Exercício Programa II - Relatório

MAC0219 - Programação Concorrente e Paralela

Prof.: Alfredo Goldman

Monitor: Diogo Pina

David de Barros Tadokoro
NºUSP: 10300507
Fernando Henrique Junqueira
NºUSP: 11795888
Guilherme Moreno Silva
NºUSP: 10843437
Julho de 2022

1 Sobre a Entrega

O diretório onde se encontra o arquivo `mac0219_ep2_10300507-10843437-11795888.pdf` (este relatório) deve ter a seguinte hierarquia de arquivos:

```
mac0219_ep2_10300507-10843437-11795888/
├─ src/
│  ├─ Makefile
│  ├─ mandelbrot_seq.c
│  ├─ mandelbrot_pth.c
│  ├─ mandelbrot_omp.c
│  ├─ mandelbrot_ompi.c
│  ├─ mandelbrot_ompi_pth.c
│  ├─ mandelbrot_ompi_omp.c
│  └─ csv_plot.m
├─ data/
│  ├─ comparison/
│  ├─ graphs/
│  ├─ mandelbrot_ompi/
│  ├─ mandelbrot_ompi_pth/
│  └─ mandelbrot_ompi_omp/
├─ mac0219_ep2_10300507-10843437-11795888.pdf
└─ README.md
```

O diretório `src/` contém os códigos fontes das implementações pedidas pelo enunciado. Além das implementações realizadas no primeiro exercício-programa, três novas implementações foram incluídas, todas fazendo uso de OpenMPI e sendo duas delas paralelizadas, uma com Pthreads e outra com OpenMP.

No diretório `data/`, os subdiretórios `mandelbrot_ompi`, `mandelbrot_ompi_pth` e também `mandelbrot_ompi_omp` contém os dados resultantes dos experimentos para a determinação dos parâmetros (Tarefa 2). Estes subdiretórios contém os arquivos `.csv` referentes aos gráficos gerados para a análise (Tarefa 3). O diretório `data/graphs/` contém todos os gráficos construídos e que serão expostos neste relatório. Por fim, o subdiretório `comparison/` possui os outputs puros dos comandos `perf stat` executados para cada implementação (com seus respectivos parâmetros ótimos) para gerarmos seus intervalos de confiança a fim de compará-los (Tarefa 4).

2 Metodologia

Após termos confirmado que as novas implementações estavam corretas, passamos a realização dos experimentos e a coleta de dados.

Antes de descrever a metodologia, iremos expor o ambiente computacional (i.e. a máquina) onde estes experimentos foram realizados:

- CPU: AMD Ryzen 5 1500X Quad-Core:
  - Cores: 4 físicos (8 núcleos lógicos)
  - Frequência do Clock: 3.6GHz
  - Índice BogoMIPS: 6985.90
  - Caches: L1 64K; L2 512K; L3 8192K
- Memória Principal: 8GB RAM DDR4
- Sistema Operacional: Ubuntu 18.04 LTS

Para gerar os arquivos `.csv`, adaptamos o *script* em Bash `run_measurements.sh` fornecido junto ao enunciado do exercício-programa anterior. Alteramos a quantidade de iterações, bem como os valores das mesmas, de acordo com nossas regiões de interesse, seja para o número de processos do OpenMPI ou também para o número de *threads*, quando aplicável. Manipulamos as informações devolvidas pelo comando `perf` usando os programas `sed` e `grep` para filtrarmos as informações de interesse no formato desejado (`.csv`). O resultado foram três *scripts*, uma para cada nova variação.

(**Obs.:** Na seção 3, a estrutura destes dados, bem como a dos gráficos, é exposta em detalhes.)

Em nossos testes, cada comando `perf` realizou 15 execuções por dado, isto é, para cada configuração considerada em nossos gráficos (ou linha no `.csv`), tivemos um intervalo de confiança correspondente gerado pelo `perf`. Todos os experimentos foram feitos contemplando a região do *Triple Spiral Valley* com um entrada com tamanho de 4096.

Em relação as Tarefa 2 e 3, que consistiam na coleta e análise de dados para a determinação dos parâmetros, foi necessária a definição das regiões de interesse. Para o OpenMPI puro, foram feitas 5 coletas de dados, variando apenas o número de nós de 1 até o dobro número de núcleos lógicos, ou seja, $\{1, 2, 4, 8, 16\}$. Já para as versões do OpenMPI com Pthreads e OpenMP, também variamos o números de *threads* na mesma escala, isto é, com número de *threads* entre $\{1, 2, 4, 8, 16\}$, resultando num total de 25 dados coletados para ambos os casos.

