Programação sequencial, paralela e distribuída : O problema do Quadrado Mágico

Artur C. Rodrigues

Departamento de Engenharia Informática, Universidade do Algarve
Unidade curricular de Sistemas Paralelos e Distribuídos

Dra. Margarida Madeira
13 março, 2020

Resumo

Com a rápida evolução tecnológica que se verifica no tempos de hoje, existe uma constante tendência de procurar melhorar ao máximo a capacidade computacional existente. Tal costumava ser feito recorrendo ao aumento consistente da frequência do relógio no *CPU*, tratando-se de um fatores mais importantes no processo de desenvolvimento de unidades centrais de processamento mais rápidas e poderosas.

Este paradigma veio, no entanto, a terminar pois o constante aumento das frequências dos CPU's provoca um enorme problema ao nível da dissipação de energia, energia que é dissipada na forma de calor devido à resistência dos circuitos elétricos integrados, criando graves problemas de sobre-aquecimento [1].

Como forma de ultrapassar esta barreira, os principais produtores de CPU's começaram então a investir no desenho de *chips multi-core*, obrigando o software a ser escrito através de abordagens *multi-threaded* ou *multi-process* no sentido de tirar vantagem destas novas arquiteturas, permitindo pois a execução paralela de tarefas e processos [2].

Este relatório investiga qual o impacto a nível de performance e eficiência de uma abordagem paralelizada e/ou distribuída na implementação de software *versus* uma abordagem estritamente sequencial, recorrendo para tal a típicos paradigmas da programação paralela e distibuida : *PThreads*, *OpenMP* , *MPI* e um modelo híbrido com *OpenMP* e *MPI*.

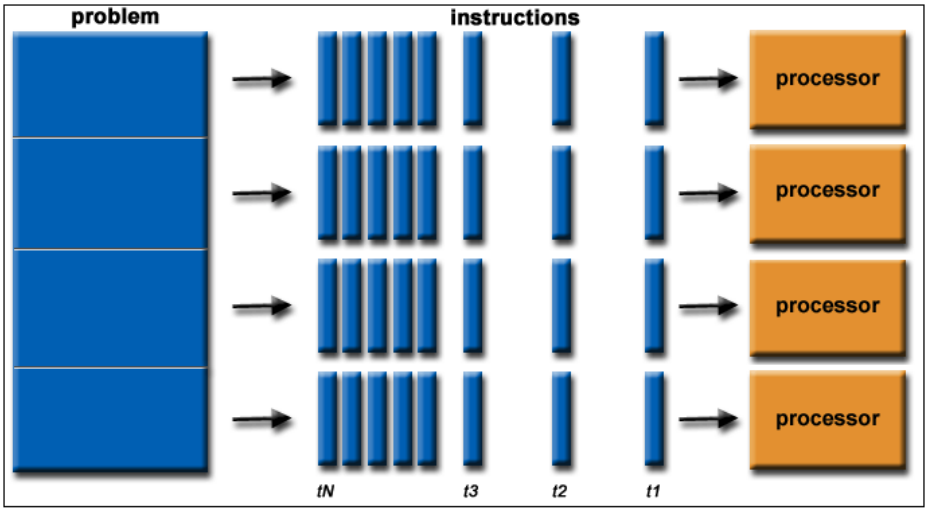


Image 1: Parallel computing

Abstract

With the rapid technological evolution nowadays, there's a constant tendency to keep improving the current computing capacity to the maximum. That used to be achieved through a consistent increase in the clock speed of the CPU, method that was considered to be one of the main factors in the development of faster and more powerful central processing units.

This paradigm came, however, to an end due to physical constraints. As the frequency of the clock in the CPU increases so does the power consumption, which consequently produces heat, generated by the resistors of the integrated electric circuits, creating over-heating issues that were no longer worth addressing, as such would not outweight the cons [1].

As a way to overcome this barrier, the main manufacturers of CPU'S started investing in the production of multi-core chips, forcing the software to be written through multi-threaded or multi-process approaches in order to take full advantage of these new architectures, allowing the parallel execution of processes and threads[2].

This paper investigates the impact in terms of performance and efficiency of a parallel computing approach in software development *versus* a strictly sequential approach, utilizing typical paradigms of parallel computing for such : *Pthreads*, *OpenMP*, *MPI* and an hybrid approach with *OpenMP* and *MPI*.

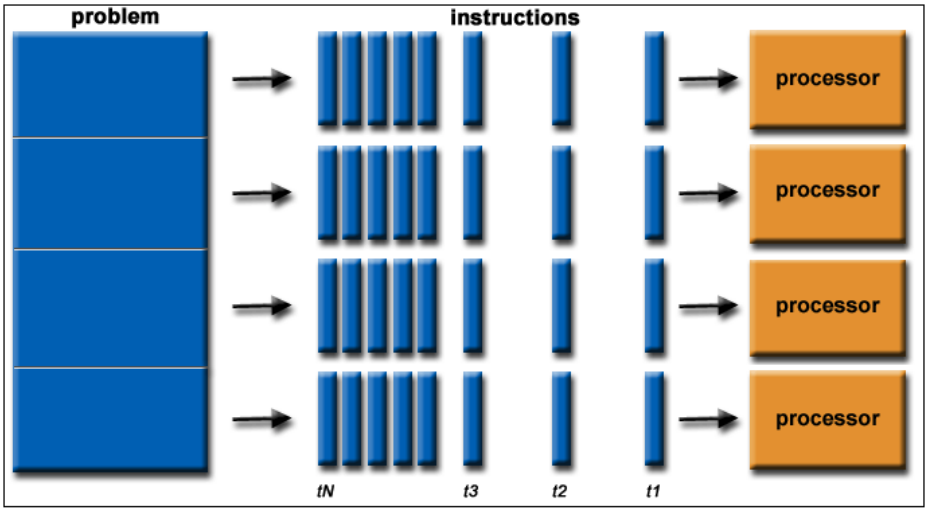


Image 1: Parallel computing

Tabela de conteúdos

1 - Introdução

O processamento paralelo e distribuído traduz-se na realização de uma programação concorrente, que permite a ocorrência simultânea de eventos num sistema computacional, recorrendo-se a mais de um elemento de processamento. Num sistema com hardware com capacidade multi-tarefa ou multi-processo, pressupõe-se então à partida que um software desenvolvido segundo abordagens de programação paralela terá um desempenho significativamente superior a um desenvolvido através de uma abordagem estritamente sequencial, especialmente à medida que a dimensão dos casos de uso aumentam.

Com o objetivo de quantificar qual o impacto das abordagens de paralelização na implementação de software *versus* abordagens sequenciais, este relatório retrata um estudo da avaliação de desempenho de uma abordagem estritamente sequencial, de duas diferentes abordagens de computação paralela, de uma abordagem de computação distribuída e por fim de uma abordagem híbrida, que integra um abordagem paralela e distribuída, sendo todas estas descritas com maior detalhe na sua respetiva secção.

Para tal, recorreu-se à linguagem de programação C, tendo os programas sido modelados para resolver o problema do quadrado mágico : uma problema clássico na matemática e suscetível de ser resolvido através de computação paralelizada de forma relativamente simples, uma vez que paralelizar o seu algoritmo de resolução não é uma tarefa complexa.

O processo de avaliação do desempenho dos programas, exceto do *MPI* e do modelo híbrido, fez-se numa só máquina, um computador com uma arquitetura de 64 *bits*, possuindo um *CPU* intel i7 8750-H com 6 cores e 12 *threads*, 8 *Gibabytes* de *RAM* e um *SSD* como hardware de armazenamento de dados. Esta máquina tem como sistema operativo o Linux Debian Buster 10. As duas exceções foram testadas numa outra máquina, que corre um sistema operativo *Debian Buster* 10 com arquitetura de 64 *bits*. Esta máquina é composta por um processador *Intel Xeon Gold 6138 2.00GHz*, que possui 20 *cores* e 40 *threads*. Possui ainda 8 *GB* de *RAM*. As suas unidades de processamento estão divididas em 4 nós, cada nó composto por 4 cores. Para determinar os requisitos dos sistemas, alguns deles necessários para o cálculo dos indicadores de desempenho, recorreram-se aos seguintes comandos:

`lsb_release -a` `uname -a` `cat /proc/cpuinfo` `cat /proc/meminfo`

Para compilação dos programas, o compilador utilizado foi o *gcc*, GNU compiler collection versão 8.3.0, não tendo sido utilizadas quaisquer opções de otimização oferecidas pelo mesmo. O comando utilizado foi o seguinte :

```
gcc program_name.c -o program_name -lpthread -myints.h
```

O processo de desenvolvimento dos programas iniciou-se pela a abordagem estritamente sequencial, onde existe apenas um processo de execução e a utilização um único core, não tendo sido utilizado qualquer tipo de ferramenta de paralelismo ou concorrência. Posteriormente foi desenvolvido o programa com recurso a *PThreads*, onde foram utilizadas 12 threads e existe uma execução paralela do programa. Continua portanto a existir apenas uma única execução do programa mas agora com a utilização de 12 cores.

De seguida seguiu-se a abordagem com *OpenMP*, uma *API* que permite especificar a alto nível uma execução paralela do programa com base em múltiplas threads e memória partilhada. Tal como com as *PThreads*, existe apenas uma única execução do programa com recurso a 12 cores.

Em penúltimo lugar, desenvolveu-se um programa com base numa abordagem *MPI*, um padrão utilizado para comunicação de dados em computação distribuída e com diversas implementações, sendo frequentemente utilizada em *clusters*, tendo a implementação utilizada sido a *MPICH*. Foram portanto utilizados quatro processos do mesmo programa , cada um destes a recorrer a apenas um core.

Por fim, foi desenvolvido um programa com base numa abordagem híbrida, isto é, com base tanto no *OpenMP* como no *MPI*. Recorreram-se a quatro processos na totalidade, sendo que cada processo fez uso de 4 *threads* de execução, ou seja 4 cores.

Após o término da implementação de cada uma das abordagens, foram feitas as avaliações de desempenho respetivas, tendo sido testadas para cada abordagem 9 matrizes com as seguintes dimensões : 501, 1001, 2001, 3001, 5001, 7501, 10001,15001 e finalmente 20001. Cada uma das matrizes possui um versão mágica perfeita, mágica imperfeita e não mágica. Para cada uma das versões, as medições foram feitas com base num bateria de 31 testes, desprezando-se a primeira por o sistema operativo ter de colocar o programa em memória, o que provoca um aumento do tempo de execução no primeiro teste e não tem significado para o problema atual.

Durante a realização das baterias de testes, não existiam aplicações terceiras a correr e a ligação à *internet* encontrava-se desligada, para diminuir ao máximo fatores externos que possam afetar o desempenho dos programas. Para obter os valores que depois serão utilizados na construção dos gráficos de análise, recorreu-se ao cálculo da média e variâncias das amostras, no sentido de diminuir ao máximo o erro existente nos resultados, uma vez que podem existir interferências de processos terceiros e do sistema operativo no scheduling do *CPU*, afetando os desempenhos dos programas. Os tempos de execução foram recolhidos através da utilização do comando *bash time* para qualquer uma das abordagens assim como à utilização do método *gettimeoftheday*, existente na biblioteca *time.h* nos sistemas *Linux*.

É importante referir que todos os quadrados utilizados nos testes foram gerados através de um programa em *Python*, que gera quadrados dos três tipos mas apenas com dimensões ímpares, daí que todos os quadrados testados assim o sejam. Os nomes dos ficheiros dos respetivos quadrados estão nomeados de acordo com a seguinte convenção : .txt em que:

r : p, i, n para perfeito, imperfeito, não é quadrado mágico;

n : dimensão da matriz.

Esta convenção permitiu alocar exatamente a memória necessária para o vetor que guardará o *input*, tornando mais eficiente o programa.

As principais conclusões que se podem retirar deste relatório são que a implementação com *Pthreads* é, sem margem de dúvidas, a implementação que para todos os tipos de quadrados e dimensões consegue obter o melhores resultados. A abordagem sequencial demonstra-se muito otimizada, com o *openMP* a mostrar-se geralmente mais lento que o sequencial.

As abordagens distribuídas, nomeadamente a abordagem *MPI* e *híbrida*, apresentam resultados muito abaixo do expectável, com o *MPI* apenas a desempenhar-se muito bem em quadrados perfeitos de dimensões elevadas. Já o *híbrido* apresenta em todos os testes e quadrados resultados muito baixos, não conseguindo competir de maneira nenhuma com as outras abordagens.

O relatório encontra-se portanto organizado da seguinte forma : na secção **2** são apresentados conceitos fundamentais à leitura deste relatório e que permitirão a sua melhor compreensão, na secção **3** é ilustrado o problema do quadrado mágico, as diferentes abordagens utilizadas, descritas em maior pormenor, e os seus respetivos resultados de desempenho. Por fim, na secção **4** é feita uma discussão dos resultados das avaliações de desempenho e comparação das métricas de desempenho e por fim, na secção **5** é feita a conclusão e observações finais sobre os resultados obtidos.

2 - Enquadramento

Entende-se por sistema paralelo um sistema que possibilita o processamento paralelo, isto é, a realização de programação concorrente num sistema com mais de um elemento de processamento. Esta programação concorrente não é mais que um modelo de programação no qual é possível ocorrer vários eventos em simultâneo, o que pode incorrer em diversos problemas como condições de corridas e inter-bloqueios, assuntos que não serão abordados neste relatório.

Os "elementos de processamento" acima referidos podem tomar variadas formas, desde unidades de processamento centrais (*CPU's*), os próprios núcleos de um *CPU* ou então elementos de processamento dedicados, como por exemplo as unidades gráficas dedicadas (*GPU's*).

O *CPU* não é mais que um circuito eletrónico responsável por concretizar as instruções de um determinado programa , através de operações aritméticas, lógicas,de controlo e de entrada/saída de dados. Este é composto por *cores*, que são a unidade mais básica de processamento do *CPU*, e são capazes de executar uma instrução de cada vez, estando a sua velocidade relacionado com a frequência do relógio que possuem[3]. O número de *threads* que um CPU suporta está diretamente relacionado com o número de cores que este possui, sendo que atualmente este número corresponde ao dobro dos cores que possui. Isto deve-se à grande maioria dos *CPU's* hoje em dia implementarem *hyper-threading*, em que o único *core* é convertido em dois *cores* virtuais.

Abordagens como a utilização de *PThreads* ou do *API Open-MP*, fazem parte de um modelo de programação paralela que é designado por programação multi-tarefa, que implementa programação concorrente, onde um programa (designado processo internamente) executa vários subprogramas (designados tarefas) paralelamente e concorrentemente. As tarefas possuem tanto uma memória local privada como uma memória global que é partilhada por todas as outras tarefas geradas pelo mesmo processo. Isto facilita a comunicação entre as diferentes tarefas e permite que todas trabalhem no mesmo conjunto de dados.[5]

Por outro lado, abordagens como a utilização do *MPI* fazem parte de um modelo de programação paralela designado por um modelo de memória distribuída, onde um programa é corrido concorrentemente por um sistema computacional sem memória partilhada. Não existe portanto qualquer memória global partilhada pelo processos nas diferentes máquinas, sendo a comunicação entre os processos feita através da troca de mensagens pela rede[6]. A informação é portanto movida do espaço de endereço de um dos processo para o de outro processo, através de operações de comunicação coletiva em cada processo. São especificações como o *MPI* que definem como é que esta comunicação é, de facto, feita.

