Universidade Federal do Rio Grande do Norte

Instituto Metrópole Digital

Análise e comparação de speedup e eficiência do código serial e paralelo no cálculo de multiplicação de matrizes quadradas

João Vitor Venceslau Coelho

Natal/RN
2020

Introdução

Este relatório consiste na explicação da ***Multiplicação de Matrizes Quadradas***, essa explicação busca deixar claros todos os detalhes do problema e das soluções utilizadas, sendo elas: *multiplicar* ***sequencialmente*** as linhas da primeira matriz com as colunas da segunda matriz e multiplicar sequencialmente as linhas da primeira matriz pelas linhas da segunda matriz ***transposta***. As soluções implementadas também serão explicadas, isto é, a versão serial e a paralela dos códigos implementados para cada uma das soluções dos problemas abordados será explanada, sendo que duas versões paralelas foram implementadas para cada abordagem de solução, essas implementações consistem numa versão utilizando o **for** do **OpenMP** e a outra versão utiliza as **tasks** do **OpenMP**. Em seguida os resultados obtidos serão mostrados e comentados. Será também brevemente comentada a corretude dos algoritmos implementados, serão expostas as análises do speedup, da eficiência e da escalabilidade dos algoritmos paralelos para os seriais. Ao final, um resumo das atividades e análises realizadas será relatado, sintetizando os principais pontos do relatório.

Multiplicação de Matrizes Quadradas

A multiplicação de matrizes é uma operação entre matrizes que só é definida caso o número de colunas da primeira matriz seja igual ao número de linhas da segunda matriz. Dadas duas matrizes $A_{m\,x\,n}$ e $B_{n\,x\,o}$, cada elemento c_{ij} da matriz resultante $C_{m\,x\,o}$ é obtido com a soma das multiplicações dos elementos correspondentes da linha i da matriz A com os elementos da coluna j da matriz B. A multiplicação de matrizes quadradas é igual a multiplicação das matrizes não quadradas, desde que respeite a condição para a definição, dada acima, da operação. Logo a multiplicação de matrizes quadradas só está definida para matrizes quadradas de mesma ordem (onde a ordem seria o número de linhas e colunas da matriz).

Desenvolvimento

Abaixo são explicados os algoritmos implementados, tanto as versões seriais quanto as paralelas das soluções utilizadas, sendo as soluções:

- Multiplicar matrizes quadradas sequencialmente nas linhas da primeira matriz;
- Multiplicar matrizes quadradas sequencialmente nas linhas da primeira matriz transpondo a segunda matriz.

A segunda abordagem possui duas etapas a mais em relação à primeira, pois nela a *transposição* da segunda matriz tem que ser calculada e a multiplicação ocorrerá entre as linhas da primeira com as linhas da segunda matriz transposta.

Soluções implementadas

Multiplicar Matrizes Quadradas Sequencialmente nas linhas (Serial)

O algoritmo inicia com a leitura dos argumentos a serem utilizados, sendo eles:

1. **Flag** que indica se as matrizes devem ser exibidas ao fim da multiplicação (0 ou 1);
2. **Seed** a ser utilizada para preencher a primeira matriz;
3. **Seed** a ser utilizada para preencher a segunda matriz;
4. Quantidade de linhas da primeira matriz;
5. Quantidade de colunas da primeira matriz;
6. Quantidade de linhas da segunda matriz;
7. Quantidade de colunas da segunda matriz;

Dito isto, o algoritmo foi feito de forma que realiza a multiplicação de matrizes mesmo quando as mesmas não são quadradas, desde que satisfaçam a condição para definição da operação de multiplicação definida anteriormente, caso essa condição não seja satisfeita, uma mensagem é exibida e o programa finaliza. Após a verificação da condição, com base nas quantidades de linhas e colunas informadas, as devidas matrizes são alocadas e depois utilizando as **seeds**, obtidas anteriormente, preenche-se as matrizes a serem multiplicadas. Após realizar essas etapas, a multiplicação de fato se inicia. Para cada linha da primeira matriz é calculada a linha correspondente da matriz resultante da multiplicação, sendo cada elemento dessa linha calculado, somando o resultado das multiplicação de cada elemento j da linha atual com o elemento correspondente na coluna j da segunda matriz. Após percorrer todas as linhas da primeira matriz, todas as linhas da matriz resultante da multiplicação são devidamente calculadas, dessa forma, tendo calculado a multiplicação, caso a **flag** que indica a exibição esteja igual a 1, é mostrado as matrizes usadas na multiplicação e o resultado obtido. Por fim, libera-se a memória alocada para armazenar as matrizes.