Estes dados foram coletados no formato `.csv`. Utilizamos um *script* na linguagem `Octave` para gerarmos os gráficos para a análise.

Sobre a Tarefa 4, tendo determinado os parâmetros ótimos de execução das implementações OpenMPI, pudemos realizar apenas uma execução do `perf` para cada uma das seis implementações com seus parâmetros ótimos. Isto nos permitiu gerar intervalos de confianças para compararmos a performance de cada implementação. Vale lembrar que, pelo primeiro exercício-programa, as implementações sequencial, com Pthreads e com OpenMP já tinham seus parâmetros ótimos determinados.

3 Estrutura dos dados e dos gráficos

Para os 3 arquivos `.csv` gerados (referentes as Tarefas 2 e 3), temos a seguinte estrutura:

- Coluna 1: Número de threads
- Coluna 2: Número de nós
- Coluna 3: Tempo médio de execução em segundos (`perf`)
- Coluna 4: Desvio padrão do tempo de execução em segundos (`perf`)
- Coluna 5: Desvio padrão do tempo de execução em porcentagem (`perf`)

Obs.: O `.csv` da implementação com OpenMPI pura não possui a primeira coluna, uma vez que não variamos o número de *threads* neste caso.

É importante frisar como cada linha destes arquivos descreve um intervalo de confiança (informação que não foi embutida nos gráficos pela legibilidade).

Sobre os gráficos, temos um referente ao OpenMPI puro, em que temos apenas uma linha, e dois referentes ao OpenMPI com Pthreads e o OpenMPI com OpenMP, em que temos cinco linhas (para cada valor de *thread*) para cada um.

4 Resultados

Nesta seção, iremos expor os resultados, isto é, os gráficos que nos possibilitarão inferir sobre os efeitos de paralelizarmos ou não um código.

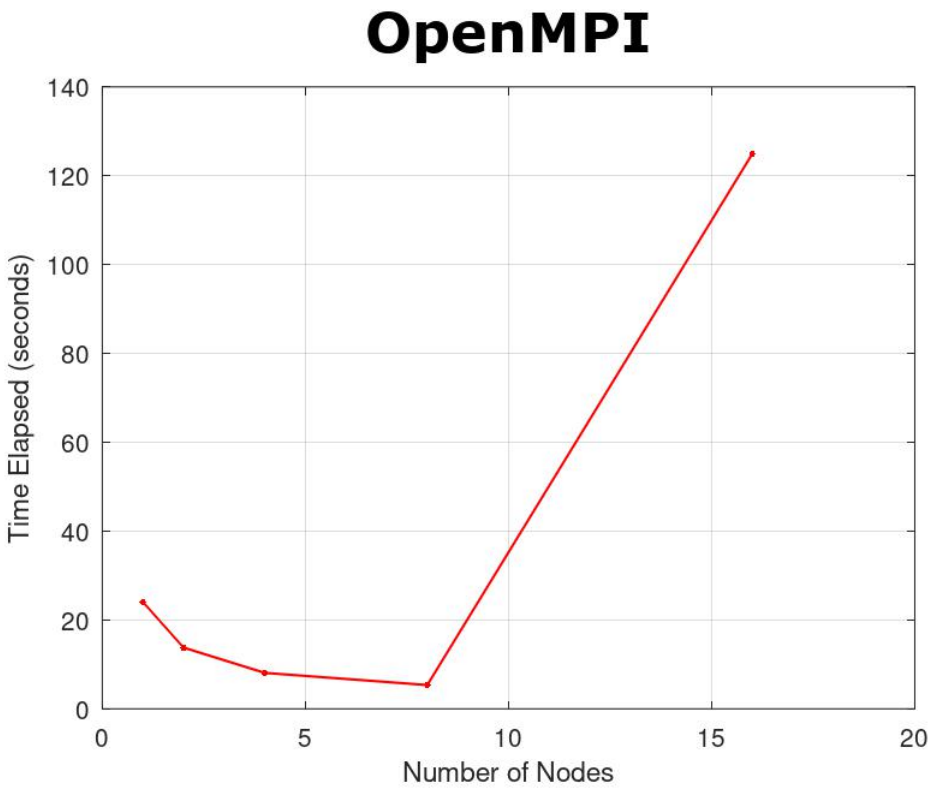

Fig. 1: Implementação com OpenMPI puro.

