Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-0117 Programación Paralela y Concurrente
Grupo 01
I Semestre

III Tarea programada: Números Primos

Profesor:
Francisco Arroyo

Estudiante:
Jennifer Villalobos Fernández | B67751

10 de julio del 2020

Índice

1. Introducción

Para resolver el problema planteado es necesario un uso simultáneo de los procesos. Para tal propósito se utilizaron diferentes métodos:

- Pthreads: tal y como lo insinúa el nombre, esta biblioteca se basa en el uso de "hilos", los cuales son considerados "procesos ligeros"que representan el segmento de código que está siedo procesado en un momento dado. El uso de hilos en un proceso hace posible ejecutar segmentos de código (que pueden o no ser los mismos) de manera concurrente, permitiendo disminuir el tiempo requerido para realizar una tarea. Los hilos poseen una única identificación, así como sus variables locales y direcciones de retorno.
- OpenMP: OpenMP es un modelo de programación paralelo basado en directivas de compilación (o pragmas) que permite, al desarrollar aplicaciones, agregar progresivamente paralelismo a los códigos de la misma. Este modelo trabaja mediante memoria compartida, es decir, los hilos se comunican utilizando variables compartidas, para esto, es necesaria una buena sincronización, así se evitan las condiciones de carrera. Al igual que Pthreads, cada hilo utilizado posee una única identificación.
- MPI: MPI es una especificación de una librería especializada en el paso de mensajes, de ahí su nombre: Message Passing Interface (MPI). Al enviar estos mensajes, los datos se mueven del segmento de dirección de un proceso, al de otro proceso, por medio de operaciones cooperativas entre cada uno de ellos. Para identificar cuál proceso envió un mensaje, y cuál de ellos es el encargado de recibirlo, MPI asigna una identificación única a cada proceso, al igual que Pthreads y OpenMP.

2. Objetivo

- Familiarizar al estudiante con el desarrollo de programas concurrentes con sus distintos métodos: Pthreads, OpenMP y MPI.

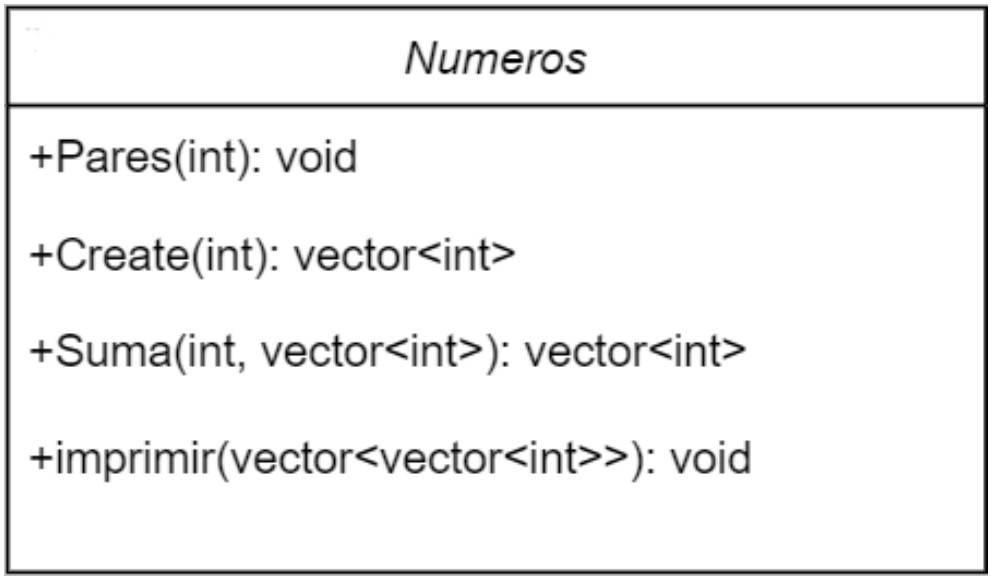
3. Descripción

Es posible representar cualquier número entero par y mayor que cuatro por la suma de dos números primos. La tarea es leer un número de la entrada estándar (n) que representará el límite superior de un conjunto de enteros] 4, n]; para cada elemento de este conjunto, se debe encontrar una representación como la suma de dos números primos. Por ejemplo:

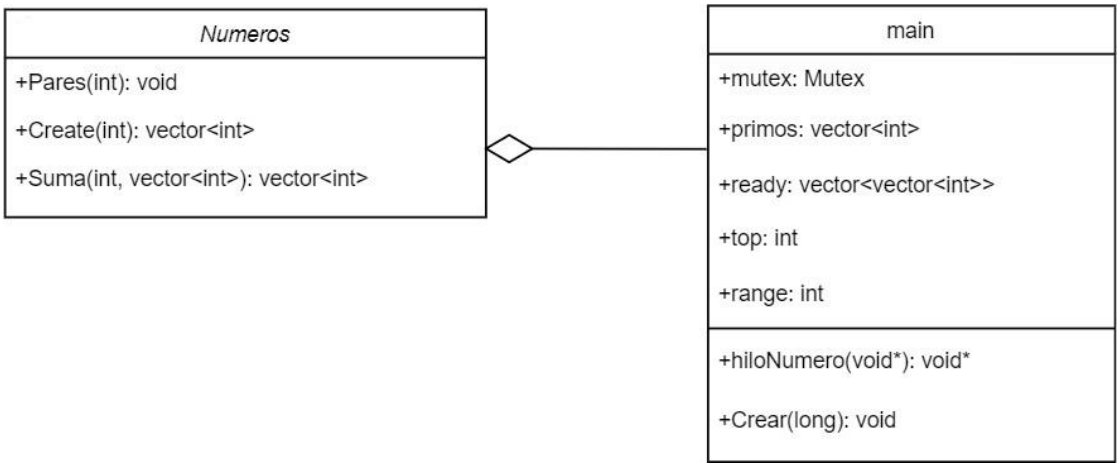
- 6=3+3, 8=3+5, 10=3+7=5+5, 12=5+7, 14=3+11=7+7, 16=3+13=5+11, 18=5+13=7+11, 20=3+17=7+13, 22=3+19=5+17=11+11, 24=5+19=7+17=11+13, 26=3+23=7+19=13+13, 28=5+23=11+17, . . .

Construir una versión serial para solucionar el problema propuesto. Construir además tres versiones paralelas utilizando las estrategias vista en clase: Pthreads, OpenMP y MPI. Hacer las mediciones de tiempo para las todas las versiones. Se debe construir una tabla comparativa de las cuatro versiones que indique los tiempos de resolución del problema y su mejora en velocidad (“speedup”), cambiando en un eje el tamaño del problema (n) y en otro eje la cantidad de trabajadores (t). Se puede utilizar una escala logarítmica para el tamaño del problema n (10, 100, 1000, etc). Y puede utilizar potencias de dos para la cantidad de trabajadores. Se debe entregar un documento con esta tabla.

