Cómputo multiprocesador:

Los números primos y la Criba de Eratóstenes

José Ramón Romero Chávez (A01700318)
Tecnológico de Monterrey, Campus Querétaro
A01700318@itesm.mx

25 de Noviembre del 2018

Resumen

Los números primos juegan un rol fundamental en muchas de las actividades relacionadas con las tecnologías de la información una de ellas en la criptografía de llave pública y privada la cual está basada en la dificultad para factorizar números grandes en sus factores primos.

Teniendo en mente estas importantes aplicaciones es necesario que toda persona relacionada a estos campos por lo menos conozca a algún algoritmo para obtener un conjunto de números primos

La criba es un algoritmo antiguo para encontrar todos los números primos hasta un límite dado, está basado en la idea de diferencias constantes entre los múltiplos de un número

Además de dar a conocer este algoritmo el principal propósito de este proyecto es exponer las ventajas el cómputo paralelo comparado con su ejecución secuencial y las implicaciones y aplicaciones que éste puede tener.

Introducción

Sobre el cómputo paralelo

"El número de transistores incorporados en un chip sera aproximadamente el doble cada 24 meses."—Gordon Moore, Intel co-fundador

Es probable que en la década de 1960 cuando Gordon Moore dijo esto por primera vez, nunca pudo imaginar todos los distintos tipos de artefactos que hay el día hoy y con ellos también el final de su predicción debido a las implicaciones físicas que el cambio de tamaño de los componentes dentro de un chip implica especialmente hablando de la cantidad de calor producido y la energía utilizada.

Pero como lo describe Herb Sutter (2005), surgió otra idea para revolucionar la industria de la computación, "los sistemas multinúcleo" y, junto con ellos, una herramienta de software que buscará sacar lo mejor de ellos mediante el uso de herramientas.

Como lo describe Marowka (2007), durante muchos años, solo pequeños sectores de la sociedad fueron los únicos capaces de explotar las capacidades de los sistemas multiprocesador, debido a su complejidad y costo, no fue hasta 2005 cuando AMD e Intel introdujeron los procesadores de arquitectura de doble núcleo y causó una caída dramática en el precio de las computadoras de escritorio y las computadoras portátiles, así como en los procesadores multinúcleo.

Los procesadores de doble núcleo fueron solo el comienzo de la era de procesadores multinúcleo.

La principal consecuencia ahora es que las aplicaciones tendrán que ser cada vez más paralelizadas para aprovechar al máximo las ganancias de rendimiento del procesador ahora que están disponibles.

Desafortunadamente, escribir un código paralelo es más complejo que escribir un código de serie. Aquí es donde las API’s, como OpenMP, Intel TBB y otras, ingresan a la escena de computación paralela.

Todos ellas ayudan a los desarrolladores a crear aplicaciones multiproceso más fácilmente, al tiempo que conservan el aspecto de la programación en serie.

Afortunadamente, esto no se ha mantenido solo en las computadoras de escritorio, finalmente se ha dado el salto a dispositivos móviles y de dimensiones similares.

Todo esto ha sido un hermoso proceso de mejora continua tanto del software como del hardware, y la búsqueda de mejores alternativas aún no se ha detenido.

Ahora la apuesta se basa en lo que se ha denominado como "Hardware As A Service", ofrecido por muchas compañías como Microsoft, Google y Amazon.

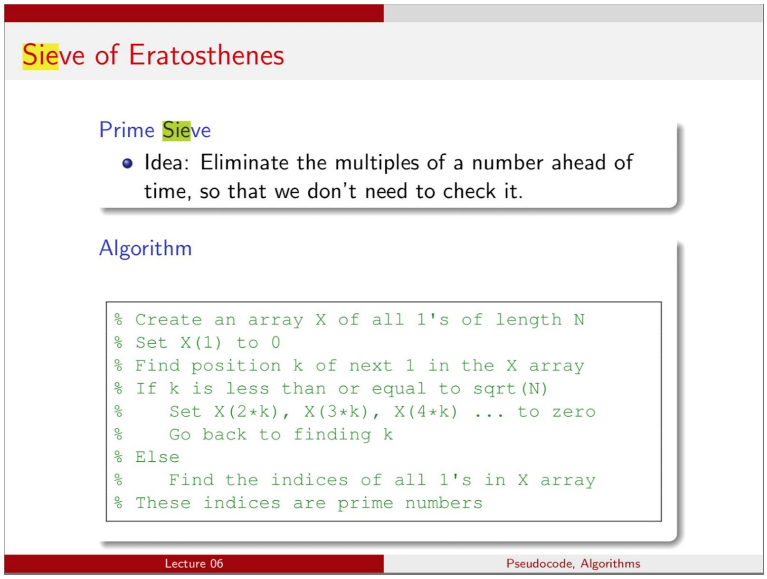
Herb Sutter en "Welcome to the Jungle" (2008) explica la oportunidad que ofrecen estos servicios para acceder a grupos masivos de supercomputadoras especializadas a nuestras necesidades sin tener que invertir el costo real de cada uno de ellos desde la comodidad de nuestros hogares.

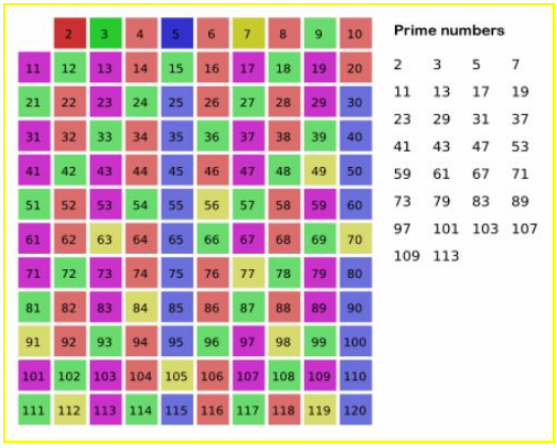
Verdaderamente este momento histórico nos brinda la oportunidad de realizar prácticamente cualquier software sin serias limitaciones de tiempo, dificultad y costos.

Recordando que siempre es necesario tener en cuenta que no todos los algoritmos y programas se pueden paralelizar, por lo que es necesario que el programador esté al tanto de la relación que existe entre los hilos lógicos y su abstracción relacionada con los núcleos y procesadores, su comunicación. y sincronización, especialmente considerar dependencias, condiciones de carrera, inconsistencias de memoria, “deadlocks”y otros conflictos, para evaluar si la paralelización es una opción de optimización.

Sobre los números primos y la Criba de Eratóstenes

Como lo describe Sierpiński (1988), cualquier número mayor que 1, que no tiene divisores naturales (enteros) excepto sí mismo y 1 se llama número primo. Son fundamentales en la teoría de los números debido al teorema fundamental de la aritmética: cada número natural mayor que 1 es un primo en sí mismo o se puede factorizar como un producto de primos que es único hasta su orden. Gracias a este teorema, es posible usarlo en múltiples algoritmos en ciencias de la computación, como la criptografía de clave pública, que se basa en la dificultad de factorizar grandes números en sus factores primos.

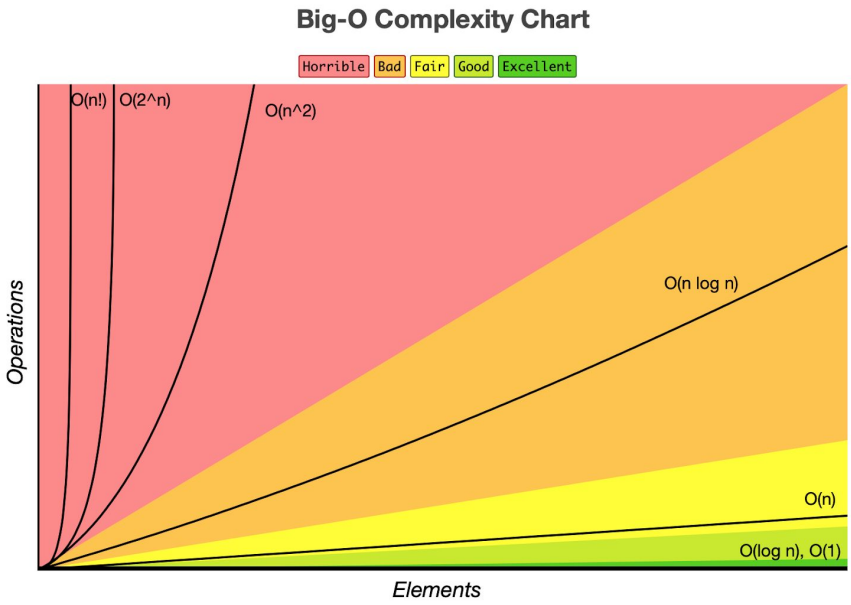




Este es el algoritmo / pseudocódigo del tamiz de Eratóstenes descrito por Erdal Yılmaz de la Universidad de Cornell (2013) y una representación gráfica del mismo.

Este algoritmo implica una gran mejora en la complejidad del tiempo del algoritmo en comparación con un método de “Trial Division”conocido como "BruteForce"

Recordando que la Criba tiene una complejidad de tiempo de O (N * log (log (N)) y un método de división de prueba tiene una complejidad de tiempo de O (N * M), ambos con una complejidad de espacio de O (N), Esto implica un consumo de tiempo significativamente menor.