Multiplicar Matrizes Quadradas Sequencialmente nas linhas (Paralelo For)

Com o uso da biblioteca **OpenMP** a versão paralela consiste em quase o mesmo código utilizado na versão serial, sendo adicionado um novo parâmetro para indicar a quantidade de threads a serem usadas e algumas diretrizes **#pragma** no laço responsável por percorrer as linhas da primeira matriz, ou seja, o laço que chama a função que calcula a linha correspondente da matriz resultante da multiplicação. Essas diretrizes foram utilizadas para estabelecer a região paralela, especificando o número de threads a serem usadas e quais variáveis serão acessíveis pelas threads nessa região. Dentro dessa região se encontra o laço alvo da paralelização, acima dele é utilizada a diretriz **for** do **OpenMP**, foi utilizado também um *schedule(guided)* para informar como se dá a distribuição de atividades das

threads. Fora isso, apenas a contagem do tempo foi alterada, agora utiliza uma função própria do **OpenMP** em vez do *clock* da biblioteca *time.h*.

Multiplicar Matrizes Quadradas Sequencialmente nas linhas (Paralelo Tasks)

A versão que utiliza **tasks** também consiste em quase o mesmo código utilizado na versão serial, sendo adicionado um novo parâmetro para indicar a quantidade de threads a serem utilizadas e algumas diretrizes ***#pragma*** no laço responsável por percorrer as linhas da primeira matriz. As diretrizes, assim como na versão com o **for**, foram utilizadas para estabelecer a região paralela, especificando o número de threads a serem usadas e quais variáveis serão acessíveis pelas threads nesta região. Dentro dessa região se encontra o laço alvo da paralelização, porém algumas diretrizes a mais foram utilizadas, Foi estabelecida uma região onde apenas uma thread pode executar, a região que irá criar as **tasks**. Nesse laço alvo, em vez de percorrer as linhas da primeira matriz e calcular o resultado de cada linha da matriz resultante, agora apenas cria uma **task** responsável por realizar esse cálculo. As outras threads, que não executam essa região definida anteriormente, assumem essas **tasks** e as executam.

Multiplicar Matrizes Quadradas Usando Transposição (Serial)

Na versão serial com transposição são feitas algumas modificações nas etapas descritas na versão serial anterior, são elas: a alocação de uma nova matriz para armazenar o resultado da transposição da segunda matriz; a transposição da segunda matriz antes de realizar a multiplicação; a multiplicação entre as matrizes agora ocorre entre as linhas da primeira matriz e as linhas da segunda matriz transposta; por fim, ocorre a liberação da memória alocada para armazenar a segunda matriz transposta.

A memória usada para armazenar a segunda matriz transposta é alocada após a alocação das outras matrizes. A transposição é realizada da seguinte maneira: para cada elemento e_{ij}, sendo i o índice da linha na matriz original e j o índice da coluna, o valor do elemento t_{ji} na matriz alocada para armazenar a transposta, será igual ao elemento e_{ij}. Assim, após percorrer todos os elementos da matriz original e os posicionar na matriz alocada para armazenar a transposição, inicia-se a multiplicação entre as matrizes. Para isso são percorridas as linhas da primeira matriz e é calculada a linha correspondente da matriz resultante da multiplicação. Ressalta-se que, em relação a versão serial anterior, a diferença ocorre no momento de calcular cada elemento, pois em vez de multiplicar os elementos da linha da primeira matriz com a coluna da segunda matriz, realiza-se a multiplicação entre os elementos da linha da primeira matriz com os elementos das linhas da segunda matriz, uma vez que, por ela ter sido transposta, as linhas dela correspondem às colunas da matriz original. Por conseguinte, o restante do algoritmo é o mesmo da versão serial apresentada inicialmente, com a adição da liberação da memória da matriz transposta junto da liberação das outras matrizes ao fim do código.

