Insper

03 - Modelo fork-join raiz

SuperComputação - 2018/2

Igor Montagner, Luciano Soares

Neste roteiro iremos implementar o modelo *fork-join* (Fig 1) usando as funções do cabeçalho `std::thread`.

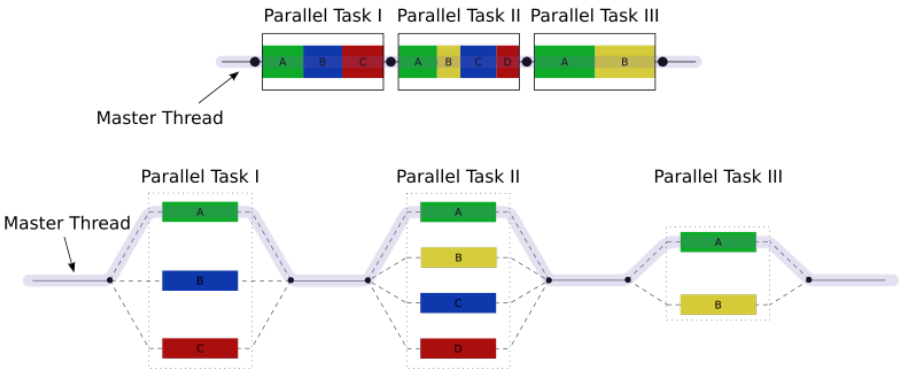

Figure 1: Modelo fork-join

Usando `std::thread`

Neste roteiro iremos aprender a criar e esperar a finalização de threads criadas usando C++11 `std::threads`. Veja abaixo um exemplo com as funções que precisaremos usar (arquivo *exemplo1-threads.cpp*).

```cpp
#include <thread>
#include <iostream>

void funcao_rodando_em_paralelo(int a, int *b) {
    std::cout << "a=" << a << std::endl;
    *b = 5;
}


int main() {
    int b = 10;

    // Cria thread e a executa.
    // Primeiro argumento é a função a ser executada.

    // Os argumentos em seguida são passados diretamente
    // para a função passada no primeiro argumento.
    std::thread t1(funcao_rodando_em_paralelo, 15, &b);


    std::cout << "Antes do join b=" << b << std::endl;

    // Espera até que a função acabe de executar.
    t1.join();

    std::cout << "Depois do join b=" << b << std::endl;
}
```

Compilamos programas usando threads com a seguinte linha de comando.

```
$ g++ (flags de compilação) arquivo.cpp -o nome_executavel
-lpthreads
```

Exercício 1: Compile o código acima e execute-o diversas vezes. A saída foi igual todas as vezes?

Exercício 2: Faça um programa que cria 4 threads e atribui a cada uma um *id* de 0 a 3. Cada thread deve executar uma função que imprime "Thread:" + id.

Exercício 3: Generalize o programa acima para detectar o número de threads paralelas do sistema. Pesquise como obter esta informação usando C++11 threads.

Pergunta: C++11 threads não permitem que as threads retornem valores. Como você faria para que suas threads calculem algo e retornem para o `main`? (Dica: olhe o exemplo acima).

Um problema "real"

Usaremos neste roteiro o problema da soma de todos os elementos em um vetor. Faremos uma versão sequencial e uma versão paralela usando *fork-join*.

Pergunta 1: como você dividiria este problema em várias tarefas paralelas?

Exercício 1: faça um programa que gera uma sequência de números aleatórios (usando `std::random`) seguindo uma distribuição normal com média 1 e variância 5.

Exercício 2: Crie uma função `double soma_vetor_seq(double vec*, int n);` que realiza a soma dos elementos do vetor.

Exercício 3: Modifique a função acima de modo a facilitar a implementação que você identificou na pergunta 1.

Exercício 4: Crie uma função `double soma_vetor_par(double *vec, int n)` que paraleliza a soma do vetor usando a estratégia da Pergunta 1.

Exercício 5: Teste sua implementação com vetores de tamanhos incrementalmente grandes e verifique se existe vantagem em usar sua implementação paralela.

Extra: plote um gráfico ilustrando as diferenças de desempenho.