Gráfica comparativa de complejidad de Tiempo en notación Big O.

Methods and Development

Desarrollo

El proyecto está enfocado en la implementación de la Criba de Eratóstenes, primeramente desde un enfoque secuencial y posteriormente haciendo uso de herramientas que permiten la paralelización como los son::

- C: OpenMP
- Java: Threads
- C++: Threading Building Blocks (TBB)
- Java: Fork Join Framework
- NVIDIA: CUDA: GPU Computing

Hardware

Primeramente es necesario dar a conocer que a la hora de realizar mediciones de tiempo, y sin importar la tecnología utilizada, estos datos dependen en gran medida del hardware utilizado para su implementación y pruebas.

Las siguientes implementaciones fueron realizadas con el siguiente hardware:

- MacBook Pro (Retina, 13-inch, Early 2015)
  - macOS Mojave 10.14 (18A391)
  - Processor Name: Intel Core i5
  - Processor Speed: 2.7 GHz
  - Number of Processors: 1
  - Total Number of Cores: 2 with hyperthreading
  - L2 Cache (per Core): 256 KB
  - L3 Cache: 3 MB
  - Memory: 8 GB 1867 MHz DDR3
  - clang --version
    - Apple LLVM version 10.0.0 (clang-1000.11.45.2)
    - Target: x86_64-apple-darwin18.0.0
    - Thread model: posix
  - javac -version: javac 9
  - java -version:
    - java version "9"
    - Java(TM) SE Runtime Environment (build 9+181)
    - Java HotSpot(TM) 64-Bit Server VM (build 9+181, mixed mode)

Implementación "Trial Division" AKA "brute force"

Hace unos meses estaba haciendo algunos desafíos de codificación en línea, algunos de los cuales me pidieron que validara los números primos en un rango, hasta ese día, la única forma en que podía hacerlo era iterar sobre el rango y buscar aquellos números que solo se pudieran dividir por 1 y sí mismo.

Esta idea parece funcionar bien, pero el hecho es que la iteración sobre un número y todos los que están debajo de sí mismo en búsqueda de algún divisor puede ser su divisor, puede funcionar bien para un rango "pequeño", pero cuando se trata de un rango "grande" simplemente su tiempo de ejecución es demasiado.

La razón de esto, cae en su complejidad de tiempo, que en Big O Notation es O (N * M), porque en el peor de los casos (un número primo) es necesario iterar para todos los números enteros debajo de él, puede ser mejorado a O (N * Log (N)) si solo iteramos hasta la raíz cuadrada de cada elemento para validar su estado como un número primo

Estos fueron los resultados de una ejecución sin mejoras al algoritmo

C/C++ (Appendix 1)

BruteForce Not Optimized C 10 times Average	
Limit	Time (ms)
10	0,001700
100	0,007100
1000	0,332000
10000	19,405500
100000	1510,602000
1000000	129228,512200

Java (Appendix 2)

BruteForce Not Optimized java 10 times Average	
Limit	Time (ms)
10	0,000000
100	0,000000
1000	0,700000
10000	25,800000
100000	1923,900000
1000000	156861,700000

Por otro lado, una mejora simple en el límite para el interior de la iteración, implica una mejora increíble de O (N * Log (N)), haciendo posible más casos de prueba, estos fueron los resultados.

C/C++ (Appendix 3)

BruteForce Optimized Java 10 times Average	
Limit	Time (ms)
10	0,001600
100	0,004800
1000	0,063200
10000	1,148500
100000	23,145000
1000000	554,788400
10000000	14047,1854

Java (Appendix 4)

BruteForce Optimized Java 10 times Average	
Limit	Time (ms)
10	0,00
100	0,10
1000	0,20
10000	1,50
100000	22,90
1000000	519,70
10000000	13818,00

Después de algunas semanas de leer algunas lecturas de programación, encontré el algoritmo de la Criba de Eratóstenes, desde la primera vez que lo leí me fascinaron las posibilidades, las mejoras y la simplicidad que tenía, así que decidí implementarlo.

La Criba de Eratóstenes, una implementación secuencial

Como se describió en la introducción, el algoritmo se basa en la diferencia constante entre un número y sus múltiplos, lo que implica una reducción importante de los tiempos que iteramos sobre los números, también incluye la otra optimización, el uso de la raíz cuadrada. del límite para reducir el número general de iteraciones.

Teniendo esto en cuenta, la complejidad del tiempo resultante es ahora una O decente (N * log (log (N))).

Estos fueron los resultados de la implementación secuencial del algoritmo, sin duda, una mejora.

C/C++ (Appendix 5)

Sequential Sieve C 10 times Average	
Limit	Time (ms)
10	0,001400
100	0,003200
1000	0,047100
10000	1,104900
100000	22,764300
1000000	567,267800
10000000	15141,640800

Java (Appendix 6)

Sequential Sieve C 10 times Average	
Limit	Time (ms)
10	0,000000
100	0,000000
1000	0,200000
10000	1,600000
100000	26,700000
1000000	650,600000
10000000	16927,200000

El punto de interés para las siguientes secciones se basa directamente en una sección del algoritmo, la búsqueda de los números que son múltiplos del número primo, que al ser una búsqueda y validación sobre el resto de la matriz, es una tarea que puede ser paralelizada.

La Criba de Eratóstenes, una implementación paralela

OpenMP

Obtenido de su sitio web, OpenMP es una interfaz de programa de aplicación (API) que admite la programación paralela de memoria compartida multiplataforma en C / C ++ y Fortran en todas las arquitecturas, incluidas las plataformas Unix y Windows NT. Abierto conjuntamente por un grupo de los principales proveedores de hardware y software de computadoras, OpenMP es un modelo escalable y portátil que brinda a los programadores paralelos de memoria compartida una interfaz simple y flexible para desarrollar aplicaciones paralelas para plataformas que van desde el escritorio hasta la supercomputadora.

Probablemente es la API más fácil de usar para la programación paralela, se usa ampliamente porque es compatible con C, C ++ y Erlang.

Para la implementación de Criba de Eratóstenes, solo fue necesario especificar las restricciones adecuadas para nuestras variables compartidas con el uso de la sentencia:

```
#pragma omp parallel for shared(a, limit) private(c)
```

Eso nos permitirá iniciar un paralelo para que sea capaz de buscar en la matriz buscando los múltiplos de los números primos

El resultado fue el siguiente:

C: OpenMP (Appendix 7)

Parallel Sieve OpenMP10 times Average	
Limit	Time (ms)
10	0,938400
100	0,392800
1000	0,413600
10000	1,807700
100000	18,612200
1000000	367,919000
10000000	10775,373300

Java: Threads

Recuperado de la documentación oficial de Java, los subprocesos a veces se denominan procesos ligeros. Tanto los procesos como las hebras proporcionan un entorno de ejecución, pero la creación de una nueva secuencia requiere menos recursos que la creación de un nuevo proceso.

Los hilos existen dentro de un proceso, cada proceso tiene al menos uno. Los hilos comparten los recursos del proceso, incluida la memoria y los archivos abiertos. Esto hace que la comunicación sea eficiente, pero potencialmente problemática.

La ejecución multiproceso es una característica esencial de la plataforma Java. Cada aplicación tiene al menos un subproceso, o varios, si cuenta los subprocesos del "sistema" que hacen cosas como la administración de la memoria y el manejo de la señal. Pero desde el punto de vista del programador de aplicaciones, usted comienza con un solo hilo, llamado el hilo principal. Este hilo tiene la capacidad de crear hilos adicionales, este último punto es fundamental para la comprensión adecuada de la implementación de Criba de Eratóstenes usando hilos.

Para esta implementación son necesarias dos clases:

- Sieve Threads: es una extensión de la superclase Threads, que compartirá la criba (arreglo) y un intervalo, definido por (Limit / # THREADS), que permitirá que el hilo itere sobre el tamiz sin causar ninguna inconsistencia de memoria en búsqueda de números múltiplos
- Main: donde se define el límite, y se crea y ejecuta un conjunto de Sieve Threads, y al final nos mostrará la Criba generada.

