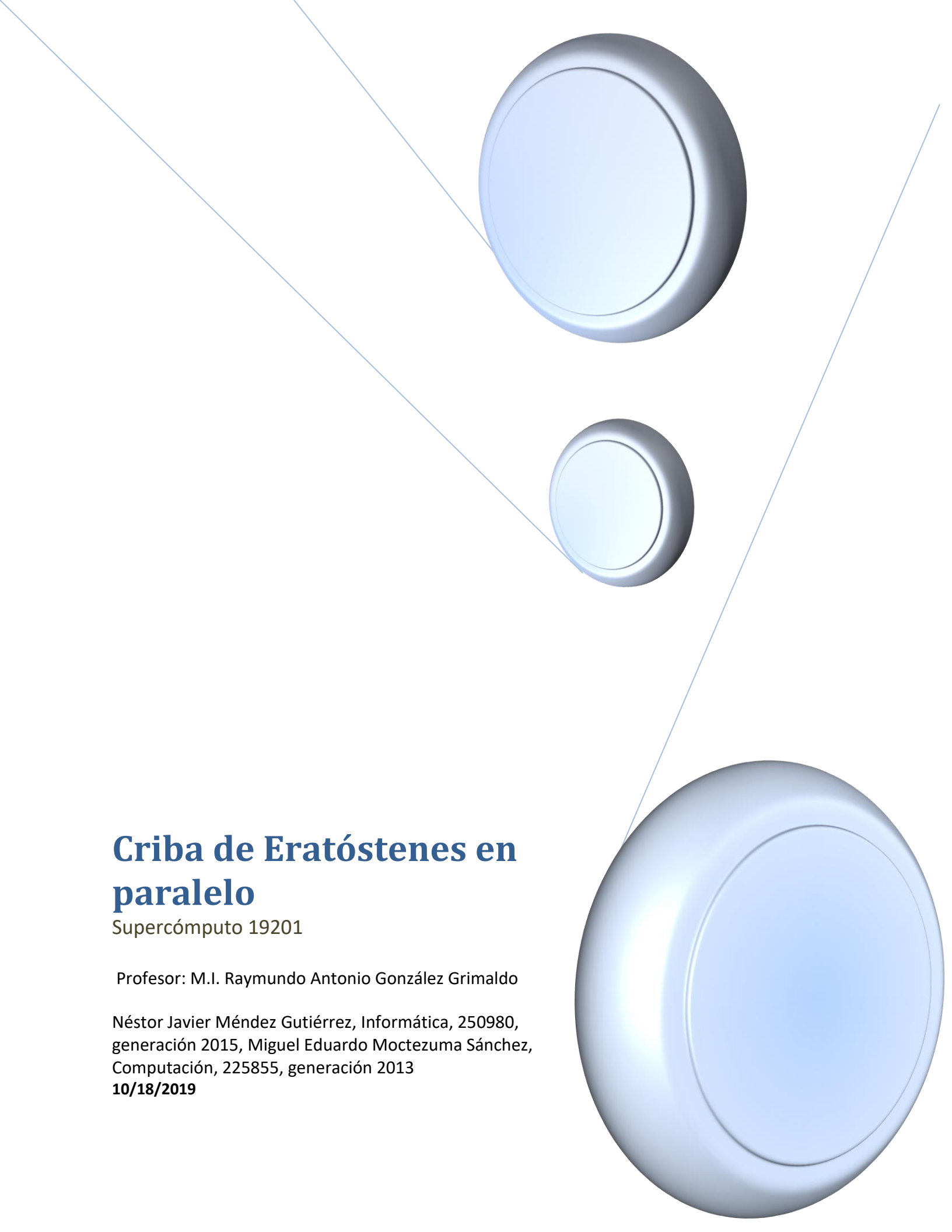

Criba de Eratóstenes en paralelo

Supercómputo 19201

Profesor: M.I. Raymundo Antonio González Grimaldo

Néstor Javier Méndez Gutiérrez, Informática, 250980, generación 2015, Miguel Eduardo Moctezuma Sánchez, Computación, 225855, generación 2013

10/18/2019

Resumen.

En esta práctica se implementó la creación de la Criba de Eratóstenes que consiste en obtener todos los números primos de desde el 2 hasta un numero dado **n** de tal manera se dará la lista de números y cuantos fueron calculados por cada proceso en forma paralela, y al final se imprimen todas en un arreglo.

