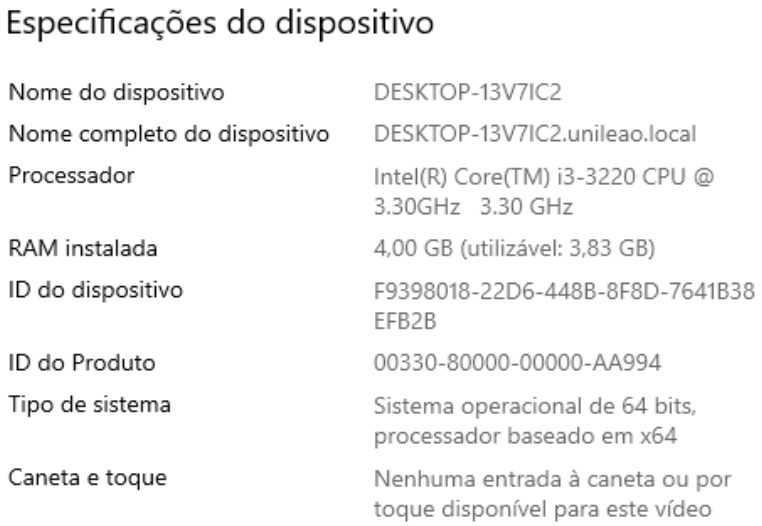
Universidade Federal do Cariri
Ciências da Computação

Crivo de Eratóstenes

Ariane Kevinny Muniz Ribeiro
Matrícula: 2019002827

Link GitHub para acesso: [ArianeKevinny/Sieve-of-Eratosthenes-with-OpenMP-and-MPI]

Informações Técnicas



Especificações do dispositivo

Nome do dispositivo	DESKTOP-13V7IC2
Nome completo do dispositivo	DESKTOP-13V7IC2.unileao.local
Processador	Intel(R) Core(TM) i3-3220 CPU @ 3.30GHz 3.30 GHz
RAM instalada	4,00 GB (utilizável: 3,83 GB)
ID do dispositivo	F9398018-22D6-448B-8F8D-7641B38EFB2B
ID do Produto	00330-80000-00000-AA994
Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
Caneta e toque	Nenhuma entrada à caneta ou por toque disponível para este vídeo

Processador: Intel(R) Core(TM) i3-3220 CPU @ 3.30GHz 3.30 GHz

RAM instalada: 4,00 GB (utilizável: 3,83 GB)

Tipo de sistema: Sistema operacional de 64 bits, processador baseado em x64

Sistema Operacional: Windows

Número de Threads: 4

Número de núcleos: 2

O seguinte relatório tem como objetivo discutir as diferentes implementações do Crivo de Erastótenes, assim como analisar seus respectivos desempenhos para entradas de no mínimo 10.000.000 (10 milhões).

Primeiro, apresento a tela main.c, ela tem como função chamar as respectivas maneiras de resolver o Crivo de Eratóstenes (serial, paralelizada com OpenMP e paralelizada com openMP), além de realizar alguns cálculos relacionados a desempenho (Speedup e Eficiência), Porém por motivos de otimização certas partes de análise de desempenho ficam dentro dos arquivos específicos;

