Relatório de Sistemas Operacionais:

Abordagens para multiplicação concorrente de matrizes

Erick Pires, Vítor Trindade

DRE 112024142,112051644

Resumo. *Este relatório compara duas abordagens diferentes para multiplicação concorrente de matrizes, utilizando múltiplas threads e utilizando múltiplos subprocessos. Além disso, uma versão sequencial do algoritmo também é comparada, para noções de ganho de performance. Também é analizado o inicio e fim de cada thread ou processo para entender o esquema de escalonamento empregado pelo sistema.*

1. Objetivo

O objetivo é a multiplicação de duas matrizes NxN, com uma implementação feitas através de subprocessos, uma através de threads. Foi feita também uma implementação sequencial por motivos de comparação.

As duas matrizes são preenchidas com números aleatórios gerados com a função `rand` biblioteca padrão.

As medições de tempo foram feitas com a biblioteca `timer.h`, desenvolvida por Peter Pacheco e anexada a este arquivo. Somente o tempo utilizado para multiplicar as matrizes é considerado.

2. A implementação em Threads

Na implementação em Threads as matrizes são variáveis globais. As threads são alocadas nas linhas

```
       pthread_t* threads = (pthread_t*) malloc(matrix_size *
sizeof(pthread_t));
     int* tids = (int*) malloc(matrix_size * sizeof(int));
```

E iniciam nas linhas

```
     for(int thread_id = 0; thread_id < matrix_size;
thread_id++) {
          tids[thread_id] = thread_id;
          pthread_create(threads + thread_id, NULL,
pthread_mul_matrix, (void*) (tids + thread_id)); }
```

A threads criadas executam a função `pthread_mul_matrix`, que chama a função `multiply_matrix` que realiza a multiplicação de exatamente uma linha da matriz A de entrada, sendo então sempre criadas N threads. O tempo de execução começa a ser contado antes desse for e para a contagem após os joins.

3. A implementação em subprocessos

Na implementação em subprocessos as matrizes precisam ser alocadas pela função `mmap` para poderem compartilhar memória já que as variáveis globais não são divididas.

Os subprocessos são criados em

```
     for(int i = 0; i < matrix_size; i++) {
      pid_t pid = fork();

      if(pid == 0) {
           multiply_matrix(i);
           return 0; // All the child processes die here
      }
    }
```

A função `fork` retorna 0 para cada um dos processos filhos, por isso só eles entram no `if` e executam `multiply_matrix(i)`, que é idêntica a função usada na implementação por threads. O tempo de execução começa a ser contado antes desse for e termina depois de que o processo pai executa `wait` para cada um dos filhos.

4. Ambiente de testes

Os teste foram executados em com computador rodando o sistema operacional Linux no kernel 4.2.5 “vanilla”. O computador possui um processador Intel® Core™ i7-3632QM CPU da geração Ivybridge. Um esquema da arquitetura de cache do processador pode ser encontrada na imagem abaixo.

