

Tarea 2 - Métodos numéricos

Erick Salvador Alvarez Valencia

Centro de Investigación en Matemáticas, A.C.,
`erick.alvarez@cimat.mx`

Resumen El presente es un reporte de la tarea no. 2 de la materia de métodos numéricos. En el cual se describe un programa que trabaja con la API de OpenMP para realizar acciones en cómputo paralelo, de entre las cuales se trabaja con vectores y matrices dinámicas.

A lo largo de este documento se describirá el modo de compilar y ejecutar el proyecto hecho en C con Codeblocks, así como una descripción del mismo y resultados de ejecuciones realizadas.

Keywords: Computo paralelo, OpenMP, Métodos numéricos.

1. Descripción del programa

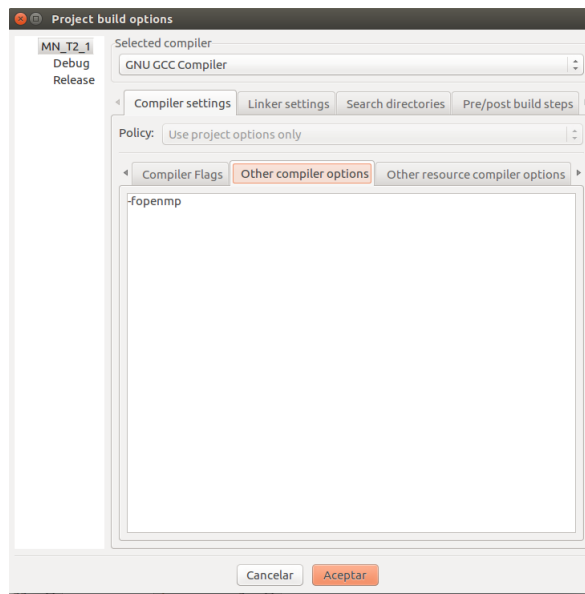
El presente programa se encarga de realizar algunas funciones que trabajan con vectores y matrices de forma paralela, esto es, usando OpenMP. A continuación se describe de manera rápida lo que realiza cada función:

1. Función 1: Esta función se encarga de realizar la suma de vectores, la salida la va colocando en otro vector que de igual manera se pasa su referencia como argumento.
2. Función 2: Esta función se encarga de realizar el producto vectores, la salida la va colocando en otro vector que de igual manera se pasa su referencia como argumento.
3. Función 3: Esta función se encarga de realizar el producto punto, la salida retorna como un tipo *double*.
4. Función 4: Esta función se encarga de realizar el producto Matriz - vector, la salida la va colocando en otro vector que de igual manera se pasa su referencia como argumento.
5. Función 5: Esta función se encarga de realizar el producto de dos matrices, la salida la va colocando en otra matriz que de igual manera se pasa su referencia como argumento.

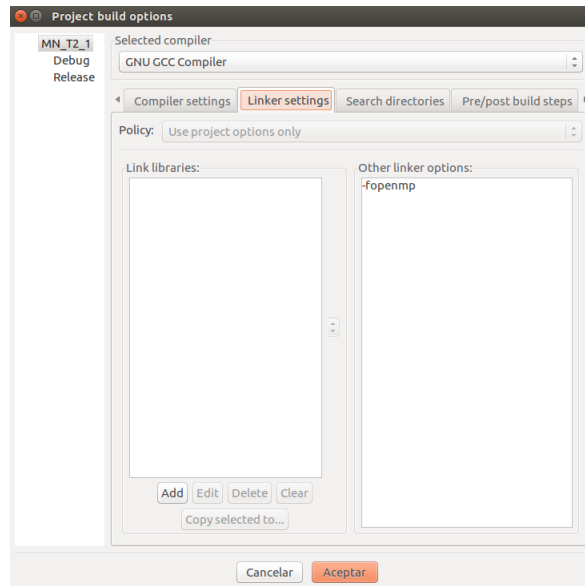
En lo que se enfoca este programa es en calcular los tiempos de ejecución de las funciones anteriores usando un diferente número de threads. Por lo cual calcula el tiempo que toma ejecutarse cada una y lo imprime en consola. Todo esto puede variar de acuerdo al tamaño, número de hilos y semilla que pase el usuario como argumentos.