Multiplicar Matrizes Quadradas Usando Transposição (Paralelo For)

Assim como nas versões paralelas anteriores, é adicionado um parâmetro a mais para identificar quantas threads usar e então adicionadas as diretrizes **#pragma**, relativas a biblioteca **OpenMP**, no código serial descrito anteriormente, o resultado final é a aplicação dessas diretrizes em dois momentos, o primeiro momento é no laço responsável por realizar a transposição da segunda matriz, o segundo momento é na etapa de percorrer as linhas da primeira matriz e calcular a linha equivalente da matriz resultante da multiplicação. O mesmo passo a passo da versão paralela com **for**, descrita anteriormente, é realizado. Inicia-se a região paralela, define-se a quantidade de threads e se especifica quais as variáveis que estarão disponíveis nela, para então utilizar o **#pragma** com o **for** especificando também o *schedule(guided)*.

Multiplicar Matrizes Quadradas Usando Transposição (Paralelo Tasks)

Assim como na versão paralela com **for**, para a solução com transposição da segunda matriz, a versão com **tasks** adiciona um parâmetro para a definição da quantidade de threads e chama as diretrizes **#pragma** em dois momentos, nos mesmo dois momentos citados na versão com **for**, na transposição e no cálculo de fato da matriz resultante. O método utilizado é similar a como foram implementadas as **tasks** na outra solução. Inicia-se a região paralela, define-se uma região onde uma única thread cria todas as **tasks** relativa a tarefa paralelizada, enquanto as outras threads são alocadas as **tasks** criadas e as executam. Na transposição, a **task** criada consistia na transposição de uma linha da matriz para uma coluna. Enquanto que a **task** para o cálculo da multiplicação continua a mesma.

Resultados encontrados

Corretude dos algoritmos implementados (Todos)

A corretude dos algoritmos implementados pode ser verificada com a seguinte instância do problema, usando os seguintes argumentos:

- Nas versões seriais: **1 0 42 2 2 2 2**
- Nas versões paralelas: **2 1 0 42 2 2 2 2**

Obtém-se o seguinte resultado em todas as implementações feitas:

Primeira Matriz

0.840188	0.394383
0.783099	0.798440

Segunda Matriz

0.033470	0.329964
0.690636	0.422487

Matriz Resultante

0.300496	0.443853
0.577641	0.595725

Ao verificar as operações realizadas, é visto que o resultado difere apenas em alguns dígitos, isso é facilmente explicável, afinal os números exibidos na tela são apenas uma aproximação dos números que são de fato usados internamente, muitas vezes arredondando os valores, pois apenas 6 dígitos de precisão foram utilizados. Segue a verificação dessas operações:

$$e_{11} = 0.840188 \cdot 0.033470 + 0.394383 \cdot 0.690636 = 0.30049618994$$
$$e_{12} = 0.840188 \cdot 0.329964 + 0.394383 \cdot 0.422487 = 0.44385348375$$
$$e_{21} = 0.783099 \cdot 0.033470 + 0.798440 \cdot 0.690636 = 0.57764173137$$
$$e_{22} = 0.783099 \cdot 0.329964 + 0.798440 \cdot 0.422487 = 0.59572499871$$

Análise de Speedup, Eficiência e Escalabilidade

Abaixo estão as tabelas com as médias dos tempos válidos obtidos, sendo que para cada instância analisada, foram coletados 15 tempos, em que os dois maiores e os dois menores foram desconsiderados como válidos para o cálculo dessa média.