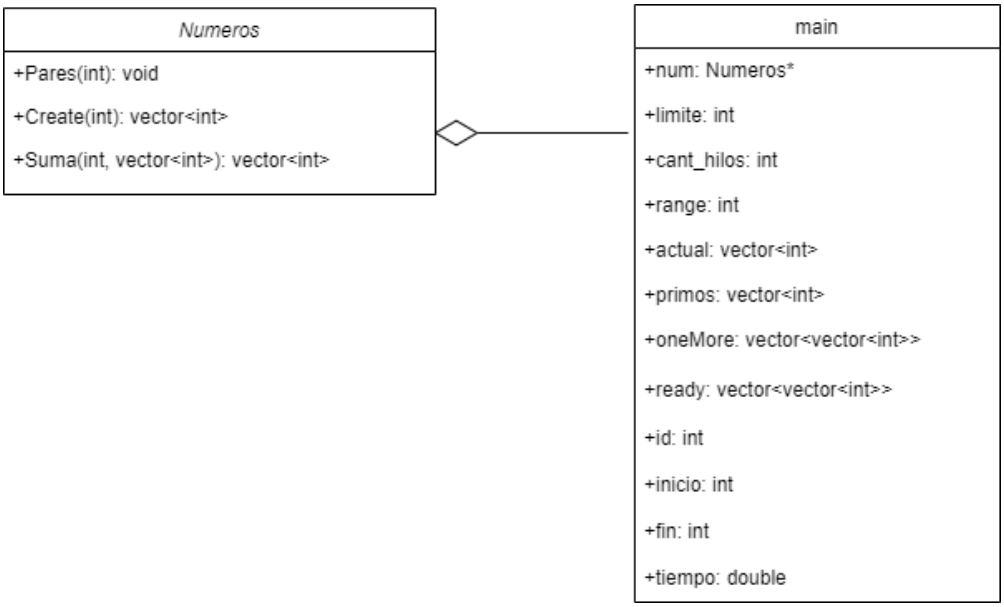
4. Diseño



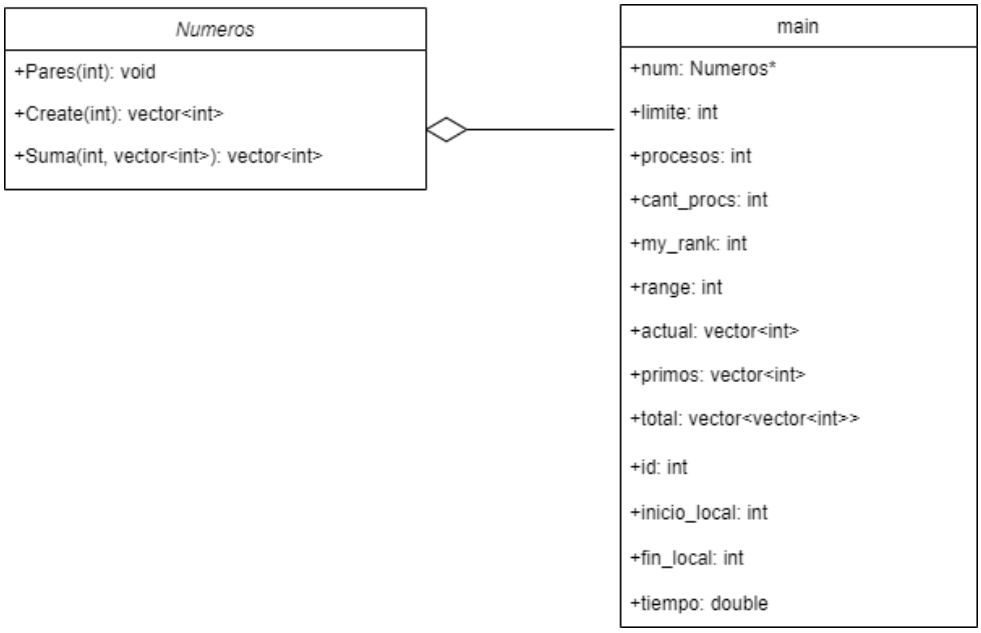
(a) UML de la clase Números en el programa NumerosSerial.



(b) UML del programa NumerosPthreads.



(c) UML del programa NumerosOpenMP.



(d) UML del programa NumerosMPI.

