Projeto Final – Introdução à Programação Paralela e Distribuída

15th Marathon of Parallel Programming: Problem E – “Yet More Primes”

Amanda Oliveira
Bianca Santos Pastos
Fauzi Asbahr
Matheus Eiji Faria Komatsu

Rio Claro, 04 de Dezembro de 2024

1 – Problema escolhido

Ao buscar uma aplicação para ser paralelizada, encontramos dentre os inúmeros problemas presentes na Maratona de Programação Paralela da 20ª Edição o problema E – “Yet More Primes”, no qual fornecemos um número P (entre 1 e 10.000) de números primos seguido de 2P linhas de números quaisquer que serão combinados entre si para formar números primos. O resultado é uma lista de P números primos em ordem crescente.

Input	Output
4 50415 100 5041 55750 5039 01 0213 55713	1000213 5041501 5575001 504155039 504155713

Figura 1: exemplo com 4 números primos

O algoritmo é simples: após receber a lista dos números a serem combinados, ele a percorre separando os números em duas variáveis, “firstHalf” e “secondHalf”, e concatenando-as em uma única variável, a qual é recebida pela função *isprime*, que verificará se o número gerado pelas duas metades é ou não primo. Ao terminar de percorrer a lista, ele realiza a função *quicksort* para ordenar os resultados em ordem crescente.

```
for (i = 0; i < numPrimes; i++)
  for (j = 0; j < numPrimes; j++)
  {
    strcpy(strToTest, firstHalf[i]);
    strcat(strToTest, secondHalf[j]);
    primeToTest = atol(strToTest);
    if (isprime(primeToTest))
    {
      result[numResults++] = primeToTest;
    }
  }
quicksort(result, 0, numResults - 1);
for (i = 0; i < numResults; i++)
  printf("%ld\n", result[i]);
```

Figura 2: trecho do código não paralelizado

2 – Paralelização do problema

De início foi evidente que o segredo para paralelizar o dado problema estava no loop responsável por percorrer a lista de números a serem concatenados para formar o possível primo. Essa tarefa poderia ser dividida em threads para agilizar a geração de possíveis primos tal que, após cada thread realizar a função *isprime* com os números que ela possui acesso, os resultados obtidos são agregados em uma única lista para posterior ordenação com *quicksort*. Nossa paralelização foi feita utilizando a api OpenMP devido ao costume e facilidade de implementação comparado ao Pthreads.

Utilizamos o pragma **omp parallel private** para declarar que certas variáveis dentro do bloco paralelizado terão escopo privado, ou seja, seus valores não serão compartilhados entre as threads, garantindo o comportamento esperado na leitura da lista de números. caso contrário, os valores de "localResults" seriam utilizadas por mais de uma thread ao mesmo tempo e threads diferentes teriam seus resultados salvos de forma errônea. Para as variáveis "result" e "numResults" foi usada a diretiva shared pois seus valores serão

Em seguida, foi implementado o pragma **omp for schedule (dynamic)** responsável por escalonar o laço duplo de leitura da lista de números, de forma a atribuir iterações para cada thread durante a execução (de forma dinâmica), garantindo balanceamento de carga para caso as threads recebam números de tamanhos muito distantes entre si e uma não demore muito mais que a outra para realizar seu trabalho.

```
  long int localResults[MAXRESULTS]; // Array to store thread-local results
  int localCount = 0;
  gettimeofday(&start, NULL);
#pragma omp parallel private(i, j, strToTest, primeToTest, localResults, localCount) shared(result, numResults)
  {
    localCount = 0;
#pragma omp for schedule(dynamic)
    for (i = 0; i < numPrimes; i++)
    {
      for (j = 0; j < numPrimes; j++)
      {
        strcpy(strToTest, firstHalf[i]);
        strcat(strToTest, secondHalf[j]);
        primeToTest = atol(strToTest);
        if (isprime(primeToTest))
        {
          localResults[localCount++] = primeToTest;
        }
      }
    }
```

Figura 3: trecho do código paralelizado para dividir a execução de *isprime* em threads

