Relatório MAC0219 EP03 2019

Parte 0: Definições gerais.

Para este EP, usamos C++, com o compilador *mpic++* na versão 7.4 e CUDA 10.0, com *nvcc* na versão 9.1.85. A versão do Open MPI utilizada nos testes é 3.0.1. Para gerar as imagens foi usado *png++*, que é um wrapper de C++ para a *libpng.*

Para escolher um delta x e um delta y, fizemos del_x = (c1.real - c0.real) / (w - 1) de del_y = (c1.imag() - c0.imag()) / (h - 1), onde c0, c1, w e h são passados pela linha de comando.

As imagens geradas nos testes estão na resolução 4000x4000 com c0 = (-2.0, -2.0) e c1 = (2.9, 2.0). A máquina utilizada nos testes é a dota da Rede Linux.

Parte 1: Implementação

Nos baseamos no código do EP2 para este EP. Primeiro, modificamos a *main()* do arquivo mandelbrot.cpp para que o MPI pudesse ser inicializado.

Utilizamos uma arquitetura “pseudo mestre-escravo”, uma vez que admitimos que nosso processo mestre realize o mesmo trabalho que os processos escravos fazem.

Ainda na função *main()*, o processo mestre faz a alocação do vetor de pixels que será preenchida. Chama a função *master_fill_matrix()*, onde o ponteiro para o vetor de pixels é passado e esta função retorna com o vetor já preenchido. Então, o processo mestre chama a função *create_picture()* e faz a desalocação de memória. Já os escravos chamam a função *slave_fill_matrix()*.

master_fill_matrix():

Esta função divide o vetor de pixels (como trabalho) entre os processos, de uma forma muito parecida com o programa pi.c.

A forma na qual dividimos os trabalhos entre os processos é muito parecida com o programa pi.c. Utilizamos um vetor de struct task tasks, onde task contém start e end que correspondem as posições do vetor de pixels que o processo correspondente trabalhará.

```
struct task {
	int start, end;
};
```

Nome: Cainã Setti Galante Nusp: 10737115
Nome: Guilherme Costa Vieira Nusp: 9790930
Nome: Victor Chiaradia Gramuglia Araujo Nusp: 9793756

Utilizamos *MPI_Send()* para mandar tasks[i] para o processo i.

Após distribuir as tarefas, chama-se a função *work()* e o mestre realiza sua parte do trabalho.
Depois, utilizamos a função MPI_Recv para juntar coletar os resultados dos processos escravos e colocar no vetor original e retornar.

slave_fill_matrix():

Utiliza-se a função MPI_Recv() para receber a struct task, com base no start e end calcula-se o work_size e alocamos um vetor de tamanho work_size e chamamos a função work(). Utiliza-se a MPI_Send() para mandar o vetor alocado com o trabalho realizado pelo processo.

work()

Nós temos uma flag global que indica se o trabalho será realizado com a CPU ou a GPU. Caso a CPU seja a escolhida, a função *work()* realiza o trabalho como no EP2, porém calculando a partir de start e terminando em end (passados como parâmetro) e utilizando Open MP para paralelizar.

Caso a GPU seja escolhida, chamamos a função *prepare()* do arquivo .cu que roda no *host()*. Pequenas modificações foram feitas no arquivo mandel.cu para realizar o trabalho de forma segregada.

Parte 2: Makefile

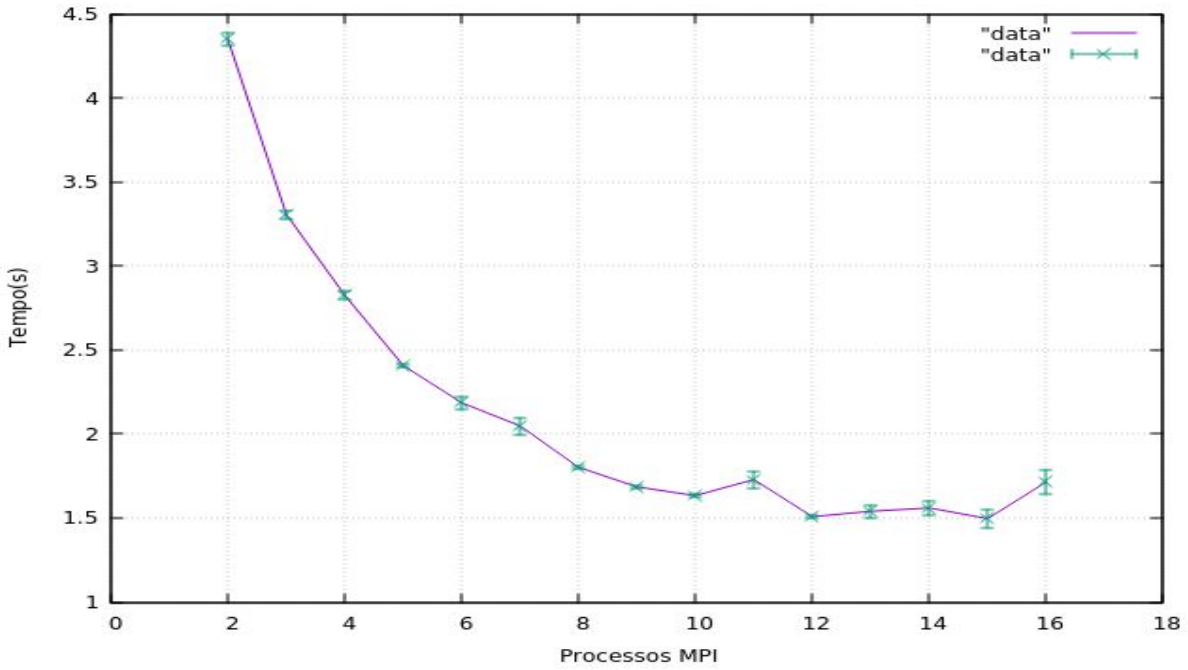
O Makefile incluso com o EP possui uma função para eliminar o arquivo binário gerado, os arquivos .o e os arquivos png. Além disso também possui dois testes que podem ser rodados usando *make cpu* e *make gpu*.