5. Desarrollo

NumerosSerial: Tanto para este programa, como para los siguientes, se utilizó la clase Numeros. Esta contiene (en esta versión serial) cuatro métodos: Pares, Create, Suma e Imprimir. Para realizar los cálculos solicitados, solo se llama al método Pares, ya que este invoca a los demás como parte de su funcionalidad. Al iniciar el método, se realiza un recorrido desde el número 5 (ya que deben ser mayores a 4) hasta la cantidad límite ingresada por el usuario. Dentro de este for, se evalúa si el número actual es par o no, en caso afirmativo, se agrega ese número al vector pares. Luego se invoca al método Create. Este se encarga de realizar otro recorrido, esta vez desde el número 2 hasta la cantidad límite. Dentro del recorrido se detectan los divisores de cada uno de los números. Si el número solo posee 2 (el 1 y si mismo) significa que es primo, por tanto se agrega al vector. Al finalizar todo el ciclo este vector es retornado. Volviendo al método Pares, cuando el vector de primos es retornado, ya se poseen los datos necesarios para realizar los calculos, así que se procede a tomar cada número del vector de Pares y el vector de primos, estos dos parámetros se envían al método Suma. Este recorre dos veces (con for anidados) el vector de primos, realizando sumas entre los dos resultados obtenidos dentro de cada ciclo. Cuando la suma de los dos números adquiridos da como resultado el número par evaluado actualmente, ambos se guardan y se agregan a un vector. Este vector posee en su casilla 0 el número par al que se le realizó el cálculo, en su índice 1 tiene el primer dígito encontrado y en el 2 se almacena el segundo. Este vector se retorna. En el método Pares, cada vector retornado se agrega al vector de vectores que almacena todas las sumas hasta la cantidad límite. Cuando este vector esté listo, se llama al método imprimir, el cual toma el vector de cada índice y lo despliega en pantalla, tomando el número par primero y obteniendo un resultado como el siguiente: *Par = primero + segundo*. Por último, en el main se establecen las mediciones de tiempo, y entre estas, se ejecuta el método Pares. Estas mediciones se realizan con la librería de C++ chrono, no con la clase provista, ya que esta presentaba fallos en los cálculos del tiempo. La librería chrono muestra el tiempo pared transcurrido en milisegundos.

NumerosPthreads: Para este programa se utilizó la misma clase Numeros empleada en la versión serial, solo que con unas pequeñas modificacines: como en esta versión es necesario dividir el trabajo, el método Pares ahora recibirá un rango para hacer sus cálculos, es decir, recibe un entero que representa el inicio y otro el final de ese rango. Además, también recibe el vector de primos, ya que este solo se calculará una vez, no se hará en cada hilo, esto para mejorar la eficiencia. Además, si el rango ingresado inicia en 1 (es el hilo 0), este se cambiará por 5, ya que según el enunciado, se debe hacer de 4 en adelante. El método imprimir se elimina. En esta, como el las siguientes versiones, lo principal es el main: este posee ciertas variables globales (compartidas entre los hilos): la cantidad límite, el rango que debe cubrir cada hilo, el vector de primos (como solo se calcula una vez, debe compartirse), el vector ready, que almacena todas las sumas y el Mutex, para evitar condiciones de carrera. Ahora los métodos: posee solo dos, el hiloNmero y Crear. Crear se encarga de generar tantos hilosNumero como la cantidad enviada como parámetro lo indique, y espera con su join a que cada uno de ellos termine su función. El método hiloNumero se encarga de realizar el trabajo de los cálculos. Dentro de este se tiene la identificación del mismo, su inicio y fin (el rango que debe calcular), el vector de vectores OneMore (que almacena todas las sumas calculadas de todo su rango), el vector actual y su instancia de la clase Numero. Al inicio se crea el rango del hilo, este calculo se realiza con respecto a su identificación. Luego se bloquea el acceso al vector global ready, utilizando el Mutex. Esto debido a que a continuación se realiza el llamado al método Pares. El vector de vectores retornado (con todas las sumas) se le asigna al vector propio OneMore. Ahora se realiza un ciclo, en donde se recorre cada índice de ese vector, tomando cada vector con la suma de cada par, y se le asigna al vector propio actual. Este a su vez, se añade al vector global ready. Al finalizar, se desbloquea el Mutex. Dentro del método main se inicializan las variables que calculan el tiempo, al estas estar corriendo, se toma la cantidad límite y la cantidad de hilos ingresadas por el usuario. Con estos datos se realizan los cálculos del rango de cada hilo y la creación de los números primos con el método Create, este resultado se asigna al vector global primos. Por último, se crean los hilosNumeros mediante el método Crear y se despliega el resultado de todas las sumas encontradas al imprimir el vector global ready, así como también se muestra el tiempo de ejecución.

