4/12/2021

Quadrado Mágico

GUI COSTA Nº61172

Quadrado mágico no âmbito de sistemas paralelos e distribuídos

Universidade do Algarve – Faculdade de Ciências e Tecnologia

Gui Costa Nº 61172

2021

Índice

Tabela de figuras

1 Resumo

Este trabalho, no âmbito das aulas de SPD (Sistemas Paralelos e Distribuídos) e dirigido pela docente Maria Margarida Madeira e Moura, centra-se no desenvolvimento e introdução dos conceitos mais fundamentais e necessários da Computação Paralela e Distribuída.

O trabalho também se concentra em tópicos como análise de desempenho, implementação de métodos e documentação.

No lado da análise do trabalho, examinamos a aceleração e a eficiência, bem como um leve toque em como as comunicações com dispositivos remotos podem influenciar os números, distorcer os dados de maneira errada e levar a conclusões erradas sobre as implementações.

No lado da implementação do trabalho, olhamos como cada implementação funciona e como ela difere de outras implementações, especificamente quando se trata de como elas são codificadas, mas em geral o algoritmo de cálculo do tipo de quadrado é mantido em todas elas.

No lado da documentação do trabalho, esta parte, concentramo-nos em documentar os dados e as implementações, ao mesmo tempo em que explicamos como funcionam a um nível mais alto, bem como explicamos e detalhamos a análise dos dados.

O trabalho também aborda como melhorar os resultados.

2 Abstract

This work, given within the class of PDS (Parallel and distributed sytems) and headed by teacher Maria Margarida Madeira e Moura, focuses on the development and introduction to the most fundamental and necessary concepts of parallel and distributed computing.

The work also focuses on topics such as analysis of performance, implementation of methods and documentation.

On the analysis side of the work, we look in speedup and efficiency as well as a light touch on how communications with remote devices might influence performance numbers and skew data in the wrong way and lead to wrong conclusions on implementations.

On implementation side of the work, we look at how each implementation works and how it differs from other implementations, specifically when it comes to how they are coded, but in general the algorithm for calculation the type of square is maintained throughout them.

On the documentation side of the work, this part, we focus on documenting data and the implementations whilst explaining how they work on a broader level as well as explaining and detailing the analysis of data.

The work also touches upon how to improve the results.

3 Introdução

3.1 Objetivo e Motivação

Este trabalho visa estudar e ensinar as abordagens de paralelização, avaliação de desempenho e documentação do processo bem como a familiarização com as abordagens de implementação sequencial, paralelas (usando pthreads e OpenMP), distribuídas (MPI) e híbridas (OpenMP + MPI) (Tutoria Ualg - SPD Lab1, 2021).

3.2 Metodologia

Para o desenvolvimento das várias abordagens foram posteriormente gerados todos os quadrados mágicos utilizando a ferramenta squareGen.py (em Python) para as seguintes ordens 101, 201, 301, 401, 501, 601, 701 ,801, 901, 1001, ou seja, com saltos de 100 unidades da ordem.

Tendo em conta que o tempo aumenta significativamente a partir 1001 a ordem aumenta por 1000 até 25001. Sendo gerados quadrados perfeitos, imperfeitos e não-mágicos para todas a ordens mencionadas.

Dado o tempo de transferência nas implementações MPI e Híbridas, que é significativamente elevado, foram apenas selecionados os testes entre 101 e 1001 de 100 em 100, e os testes 2001 e 3001, para estas mesmas implementações.

3.3 Principais resultados e conclusões

Uma boa e bem pensada implementação sequencial neste problema do quadrado mágico, resulta em melhores tempos do que implementações rápidas ou pouco desenvolvidas. Especialmente tendo em conta que este problema está seriamente dependente de I/O.

4 Enquadramento

4.1 Quadrado mágico

O quadrado magico é um problema matricial que remota a 190 AC na china (Magic Square History in China, 2021), incorporando vários elementos culturais e mitológicos, sendo relevante como símbolo em distintos trabalhos artísticos. Os métodos usados foram: sequencial, pthreads, OpenMP, por MPI e OpenMP + MPI, todos na linguagem C.

Um quadrado magico toma esse nome quando se garante uma matriz de ordem $n \geq 2$ na qual a soma de cada linha, cada coluna e de cada diagonal tem o mesmo valor. Um quadrado mágico diz-se imperfeito quando apenas as somas das diagonais diferem das somas das colunas ou linhas.

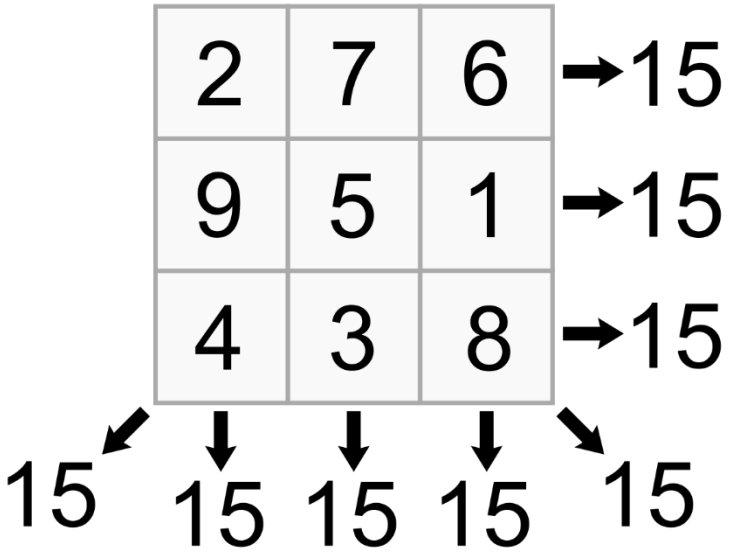

Figura 1 - Exemplo de um quadrado mágico
src: https://en.wikipedia.org/wiki/Magic_square#/media/File:Magicsquareexample.svg

4.2 Pthreads – POSIX Threads

É um modelo de execução paralela que permite a divisão de tarefas em diferentes fluxos chamados de "threads" as quais podem ser controladas e sincronizadas através de chamamentos de funções à API (POSIX Threads, 2020). Como as threads tem a capacidade de trabalhar em simultâneo o tempo de trabalho pode ser dividido (e por consequência reduzido).

4.3 OpenMP – Open Multi-Processing

É um modelo de execução que consiste num conjunto de diretivas, livrarias e variáveis de ambiente que modificam o comportamento de um programa durante a execução, este modelo oferece ferramentas fáceis e flexíveis para desenvolver aplicações em paralelo para vários tipos diferentes de máquinas (OpenMP, 2020). Traz os mesmos benefícios que o modelo anterior, mas com uma maior facilidade de uso. O modelo interpreta automaticamente vários componentes que no modelo anterior teriam de ser adaptados manualmente, relativamente a uma implementação sequencial.

4.4 MPI – Message Passing Interface

É um standard de transferência de mensagens entre processos desenhado para funcionar numa variedade de arquiteturas de processamento paralelo, este standard define a sintaxe e semânticas para esse efeito. Permite comunicação entre duas ou várias máquinas em simultâneo. Permitindo a sincronização e comunicação de funcionalidades em vários processos (Message Passing Interface, 2020). A implementação de MPI usada neste relatório foi OpenMPI.