2. Compilación y ejecución del proyecto

Como se mencionó anteriormente, este proyecto fue elaborado en Codeblocks, por lo que se requiere tener este software instalado, así como la API de OpenMP de acuerdo al Sistema Operativo que se esté corriendo. De igual manera previo a ejecutar el programa es necesario asegurar que se tenga activa la bandera de **-fopenmp** la cual podemos establecer en **Project** → **Build options** → **Compiler settings** → **Other compiler options**. Así como en **Linker settings** → **Other linker options**. Tal como se muestra en las Figuras 1 y 2.



(a) Figura 1. Banderas de OpenMP en Codeblocks (Compiler settings).



(b) Figura 2. Banderas de OpenMP en Codeblocks (Linker settings).

De esta manera nos aseguramos que el software esté compilando de manera correcta.

2.1. Argumentos en línea de comandos

Para ejecutar el programa es necesario proveer argumentos por línea de comandos, para hacer esto en Codeblocks recurrimos a **Project** → **Set program's arguments**. A continuación se presenta la lista de argumentos

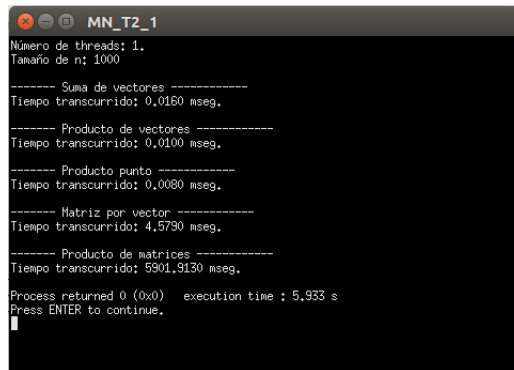
que recibe el programa:

1. Argumento 1 (N) - Tipo entero: El tamaño de los vectores y las matrices.
2. Argumento 2 (T) - Tipo entero: El número de hilos con los que trabajará OpenMP. Si se pasa un número menor que cero, el programa colocará sólo un hilo en ejecución, y si se pasa un número mayor que la cantidad de cores que tiene la máquina (definido por la función *omp_get_max_threads()*) este lo cambiará por la máxima cantidad de cores.
3. Argumento 3 (S) - Tipo entero: (Opcional) La semilla con la que se inicializarán los números aleatorios.

Notas: Es necesario respetar el orden en el que se proponen los argumentos descritos, así como su tipo. En caso de no ingresar el tercer argumento (la semilla), este se elegirá con una función de tiempo.

3. Ejemplo de ejecución del programa

A continuación se mostrarán ejemplos de ejecución del programa configurando el número de cores de 1 a 4 y un tamaño de 1000 elementos para los vectores y las matrices. Cabe destacar que no se pasó ningún valor hacia la semilla para llenar los vectores y las matrices de manera diferente en cada ejecución.



```

MN_T2_1
Número de threads: 1.
Tamaño de n: 1000

----- Suma de vectores -----
Tiempo transcurrido: 0,0160 mseg.

----- Producto de vectores -----
Tiempo transcurrido: 0,0100 mseg.

----- Producto punto -----
Tiempo transcurrido: 0,0080 mseg.

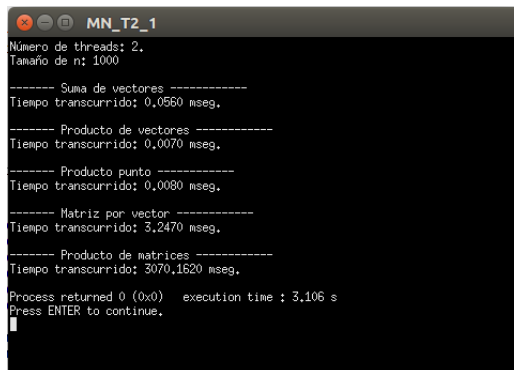
----- Matriz por vector -----
Tiempo transcurrido: 4,5790 mseg.

----- Producto de matrices -----
Tiempo transcurrido: 5901,9130 mseg.

Process returned 0 (0x0)   execution time : 5,933 s
Press ENTER to continue.