Los resultados de las ejecuciones fueron los siguientes:

Java: Threads: (Appendix 8)

Parallel Sieve Threads Java 10 times Average	
Limit	Time (ms)
10	0,100000
100	0,200000
1000	0,300000
10000	1,500000
100000	19,300000
1000000	558,100000
10000000	14911,900000

C++: Intel® Threading Building Blocks (TBB)

Obtenido del sitio web de Intel® Threading Building Blocks, Intel® TBB es una popular biblioteca de plantillas C ++ de software que simplifica el desarrollo de aplicaciones de software que se ejecutan en paralelo (clave para cualquier computadora multinúcleo). Intel® TBB extiende C ++ para el paralelismo de una manera fácil de usar y eficiente. Está diseñado para funcionar con cualquier compilador de C ++, simplificando así el desarrollo de aplicaciones para sistemas multi-core. Intel® TBB es una biblioteca de plantillas de C ++ que agrega programación paralela para los programadores de C ++. Utiliza la programación genérica para ser eficiente. Threading Building Blocks incluye algoritmos, contenedores altamente concurrentes, bloqueos y operaciones atómicas, un programador de tareas y un asignador de memoria escalable. Estos componentes en Intel® TBB pueden usarse individualmente o todos juntos para facilitar el desarrollo de C ++ para multi-core. Intel® TBB proporciona una abstracción para el paralelismo que evita la programación de bajo nivel inherente al uso directo de paquetes de subprocesos, como los subprocesos p o subprocesos de Windows. Tiene programadores para expresar tareas en lugar de hilos. Intel® TBB facilita el rendimiento escalable de una manera que funciona en una variedad de máquinas para hoy y prepara los programas para mañana. Detecta la cantidad de núcleos en la plataforma de hardware y realiza los ajustes necesarios a medida que se agregan más núcleos para permitir que el software se adapte. Por lo tanto, Intel® TBB aprovecha más eficazmente el hardware de múltiples núcleos.

La implementación para Criba de Eratóstenes es similar a la que se realizó para Java Thread, como se describió anteriormente se hace uso de la biblioteca de plantillas Intel® TBB, teniendo en cuenta esto que podemos usarlas para crear nuestras clases y definir las acciones a realizar de manera paralizada. Para la Criba, solo fue necesario definir una clase Parallel Sieve, que recibirá la criba (arreglo) y de acuerdo con el tamaño de la misma, seguirá dividiéndolo y ejecutando la búsqueda de números múltiplos para al final unir las secciones inspeccionadas.

Resultados:

C++: Intel® TBB: (Appendix 9)

Parallel Sieve C++ TBB 10 times Average	
Limit	Time (ms)
10	0,615400
100	0,193800
1000	0,263800
10000	2,110300
100000	38,966700
1000000	1133,330000
10000000	30988,400000

Java: Fork Join Framework

Tal como lo describe la documentación oficial de Java, el framework fork / join es una implementación de la interfaz ExecutorService que le ayuda a aprovechar los múltiples procesadores. Está diseñado para el trabajo que se puede dividir recursivamente en piezas más pequeñas. El objetivo es utilizar toda la potencia de procesamiento disponible para mejorar el rendimiento de su aplicación.

Para la implementación de la Criba de Eratóstenes usando este framework, de manera similar a los métodos descritos anteriormente, es necesario crear una clase (ForkSieve) que sea una extensión de la clase RecursiveAction. En este caso, la clase recibirá la criba (arreglo) y, de acuerdo con un límite especificado, la criba se dividirá en trozos más pequeños para iterar sobre ellos en busca de los múltiplos de un número, similar a las técnicas anteriores, se enfoca en la idea de divide y conquistaras.

Estos fueron los resultados obtenidos:

Java Fork/Join Framework: (Appendix 10):

Parallel Sieve ForkJoin Java 10 times Average	
Limit	Time (ms)
10	1,700000
100	2,700000
1000	5,600000
10000	19,600000
100000	167,400000
1000000	1262,000000
10000000	24197,600000

NVIDIA: C/CUDA: GPU/CLOUD Computing

Como implementación final y extra de la Criba de Eratóstenes, daremos un salto a la nube, ya que esta implementación se está utilizando como una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de Amazon Web Services porque, como Herb Sutter (2008) dijo, estos servicios en la nube nos permiten tener acceso a una potencia informática increíble sin tener que gastar muchos recursos. **(Descripción del hardware y GPU en (Apéndice 11))**

Obtenido del sitio de desarrolladores de NVIDIA, CUDA® es una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA para computación general en unidades de procesamiento gráfico (GPU). Con CUDA, los desarrolladores pueden acelerar drásticamente las aplicaciones informáticas aprovechando el poder de las GPU.

En las aplicaciones aceleradas por GPU, la parte secuencial de la carga de trabajo se ejecuta en la CPU, que está optimizada para el rendimiento de un solo subproceso, mientras que la parte de uso intensivo de cómputo se ejecuta en miles de núcleos de GPU en paralelo. Cuando se utiliza CUDA, los desarrolladores programan en lenguajes populares como C, C++, Fortran, Python y MATLAB y expresan el paralelismo a través de extensiones en forma de algunas palabras clave básicas.

Para la implementación paralela de la Criba de Eratóstenes, la funcionalidad principal se basará en la ejecución de una simple función paralelizada, que gracias a la arquitectura de una GPU tendrá un rendimiento excepcional, esta función recibirá la criba (arreglo) y, dependiendo del número de hilos, bloques y dimensiones, se accederá a la ubicación exacta del arreglo para evaluarlo como un múltiplo o no de un número.

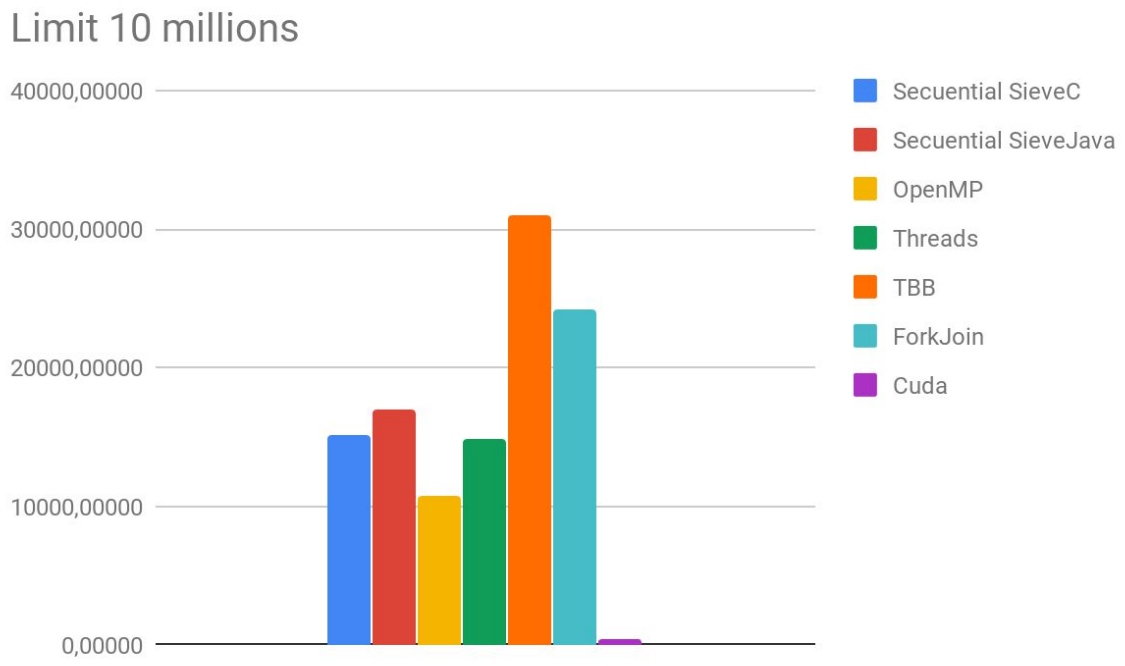
Resultados:

NVIDIA : C/CUDA: GPU/CLOUD COMPUTING (Appendix 12)

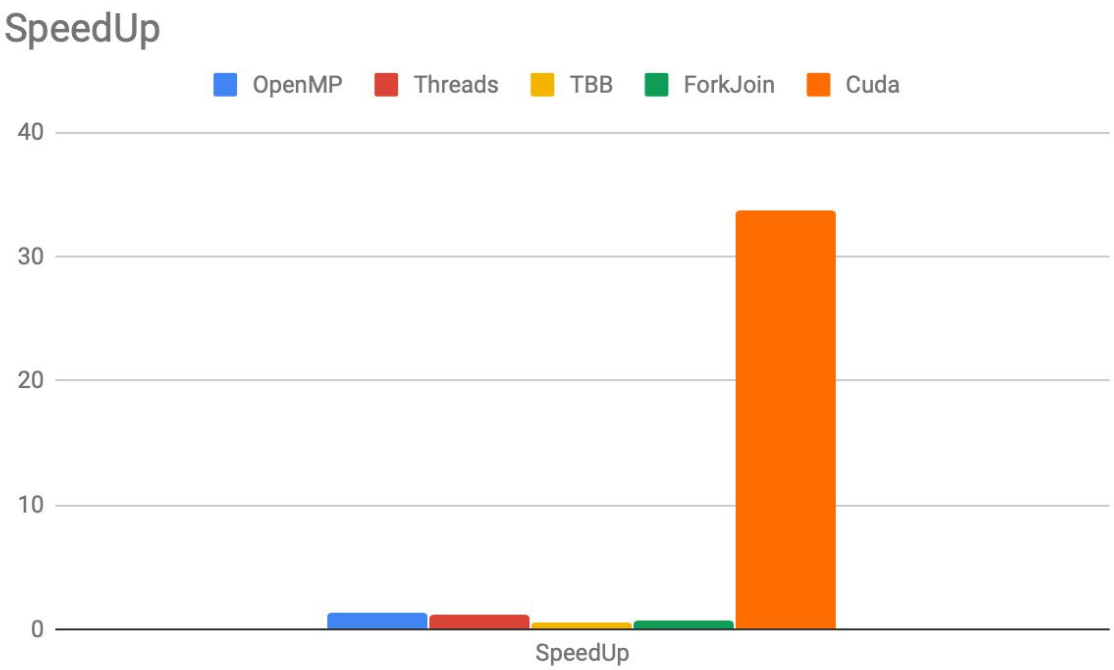
Parallel Sieve CUDA 10 times Average	
Limit	Time (ms)
10	6,689600
100	5,507600
1000	6,674100
10000	6,756000
100000	7,952100
1000000	30,255500
10000000	448,502800