4.5 Dados analisados

Para efeitos de estudo, foram analisados os tempos das várias implementações, bem como os conceitos de:

- Aceleração – definida como a razão entre o tempo de execução dum problema num único processador e o tempo necessário na resolução desse mesmo problema em processadores idênticos (Aula 2 Avaliação desempenho, 2020);

$$SpeedUp = \frac{Tempo\ médio\ do\ melhor\ sequencial}{Tempo\ médio\ em\ estudo}$$

- Eficiência – Permite caracterizar a forma como um algoritmo dispõe dos recursos computacionais, não sendo diretamente dependente da dimensão do sistema. Define-se eficiência como sendo a fração de tempo que os processadores realizam trabalho útil, ou seja, o quociente entre a aceleração e o número de processadores (Aula 2 Avaliação desempenho, 2020);

$$E = \frac{SpeedUp}{Processors}$$

5 Estudo de casos

5.1 Ambiente de teste

Como este trabalho centra-se no estudo de desempenho, este depende consideravelmente da plataforma (inclusive a capacidade do sistema) no qual foi realizado e testado.

A plataforma experimental usada foi um computador fixo com as seguintes especificações:

- Windows 10 Pro x64;
- Processador AMD Ryzen 7 5800X 8-Core, relógio base de 3.8Ghz com boost até 4.7GHz com 16 threads;
- L1d 8x32 KBytes;
- L1i 8x32KBytes;
- L2 Cache 8x512KBytes;
- L3 Cache 32MBytes;
- 16 GBytes 3000MHz de RAM.

Relativamente aos testes MPI e Híbrido, a segunda máquina foi a disponibilizada pela FCT, e tem as seguintes especificações:

- Architecture: x86_64;
- CPU(s): 4;
- Thread(s) per core: 1;
- Model name: Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz;
- L1d cache: 32K;
- L1i cache: 32K;
- L2 cache: 1024K;
- L3 cache: 28160K;
- Debian 4.19.118-2+deb10u1 (2020-06-07) x86_64 GNU/Linux;
- 8 GBytes RAM.

5.1.2 Processamento de Dados

Os valores correspondentes às entradas da matriz são lidos de um ficheiro com apenas uma linha, o número de elementos a ler é obtido a partir do nome do ficheiro que contem a ordem da matriz, este elevado á segunda potencia devolve-nos o número de elementos a ler. Para ser o mais eficiente a nível da memoria quanto possível, para se poder aplicar testes maiores) a decisão tomada para a leitura foi nesse sentido, o ficheiro é lido e em simultâneo são feitas as somas para as linhas e colunas. Deste forma não existe a limitação em termos de memorio pois não existe a necessidade de guardar os dados numa matriz em RAM.

Da mesma forma, isto permite uma execução muito mais rápida do algoritmo visto que apenas se necessita de um único ciclo para percorrer o ficheiro e um único ciclo para percorrer as somas das colunas no final.

O algoritmo em geral para verificar se um quadrado é magico ou imperfeito funciona da seguinte forma:

1. Ler um número da matriz;
2. Adicionar o número lido a soma da linha a que pertence;
3. Adicionar o número lido a soma da coluna a que pertence;
4. Se pertencer a alguma diagonal, adicionar as diagonais a que pertence.

Depois de somados os valores, são verificados para ver se consistem entre si, ou seja, se todas as linhas são iguais a todas as colunas e às diagonais. Caso apenas as diagonais sejam diferentes, então o quadrado é imperfeito, caso alguma coluna ou linha difira do esperado, então não é quadrado magico.

5.1.3 Casos de teste

Para se aplicar vários testes ao programa, foi criado um pequeno programa em python que gera os quadrados dos vários tipos, a partir de um número introduzido, de preferência ímpar. Este programa gera 3 ficheiros, um para cada tipo, com a matriz da ordem indicada.

Para a realização deste trabalho e analise de resultado foi também criado um script Powershell que percorre esta lista de matrizes e executa o programa (independente do método) 33 vezes para cada tipo e 3 tipos para cada tamanho (no total 99 vezes para cada tamanho) e guarda os tempos.

Foi recriado o mesmo script em bash para correr os testes relativos ao MPI e Híbrido (OpenMP + OpenMPI).

Para todos os métodos foram aplicados testes para os 3 tipos de matrizes com as seguintes ordens:

1. 101 a 1001 de 100 em 100;
2. 1001 a 25001 de 1000 em 1000 ou até 3001 para o caso do MPI e Híbrido.

Os dados foram posteriormente exportados para Excel e lá calculados os desvios padrão e medias, bem como todos os outros cálculos referidos no ponto 4.5.

5.2 Sequencial

O problema do quadrado mágico mostra-se simples de resolver quando se pretende apenas determinar o seu tipo, necessita-se apenas de somar todos os elementos de cada linha, linha a linha e verificar se os valores correspondem, análogo as colunas e diagonais. No método sequencial isto foi implementado da seguinte forma:

O ficheiro é recebido como argumento e a ordem obtida a partir do nome, com este valore é preparado o número de dados a ler, a medida que se lê os dados as somas são feitas de seguida, ou seja, lê-se um número, soma-se à linha, a coluna a que corresponde e as diagonais se a alguma pertencer. Quando se acaba de somar uma linha verifica-se se é consistente com as linhas anteriores, se não for, para-se o programa e determina-se que o quadrado é não-mágico, caso sejam todas as linhas consistentes, verifica-se o mesmo para as colunas. No final verificam-se as diagonais para certificar que é completamente magico ou imperfeito.

Deste forma é possível verificar o quadrado sem guardar nenhum dos valores da matriz, apenas guarda-se uma soma da primeira linha, a soma das colunas e a soma das diagonais, permitindo uma implementação bastante rápida.

Outras alternativas, embora menos eficientes, seriam ler primeiro e depois fazer os cálculos ou até ler aos bocados e fazer os cálculos aos bocados.

5.3 pThreads

O primeiro método para a divisão do trabalho em tarefas foi com uso de pthreads, separou-se a matriz em intervalos correspondentes ao número de threads disponibilizados, ou seja, se testarmos com 4 threads, separamos a matriz em 4 ou menos intervalos.

12	31	12	1	30	24
132	135	61	2	31	51
252	239	110	3	32	78
372	343	159	4	33	105
492	447	208	5	34	132
612	551	257	6	35	159

Figura 2 - Exemplo da divisão da matriz

Como se pode verificar a matriz e dividida em 3 intervalos iguais e como não sobra nenhuma linha para a última thread, apenas são criadas 3.

Os tamanhos dos intervalos são obtidos das seguintes formas:

$$ceil\left(\frac{ordem}{numero\ de\ threads}\right)$$

Com estes intervalos definidos são lidas as correspondentes linhas e iniciada uma thread para operar o algoritmo de quadrado magico sobre essas linhas. Caso uma das thread detete falha no quadrado perfeito, esta faz a mudança de uma variável global que vai parar as outras threads de executarem.