Em relação à avaliação de desempenho, esta teve em conta quatro critérios, sendo estes[4] :

- O **tempo de execução**, que contabiliza o tempo total de execução do programa, isto é desde o momento em que o processo é despoletado pelo programa até ao momento em que o processo é terminado pelo sistema operativo, finalizando-se assim a execução do programa.
- A **aceleração**, definida como o ganho obtido no desempenho de uma determinada paralelização face à melhor implementação sequencial conhecida.

  Esta métrica é portanto calculada através da seguinte fórmula matemática :

  $$S = \frac{t_1}{t_p}$$

  onde por **t1** se entende o tempo de execução da melhor implementação sequencial conhecida e por **tp** o tempo de execução para a abordagem paralelizada.
- A **eficiência**, que é a fração de tempo em que os processadores estão efetivamente a realizar trabalho útil.

  Esta métrica é calculada através da seguinte fórmula matemática:

  $$E = \frac{S_1}{p}$$

  onde por **S1** se entende a aceleração de uma determinada abordagem paralelizada e por **p** o número de processadores utilizados.
- O **gradiente de execução**, que mede como é que o tempo de execução cresce de acordo com o aumento da dimensão do problema.

  Esta métrica é calculada através da seguinte fórmula matemática:

  $$G = \frac{t_1}{n^2}$$

  onde por **t1** se subentende o tempo de execução da abordagem e por **n** a dimensão do caso testado pela abordagem.

3 - Estudo de casos

3.1 - O problema do quadrado mágico

Os quadrados mágicos, que se pensa terem a sua origem por volta dos anos 2200 BC na China, são um tipo de quadrado de grande interesse para o ramo de matemática, sendo estudados ao nível da sua construção, enumeração e classificação.[7]

Estes quadrados distinguem-se de todos os outros pela sua característica única : a soma de todos os números de cada linha, coluna e de ambas as diagonais é a mesma, dai serem chamados de quadrados mágicos. Ao valor da sua soma deu-se, por convenção, o nome de *constante mágica*. Dentro desta categoria existe mais um tipo de quadrado mágico, denominado quadrado mágico imperfeito, onde as somas de todas as linhas e colunas são iguais à exceção de pelo menos uma das diagonais. A todo o restantes quadrados dá-se o nome de quadrados não mágicos.

Tratando-se de um problema de simples implementação e o seu algoritmo de resolução facilmente paralelizado, é este o problema que todas as abordagens resolvem : identificar se um dado quadrado é ou não mágico.

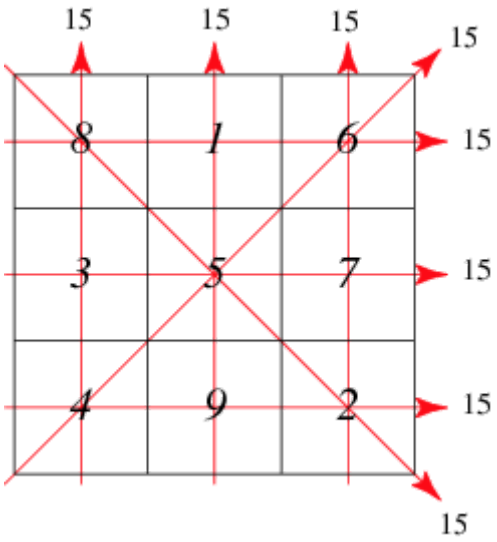


Image 2: Perfect magic square

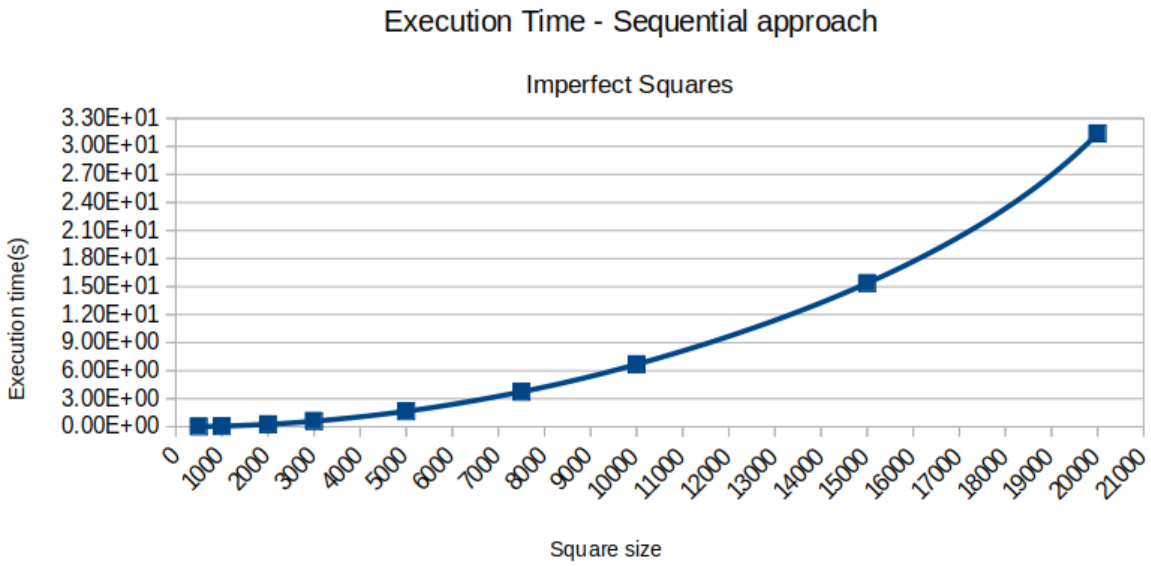
3.2 - Abordagem sequencial

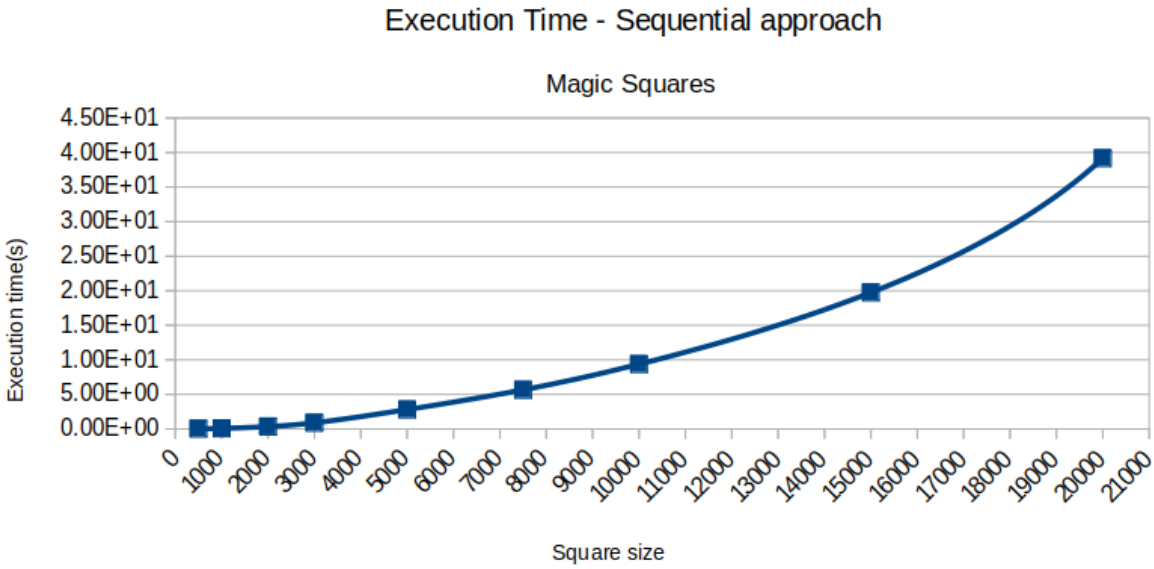
Esta secção detalha a implementação de um programa que identifica quadrado mágicos através de uma abordagem estritamente sequencial.

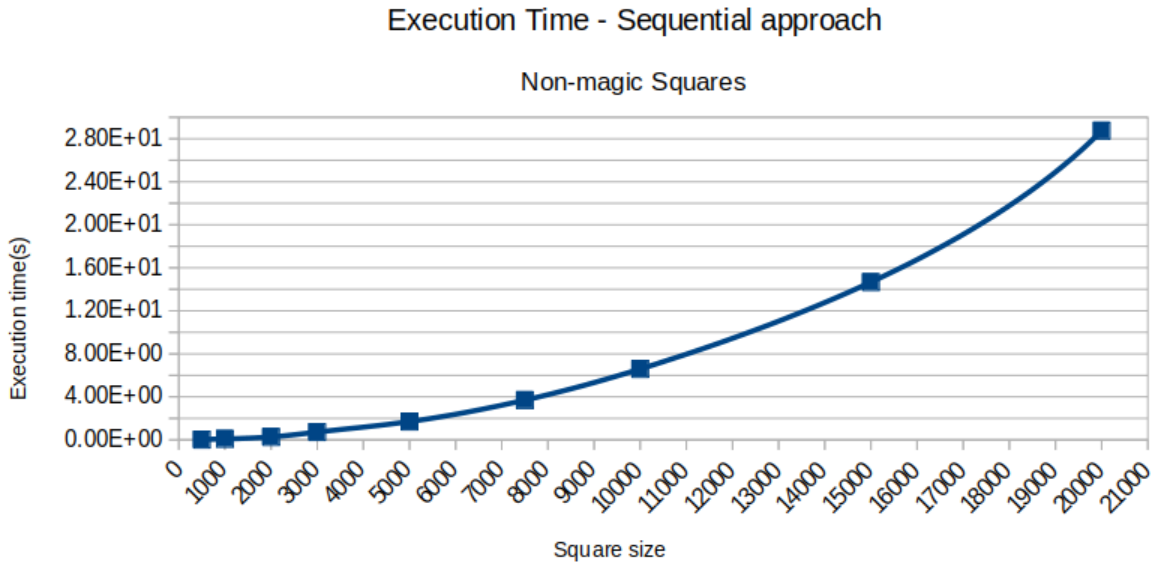
A estratégia utilizada nesta abordagem foi a seguinte : ler através de um ficheiro de texto os elementos do quadrado mágico, colocá-los em memória numa matriz bidimensional, calcular o valor da soma de todos os elementos da primeira linha e definir esse valor como a constante mágica.

Depois, avaliar todas as restantes linhas e verificar se a sua soma corresponde ao valor mágico. Apenas se tal acontecer é que se seguem as colunas e, verificando-se novamente o mesmo, passasse às diagonais. No caso de os três testes passarem com sucesso, o quadrado é mágico. Se apenas falhar alguma das diagonais, trata-se quadrado mágico imperfeito. Se falhar alguma das colunas ou linhas, é um quadrado não mágico.

Abaixo seguem-se os resultados dos tempos de execução correspondentes a execução do programa com quadrados de diversos tamanhos:







3.3 - Abordagem Pthreads

Esta secção detalha a implementação de um programa que identifica quadrado mágicos através de uma abordagem paralela, com recurso a *Pthreads*.

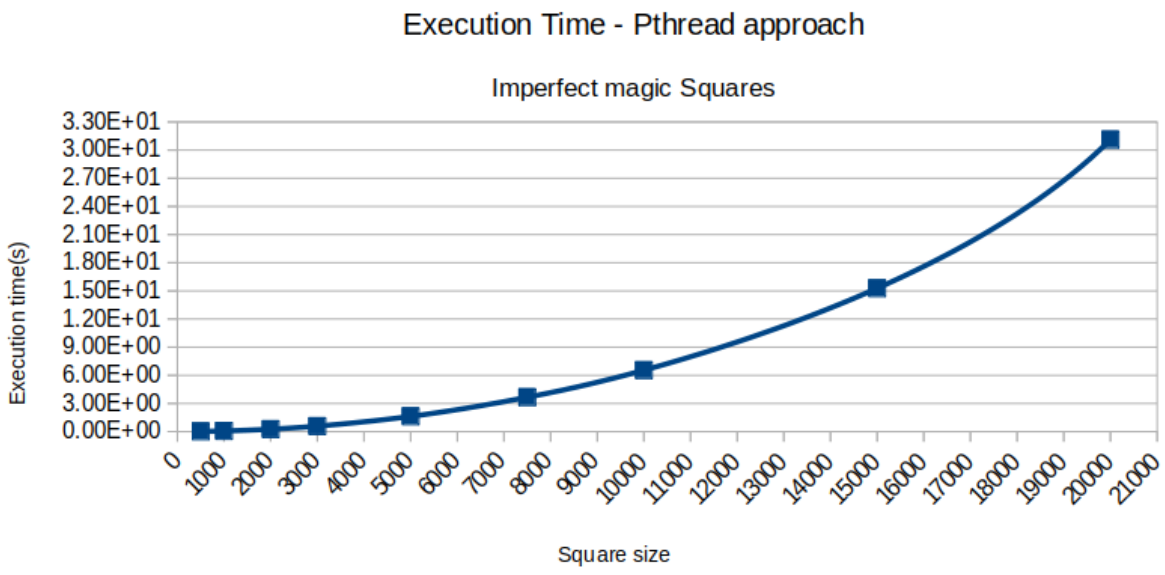
Antes de abordarmos o algoritmo utilizado, é importante esclarecer primeiramente o que são as *Pthreads*: As *Pthreads*, também chamadas de *POSIX* threads, são *threads* que obdecem a um padrão de definição denominado *POSIX.1c, Threads extensions* (IEEE Std 1003.1c-1995).[8] Para as utilizar recorre-se ao *POSIX threads API*, uma *API* de baixo nível que permite instanciar uma nova linha de execução concorrente, ou seja uma *thread*, no mesmo processo.