Parallel Benchmarking:

Limit	Sequential Sieve C	Sequential Sieve Java	OpenMP	Threads	TBB	ForkJoin	Cuda
10	0,00140	0,00000	0,93840	0,10000	0,61540	1,70000	6,68960
100	0,00320	0,00000	0,39280	0,20000	0,19380	2,70000	5,50760
1000	0,04710	0,20000	0,41360	0,30000	0,26380	5,60000	6,67410
10000	1,10490	1,60000	1,80770	1,50000	2,11030	19,60000	6,75600
100000	22,76430	26,70000	18,61220	19,30000	38,96670	167,40000	7,95210
1000000	567,26780	650,60000	367,91900	558,10000	1133,33000	1262,00000	30,25550
10000000	15141,64080	16927,20000	10775,37330	14911,90000	30988,40000	24197,60000	448,50280
Limit	Sequential Sieve C	Sequential Sieve Java	OpenMP	Threads	TBB	ForkJoin	Cuda
10000000	15141,64080	16927,20000	10775,37330	14911,90000	30988,40000	24197,60000	448,50280



	OpenMP	Threads	TBB	ForkJoin	Cuda
SpeedUp	1,405208003	1,135147097	0,4886228653	0,6995404503	33,76041532



Resultados y Conclusiones

La computación paralela está aquí para quedarse, como se observó con las diferentes implementaciones la mayor parte del tiempo una implementación paralela mejorará el rendimiento del algoritmo con respecto a una implementación secuencial. Es importante recordar esto último, ya que no todos los problemas se pueden paralelizar, debemos tener en cuenta aspectos como las condiciones de carrera, “deadlocks”, alocación de memoria, las inconsistencias en la memoria y la herramienta adecuada para usar.

A través de la implementación de la Criba de Eratóstenes en estos 5 frameworks, tenemos otro punto a favor del cómputo paralelo, especialmente ahora con Cloud Computing, ya que como fue posible observar y comparar en el benchmark CUDA (que se estaba ejecutando en una instancia de Amazon Elastic Compute Cloud (Amazon EC2) de Amazon Web Services), fue la implementación que tuvo un rendimiento muy superior en comparación con la implementación secuencial y los otros frameworks debido a la arquitectura de una GPU, seguida de C con OpenMP y, sorprendentemente, Threads en Java en la implementación local.

References

Zaldívar, F. (2012). Introducción a la teoría de números. Retrieved from https://0-ebookcentral.proquest.com.millenium.itesm.mx

Sierpiński, Wacław (1988). Elementary Theory of Numbers. North-Holland Mathematical Library. 31 (2nd ed.). Elsevier.

Moore's Law and Intel Innovation. (2018). Retrieved from https://www.intel.com/content/www/us/en/history/museum-gordon-moore-law.html

The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. (2018). Retrieved from http://www.gotw.ca/publications/concurrency-ddj.htm

Welcome To The Jungle (2008). Herb Sutter. Retrieved from https://herbsutter.com/welcome-to-the-jungle/

Marowka, A. (2007). Parallel Computing on any Desktop. Retrieved from Communications of the ACM

Yılmaz, E. (2013). Pseudocode, Algorithms. Retrieved from http://www.cs.cornell.edu/courses/cs1109/2013su/lecs/lec06.pdf

OpenMP.org. (2013). About the OpenMP. Retrieved from https://www.openmp.org/about/about-us/

CUDA Zone | NVIDIA Developer. (2018) Retrived from https://developer.nvidia.com/cuda-zone

Java Fork/Join. (2018). Retrieved from https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html

Java Threads. (2018). Retrieved from https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

Threading Building Blocks. (2018). Retrieved from https://www.threadingbuildingblocks.org

Apéndices de código.

GitHub: https://github.com/RamonRomeroTec/TheSieveOfErastothenes

Más resultados y comandos de terminal adjuntos en Results.xlsx

Appendix 1: C: Brute Force Not Optimized

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>

struct timeval startTime, stopTime;
int started = 0;

void start_timer() {
 started = 1;
 gettimeofday(&startTime, NULL);
}

double stop_timer() {
 long seconds, useconds;
 double duration = -1;

 if (started) {
   gettimeofday(&stopTime, NULL);
   seconds  = stopTime.tv_sec  - startTime.tv_sec;
```

```
   useconds = stopTime.tv_usec - startTime.tv_usec;
   duration = (seconds * 1000.0) + (useconds / 1000.0);
   started = 0;
 }
 return duration;
}


void initarr(int limit,int *a) {
  int c;
  //#pragma omp parallel for shared(a, limit) private(c)
  for(c = 2; c < limit; c++) {
      a[c] = 0;
  }
}



void  printprimes(int limit, int *arr) {
  int c;
  //#pragma omp parallel for shared(arr, limit) private(c)
  for(c = 2; c <limit; c++) {
      if(arr[c] == 0) {
          fprintf(stdout,"%d ", c);
      }
  }
  fprintf(stdout,"\n");
  /* code */
```

```
}



int main(int argc, char **argv) {
  int N=10;
  int limit ;
  if (argc>3){
                   fprintf(stderr,     "Error:     uso:     %s
[limite_superior_positivo]\n", argv[0]);
    return -1;


  }else if (argc==2 || argc==3) {
    int parsed=atoi(argv[1]);
    if (parsed<0){
                        fprintf(stderr,    "Error:    uso:    %s
[limite_superior_positivo]\n", argv[0]);
      return -1;
    }else{
      limit=parsed;
    }
    if (argc==3) {
      N=1;
    }
  }else {
    limit=16;
  }
```

```
int *arr;

double ms;
ms = 0;
int i;
for (i = 0; i < N; i++) {
  start_timer();
  //->
  arr = (int*)malloc(limit * sizeof(int));
  memset(arr,0,limit*sizeof(int));
  int c;

  for(c = 2; c < limit; c++) {
      int m;
      for ( m = 1; m < c; m++) {
        if (c%m==0 && m!=1 && m!=c ){
          arr[c]=1;
          break;
        }else{
          arr[c]=0;
        }
      }

  }


  //->
  ms += stop_timer();
```

```
    }

    if (argc==2){
      printf("times %i - avg time = %.5lf ms\n",N,(ms / N));
    }


    printprimes(limit, arr);




    free(arr);

    return 0;
}
```

Appendix 2: Java : BruteForce Not Optimized

```
/*-------------------------------------------------------------
-----

* Autor: A01700318 Jose Ramon Romero Chavez
*-------------------------------------------------------------
--*/

public class Main {


     public static void  printprimes(int limit, int []arr) {
```

```
        int c;


        for(c = 2; c <limit; c++) {
            if(arr[c] == 0) {
                        System.out.print(""+c+" ");
            }
        }
           System.out.print("\n");
      }
     private     static     final     int     MAXTHREADS     =
Runtime.getRuntime().availableProcessors();


    public static void main(String[] args) {



                      //System.out.println("IO: "+args[0]+" "+
args.length);


                      int N = 10;
                int limit=16;
                if (args.length>2){
                          System.err.println("Error:  uso:  Main
[limite_superior_positivo]\n");
                  System.exit(-1);
                }else if (args.length==1 || args.length==2) {
                  int parsed=Integer.parseInt(args[0]);
                  //System.err.println(parsed);


                  if (parsed<0){
```

```
                        System.err.println("Error: uso: %s
[limite_superior_positivo]\n");
                  System.exit(-1);
                }else{
                  limit=parsed;
                }
                if (args.length==2){
                  N=1;

                }

              }


                  //initarr(limit, arr);

                  //
                  long startTime, stopTime;
                  double acum = 0;
                  acum = 0;
                  int [] arr = new int[limit];

                  for (int j = 1; j <= N; j++) {
                  startTime = System.currentTimeMillis();
                  arr = new int[limit];
          int c;
                  int m;

          for(c = 2; c < limit; c++) {
```

```
                for ( m = 1; m <  c; m++) {
                  if (c%m==0 && m!=1 && m!=c ){
                    arr[c]=1;
                    break;
                  }else{
                    arr[c]=0;
                  }
                }