Para este caso foram usadas 16 threads.

5.4 OpenMP

Para o OpenMP a implementação é idêntica à POSIX Threads, exceto que são lidos primeiro os dados e apenas depois criadas as threads, os intervalos são também calculados para o número inteiro imediatamente a baixo, e a última thread fica com as colunas que restarem da divisão.

A necessidade de ler tudo primeiro aparece pois não existe tanto controlo sobre quando se inicia uma thread, visto que isto é um processo automatizado pelo OpenMP.

Para este caso foram usadas 8 threads.

5.5 MPI

Para o MPI, o problema mantém-se o mesmo, muda-se apenas o método que, em vez de se dividir as tarefas por threads, divide-se por processos que podem correr em máquinas separadas, que foi o testado neste estudo. Como a segunda máquina é uma máquina remota, os tempos das execuções vão conter um tempo adicional oriundo da comunicação entre processos.

5.6 MPI + OpenMP (Híbrido)

Como o nome sugere, esta implementação usa os princípios das implementações em 5.5 e 5.4. Em cada máquina as tarefas são divididas uma segunda vez por entre threads, ou seja, existe primeiro uma divisão entre duas máquinas (processos) e depois dentro de cada máquina existe outra divisão entre threads.

5.7 Ferramentas utilizadas

Para alem da ferramenta para a criação de casos de teste (squareGen.py), foram utilizadas outras ferramentas externas, nomeadamente Excel para analise de resultados, bem como uma ferramenta para transposição de dados em formato TXT para Excel criada por Diogo Faustino e chamada Data.py.

Todas as ferramentas e dados mencionados estão disponibilizados como anexo a este documento, as inputs devem ser geradas com o uso da ferramenta mencionada.

6 Analise de resultados e discussão

6.1 Tempos médios

A ordem dos dados apresentados segue sempre a mesma ordem (ordem alfabética), primeiro os imperfeitos “i”, segundo os perfeitos “p” e os não mágicos “r”.

A implementação sequencial é a implementação que é usada como base de comparação para as outras, isto é, através dos tempos da implementação sequencial, obtém-se as melhorias (ou pioria) das restantes implementações.

Devido á falta de implementações sequencias recebidas, toma-se a implementação desenvolvida no âmbito deste trabalho como a melhor implementação sequencial conhecida.

Tendo em conta paralelização de I/O não era no âmbito deste trabalho, acredita-se que esta implementação foi a que melhor resultado deu. O código desenvolvido foi compilado com a opção de otimização “-O3” que ativa uma serie de flags de otimização[1].

Os tempos resultantes desta implementação resultam nos seguintes gráficos em escala logarítmica (ordem por tempo).

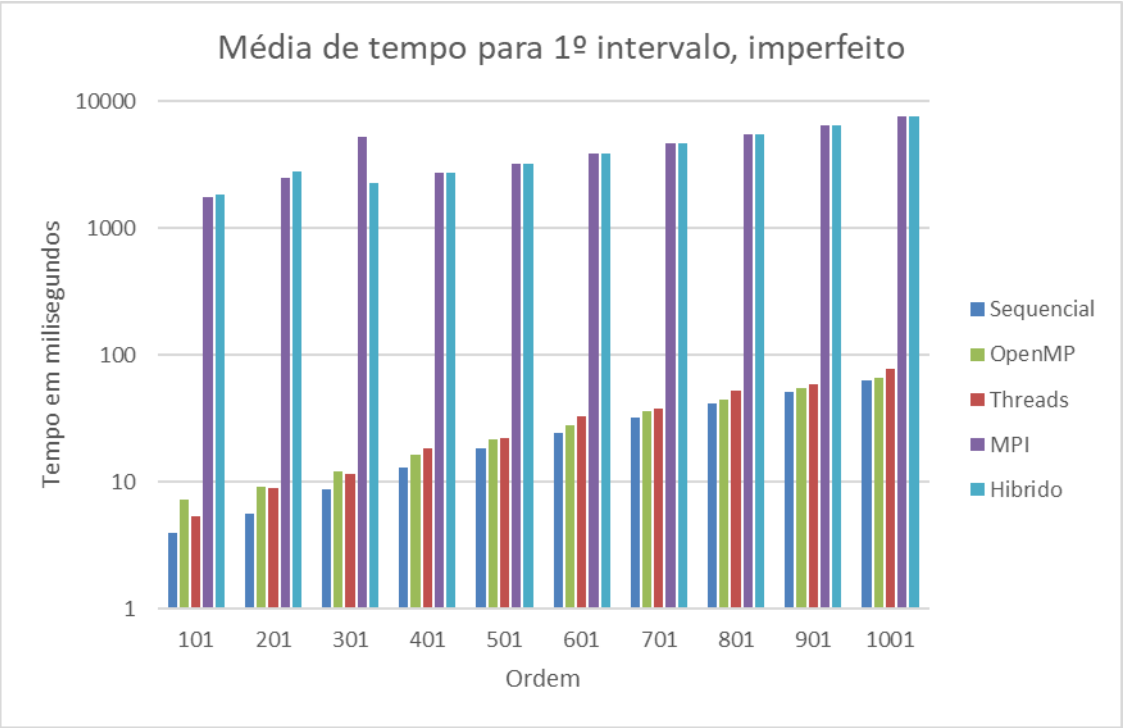

Figura 3 – Tabela I1

[1] Ver: Optimize Options (Using the GNU Compiler Collection (GCC))