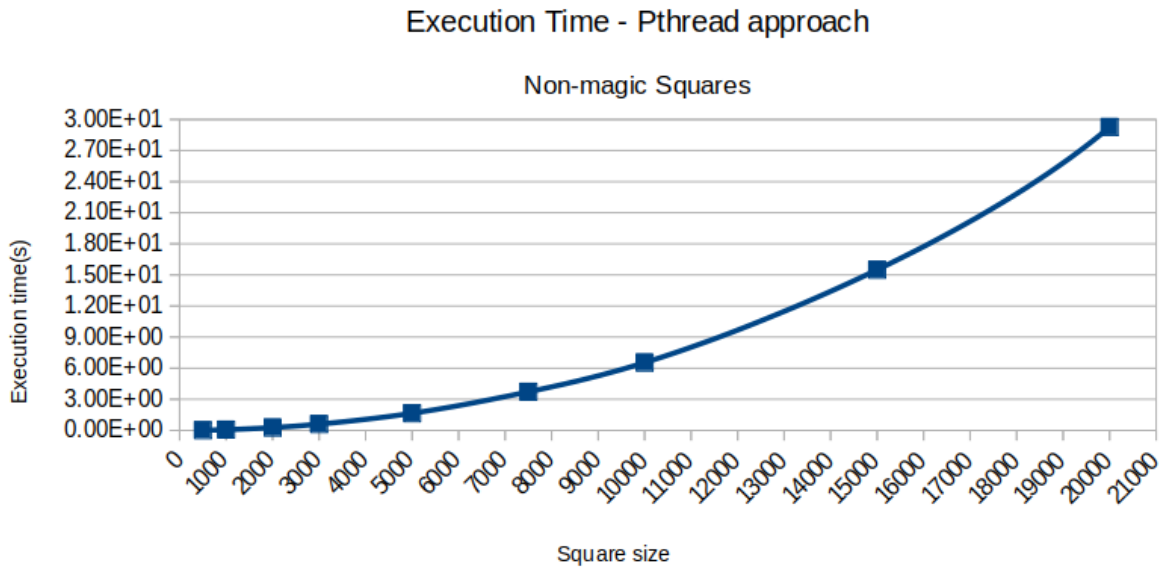
A estratégia utilizada nesta abordagem foi a seguinte : ler através de um ficheiro de texto os elementos do quadrado mágico, colocá-los em memória numa matriz bidimensional, calcular o valor da soma de todos os elementos da primeira linha e definir esse valor como a constante mágica. Existem três variáveis globais, todas elas inicializadas com o valor 1, que representam que por enquanto o quadrado tem todas as linhas, colunas e diagonais mágicas.

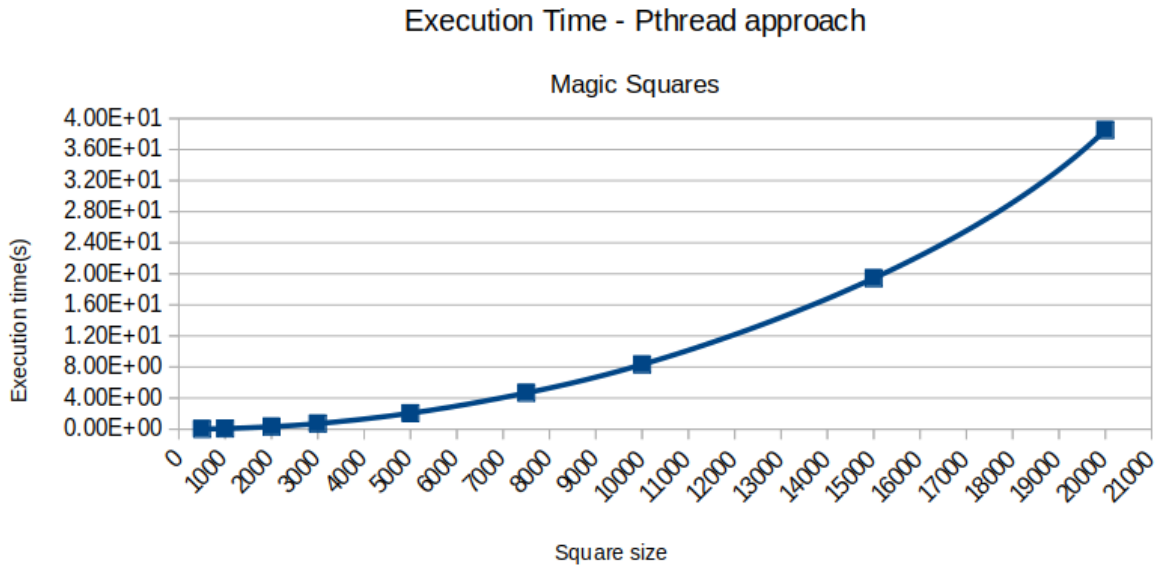
De seguida inicializam-se 12 *threads*, número máximo de threads úteis suportadas pelo processador da máquina de teste, tendo sido feita uma divisão de 6 *threads* para análise das linhas e 6 *threads* para a análise das colunas. Cada uma das threads avalia então uma porção das colunas ou linhas, sendo que se uma delas encontrar uma linha ou coluna imperfeita será alterado o valor da variável global correspondente para 0, indicando a todas as outras threads que devem parar a análise. No caso de tal não acontecer e as variáveis globais continuarem com o valor 1, a *thread* principal analisa as diagonais, sendo que só se for encontrada uma diagonal não mágica é que o valor da variável global correspondente é alterado para 0. Como esta análise consiste numa única iteração não é feita a sua paralelização.

É depois feita então a análise de acordo com os valores das variáveis globais, sendo que se todas estiverem a 1, o quadrado é mágico, se apenas a das diagonais é 0 então o quadrado é imperfeito e para os restantes casos trata-se de uma quadrado não mágico.

Abaixo seguem-se os resultados dos tempos de execução correspondentes a execução do programa com quadrados de diversos tamanhos:







3.4 - Abordagem OpenMP

Esta secção detalha a implementação de um programa que identifica quadrado mágicos através de uma abordagem paralela através da utilização da *API OpenMP*.

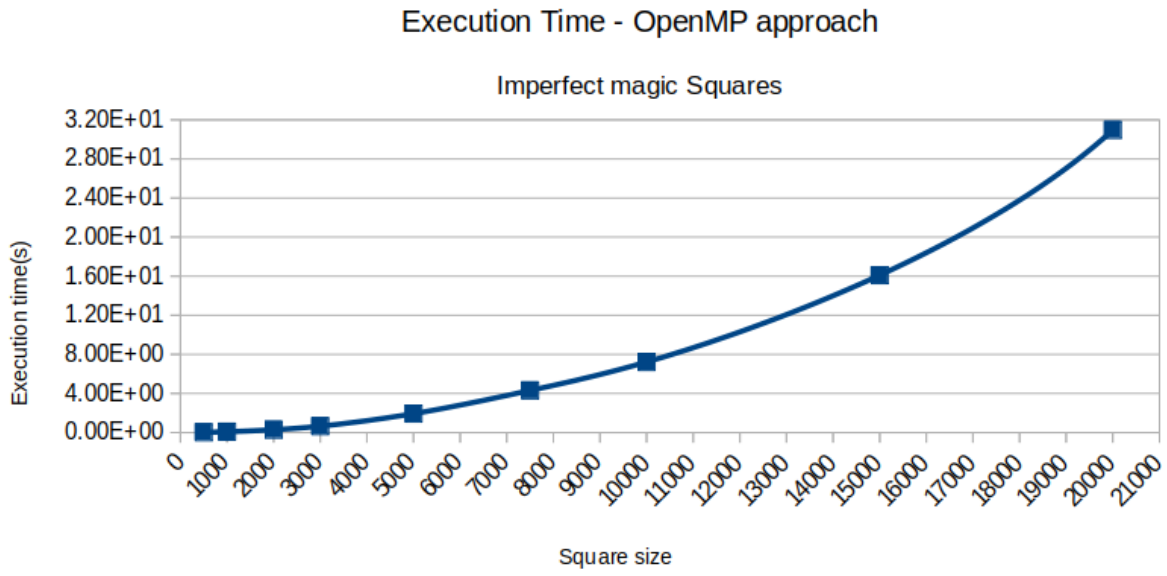
O *OpenMP* é uma *API* de alto nível, associada ao compilador, que permite paralelizar o código desenvolvido de forma muito simplificada, passando o mesmo a utilizar *threads*. Torna portanto a programação muito mais fácil uma vez que é possível paralelizar código através de simples *scoping* e também recorrer a inúmeros *thread-safe* métodos úteis.[9]

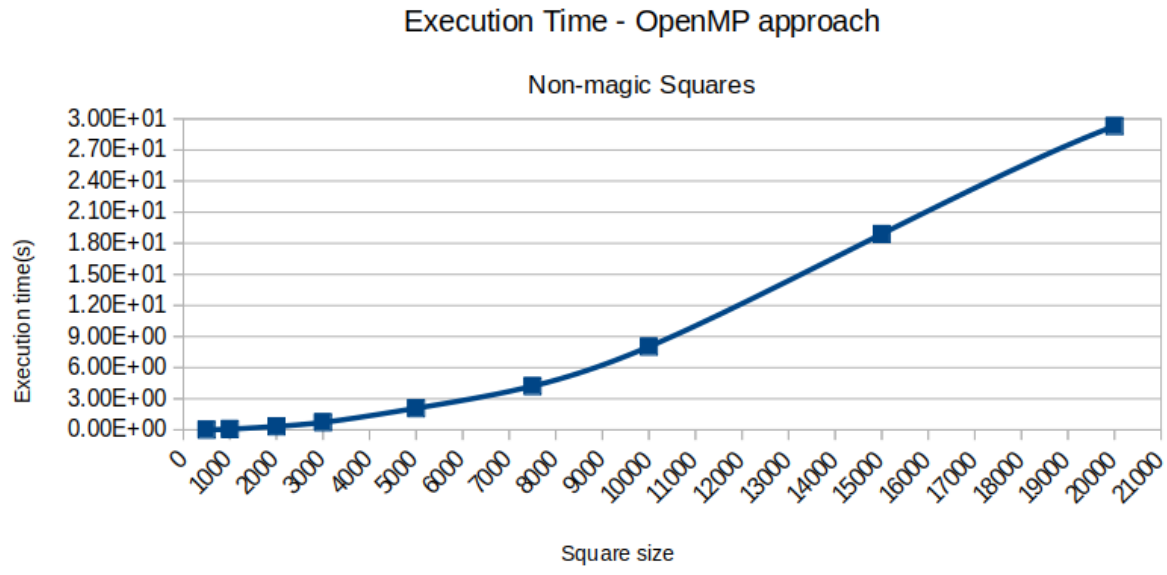
A estratégia utilizada nesta abordagem foi praticamente igual à utilizada pelas *Pthreads* : ler através de um ficheiro de texto os elementos do quadrado mágico, colocá-los em memória numa matriz bidimensional, calcular o valor da soma de todos os elementos da primeira linha e definir esse valor como a constante mágica. Existem três variáveis globais, todas elas inicializadas com o valor 1, que representam que por enquanto o quadrado tem todas as linhas, colunas e diagonais mágicas.

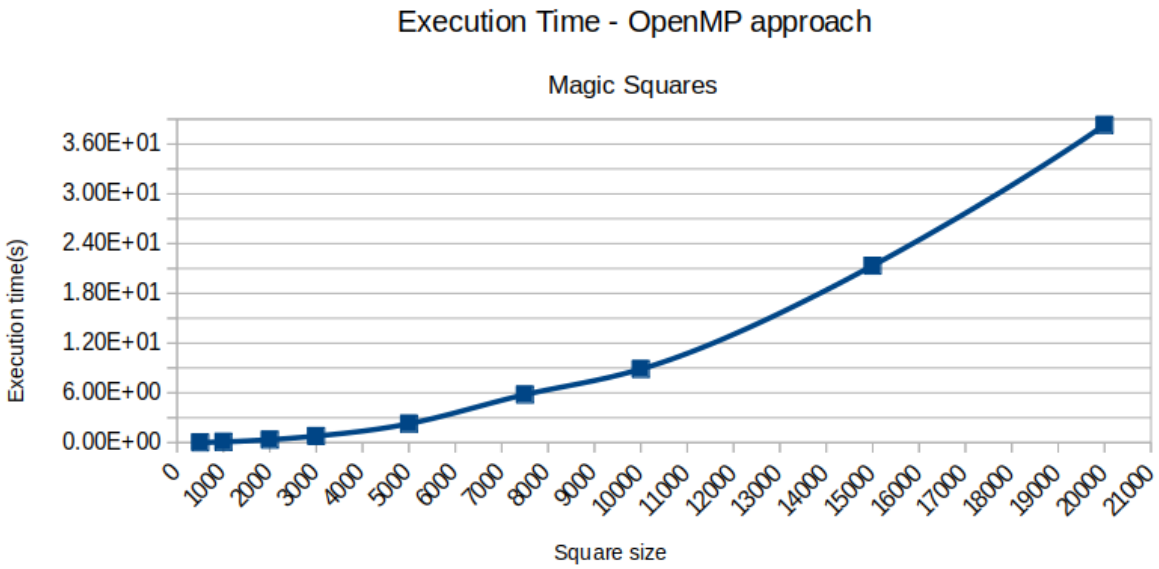
Define-se depois um limite máximo de 12 *threads*, número máximo de *threads* úteis suportadas pelo processador da máquina de teste, através de uma variável interna do *openMP*, e inicializa-se um scope de paralelização através de um *parallel pragma*. Foi feita uma divisão de 6 *threads* para análise das linhas e 6 *threads* para a análise das colunas. Cada uma das *threads* avalia então uma porção das colunas ou linhas, sendo que se uma delas encontrar uma linha ou coluna imperfeita será alterado o valor da variável global correspondente para 0, indicando a todas as outras *threads* que devem parar a análise. No caso de tal não acontecer e as variáveis globais continuarem com o valor 1, a *thread* principal analisa as diagonais, sendo que só se for encontrada uma diagonal não mágica é que o valor da variável global correspondente é alterado para 0. Como esta análise consiste numa única iteração não é feita a sua paralelização.

É depois feita então a análise de acordo com os valores das variáveis globais, sendo que se todas estiverem a 1, o quadrado é mágico, se apenas a das diagonais é 0, então o quadrado é imperfeito e para os restantes casos trata-se de uma quadrado não mágico.

Abaixo seguem-se os resultados dos tempos de execução correspondentes a execução do programa com quadrados de diversos tamanhos:







3.5 - Abordagem MPI

Esta secção detalha a implementação de um programa que identifica quadrado mágicos através de uma abordagem distribuída recorrendo à utilização do *MPICH*, com 4 processos no total.

O *MPI* é uma especificação de bibliotecas para computação parelelizada e distribuída, que define como é que a comunicação de dados entre dois processos deve ser realizada, isto mesmo que os dois processos se encontrem em máquinas diferentes.

Através do *MPICH*, uma implementação que obdece à definição de cima, é então possível tirar partido do hardware de sistemas computacionais para correrem concorrentemente vários processos de um mesmo programa, permitindo assim acelerar a sua execução [10].

A estratégia utilizada nesta abordagem foi a seguinte: Todos os processos leem através de um ficheiro de texto os elementos do quadrado mágico, colocam-nos em memória numa matriz bidimensional, calculam o valor da soma de todos os elementos da primeira linha e definem esse valor como a constante mágica. Existem três variáveis globais, todas elas inicializadas com o valor 0, que representam que por enquanto o quadrado não tem nenhuma linha, coluna ou diagonal imperfeita. Existe ainda uma variável local inicializada a 0 que identifica simultaneamente se foi encontrada uma coluna ou linha imperfeita.