Após a execução de *isprime* por cada thread com todas as permutações possíveis de números testados, os resultados obtidos por cada uma deve ser armazenado em um mesmo vetor de resultados, o qual posteriormente será ordenado com *quicksort*. Esta parte do algoritmo deve ser implementada com o pragma **omp critical** para garantir exclusão mútua no momento de sincronização dos resultados, pois todas as threads poderiam tentar acessar a mesma posição no vetor de resultados já que numResults é compartilhado, isso nos obriga a garantir que as threads não poderão acessar a mesma posição de numResults antes que haja um incremento do mesmo. Finalizado esse processo, todos os primos encontrados são impressos na tela junto ao tempo gasto para a completude do programa.

```
// Combine results into the shared array
#pragma omp critical
    {
      for (int k = 0; k < localCount; k++)
      {
        result[numResults++] = localResults[k];
      }
    }
  }

  quicksort(result, 0, numResults - 1);
  for (i = 0; i < numResults; i++)
    printf("%ld\n", result[i]);
```

Figura 4: trecho do código paralelizado para combinar os resultados de cada thread em uma única lista

3 - Demonstrações

Obs: Todos os testes foram realizados em um sistema UBUNTU 20.0.4 , com 8 processadores e 16 processadores lógicos (threads).

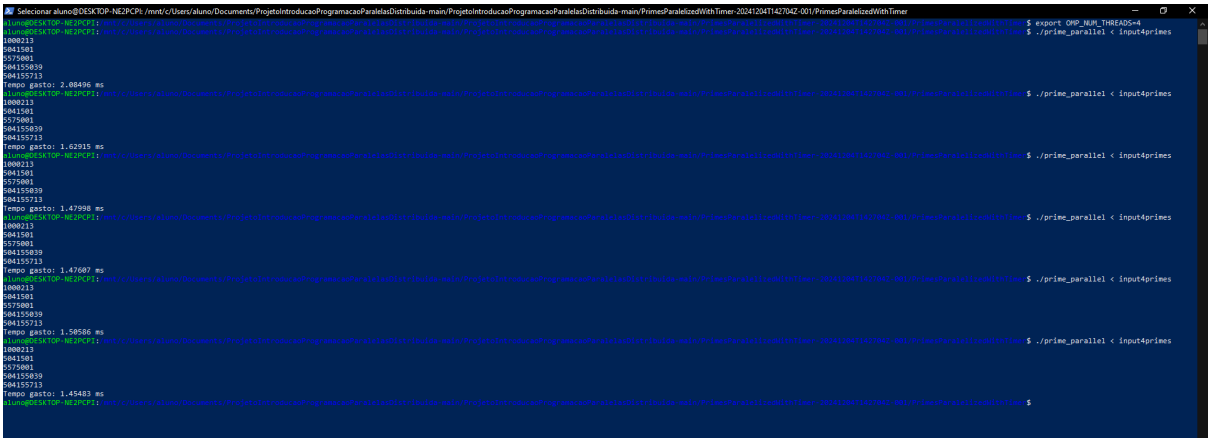
4 números primos

Input:

4
50415
100
5041
55750
5039
01
0213
55713

Paralelizado

OMP THREADS = 4



Output

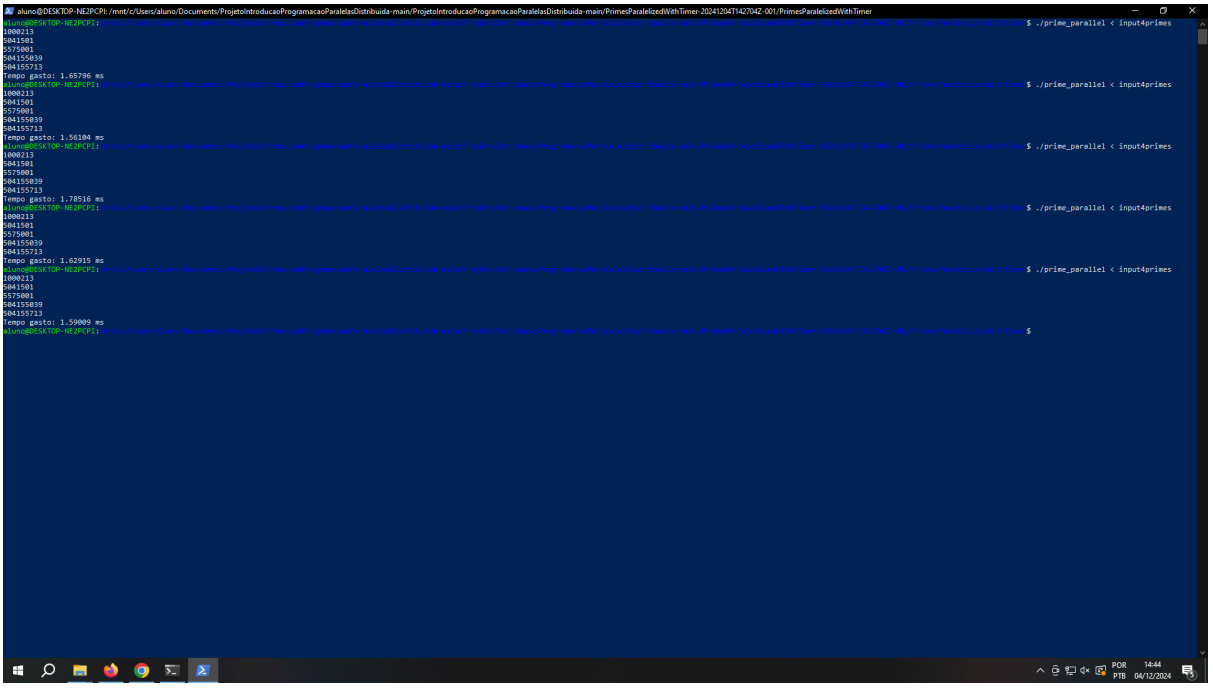
1000213	5041501	5575001	504155039	504155713

Tempo

1.47192 ms	1.45801 ms	1.63501 ms	1.5 ms	1.40698 ms

Média: 1,494384 ms

OMP THREADS = 8



Output

1000213	5041501	5575001	504155039	504155713

Tempo

1.72705 ms	1.91284 ms	1.68286 ms	1.69409 ms	1.6709 ms

Média: 1,737548 ms

Sequencial

Output

1000213	5041501	5575001	504155039	504155713

Tempo

2.84717 mss	2.88306 ms	2.948 ms	2.75098 mss	2.88208 mss

Média: 2,862258 ms

8 primos

Input

8
50415
100
5041
55750
5039
01
0213
55713
10

100
1
100
007
09
0037
39

Paralelizado

OMP THREAD = 4

Output

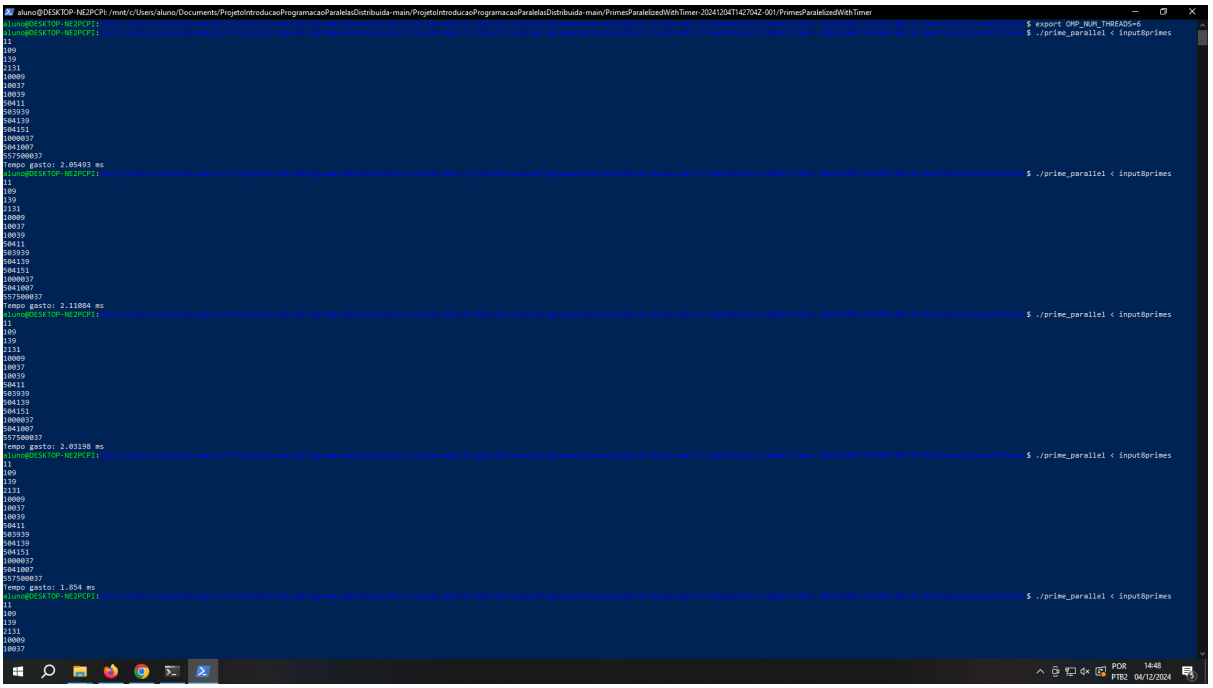
11	109	139	2131	10009
10037	10039	50411	503939	504139
504151	1000037	5041007	557500037	