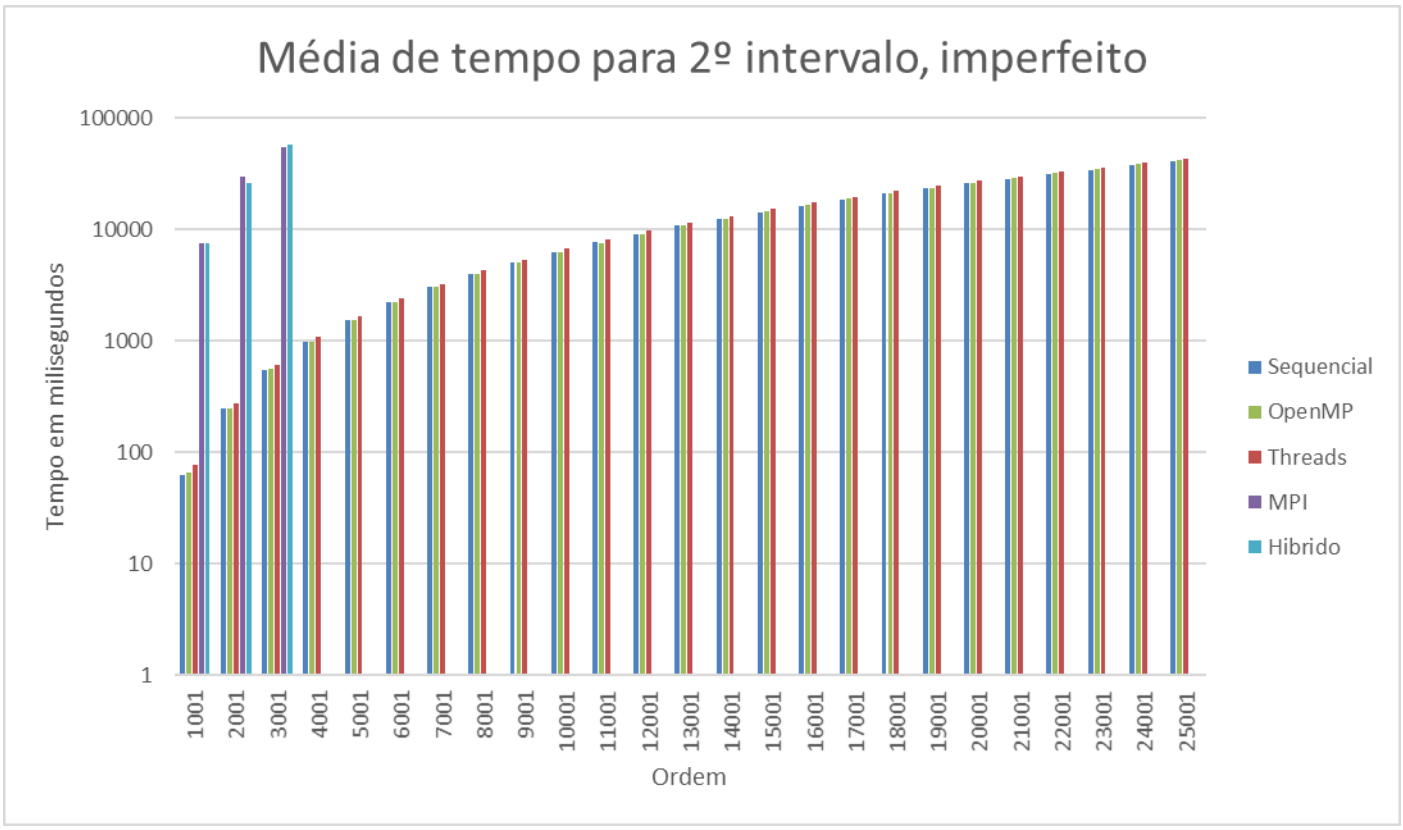

Figura 4 - Gráfico I2

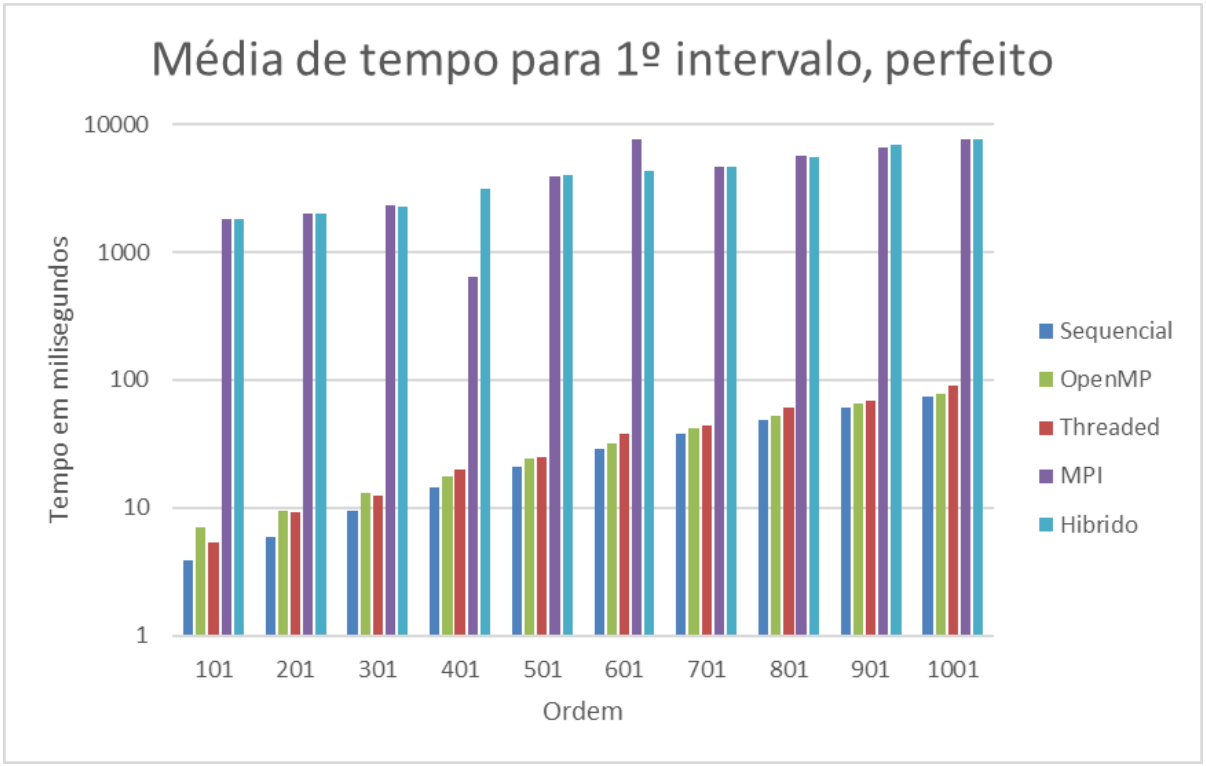

Figura 5 - Gráfico P1

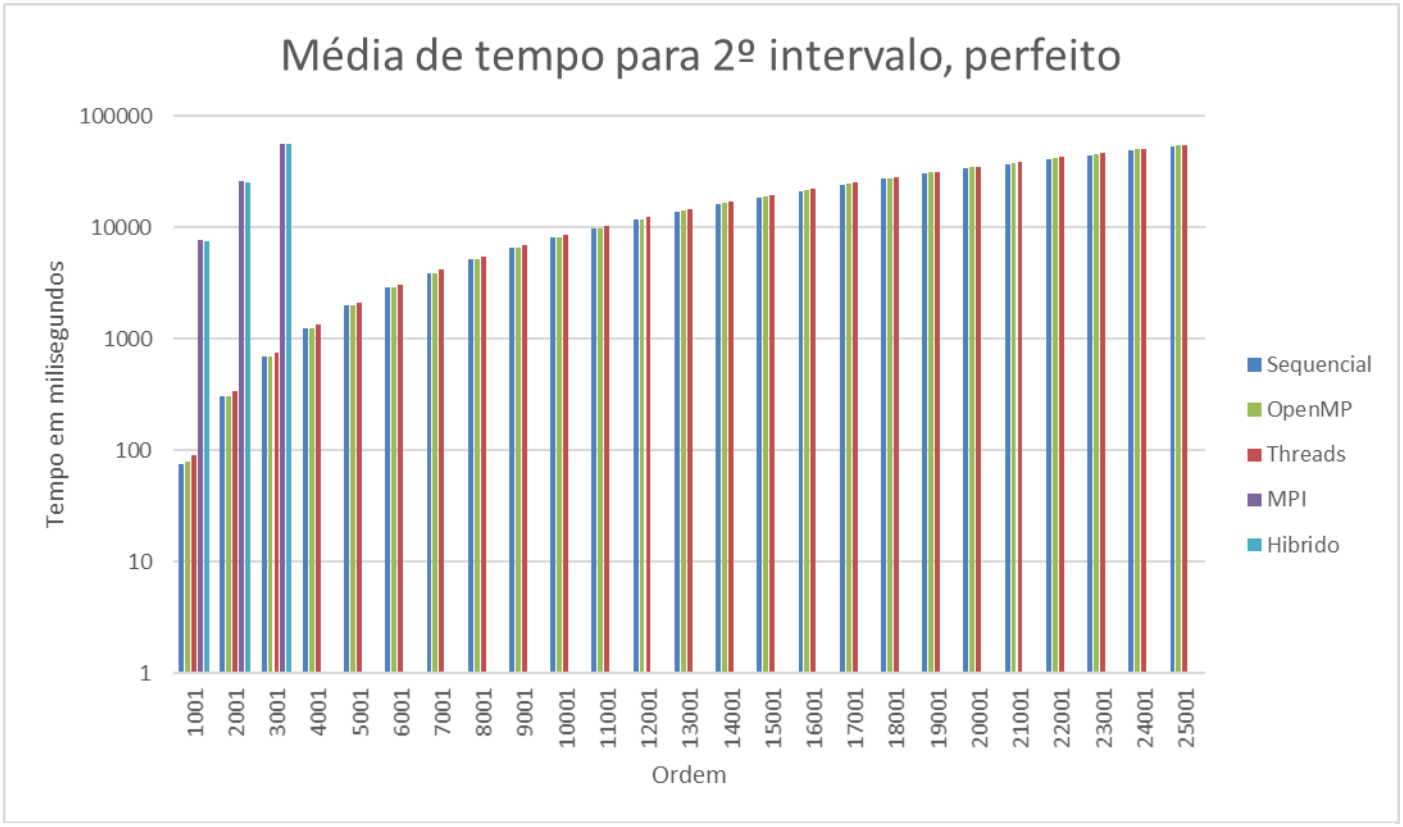

Figura 6 - Gráfico P2

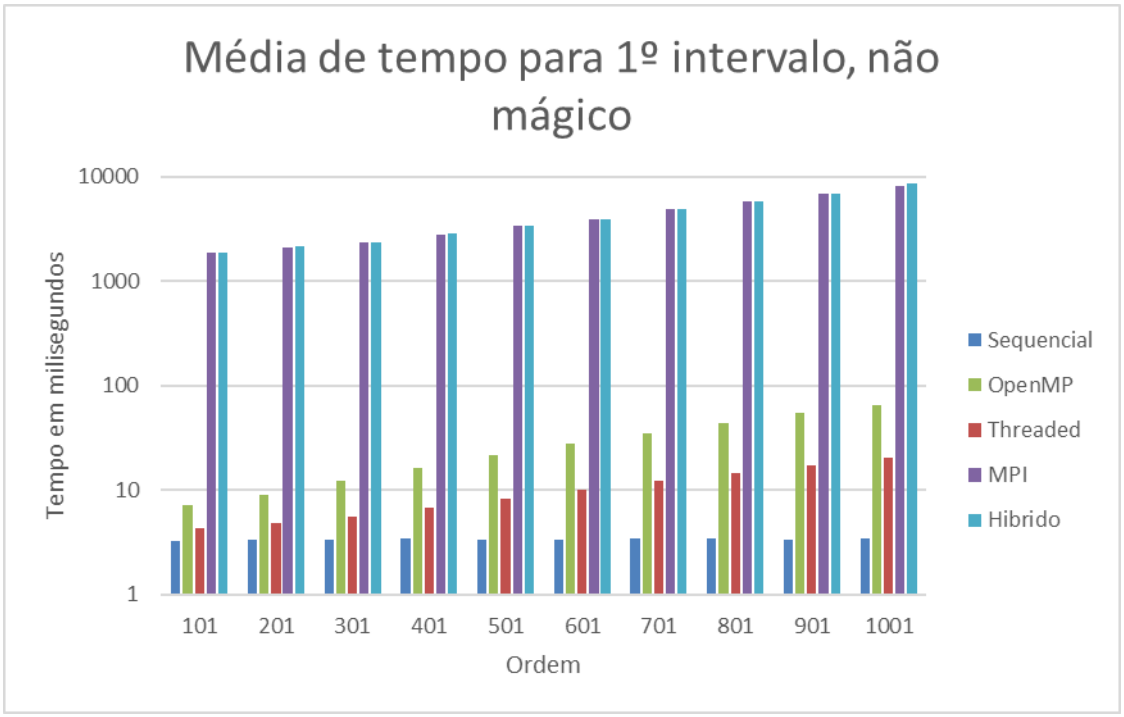

Figura 7 - Gráfico R1

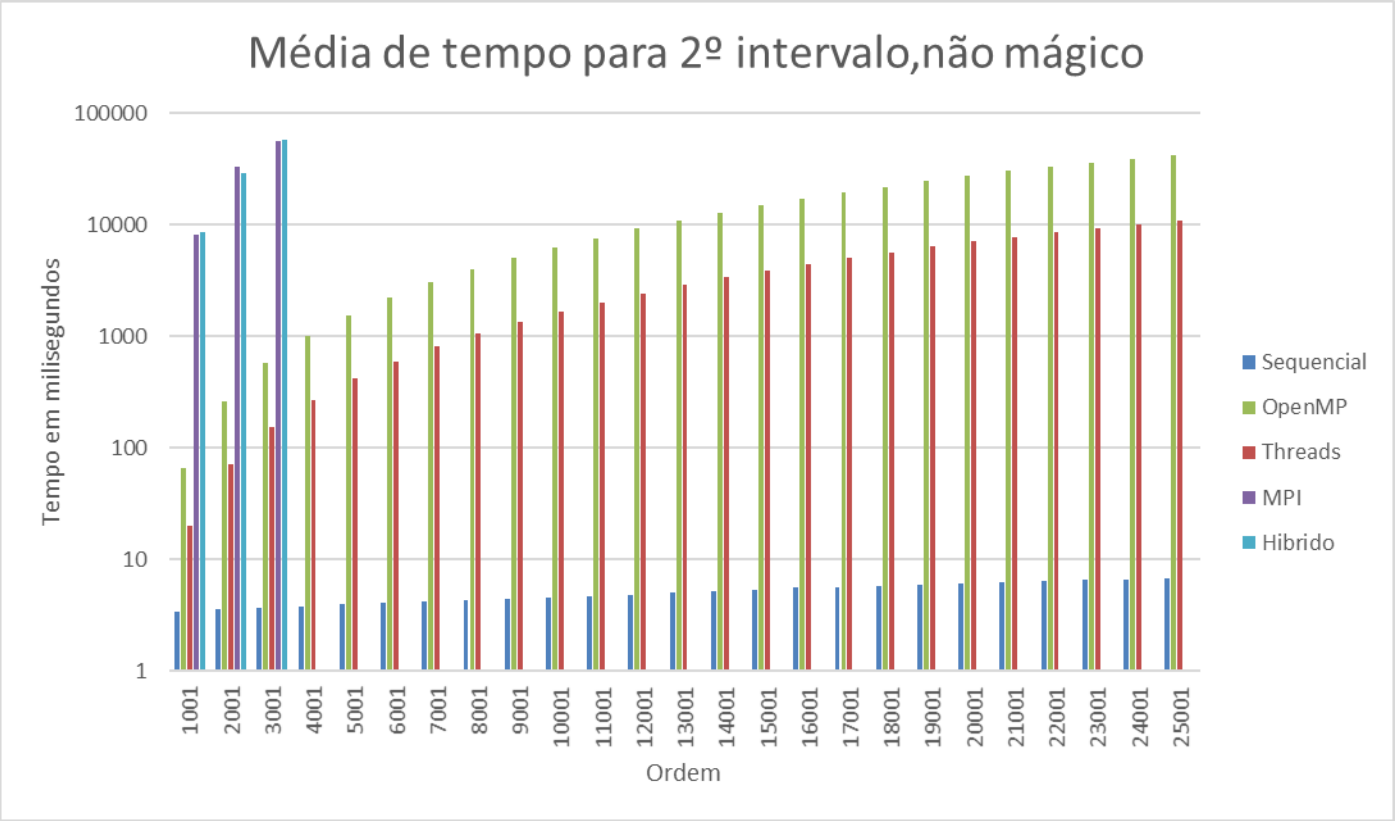

Figura 8 - Gráfico R2

Observa-se nos gráficos I1, I2 e P1 e P2 tempos consistentes com o esperado (assumindo que a implementação sequencial é a melhor possível), os tempos de threaded e OpenMP mantem-se a par com a implementação sequencial com poucas diferenças.

Quando se olha para os últimos dois gráficos nota-se já uma grande diferença entre os tempos sequenciais e os outros, isto deve-se ao facto de que as restantes implementações necessitam de ler o ficheiro por completo ou pelo