Segue abaixo o código do arquivo “main.c”, com a função “tempo” responsável pela análise de desempenho da paralelização do Crivo de Eratóstenes, que recebe como parâmetro três variáveis do tipo *double,* respectivamente, o tempo de execução serial, o tempo início da paralelização e o tempo final da paralelização, e a função main() que chama a solução de acordo com a escolha do usuário, além de chamar as funções que foram definidas nos arquivos "serial.h", "openMP.h" e “MPI.h", e a função “tempo”.

```c
#include "serial.h" //eratostenes

#include "openMP.h" //eratostenesOpenMP

#include "MPI.h" //eratostenesMPI



#include <stdio.h>

#include <stdlib.h>

#include <time.h>



int tempo(double t_serial, double inicio, double fim){



    double t_paralelo = fim - inicio;

    double speedup = t_serial/t_paralelo;
```

```
    printf("Execucao paralela(s): %f\n", t_paralelo);
    printf("Speedup: %.4f\n", speedup);
    printf("Eficiencia: %.4f\n",speedup/4.0);

}

int main(){
    clock_t t; //variável para armazenar tempo
    int ultimoNumero;
    int result;
    double inicio;
    double fim;
    double t_serial;
    int escolha;

    printf("Selecione: \n");
    printf("1 -> Serial \n");
    printf("2 -> Paralelização com openMP \n");
    printf("2 -> Paralelização com MPI \n");
    printf("Selecione:");
    scanf("%d", &escolha);

    if(escolha == 1){
```

```
        printf("Solução Serial! \n");


        printf("Qual o ultimo valor do intervalo desejado: ");
        scanf("%d", &ultimoNumero);


        printf("\n Resultado: ");


        inicio = clock(); //armazena tempo
        result = eratostenes(ultimoNumero);
        fim = clock(); //tempo final - tempo inicial


        t_serial = fim - inicio;


        printf("%d \n", result);
                    printf("Tempo de execucao Serial: %lf ms \n",
((double)t_serial)/((CLOCKS_PER_SEC/1000))); //milissegundos


    }


    if(escolha == 2){


        printf("Solução com OpenMP! \n");


        printf("Qual o ultimo valor do intervalo desejado: ");
        scanf("%d", &ultimoNumero);
```

```
        printf("Resultado: ");

        inicio = clock(); //armazena tempo
        result = eratostenesOpenMP(ultimoNumero);
        fim = clock() - inicio; //tempo final - tempo inicial

        printf("%d \n", result);

        tempo(t_serial, inicio, fim);
    }

    if(escolha == 3){

        printf("Solução com MPI! \n");

        printf("Resultado: ");

        inicio = clock(); //armazena tempo
        result = eratostenesMPI();
        fim = clock() - inicio; //tempo final - tempo inicial

        printf("%d \n", result);
```

```
        printf("Analise do Tempo: \n");

        tempo(t_serial, inicio, fim);



    }

    return ("\n FIM!");

}
```

Primeiramente, temos o código com uma solução serial na linguagem C, utilizando-se apenas das funções básicas da linguagem. Esse algoritmo funciona da seguinte forma: O termo ‘crivo’ faz analogia a uma peneira, logo podemos entender que o Crivo de Erastótenes tem como objetivo separar algum fragmento de informação, neste sendo especificamente o número menor ou igual a um inteiro positivo de valor qualquer (n).

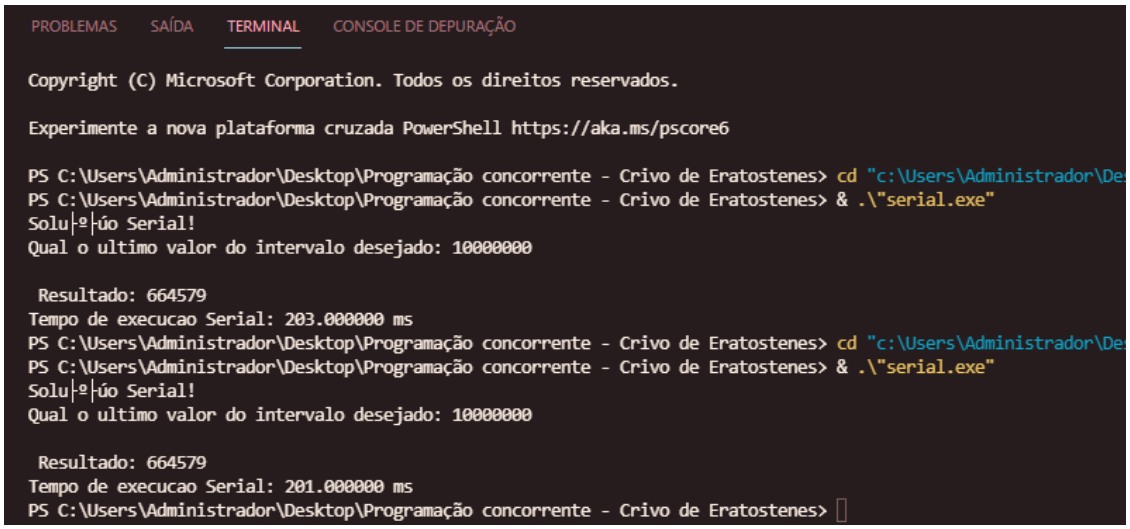
Para chegar a isso, o algoritmo recebe uma lista com números naturais de 2 a n. Após receber a lista, o algoritmo irá caminhar pela lista removendo os múltiplos do valor onde ele está atualmente. Exemplos, ao chegar na posição 1 da lista, ele encontrará o valor 2, e irá remover os múltiplos de 2 que estão nas posições seguintes. Ele irá parar de realizar essa remoção quando chegar em um valor que o seu quadrado irá ultrapassar o valor de n.

