EP1: Cálculo do Conjunto de Mandelbrot em Paralelo com Pthreads e OpenMP

Felipe Brigalante 8941280
Mateus Rocha 8941255

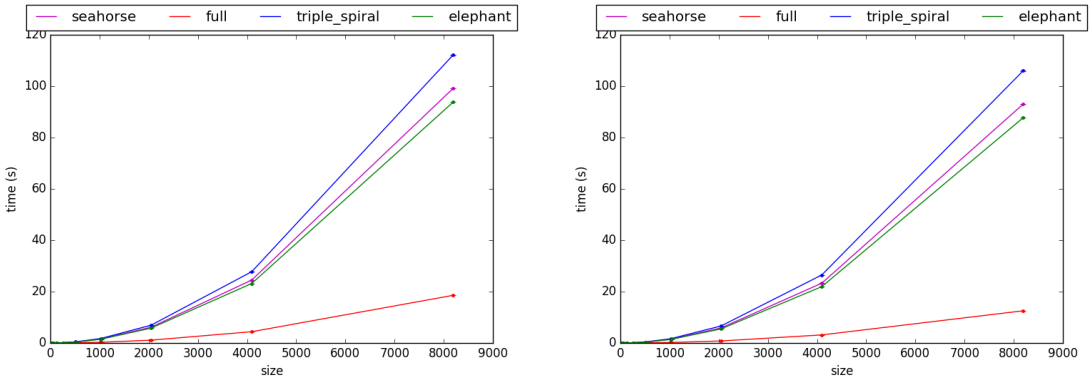
MAC 5742-0219 Introdução à Programação Concorrente, Paralela e Distribuída

1. Apresentação e Análise de Medições para o Programa Sequencial

1.1. Detalhes de Implementação

O programa sequencial foi implementado de duas maneiras, com alocação de memória e I/O e sem. O programa com alocação está igual ao disponibilizado, enquanto no sem alocação e I/O foram retiradas as função e variáveis responsáveis por isso.

1.2. Resultados



(a) Sequencial com alocação

(b) Sequencial sem alocação

Figura 1: 1 Thread

Em ambas implementações, a região "full"é calculada consideravelmente mais rápido que as outras e isso se deve ao fato de que possivelmente os pontos considerados nessa região demoram menos para convergir. Por isso, tanto no programa com e sem alocação, a proporção de tempo entre as regiões são parecidas.

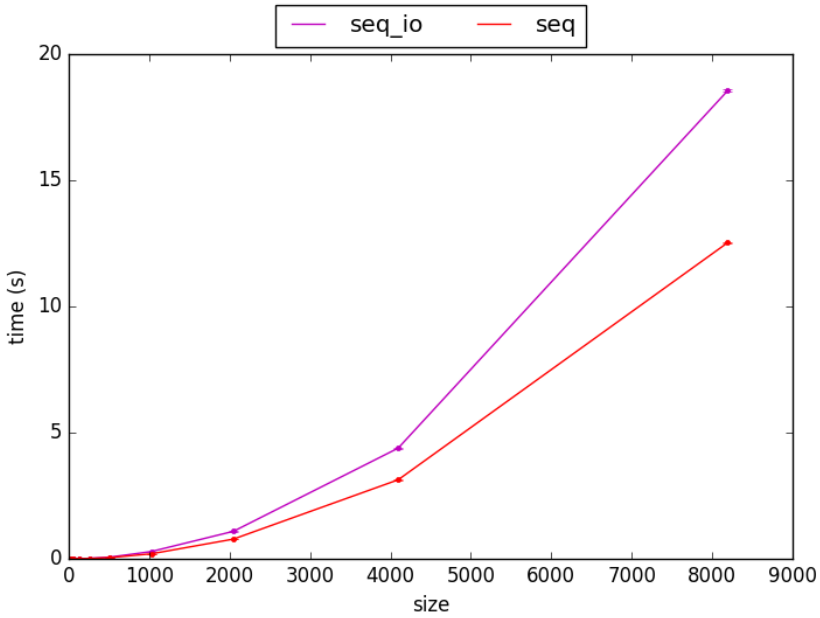


Figura 2: Thread: 1; Região: "full"

Podemos verificar que as operações de I/O e de alocação de memória aumentam consideravelmente o tempo de execução do programa, em especial em entradas com tamanho grande.