menos parcialmente. Como o quadrado magico é um problema que é bastante dependente da I/O maior parte do tempo de operação recai sobre a leitura do ficheiro.

No caso do quadrado não magico, como as implementações (exceto a sequencial) requerem leitura completa, os tempos são bastante elevados relativamente ao sequencial que, como lê e calcula, assim que lê uma linha errada termina a sua execução, luxo o qual não existe nas threads pois tem pelo menos de ler $\frac{1}{16}$ da matriz. O mesmo que se aplica as threads aplica-se ao OpenMP exceto que a matriz tem mesmo toda de ser lida, que não é verdade para threads.

6.2 Aceleração

Devido ao discutido no ponto anterior, será de esperar que aceleração, que idealmente tenderia para p (número de processadores), tenda para um número < 1 visto que a melhor implementação foi a sequencial. Veja-se então:

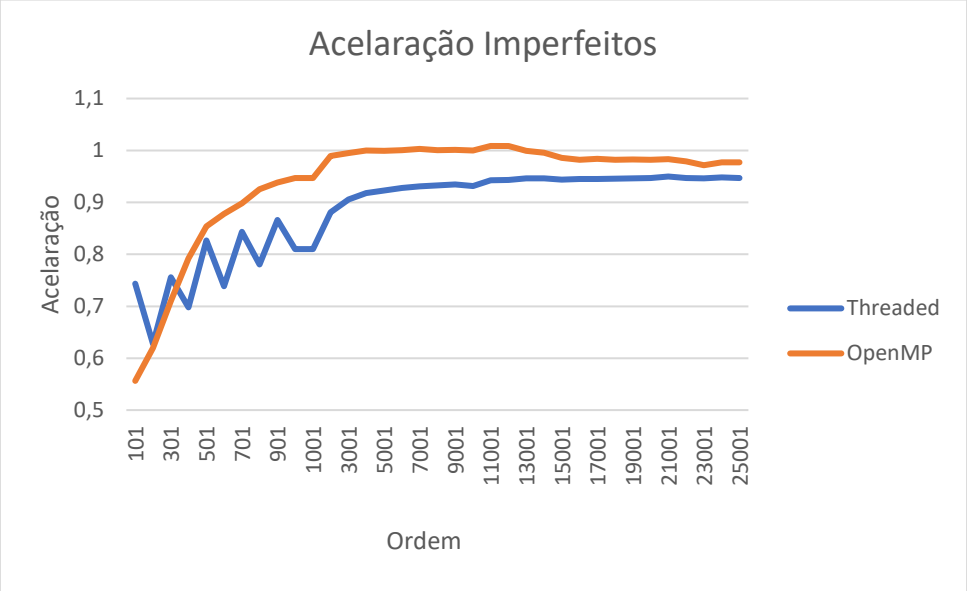

Figura 9 - Aceleração imperfeitos parte 1

Como podemos ver, apesar de o OpenMP ler a matriz toda e usar menos 8 threads, continua a ser melhor escolha que a implementação POSIX Threads.

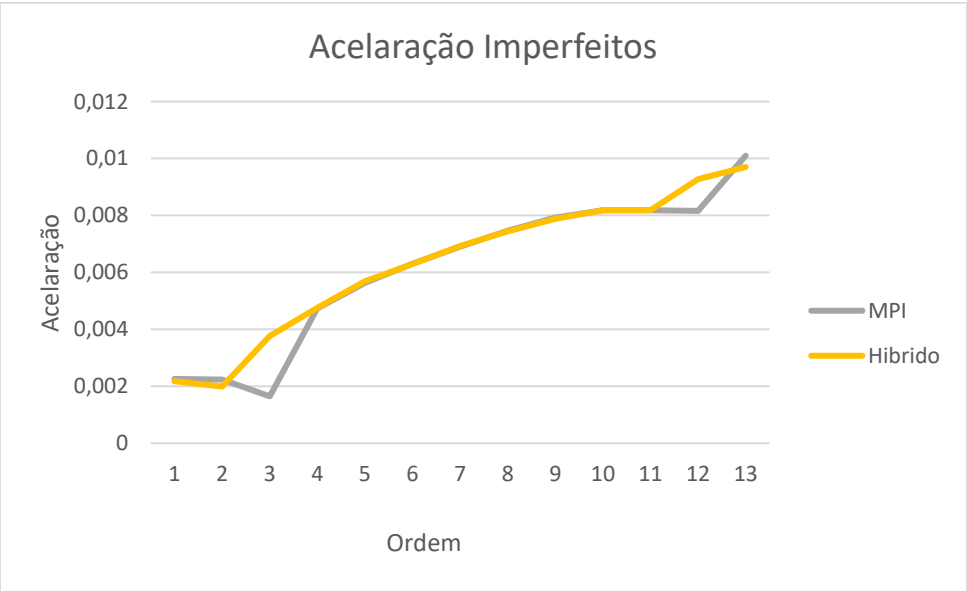

Figura 10 - Aceleração imperfeitos parte 2

Como podemos ver, as implementações que foram corridas em conjunto com a segunda máquina (especificada em 5.1 Ambiente de teste) que é remota, a aceleração é muito baixa, visto que o tempo de comunicação é bastante elevado. Por esta razão a aceleração é bastante baixa. Se apenas fosse considerado o tempo de cálculos e leitura, então certamente teríamos valores de aceleração mais aproximados aos anteriores.

