EP3 - Cálculo do Conjunto de Mandelbrot em Paralelo com OpenMPI

Hélio Assakura
8941064

Julho de 2017

1 Introdução

O conteúdo deste relatório consiste na análise do impacto da programação paralela na construção de regiões do Conjunto de Mandelbrot[1]. Os resultados serão mostrados em forma de gráfico de barras gerados usando a biblioteca `matplotlib` da linguagem `Python`. Era obrigatório usar a biblioteca da ferramenta `OpenMPI`, e para os cálculos, variar a quantidade de *instâncias* e o tamanho da entrada como na Tabela 1:

	OpenMPI	Sequencial
Regiões	*Triple Spiral*, *Elephant*, *Seahorse* & *Full*	
№ de *Instâncias*	$2^0 \dots 2^4$	
№ de *Cores*	$2^4 \dots 2^0$	
Tamanho da Entrada	8192	

Tabela 1: Experimentos

Para cada valor de entrada, foram realizadas 10 medições. Os testes foram realizados na Google Compute Engine (GCE), usando as instâncias do tipo `n1-standard` e a imagem da VM disponibilizada pelo monitor (Pedro Bruel). Os resultados obtidos na VM, o código-fonte para manipular esses resultados, os gráficos obtidos e até mesmo o código-fonte dos programas estão disponíveis em `https://github.com/hassakura/MAC0219` .

[1] `https://en.wikipedia.org/wiki/Mandelbrot_set` [Acessado em 04/07/2017]

2 OpenMPI

O código desenvolvido foi uma modificação do código base disponibilizado pelo monitor[2]. As mudanças foram:

- Adaptação do código para o uso da ferramenta OpenMPI usando a biblioteca `mpi.h`;
- A criação da imagem é feita somente no final do programa, quando o número de iterações de cada píxel já tiver sido calculado;
- Uso da função `MPI_Gather` para juntar os cálculos dos processos.

Não foram usadas diretivas do `OpenMP`. Os experimentos foram divididos em 8 processos para todas as situações, independente do numero de cores, mudando apenas o número de slots de cada host.

3 Resultados

Os tempos obtidos na execução sequencial e distribuída (OpenMPI) variaram bastante. Neste EP, ao contrário do EP1, não houve o problema de inconsistência de resultados devido ao tamanho da entrada (2^4 por exemplo). As operações que consumiam tempo fora o processamento e I/O eram de troca de mensagem e latência, essa largamente discutida entre a comunidade do OpenMPI.

Notamos que o desvio padrão é relativamente pequeno, não passando de 2%. Os dados dos tempos também foram bem consistentes, mostrando o impacto do uso da ferramenta no calculo das regiões. O gráfico 1 mostra os tempos obtidos nos testes. Ele mostra a média e o desvio padrão de 10 medições de cada região, para 1, 2, 4 e 8 instâncias da Google Compute Engine (GCE) com 8, 4, 2 e 1 cores, respectivamente.

[2] `https://github.com/phrb/MAC5742-0219-EP1/blob/master/src/mandelbrot_omp.c` [Acessado em 09/07/2017]