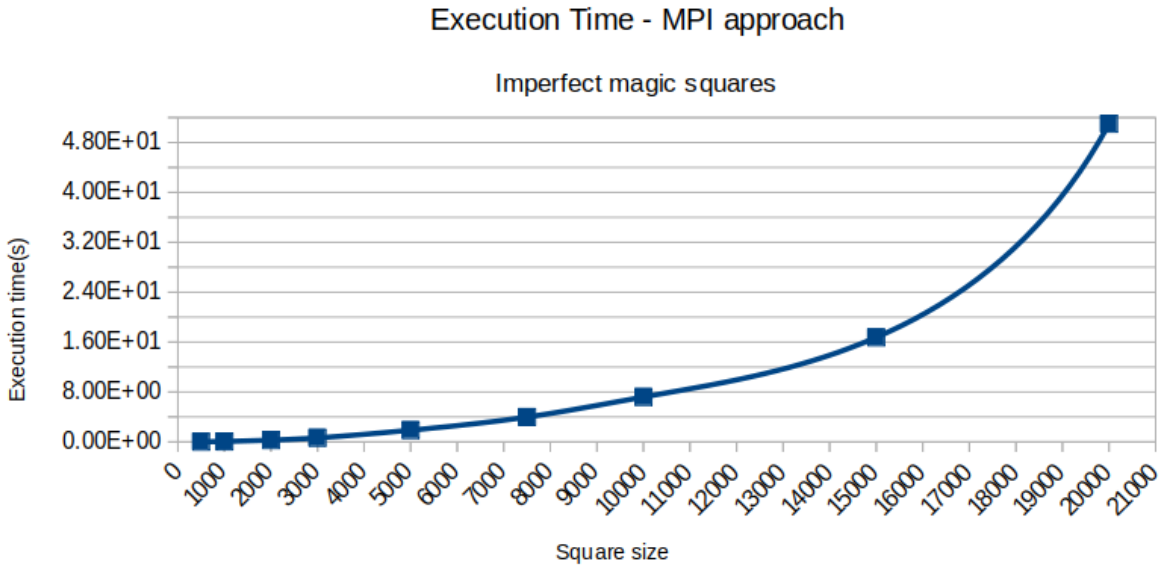
De seguida entra-se então na zona multi-processo do *MPI* : é feita uma divisão de metade dos processos para o processamento das linhas e a outra metade para o processamento das colunas. No caso de algum dos processos encontrar ou uma coluna imperfeita ou uma linha imperfeita, de acordo com a sua responsabilidade, este altera o valor da respetiva variável global correspondente para 1 e enviam-na, através um método de comunicação coletiva inerente ao *MPI*, para todos os outros processos, sendo que eles também fazem o mesmo.

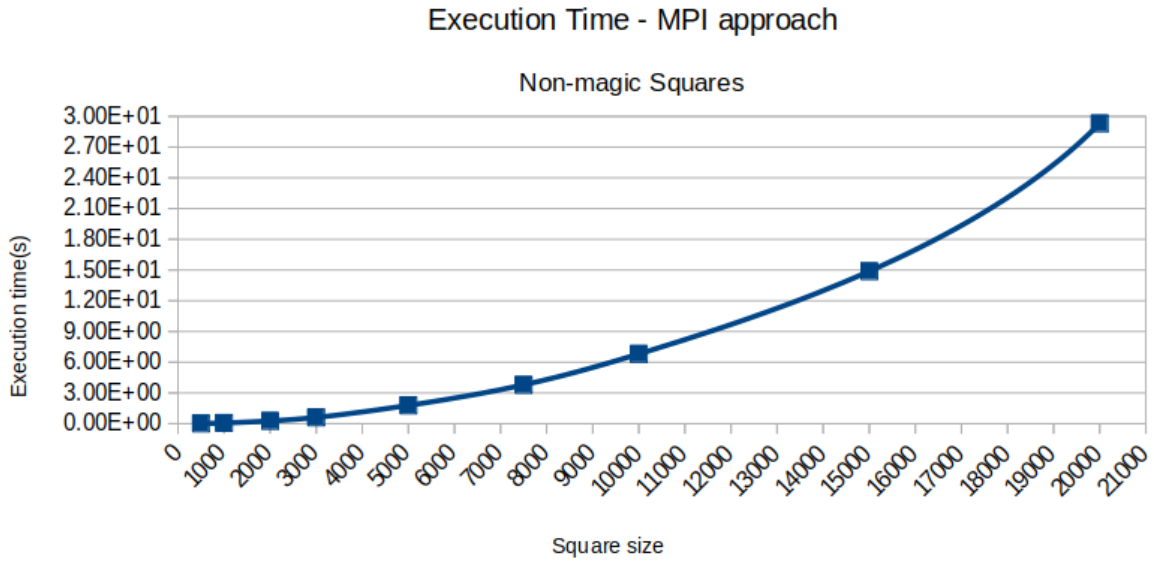
Todos estes valores são somados e guardados na variável local acima mencionada, sendo que se ela possuir um valor superior a 1 significa que existe pelo menos uma linha ou coluna imperfeita. Os processos através da mesma identificam quando deixam de fazer análises. Quando os processos identificam que o valor dessa variável é superior 1, alteram os valores das variáveis globais correspondentes para 0, isto de acordo com estarem a processar linhas ou colunas.

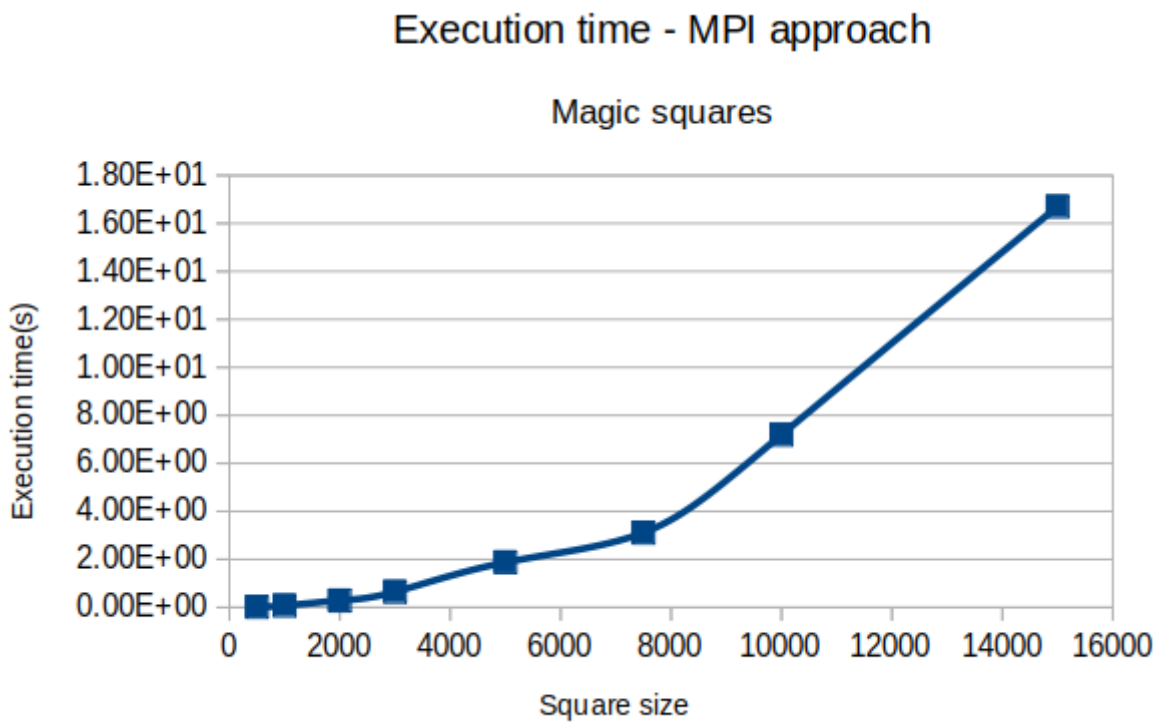
Terminado o processo de análise das linhas e colunas em todos os processos, o processo mãe verifica se o valor das variáveis globais correspondentes às colunas e linhas se encontram a 0. Se sim, analisa então as diagonais da matriz, sendo que se alguma delas não for mágica o valor da variável global das diagonais passa a 1. A análise das diagonais não é distribuída entre processos por se tratar de uma computação simples.

Por fim, de acordo com os valores das variáveis globais é determinado o tipo de quadrado : se se a variável das linhas ou colunas possuir o valor 1, então trata-se de um quadrado não mágico. Se ambas as variáveis possuírem o valor 0, e a variável global das diagonais possuir o valor 1, trata-se de um quadrado mágico imperfeito. Para o caso restante trata-se de um quadrado perfeito.

Abaixo seguem-se os resultados dos tempos de execução correspondentes a execução do programa com quadrados de diversos tamanhos:







3.6 - Abordagem híbrida - OpenMP e MPI

Esta secção detalha a implementação de um programa que identifica quadrado mágicos através de uma abordagem paralela e distribuída, recorrendo à utilização do *OpenMP* e do *MPICH*, com 4 processos no total.

A estratégia utilizada nesta abordagem foi a seguinte: Todos os processos leem através de um ficheiro de texto os elementos do quadrado mágico, colocam-nos em memória numa matriz bidimensional, calculam o valor da soma de todos os elementos da primeira linha e definem esse valor como a constante mágica. Existem três variáveis globais inicializadas com o valor 0, que representam que por enquanto o quadrado não tem nenhuma linha, coluna ou diagonal imperfeita. Existe ainda uma variável local inicializada a 0 que identifica simultaneamente se uma coluna ou linha é imperfeita.

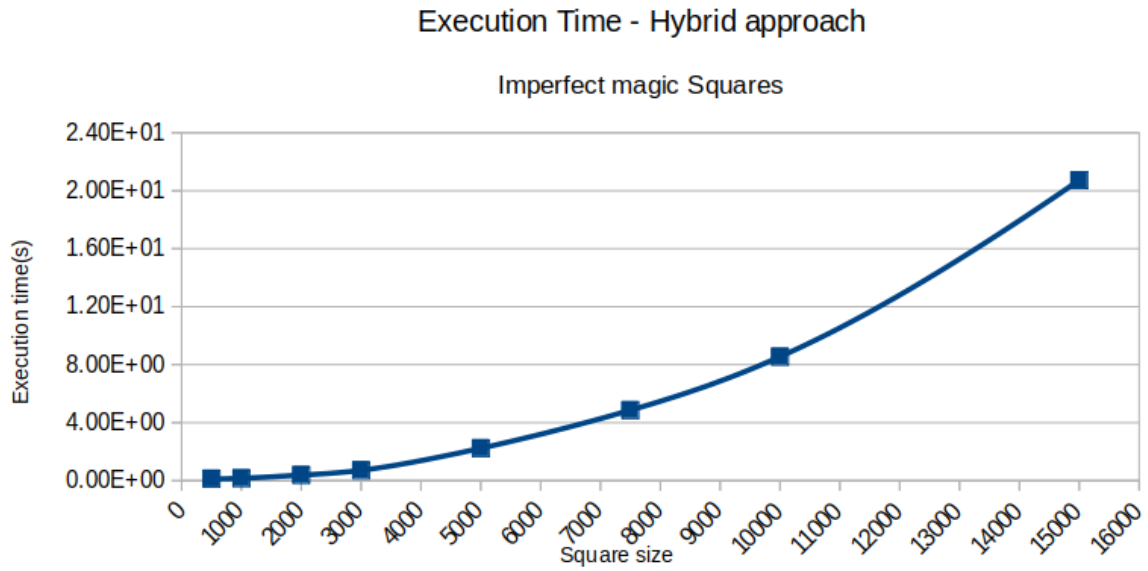
De seguida entra-se então na zona multi-processo do *MPI* : é feita uma divisão de metade dos processos para o processamento das linhas e a outra metade para o processamento das colunas. No caso de algum dos processos encontrar ou uma coluna imperfeita ou uma linha imperfeita, de acordo com a sua responsabilidade, este altera o valor da respetiva variável global correspondente para 1 e enviam-na, através um método de comunicação coletiva inerente ao *MPI*, para todos os outros processos, sendo que eles também fazem o mesmo. Cada um dos processos divide a secção porque que está responsável por 8 *threads*. Cada *thread* irá portanto tratar de uma sub-secção desta secção.

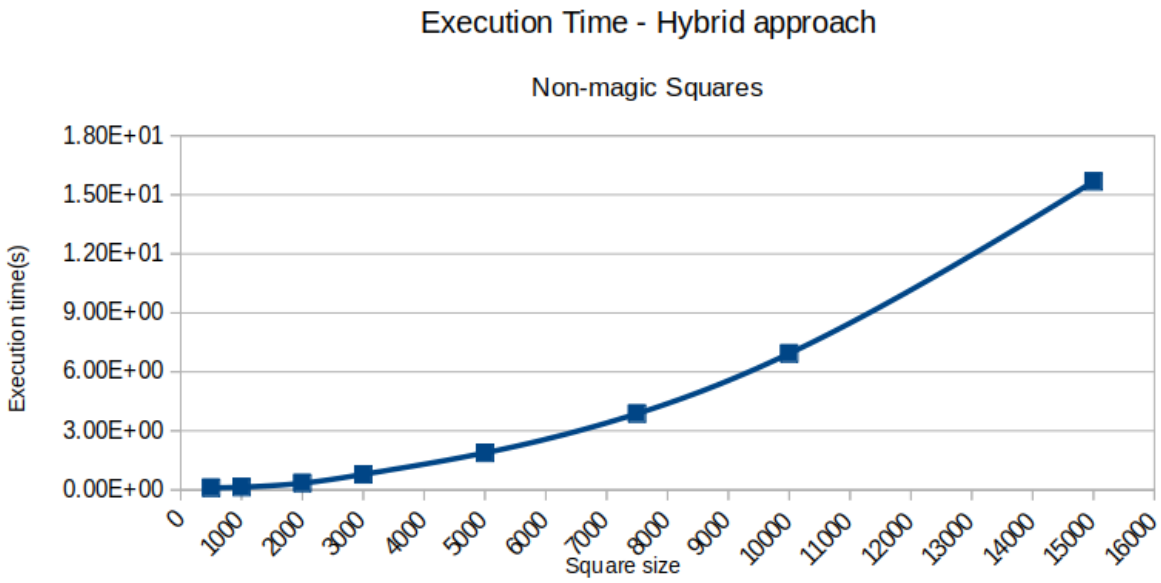
Todos estes valores são somados e guardados na variável local acima mencionada, sendo que se ela possuir um valor superior a 1 significa que existe pelo menos uma linha ou coluna imperfeita. Os processos através da mesma identificam quando deixam de fazer análises. O mesmo acontece com as *threads*, quando os processos identificam que o valor dessa variável é superior 1, alteram os valores das variáveis globais correspondentes para 0.