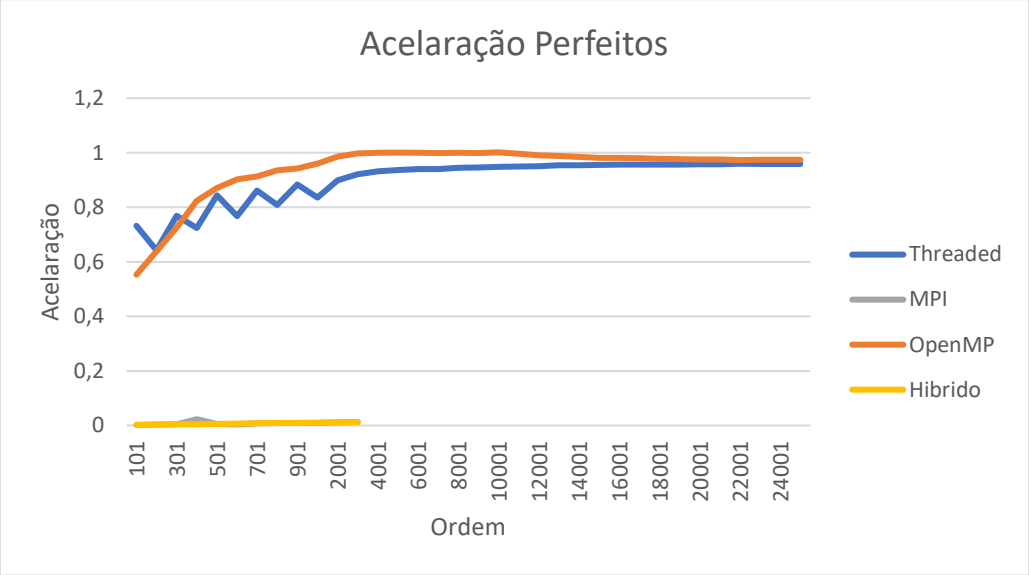

Figura 11 - Aceleração Perfeitos

Verificamos novamente que a aceleração se mantém < 1 a favor da implementação OpenMP e o Híbrido e MPI muito abaixo do esperado pelas mesmas razoes mencionadas anteriormente.

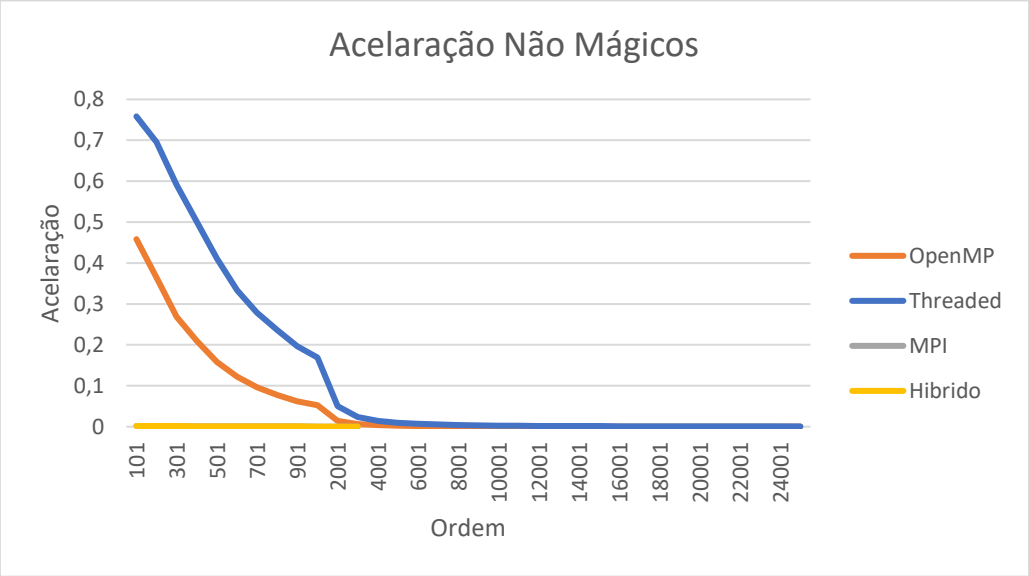

Figura 12 - Aceleração Não Mágicos

Como se verifica, nos quadrados não mágicos é que se vê bastante a diferença nas implementações, como o sequencial consegue parar de ler assim que deteta a falha nos quadrados mágicos os tempos são sempre bastante baixos (para os casos testados, sempre inferiores a 10s), ao contrário das restantes implementações que tem sempre de ler pelo menos uma divisão da matriz, tornando-se menos eficazes com o aumento da ordem, especialmente para estes casos.