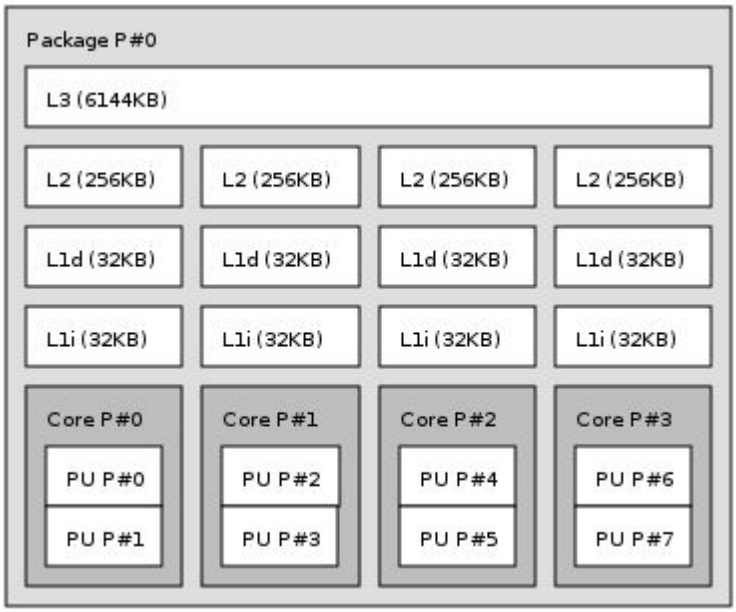

Figura 1. Cache do processador

5. Resultados

5.1. Corretude do resultado das multiplicações

O resultado da multiplicação das duas implementações (com threads e com subprocessos) foi verificado utilizando o programa `octave-cli` computando

```
(A * B) - C
```

e confirmando que a subtração é a matriz zero.

5.2. Tempos de execução da multiplicação

As três implementações foram executadas com os seguintes valores para N {3, 10, 50, 100, 200, 500}. Para cada valor as implementações foram executadas 50 vezes. Os tempos médios medidos dessas execuções são os seguintes.

Tabela 1. Tempos de execução

Tipo \ N	3	10	50	100	200	500
Sequencial	0.0000003	0.0000040	0.0005004	0.0034875	0.0296358	0.5032474
Threads	0.0036139	0.0036967	0.0043888	0.0057028	0.0165264	0.1635646
Sub-processos	0.0032096	0.0034395	0.0046996	0.0065831	0.0143784	0.1544734

Também foram gerados histogramas de frequência para cada um dos casos. Esses histogramas encontram-se na sessão 6.

5.3.Ordem de execução das unidades de execução

Para verificar a ordem de execução de cada unidade de execução, cada unidade armazena o instante em que começou a executar e o instante em que terminou de executar e por fim imprime esses valores na saída padrão em formato `CSV`. Esta abordagem foi utilizada em vez de imprimir antes e depois da multiplicação para garantir que a system call utilizada para imprimir não iria atrapalhar na medição.

Os dados foram gerados utilizando o parâmetro N = 50 e utilizados para gerar os gráficos de escalonamento abaixo. No eixo das abscissas temos o tempo em segundos e no eixo das ordenadas temos o identificador de cada unidade de processamento.