NumerosOpenMP: En esta versión, la clase Numeros se mantiene igual a la de Pthreads. En el main tenemos al inicio la ejecución del tiempo, luego la declaración del límite y los hilos por utilizar ingresados por el usuario. Con estos datos se genera el rango y también se crea el vector de primos mediante el método Create. La sección paralela inicia en este momento: se declara el comando pragma omp parallel con la cantidad de hilos especificada por el usuario. En este mismo comando, se declara que las variables compartidas entre los hilos serán el vector de primos y el vector global (donde se almacenan todas las sumas). Además, se declaran como variables privadas inicio y fin (definen el rango del hilo), oneMore (el vector encargado de tomar todas las sumas de ese hilo en específico) y el vector actual, encargado de agregar cada una de las sumas al vector global ready. El comando completo sería:

```
$ #pragma omp parallel num_threads (cant_hilos) shared (ready, primos) private (oneMore, actual, inicio,
    fin)
```

Dentro de esta sección paralela se toma el identificador del hilo entrante mediante el comando omp get thread num. Con este id se genera el rango (variables inicio y fin) sobre el cual trabajará este hilo. Se invoca al método Pares y el vector resultante se asigna a oneMore. A su vez, dentro de un ciclo for, cada índice de oneMore se asigna al vector actual, y este se agrega al vector global ready. Para esta última acción se utiliza el comando pragma omp critical, así solo un hilo a la vez podrá acceder al vector global, evitando condiciones de carrera. Por último se utiliza pragma omp single para que solo un hilo se encargue de imprimir los resultados (el vector ready) así como el tiempo de ejecución.

NumerosMPI: La clase Numero de esta versión es la misma utilizada en las dos versiones anteriores. Con respecto al main, este se realiza utilizando el paso de mensajes. Al iniciar el método, al igual que antes, se ejecuta la medición del timepo y se toma la cantidad límite y la cantidad de procesos a utilizar. Aquí hay un ligero cambio: el proceso 0 se encargará de recibir los mensajes enviados por los demás procesos, eso implica que es un proceso menos al momento de dividir la cantidad de procesos entre la cantidad límite. Por este motivo, se crea la variable cant procs, la cual contiene la cantidad de procesos entrante menos 1. Ahora con eso ya definido sí se puede realizar el cálculo del rango de cada proceso (a excepción del 0). Antes de iniciar la sección MPI, se calculan los números primos mediante Create y se asigna al vector correspondiente. En este momento se inicia la sección paralela: si el id del proceso (variable my rank) es distinto de 0, ingresa a calcular las sumas de su rango, el cual se define mediante s id, al igual que antes. Se invoca al método Pares, y el vector con todas las sumas resultantes se asigna al vector local. El tamaño de este vector indica cuántos dígitos fueron calculados con ese hilo, por lo tanto, este tamaño (variable veces) es enviado al proceso 0, para que este sepa cuántas sumas estará por recibir de este proceso. El envío del mensaje se hace por medio de este comando:

```
$ MPI_Send(&veces, 1, MPI_INT, 0, 0, MPI_COMM_WORLD)
```

Luego de enviar el mensaje, cada índice del vector oneMore es asignado al vector actual, este a su vez, es enviado mediante un mensaje al proceso 0, para que este lo agregue al vector global total. El proceso 0 tiene su vector actual y su variable veces. Se inicia un ciclo for, el cual se ejecuta desde 1 hasta el total de procesos ingresados por el usuario. Dentro de este ciclo se recibe el mensaje del tamaño (es decir, la cantidad de sumas) que se recibirán del proceso indicado por el ciclo. Con esta variable (veces) se crea otro ciclo for, el cual inicia desde 0 hasta veces, y se encarga de recibir cada vector de suma enviado por dicho proceso. Este vector recibido es asignado al vector actual, y este a su vez es agregado al vector global total. Por último, cuando se finalizan todos los procesos, el proceso 0 imprime los resultados obtenidos (vector total) así como el tiempo de ejecución.