Tempo

1.71118 ms	1.7771 ms	1.7019 ms	1.72705 ms	1.72705 ms

Média: 1,728856 ms

OMP THREAD = 8



Output

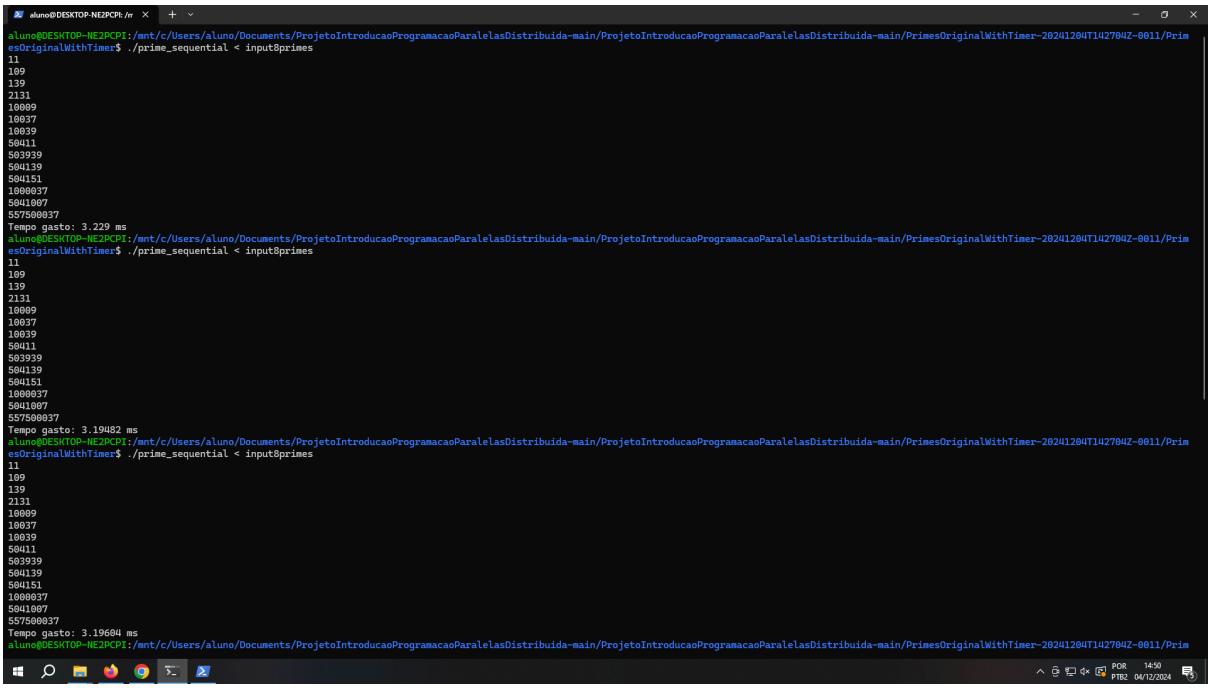
11	109	139	2131	10009
10037	10039	50411	503939	504139
504151	1000037	5041007	557500037	

Tempo

1.92505 ms	1.92798 ms	1.91895 ms	2.47485 ms	2.04102 ms

Média: 2,05757 ms

Sequencial



Output

11	109	139	2131	10009
10037	10039	50411	503939	504139
504151	1000037	5041007	557500037	