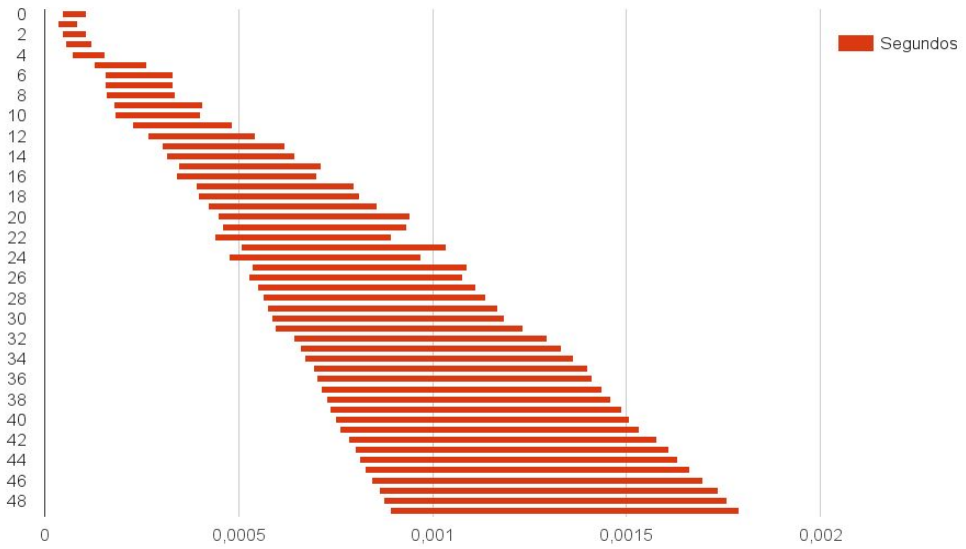

Figura 2. Início e fim das threads

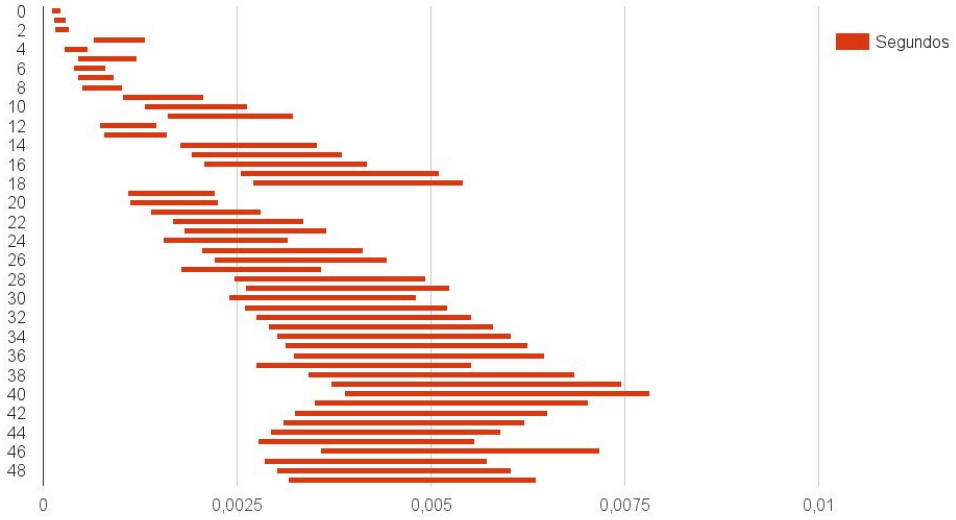

Figura 3. Início e fim dos subprocessos

5. Conclusões

A ordem de inícios das threads são bem lineares, em parte devido ao fato de terem tids sequenciais bem definidos, e possuem escalonamento justo e relativamente previsível.

Subprocessos, embora justos, são menos previsíveis, por isso com N = 3, 10 ele é mais rápido que threads, mas fica atrás com N= 50, 100, e volta a ganhar com N= 200, 300. Ainda assim a diferença de tempos entre threads e subprocessos é muito pequena, se comparado a sequencial. Com N menores que 200 o tempo ganho dividindo a execução não é o suficiente para compensar o tempo perdido criando e alocando as estruturas.