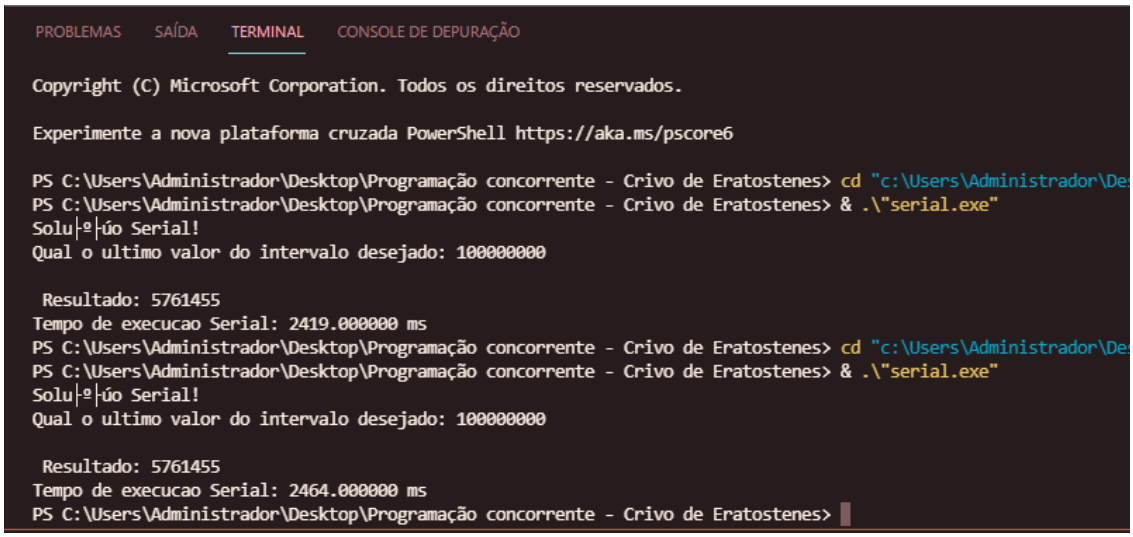
Segue o código:

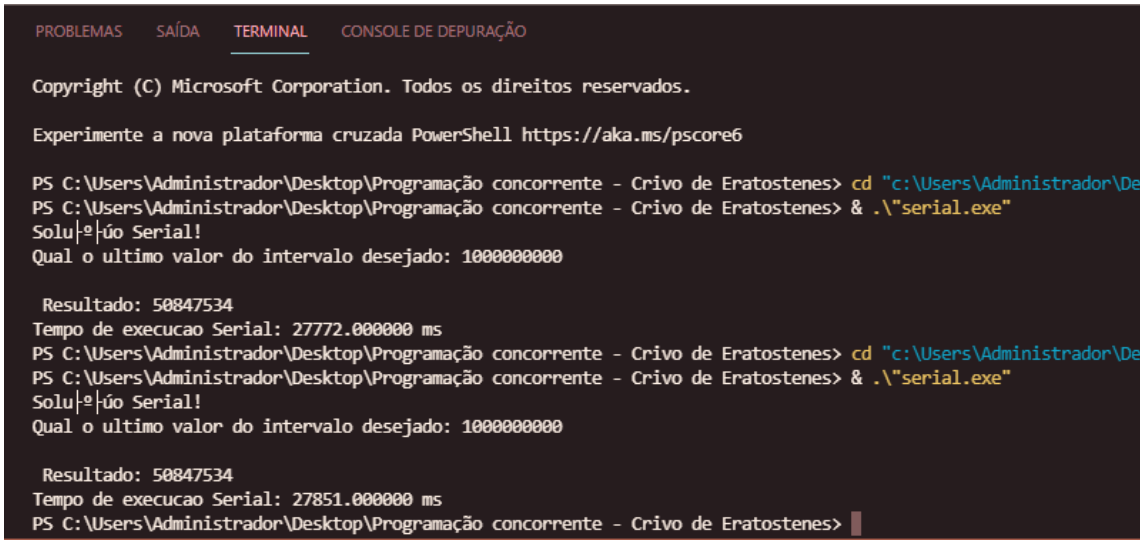
```
#include "serial.h"

#include <stdio.h>

#include <math.h>

#include <stdlib.h>



int eratostenes(int lastNumber){

  char* isPrime = malloc(lastNumber+1 * sizeof(int));

  for (int i = 0; i <= lastNumber; i++){

    isPrime[i] = 1;
```

```
  }


  for (int i = 2; i*i <= lastNumber; i++){
    if (isPrime[i]);
      for (int j = i*i; j <= lastNumber; j += i){
        isPrime[j] = 0;
      }
  }


  int found = 0;
  for (int i = 2; i <= lastNumber; i++)
    found += isPrime[i];


    free(isPrime);
  return found;
}
```

Analisando o código na máquina descrita no início deste relatório temos que, temos que a média de resultados é essa:


PROBLEMAS SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO
Copyright (C) Microsoft Corporation. Todos os direitos reservados.
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\Users\Administrador\De
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"serial.exe"
Solu├║├úo Serial!
Qual o ultimo valor do intervalo desejado: 10000000
Resultado: 664579
Tempo de execucao Serial: 203.000000 ms
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\Users\Administrador\De
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"serial.exe"
Solu├║├úo Serial!
Qual o ultimo valor do intervalo desejado: 10000000
Resultado: 664579
Tempo de execucao Serial: 201.000000 ms
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes>


PROBLEMAS SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO
Copyright (C) Microsoft Corporation. Todos os direitos reservados.
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\Users\Administrador\De
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"serial.exe"
Solu├║├úo Serial!
Qual o ultimo valor do intervalo desejado: 100000000
Resultado: 5761455
Tempo de execucao Serial: 2419.000000 ms
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\Users\Administrador\De
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"serial.exe"
Solu├║├úo Serial!
Qual o ultimo valor do intervalo desejado: 100000000
Resultado: 5761455
Tempo de execucao Serial: 2464.000000 ms
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes>


PROBLEMAS SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO
Copyright (C) Microsoft Corporation. Todos os direitos reservados.
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\Users\Administrador\De
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"serial.exe"
Solu├║├úo Serial!
Qual o ultimo valor do intervalo desejado: 1000000000
Resultado: 50847534
Tempo de execucao Serial: 27772.000000 ms
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\Users\Administrador\De
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"serial.exe"
Solu├║├úo Serial!
Qual o ultimo valor do intervalo desejado: 1000000000
Resultado: 50847534
Tempo de execucao Serial: 27851.000000 ms
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes>

Solução*Q.Elem	10.000.000	100.000.000	1.000.000.000	10.000.000.000
SERIAL	201 ms	2419 ms	27772	*