Parte 3: Testes

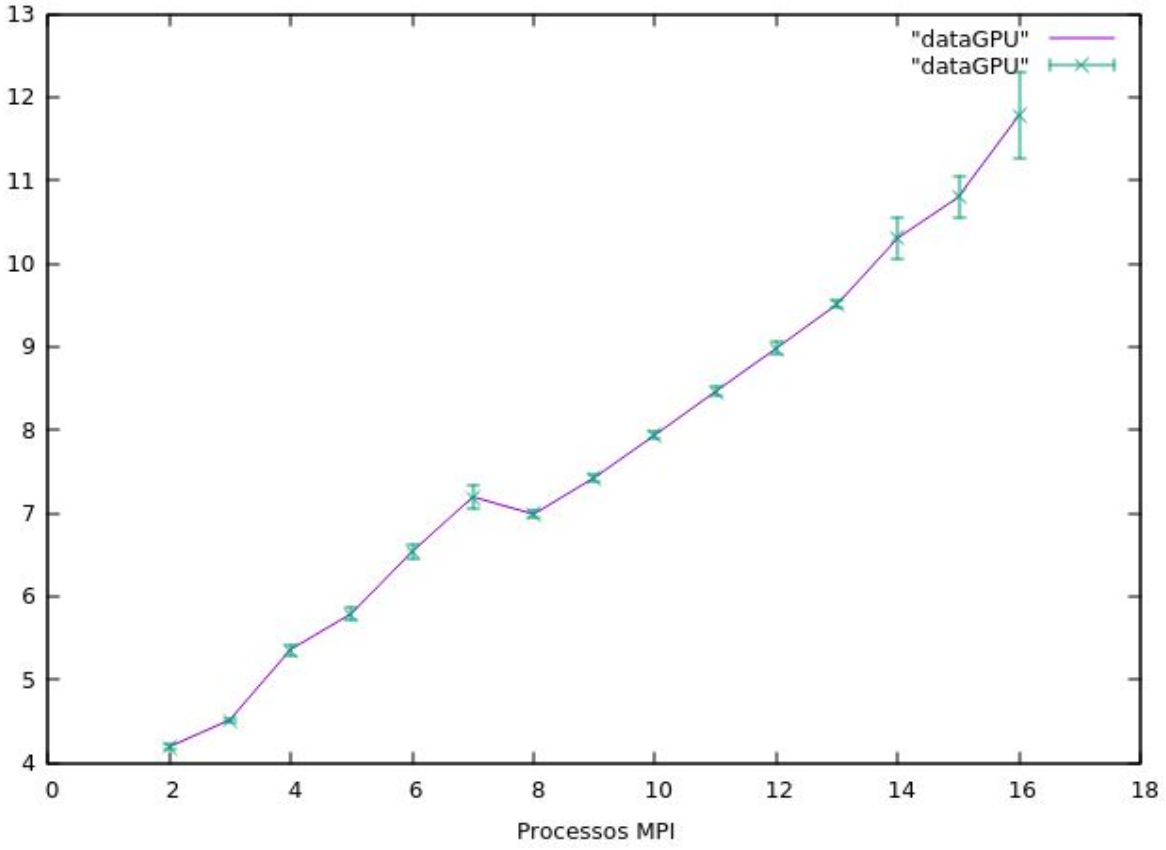
Para a coleção de dados, cada teste foi rodado 40 vezes.

Nome: Cainã Setti Galante Nusp: 10737115
Nome: Guilherme Costa Vieira Nusp: 9790930
Nome: Victor Chiaradia Gramuglia Araujo Nusp: 9793756

CPU



GPU



Nome: Cainã Setti Galante Nusp: 10737115
Nome: Guilherme Costa Vieira Nusp: 9790930
Nome: Victor Chiaradia Gramuglia Araujo Nusp: 9793756

5: Conclusão:

Como podemos analisar pelos gráficos que para a cpu o processo MPI reduziu o tempo do processo significativamente, porém para a gpu teve um aumento do tempo

Nome: Cainã Setti Galante Nusp: 10737115
Nome: Guilherme Costa Vieira Nusp: 9790930
Nome: Victor Chiaradia Gramuglia Araujo Nusp: 9793756