6. Apêndice

6.1. Tempos de subprocessos

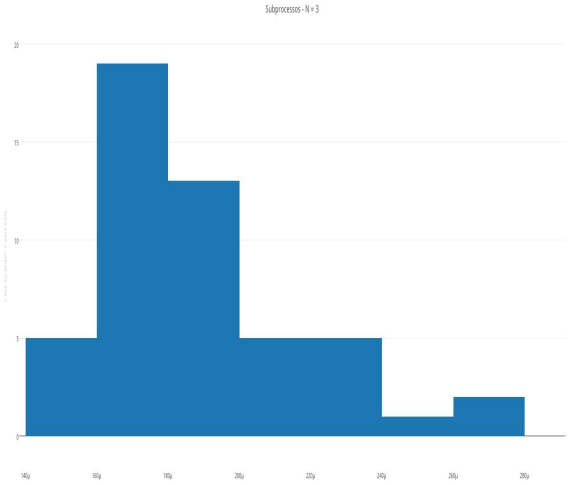

Figura 4. N = 3

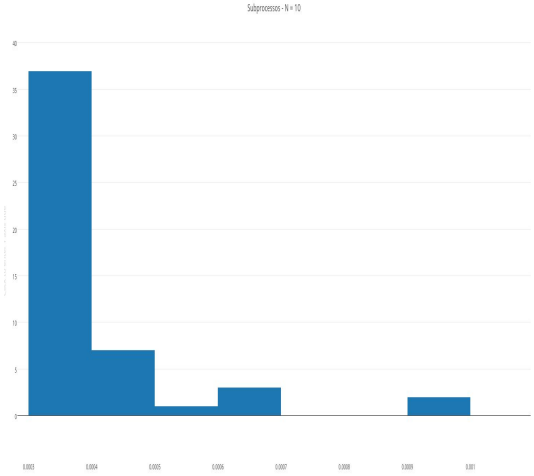

Figura 5. N = 10

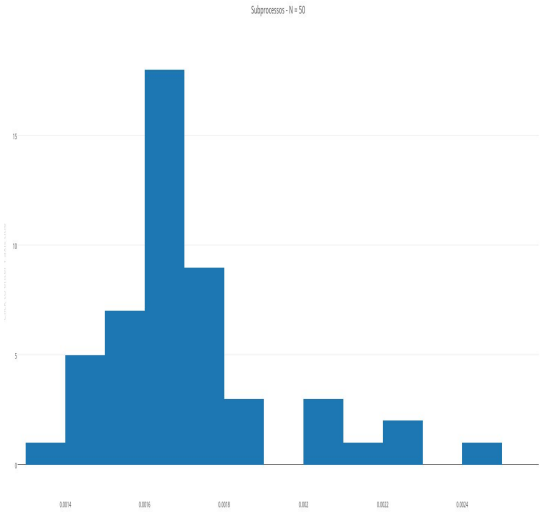

Figura 6. N = 50

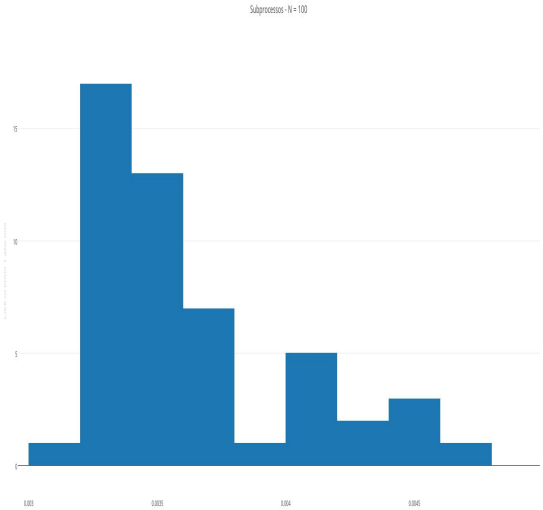

Figura 7. N = 100

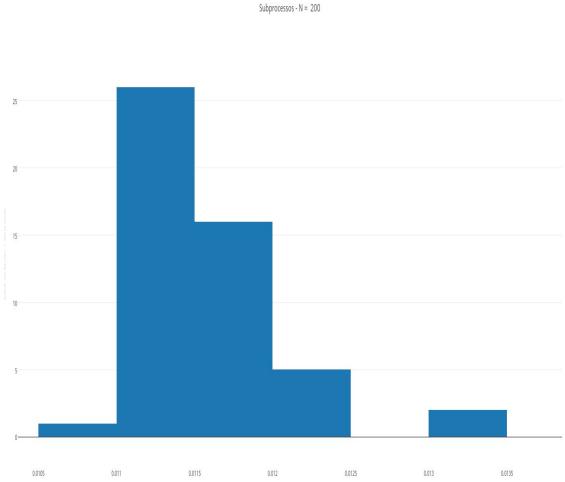

Figura 8. N = 200