*Computador Não processou a informação

**Aumento de Elemento em 10 e 10 vezes;.

Devido a máquina não conseguir processar o valor de 10.000.000.000, fiz a seguinte página teste com alteração dos tipos de dados, de int para long int. Porém, continuou dando a seguinte tela:

```
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> cd "c:\U
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes> & .\"ser
Solu├º├úo Serial!
Qual o ultimo valor do intervalo desejado: 10000000000

 Resultado:
PS C:\Users\Administrador\Desktop\Programação concorrente - Crivo de Eratostenes>
```

(Segue código com alteração abaixo)

```c
#include "serial.h"



#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>



long int eratostenes(long int lastNumber){



  char* isPrime = malloc(lastNumber+1 * sizeof(long int));

  for (long int i = 0; i <= lastNumber; i++){

    isPrime[i] = 1;
```

```
  }


  for (long int i = 2; i*i <= lastNumber; i++){
    if (isPrime[i]){
      for (long int j = i*i; j <= lastNumber; j += i){
        isPrime[j] = 0;
      }
    }
  }


  int found = 0;
  for (long int i = 2; i <= lastNumber; i++){
      found += isPrime[i];
  }


  free(isPrime);
  return found;
}


int main(){


    long int ultimoNumero;
    long int result;
```

```
    double inicio;
    double fim;
    double t_serial;

  printf("Solução Serial! \n");

  printf("Qual o ultimo valor do intervalo desejado: ");
  scanf("%ld", &ultimoNumero);

  printf("\n Resultado: ");

  inicio = clock(); //armazena tempo
  result = eratostenes(ultimoNumero);
  fim = clock(); //tempo final - tempo inicial

  t_serial = fim - inicio;

  printf("%ld \n", result);
  printf("Tempo de execucao Serial: %lf ms \n",
((double)t_serial)/((CLOCKS_PER_SEC/1000))); //milissegundos

}
```

Para a paralisação utilizando a interface OpenMP,

Utilizei as seguintes linhas para definir o espaço de código paralelizado

1. #pragma omp parallel num_threads(NTHREADS) - Para dar Inicio a paralelização (Indicando o número de Threads, neste caso, 4)
2. #pragma omp single -> Para printar a seguinte informação textual("Inicio da região paralela \n Número de threads = %d \n", omp_get_num_threads());
3. #pragma omp for - Já que a cada laço de repetição do for, o comando iria para um processo definido. Foi deixado seguinte comando, para que o usuário compreendesse o que estava acontecendo:
   printf("Thread %d executa interação %d do for \n", omp_get_thread_num(),i);

```
#include "openMP.h"


#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define NTHREADS 4




int eratostenesOpenMP(int lastNumber){


  char* isPrime = malloc(lastNumber+1 * sizeof(int));


  for (int i = 0; i <= lastNumber; i++){

    isPrime[i] = 1;

  }
```

```
  #pragma omp parallel num_threads(NTHREADS){
    #pragma omp single
        printf("Inicio da região paralela \n Número de threads = %d \n",
omp_get_num_threads());

    #pragma omp for
      for (int i = 2; i*i <= lastNumber; i++){
        if (isPrime[i]){
          for (int j = i*i; j <= lastNumber; j += i){
            isPrime[j] = 0;
          }
        }

              printf("Thread %d executa interação %d do for \n",
omp_get_thread_num(),i);
      }
  }

  int contador = 0;

  for (int i = 2; i <= lastNumber; i++){
    if(isPrime[i]){
        contador = contador + 1;
    }
```

```
  }


  free(isPrime);

  return contador;


}
```

A solução com a interface MPI, a mais complexa de implementar em comparação às últimas. Nesta em específico, adicionei a análise do tempo dentro do arquivo MPI.c, já que foi realizada com comandos específicos que são importando na biblioteca MPI.

Foi utilizado dois comandos de envio de processos, o MPI_Scatter e o MPI_Gather, respectivamente. O MPI_Scatter foi utilizado para enviar as informações presentes no vetor criado, chamado “vetor”, com a saída definida para um “subvetor”, que possuía as mesmas características do “vetor” Assim dividimos as informações que serão processadas de maneiras paralelas de tamanho iguais.

