Universidade Federal do Rio Grande do Norte

DIM0612

Programação Concorrente

Trabalho prático: Programação com threads

Discentes:
Marciel Manoel Leal

Matrícula:
2015039785

7 de Outubro de 2017

Conteúdo

1 Introdução

Tendo em vista que muitos processadores modernos funcionam com mais de um núcleo, muitas vezes os programadores fazem softwares que subutilizam os recursos computacionais disponíveis, pois não possuem muitos conhecimentos em programação concorrente ou não querem fazer uso dela **TODO**. No entanto a programação concorrente pode gerar muitos bugs inesperados e muitas vezes ir-rastreáveis, além disso, ela pode não ter um ganho tão significativo, dependendo do contexto.

Este documento visa relatar um pequeno experimento sobre programação concorrente. O objetivo desse experimento é comparar a eficiência da programação concorrente com a programação sequencial no problema de multiplicação de matrizes.

2 Detalhes de implementação

A implementação auxiliar se divide em dois pacotes descritos a seguir:

- **squarematrix.py** possui a implementação da classe **SquareMatrix** que encapsula a criação de uma matriz quadrada e o acesso a seus elementos
- **multiplier.py** possui a implementação de algumas funções para a multiplicação de matrizes:
  - **matrixmult** é a função chamada na multiplicação de matrizes. Ela possui os parâmetros A e B, que são as matrizes a serem multiplicadas e possui um parâmetro opcional que é o número de threads, caso ele seja passado e esse número for maior que 1, a função fará a multiplicação utilizando multiplas threads, caso contrário, fará de forma sequencial.
  - Seguindo as especificações do PEP8, as seguintes funções são funções internas do pacote, que auxiliam na execução da função **matrixmult**:
    - **__rowmultmatrix** multiplica uma linha de uma matriz quadrada A pelas colunas de B (é, digamos, o núcleo da multiplicação);
    - **_runthread** é uma função target para a criação das Threads.

Há mais dois arquivos que são os scripts de fato. Esses scripts recebem um tamanho de matriz como parâmetro, caso esse tamanho não for de acordo com os requisitos, o programa não executa. Os scripts são descritos abaixo:

- **multimat_sequencial.py** é o script que multiplica matrizes sequencial. É necessário a passagem de um parâmetro para a sua execução. Esse parâmetro

é a dimensão da matriz;

- **multimat_concorrente.py** é o script que multiplica matrizes de forma concorrente. É necessário a passagem de dois parâmetros para a sua execução. O primeiro parâmetro é a dimensão da matriz e o segundo é a quantidade máxima de threads que podem ser criadas.

O objetivo dos scripts é fazer a multiplicação de duas matrizes: A e B. Essas matrizes estão armazenadas em arquivos nomeados como *tipoDimensaoxDimensao*, onde o tipo pode ser A ou B, e o Dimensao é o valor da dimensão da matriz. Todas essas matrizes estão dentro da pasta **Matrizes**. Logo, ao ser passada uma dimensão d para um dos scripts, ele irá abrir os arquivos **Adxd** e **Bdxd**. Criará dois objetos do tipo **SquareMatrix** e irá chamar a função **matrixmult**, passando esses objetos como parâmetros. No caso do script concorrente, ele passará também o número de threads que foi recebido via terminal.

3 Metodologia

O experimento utilizou-se de matrizes quadradas. Os requisitos para as dimensões dessas matrizes foram:

- ser uma potência de 2
- estar no intervalo estão no intervalo $[4, 1024]$

O tempo foi medido utilizando a função time da biblioteca time. O tempo de cada script foi impresso na saída padrão.

As exceções foram colocadas para serem impressas na saída padrão. Nenhuma exceção foi levantada no experimento definitivo.

Os experimentos foram realizados utilizando um script **tests.sh**. Esse script possui um loop de 20 iterações, onde chama os dois scripts e redireciona suas saídas para arquivos. As saídas do executável sequencial foram armazenadas na pasta **MeasuresSeq** e as do concorrente na pasta **MeasuresConc**. O nome desses arquivos de saída seguem a sintaxe *dimensaoxdimensao*, onde dimensao é a dimensão passada ao script. O número de Threads utilizada para cada execução do script concorrente foi metade da dimensão da matriz, ou seja, cada Thread multiplicou duas linhas da sua respectiva matriz A.

Os dois scripts foram testados para todas as dimensões de matrizes antes do experimento, verificando-se a corretude da matriz-resultado, e, portanto, dos algoritmos.

No experimento, a matriz-resultado foi calculada, mas não foi impressa, já que a corretude da saída não era o objetivo dos testes.

Utilizou-se de um computador com as seguintes especificações:

- Memória RAM: 8GB
- Sistema operacional: Ubuntu 16.04 LTS 64 bits
- Processador: Intel Core i7-5500U CPU @ 2.40GHz × 4

A linguagem utilizada para o script de testes foi Shell Script, executado no bash versão 4.3.48.

A linguagem utilizada foi Python em sua versão 3.5.2.

4 Resultados

As podem ser vistas nessa planilha e alguns resultados tirados podem ser lidos abaixo (todos os números estão em segundos):

Tabela 1: Script sequencial

Dimensão	Média	Máximo	Mínimo	Desvio Padrão
4	0.0002907991409	0.0006022453308	0.0002374649048	0.0001066008816
8	0.002117300034	0.02339220047	0.00088763237	0.005010325641
16	0.005999982357	0.007823228836	0.00562620163	0.000496041051
32	0.04389901161	0.0468583107	0.04241871834	0.00111283432
64	0.3491618276	0.4293854237	0.3288066387	0.0245424195
128	2.660050738	2.954339743	2.540849924	0.1020514268
256	20.83647889	21.44054699	20.37756658	0.2901211262
512	168.2958302	175.4287961	163.7993152	2.563394119
2014	1361.676831	1448.133599	1304.051964	34.12679676

Tabela 2: Script sequencial

Dimensão	Média	Máximo	Mínimo	Desvio Padrão
4	0.001057624817	0.004051923752	0.0006701946259	0.0007554259476
8	0.006040561199	0.08232426643	0.0017619133	0.01795630738
16	0.009890294075	0.04310035706	0.007635116577	0.007829143285
32	0.0488155961	0.0677189827	0.04609394073	0.004545745098
64	0.3510119915	0.4072928429	0.3347120285	0.01566897484
128	2.711036754	2.787316322	2.645894527	0.04007794359
256	24.11169454	30.74346662	21.43333721	2.155104241
512	173.1249938	195.6851428	167.7998388	6.614993082
1024	1415.55427	1486.977663	1372.838478	33.68285217

5 Discussão

Vê-se que o algoritmo sequencial foi melhor que o concorrente na maioria dos casos. Acredita-se que a complexidade para a criação das Threads venceu a tarefa de multiplicação, como esperado. Esse é um exemplo de que, dependendo da implementação, Threads não necessariamente serão mais eficientes.