2. Apresentação e Análise de Medições para o Programa em Pthreads

2.1. Detalhes de Implementação

Para implementar a versão com pthreads foi paralelizado o método *compute_mandelbrot*. Primeiramente transformamos o *for* duplo em um único *for* para facilitar a paralelização do programa. Foram criadas várias threads que calculam um intervalo de pontos (dependendo do tamanho do chunk) e após terminarem verificam qual o próximo intervalo de pontos a ser calculado. Para isso utilizamos uma variável compartilhada que indica qual o próximo intervalo que deve ser calculado. Naturalmente essa variável tem acesso controlado por meio de um mutex.

O tamanho do chunk utilizado foi 8, e essa escolha foi por duas razões:

1. A quantidade de pixels sempre vai ser múltipla de 8, Então toda vez que uma thread for processar um chunk, o tempo de execução vai ser

parecido (claro que o tempo também vai depender de quanto tempo demora para cada ponto convergir)

2. Com um chunk de tamanho 8, sempre haverão pelo menos 32 chunks para serem processados. Então no pior dos casos, ou seja, uma imagem 16x16 executando com 32 threads, é esperado que cada thread pelo menos processe um dos chunks.

2.2. Resultados

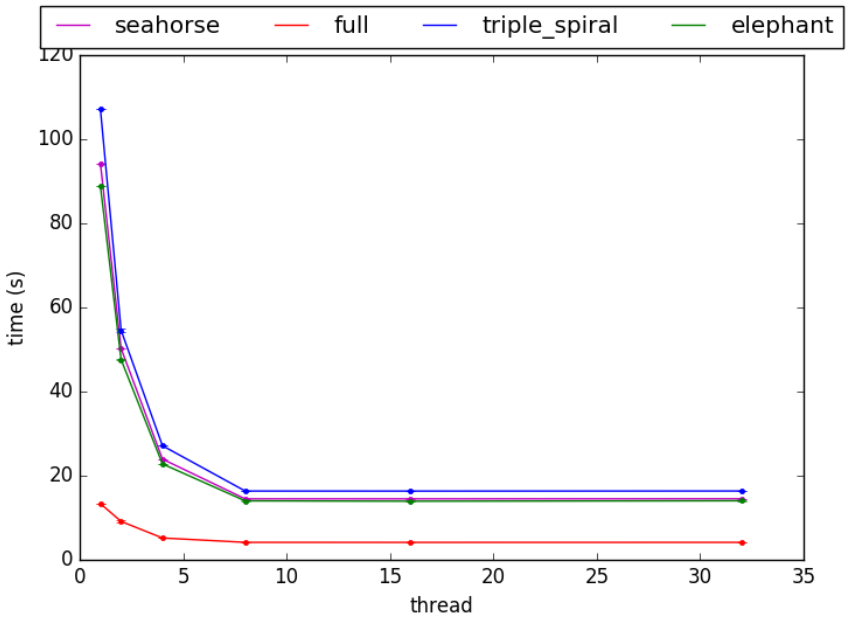


Figura 3: Tamanho: 8192; Método: pthread

Pode-se notar que nos experimentos com o tamanho da entrada grande, o tempo gasto para execução do programa é inversamente proporcional ao número de threads até que esse se equipare ao número de cores da máquina, nesse caso 8. Depois disso, podemos notar que o tempo gasto permanece praticamente constante, pois somente 8 threads podem rodar paralelamente, isto é, não há ganho de desempenho.