Multiplicar Matrizes Quadradas Sequencialmente nas Linhas

Em relação ao ganho de tempo com o uso da versão paralela pode-se perceber uma diminuição considerável no tempo gasto para o cálculo da multiplicação. Seguem os tempos entre a versão serial e a paralela usando **for** e em seguida os tempos da versão com **tasks**, seguindo a ordem das linhas e sem transposição:

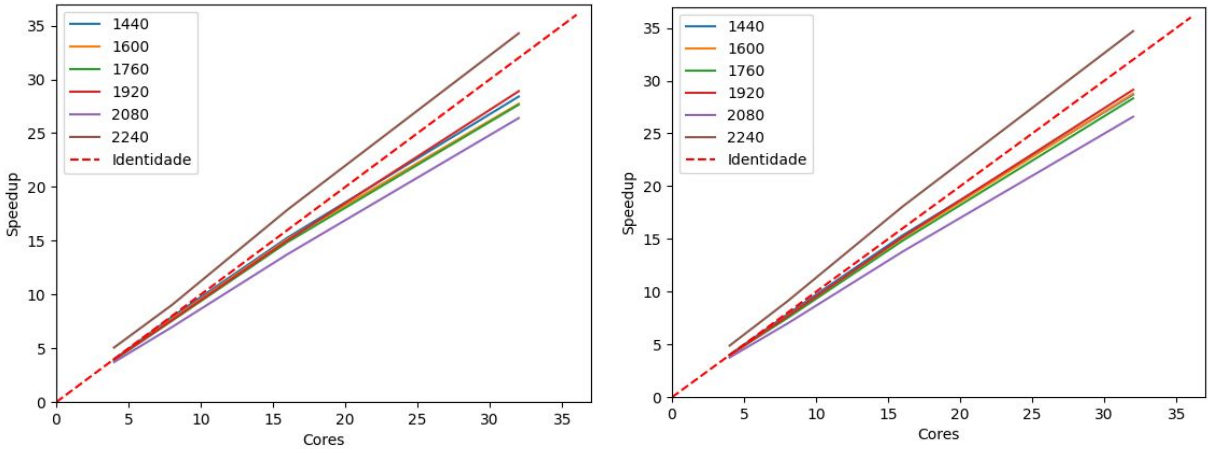
Tamanho do Problema	Serial	4	8	16	32
1440	31,790000000	8,013437736	4,014977721	2,079808993	1,119140705
1600	44,543636364	11,386977738	5,777305011	2,942084395	1,606229267
1760	60,504545455	15,487768344	8,010730976	4,068668341	2,188268454
1920	84,679090909	21,352408002	11,106709758	5,632314525	2,930185527
2080	105,297272727	28,326809137	15,109897887	7,659259366	3,988046816
2240	182,110000000	35,904686594	20,175568718	10,183283619	5,312673599

Tabela 1: Apresenta os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela com **for**.

Tamanho do Problema	Serial	4	8	16	32
1440	31,790000000	7,889139481	4,047388118	2,072723932	1,108246146
1600	44,543636364	11,385512592	5,793461270	2,954157227	1,548002690
1760	60,504545455	15,141819306	8,074808319	4,078648509	2,135917985
1920	84,679090909	21,243801316	11,049880579	5,578279214	2,905743259
2080	105,297272727	27,950202115	15,096981257	7,631317605	3,960528829
2240	182,110000000	37,229447440	20,049871461	10,077331620	5,248227747