```

(c) Figura 3. Ejecución del programa usando 1 core.



```

MN_T2_1
Número de threads: 2.
Tamaño de n: 1000

----- Suma de vectores -----
Tiempo transcurrido: 0,0560 mseg.

----- Producto de vectores -----
Tiempo transcurrido: 0,0070 mseg.

----- Producto punto -----
Tiempo transcurrido: 0,0080 mseg.

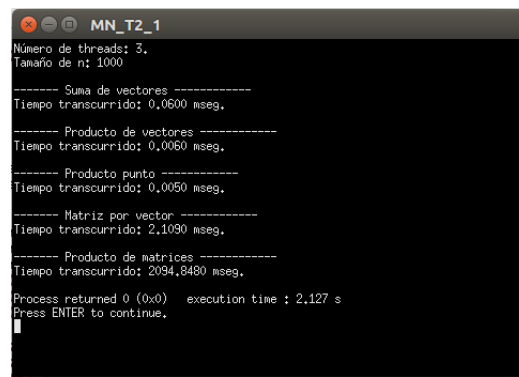
----- Matriz por vector -----
Tiempo transcurrido: 3,2470 mseg.

----- Producto de matrices -----
Tiempo transcurrido: 3070,1620 mseg.

Process returned 0 (0x0)   execution time : 3,106 s
Press ENTER to continue.

```

(d) Figura 4. Ejecución del programa usando 2 cores.



```

MN_T2_1
Número de threads: 3.
Tamaño de n: 1000

----- Suma de vectores -----
Tiempo transcurrido: 0,0600 mseg.

----- Producto de vectores -----
Tiempo transcurrido: 0,0080 mseg.

----- Producto punto -----
Tiempo transcurrido: 0,0050 mseg.

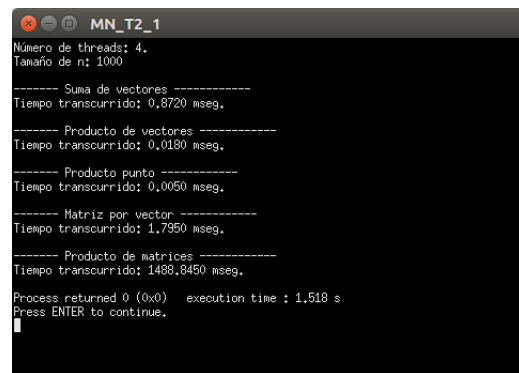
----- Matriz por vector -----
Tiempo transcurrido: 2,1090 mseg.

----- Producto de matrices -----
Tiempo transcurrido: 2094,8480 mseg.

Process returned 0 (0x0)   execution time : 2,127 s
Press ENTER to continue.

```

(e) Figura 5. Ejecución del programa usando 3 cores.



```

MN_T2_1
Número de threads: 4.
Tamaño de n: 1000

----- Suma de vectores -----
Tiempo transcurrido: 0,0720 mseg.

----- Producto de vectores -----
Tiempo transcurrido: 0,0180 mseg.

----- Producto punto -----
Tiempo transcurrido: 0,0050 mseg.

----- Matriz por vector -----
Tiempo transcurrido: 1,7350 mseg.

----- Producto de matrices -----
Tiempo transcurrido: 1488,8450 mseg.

Process returned 0 (0x0)   execution time : 1,518 s
Press ENTER to continue.