Agora, depois de aplicar os comandos do algoritmo do Crivo de Eratóstenes, utilizamos o MPI_Gather para fazer o exato oposto do MPI_Scatter. Agora a informação que já passou pela divisão e foi processada em diferentes threads, agora se une para ser trabalhado como algo unico.

Assim chamamos o MPI_Gather, colocando o “subvetor” como endereço de entrada e um novo vetor chamado “listaFinal” como endereço de saída.
Assim, termina a paralelização do Crivo de Eratóstenes, concluímos a contagem de números primos e finalizamos o MPI.

Foi utilizado os comandos de MPI_Wtime() para receber a informação do tempo de execução da paralelização, e para o análise de desempenho de usa a função tempo(), presente no arquivo main.c

Segue o código:

```
#include "MPI.h"
```

```c
#include <stdio.h>

#include <stdlib.h>

#include <stdlib.h>

#include <math.h>

#include <mpi.h>

#include <time.h>


int erastotenesMPI(int argc, char** argv) {

    //Iniciando MPI

    MPI_Init(NULL, NULL);

    int ncpus;

    MPI_Comm_size(MPI_COMM_WORLD, &ncpus);

    int meu_rank;

    MPI_Comm_rank(MPI_COMM_WORLD, &meu_rank);


    double inicio;

    double fim;

    long int i, j;

    int *vetor = NULL;

    int *subvetor = NULL;

    int lastNumber;

    long int count = 0;


        printf("Qual o ultimo valor do intervalo desejado: ");
```

```
        scanf("%d", &ultimoNumero);


        //Alocando o vetor que estará setado (1) ou não (0)


        if (rank == 0){ //identificando o primeiro processo
        vetor = (int *)malloc(lastNumber*sizeof(int));
        for(i = 0; i < lastNumber; i++){
            vetor[i] = 1;


            inicio = MPI_Wtime();  //Inciamos a conometragem


        }


        subvetor = (int *)malloc(lastNumber * sizeof(int));
          MPI_Scatter(&vetor, lastNumber, MPI_INT, &subvetor, lastNumber,
MPI_INT, 0, MPI_COMM_WORLD);




        //CORRENDO O VETOR EM BUSCA DOS MULTIPLOS (NÃO PRIMOS)
        for (int i = 2; i*i <= lastNumber; i++){
        if (subvetor[i]);
            for (int j = i*i; j <= lastNumber; j += i){
                subvetor[j] = 0;
            }
        }
```

```
        int *listaFinal = (int *)malloc(lastNumber*sizeof(int));

MPI_Gather(&subvetor,1,MPI_INT,&listaFinal,1,MPI_INT,0,MPI_COMM_WORLD);



        if (rank == 0){
            fim = MPI_Wtime();
            for(i = 2; i < lastNumber; i++){
                if(listaFinal[i]){
                    count++;
                }
            }
        }
        printf("Tempo de processamento: %f\n", (fim -inicio));


    MPI_Finalize();
    return count;
}
```

Todos os arquivos descritos neste documento conseguem se comunicar entresi, através do uso de arquivos .h, disponíveis na linguagem de programação C.

No sistema operacional Windows, a compilação de arquivos que incluem as bibliotecas <mpi.h> e <omp.h> geram o seguinte erro:

```
c:/programdata/chocolatey/lib/mingw/tools/install/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/10
.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe:
C:\Users\ADMINI~1\AppData\Local\Temp\cc6VP2BL.o:main.c:(.text+0x2b4): undefined reference to
`eratostenesOpenMP'

c:/programdata/chocolatey/lib/mingw/tools/install/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/10
.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe:
C:\Users\ADMINI~1\AppData\Local\Temp\cc6VP2BL.o:main.c:(.text+0x33a): undefined reference to
`eratostenesMPI'

collect2.exe: error: ld returned 1 exit status
```