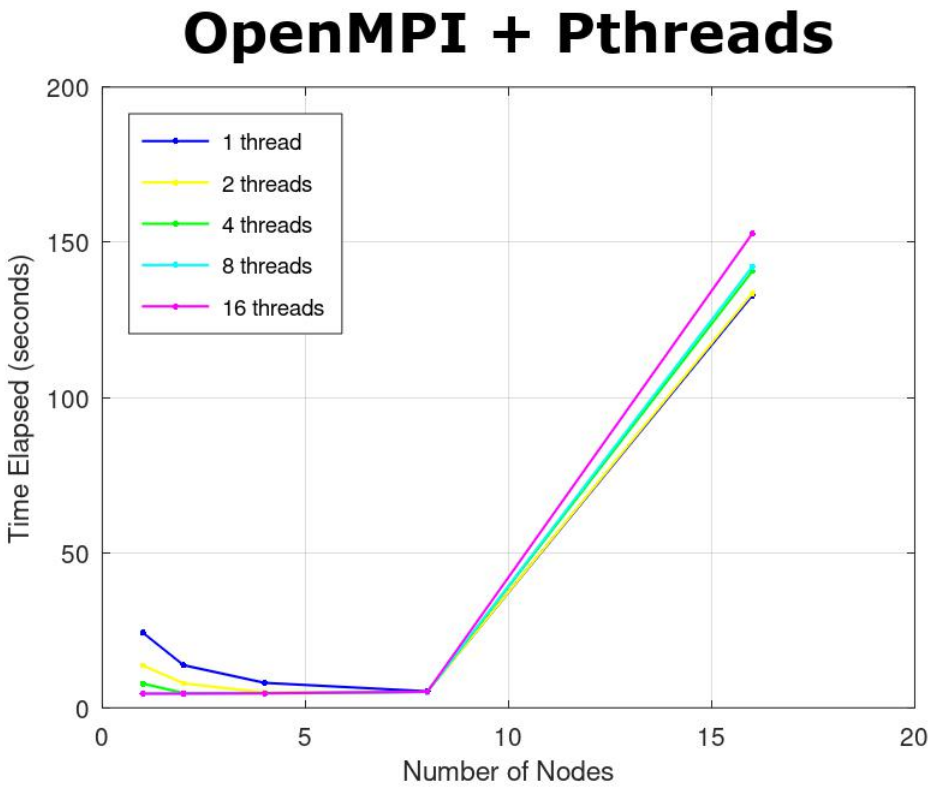

Fig. 2: Implementação com OpenMPI mais Pthreads.