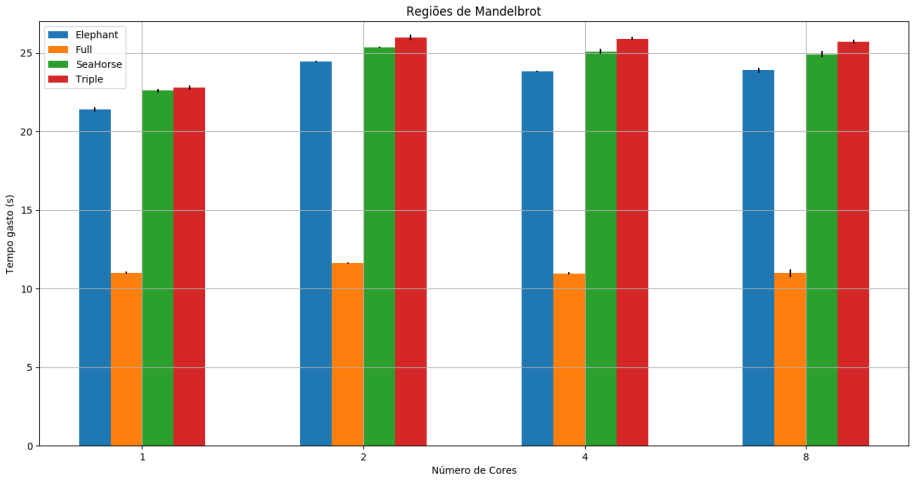

Figura 1: Tempos das execuções usando OpenMPI

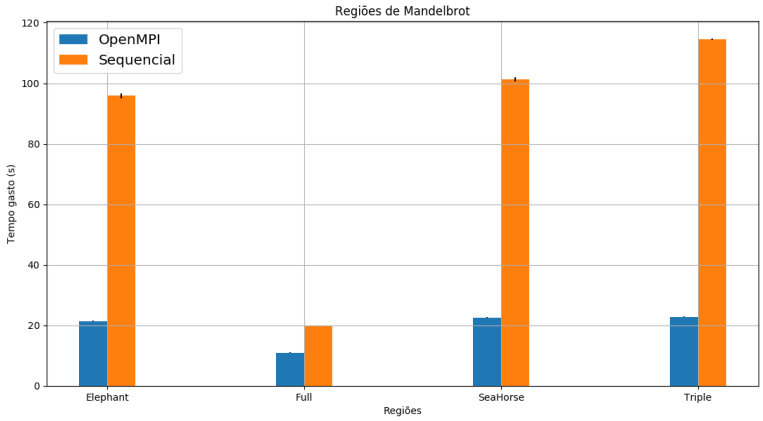

Figura 2: Tempos da implementação sequencial x implementação OpenMPI

Podemos ver claramente no gráfico 2 que a execução usando OpenMPI foi mais rápida que a sequencial. Isso se dá pois os cálculos foram distribuídos, com cada processo calculando uma parte da imagem, sendo construída posteriormente. Entre as execuções, notamos que o uso de mais máquinas melhora o desempenho do programa, porém os testes mostram que não houve uma variação muito grande entre os resultados paralelizados. Quando há uma quantidade de processamento maior, a implementação distribuída melhora e muito a rapidez dos cálculos. Por exemplo, em regiões como *Triple Spiral Valley*, a imagem foi gerada quase 6 vezes mais rápido. Os tempos coletados do programa sequencial

foram da instância `n1-standard-8` (8 vCPUs, 30 GB de memória).

4 Impacto de operações I/O, Aloc. Memória e Latência

Assim como em execuções usando `pthreads`, `OpenMP` e `CUDA`, as operações alheias ao processamento para `OpenMPI` tem impacto no tempo de execução. Enquanto no caso da GPU há um grande gasto de tempo para trocar informações entre host e device, no OpenMPI há o problema da latência. Como ocorre a troca de mensagens entre nós, dependo da estrutura da rede usada, pode acarretar em lentidão. Como usamos o *Google Compute Engine (GCE)* e um número pequeno de nós, achamos que a latência não seria fator determinante na medida de tempo do programa. Os dados levantados sobre o impacto de operações de I/O no conjunto de Mandelbrot[3] e na implementação de algoritmos de criptografia usando CUDA[4] já foram analisados anteriormente.

5 Considerações sobre a GCE

Apesar de bastante eficiente e prática, sua configuração para a execução do programa distribuído foi bastante difícil, principalmente para validar o acesso SSH das instâncias. Idealmente, uma instância criada deveria poder ser vista por todas as demais, mesmo se fosse através de um comando ou algo do gênero. De acordo com a documentação da GCE[5], ao conectar-se a uma instancia via SSH usando o `gcloud`[6], sua chave já estaria aplicada a todas as demais instâncias. Mesmo assim, não era possível acessar remotamente nenhuma instância criada por nós. Apesar dos problemas, achamos uma solução e concluímos o EP.

6 Conclusão

Depois de executarmos os programas para as diversas regiões do Conjunto de Mandelbrot, o tempo necessário para gerar a imagem fica menor do que um sequencial. Usando a ferramente de distribuição de processos, pudemos ver que há um ganho considerável no tempo de execução. Muitos dos supercomputadores usam o OpenMPI, e a quantidade de dados processados neles é imensa. Os testes realizados eram bem simples, mas conseguimos ver que o ganho de eficiência é muito grande, e se usado conjuntamente com o OpenMP, pode ficar ainda maior.

[3] `https://github.com/hassakura/MAC0219/blob/master/EP1/doc/Relat_EP1.pdf` Acesso em 09/07/2017

[4] `https://github.com/hassakura/MAC0219/blob/master/EP2/EP2.pdf` Acesso em 09/07/2017

[5] `https://cloud.google.com/compute/docs/instances/connecting-to-instance` Acesso em 09/07/2017

[6] `https://cloud.google.com/sdk/gcloud/` Acesso em 09/07/2017

Ao estudar as ferramentas de paralelismo e distribuição, conseguimos perceber que no geral, independente de qual for utilizada, a eficiência é maior comparada a execuções sequenciais.