            }


                        stopTime                                =
System.currentTimeMillis();
                        acum +=  (stopTime - startTime);
                    }
                    if (args.length==1){
                        System.out.printf("times  %d  -  avg
time = %.5f ms\n", N, (acum / N));
                    }
                    //


                printprimes(limit, arr);
```

```
        System.exit(0);
    }
}
```

Appendix 3: C :Brute Force Optimized

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>



struct timeval startTime, stopTime;
int started = 0;

void start_timer() {
 started = 1;
 gettimeofday(&startTime, NULL);
}

double stop_timer() {
 long seconds, useconds;
 double duration = -1;
```

```
 if (started) {
   gettimeofday(&stopTime, NULL);
   seconds  = stopTime.tv_sec  - startTime.tv_sec;
   useconds = stopTime.tv_usec - startTime.tv_usec;
   duration = (seconds * 1000.0) + (useconds / 1000.0);
   started = 0;
 }
 return duration;
}
```

```
void initarr(int limit,int *a) {
  int c;
  //#pragma omp parallel for shared(a, limit) private(c)
  for(c = 2; c < limit; c++) {
      a[c] = 0;
  }
}
```

```
void  printprimes(int limit, int *arr) {
  int c;
  //#pragma omp parallel for shared(arr, limit) private(c)
  for(c = 2; c <limit; c++) {
      if(arr[c] == 0) {
          fprintf(stdout,"%d ", c);
      }
  }
  fprintf(stdout,"\n");
  /* code */
}



int main(int argc, char **argv) {
  int N=10;
  int limit ;
  if (argc>3){
                   fprintf(stderr,     "Error:     uso:     %s
[limite_superior_positivo]\n", argv[0]);
    return -1;


  }else if (argc==2 || argc==3) {
    int parsed=atoi(argv[1]);
    if (parsed<0){
                        fprintf(stderr,    "Error:    uso:    %s
[limite_superior_positivo]\n", argv[0]);
      return -1;
    }else{
```

```
    limit=parsed;
  }
  if (argc==3) {
    N=1;
  }
}else {
  limit=16;
}




  int *arr;

  double ms;
  ms = 0;
  int i;
  for (i = 0; i < N; i++) {
    start_timer();
    //->
    arr = (int*)malloc(limit * sizeof(int));
    memset(arr,0,limit*sizeof(int));
    int c;

    for(c = 2; c < limit; c++) {
        int m;
        for ( m = 1; m < (int)sqrt(c); m++) {
          if (c%m==0 && m!=1 && m!=c ){
```

```
          arr[c]=1;
          break;
        }else{
          arr[c]=0;
        }
      }

  }









  //->
  ms += stop_timer();
}

if (argc==2){
  printf("times %i - avg time = %.5lf ms\n",N,(ms / N));
}
```

```
    //



    printprimes(limit, arr);




    free(arr);

    return 0;
}
```

Appendix 4: Java: Brute Force Optimized

```
/*-------------------------------------------------------------
-----

* Autor: A01700318 Jose Ramon Romero Chavez
*-------------------------------------------------------------
--*/

public class Main {


     public static void  printprimes(int limit, int []arr) {
       int c;
```

```
       for(c = 2; c <limit; c++) {
           if(arr[c] == 0) {
                        System.out.print(""+c+" ");
           }
       }
          System.out.print("\n");
     }
     private     static     final     int     MAXTHREADS     =
Runtime.getRuntime().availableProcessors();

    public static void main(String[] args) {


                      //System.out.println("IO: "+args[0]+" "+
args.length);


                      int N = 10;
               int limit=16;
               if (args.length>2){
                        System.err.println("Error:  uso:  Main
[limite_superior_positivo]\n");
                 System.exit(-1);
               }else if (args.length==1 || args.length==2) {
                 int parsed=Integer.parseInt(args[0]);
                 //System.err.println(parsed);


                 if (parsed<0){
                          System.err.println("Error:  uso:  %s
[limite_superior_positivo]\n");
                   System.exit(-1);
```

```
      }else{
        limit=parsed;
      }
      if (args.length==2){
        N=1;

      }

    }


        //initarr(limit, arr);

        //
        long startTime, stopTime;
        double acum = 0;
        acum = 0;
        int [] arr = new int[limit];

        for (int j = 1; j <= N; j++) {
        startTime = System.currentTimeMillis();
        arr = new int[limit];
int c;
        int m;

for(c = 2; c < limit; c++) {
    for ( m = 1; m <  (int)Math.sqrt(c); m++) {
      if (c%m==0 && m!=1 && m!=c ){
```

```
                    arr[c]=1;
                    break;
                  }else{
                    arr[c]=0;
                  }
                }

            }


                          stopTime                              =
System.currentTimeMillis();
                          acum +=  (stopTime - startTime);
                    }
                    if (args.length==1){
                          System.out.printf("times  %d  -  avg
time = %.5f ms\n", N, (acum / N));
                    }
                    //


                printprimes(limit, arr);









        System.exit(0);
    }
}
```

Appendix 5: C: The Sieve of Eratosthenes Sequential

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <time.h>
#include <sys/time.h>
#include <sys/types.h>



struct timeval startTime, stopTime;
int started = 0;

void start_timer() {
 started = 1;
 gettimeofday(&startTime, NULL);
}

double stop_timer() {
 long seconds, useconds;
 double duration = -1;

 if (started) {
   gettimeofday(&stopTime, NULL);
```

```
    seconds  = stopTime.tv_sec  - startTime.tv_sec;
    useconds = stopTime.tv_usec - startTime.tv_usec;
    duration = (seconds * 1000.0) + (useconds / 1000.0);
    started = 0;
  }
  return duration;
}
```

```
void initarr(int limit,int *a) {
  int c;
  //#pragma omp parallel for shared(a, limit) private(c)
  for(c = 2; c < limit; c++) {
      a[c] = 0;
  }
}
```

```
void  printprimes(int limit, int *arr) {
  int c;
  //#pragma omp parallel for shared(arr, limit) private(c)
```

```
  for(c = 2; c <limit; c++) {
      if(arr[c] == 0) {
          fprintf(stdout,"%d ", c);
      }
  }
  fprintf(stdout,"\n");
  /* code */
}
```

```
int main(int argc, char **argv) {
  int N=10;
  int limit ;
  if (argc>3){
                    fprintf(stderr,     "Error:     uso:     %s
[limite_superior_positivo]\n", argv[0]);
    return -1;

  }else if (argc==2 || argc==3) {
    int parsed=atoi(argv[1]);
    if (parsed<0){
                        fprintf(stderr,    "Error:    uso:    %s
[limite_superior_positivo]\n", argv[0]);
      return -1;
    }else{
      limit=parsed;
    }
    if (argc==3) {
```

```
      N=1;
    }
  }else {
    limit=16;
  }




    int *arr;

    double ms;
    ms = 0;
    int i;
    for (i = 0; i < N; i++) {
      start_timer();
      //->
      int sqroot = (int)sqrt(limit);
      arr = (int*)malloc(limit * sizeof(int));
      //initarr(limit, arr);
      memset(arr,0,limit*sizeof(int));
      int c;
      int m;
      for(c = 2; c <= sqroot; c++) {
          if(arr[c] == 0) {
             //#pragma omp parallel for shared(arr, limit, c)
private(m)
              for(m = c+1; m < limit; m++) {
                  if(m%c == 0) {
```

```
                    arr[m] = 1;
                }
            }

        }
    }
```

```
    //->
    ms += stop_timer();
}

if (argc==2){
    printf("times %i - avg time = %.5lf ms\n",N,(ms / N));
}
```

```
printprimes(limit, arr);
```

```
    free(arr);

    return 0;
}
```

Appendix 6: Java: The Sieve of Eratosthenes Sequential

```
/*-------------------------------------------------------------
-----

* Autor: A01700318 Jose Ramon Romero Chavez
*-------------------------------------------------------------
--*/

public class Main {



     public static void  printprimes(int limit, int []arr) {
       int c;

       for(c = 2; c <limit; c++) {
           if(arr[c] == 0) {
                        System.out.print(""+c+" ");
           }
       }
```

```
          System.out.print("\n");
     }
     private     static     final     int     MAXTHREADS     =
Runtime.getRuntime().availableProcessors();

    public static void main(String[] args) {


                    //System.out.println("IO: "+args[0]+" "+
args.length);


                    int N = 10;
               int limit=16;
               if (args.length>2){
                       System.err.println("Error:  uso:  Main
[limite_superior_positivo]\n");
                 System.exit(-1);
               }else if (args.length==1 || args.length==2) {
                 int parsed=Integer.parseInt(args[0]);
                 //System.err.println(parsed);


                 if (parsed<0){
                         System.err.println("Error:  uso:  %s
[limite_superior_positivo]\n");
                   System.exit(-1);
                 }else{
                   limit=parsed;
                 }
                 if (args.length==2){
                   N=1;
```

```
                }

            }


                //initarr(limit, arr);

                //
                long startTime, stopTime;
                double acum = 0;
                acum = 0;
                int [] arr = new int[limit];

                for (int j = 1; j <= N; j++) {
                startTime = System.currentTimeMillis();
                int sqroot = (int)Math.sqrt(limit);
                arr = new int[limit];
        int c;
        int m;
      for(c = 2; c <= sqroot; c++) {
          if(arr[c] == 0) {
             //#pragma omp parallel for shared(arr, limit, c)
private(m)
             for(m = c+1; m < limit; m++) {
                 if(m%c == 0) {
                     arr[m] = 1;
                 }
             }
```