Tempo:

3.19702 ms	3.08496 mss	3.05615 ms	3.04004 ms	3.29199 ms

Média: 3,134032 ms

16 primos

Input

16
100
10
100
10
10
10
10

10
1009
1
10
101
1
101
101
1
07
009
37
039
061
067
069
079
1
0093
099
03
0111
33
39
0141

Paralelizado

OMPT_THREAD=4

Output

11	11	11	101	101
101	101	101	101	103
103	103	107	107	107
137	137	137	139	139
1009	139	1009	1009	1033
1033	1033	1033	1033	1039
1039	1039	1039	1039	1039
1039	1039	1039	1061	1061
1069	1061	1069	1069	10007
10009	10007	10009	10009	10009
10009	10009	10009	10037	10037
10039	10039	10039	10039	10039
10039	10039	10039	10039	10061
10061	10061	10061	10061	10061
10067	10061	10067	10067	10067
10067	10067	10067	10069	10069
10069	10069	10069	10079	10079

10079	10079	10079	10079	10091
10093	10093	10093	10099	10099
10099	10099	10099	10099	10103
10103	10103	10111	10111	10111
10133	10133	10133	10139	10139
10141	10139	10141	10141	100069
100907	100069	100937	101009	101009
1009061	101009	10090111		

Tempos

0.742188 ms	0.628906 ms	0.609863 ms	0.573975 ms	0.678955 ms

Média: 0,646777 ms

OMP_THREADS=8

Output

11	11	11	101	101
101	101	101	101	103
103	103	107	107	107
137	137	137	139	139

139	1009	1009	1009	1033
1033	1033	1033	1033	1039
1039	1039	1039	1039	1039
1039	1039	1039	1061	1061
1061	1069	1069	1069	10007
10007	10009	10009	10009	10009
10009	10009	10009	10037	10037
10039	10039	10039	10039	10039
10039	10039	10039	10039	10061
10061	10061	10061	10061	10061
10061	10067	10067	10067	10067
10067	10067	10067	10069	10069
10069	10069	10069	10079	10079
10079	10079	10079	10079	10091
10093	10093	10093	10099	10099
10099	10099	10099	10099	10103
10103	10103	10111	10111	10111
10133	10133	10133	10139	10139
10139	10141	10141	10141	100069
100069	100907	100937	101009	101009
101009	1009061	10090111		

Tempos

0.726074 ms	0.75708 ms	0.905029 ms	0.75708 ms	0.739014 ms

Média: 0,776855 ms

Sequencial

Output

11	11	11	101	101
101	101	101	101	101
101	103	103	103	107
107	107	137	137	137
139	139	139	1009	1009
1009	1033	1033	1033	1033
1033	1033	1039	1039	1039
1039	1039	1039	1039	1061
1061	1061	1069	1069	1069
10007	10007	10009	10009	10009
10009	10009	10009	10009	10037
10037	10039	10039	10039	10039
10039	10039	10039	10039	10061
10061	10061	10061	10061	10061
10061	10067	10067	10067	10067
10067	10067	10067	10069	10069
10069	10069	10069	10069	10069
10069	10079	10079	10079	10079
10079	10079	10079	10091	10093
10093	10093	10099	10099	10099

10099	10099	10099	10099	10103
10103	10103	10111	10111	10111
10133	10133	10133	10139	10139
10139	10141	10141	10141	100069
100069	100907	100937	101009	101009
101009	1009061	10090111		