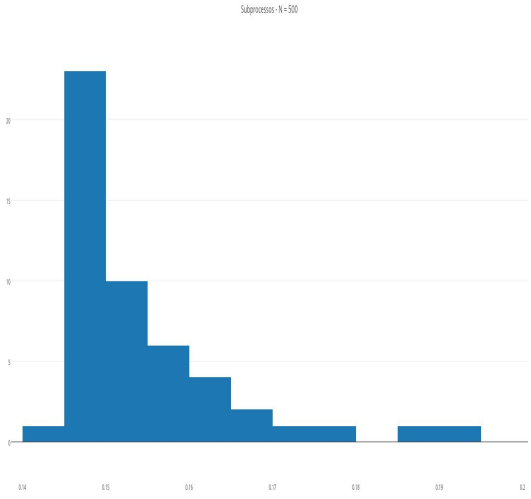

Figura 9. N = 500

6.2. Tempos de threads

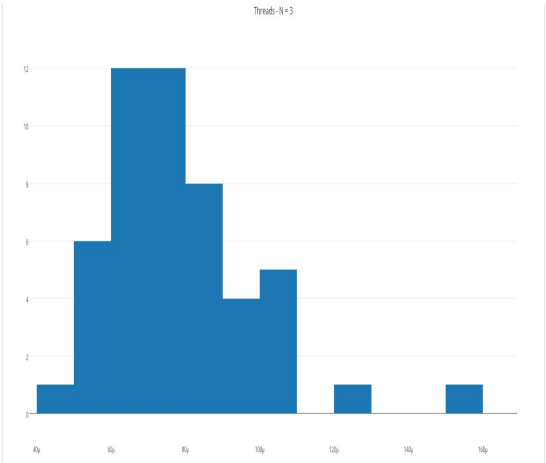

Figura 10. N = 3

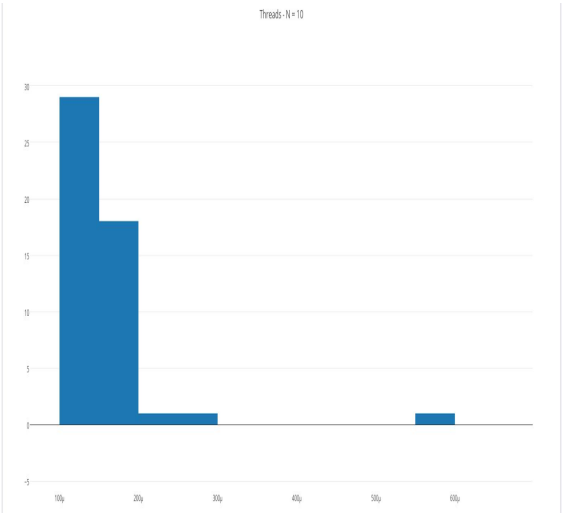

Figura 11. N = 10

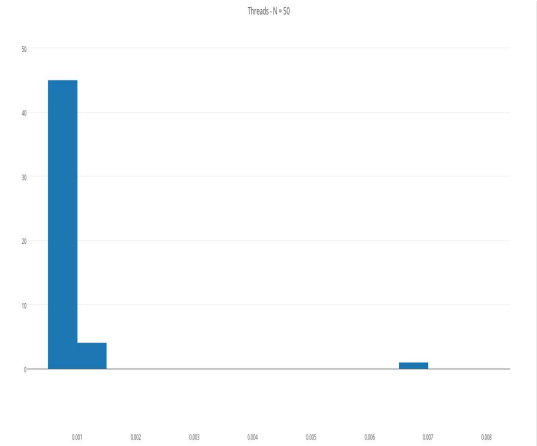

Figura 12. N = 50

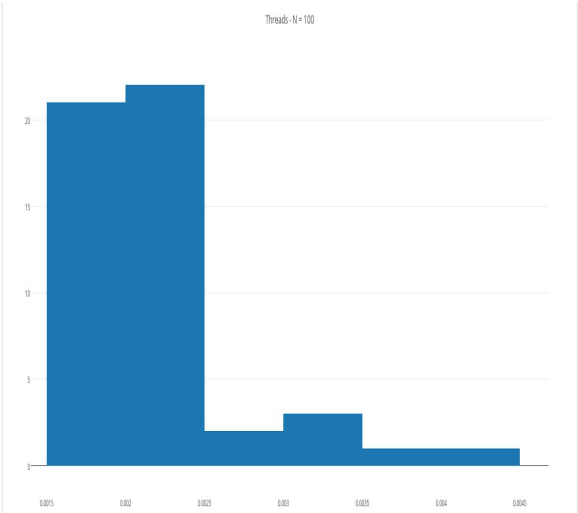

Figura 13. N = 100

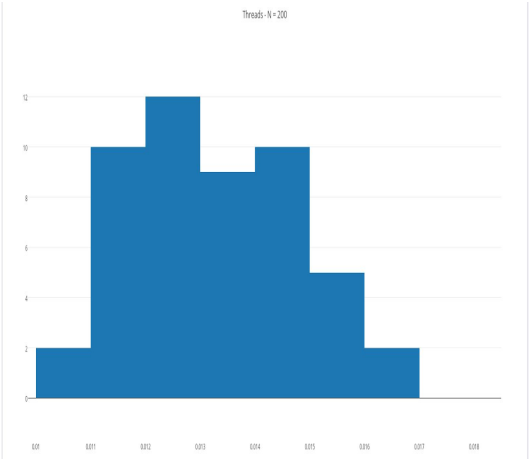