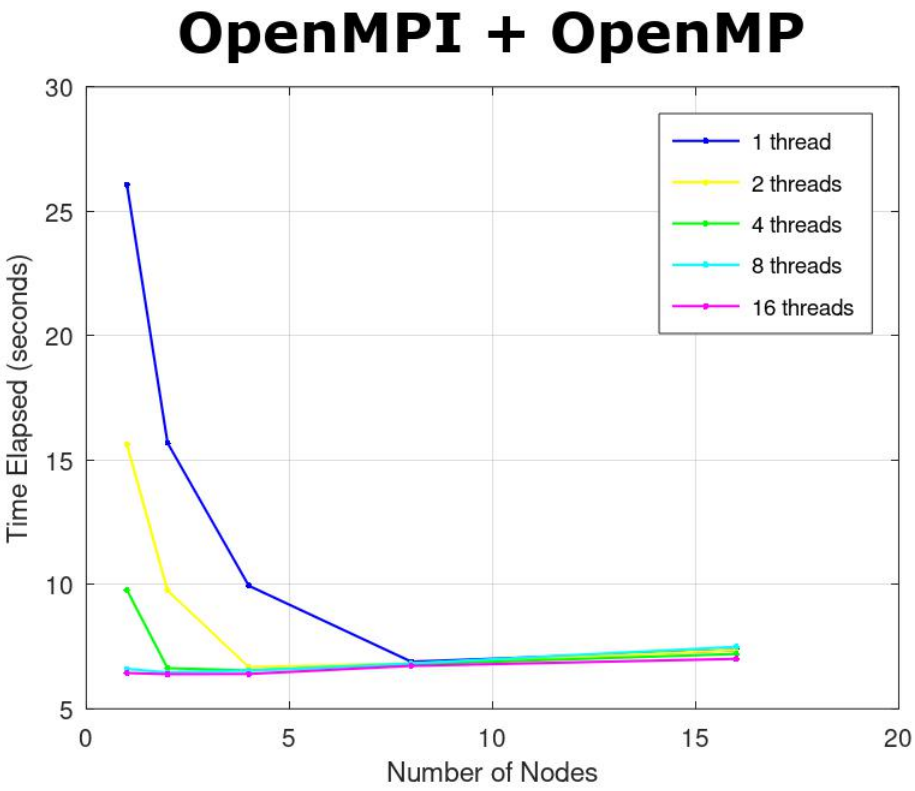

Fig. 3: Implementação com OpenMPI mais OpenMP.

Abaixo, temos uma tabela que representa os intervalos de confiança das 6 implementações medidos para a Tarefa 4:

Implementação	**Média (segundos)**	**Desv. Padrão**
Sequencial	23,6944	0,0534
Pthreads	4,18717	0,00320
OpenMP	3,90595	0,00911
OpenMPI	5,33655	0,00551
OpenMPI + Pthreads	4,74770	0,00998
OpenMPI + OpenMP	6,6639	0,0279

Table 1: Médias e desvios padrão para cada uma das implementações.

5 Análise

Nesta seção, iremos tecer comentários comparativos entre os casos e tentar inferir o motivo das tendências que os resultados mostraram. Além disso, iremos discutir as particularidades de cada um dos casos.

5.1 Comparações Gerais

Analisando o gráfico de comparação geral entre as seis implementações, podemos traçar algumas observações:

- O caso sequencial, fatalmente, tem uma performance muito pior quando comparado com qualquer uma das 5 implementações. Isto indica como, em casos similares a este, o uso de um programa totalmente sequencial subutiliza os recursos de *hardware*;
- Apesar de não tão significativo, podemos notar o *overhead* da troca de mensagens e manipulação de estruturas que as versões OpenMPI (com Pthreads e OpenMP) têm. Isto pode ser explicado pois, nestes casos, o *hardware* é completamente aproveitado, porém, como nos casos com OpenMPI temos que trafegar dados e manipulá-los entre as máquinas (ou núcleos, neste caso), faz sentido que haja um *overhead*;
- Não é possível traçar uma conclusão clara sobre qual modo de paralelização é melhor entre OpenMP e Pthreads. Vemos que uma teve uma performance melhor no caso OpenMPI e a outra no caso sem OpenMPI;
- Apesar de apontado como o OpenMPI possui um *overhead* notável, é de se esperar que, havendo maiores nós/máquinas no *cluster*, as implementações com OpenMPI são as que possuem o maior "teto", isto é, o maior potencial de escabilidade de performance. Em comparação, as versões sem OpenMPI só poderiam escalar melhorando o *hardware* de uma máquina, algo que tem um teto mais claro.

5.2 Comentários adicionais

Como pode-se notar, nos casos OpenMPI e OpenMPI com Pthreads, houve um aumento absurdo no tempo de execução quando se ultrapassou do número máximo de núcleos lógicos. Este aumento também é visto no caso OpenMPI com OpenMP, porém numa escala muito menor.

A justificativa para isso, pode ser pura implementação, uma vez que as duas primeiras foram feitas por um integrante do grupo, enquanto a terceira foi feita por outro integrante. Dito isto, estas implementações foram analisadas e, apesar de diferentes, não indicaram um motivo claro para esta discrepância.

Abstratamente, podemos justificar este comportamento com o fato de o OpenMPI tentar alocar mais nós que existem (16 nós, quando na prática só existem 8), porém isto não explica a diferença na proporção do aumento apontada nos dois parágrafos acima.

Além disso, vale notar como tanto no caso OpenMPI com Pthreads como no caso OpenMPI com OpenMP, temos uma espécie de comportamento muito similar com combinações da forma

$$n_{nós} \times n_{threads} \approx 8$$

o que faz sentido com o que foi discutido, no sentido destes arranjos representarem um "uso máximo" do *hardware*.