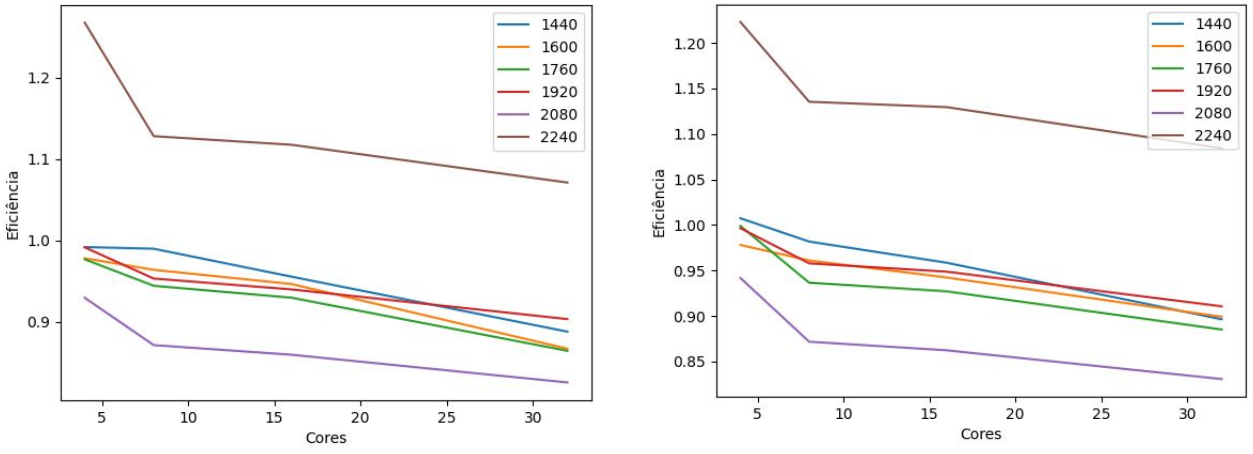
Tabela 2: Apresenta os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela com **tasks**.

Abaixo seguem as análises e comparações, de speedup e eficiência, das versões paralelas do algoritmo de multiplicação das matrizes percorrendo *sequencialmente* as linhas, sem utilizar transposição na segunda matriz, e seus devidos gráficos.

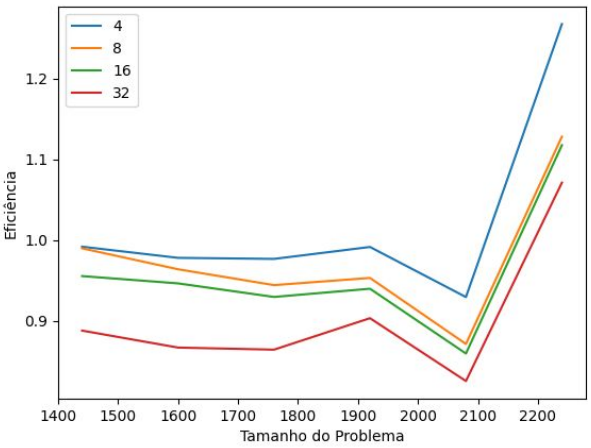


Figuras 1 e 2: Comparações do speedup em cada um dos tamanhos do problema com o speedup ideal. À direita temos a versão com **for** e à esquerda utilizando **tasks**. É interessante notar que para o tamanho do problema 2240 um comportamento não esperado ocorreu, ele se manteve com um speedup superlinear.

Esse comportamento de speedup superlinear não é o esperado, pode ter sido causado por algum detalhe do nó utilizado no supercomputador durante a execução do experimento. Novos testes precisam ser executados para validar esse comportamento. Seguem agora os gráficos sobre a eficiência.



Figuras 3 e 4: Apresentam a eficiência por cores em cada tamanho de problema utilizado, à direita temos a versão com **for** e à esquerda utilizando **tasks**. Pode-se observar um comportamento comum, a diminuição da eficiência com o aumento do número de cores usados. Mesmo que no último tamanho do problema a eficiência inicie bem mais alta.



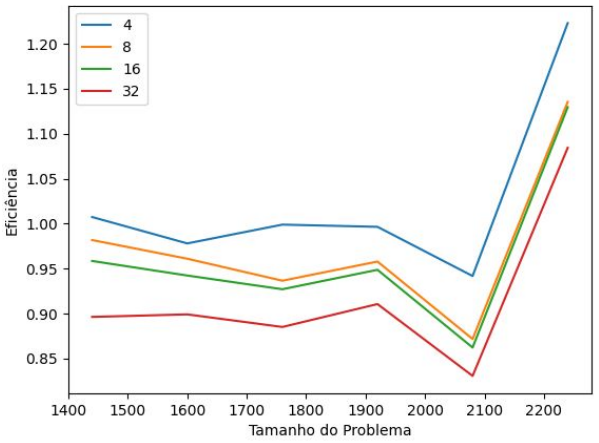


Figura 5 e 6: Apresentam a eficiência por tamanho de problema em cada número de cores utilizados, à direita temos a versão com **for** e à esquerda utilizando **tasks.**

Diante desses dados, é concluído que a versão da multiplicação sequencialmente nas linhas se categoriza como **Fracamente Escalável**, pois aumentando o tamanho do problema e o número de cores em proporções iguais, a eficiência se comporta de forma crescente, com algumas flutuações.

