**Laboratório de Programação Avançada**

**Trabalho 12**

**Kevin Takano – 21204071**

**Solução Proposta**

Como resolução deste trabalho de LPAv, foi calculado o tempo de execução para soma dos elementos do vetor da thread main, e foi calculado o tempo de execução para soma dos elementos do vetor com *k* threads.

Para o primeiro caso, a ideia feita foi muito simples, apenas foi utilizado uma variável de soma, que vai somando todos os elementos do vetor e assim retornado o valor da soma.

Para o segundo caso, foi criado um vetor de threads, e assim ativado cada thread em um *for* com a função *pthread\_create* de cada elemento do vetor de threads. Em cada chamada da função *pthread\_create* dentro do for, é passado por parâmetro, a função *calculate\_two\_elements* para que cada thread calcule duas posições do vetor.

Como a função *calculate\_two\_elements* trabalha com a região de memória, foram criado dois *mutexes* que ‘lockam’ a região crítica. Além disso, a função só faz os cálculos enquanto *amount\_of\_elements\_in\_run* é menor que um, para poder as threads trabalharem até onde devem, e não mais ou menos que devem. Outro ponto importante, é que o cálculo é feito sempre utilizando a variável *amount\_of\_elements\_in\_run* como referência a posição do vetor e somado ao início do vetor. Portanto, a variável *amount\_of\_elements\_in\_run* deve ser decrementada e o valor final da soma estará na posição 0 do vetor.

**Script de Execução**

Para execução do programa e não fosse muito exaustivo para construir a tabela, foram utilizados os seguintes scripts no terminal do Linux, para as respectivas entradas 500, 1000, 2000, 5000,50000. Observe:

./tp12 500 2 && ./tp12 500 4 && ./tp12 500 8 && ./tp12 500 16 && ./tp12 500 32 && ./tp12 500 64

./tp12 1000 2 && ./tp12 1000 4 && ./tp12 1000 8 && ./tp12 1000 16 && ./tp12 1000 32 && ./tp12 1000 64

./tp12 2000 2 && ./tp12 2000 4 && ./tp12 2000 8 && ./tp12 2000 16 && ./tp12 2000 32 && ./tp12 2000 64

./tp12 5000 2 && ./tp12 5000 4 && ./tp12 5000 8 && ./tp12 5000 16 && ./tp12 5000 32 && ./tp12 5000 64

./tp12 10000 2 && ./tp12 10000 4 && ./tp12 10000 8 && ./tp12 10000 16 && ./tp12 10000 32 && ./tp12 10000 64

./tp12 50000 2 && ./tp12 50000 4 && ./tp12 50000 8 && ./tp12 50000 16 && ./tp12 50000 32 && ./tp12 50000 64

**Saídas**

**Como Executar**

Para execução do código, basta compilar no terminal e executar com três parâmetros na forma:

./<arquivo-executavel> <tamanho do vetor> <quantidade de threads>

**Tabela de Execução**

Foi criado duas tabelas para a execução do código. A primeira tabela se refere aos tempos de execução para thread *main* e as entradas na primeira linha da horizontal. A segunda refere-se aos tempos de execução utilizando se threads. Na primeira coluna vertical da segunda tabela, temos as quantidades de threads executadas, e na primeira linha vertical temos as entradas utilizadas.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 500 | 1000 | 2000 | 5000 | 10000 | 50000 |
| Tempo | 3.000000 | 4.000000 | 9.000000 | 23.000000 | 88.000000 | 186.000000 |

Tabela 1. Tempo de execução para cada entrada para a thread *main.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 500 | 1000 | 2000 | 5000 | 10000 | 50000 |
| 2 | 249.000000 | 89.000000 | 121.000000 | 110.000000 | 261.000000 | 705.000000 |
| 4 | 312.000000 | 84.000000 | 160.000000 | 121.000000 | 320.000000 | 447.000000 |
| 8 | 492.000000 | 179.000000 | 201.000000 | 346.000000 | 551.000000 | 499.000000 |
| 16 | 557.000000 | 284.000000 | 310.000000 | 437.000000 | 3712.000000 | 609.000000 |
| 32 | 2540.000000 | 551.000000 | 546.000000 | 996.000000 | 1126.000000 | 1567.000000 |
| 64 | 2118.00000 | 1070.000000 | 1104.000000 | 3354.000000 | 2222.000000 | 2975.000000 |

Tabela 2. Tempo de execução para cada entrada com *k* threads.

É interessante perceber, que conforme são executadas mais threads, o tempo de execução é aumentado.