Instituto de Matemática e Estatística

EP1 - MAC0219

Cálculo do Conjunto de Mandelbrot em Paralelo com Pthreads e OpenMP

Professor: Alfredo Goldman

Alunos: Bruno Sesso

Gustavo Estrela de Matos

Lucas Sung Jun Hong

São Paulo, 23 de Abril de 2017

Conteúdo

1 Introdução

2 Código Sequencial

3 Código em OpenMP

4 Código em Pthreads

Utilizamos duas diferentes abordagens para paralelizar o código com o uso de pthreads, tentando se aproximar ao comportamento do OpenMP e suas diretivas *omp parallel for schedule (dynamic)* e *omp parallel for schedule (static)*. As diferenças de funcionamento dos dois códigos está na maneira em que o trabalho é dividido e como ele é distribuido para cada thread.

4.1 Implementação com Divisão Estática

Chamamos de implementação com divisão estática a versão do nosso código em pthreads no qual o trabalho é dividido em n pedaços de mesmo tamanho, e cada pedaço é dado a uma thread. Chamamos essa implementação de estática porque cada thread recebe apenas um bloco de trabalho, pré-determinado pela divisão feita, que será processado do começo ao fim (no escopo da thread) pela mesma thread.

Para implementar esse código, precisamos apenas construir uma estrutura de dados que era capaz de guardar um bloco de pixels a ser calculado. Dado essa estrutura, basta criar uma thread para cada bloco de trabalho, que por sua vez deve calcular os pixels correspondentes e atualizar o buffer de cores.

Devemos observar que essa implementação pode implicar em threads ociosas enquanto outras estão trabalhando. Como a quantidade de iterações necessárias para se calcular o valor de um pixel varia, é possível que um bloco de pixels seja calculado muito mais rápido do que outro; imagine por exemplo um bloco onde cada ponto calculado diverge rápidamente e outro bloco onde isso não acontece. Portanto, como a divisão de trabalho é estática, é provável que

uma thread termine seu trabalho muito antes de outra, o que significa em um uso não muito bom de recursos da máquina.

Uma possível solução para esse problema seria o aumento no número de threads, o que cria uma fragmentação maior do trabalho. Essa fragmentação ameniza o problema anterior porque divide mais o trabalho, deixando menor a diferença de tempo necessário para se calcular cada parte. Entretanto, criar um número excessivo de threads pode dar mais trabalho ao escalonador do sistema operacional, que deve lidar com várias linhas de processamento.

4.2 Implementação com Divisão Dinâmica

5 Discussões Gerais

6 Conclusão