Geração de Imagens do Conjunto Mandelbrot

Diogo H. M. Schaffer e Gabriel W. Rockenbach

I. Introdução

Imagens do conjunto de Mandelbrot exibem uma fronteira infinitamente complexa, a qual revela detalhes recursivos progressivamente finos em maiores ampliações. O limite do conjunto é definido como uma curva fractal, e o estilo de seu detalhe depende da região examinada. O limite do conjunto incorpora versões menores do formato principal, definindo uma propridade de auto-similaridade para todo o conjunto,e não somente suas partes.

O conjunto de mandeborot é o conjunto de números complexos c para qual a função $f_c(z) = z^2 + c$ não diverge quando iterada de $z = 0$, por exemplo, para qual a sequência $f_c(0)$,$f_c(f_c(0))$, etc., se mantém limitada em valor absoluto. Assim o objetivo deste trabalho é paralelizar, por meio de trocas de mensagens entre diferentes computadores pela biblioteca OpenMPI, a construção do conjunto Mandelbrot.

II. Implementação

Todos os pontos do conjunto de Mandelbrot existem dentro de um círculo de raio 2 e estão centrados na origem. Para construir o gráfico do conjunto de Mandelbrot, o processo é iniciado em um ponto C e continua a iteração por um certo número de passos. Se a iteração ultrapassa magnitude 2, o ponto C é definido como um membro do conjunto. Assim, para construir a imagem do conjunto necessitamos apenas do tamanho desta, m linhas por n colunas, quanto maior o tamanho maior o detalhamento da imagem e maior o tempo de processamento.

O processamento acontece pixel a pixel, dado que o ponto C, um número complexo que possui uma parte imaginaria e uma parte real, pode ter suas partes traduzidas para coordenadas de imagem x e y. Portanto podemos calcular cada pixel individualmente e paralelamente [1], porém, como dito antes, C esta limitado a um raio de 2 da origem, assim se este ponto (x, y) da imagem é detectado fora do conjunto podemos descartar este pixel, gerando desbalanceamento de carga para as $threads$ de processamento.

Desenvolvemos dois métodos de solução, o chamado de *static*, em que o trabalho total é dividido igualmente entre as *threads* disponíveis, e o *dynamic* no qual implementamos uma divisão do trabalho em pequenos pacotes e quando sua respectiva *thread* termina de processa-lo passamos outro pacote ao mesmo, assim, dado o desbalanceamento de carga, *threads* que inicialmente estão processando cargas leves podem termina-las rapidamente e receber pacotes de trabalho realmente significativos. Como estamos utilizando troca de mensagens, o que gera um *overhead* de processamento, precisamos achar o tamanho de pacote de trabalho ideal para cada máquina, minimizando o tempo de processamento e overhead. Achamos como o tamanho ideal de pacote de trabalho $(n_{threads} - 1) * 10$ linhas por cada *thread* no caso *dynamic*.

III. Experimentos

Foram utilizadas 2 maquinas do Laboratório de Alto Desempenho (LAD) que possuem 8 núcleos e 16 $threads$ de processamento cada, totalizando 32 threads e 16 núcleos. Utilizamos como tamanho da imagem a ser gerada 16000 X 16000 pixels. Podemos ver na Figura 1 a diferença entre os modos de escalonamento implementados.

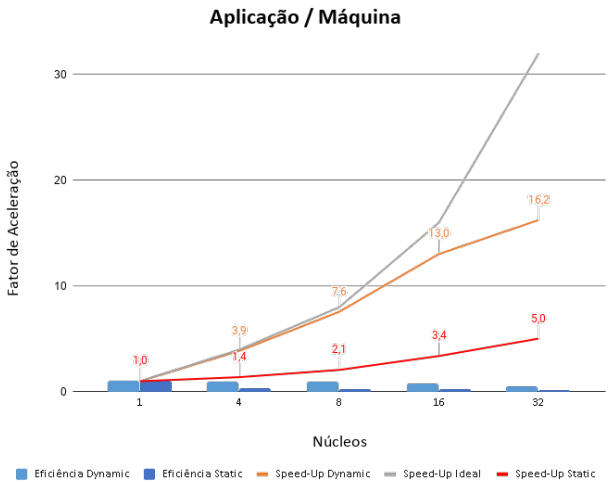

Fig. 1: Conjunto de experimentos com escalonamento *dynamic* e *static*

Podemos observar que somente o método de escalonamento *dynamic* leva a um *speed up* considerável, dado que o escalonamento dinâmico resulta em uma eficiência e *speed up* maior que o *static*. Isso acontece devido á distribuição de carga, dado que nosso escalonador divide e distribuí o trabalho de modo a não ter unidades de processamento ociosas, aumentando a eficiência da execução. A dificuldade dessa distribuição para este programa está na característica descrita na implementação, em que pontos detectados fora do conjunto resultam em suspensão do cálculo, assim gerando tempos variados para o cálculo de cada ponto, de forma que threads terminam o trabalho alocado em tempos diferentes.

References

[1] "Mandelbrot image generation." https://people.sc.fsu.edu/~jburkardt/cpp_src/mandelbrot_openmp/mandelbrot_openmp.html. Accessed: 2020-04-28.