LA CRIBA DE ERATÓSTENES

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Imagen I "Ejemplo de uso de la Criba de Eratóstenes"

Contenido.

Para poder paralelizar el programa primero nos tuvimos que documentar acerca del algoritmo de la Criba de Eratóstenes y como poder implementarlo de manera secuencial, una vez echo esto comenzamos a generar ideas de cómo y que parte del algoritmo era posible paralelizar, nos dimos cuenta que la parte que podríamos paralelizar seria la de recorrer el arreglo de números, ya que esto se realiza con ciclos y nos hemos percatado que la parte mas evidente de los algoritmos que es posible paralelizar son los ciclos, para esto lo que planteamos fue lo siguiente:

- Cada uno de los procesos verificaría una parte del arreglo total de números.
    - Por ejemplo: si el numero limite dado **P** es 1000, y el número de procesos es 4, el proceso 0 verifica de 2 a 250, el proceso 1 verifica de 251 a 500 y así sucesivamente, para evitar enviar a cada uno de los procesos el rango que le toca procesar, simplemente la variable que contiene el número limite dado, la conocen todos los procesos y de esta forma cada proceso calcula que parcialidad tiene que procesar.
- Procedimiento que ha de ejecutar cada uno de los procesos.
    - Ya dentro de cada proceso la tarea fue verificar los números primos existentes en ese segmento de números, para esto, aquí es donde adaptamos el algoritmo de Criba de Eratóstenes, donde verificamos con los primeros números primos que sean menores o iguales a la raíz del límite dado **P**.
- Una vez calculados los números primos de cada segmento.
    - Después de haber calculado los números primos existentes en cada segmento los reunimos en un arreglo el cual posteriormente enviamos al proceso raíz.
- Envió de la información.
    - Usamos la función **MPI_Gather** para enviar el número de números primos calculados en cada proceso, esto para poder generar los arreglos “recvcouts”, “displs” y “recvbuffer”, los dos primeros los usa la función **MPI_Gatherv** para verificar cuantos elementos está recibiendo y en que posición del “buffer” los ha de posicionar y el tercer arreglo “buffer” es usado por la función **MPI_Gatherv** para depositar en orden cada una de las partes generadas por los procesos.
- Mostrar el resultado.
    - Una vez que el proceso raíz tiene el arreglo “buffer” con los datos generados por el resto de los procesos, solo lo imprimimos.

Imagen.

En la imagen de abajo se puede observar el algoritmo en C que usamos para el cálculo de los números primos.

```c
void CalculaPrimos(int *arrAux, int desde, int hasta, int nDatos, int idProc, int n)
{
    int despl = 1, i, j;
    for(i = 2; i*i <= n; i++)
    {
        for (j = desde; j < hasta; j+=despl)
        {
            if(j%i == 0 && j == i)  //en el caso de que esta condición sea cierta significa que i y j tienen el mismo valor
            {                       //entonces la variable despl se iguala al numero sospechoso de ser un primo, de esta forma
                despl = i;          //el desplazamiento dentro del arreglo nos permitira trabajar solo con los multiplos de ese numero
            }
            else if(j%i == 0 && j != i) //En el caso en el cual el residuo es cero y j e i son diferentes sognifica que es un multiplo del
            {                           //sospechoso y por tanto no es un numero primo, por lo que proseguimos a marcarlo en cero, lo cual
                arrAux[j-((nDatos*idProc)+1)] = 0;  //dentro del arreglo indica que no es un numero primo.
            }
        }
        despl = 1;
    }
}
```

Conclusiones.

En esta práctica logramos fortalecer la programación paralela en un ejercicio llamado “Criba de Eratóstenes” el cual es una buena manera de practicar la programación paralela, se nos dijo que primero lo hiciéramos en forma lineal y así darnos una idea de cómo trabajar el algoritmo, después pasamos a la forma paralela, el cual fue algo confuso al comienzo pero de igual manera se logró obtener, en nuestro punto de vista nos parece una excelente forma de seguir practicando el MPI con las diferentes formas de implementar en este problema.

Bibliografías

https://app.schoology.com/course/2179894018/materials/gp/2179894024.

http://informatica.uv.es/iiguia/ALP/materiales2005/2_2_introMPI.htm

https://es.wikipedia.org/wiki/Criba_de_Erat%C3%B3stenes