6.3 Eficiência

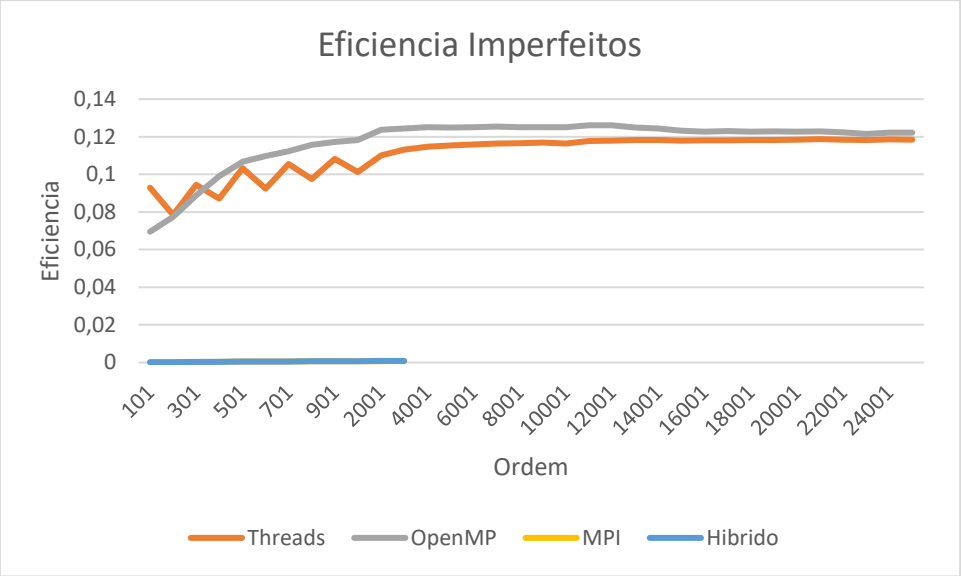

Figura 13 - Eficiência para quadrados imperfeitos

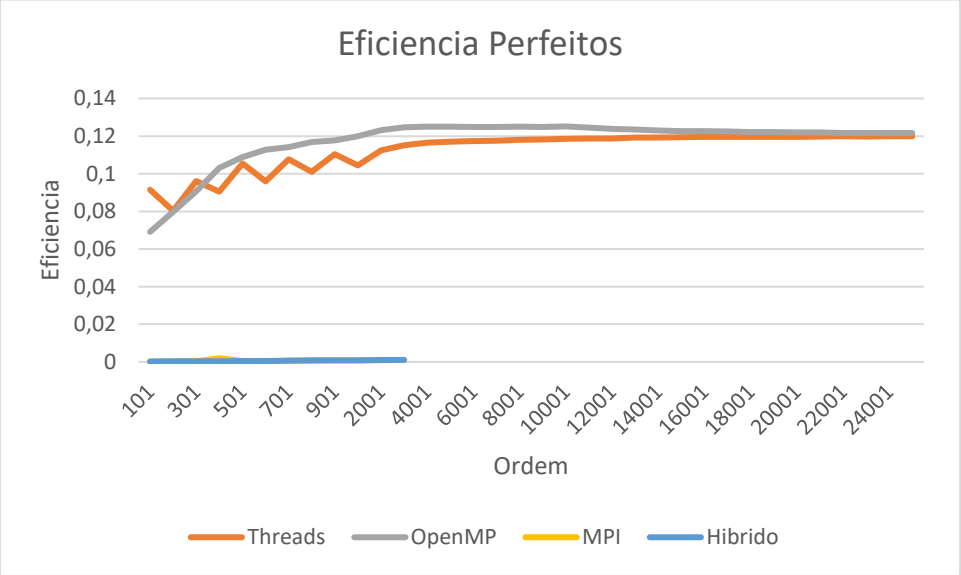

Figura 14 - Eficiência para quadrados perfeitos

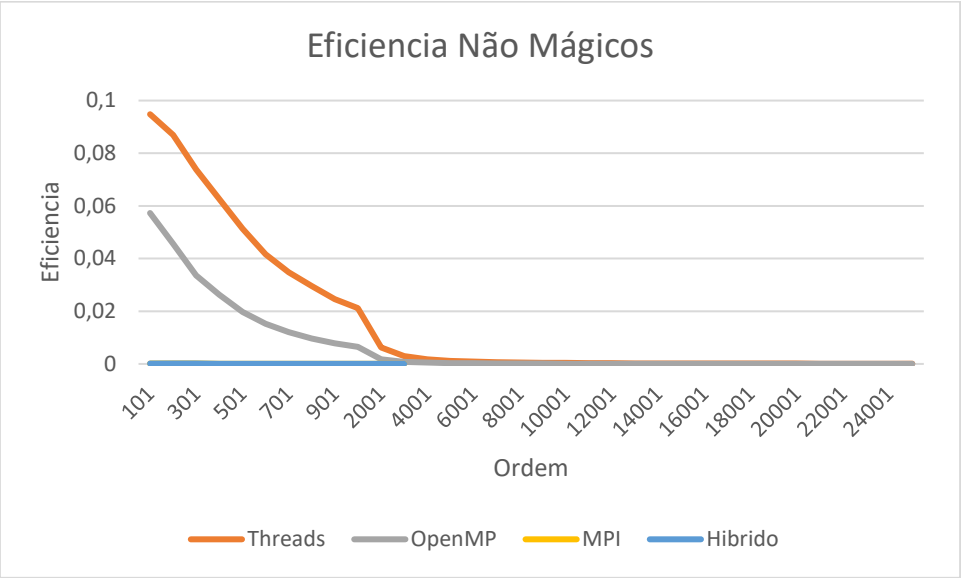

Figura 15 - Eficiência para quadrados não mágicos

Tendo em conta os resultados da aceleração, quando olhamos para os resultados da eficiência conseguimos perceber a razão destes valores, como nenhum dos valores foi significativamente acima de 1, a eficiência para os 8 núcleos disponibilizados vai ser bastante baixa. Não há muito a dizer, apenas que estes valores se devem as mesmas razoes mencionadas na aceleração.

Os dados mais detalhados podem ser vistos nas tabelas fornecidas em anexo, em formato Excel.

7 Comentários finais

Como se pode verificar nos pontos do capítulo 6 desde documento, os testes a quadrados mágicos (perfeitos) são os mais interessantes de analisar pois permitem realmente descobrir, pelo menos em tempos de calculo, quais das implementações são as melhores.

Se observarmos a Figura 5 - Gráfico P1 e Figura 6 - Gráfico P2, vemos que os tempos médios para as implementações sequencial, threaded e OpenMP são bastante semelhantes. Como estamos a olhar para os testes perfeitos, a leitura tem de ser feita por completo em todos os casos para se poder confirmar isto, logo a diferença caberá no tempo de calculo ou no chamamento a funções e outros processos não diretamente relacionados com a soma de valores.

Na implementação OpenMP vê-se uma melhoria relativamente a threaded, isto pode dever-se ao facto de existirem standards mais específicos para que uma determinada implementação OpenMP seja valida, e também porque a criação e interrupção de threads pode ser decidida pelo compilador, enquanto no pthreads requer um pensamento adicional sobre como criar (pthread_create) e esperar o término (pthread_join) de threads.

Uma maneira de melhorar estes resultados em geral teria sido melhorar a leitura de dados para ser algo mais semelhante a implementação sequencial, desta forma teríamos tempos mais aproximados ou até melhores. Como paralelização de I/O não estava no quadro de estudo deste trabalho, foi decidido que não seria feita. Adicionalmente, na máquina principal em que foram testadas as implementações, apenas existe um disco e a leitura múltipla do mesmo ficheiro em várias threads resultaria em performance semelhante á sequencial pois o disco (e dependendo do File System) não teria necessariamente a capacidade de ler várias partes do ficheiro em simultâneo.

Outra maneira seria ter usado o mesmo número de threads em OpenMP que na versão threaded e o equivalente para MPI e Híbrido.

Nas implantações MPI e Híbridas, os tempos são bastante piores que nas outras implementações devido ao tempo de comunicação entre a primeira máquina e a máquina remota, se estes testes tivessem sido executados em máquinas locais (separadas mas na mesma rede) seria de esperar muito melhores tempos e praticamente em linha com as outras implementações.

8 Bibliografia e Documentação

Aula 2 Avaliação desempenho. (14 de 04 de 2020). Obtido de
https://tutoria.ualg.pt/2020/pluginfile.php/195300/mod_resource/content/0/2.Avalia%C3%A7%C3%A3o_desempenho.pdf

Magic Square History in China. (2021, 03 1). Retrieved 03 2021, from Wikepedia:
https://en.wikipedia.org/wiki/Magic_square#China

Message Passing Interface. (14 de 04 de 2020). Obtido de Wikipedia:
https://en.wikipedia.org/wiki/Message_Passing_Interface

OpenMP. (14 de 04 de 2020). Obtido de Wikipedia: https://en.wikipedia.org/wiki/OpenMP

POSIX Threads. (14 de 04 de 2020). Obtido de Wikepedia:
https://en.wikipedia.org/wiki/POSIX_Threads

Tutoria Ualg - SPD Lab1. (02 de 03 de 2021). Obtido de Tutoria Ualg:
https://tutoria.ualg.pt/2020/mod/resource/view.php?id=96369

Documentação

Open MPI -Open Source High Performance Computing:
https://www.open-mpi.org/

OpenMP:
https://www.openmp.org/