvimento do trabalho de Sistemas Operacionais, comparando o desempenho entre aplicações sequenciais – ou *singlethreads* – e aplicações paralelas – ou *multithreads*. Pode-se observar que há diferença de tempos de execução entre os tipos de aplicações, e que, conforme a quantidade de *threads* aumenta, o tempo de execução diminui. Vale ressaltar que, teoricamente, há um momento em que, por mais que a quantidade de *threads* fosse aumentada, o tempo de execução não iria diminuir, pois o processador já estaria trabalhando com a sua melhor performance possível. Portanto, utilizando um processador Intel Core i7-7700k para multiplicação de matrizes, uma aplicação que utiliza oito *threads* é a melhor abordagem, visto que acima disso o tempo de execução é superior.

Referências

OAKS, Scott; WONG, Henry. **Java Threads: Understanding and Mastering Concurrent Programming**. " O'Reilly Media, Inc.", 2004.

1. Professor das disciplinas de Introdução à Ciência da Computação e Programação Distribuída do curso de Ciência da Computação, e da disciplina de Sistemas Distribuídos do curso de Sistemas de Informação, da Faculdade de Informática da PUCRS. [↑](#footnote-ref-1)