```
            }
        }


                            stopTime                                =
System.currentTimeMillis();
                            acum +=  (stopTime - startTime);
                        }
                        if (args.length==1){
                            System.out.printf("times  %d  -  avg
time = %.5f ms\n", N, (acum / N));
                        }
                        //


                printprimes(limit, arr);









          System.exit(0);
    }
}
```

Appendix 7: C: OpenMP : The Sieve of Eratosthenes Parallel

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <string.h>

struct timeval startTime, stopTime;
int started = 0;

void start_timer() {
 started = 1;
 gettimeofday(&startTime, NULL);
}

double stop_timer() {
 long seconds, useconds;
 double duration = -1;

 if (started) {
   gettimeofday(&stopTime, NULL);
   seconds  = stopTime.tv_sec  - startTime.tv_sec;
   useconds = stopTime.tv_usec - startTime.tv_usec;
   duration = (seconds * 1000.0) + (useconds / 1000.0);
   started = 0;
```

```
 }
 return duration;
}
```

```
void initarr(int limit,int *a) {
  int c;
  #pragma omp parallel for shared(a, limit) private(c)
  for(c = 2; c < limit; c++) {
      a[c] = 0;
  }
}
```

```
void  printprimes(int limit, int *arr) {
  int c;
  //#pragma omp parallel for shared(arr, limit) private(c)
  for(c = 2; c <limit; c++) {
      if(arr[c] == 0) {
          fprintf(stdout,"%d ", c);
      }
```

```
  }
  fprintf(stdout,"\n");
  /* code */
}



int main(int argc, char **argv) {
  int N=10;
  int limit ;
  if (argc>3){
                   fprintf(stderr,     "Error:     uso:     %s
[limite_superior_positivo]\n", argv[0]);
    return -1;

  }else if (argc==2 || argc==3) {
    int parsed=atoi(argv[1]);
    if (parsed<0){
                        fprintf(stderr,    "Error:    uso:    %s
[limite_superior_positivo]\n", argv[0]);
      return -1;
    }else{
      limit=parsed;
    }
    if (argc==3) {
      N=1;
    }
  }else {
    limit=16;
```

```
  }



    int *arr;

    double ms;
    ms = 0;
    int i;
    for (i = 0; i < N; i++) {
      start_timer();
      //->
      int sqroot = (int)sqrt(limit);
      arr = (int*)malloc(limit * sizeof(int));
      //initarr(limit, arr);
      memset(arr,0,limit*sizeof(int));

      int c;
      int m;
      for(c = 2; c <= sqroot; c++) {
          if(arr[c] == 0) {
               #pragma omp parallel for shared(arr, limit, c)
private(m)
              for(m = c+1; m < limit; m++) {
                  if(m%c == 0) {
                      arr[m] = 1;
                  }
              }
```

```
        }
    }
```

```
    //->
    ms += stop_timer();
}
if (argc==2){
    printf("times %i - avg time = %.5lf ms\n",N,(ms / N));
}
```

```
//
```

```
    printprimes(limit, arr);



    free(arr);

    return 0;
}
```

Appendix 8: Java: Threads : The Sieve of Eratosthenes Parallel

```
/*-------------------------------------------------------------
-----

* Autor: A01700318 Jose Ramon Romero Chavez
*-------------------------------------------------------------
--*/

public class Main {



     public static void initarr(int limit,int []a) {
       int c;
       for(c = 2; c < limit; c++) {
           a[c] = 0;
       }
     }
```

```
     public static void  printprimes(int limit, int []arr) {
       int c;

       for(c = 2; c <limit; c++) {
           if(arr[c] == 0) {
                        System.out.print(""+c+" ");
           }
       }
          System.out.print("\n");
       /* code */
     }
     private     static     final     int     MAXTHREADS     =
Runtime.getRuntime().availableProcessors();

    public static void main(String[] args) {


                     //System.out.println("IO: "+args[0]+" "+
args.length);


                     int N = 10;
               int limit=16;
               if (args.length>2){
                        System.err.println("Error:  uso:  Main
[limite_superior_positivo]\n");
                 System.exit(-1);
               }else if (args.length==1 || args.length==2) {
                 int parsed=Integer.parseInt(args[0]);
                 //System.err.println(parsed);
```

```
                if (parsed<0){
                        System.err.println("Error: uso: %s
[limite_superior_positivo]\n");
                  System.exit(-1);
                }else{
                  limit=parsed;
                }
                if (args.length==2){
                  N=1;

                }

              }


                   //initarr(limit, arr);

                   //
                   long startTime, stopTime;
                   double acum = 0;
                   acum = 0;
                   int [] arr = new int[limit];

                   for (int j = 1; j <= N; j++) {
                   startTime = System.currentTimeMillis();
                   int a=(int)(limit/MAXTHREADS);
                   int sqroot = (int)Math.sqrt(limit);
                   arr = new int[limit];
```

```
                    SieveThread          []          hilos=new
SieveThread[MAXTHREADS];

        for (int i = 0; i < MAXTHREADS; i++) {
                if (i != MAXTHREADS - 1) {
                                                    hilos[i]=  new
SieveThread((a*i),(a*(i+1)), arr);
                } else {
                          hilos[i] = new SieveThread((i * a),
limit, arr);
                }
        }

                    int c;
                    int m;



                    for(c = 2; c <= sqroot; c++) {
                              if(arr[c] == 0) {

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

     hilos[i].ct=c;
                                        }


                                        for  (int  i  =  0;  i  <
MAXTHREADS ; i++) {
```

```
                                        hilos[i].run();
                                    }


                                }
                    }
                    try {
                          for (int i = 0; i < MAXTHREADS ;
i++) {
                                    hilos[i].join();
                          }
                    } catch (InterruptedException e) {
                          e.printStackTrace();
                    }




                          stopTime                          =
System.currentTimeMillis();
                          acum +=  (stopTime - startTime);
                    }
                    if (args.length==1){
                          System.out.printf("times  %d  -  avg
time = %.5f ms\n", N, (acum / N));
                    }
                    //


               printprimes(limit, arr);
```

```
        System.exit(0);
    }
}



class SieveThread extends Thread {
     public int [] arr;
     public int ct=-1;
           public int begin;
           public int end;


    public SieveThread( int b, int e, int [] a ) {
        this.arr     = a;
        this.begin  = b;
        this.end   = e;
    }

    public int[] getArr() {
        return arr;
    }
```

```
    public void run() {
                for (int i = this.begin; i != this.end; i++) {

                    this.arr[ct] = 0;
                    //System.err.println(ct+" "+i);

                    if(i%ct == 0) {
                              this.arr[i] = 1;

                    }

                }

    }

}
```

Appendix 9: C++: Intel® Threading Building Blocks (TBB) : The Sieve of Eratosthenes Parallel

```
#include <math.h>
#include <iostream>
#include <tbb/parallel_for.h>
#include <tbb/parallel_reduce.h>
```

```
using namespace std;
using namespace tbb;

const int GRAIN = 100000;



const int N = 10;
const int DISPLAY = 100;
const int MAX_VALUE = 10000;

class Timer {
private:
    timeval startTime;
    bool  started;

public:
    Timer() :started(false) {}

    void start(){
     started = true;
        gettimeofday(&startTime, NULL);
    }

    double stop(){
        timeval endTime;
```

```
        long seconds, useconds;
        double duration = -1;

        if (started) {
               gettimeofday(&endTime, NULL);

               seconds  = endTime.tv_sec  - startTime.tv_sec;
               useconds       =       endTime.tv_usec       -
startTime.tv_usec;

               duration  =  (seconds  *  1000.0)  +  (useconds  /
1000.0);
               started = false;
        }
          return duration;
    }
};

/*
State Pattern for Initialization


class ParallelSieve {

public:
     int *myArray;
     int c;
     int state;
     ParallelSieve(int   *array,   int   cm,   int   s)   :
myArray(array), c(cm), state(s) {}
```

```
    ParallelSieve(ParallelSieve &x, split)
        : myArray(x.myArray), c(x.c) , state(x.state) {}
    void operator() (const blocked_range<int> &r) {
        if (state==0){
            for (int i = r.begin(); i < r.end(); i++) {
                    myArray[i] = 0;
        }
    }    else{
            for (int i = r.begin()+1; i != r.end(); i++) {
                myArray[c] = 0;
                if(i%c == 0) {
                        myArray[i] = 1;

                }
        }
    }
}




    void join(const ParallelSieve &x) {
        myArray = x.myArray;
    }
};
```

```
*/


class ParallelSieve {


public:
     int *myArray;
     int c;
     ParallelSieve(int  *array,  int  cm)  :  myArray(array),
c(cm)  {}


     ParallelSieve(ParallelSieve &x, split)
          : myArray(x.myArray), c(x.c) {}
     void operator() (const blocked_range<int> &r) {


               for (int i = r.begin()+1; i != r.end(); i++) {
                    myArray[c] = 0;
                    if(i%c == 0) {
                              myArray[i] = 1;


                    }
          }


}


     void join(const ParallelSieve &x) {
          myArray = x.myArray;
     }
};
```

```
void  printprimes(int limit, int *arr) {
  int c;

  for(c = 2; c <limit; c++) {
      if(arr[c] == 0) {
          fprintf(stdout,"%d ", c);
      }
  }
  fprintf(stdout,"\n");
  /* code */
}



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

     int N=10;
  int limit ;
  if (argc>3){
                   fprintf(stderr,     "Error:     uso:     %s
[limite_superior_positivo]\n", argv[0]);
    return -1;

  }else if (argc==2 || argc==3) {
    int parsed=atoi(argv[1]);
    if (parsed<0){
```

```
                        fprintf(stderr,    "Error:    uso:    %s
[limite_superior_positivo]\n", argv[0]);
      return -1;
    }else{
      limit=parsed;
    }
    if (argc==3) {
      N=1;
    }
  }else {
    limit=16;
  }

     Timer t;
     double ms;
     double result;
     ms = 0;
     int *arr ;

     for (int i = 0; i < N; i++) {
     t.start();




//

int sqroot = (int)sqrt(limit);
arr = (int*)malloc(limit * sizeof(int));
```

```
if(arr == NULL) {
          fprintf(stderr, "Error: Failed to allocate memory
for arr.\n");
          return -1;
}

//initarr(limit, arr);

int c=2;
int m;

for(c = 2; c <= sqroot; c++) {
          if(arr[c] == 0) {

      ParallelSieve  init(arr, c);
               parallel_reduce( blocked_range<int>(0, limit,
GRAIN),init );
               arr = init.myArray;

          }
}

//
```

```
            ms += t.stop();
            }

            if (argc==2){
                 cout << "times "<< N <<" - avg time = " <<
(ms/N) << " ms" << endl;
    }

    printprimes(limit, arr);



    free(arr);

    return 0;
}
```