6. Manual de usuario

Requerimientos de Software

- **Sistema Operativo:** Linux.
- **Arquitectura:** 64 bits.
- **Ambiente:** Code::Blocks o Terminal.

Compilación

Para compilar el programa "NumerosSerial"puede usarse el Makefile, con solo llamar al comando make en terminal. También se puede utilizar `g++` en la siguiente sentencia:

```
$ g++ Numeros.cpp main.cpp -o nombre_ejecutable
```

Para ejecutar el mismo, se debe insertar el nombre del ejecutable y la cantidad límite de los números por procesar. El comando completo sería el siguiente:

```
$ ./nombre_ejecutable <cantidad_limite>
```

Para compilar el programa "NumerosPthreads"puede usarse el Makefile, con solo llamar al comando make en terminal. También se puede utilizar `g++`, así como `lpthread` en la siguiente sentencia:

```
$ g++ Numeros.cpp main.cpp Mutex.cpp -lpthread -o nombre_ejecutable
```

Para ejecutar el mismo, se debe insertar el nombre del ejecutable, la cantidad límite de los números por procesar y la cantidad de hilos. El comando completo sería el siguiente:

```
$ ./nombre_ejecutable <cantidad_limite> <cantidad_hilos>
```

Para compilar el programa "NumerosOpenMP"puede usarse el Makefile, con solo llamar al comando make en terminal. También se puede utilizar `g++`, así como `-fopenmp` en la siguiente sentencia:

```
$ g++ Numeros.cpp main.cpp -fopenmp -o nombre_ejecutable
```

Para ejecutar el mismo, se debe insertar el nombre del ejecutable, la cantidad límite de los números por procesar y la cantidad de hilos. El comando completo sería el siguiente:

```
$ ./nombre_ejecutable <cantidad_limite> <cantidad_hilos>
```

Para compilar el programa "NumerosMPI"puede usarse el documento Çompilar y Ejecutar"que se encuentra dentro de la carpeta del programa. Este contiene los comandos para compilar y ejecutar el programa correctamente. El comando para compilarlo en terminal se muestra en la siguiente sentencia:

```
$ mpiCC -g -o nombre_ejecutable Numeros.cpp main.cpp
```

Para ejecutar el mismo, se debe insertar el comando `mpiexec -n` junto con la cantidad de procesos a utilizar, el nombre del ejecutable y la cantidad límite de los números por procesar. El comando completo sería el siguiente:

```
$ mpiexec -n <cantidad_procesos> ./nombre_ejecutable  <cantidad_limite>.
```

7. Casos de Prueba

Pruebas de los cuatro programas: Estas pruebas verifican el funcionamiento de los cuatro programas. El código utilizado en la clase Numeros de los tres programas paralelos, corresponde a:

```
#include "Numeros.h"
#include <vector>
#include <string.h>
#include <iostream>

Numeros::Numeros()
{
}

Numeros::~Numeros()
{
}

/*
*Metodoencargadodeencontrarlosnumerospares
*yhacerelcalculorespectivoconcadaunodeellos.
*/
std::vector<std::vector<int>> Numeros::Pares(int inicio, int fin, std::vector<int>
    primos)
{
    std::vector<int> pares;//Vectorquealmacenatodoslosnumerosparesdelrango
        ingresado.
    std::vector<int> calculo;//Vectorquealmacenalosdigitosdesumadeunnumero
        dentrodelrango.
    std::vector<std::vector<int>> todos;//Almacenatodaslassumasdetodoslos
        numerospares.
    int start = inicio;//Variablequetomaelvalorinicialdelrango

    //Sielrangodelhiloiniciaen1,setomanlosnumerosdesde5.
    if(inicio == 1){
        start = 5;
    }

    for(int number=start; number <= fin; number++)
    {
        //Sielnumeroespar,seagregaalvector
        if(number % 2 == 0)
        {
            pares.push_back(number);
        }
    }

    for(size_t i = 0; i < pares.size(); i++)
    {
        calculo = Suma(pares[i], primos);
        todos.push_back(calculo);
    }


return todos;
}