Figura 14. N = 200

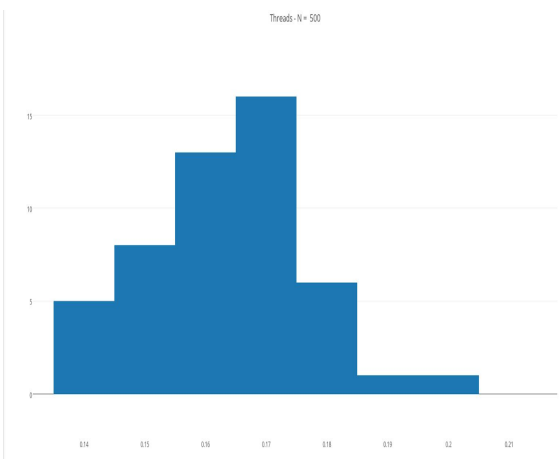

Figura 15. N = 500

6.2. Tempos sequencial

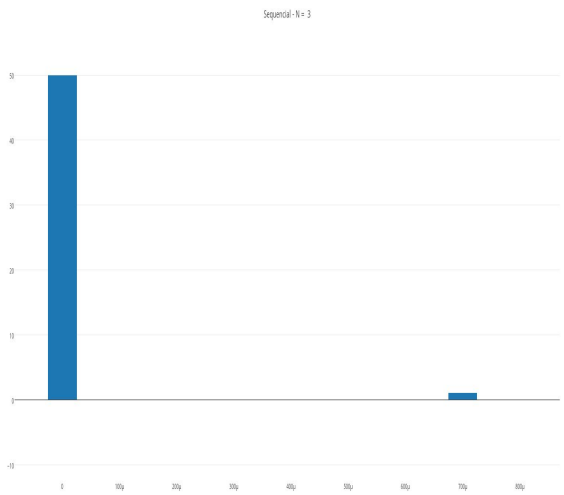

Figura 16. N = 3

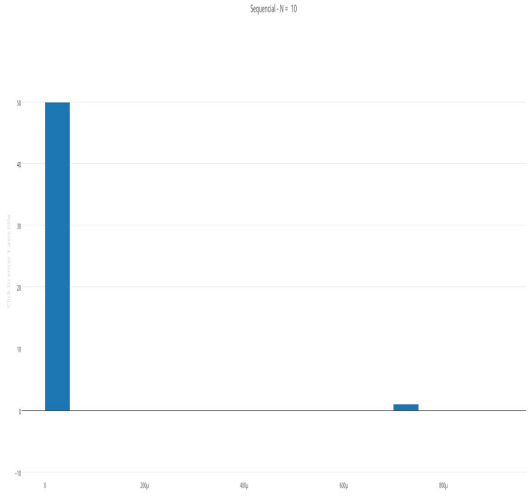

Figura 17. N = 10

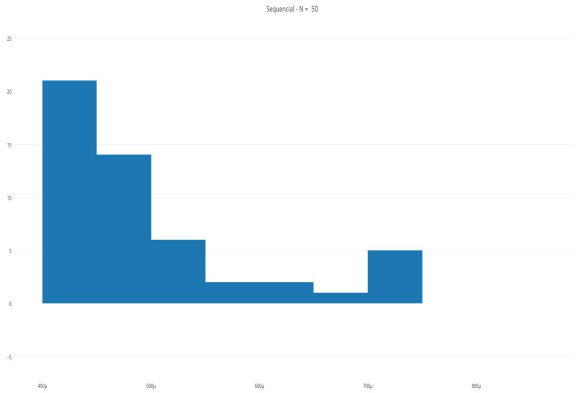

Figura 18. N = 50

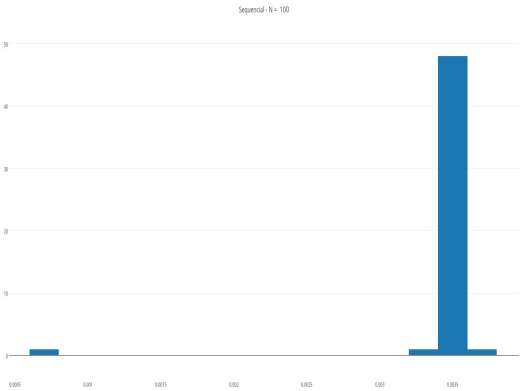

Figura 19. N = 100