Multiplicar Matrizes Quadradas Usando Transposição

Ao realizar a *transposição* da segunda matriz e adaptando a lógica da multiplicação para linhas com linhas, obtém-se uma diminuição considerável no tempo gasto para realizar a multiplicação das matrizes, pois o princípio da localidade é melhor explorado, visto que durante a multiplicação são acessados apenas elementos que estão próximos uns dos outros e, como uma matriz é armazenada de forma contígua na memória, acessar elementos em linhas diferentes, mas numa mesma coluna, é mais custoso, pois estão mais distantes do que o elemento na mesma linha, porém em outra coluna. Isso se dá pela forma como a hierarquia de memória é estruturada. Essa diminuição no tempo gasto é visível nos tempos exibidos nas tabelas abaixo, tanto na versão com **for** quanto na com **tasks**.

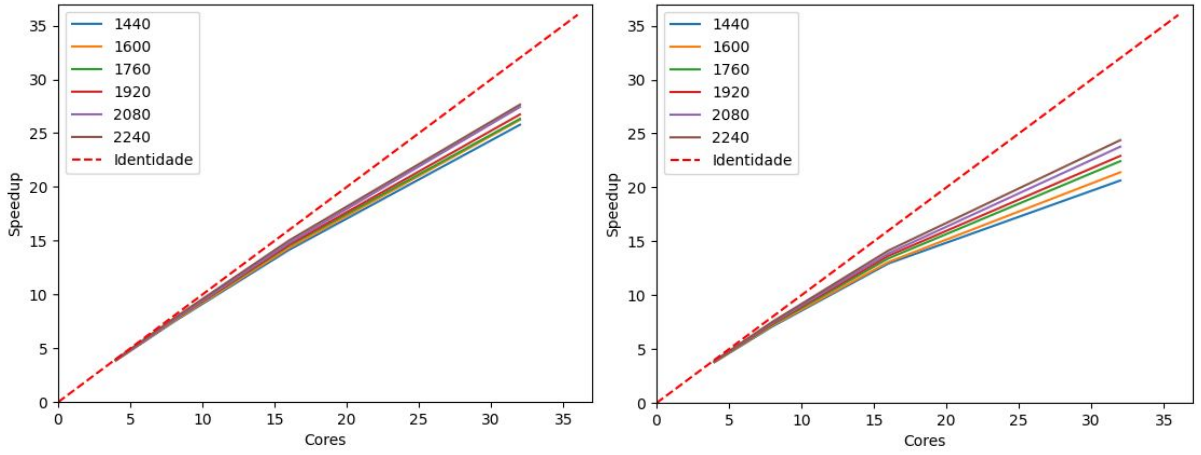
Tamanho do Problema	Serial Transpose	4	8	16	32
1440	13,503636364	3,492781354	1,812161452	0,954283027	0,523760734
1600	18,494545455	4,763939761	2,464167802	1,290540301	0,704999521
1760	24,598181818	6,303981911	3,237017994	1,692033471	0,934236815
1920	31,928181818	8,185955313	4,190019298	2,182293925	1,194173802
2080	40,627272727	10,392114592	5,317946080	2,745897430	1,481160333
2240	51,317272727	12,956323040	6,602584971	3,416099039	1,855932436

Tabela 3: Apresenta os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela com **for**.