```

```
48
49  /*
50  *Metodoencargadodeencontrartodoslosnumeros
51  *primosdelnumeroentrante.
52  */
53  std::vector<int> Numeros::Create(int top)
54  {
55      std::vector<int> primos;
56      int divisores;
57      int divisor;
58
59      //Seguardantodoslosnumerosprimos
60      for (int num=2; num <= top; num++)
61      {
62          divisores = 0;
63          divisor = 1;
64
65          do
66          {
67              if(num % divisor == 0)
68              {
69                  divisores++;
70              }
71              divisor++;
72          }
73          while(divisor <= num);
74
75          //Elnumeroesimpar:solosedivideentre1ysimismo.
76          if(divisores == 2)
77          {
78              primos.push_back(num);
79          }
80      }
81
82      return primos;
83  }
84
85
86  /*
87  *Metodoencargadodeencontrarlosdosdigitosqueal
88  *sumarlos,dacomoresultadoelnumeroingresado.
89  */
90  std::vector<int> Numeros::Suma(int par, std::vector<int> primo)
91  {
92      int resultado;
93      int termina = 1;
94      int first;
95      int second;
96      std::vector<int> foundit (3, 0);
97      int inicio;
98
99      //Paravariarlosresultados,de14enadelante,nosetomael3.
100     if(par < 14)
101     {
102         inicio = 0;
103     }
```

```
104         else
105         {
106             inicio = 1;
107         }
108 
109         for(size_t primero = inicio; primo[primero] < par; primero++)
110         {
111             if(termina == 1)//Sinosehaencontradounresultado
112             {
113                 for(size_t segundo = 0; primo[segundo] < par; segundo++)
114                 {
115                     if(termina == 1)//Sinosehaencontradounresultado
116                     {
117                         resultado = primo[primero]+primo[segundo];
118                         if(resultado == par)
119                         {
120                             first = primo[primero];
121                             second = primo[segundo];
122                             termina = 0;
123                         }
124                     }
125                 }
126             }
127         }
128 
129         foundit.at(0) = par;
130         foundit.at(1) = first;
131         foundit.at(2) = second;
132         return foundit;
133     }
```

Los resultados obtenidos con los distintos programas se pueden verificar en las siguientes capturas:

```
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosSerial$ make
g++ -c -g Numeros.cpp main.cpp
g++ -g Numeros.o main.o -o num
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosSerial$ ./num 50
6 = 3 + 3.
8 = 3 + 5.
10 = 3 + 7.
12 = 5 + 7.
14 = 3 + 11.
16 = 3 + 13.
18 = 5 + 13.
20 = 3 + 17.
22 = 3 + 19.
24 = 5 + 19.
26 = 3 + 23.
28 = 5 + 23.
30 = 7 + 23.
32 = 3 + 29.
34 = 3 + 31.
36 = 5 + 31.
38 = 7 + 31.
40 = 3 + 37.
```

(e) Salida del programa Serial con 50 términos.

```
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosPthread$ make
g++ -c -g Numeros.cpp Mutex.cpp main.cpp -lpthread
g++ -g Numeros.o Mutex.o main.o -lpthread -o num
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosPthread$ ./num 50 3
38 = 7 + 31.
40 = 3 + 37.
42 = 5 + 37.
44 = 3 + 41.
46 = 3 + 43.
48 = 5 + 43.
50 = 3 + 47.
20 = 3 + 17.
22 = 3 + 19.
24 = 5 + 19.
26 = 3 + 23.
28 = 5 + 23.
30 = 7 + 23.
32 = 3 + 29.
34 = 3 + 31.
36 = 5 + 31.
6 = 3 + 3.
8 = 3 + 5.
```

