Trabalho de Programação Paralela e Distribuída - Mandelbrot Set com Cuda

Daniela Kuinchtner – 152064@upf.br

Implementação

O algoritmo foi implementado como vetor ao invés de matriz, sendo assim, precisou-se de blocos com apenas uma dimensão. Primeiramente, usava-se o tipo *complex* como declaração de uma variável de número complexo, porém, esse tipo de variável dá erro ao ser compilada com nvcc, então basicamente, transformou-se para duas multiplicações de números de tipo *float*.

Recursos utilizados

Foi utilizada a variável *THREADSPERBLOCK* como *define* no valor de 1024, para especificar quantas *threads* cada bloco terá. A variável *nBlocks* recebe o resultado do cálculo que definirá quantos blocos serão utilizados para cobrir todas as posições do vetor, onde cada *thread* ficará responsável por uma dessas posições. A função *cudaSetDevice* (0) selecionará o dispositivo a ser usado. A função *cudaMalloc* alocará na GPU para o *device d_A* o espaço determinado pela variável *size*, já a função *cudaMemcpy* copiará do *host h_A* para o *device d_A*, ou seja, copiará os dados que estão na memória RAM para a memória da GPU. Após, o *kernel* é invocado pela sintaxe *brot<<<nBlocks, THREADSPERBLOCK>>>(d_A, max_row, max_column, max_n, n)* passando o número de blocos e a quantidade de *threads* por bloco e seu *device d_A*, bem como as variáveis de linhas, colunas e iterações, e a variável *n*, onde esta é uma variável de controle dos índices das *threads*. O *kernel*, determinado com o especificador *__global__*, executará o cálculo do conjunto *mandelbrot*, onde cada *thread* acessará somente uma posição do vetor, a posição da mesma é determinada pela variável *k*. Os laços *for*, usados no código em sequencial, foram retirados, pois na chamada do *kernel*, a execução se tornará paralela. O vetor do *device d_A* receberá os resultados para a formação do conjunto *mandelbrot* e enviará de volta para o *host* pela função *cudaMemcpy*. Por fim, será mostrado o vetor e o *cudaFree(d_A)* liberará a memória do *device d_A*.

Resultados e análise

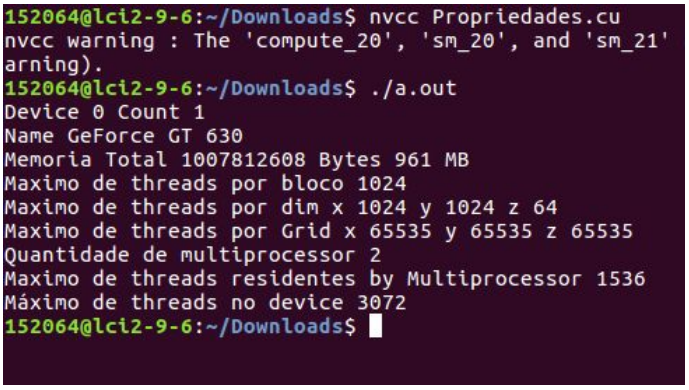
Foi utilizado um computador com processador Intel® Core™ i7-2600 CPU @ 3.40GHz × 8, 4 núcleos físicos e 8 *threads*, 8 GB de memória RAM e o sistema operacional Ubuntu 16.04 LTS, 64-bit e a GPU Geforce GT 630, que possui 2 multiprocessadores com 1536 *threads* para cada, 1GB de memória, máximo de 1024 *threads* por bloco e 3072 *threads* para todo o *device*. O anexo 2 mostra mais detalhes sobre a GPU. As médias de tempo obtidas após 10 execuções paralelas e 10 execuções sequenciais, foram de 2 segundos e 45 segundos, respectivamente. Essas execuções foram realizadas com as entradas de 1024 linhas, 768 colunas e 18000 iterações. O Anexo 1 mostra todos os tempos das 10 execuções paralelas e sequenciais. Nota-se que foram obtidos ganhos de desempenho comparando a execução paralela com a sequencial, pois mesmo com uma GPU de baixo processamento, a execução paralela foi 22 vezes mais rápida. Lembrando que, a execução sequencial, apenas com a remoção do tipo *complex*, teve uma redução de tempo, de 582 para 45 segundos.

Considerações finais

Conclui-se que a utilização do Cuda para a paralelização do Mandelbrot Set foi satisfatória. O fato de não haver comunicação com outros computadores, e a grande quantidade de threads disponível a torna uma das melhores alternativas como ferramenta quando utilizada independentemente.

Execuções	1024x768x18000 Sequencial	1024x768x18000 Paralelo
1	0m46.255s	0m2.308s
2	0m45.585s	0m2.121s
3	0m44.916s	0m2.134s
4	0m46.453s	0m2.103s
5	0m45.018s	0m2.094s
6	0m44.988s	0m2.056s
7	0m45.256s	0m2.204s
8	0m45.871s	0m2.301s
9	0m45.479s	0m2.129s
10	0m46.112s	0m2.114s

Anexo 1 – Tabela com os tempos das 10 execuções sequenciais e paralelas.



Anexo 2 – Propriedades da GPU.