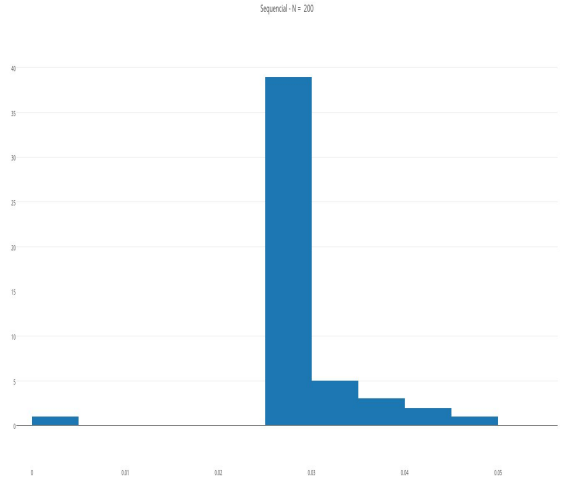

Figura 20. N = 200

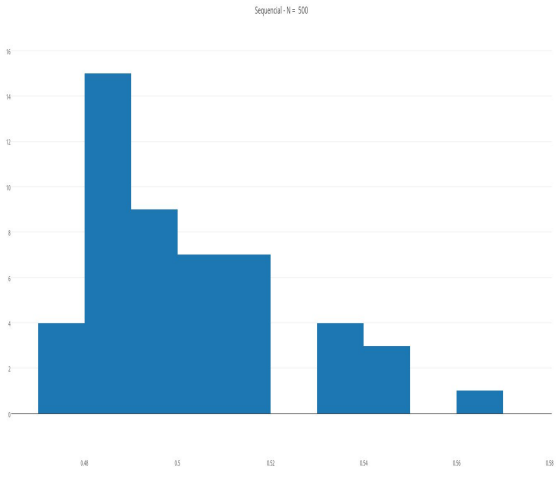

Figura 21. N = 500

7. Referências

http://stackoverflow.com/questions/13274786/how-to-share-memory-between-process-fork, como compartilhar memória entre subprocessos

[Peter Pacheco], http://www.cs.usfca.edu/~peter/cs625/code/timer.h, biblioteca timer.h.