Tempos

0.940918 ms	0.973877 ms	0.907959 ms	0.89917 ms	0.924072 ms

Média: 0,9291992 ms

32 primos

Paralelizado

OMP_THREAD=4

Output

11	101	101	101	101
101	101	101	103	103
103	103	103	103	103

103	103	103	103	103
103	103	103	103	103
103	107	107	107	107
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
109	109	109	109	109
1009	1009	1009	1009	1013
1013	1013	1013	1013	1013
1013	1013	1013	1013	1019
1019	1019	1019	1021	1021
1021	1021	1021	1039	1039
1039	1039	1039	1039	1039
1039	1039	1039	1039	1069
1069	1069	1069	1069	1069
1091	1091	1091	1091	1091
1091	1091	1091	1091	1093
1093	1093	1093	1093	1093
1093	1093	1093	1093	1093
10007	10007	10007	10007	10009
10009	10009	10009	10009	10009
10009	10009	10009	10009	10037
10037	10037	10037	10037	10037
10039	10039	10039	10039	10039
10039	10039	10039	10039	10061
10061	10061	10061	10061	10061
10061	10061	10061	10067	10067
10067	10067	10067	10067	10067
10067	10069	10069	10069	10069
10069	10069	10069	10069	10069
10069	10069	10069	10069	10079
10079	10079	10079	10079	10079
10079	10091	10091	10091	10091
10093	10093	10093	10093	10093
10093	10093	10099	10099	10099
10099	10099	10099	10099	10103
10103	10103	10103	10111	10111

10111	10111	10111	10111	10111
10133	10133	10133	10139	10139
10139	10139	10139	10139	10141
10151	10151	10151	10151	10151
10151	10151	10151	10159	10159
10159	10159	10159	10159	10163
10163	10163	10163	10169	10169
10169	10169	10169	10169	10177
10177	10177	10177	10177	10177
10181	10181	10181	10181	10181
10181	10181	10181	10193	10193
10193	10193	10193	10211	10211
10211	10211	10211	10223	10223
10223	10223	10223	10243	10243
10243	10243	10243	10243	10243
10243	10243	10247	10247	10247
10247	10247	10253	10253	10253
10253	10253	10253	10253	10259
10259	10259	10259	10259	10259
10259	10259	10267	10267	10267
10267	10267	10271	10271	10271
10271	10271	10271	10271	10273
10273	10273	10273	10273	100069
100069	100069	100069	100069	100151
100151	100151	100151	100151	100271
100271	100271	100271	100271	100391
100393	100669	100673	100693	100699
100703	100769	100799	100907	100999
101009	101009	101009	101009	101111
101111	101111	101111	101323	101359
101359	101399	101411	101603	101611
101693	101723	102061	102061	102061
102061	102139	102139	102139	102139
102181	102181	102181	102181	102253
102253	102253	102253	102253	102407
102499	102607	102611	102673	1003111
1006151	1006253	1007243	1009061	1009139

Tempos

1.20117 ms	1.30713 ms	1.46899 ms	1.15381 ms	1.198 ms

Média: 1,26582 ms

OMP_THREAD=8

Output

11	101	101	101	101
101	101	101	103	103
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
103	107	107	107	107
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
109	109	109	109	109
1009	1009	1009	1009	1013
1013	1013	1013	1013	1013
1013	1013	1013	1013	1019

1019	1019	1019	1021	1021
1021	1021	1021	1039	1039
1039	1039	1039	1039	1039
1039	1039	1039	1039	1069
1069	1069	1069	1069	1069
1091	1091	1091	1091	1091
1091	1091	1091	1091	1093
1093	1093	1093	1093	1093
1093	1093	1093	1093	1093
10007	10007	10007	10007	10009
10009	10009	10009	10009	10009
10009	10009	10009	10009	10037
10037	10037	10037	10037	10037
10039	10039	10039	10039	10039
10039	10039	10039	10039	10061
10061	10061	10061	10061	10061
10061	10061	10061	10067	10067
10067	10067	10067	10067	10067
10067	10069	10069	10069	10069
10069	10069	10069	10069	10069
10069	10069	10069	10069	10079
10079	10079	10079	10079	10079
10079	10091	10091	10091	10091
10093	10093	10093	10093	10093
10093	10093	10099	10099	10099
10099	10099	10099	10099	10103
10103	10103	10103	10111	10111
10111	10111	10111	10111	10111
10133	10133	10133	10139	10139
10139	10139	10139	10139	10141
10151	10151	10151	10151	10151
10151	10151	10151	10159	10159
10159	10159	10159	10159	10163
10163	10163	10163	10169	10169
10169	10169	10169	10169	10177
10177	10177	10177	10177	10177
10181	10181	10181	10181	10181

