

Programação com Threads

Raquel Lopes de Oliveira¹

Resumo

Este é um relatório desenvolvido para a disciplina de Programação Corrente, período 2017.2, professor Everton Ranielly de Sousa Cavalcante. O relatório consiste na análise do registro dos tempos de execução realizando o mesmo o objetivo, multiplicação de matrizes, mas comparando a abordagem sequencial e concorrente.

Palavras-chave

Threads — Programação — Concorrência

¹2013023946

1. Introdução

O propósito desse trabalho é realizar um teste para validar ou não a intuição em relação ao uso da concorrência. Para isso devemos fazer testes fazendo uso da concorrência e uso da solução sequencial nas mesmas condições e com diferentes parâmetros (por exemplo, número de threads). No presente trabalho o algoritmo implementado foi o da multiplicação de matrizes.

2. Implementação

As implementações dos códigos foram feitas na linguagem C++. Para computarmos o tempo, foi usada a biblioteca `chrono`¹. Uma classe `Matrix` foi criada para representar uma matrix, uma versão dessa classe já tinha sido previamente parcialmente implementada para outra disciplina e pode ser verificada no repositório `numerical-analysis` no github, essa classe faz uso de `template` e já possui diversos métodos, que fazem jus ao nome da classe, implementados. A multiplicação do algoritmo sequencial foi feita usando a sobrecarga do operador `*` e para o algoritmo que faz uso de concorrência foi criado dois métodos, um responsável por fazer a atualização da matriz para determinados elementos da matriz resultante e outro (`multiply`) responsável por ser chamar o método anterior (`multiplyAtomic`) dado o número de threads que for ser usado.

O número de threads que é aceito pelo programa é igual ao número de linhas e colunas das matrizes, dado o fato do procedimento que é realizado para a multiplicação de matrizes. Caso o número de threads seja menor que o número de linhas². O número de operações atômicas (multiplicação de uma linha por uma coluna que resulta em uma célula da matriz produto) é determinado pelo seguinte cálculo:

$$nb_op = \frac{num_linhas \times num_colunas}{num_threads}$$

Para a última thread o cálculo tem uma alteração para que ela possua o valor do restante das operações (atômicas) que não foram realizadas:

$$nb_op = \frac{num_linhas \times num_colunas}{num_threads} + (num_linhas \times num_colunas) \% num_threads$$

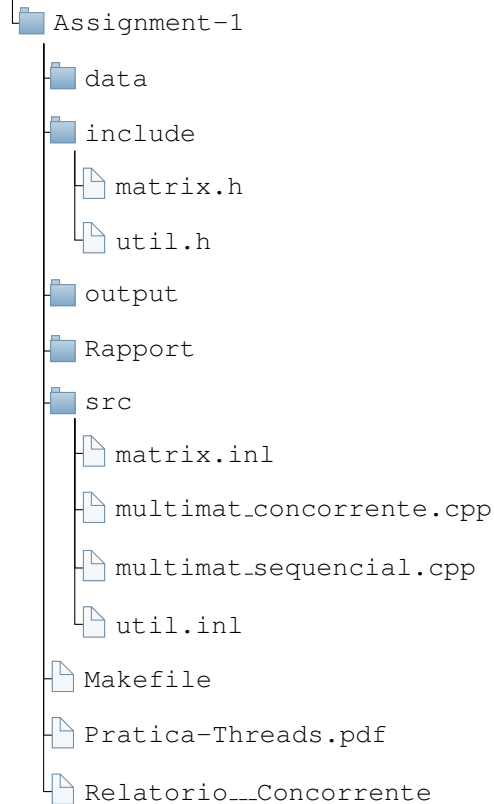
Ao usar threads, usamos a chamada `join()` de forma que possamos garantir que tudo já tenha sido computado antes que o programa chegue ao fim, ou seja, é a forma que temos para confirmar que todo o fluxo execução destinado aquela thread foi computado. Caso tivéssemos usado a chamada `detach()` não poderíamos garantir isso, normalmente esse mecanismo é indicado para coisas que queremos que rode em *background*.

¹<http://www.cplusplus.com/reference/chrono/>

²ou colunas já que se trata de uma matriz quadrada

2.1 Organização

Concurrent-Computing



2.2 Como executar

Uma vez dentro do diretório *Assignment-1* basta executar o comando:

```
$ make
```

que será criado dois executáveis: **multimat_sequencial** e **multimat_concorrente**. Ele funciona da mesma forma como descrita na especificação do projeto: “O programa principal deverá ser executado via linha de comando da seguinte forma:

```
$ ./multimat_sequencial 2
```

em que o número inteiro seguido do nome do programa representa a dimensão das matrizes quadradas que serão tratadas pelo programa. Todas as matrizes utilizadas como casos de teste para este trabalho possuem dimensões como potências de base 2, logo qualquer valor fornecido como argumento de linha de comando ao programa deve atender a essa restrição. No caso da solução concorrente, o programa principal deverá ser executado via linha de comando da seguinte forma:

```
$ ./multimat_concorrente 2 2
```

em que os números inteiros seguidos do nome do programa representam, respectivamente, a dimensão das matrizes quadradas que serão tratadas pelo programa e o número de threads a serem utilizadas.”No caso da solução concorrente, caso não

seja definido o número de threads ele vai ser setado com o valor máximo (definido pela constante **NUMBER_THREADS**(10) e caso a dimensão da matriz seja menor que a constante, o número de threads é igual ao valor máximo (número de colunas/linhas) dado que a sub-tarefa mínima para uma thread é o cálculo de um elemento da matriz produto, ou seja, a multiplicação de uma linha por uma coluna.

Os documentos de entrada devem estar dentro da pasta *data* e as matrizes resultantes da multiplicação se encontram na pasta *output*, ambas as pastas já estão previamente criadas e é possível verificar a hierarquia na seção 2.1.

3. Metodologia

A máquina usada para a realização de testes foi

4. Resultados

5. Discussão