Insper

06 - For paralelo e compartilhamento de dados

Super Computação 2018/2

Igor Montagner, Luciano Soares

Exercício 1: na última aula vocês devem ter produzido uma solução parecida com a abaixo (arquivo *pi-omp-critical.cpp*. Modifique-a para usar as construções de for paralelo e redução.

```
#include <omp.h>

double pi_omp_critical(long num_steps) {
    double step = 1.0/(double) num_steps;
    double sum = 0;

    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        long nt = omp_get_num_threads();
        double d = 0;
        for (long i = id; i < num_steps; i+=nt) {
            double x = (i-0.5)*step;
            d += 4.0/(1.0+x*x);
        }
        #pragma omp critical
        {
            sum += d;
        }
    }

    return sum * step;
}
```

Exercício 2: Ao invés de usar a diretiva de redução poderíamos simplesmente colocar uma diretiva `critical` ao redor da atualização da variável `sum` e usar somente o for paralelo.

1. Faça esta modificação no seu programa. Os resultados numéricos são os mesmos? O quê aconteceu com o tempo de execução?

2. Se existir diferença de desempenho, explique sua razão.

Exercício 3: O arquivo *mandel.cpp* tem uma implementação que calcula a área abaixo do fractal de Mandelbrot. Foi feita uma tentativa preguiçosa de paralelização usando OpenMP que está dando resultados muito estranhos. Seu trabalho neste exercício é modificar o `#pragma omp` para que ele não permite o compartilhamento indevido de dados entre as threads.

Exercício 4: Apesar do programa agora funcionar, os erros do exercício acima eram causados essencialmente por más práticas de programação. Reestruture o código para que o resultado de suas funções só dependa dos valores passados nos argumentos. Isto costuma implicar na conversão de valores lidos/escritos em variáveis globais para valores passados nos parâmetros da função. Neste exercício você pode mudar o programa extensamente, desde que os resultados continuem os mesmos.

Teoria*: Dizemos que quando uma função escreve/lê em variáveis globais (ou* `static`*) ela possui* efeitos colaterais*. Ou seja, após rodar ela modifica o estado do programa. Em comparação, uma função que só depende dos valores passados nos argumentos e não escreve seu resultado (ou valores intermediários) em variáveis globais é dita* sem efeitos colaterais*. Este tipo de função pode ser chamado por várias threads simultâneamente e é uma boa prática de programação paralela criar funções sem efeitos colaterais.

Curiosidade: fractais são estruturas matemáticas que são definidas por sua auto-similaridade. Eles são úteis para modelar objetos e fenômenos que possuem as mesmas características em escalas completamente diferentes, como nuvens, montanhas e compressão de arquivos.

Exercício 4:

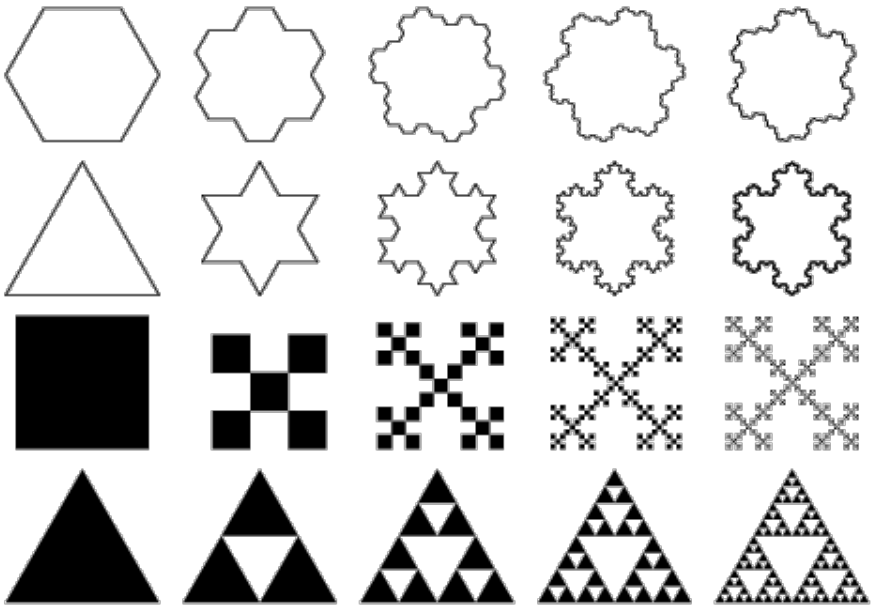

Figure 1: Exemplos de fractais, Fonte: mathworld.wolfram.com.