Tamanho do Problema	Serial Transpose	4	8	16	32
1440	13,503636364	3,556507900	1,900344074	1,043112371	0,654169624
1600	18,494545455	4,868305291	2,571626407	1,414003895	0,864062189
1760	24,598181818	6,435845426	3,372886156	1,833511077	1,096341432
1920	31,928181818	8,313620570	4,335806452	2,337496562	1,392680599
2080	40,627272727	10,544062671	5,460174870	2,927845519	1,708430670
2240	51,317272727	13,136950012	6,807697531	3,628274103	2,104416786

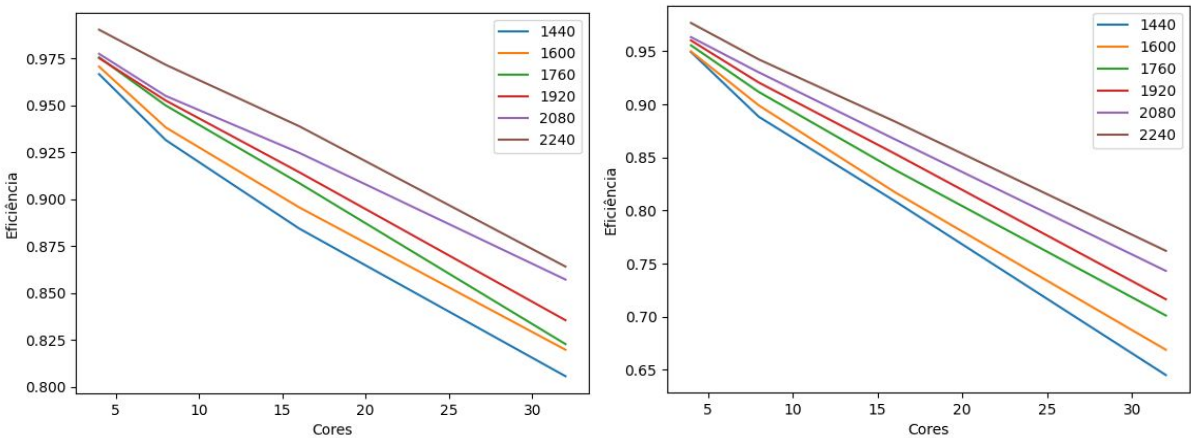
Tabela 4: Apresenta os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela com **tasks**.

Abaixo seguem as análises e comparações, de speedup e eficiência, das versões paralelas do algoritmo de multiplicação das matrizes utilizando transposição na segunda matriz, e seus devidos gráficos.

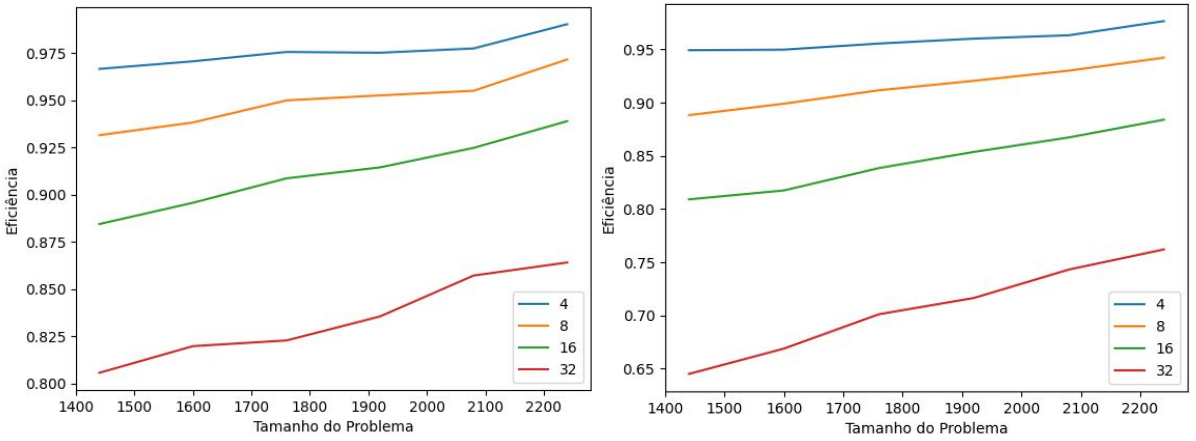


Figuras 7 e 8: Comparações do speedup em cada um dos tamanhos do problema com o speedup ideal. À direita temos a versão com **for** e à esquerda utilizando **tasks**. Vale destacar o desempenho da versão com **for** em relação à versão com **tasks**, pois o speedup do **for** acabou sendo melhor.