Appendix 10: Java: Fork/Join Framework : The Sieve of Eratosthenes Parallel

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class Main {


     public static void initarr(int limit,int []a) {
       int c;
```

```
        for(c = 2; c < limit; c++) {
            a[c] = 0;
        }
      }

      public static void  printprimes(int limit, int []arr) {
        int c;

        for(c = 2; c <limit; c++) {
            if(arr[c] == 0) {
                        System.out.print(""+c+" ");
            }
        }
           System.out.print("\n");
        /* code */
      }

              private   static   final   int   MAXTHREADS   =
Runtime.getRuntime().availableProcessors();

      public static void main(String args[]) throws Exception
{

    int N = 10;
    int limit=16;
    if (args.length>2){
                      System.err.println("Error:   uso:   Main
[limite_superior_positivo]\n");
       System.exit(-1);
    }else if (args.length==1 || args.length==2) {
```

```
      int parsed=Integer.parseInt(args[0]);
      //System.err.println(parsed);

      if (parsed<0){
                          System.err.println("Error:   uso:   %s
[limite_superior_positivo]\n");
        System.exit(-1);
      }else{
        limit=parsed;
      }
      if (args.length==2){
        N=1;

      }

    }

    long startTime, stopTime;
    double acum = 0;
    acum = 0;
    int [] arr = new int[limit];


    for (int j = 1; j <= N; j++) {
    startTime = System.currentTimeMillis();
    arr = new int[limit];
```

```
//

int sqroot = (int)Math.sqrt(limit);
ForkJoinPool pool = new ForkJoinPool(MAXTHREADS);;
//

//
//initarr(limit, arr);
int c;
int m;


for(c = 2; c <= sqroot; c++) {
    if(arr[c] == 0) {

      //
        pool.invoke(new ForkSieve(0, limit, arr, c));

      //


    }
}
//
  stopTime = System.currentTimeMillis();
  acum +=  (stopTime - startTime);
}
```

```
    if (args.length==1){
      System.out.printf("times %d - avg time = %.5f ms\n", N,
(acum / N));
    }

  printprimes(limit, arr);

    System.exit(0);

     }
```

```
}
/*-------------------------------------------------------------
-----

*

* Actividad de programación: Fork-join framework

* Fecha: 23-Sep-2018

* Autor: A01700318 Ramon Romero

*

*-------------------------------------------------------------
--*/

class ForkSieve extends RecursiveAction {
     private static final long MIN = 10;
  public int [] arr;
  public int ct=-1;
  public int begin;
  public int end;


  public ForkSieve( int b, int e, int [] a, int c ) {
     this.arr   = a;
     this.begin = b;
     this.end   = e;
```

```
    this.ct    = c;
  }


    public void computeDirectly() {

   for (int i = this.begin; i < this.end; i++) {
     //System.err.println(ct+" "+i);

     this.arr[ct] = 0;
     if(i%ct == 0) {
         this.arr[i] = 1;

     }

   }
    }

    @Override
    protected void compute() {
         if ( (this.end - this.begin) <= ForkSieve.MIN ) {
              computeDirectly();

         } else {
              int middle = (end + begin) / 2;

              invokeAll(new  ForkSieve(  begin,  middle,  arr,
ct),
```

```
                              new ForkSieve( middle, end, arr,
ct));
          }
     }
}
```

Appendix 11: NVIDIA: C/CUDA: GPU / Cloud Computing: Hardware

```
H/W path    Device  Class      Description
=========================================
                    system     HVM domU
/0                  bus        Motherboard
/0/0                memory     96KiB BIOS
/0/401                processor  Intel(R) Xeon(R) CPU E5-2686
v4 @ 2.30GHz
/0/402              processor  CPU
/0/403              processor  CPU
/0/404              processor  CPU
/0/1000             memory     61GiB System Memory
/0/1000/0           memory     16GiB DIMM RAM
/0/1000/1           memory     16GiB DIMM RAM
/0/1000/2           memory     16GiB DIMM RAM
/0/1000/3           memory     13GiB DIMM RAM
/0/100              bridge     440FX - 82441FX PMC [Natoma]
/0/100/1                    bridge        82371SB  PIIX3  ISA
[Natoma/Triton II]
/0/100/1.1                  storage       82371SB  PIIX3  IDE
[Natoma/Triton II]
/0/100/1.3          bridge     82371AB/EB/MB PIIX4 ACPI
```

```
/0/100/2            display    GD 5446
/0/100/3    ens3    network    Ethernet interface
/0/100/1e           display    GK210GL [Tesla K80]
/0/100/1f           generic    Xen Platform Device

ubuntu@ip-172-31-41-114:~/ramoncuda$ sudo lshw
ip-172-31-41-114
    description: Computer
    product: HVM domU
    vendor: Xen
    version: 4.2.amazon
    serial: ec211522-4aa4-9cf2-d830-86869fcf74b0
    width: 64 bits
    capabilities: smbios-2.7 dmi-2.7 vsyscall32
                              configuration:       boot=normal
uuid=221521EC-A44A-F29C-D830-86869FCF74B0
  *-core
       description: Motherboard
       physical id: 0
     *-firmware
          description: BIOS
          vendor: Xen
          physical id: 0
          version: 4.2.amazon
          date: 08/24/2006
          size: 96KiB
          capabilities: pci edd
     *-cpu:0
          description: CPU
```

```
          product: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
          vendor: Intel Corp.
          physical id: 401
          bus info: cpu@0
          slot: CPU 1
          size: 2701MHz
          capacity: 3GHz
          width: 64 bits
          capabilities: fpu fpu_exception wp vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx pdpe1gb rdtscp x86-64 constant_tsc rep_good nopl xtopology nonstop_tsc aperfmperf pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single kaiser fsgsbase bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx xsaveopt cpufreq
     *-cpu:1
          description: CPU
          vendor: Intel
          physical id: 402
          bus info: cpu@1
          slot: CPU 2
          size: 2697MHz
          capacity: 3GHz
          capabilities: cpufreq
     *-cpu:2
          description: CPU
          vendor: Intel
          physical id: 403
          bus info: cpu@2
```

```
     slot: CPU 3
     size: 2241MHz
     capacity: 3GHz
     capabilities: cpufreq
*-cpu:3
     description: CPU
     vendor: Intel
     physical id: 404
     bus info: cpu@3
     slot: CPU 4
     size: 2699MHz
     capacity: 3GHz
     capabilities: cpufreq
*-memory
     description: System Memory
     physical id: 1000
     size: 61GiB
   *-bank:0
        description: DIMM RAM
        physical id: 0
        slot: DIMM 0
        size: 16GiB
        width: 64 bits
   *-bank:1
        description: DIMM RAM
        physical id: 1
        slot: DIMM 1
        size: 16GiB
```

```
          width: 64 bits
     *-bank:2
          description: DIMM RAM
          physical id: 2
          slot: DIMM 2
          size: 16GiB
          width: 64 bits
     *-bank:3
          description: DIMM RAM
          physical id: 3
          slot: DIMM 3
          size: 13GiB
          width: 64 bits
*-pci
     description: Host bridge
     product: 440FX - 82441FX PMC [Natoma]
     vendor: Intel Corporation
     physical id: 100
     bus info: pci@0000:00:00.0
     version: 02
     width: 32 bits
     clock: 33MHz
   *-isa
          description: ISA bridge
          product: 82371SB PIIX3 ISA [Natoma/Triton II]
          vendor: Intel Corporation
          physical id: 1
          bus info: pci@0000:00:01.0
```