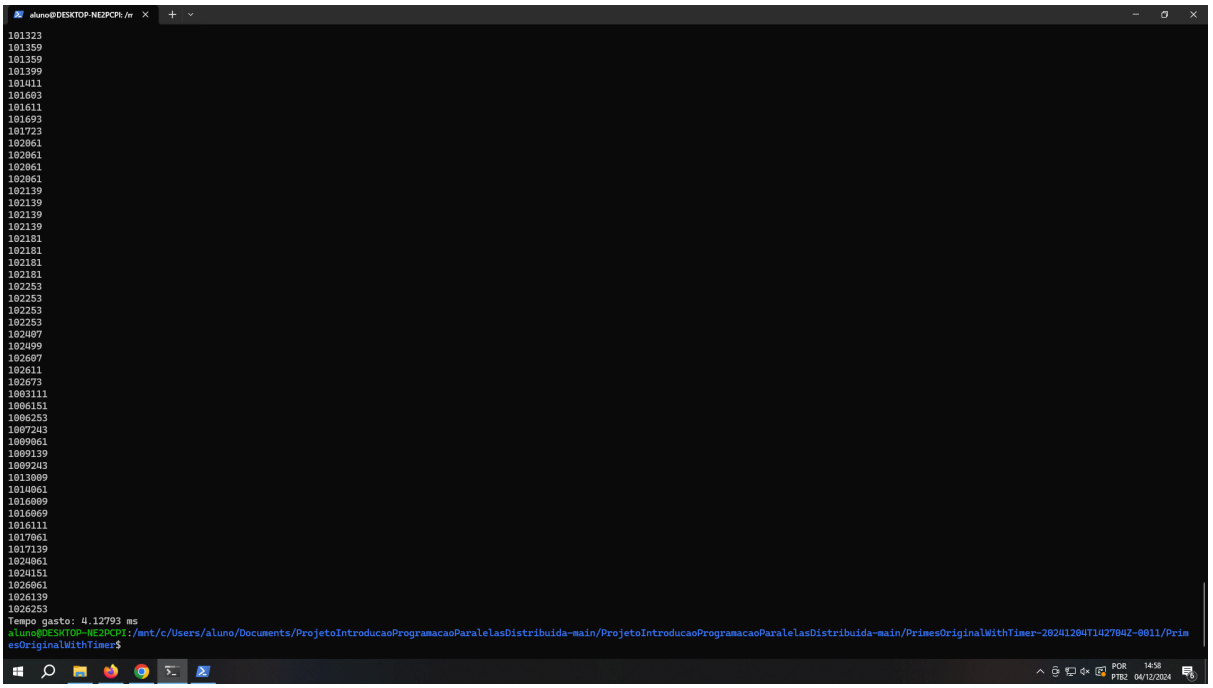
10181	10181	10181	10193	10193
10193	10193	10193	10211	10211
10211	10211	10211	10223	10223
10223	10223	10223	10243	10243
10243	10243	10243	10243	10243
10243	10243	10247	10247	10247
10247	10247	10253	10253	10253
10253	10253	10253	10253	10259
10259	10259	10259	10259	10259
10259	10259	10267	10267	10267
10267	10267	10271	10271	10271
10271	10271	10271	10271	10273
10273	10273	10273	10273	100069
100069	100069	100069	100069	100151
100151	100151	100151	100151	100271
100271	100271	100271	100271	100391
100393	100669	100673	100693	100699
100703	100769	100799	100907	100999
101009	101009	101009	101009	101111
101111	101111	101111	101323	101359
101359	101399	101411	101603	101611
101693	101723	102061	102061	102061
102061	102139	102139	102139	102139
102181	102181	102181	102181	102253
102253	102253	102253	102253	102407
102499	102607	102611	102673	1003111
1006151	1006253	1007243	1009061	1009139

Tempos

1.00806 ms	1.18896 ms	1.1792 ms	1.06592 ms	1.14917 ms

Média: 1,118262 ms

Sequencial



Output

101	101	101	101	101
101	101	101	101	101
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
103	103	103	103	103
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
107	107	107	107	107
109	109	109	109	109
109	109	109	109	109

