UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Fábio A. Bocampagni

Computação Concorrente

Relatório do segundo laboratório

RIO DE JANEIRO
2023

O programa prime.c pode ser .encontrado aqui

1. Instruções

Para executar o prime.c basta digitarmos no shell o seguinte comando:

gcc -o prime prime.c -pthread -lm

Lembre-se de clonar, caso necessário, o repositório inteiro, pois há dependências, como o arquivo timer.h que define as sub rotinas utilizadas para a marcação temporal do prime.c.

O binário pode receber uma combinação de parâmetros, porém, o comportamento de controle por trás dele espera duas constantes: Valor do intervalo de N, número de Threads. Caso algum desses valores não seja fornecido, será utilizado o valor padrão definido em tempo de construção do código: 1000 passos e o valor das threads será igual ao número da quantidade de processadores da máquina que executará o prime.c

Por exemplo, se executarmos:

./prime 10000000

Como omitimos o valor que refere-se a quantidade de threads, será utilizado o valor de retorno da subrotina get_nprocs da sysinfo.h. No **meu** caso, 12.

Novamente, se executamos:

./prime

Nesse caso, foi omitido tanto o número de passos quanto o número de threads. Logo, será usado, como mencionado anteriormente, 1000 como número de passos e número da quantidade de processadores da máquina como número de threads.

2. Relatório de execução do prime.c

./prime 1000 1

```
A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000186 para 1 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000284 para 1 threads
```

```
A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000184 para 1 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000203 para 1 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000190 para 1 threads
```

./prime 1000 2

```
A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000200 para 2 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000217 para 2 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000195 para 2 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000214 para 2 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000194 para 2 threads
```

./prime 1000 3

```
A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000205 para 3 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000246 para 3 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000199 para 3 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000215 para 3 threads

A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000245 para 3 threads
```

```
A quantidade de números primos entre 0 e 1000 é: 168
Tempo de execução: 0.000244 para 3 threads
```

Nota-se, de forma empírica que, como o número N é muito pequeno, não existe um ganho relativo ao aumento do número de threads. Com 3 threads o prime.c consegue no máximo ser tão eficiente quanto com 1 e 2 threads, notando claramente um ponto de inflexão em sua performance dada o parâmetro fixo N. Obviamente, se aumentarmos o valor de N, aumentando assim o valor da carga de trabalho por unidade de threads, voltaríamos a ver uma melhora da relação tempo/número de threads.

3. Aumentando a quantidade de carga

./prime 1000000 1

```
A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.099187 para 1 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.099092 para 1 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.098221 para 1 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.098999 para 1 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.098008 para 1 threads
```

./prime 1000000 2

```
A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.060975 para 2 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.061151 para 2 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.061673 para 2 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
```

```
Tempo de execução: 0.061130 para 2 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.065395 para 2 threads
```

./prime 1000000 3

```
A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.043093 para 3 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.042716 para 3 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.042405 para 3 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.047789 para 3 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.042805 para 3 threads
```

Como mencionado, com o aumento do número de carga por threads, podemos ver que o aumento de threads realmente é benéfico e melhora o tempo de execução.

./prime 1000000 4

```
A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.033544 para 4 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.035578 para 4 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.036797 para 4 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.036847 para 4 threads

A quantidade de números primos entre 0 e 1000000 é: 78498
Tempo de execução: 0.036219 para 4 threads
```

4. Aceleração e Eficiência

Para calcularmos a aceleração e a eficiência alcançada em cada configuração, podemos construir uma tabela e seguir a seguinte ordem de construção:

- Conf - Configuração
- N - Representa o intervalo da variável N, ou quantidade de carga.
- $threads$ - Número de threads
- $T_s(S)$ - Tempo sequencial, ou tempo com uma threads.
- $T_C(S)$ - Tempo concorrente.
- A - Aceleração
- E - Eficiência

Conf	N	$threads$	$T_s(S)$	$T_C(S)$	A	E
1	10^3	1	0.000186	0.000186	1	1
2	10^3	2	0.000186	0.000200	0,93	0,465
3	10^3	3	0.000186	0.000205	0,90	0,30
4	10^6	1	0.099187	0.099187	1	1
5	10^6	2	0.099187	0.060975	1,62	0,81
6	10^6	3	0.099187	0.043093	2,30	0,76
7	10^6	4	0.099187	0.033544	2,95	0,73