(f) Salida del programa con Pthreads con 50 términos y 3 hilos.

```
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosOpenMP$ make
g++ -c -g Numeros.cpp main.cpp -fopenmp
g++ -g Numeros.o main.o -fopenmp -o num
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosOpenMP$ ./num 100 6
6 = 3 + 3.
8 = 3 + 5.
10 = 3 + 7.
12 = 5 + 7.
14 = 3 + 11.
16 = 3 + 13.
18 = 5 + 13.
20 = 3 + 17.
82 = 3 + 79.
84 = 5 + 79.
86 = 3 + 83.
88 = 5 + 83.
90 = 7 + 83.
92 = 3 + 89.
94 = 5 + 89.
96 = 7 + 89.
98 = 19 + 79.
100 = 3 + 97
```

(g) Salida del programa con OpenMP con 100 términos y 6 hilos.

```
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosMPI$ mpiCC -g -o num Numeros.cpp main.cpp
jennifer@jennifer-VirtualBox:~/Escritorio/ProgramacionParalela_B67751_Villalobos
/Tareas/Tarea3/NumerosMPI$ mpiexec -n 8 ./num 100
Total.size: 48
6 = 3 + 3.
8 = 3 + 5.
10 = 3 + 7.
12 = 5 + 7.
14 = 3 + 11.
16 = 3 + 13.
18 = 5 + 13.
20 = 3 + 17.
22 = 3 + 19.
24 = 5 + 19.
26 = 3 + 23.
28 = 5 + 23.
30 = 7 + 23.
32 = 3 + 29.
34 = 3 + 31.
36 = 5 + 31.
38 = 7 + 31.
40 = 3 + 37.
42 = 5 + 37.
```

(h) Salida del programa con MPI con 100 términos y 8 hilos.

Tabla tiempos de ejecución

Tiempos de ejecución de los cuatro programas: Esta tabla muestra el tiempo transcurrido desde el inicio hasta la finalización de la ejecución de los cuatro programas. A la derecha se encuentra la cantidad límite ingresada. A la izquierda de la ejecución Serial se muestra el número de hilos con los cuales se ejecutó el programa (con la cantidad de términos indicada a la derecha).

Términos	Serial (ms)	Trabajadores	Pthreads (ms)	OpenMP (ms)	MPI (ms)
10	1.661	2	1.540	1.179	52.632
50	4.160	3	3.117	2.186	105.157
		5	3.457	2.957	186.231
100	5.839	6	5.774	3.855	234.070
		8	6.644	5.602	338.233
500	22.498	10	22.540	21.608	431.196
		12	25.213	23.269	494.133
1000	53.708	15	50.653	49.796	748.880
		18	54.290	48.514	873.272
5000	464.449	20	442.410	439.881	3931.744
		25	450.088	441.620	4729.024
10 000	1493.409	28	1484.428	1464.924	17259.942
		34	1490.942	1452.909	20844.746
50 000	36021.445	35	30654.486	32301.942	465779.258
		40	24145.616	32812.191	529642.848

(i) Tabla con los tiempos de ejecución.

Tabla Speedup

Speedup de los cuatro programas: Esta tabla muestra la mejora en velocidad entre los cuatro programas.

Términos	Trabajadores	Speedup Pthreads (ms)	Speedup OpenMP (ms)	Speedup MPI (ms)
10	2	1.078	1.408	0.031
50	3	1.334	1.903	0.039
	5	1.203	1.406	0.022
100	6	1.011	1.514	0.024
	8	0.878	1.042	0.017
500	10	0.998	1.041	0.052
	12	0.892	0.966	0.045
1000	15	1.060	1.078	0.071
	18	0.989	1.107	0.061
5000	20	1.049	1.055	0.118
	25	1.031	1.051	0.098
10 000	28	1.006	1.019	0.086
	34	1.001	1.027	0.071
50 000	35	1.175	1.175	0.077
	40	1.491	1.097	0.068

(j) Tabla comparativa de los cuatro programas.