O speedup nessa versão se comportou como o esperado, se mantendo abaixo do speedup ideal e curiosamente os valores mantiveram uma consistência bem semelhante no aumento conforme o número de cores, mesmo com tamanho de problema diferentes, sendo que na versão com **tasks** pode-se observar o impacto do tamanho do problema mais facilmente, pois a discrepância entre os speedups deles é mais perceptível visualmente. Seguem agora os gráficos sobre a eficiência.



Figuras 9 e 10: Apresentam a eficiência por cores em cada tamanho de problema utilizado, assim como no gráfico equivalente do algoritmo anterior, o comportamento obtido é o esperado, diminuição da eficiência com o aumento dos cores em ambas as versões com **for** e a com **tasks**. Vale destacar a escala em cada um deles, o uso de **tasks** (direita) tem um impacto forte na eficiência obtida.



Figuras 11 e 12: Apresentam a eficiência por tamanho de problema em cada número de cores utilizados, o comportamento geral é bem semelhante entre os gráficos, percebe-se facilmente que com o aumento do tamanho do problema a eficiência aumenta, porém no gráfico à direita, representando a abordagem com **tasks**, ao observar a escala, percebe-se que os valores são menores, mostrando o impacto que o uso das mesmas causa da eficiência.

Dadas essas figuras, pode-se concluir que mesmo o uso das **tasks**, mesmo sendo menos adequado para a tarefa em questão, ainda compartilha das mesmas características da versão com **for**, que se mostrou bem mais eficiente na atividade de multiplicar as matrizes, quando é realizada a transposição da segunda matriz previamente.

Assim, analisando esses gráficos, mesmo que a versão com **tasks** tenha um desempenho pior do que o uso do **for**, ambas apresentam características de serem **Fracamente Escaláveis**, visto que com o aumento do tamanho do problema e do número de cores em proporções iguais, a eficiência se mantém crescente.

Considerações Finais

Neste relatório apresentou-se como é definida a multiplicação de duas matrizes, quais as condições para a execução dessa operação, como essa operação é feita e como ela se aplica para matrizes quadradas. Foram explanadas duas versões seriais e quatro versões paralelas para a multiplicação de matrizes quadradas, sendo que a primeira é a forma comum, multiplicando linhas e colunas e a outra versão serial realiza a *transposição* da segunda matriz antes de realizar a multiplicação linha a linha. A primeira versão paralela implementada tem como base a primeira versão serial e utiliza uma abordagem usando o **for** do **OpenMP**, a segunda versão paralela também tem como base a primeira versão serial, porém utiliza uma abordagem voltada para as **tasks**. O mesmo ocorre com a segunda versão serial, a terceira versão paralela utiliza **for** junto da transposição e a última versão paralela faz o mesmo numa abordagem com **tasks**. Em seguida, foi realizada uma breve apresentação da corretude dos algoritmos implementados, exemplificando a execução dos mesmos por meio de uma instância de execução dos algoritmos implementados e depois verificando o resultado realizando os cálculos manualmente, constatando que o resultado é correto, relevando o erro por aproximação dos números de ponto flutuante. Por conseguinte, a análise e comparação do speedup e eficiências dos algoritmos paralelos implementados foi feita, expondo os tempos médios obtidos nos experimentos realizados e discutindo os gráficos de speedup e eficiência gerados por esses tempos. É facilmente perceptível o ganho no tempo gasto na versão com transposição em relação a versão sem transposição, mesmo que as métricas obtidas sobre speedup e escalabilidade sejam menores na versão com transposição. Por fim, foi realizada a categorização das versões paralelas em relação à escalabilidade delas, sendo ambas as versões **Fracamente Escaláveis**.

Referências

Introdução a Sistemas Paralelos

Produto de matrizes

Matrix multiplication algorithm

How can I shuffle the contents of an array?