Práctica 3: Cálculo del número de primos hasta un número M

Autor: Antonio Felipe Guzmán Martín

Mi práctica se basa en el cálculo de número de primos desde 2 que es el primer primo hasta un número M que se determina con la variable "n_hi".

Para ver como se van calculando los primos hay que descomentar la línea siguiente en prime.c y primeV2.c.

```
 printf ( "  %8d         \t%8d         \t%14f\n", n, primes,
wtime );
```

Mi programa se basa en un bucle que itera hasta calcular el número de primos que contiene un número M tope fijado por el programa.

Iterará calculando el número de primos que haya hasta N_i (siendo este una potencia de dos).

Si por ejemplo queremos saber el número de primos que hay hasta el número 18.

Mi programa toma como tope el número M=18.
Las veces que se calculará el número de primos depende de las potencias de 2 que hayan hasta 18.
En este caso número de potencias de 2 hasta 18 es : i=2,4,8,16. (los llamaremos N_i)
En la última iteración el bucle calculará el número de primos hasta 16 y se parará.

La primera columna el número representa N_i y la segunda el número de primos en hasta cada uno de los N_i:

```
[cvi066049:p3 antonioguzman$ mpirun -np 11 --hostfile myhostfile ./primeV2 18
         2                       1                       0.001196
         4                       2                       0.000026
         8                       4                       0.000004
        16                       6                       0.000002
```

El cálculo se reparte de la siguiente manera:

Se reparte el trabajo entre los procesos:
ierr = MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

y se manda a ejecutar al proceso id la carga de trabajo que corresponda:
primes_part = prime_number (n, id, p);

Se reparte de la siguiente manera:
El proceso id comienza en su id+2 y llega hasta N_i avanzando en i+p. Siendo p el número de procesos que hemos determinado en mpirun.

Para –np=2 el proceso 0 ejecuta: 2,4,6,8,10,12,14,16
Para –np=2 el proceso 1 ejecuta: 3,5,7,9,11,13,15

Se observa que para el caso de que el número de procesos sea par no se repartirá uniformemente la carga. Alguno/s de los seleccionados ejecuta más trabajo que los demás.
Esto afecta al tiempo de ejecución para números "M" tope altos.

Finalmente se suman los resultados parciales del total de números de primos que cada procesos a obtenido:

```
ierr = MPI_Reduce ( &primes_part, &primes, 1, MPI_INT, MPI_SUM, 0,
        MPI_COMM_WORLD );
```

El **tamaño del problema es para todos los casos M=400000.** Si bien en los scripts suministrados (acapv1.sh, acapv2.sh y local.sh) se puede cambiar este número por otro y probar con diferentes tamaños

Version mejorada VS normal

En la gráfica los datos pertenecientes a la línea violeta pertenecen a la versión normal y la línea verde a la versión mejorada.

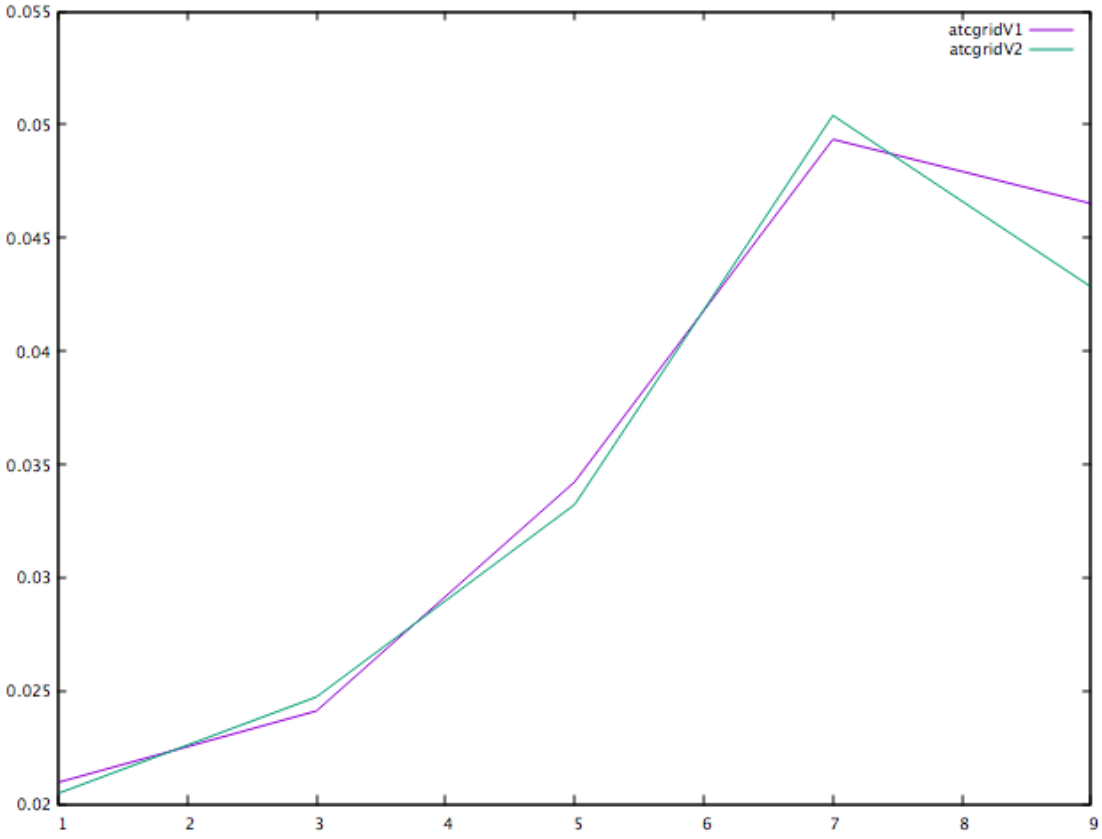
Se puede observar fácilmente que la versión mejorada tiene tiempos inferiores para cualquier número de procesos que se indique.