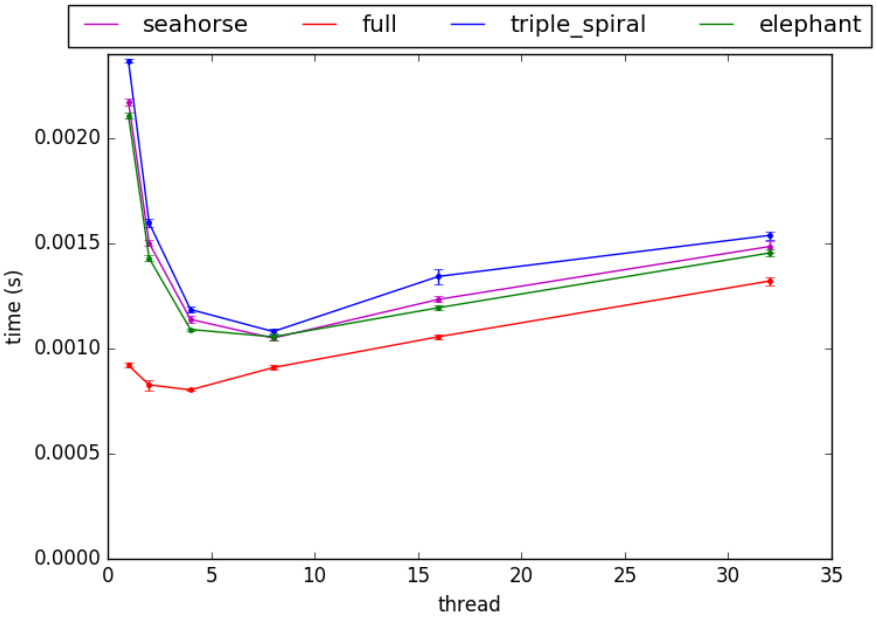


Figura 4: Tamanho: 32; Método: pthread

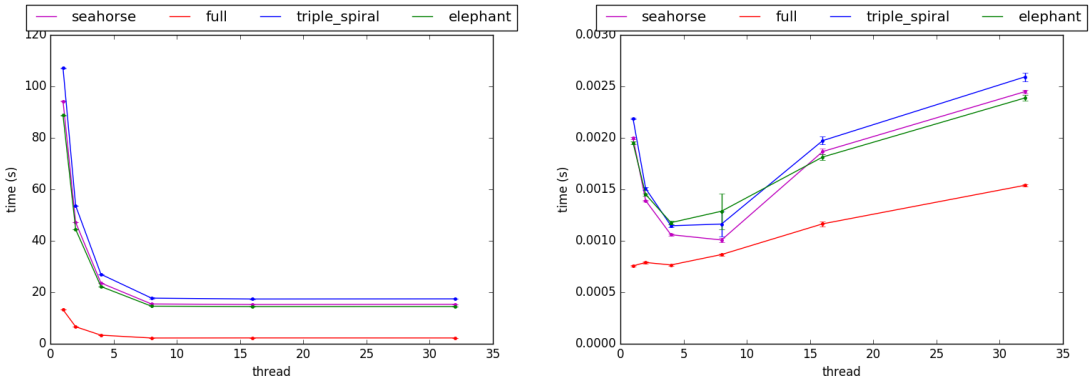
Por outro lado, quando há muitas threads e a entrada do programa é pequena, podemos verificar que o tempo gasto para alocar/organizar as threads acaba interferindo substancialmente no tempo de execução do programa deixando-o algumas vezes até mais lento do que se tivesse menos threads.

3. Apresentação e Análise de Medições para o Programa em OpenMP

3.1. Detalhes de Implementação

Assim como na implementação usando Pthreads, transformamos o *for* duplo num único *for* e colocamos o pragma adequado do OpenMp para que esse for fosse paralelizado.

3.2. Resultados



(a) Tamanho: 8192 (b) Tamanho: 32

Figura 5: OpenMP

Os resultados obtidos são semelhantes aos resultados da implementação com Pthreads e as conclusões são as mesmas.

4. Comparação de Medições entre os Programas

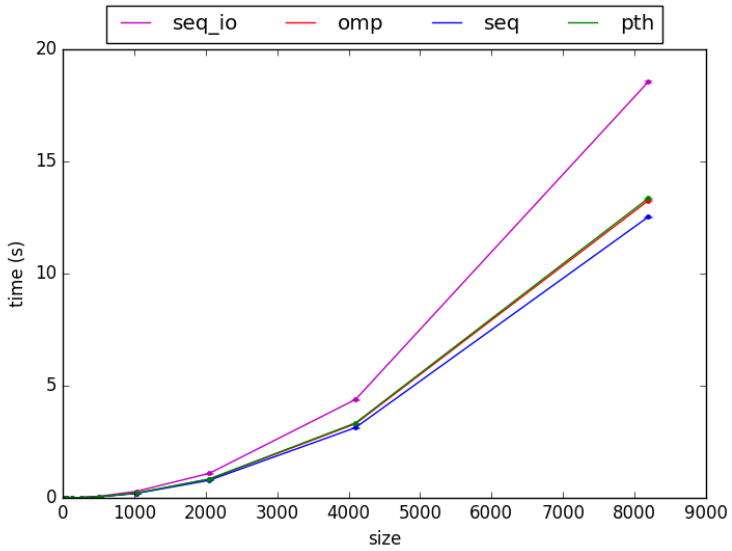


Figura 6: Thread: 1; Região: "full"

Podemos perceber que a versão sequencial é mais rápida justamente por não ter nenhuma computação adicional, enquanto as versões paralelas tem que alocar/organizar as threads e a versão sequencial com I/O tem que lidar com a alocação de memória e I/O.