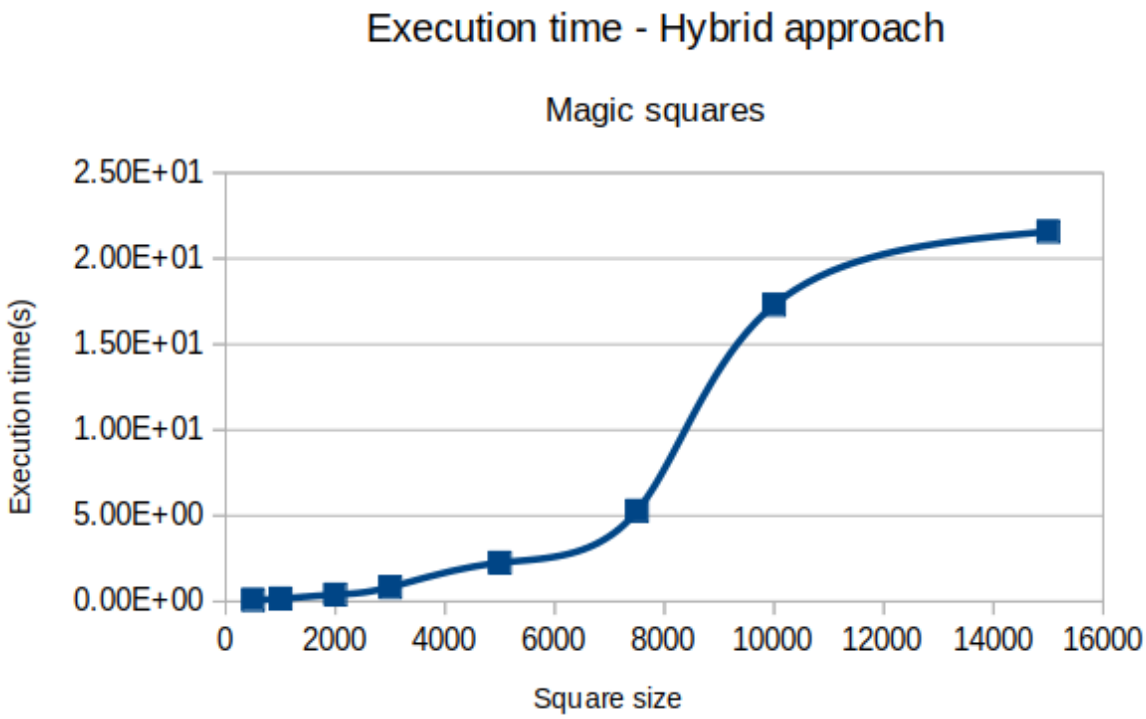
Terminado o processo de análise das linhas e colunas em todos os processos, o processo mãe verifica o valor das variáveis globais correspondentes às colunas e linhas se encontram a 0. Se sim, analisa então as diagonais da matriz, sendo que se alguma delas não for mágica, o valor da variável global das diagonais passa a 1. A análise das diagonais não é distribuída entre processos por se tratar de uma computação simples.

Por fim, de acordo com os valores das variáveis globais é determinado o tipo de quadrado : se se a variável das linhas ou colunas possuir o valor 1, então trata-se de um quadrado não mágico. Se ambas as variáveis possuírem o valor 0, e a variável global das diagonais possuir o valor 1, trata-se de um quadrado mágico imperfeito. Para o caso restante trata-se de um quadrado perfeito.

Abaixo seguem-se os resultados dos tempos de execução correspondentes a execução do programa com quadrados de diversos tamanhos:





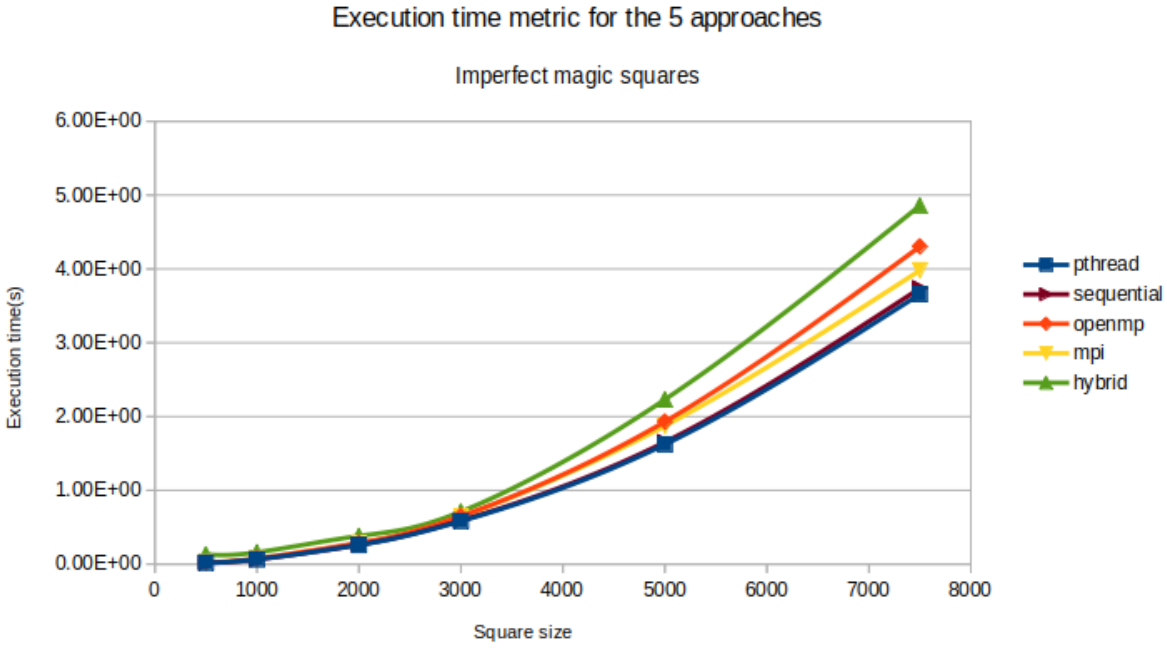


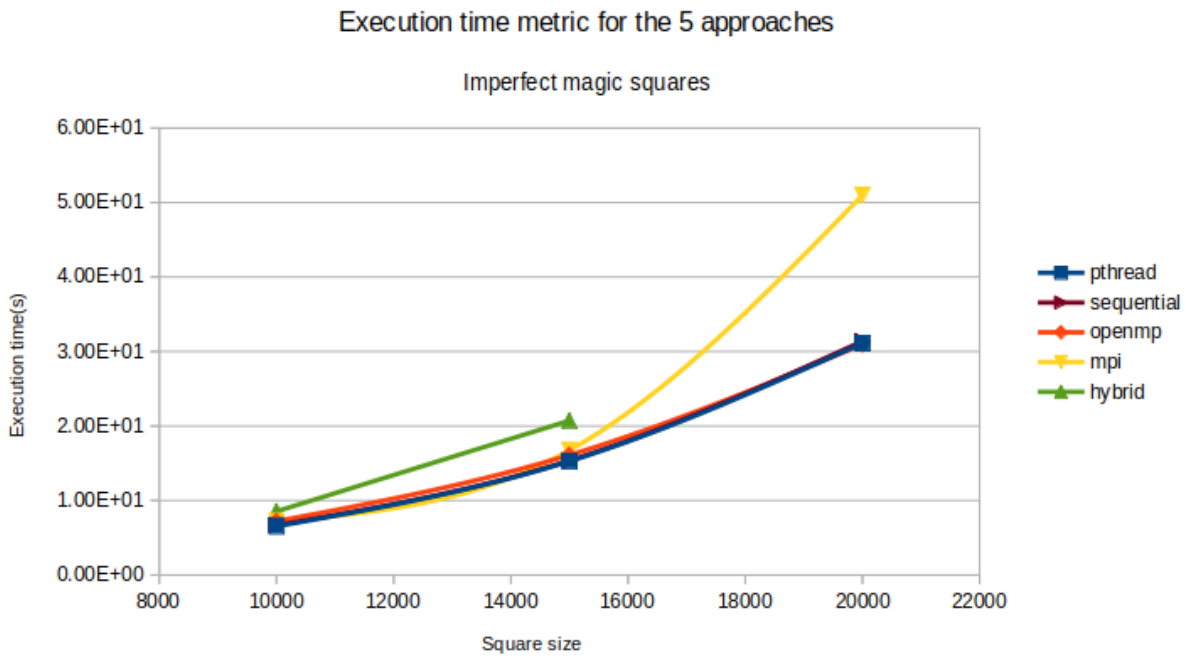
4 - Análise e discussão de resultados

A discussão e análise dos resultados assentará na análise de 4 métricas, já descritas na metodologia : *tempo de execução*, *gradiente de execução*, *aceleração* e *eficiência*.

4.1 - Tempos de execução

Começando pela análise dos *tempos de execução*, abaixo seguem-se os gráficos onde constam todos os tempos de execução para as diferentes abordagens desenvolvidas, incluindo a sequencial, para os **quadrados mágicos imperfeitos**. Os tempos de execução estão divididos em dois gráficos devido à grandeza dos tempos, no sentido de deixar o gráfico mais claro.

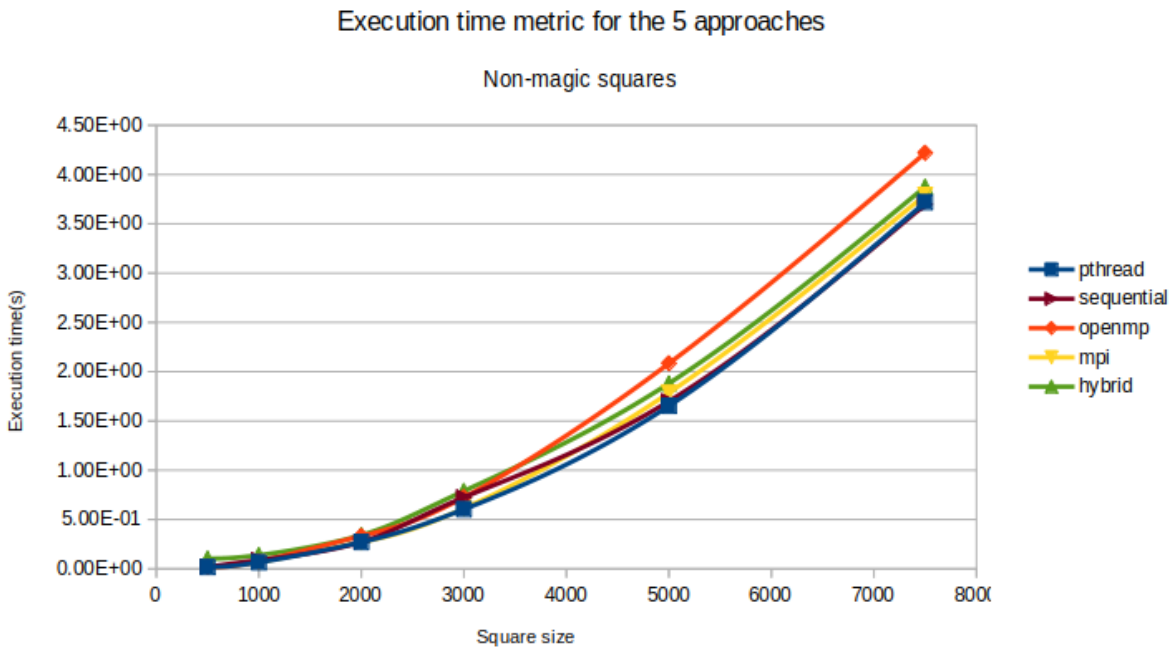


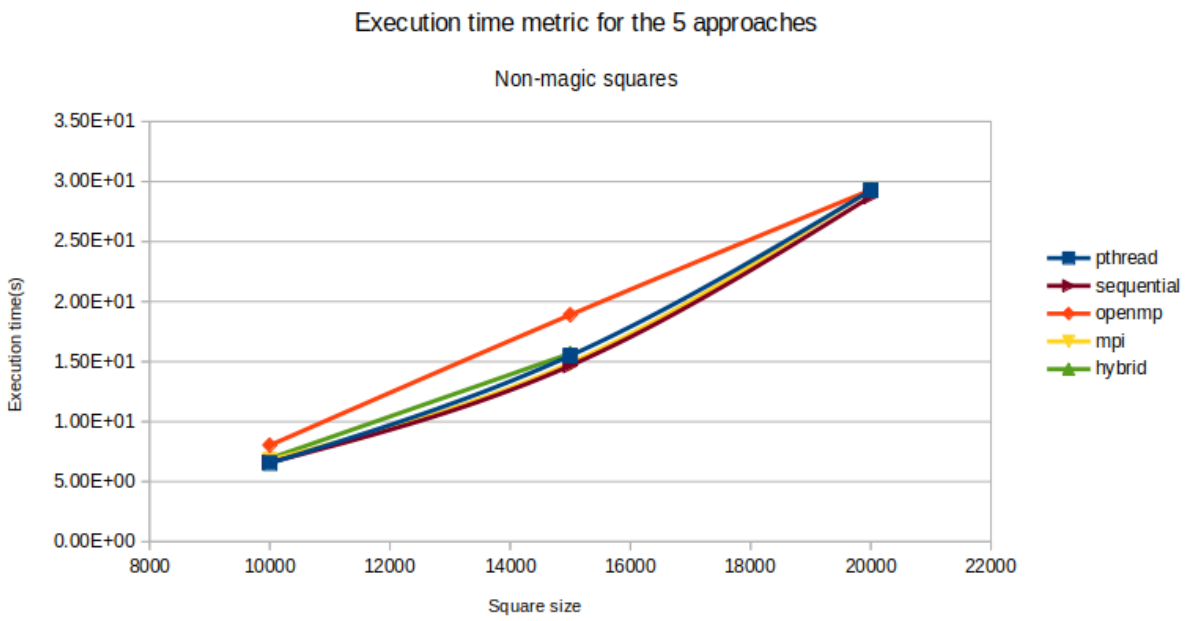


Através do gráfico consegue-se determinar que em relação ao tempos de execução em quadrados mágicos imperfeitos, para dimensões pequenas a intermédias as abordagens sequencial e com *Pthreads* são as mais rápidas, com a *MPI* um pouco mais lenta que essas duas e próxima da abordagem *openMP* e por fim a *híbrida*, bem mais lenta que as restantes à medida que a dimensão aumenta.

Para quadrados de grandes dimensões, já se verifica uma grande proximidade de tempos entre a abordagem sequencial, *Pthreads* , *OpenMP* e *MPI*, continuando a abordagem híbrida muito distante em termos de tempos. É de mencionar que para os quadrados de dimensão de 20000, a implementação com *MPI* demonstra-se extremamente lenta. Isto deve-se, muito provavelmente, à memória RAM do servidor que fica praticamente esgotada com quatro matrizes de 20 mil, interferindo muito negativamente no desempenho do *MPI*.