La mejora consiste en guardar el último número N_i para el que se han calculado el número de primos inferiores. Así la próxima vez que se ejecute el cálculo de primos solo se calculen: N_i +1 hasta N_i ^2. También se guardan en una variable "primes_ant" el número de primos hasta ese número N_i.

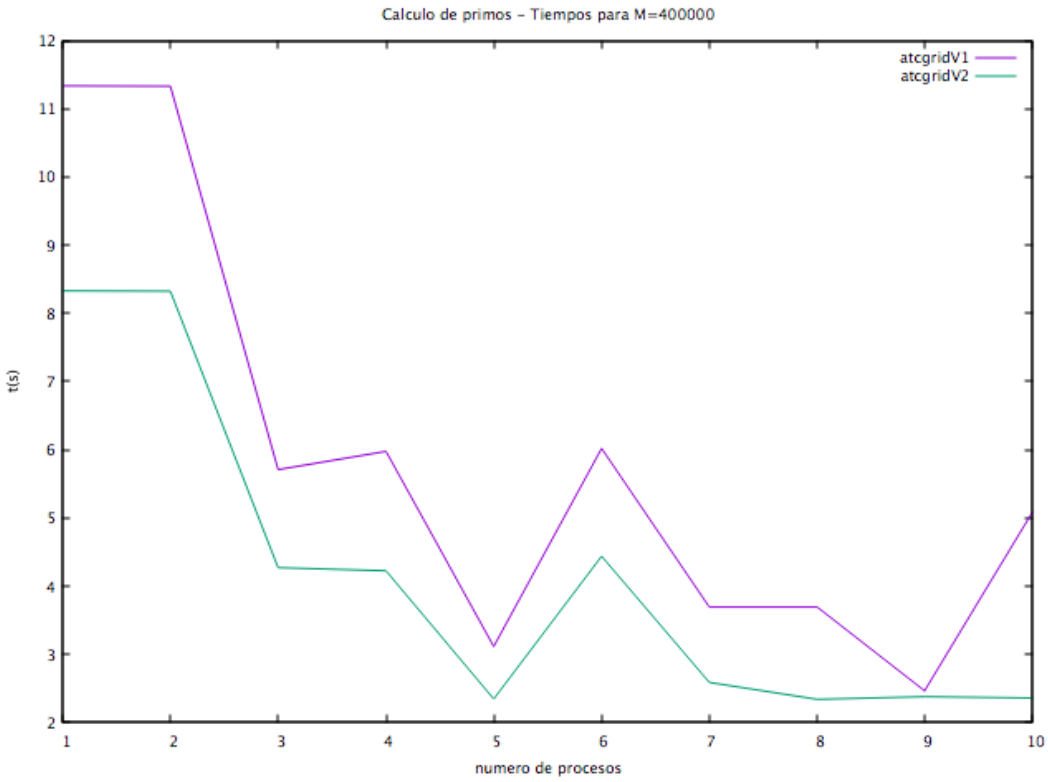
Para recoger los tiempos de creación y destrucción de procesos he sumado el tiempo de MPI_Init() y MPI_Finalize().