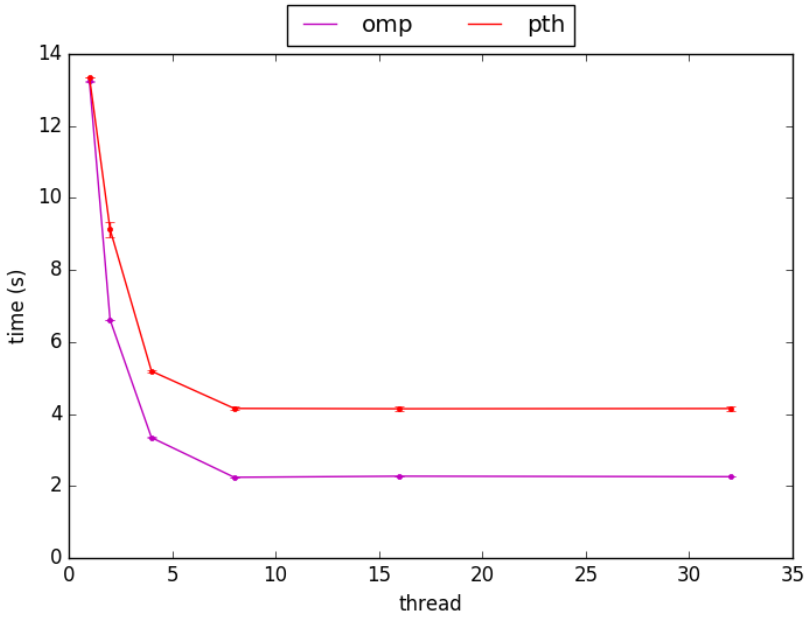


Figura 7: Tamanho: 8192; Região: "full"

Em execuções com tamanho de entrada grande vemos que a versão em OpenMP é um pouco mais rápida que a versão em Pthreads, e isso provavelmente se deve ao fato que a versão em OpenMP possui seus códigos mais otimizados do que o os que desenvolvemos na versão em PThreads.

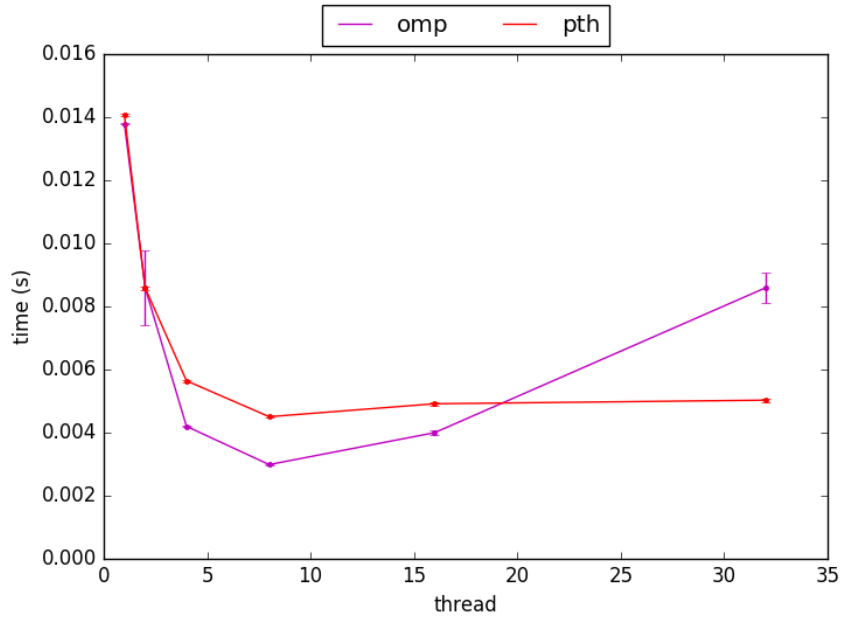


Figura 8: Tamanho: 256; Região: "full"

Por outro lado em execuções com tamanho de entrada pequena, a versão com Pthreads fica mais rápida e isso acontece pois possivelmente a versão em Pthreads possui um código mais simples do que o código gerado em OpenMP. Pode ser que o OpenMP demore mais na preparação/criação das threads e esse tempo não consegue ser recuperado na execução do algoritmo em si.

5. Discussão sobre as Medições

Programas concorrente são dependentes das decisões do escalonador e essas decisões podem variar de acordo com a disponibilidade dos processadores e a troca de contexto. São realizadas várias repetições, pois desse modo é possível identificar o tempo de execução dos programas na maioria dos casos, evitando obter resultados enviesados.

6. Links

Todo código feito estará disponível no link abaixo depois do dia de entrega do EP.

`https://github.com/mrocha94/MAC5742-0219-EP1`