1009	1009	1009	1009	1013
1013	1013	1013	1013	1013
1013	1013	1013	1013	1019
1019	1019	1019	1019	1021
1021	1021	1021	1021	1039
1039	1039	1039	1039	1039
1039	1039	1039	1039	1039
1039	1039	1039	1039	1069
1069	1069	1069	1069	1069
1069	1069	1069	1069	1069
1091	1091	1091	1091	1091
1091	1091	1091	1091	1091
1093	1093	1093	1093	1093
1093	1093	1093	1093	1093
10007	10007	10007	10007	10009
10009	10009	10009	10009	10009
10009	10009	10009	10009	10037
10037	10037	10037	10037	10037
10039	10039	10039	10039	10039
10039	10039	10039	10039	10061
10061	10061	10061	10061	10061
10061	10061	10061	10061	10067
10067	10067	10067	10067	10067
10067	10069	10069	10069	10069
10069	10069	10069	10069	10069
10069	10069	10069	10069	10079
10079	10079	10079	10079	10079
10079	10091	10091	10091	10091
10093	10093	10093	10093	10093
10093	10093	10099	10099	10099
10099	10099	10099	10099	10103
10103	10103	10103	10103	10111
10111	10111	10111	10111	10111
10111	10111	10111	10111	10133
10133	10133	10133	10139	10139
10139	10139	10139	10139	10139
10139	10139	10139	10139	10139

10139	10139	10139	10139	10151
10151	10151	10151	10151	10151
10151	10151	10151	10151	10159
10159	10159	10159	10159	10159
10159	10159	10159	10163	10163
10163	10163	10163	10169	10169
10169	10169	10169	10169	10177
10177	10177	10177	10177	10177
10181	10181	10181	10181	10181
10181	10181	10181	10181	10193
10193	10193	10193	10193	10211
10211	10211	10211	10211	10223
10223	10223	10223	10223	10243
10243	10243	10243	10243	10243
10243	10243	10243	10247	10247
10247	10247	10247	10247	10253
10253	10253	10253	10253	10253
10253	10253	10253	10253	10259
10259	10259	10259	10259	10259
10259	10259	10259	10259	10267
10267	10267	10267	10267	10267
10271	10271	10271	10271	10271
10271	10271	10271	10271	10273
10273	10273	10273	10273	100069
100069	100069	100069	100069	100151
100151	100151	100151	100151	100271
100271	100271	100271	100271	100391
100393	100669	100673	100693	100699
100703	100769	100799	100907	100999
101009	101009	101009	101009	101111
101111	101111	101111	101323	101359
101359	101399	101411	101603	101611
101693	101723	102061	102061	102061
102061	102139	102139	102139	102139
102181	102181	102181	102181	102253
102253	102253	102253	102253	102407
102499	102607	102611	102673	1003111

1006151	1006253	1007243	1009061	1009139

Tempos

3.87207 ms	3.50903 ms	3.62109 ms	3.56909 ms	3.64185 ms

Média: 3,642626 ms

4 - Resultados e conclusões

Tendo em vista as médias de tempo gastas em cada programa, podemos obter o speedup de cada paralelização em relação ao tempo sequencial, sabendo que Speedup = tempo sequencial / tempo paralelo. Logo:

Para input de 4 números primos:
Speed up Paralelas 4 OMP_THREADS = 2,862258/1,494384 = 1,915
Speed up Paralelas 8 OMP_THREADS = 2,862258/1,73754 = 1,647

Para input de 8 números primos:
Speed up Paralelas 4 OMP_THREADS = 3,134032/1,728856 = 1,813
Speed up Paralelas 8 OMP_THREADS =3,134032/2,05757 = 1,523

Para input de 16 números primos:
Speed up Paralelas 4 OMP_THREADS = 0,9291992/ 0,646777 = 1,437
Speed up Paralelas 8 OMP_THREADS = 0,9291992/ 0,776855 = 1,196

Para input de 32 números primos:
Speed up Paralelas 4 OMP_THREADS = 3,642626/1,26582 = 2,878
Speed up Paralelas 8 OMP_THREADS = 3,642626/1,118262 = 3,257

Portanto, fica claro que o problema em questão pode ser paralelizado com grande sucesso, dado que em todos os casos, o tempo de execução do programa paralelo foi menor que o sequencial. Curiosamente, o speedup do programa com OMP_THREADS 8 foi, na maioria dos casos, menor que o com OMP_THREADS 4. Isso se deve, provavelmente, à configuração do computador em que foram realizados os testes.

Concluindo, pode-se entender que o problema foi paralelizado com sucesso.