Tiempos de creación y destrucción de procesos

Como es de esperar el tiempo sin mejora y con mejora es el mismo porque se generan y destruyen el mismo número de procesos por lo que los tiempos son prácticamente iguales. En la gráfica vemos que a más procesos se generan más comunicaciones para inicializarlos y finalizarlos aumentando por tanto el tiempo.



Tiempos V1 vs V2



En la imagen de arriba observamos la gráfica número de procesos - tiempo (s).
En tiempo la versión 2 mejora a la versión1 si bien a partir de 7 procesos disminuye mucho la mejora. Esto se produce porque a más número de procesos el num_ant (desde el que comenzamos el bucle) a más número de procesos estén ejecutando más lejos del último proceso que entre a ejecutar estará.
Todos los procesos usan el mismo número de partida. Por ejemplo si el último número fue el 256 y tenemos 10 procesos, los 10 tendrán como último primo calculado hasta el 256.

Gráfica de la ganancia V1 vs V2

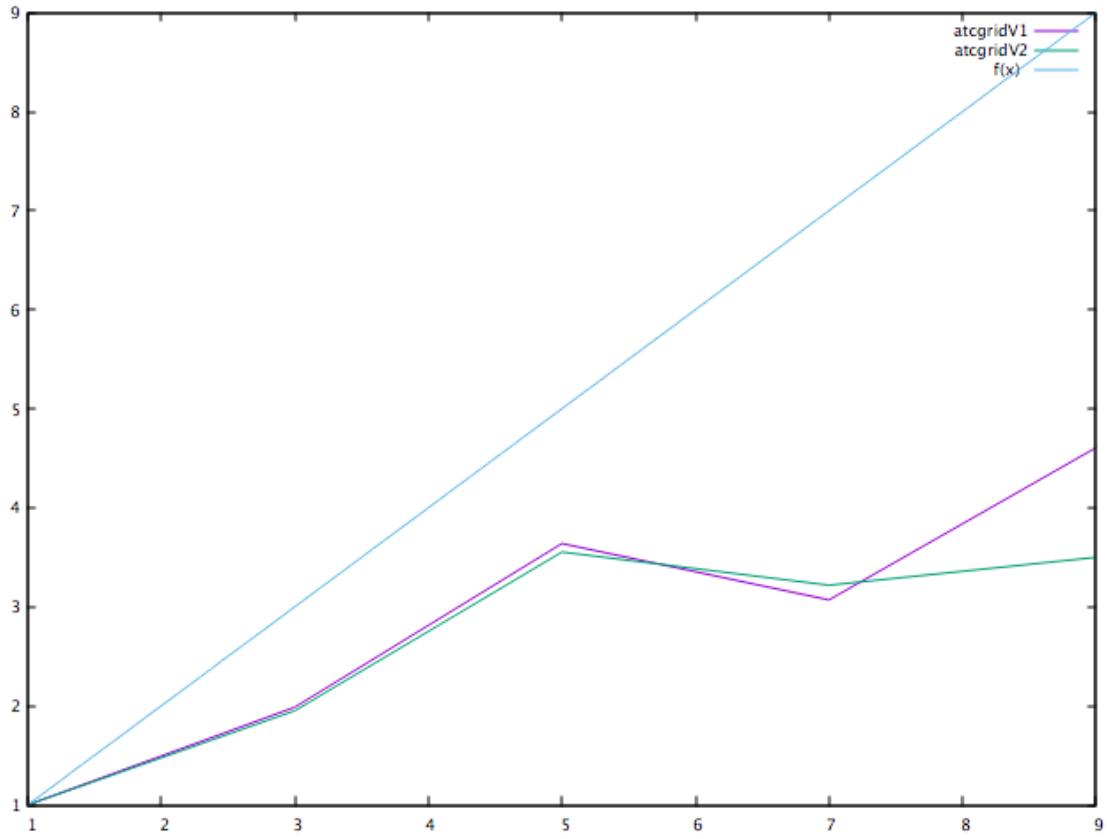
Ganancia 1 se refiere a la ganancia de las ejecuciones con múltiples procesos para la versión 1 con un proceso.

Ganancia 2 se refiere a la ganancia de las ejecuciones con múltiples procesos para la versión 2 con un proceso.