Seguem-se, de seguida, os gráficos relativos ao **quadrados não-mágicos**, também divididos em dois:

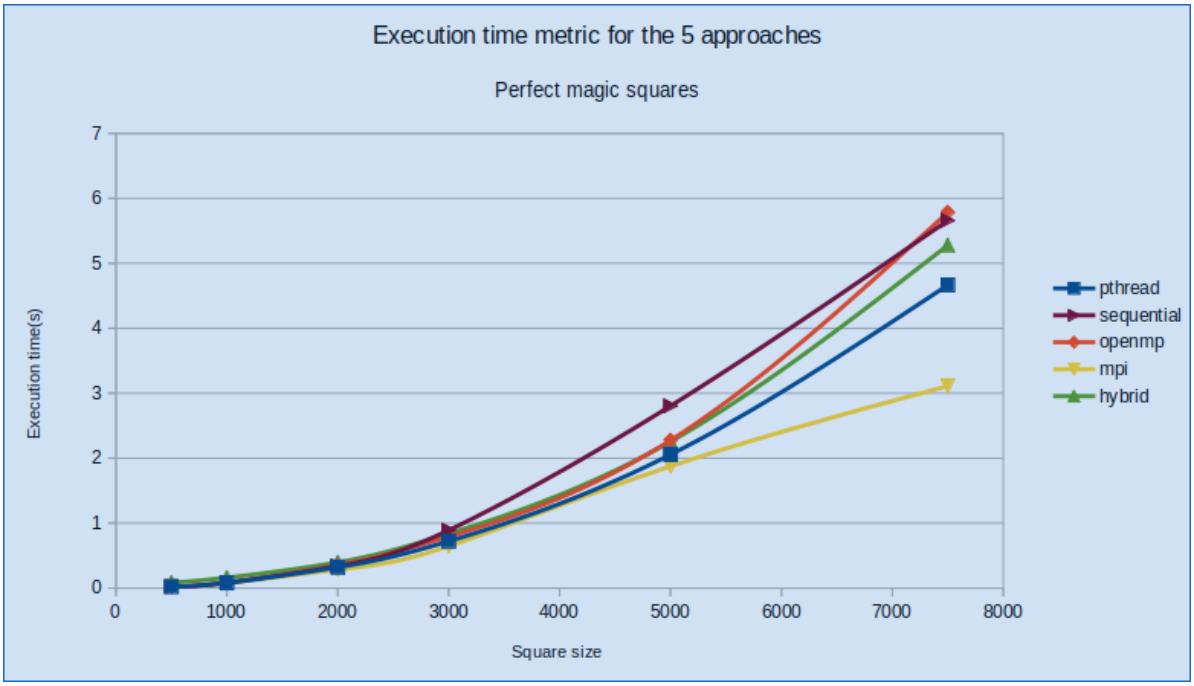


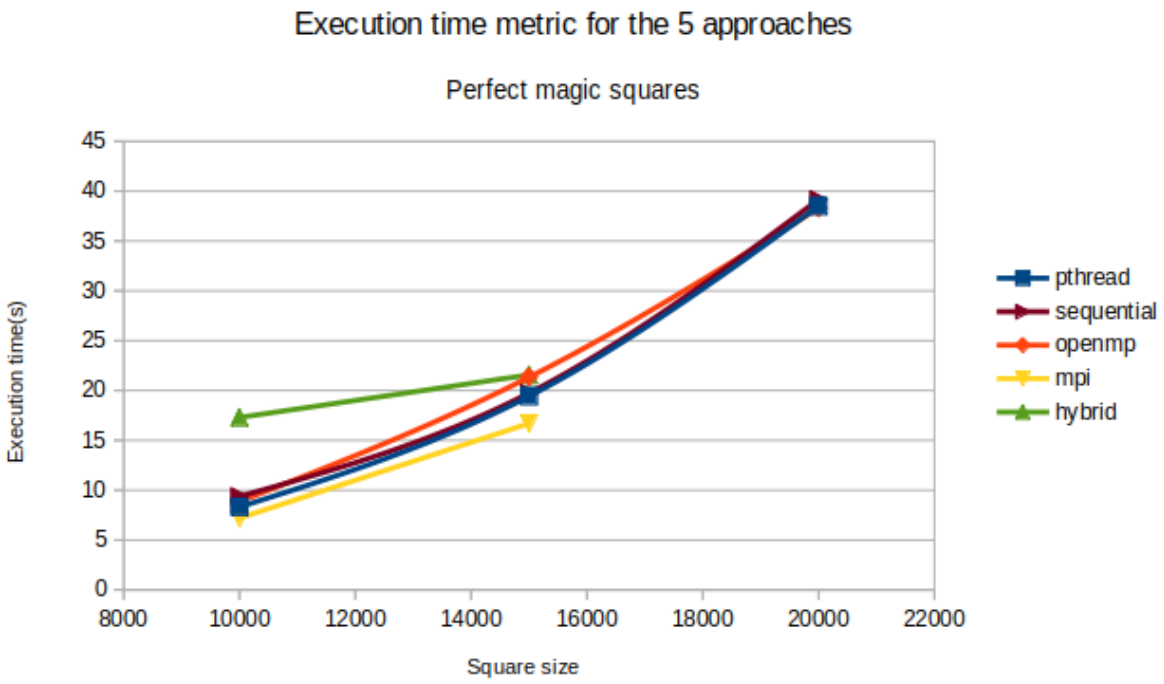


Em primeira análise, é possível concluir que para dimensões pequenas a intermédias, a abordagem com *Pthreads* é, sem margem de dúvidas, a implementação rápida. A implementação sequencial consolida o segundo lugar, com a implementação *MPI* muito próxima e logo de seguida a implementação *híbrida*. Em última lugar encontra-se a abordagem com *openMP*, que foi geralmente sempre o mais lento.

Para grandes dimensões, pode-se verificar uma grande proximidade em termos de tempos em relação a quatro abordagens, nomeadamente entre a sequencial, *pthreads*, *MPI* e *híbrida*. A abordagem com *openMP* encontra-se novamente muito mas lenta que as restantes. A abordagem que se considera a mais rápida é a *sequential*, ainda que com uma margem mínima em relação às restantes.

Em último lugar, temos os gráficos para os testes com **quadrados perfeitos** para todas as abordagens. Como para os outros dois tipos de quadrados, os testes estão novamente divididos em dois gráficos :





Para quadrados de baixa e intermédia dimensão, pode-se determinar rapidamente que a implementação com base no *MPI* foi, sem margem de dúvidas, a mais rápida seguindo-se a *pthread*, encontrado-se depois a *openMP* e a *híbrida* muito renhidas, com a *openMP* a piorar em muito para o teste de 20 mil. Em último lugar temos a implementação sequencial.

Para os quadrados de elevada, temos o *MPI* novamente como a abordagem mais rápida, constituíndo-se a abordagem *híbrida* a mais lenta. A abordagem *sequential* aparece como a segunda abordagem mais rápida, seguindo-se da abordagem *sequential* e por fim a abordagem com *openMP*. É de notar que a abordagem *sequential* e *pthreads* estão muitissimo próximas uma da outra.

4.1.1 - Discussão dos tempos de execução

Antes de mais, é importante explicar que para os quadrados perfeitos e imperfeitos, a abordagem *híbrida*/*MPI* e a abordagem *MPI* não apresentam resultados para quadrados de dimensão 20 mil devido a problemas de desempenho da máquina utilizada. Esta tinha, muitas das vezes, problemas em lidar com quatro matrizes com dimensão de 20 mil, tornando-se o seu desempenho muitíssimo baixo e por consequente os tempos de execução muito elevados. Por isso mesmo, decidiu-se optar por não incluir esses resultados no estudo, uma vez que têm um significado muito pouco relevante e conclusivo.

Uma das principais ilações que se pode retirar destes testes é que o *MPI*, apesar de ter resultados muito medíocres em quadrados imperfeitos e não mágicos, é a melhor abordagem para quadrados perfeitos, batendo todas as outras abordagens de forma expressiva. Os maus resultados do *MPI* nos restantes testes poderão explicar-se pelo desempenho do próprio servidor, que nas alturas em que os testes foram feitos poderiam estar ocupado a processar outros processos, impactando severamente o desempenho da abordagem.

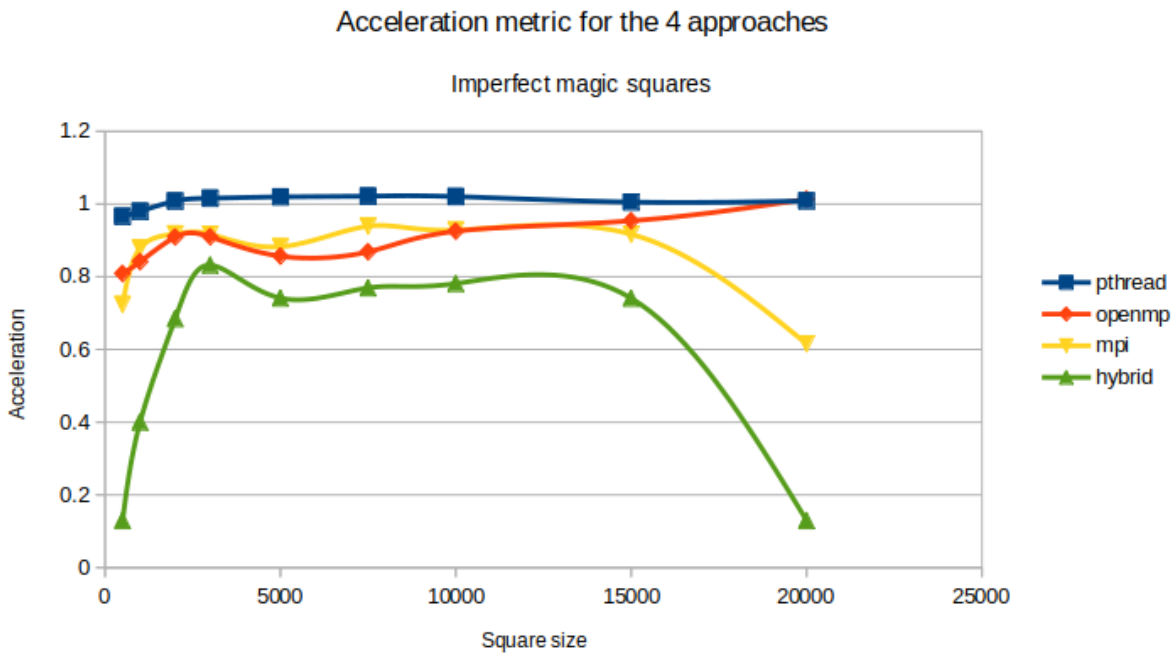
A abordagem *híbrida* demonstra também resultados desapontantes, especialmente considerando que recorre a múltiplos processos e *threads*, possuindo resultados piores que a abordagem sequencial na grande maioria dos casos. Uma possível e plausível explicação para estes resultados é também o desempenho do próprio servidor, que poderia estar a processar

outros programas enquanto se faziam os testes.

Outra ilação relevante é que a abordagem que de facto se demonstra mais estável em termos de resultados positivos, estando muitas vezes próxima de ser a melhor, é a abordagem com recurso a *Pthreads*. A utilização da biblioteca das *Pthreads* sem recurso a nenhuma *API* de alto nível, como é o caso do *OpenMP*, demonstra-se a melhor alternativa para obtermos o melhor desempenho geral, comportando-se muito bem em todas as dimensões e para os diferentes tipos de quadrados.

4.2 - Aceleração

A segunda métrica a ser revista nas abordagens é a aceleração. A análise irá começar pela quadrados mágicos imperfeitos, seguindo-se imediatamente os gráficos :

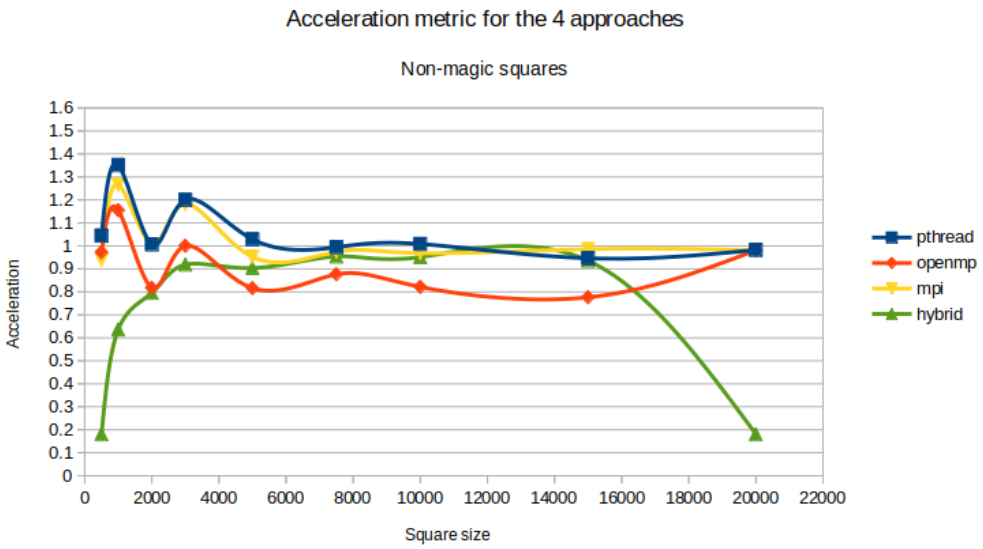


Através do gráfico é possível concluir que para quadrados imperfeitos, apenas a implementação com *pthreads* consegue, de forma relativamente consistente, ter um desempenho superior à implementação sequencial.

As abordagens *openMP* e *MPI* possuem para as diferentes dimensões dos quadrados valores de aceleração elevado e próximos de 1, com o *openMP* a ser capaz de ultrapassar esse teto com a dimensão de 20 mil. Por outro lado, o *MPI* nunca chega a alcançar ou a ultrapassar o valor 1 da aceleração, mostrando-se sempre mais lento que o sequencial e inclusive piorando significativamente para a dimensão de 20 mil.

A abordagem *híbrido* apresenta resultados muito pobres, mostrando-se consideravelmente pior em termos de desempenho em relação à abordagem sequencial, atingindo o seu pico com a dimensão de 3001 e com um valor pouco acima de 0.8.

Seguem-se, de seguida, o gráfico relativo à aceleração em **quadrados não mágicos** :

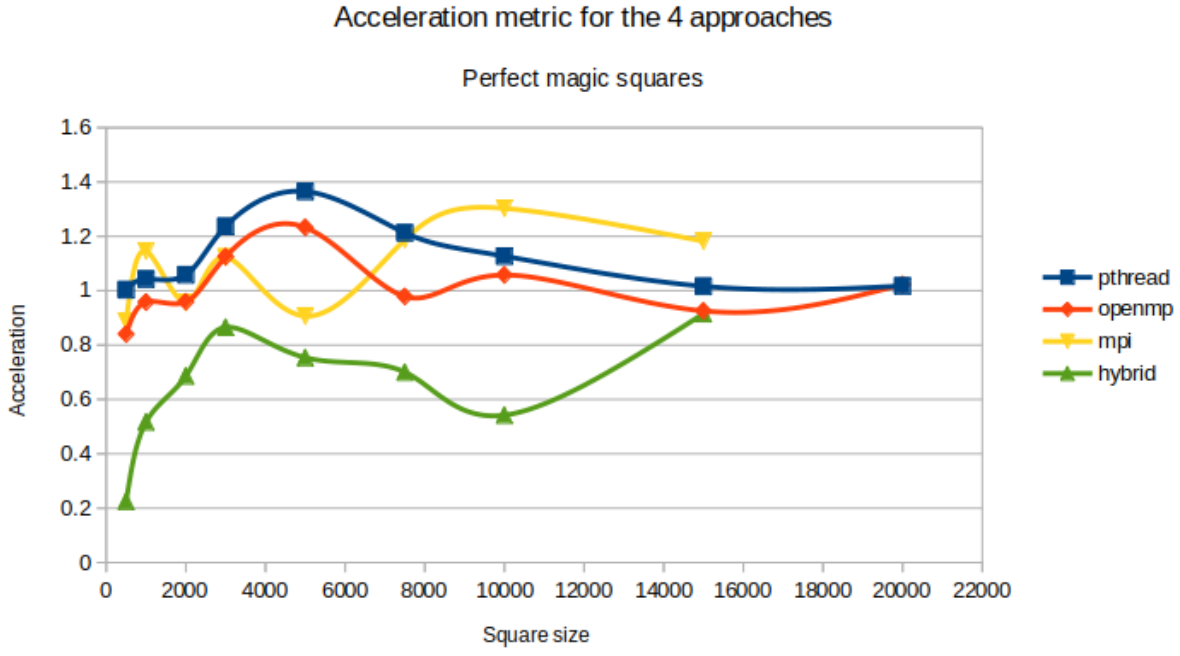


Em relação aos testes a quadrados não mágicos, a abordagem com *pthreads* continua a ser a abordagem que melhores resultados produz, sendo a que de forma mais consistente se mantém com uma aceleração superior a 1. Apenas para os teste de maior dimensão, nomeadamente para as dimensões de 15 mil e 20 mil, é que as *Pthreads* possuem um valor de aceleração inferior a 1.