```
             version: 00
             width: 32 bits
             clock: 33MHz
             capabilities: isa bus_master
             configuration: latency=0
        *-ide
             description: IDE interface
             product: 82371SB PIIX3 IDE [Natoma/Triton II]
             vendor: Intel Corporation
             physical id: 1.1
             bus info: pci@0000:00:01.1
             version: 00
             width: 32 bits
             clock: 33MHz
             capabilities: ide bus_master
             configuration: driver=ata_piix latency=64
               resources: irq:0 ioport:1f0(size=8) ioport:3f6
ioport:170(size=8) ioport:376 ioport:c100(size=16)
        *-bridge UNCLAIMED
             description: Bridge
             product: 82371AB/EB/MB PIIX4 ACPI
             vendor: Intel Corporation
             physical id: 1.3
             bus info: pci@0000:00:01.3
             version: 01
             width: 32 bits
             clock: 33MHz
             capabilities: bridge bus_master
             configuration: latency=0
```

```
        *-display:0 UNCLAIMED
             description: VGA compatible controller
             product: GD 5446
             vendor: Cirrus Logic
             physical id: 2
             bus info: pci@0000:00:02.0
             version: 00
             width: 32 bits
             clock: 33MHz
             capabilities: vga_controller bus_master
             configuration: latency=0
                         resources:  memory:80000000-81ffffff
memory:86004000-86004fff
        *-network
             description: Ethernet interface
             physical id: 3
             bus info: pci@0000:00:03.0
             logical name: ens3
             version: 00
             serial: 0a:ca:af:93:f8:44
             width: 32 bits
             clock: 33MHz
                    capabilities:  pciexpress  msix  bus_master
cap_list ethernet physical
                     configuration:  broadcast=yes  driver=ena
driverversion=1.5.0K   ip=172.31.41.114   latency=0   link=yes
multicast=yes
             resources: irq:0 memory:86000000-86003fff
        *-display:1
             description: 3D controller
```

```
             product: GK210GL [Tesla K80]
             vendor: NVIDIA Corporation
             physical id: 1e
             bus info: pci@0000:00:1e.0
             version: a1
             width: 64 bits
             clock: 33MHz
                  capabilities: pm msi pciexpress bus_master
cap_list
             configuration: driver=nvidia latency=248
                           resources: iomemory:100-ff irq:79
memory:84000000-84ffffff         memory:1000000000-13fffffff
memory:82000000-83ffffff
        *-generic
             description: Unassigned class
             product: Xen Platform Device
             vendor: XenSource, Inc.
             physical id: 1f
             bus info: pci@0000:00:1f.0
             version: 01
             width: 32 bits
             clock: 33MHz
             capabilities: bus_master
             configuration: driver=xen-platform-pci latency=0
                     resources: irq:47 ioport:c000(size=256)
memory:85000000-85ffffff


CUDA Device Query...
There are 1 CUDA devices.
```

```
CUDA Device #0
Major revision number:         3
Minor revision number:         7
Name:                          Tesla K80
Total global memory:           11996954624
Total shared memory per block: 49152
Total registers per block:     65536
Warp size:                     32
Maximum memory pitch:          2147483647
Maximum threads per block:     1024
Maximum dimension 0 of block:  1024
Maximum dimension 1 of block:  1024
Maximum dimension 2 of block:  64
Maximum dimension 0 of grid:   2147483647
Maximum dimension 1 of grid:   65535
Maximum dimension 2 of grid:   65535
Clock rate:                    823500
Total constant memory:         65536
Texture alignment:             512
Concurrent copy and execution: Yes
Number of multiprocessors:     13
Kernel execution timeout:      No
```

Appendix 12: NVIDIA: C/CUDA: GPU / Cloud Computing: The Sieve of Eratosthenes Parallel

```
#include <math.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>


struct timeval startTime, stopTime;
int started = 0;

void start_timer() {
 started = 1;
 gettimeofday(&startTime, NULL);
}

double stop_timer() {
 long seconds, useconds;
 double duration = -1;

 if (started) {
   gettimeofday(&stopTime, NULL);
   seconds  = stopTime.tv_sec  - startTime.tv_sec;
   useconds = stopTime.tv_usec - startTime.tv_usec;
   duration = (seconds * 1000.0) + (useconds / 1000.0);
   started = 0;
 }
 return duration;
}
```

```
void  printprimes(int limit, int *arr) {
  int c;
  //#pragma omp parallel for shared(arr, limit) private(c)
  for(c = 2; c <limit; c++) {
      if(arr[c] == 0) {
          fprintf(stdout,"%d ", c);
      }
  }
  fprintf(stdout,"\n");
  /* code */
}
```

```
__global__ void init(int *arr, int sqroot, int limit) {
  int c;
  for(c = 2; c <= sqroot; c++) {
      if(arr[c] == 0) {
          /*
              #pragma omp parallel for shared(arr, limit, c)
private(m)
          for(m = c+1; m < limit; m++) {
              if(m%c == 0) {
                  arr[m] = 1;
              }
          }
```

```
          */
                int tid = c+1+ threadIdx.x + (blockIdx.x *
blockDim.x);
          if (tid<limit){
            if (tid % c ==0) {
                    arr[tid] = 1;
                }
          }



      }
  }
}


double getThreadAndInfo(cudaDeviceProp devProp)
{
  /*
       printf("Major revision number:                %d\n",
devProp.major);
       printf("Minor revision number:                %d\n",
devProp.minor);
     printf("Name:                                   %s\n",
devProp.name);
      printf("Total global memory:                  %lu\n",
devProp.totalGlobalMem);
        printf("Total shared memory per block:  %lu\n",
devProp.sharedMemPerBlock);
       printf("Total registers per block:          %d\n",
devProp.regsPerBlock);
     printf("Warp size:                              %d\n",
devProp.warpSize);
```

```
       printf("Maximum memory pitch:                  %lu\n",
devProp.memPitch);

        printf("Maximum threads per block:           %i\n",
devProp.maxThreadsPerBlock);

    for (int i = 0; i < 3; ++i)

       printf("Maximum dimension %d of block:   %d\n", i,
devProp.maxThreadsDim[i]);

    for (int i = 0; i < 3; ++i)

       printf("Maximum dimension %d of grid:    %d\n", i,
devProp.maxGridSize[i]);

      printf("Clock rate:                             %d\n",
devProp.clockRate);

       printf("Total constant memory:                %lu\n",
devProp.totalConstMem);

      printf("Texture alignment:                     %lu\n",
devProp.textureAlignment);

         printf("Concurrent copy and execution:  %s\n",
(devProp.deviceOverlap ? "Yes" : "No"));

        printf("Number of multiprocessors:           %d\n",
devProp.multiProcessorCount);

        printf("Kernel execution timeout:            %s\n",
(devProp.kernelExecTimeoutEnabled ? "Yes" : "No"));

    */

    return devProp.maxThreadsPerBlock;

}


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

    // Number of CUDA devices

    int threads=1000000;

    int devCount;

    cudaGetDeviceCount(&devCount);

    //printf("CUDA Device Query...\n");
```

```
    //printf("There are %d CUDA devices.\n", devCount);

    // Iterate through devices
    for (int i = 0; i < devCount; ++i)
    {
        // Get device properties
        //printf("\nCUDA Device #%d\n", i);
        cudaDeviceProp devProp;
       cudaGetDeviceProperties(&devProp, i);
       if (getThreadAndInfo(devProp)<threads){
         threads=getThreadAndInfo(devProp);
       }

    }

    int N=10;
    int limit ;
    if (argc>3){
                        fprintf(stderr,    "Error:    uso:    %s
[limite_superior_positivo]\n", argv[0]);
      return -1;

    }else if (argc==2 || argc==3) {
      int parsed=atoi(argv[1]);
      if (parsed<0){
                          fprintf(stderr,   "Error:   uso:   %s
[limite_superior_positivo]\n", argv[0]);
        return -1;
      }else{
```

```
    limit=parsed;
  }
  if (argc==3) {
    N=1;
  }
}else {
  limit=16;
}
```

```
  int *arr;

  double ms;
  ms = 0;
  int i;
  int *p_array;
  for (i = 0; i < N; i++) {
    start_timer();
    //->
    int sqroot = (int)sqrt(limit);
    arr = (int*)malloc(limit * sizeof(int));
    cudaMalloc((void**) &p_array, limit * sizeof(int));
    cudaMemset(p_array, 0, limit*sizeof(int));
    if (limit<=threads){
      threads=limit;
      init<<<1, threads>>>(p_array, sqroot, limit);
```

```
        }else{
             init<<<int(limit/threads)+1, threads>>>(p_array,
sqroot, limit);
        }


              cudaMemcpy(arr, p_array, limit * sizeof(int),
cudaMemcpyDeviceToHost);


        //->
        ms += stop_timer();
      }


      if (argc==2){
              printf("times %i - avg time = %.5lf ms, %i
threads\n",N,(ms / N), threads);
      }









          printprimes(limit, arr);




      free(arr);


  cudaFree(p_array);
```

```
    return 0;
}
```