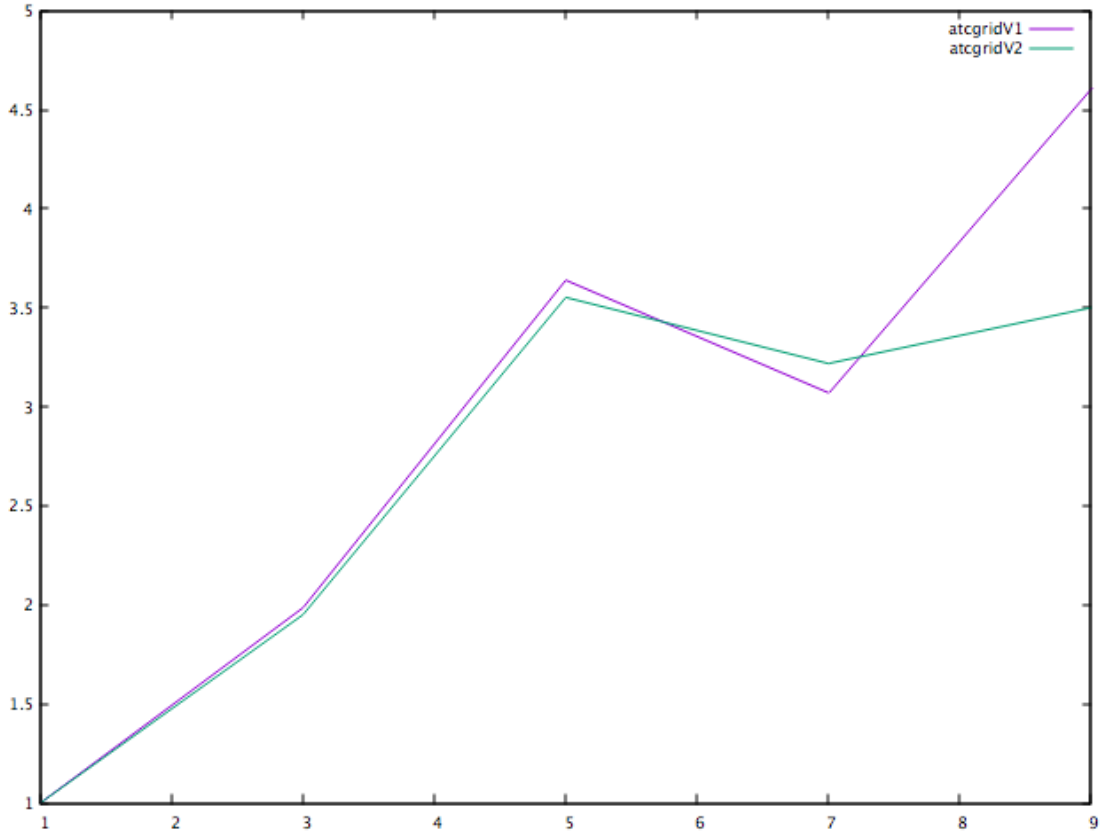
Procesos	Versión 1	Versión 2	Ganancia1	Ganancia2
1	11,336467	8,329013	1	1
2	11,333167	8,325763	1,000291181	1,000390355
3	5,708543	4,27052	1,985877482	1,950351011
4	5,975717	4,22536	1,897089002	1,971196064
5	3,115776	2,344846	3,638408859	3,552051179
6	6,021126	4,439431	1,882781892	1,876144263
7	3,692424	2,588154	3,070196435	3,218128829
8	3,692736	2,340791	3,069937033	3,55820447
9	2,460576	2,379868	4,60724115	3,4997794
10	5,082514	2,359473	2,230484166	3,53003107

Nota: En la siguiente gráfica no se han incluido las ganancias para número de procesos par porque meten ruido para observar la evolución de la ganancia. No escala bien para procesos pares.

Con la siguiente gráfica vemos que las ganancias no son las ideales por lo que el código no está bien paralelizado (no se ajusta a la función lineal).
También hemos de considerar la sobrecarga de las comunicaciones entre los procesos. Si las comunicaciones fueran mucho más pequeñas que el del cálculo que se hace, probablemente conseguiríamos una ganancia lineal.



Por lo general la ganancia entre la versión mejorada y la normal es muy similar. A partir de nueve procesos se dispara a favor de la versión normal. Lo que terminará por llegar al punto en el que la normal alcance en tiempos a la mejorada



Conclusiones:

Según lo comentado anteriormente la ganancia disminuye bastante en el caso de que el número de procesos sea par respecto a un número impar. Esto ocurre porque no todos los procesos comprueban el mismo número de números y tampoco son del mismo tamaño. Si omitimos los pares queda así.