De seguida temos o *MPI* , com resultados inferiores mas muito próximos aos do *Pthread* ainda que a maioria seja inferior a 1. Depois vêm o *openMP*, que apenas ultrapassa a barreira de aceleração para a dimensão 1001, mantendo-se os seus restantes valores inferiores a 1.

Por fim temos a abordagem híbrida, que para as matrizes iniciais possui resultados péssimos, melhorando progressivamente e ultrapassando até o *openMP* nas matrizes de dimensão intermédia e grande, à exceção da matriz de 20 mil, onde volta a piorar de forma agressiva.

Em último caso, temos a análise da aceleração para os **quadrados perfeitos**:



No que toca à análise da aceleração para os quadrados, podemos verificar que em todos os testes a abordagem com *Pthreads* conseguiu ter uma aceleração sempre superior a 1, constituindo-se portanto a implementação mais consistente em termos de aceleração.

Surpreendentemente, depois dos seus resultados passados, temos a implementação *MPI* que também obteve muitos bons resultados, ultrapassando em alguns testes a aceleração das *Pthreads*, em especial para quadrados de maior dimensão.

Temos ainda a abordagem *openMP* que para testes de dimensão intermédia e dimensão já mais elevada,conseguiu obter valores de aceleração superiores a 1, não conseguindo no entanto competir com a abordagem *MPI*.

Por fim, temos a abordagem *híbrida*, que em nenhum dos testes conseguiu ultrapassar a fasquia da aceleração 1, mantendo-se sempre mais lenta que a sequencial.

4.2.1 - Discussão da métrica da aceleração

Para os três tipos de quadrados, a implementação mais bem sucedida é sem dúvida a abordagem com *Pthreads*. Para a grande maioria dos casos esta possui sempre um valor de aceleração superior a 1, mantendo-se sempre muito consistente nesse sentido.

O *OpenMP* apresenta resultados abaixo do desejados, não conseguindo para a maioria dos casos nos três tipos de quadrados ser mais veloz que a implementação sequencial, excedendo-se apenas com alguma expressividade no caso dos quadrados perfeitos.

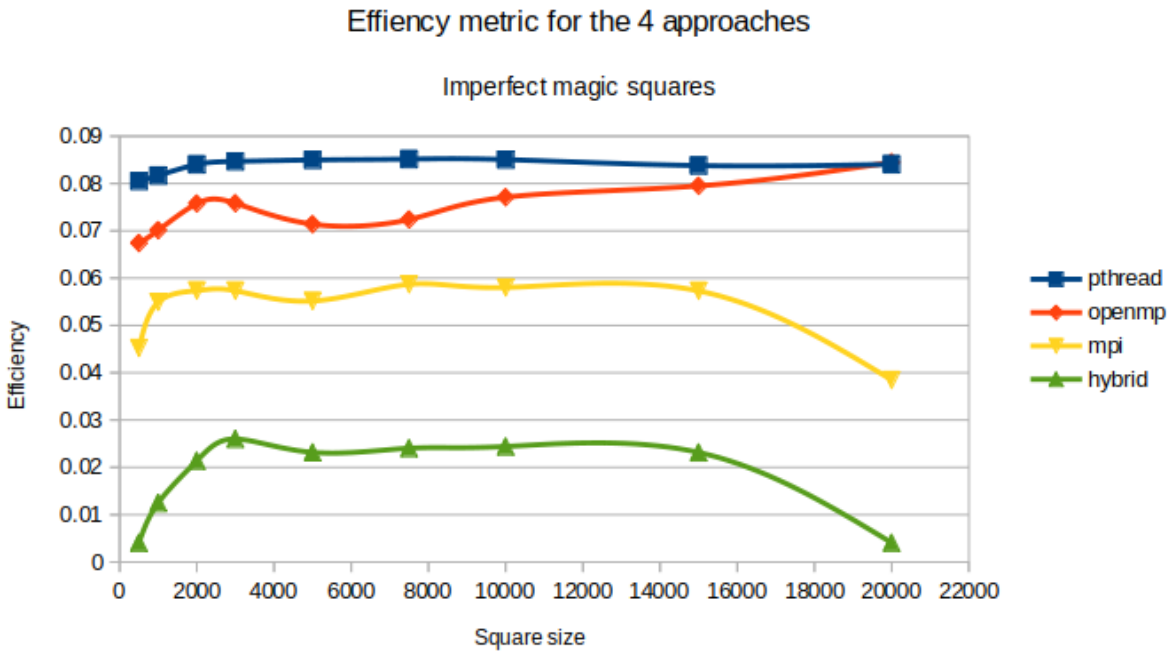
A implementação *MPI* também possui resultados muito desapontantes, especialmente no que toca aos quadrados imperfeitos, sobressaindo-se no entanto nos quadrados perfeitos com os melhores valores de aceleração, estes bastante expressivos, para as matrizes de maior dimensão.

Para a abordagem *híbrida*, não houve qualquer teste em que tenha conseguido ter um melhor performance que a abordagem sequencial, mostrando-se extremamente lenta.

Uma razão válida para o mau desempenho das implementações *MPI* e *híbrida*, como já referida na discussão dos tempos de execução, poderá estar relacionada com o sobre-carregamento do servidor. Poderá ainda contribuir para os desempenhos péssimos do híbrido o facto de as abordagens *multi-threading* serem pouco compatíveis com os métodos do *MPI*, em especial com métodos de comunicação coletiva. Poderá, portanto, ser também por isso que a abordagem híbrida, apesar de ter tudo para brilhar, esteja a produzir resultados muito fracos.

4.3 - Eficiência

A terceira métrica a ser analisada é então a eficiência. A análise da mesma começará também pelos testes a quadrados mágicos imperfeitos, sendo agora apresentado o gráfico com os respetivos resultados:



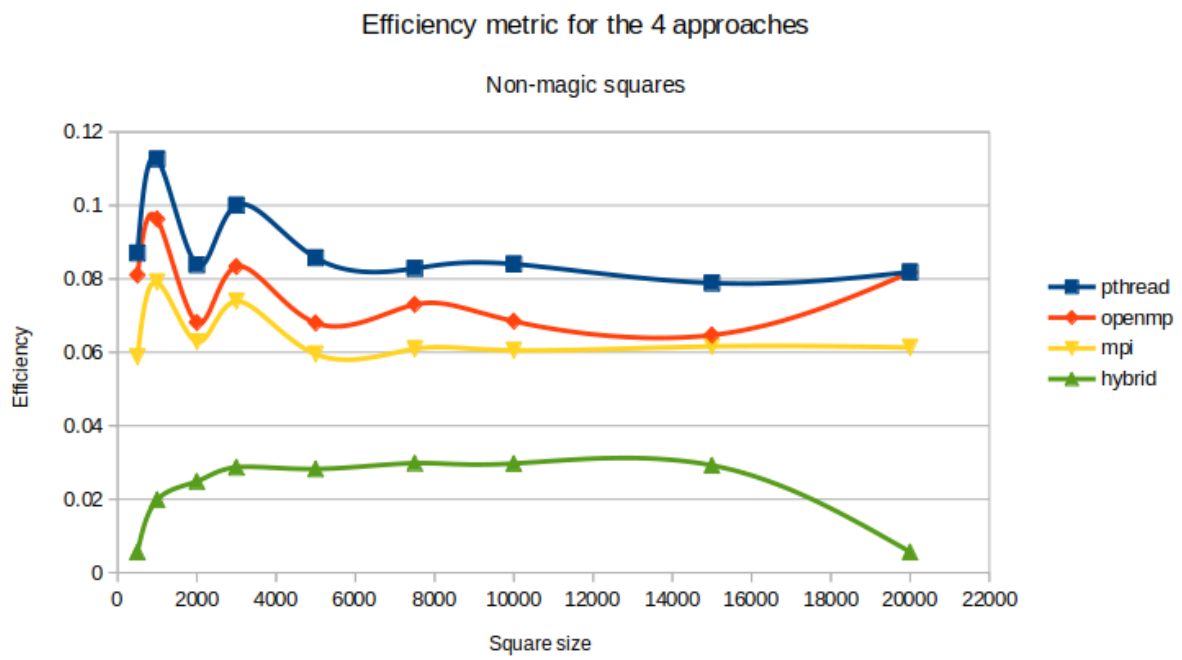
Aquilo que podemos verificar deste gráfico é que a abordagem com *Pthreads*, que recorreu a 8 *threads* de execução já com *HyperThreading*, possui uma eficiência que varia entre 0.08 e 0.09, ou seja, ainda que utilize 12 unidades de processamento, o seu desempenho está muito longe de ser 12 vezes mais rápido.

A abordagem *openMP*, que se apresenta nos mesmo modelos que a *Pthreads*, apresenta resultados de eficiência ainda piores que as *Pthreads*, provando-se uma implementação ainda menos eficiente.

No que toca à abordagem *MPI*, que recorreu a 16 **cores** do processador divididos por 4 nós para execução do programa, é possível concluir que a implementação encontra-se extremamente longe de ser 16 vezes mais rápida, com uma eficiência a rondar entre 0.04 e 0.06.

Por fim, para a abordagem híbrida, que recorreu a 32 *threads* de execução, esta possui uma eficiência que varia entre valores próximos de 0 e 0.0275. Constituindo-se a abordagem mais lenta de todas, é natural que a sua eficiência, especialmente quando recorrendo a 32 *threads de execução*, seja extremamente baixa.

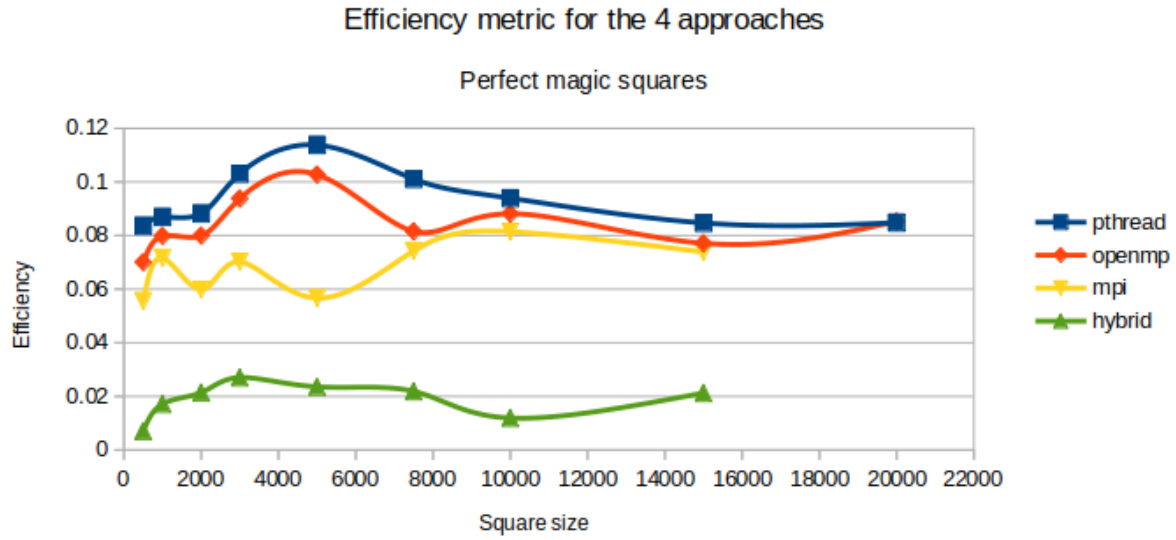
De seguida, apresenta-se o gráfico da eficiência para **quadrados não mágicos** :



No que toca a análise de quadrados não mágicos, a implementação com melhor eficiência continua a ser a *Pthreads*, seguida da *openMP*, *MPI* e de seguida a *hybrid*, tal como acontece para o quadrado mágico imperfeito.

O que é de notar é que na generalidade, os valores das eficiências verificam um aumento quando comparados com os quadrados imperfeitos, à exceção da implementação híbrida que permanece praticamente idêntica.

Por fim, apresenta-se o gráfico da eficiência para **quadrados mágicos perfeitos**:



Como já foi referido na análise dos tempos de execução para os quadrados perfeitos, não foram feitas medições para dimensões de 20 mil para as abordagens *MPI* e *hybrid*.

As ilações que se podem retirar deste gráfico é que, como se tem vindo a verificar, as *Pthreads* são a abordagem que possui a maior eficiência em todos os testes, seguida da abordagem *OpenMP* e depois o *MPI*. Em último lugar permanece a implementação híbrida.

É de notar que os valores das eficiências obtidos são, na sua generalidade, muito semelhantes aos obtidos para os quadrados não perfeitos.

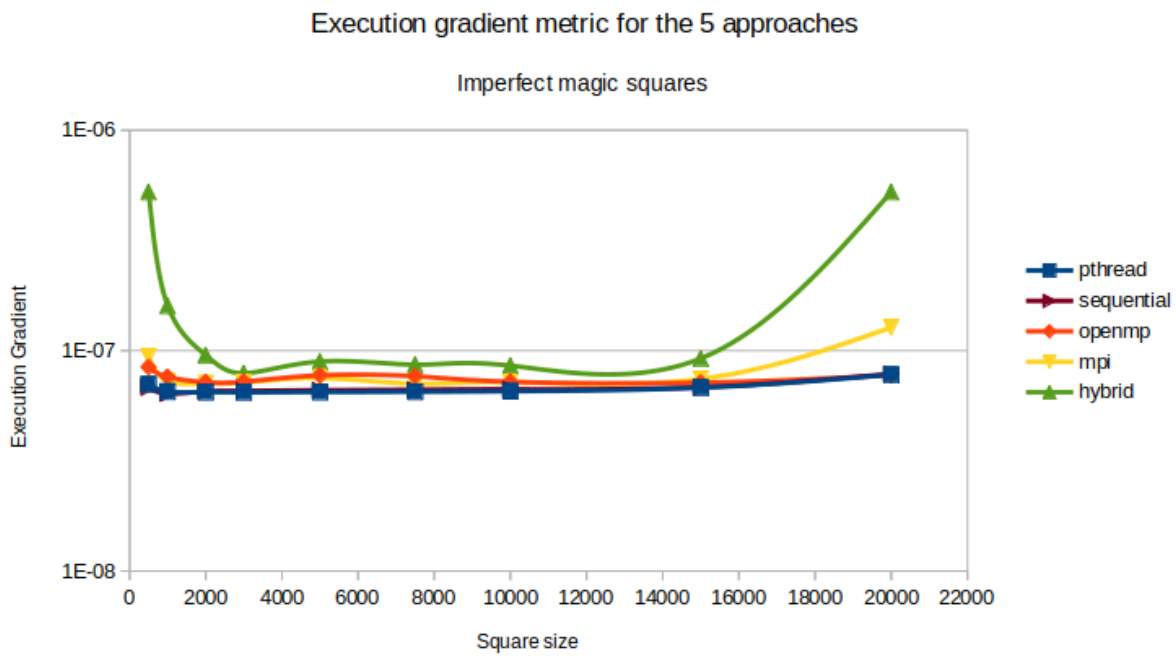
4.3.1 - Discussão da métrica eficiência

É possível perceber através destes três gráficos que, sem margem de discussão, a implementação com *Pthreads* é a mais eficiente de todas, ainda que o valor da eficiência seja bastante baixo, chegando a atingir 0.15 em alguns casos. É também claro que a pior implementação, de longe, é a implementação *híbrida*, com valores de eficiência extremamente baixos.

Nenhuma das abordagens conseguiu, de longe, ter valores de eficiência unitários, o que demonstra que o impacto de adicionar mais processadores teve um impacto muito reduzido no aumento do desempenho.

4.4 - Gradiente de execução

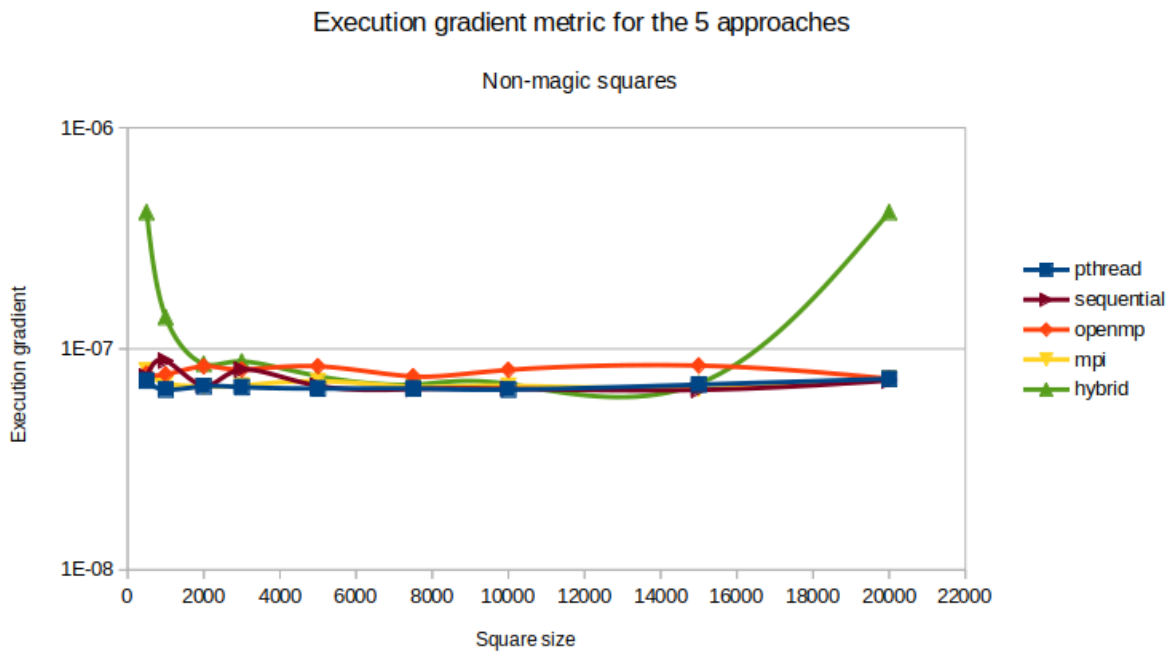
A última métrica a ser analisada será o gradiente de execução. Abaixo apresenta-se o primeiro gráfico referente a **quadrados mágicos imperfeitos**:



Pode-se verificar que a abordagem *sequential* e *pthread* mantêm constantes o seu gradiente de execução ao longo das diferentes dimensões. Já o *MPI* e o *openMP* demonstram algumas perturbâncias no seu gradiente de execução, ainda que relativamente baixas.

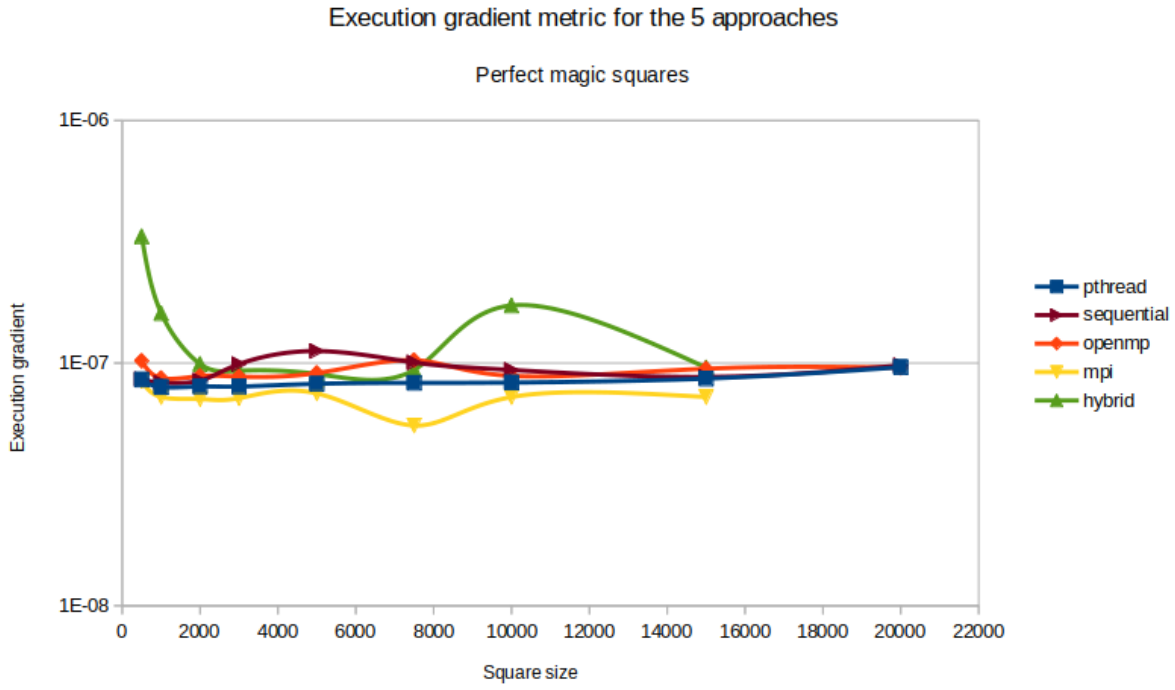
A implementação *híbrida* possui um gradiente de execução completamente irregular.

De seguida, apresenta-se o gráfico associado aos **quadrados não mágicos:**



Aqui verificam-se algumas perturbações mínimas no gradiente de execução da abordagem sequential, com o gradiente de execução do *MPI* e das *Pthreads* a manter-se muitíssimo estável. O *OpenMP* apresenta algumas perturbações já consideráveis no seu gradiente e a implementação *híbrida* continua com perturbações enormes no seu, especialmente nos extremos das dimensões dos testes.

Por fim, apresenta-se o gráfico associado aos **quadrados mágicos perfeitos** :



Para este tipo de quadrados, a única abordagem que manteve o seu gradiente de execução completamente estável durante todos os testes foi novamente a *Pthreads*. A abordagem *openMP* apresenta uma ligeira perturbação nos testes intermédios e a abordagem *sequential* e *MPI* apresentam perturbações mais significativas também nos testes intermédios.

A implementação híbrida continua perturbações enormes ao longo dos vários testes.

4.4.1 - Discussão do gradiente de execução

A implementação que mais manteve o seu gradiente de execução estável, independentemente do tipo de quadrados, foi a abordagem com *Pthreads*, existindo muito poucas alterações na mesma. A abordagem *sequential* também se demonstra muito estável, apenas com ligeiras perturbações em alguns quadrados perfeitos.

Já com algumas perturbações nos seu gradiente temos o *MPI* , que apenas apresentou perturbações algo consideráveis nos testes intermédios em quadrados perfeitos. Já o *openMP* demonstrou sempre algumas perturbações já consideráveis, tendo-se saído melhor nos quadrados do tipo imperfeitos.

A implementação híbrida em qualquer um dos tipos de quadrados teve um gradiente de execução repleto de perturbações muito elevadas. Acontece que os tempos de execução sobem de forma muito mais acentuada quando comparados com o aumento da dimensão do problema, causando estas grandes picos.

5 - Conclusão e comentários

É possível concluir, após estas baterias de testes e diferentes análises às métricas propostas, que a melhor implementação em termos de resultados é sem dúvida a abordagem com *Pthreads*. Demonstra-se a abordagem com melhores acelerações, eficiências e possui um gradiente de execução muitíssimo estável ao longo dos diferentes testes realizados. É ainda relevante mencionar que se trata de uma abordagem de relativa fácil implementação, não requerendo uso de métodos complexos.

A abordagem *openMP*, que é em muito semelhante às *Pthreads*, apresenta na generalidade resultados significativamente piores à sua irmã, desaconselhando-se o seu uso. Tal poderá estar relacionado com o facto de ser um biblioteca de alto nível capaz de trabalhar com as diferentes threads de acordo com o sistema operativo em causa, o que por sua vez requer uma abstração mais elaborada tornado-a mais lenta. Convém referir que, apesar de mais lenta, é extremamente direta e fácil de utilizar, permitindo paralelizar código rapidamente e sem grandes preocupações. Infelizmente, quando comparada com a abordagem sequencial, esta paralelização não contribui para um aumento do desempenho, e cuja razão também poderá assentar naquilo que foi explicado mais acima. A abordagem *sequential* em muitos dos casos demonstrou ter um desempenho superior à *openMP*, pelo o que não compensa implementar paralelização através do *openMP*.

No que toca à abordagem *MPI*, ela apresenta tempos muito desapontantes para quadrados imperfeitos, melhorando para os quadrados não perfeitos, e apresentando depois os melhores resultados para quadrados perfeitos de elevada dimensão. Os resultados mais lentos para dimensões baixas podem-se explicar com a necessidade de inicializar os diferentes processos, cujo tempo acabava por ser maior que o tempo que a paralelização diminui, acabando por não compensar no final.

Em relação à abordagem híbrida, ela para quase todo o tipo de quadrados e respetivas dimensões apresenta resultados de desempenho muito baixos, mostrando-se extremamente lenta. Um dos maiores fatores para tal está relacionado com o facto dos métodos de comunicação coletiva do *MPI* não funcionarem bem quando paralelizados por *threads*. Muito provavelmente não serão *thread-safe*, provocando um *clog* enorme no processamento e aumentando em muito os tempos de execução.

Outro fator que afeta em muito os tempos de execução da abordagem *MPI* e *híbrida* está relacionado com o *load average* da máquina utilizada, que por ser um servidor com muitos processos a correrem em simultâneo e também por poder haver acesso paralelo dos utilizadores na máquina, pode ter subidas muito significativas, impactando em muito o desempenho da máquina, tornando-se esta muito lenta. É previsível que em condições mais estáveis, os resultados *MPI* e *híbrido* tivessem sido um pouco melhores.

Uma métrica que não foi abordada neste *paper* e que merece ser estudada com algum cuidado é a escalabilidade. O estudo do desempenho das abordagens em função do número de processadores utilizados é um fator decisivo para ajudar a definir a partir de momento é que não vale a pena paralelizar mais, sendo que neste relatório foram sempre utilizados os limites máximos para definir o número de *threads* e processos possíveis.

Isto significa que a partir do momento em que um programa não é propriamente escalável com a adição de mais processadores e continuam a ser adicionadas unidades de processamento ao mesmo, a eficiência desce a pique, o que provavelmente aconteceu para as eficiências obtidas neste trabalho. Os programas desenvolvidos têm, com grande probabilidade, uma escalabilidade já muito baixa, porque na prática a aceleração de um programa não cresce de forma proporcionalmente direta com o número de processadores, tornando-se, por consequência, a eficiência muito baixa.

Outra métrica que é de facto interessante estudar e que não foi abordada neste *paper* é o rácio **R/C**, uma vez que não foram utilizados qualquer tipo de *clusters* para correr os programas, não existindo por isso tempo de comunicação. No entanto, faz todo o sentido de que as abordagens distribuídas, nomeadamente o *MPI* e o *híbrido*, sejam também testadas em *clusters* e não apenas numa só máquina, existindo portanto lugar para estudar em profundidade esta métrica, especialmente para estas duas abordagens.

Bibliografia

[1] En.wikipedia.org. 2021. Parallel computing - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Parallel_computing [Accessed 7 April 2021].

[2] Asanovic, Krste et al. (December 18, 2006). "The Landscape of Parallel Computing Research: A View from Berkeley" (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183.

[3] Super User. 2021. Definition of a processor vs core (multiprocessor vs multicore). [online] Available at: https://superuser.com/questions/1041370/definition-of-a-processor-vs-core-multiprocessor-vs-multicore [Accessed 7 April 2021].

[4] 2021. Avaliação_desempenho. [pdf] Margarida Moura. Available at: https://tutoria.ualg.pt/2020/pluginfile.php/195300/mod_resource/content/0/2.Avalia%C3%A7%C3%A3o_desempenho.pdf [Accessed 7 April 2021].

[5] Curc.readthedocs.io. 2021. Fundamentals of parallel programming — Research ComputingUniversity of Colorado Boulder documentation. [online] Available at: https://curc.readthedocs.io/en/latest/programming/parallel-programming-fundamentals.html [Accessed 8 April 2021].

[6] Blaise Barney, L., 2021. What is MPI?. [online] LLNL HPC Tutorials. Available at: https://hpc-tutorials.llnl.gov/mpi/what_is_mpi/ [Accessed 8 April 2021].

[7]Nrich.maths.org. 2021. An Introduction to Magic Squares. [online] Available at: https://nrich.maths.org/magic-square-intro [Accessed 8 April 2021].

[8]En.wikipedia.org. 2021. POSIX - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/POSIX#POSIX.1-2017 [Accessed 8 April 2021].

[9]OpenMP. 2021. OpenMP FAQ - OpenMP. [online] Available at: https://www.openmp.org/about/openmp-faq/ [Accessed 8 April 2021].

[10] Wiki.mpich.org. 2021. Frequently Asked Questions - Mpich. [online] Available at: https://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions [Accessed 8 April 2021].