```

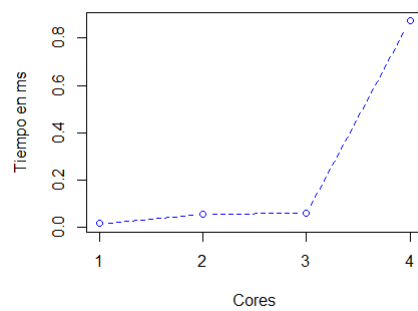
(f) Figura 6. Ejecución del programa usando 4 cores.

Notas: El programa fue ejecutado en una computadora con un procesador Intel Core i7 con ocho núcleos. Y pese a que el procesador tiene el sistema Hyper threading, ya no se reduce sustancialmente el tiempo de ejecución con un número de cores mayor a cuatro.

3.1. Tiempos de ejecución

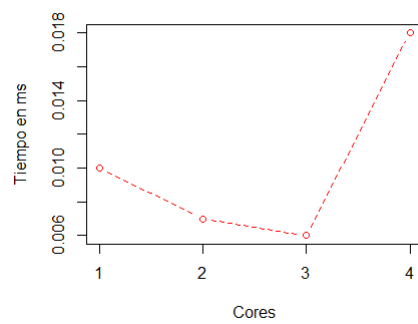
A continuación se muestran las gráficas generadas en base a los tiempos de ejecución del programa con mil datos:

Ejecución de la función 1



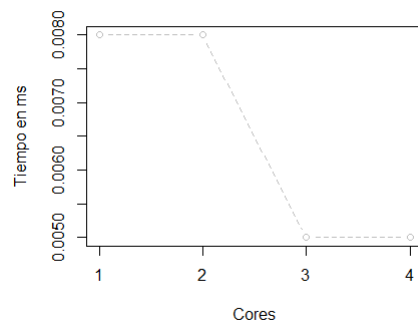
(g) Figura 7.

Ejecución de la función 2



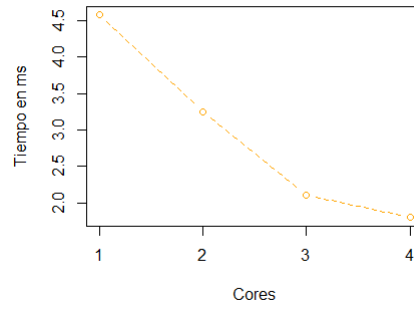
(h) Figura 8.

Ejecución de la función 3



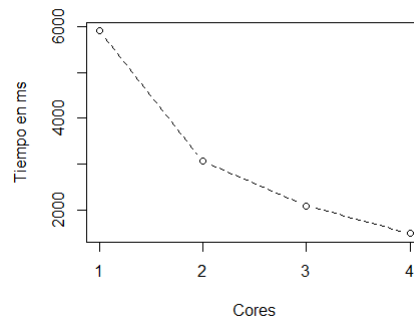
(i) Figura 9.

Ejecución de la función 4



(j) Figura 10.

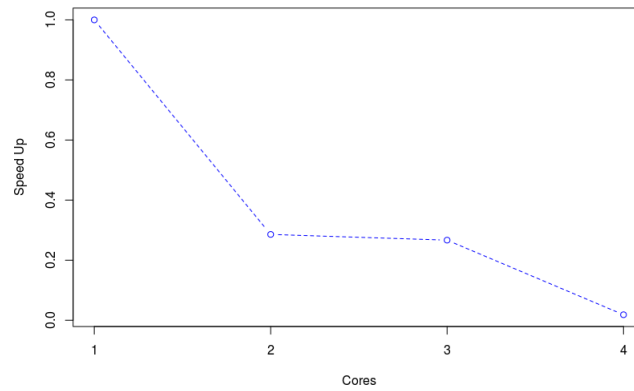
Ejecución de la función 5



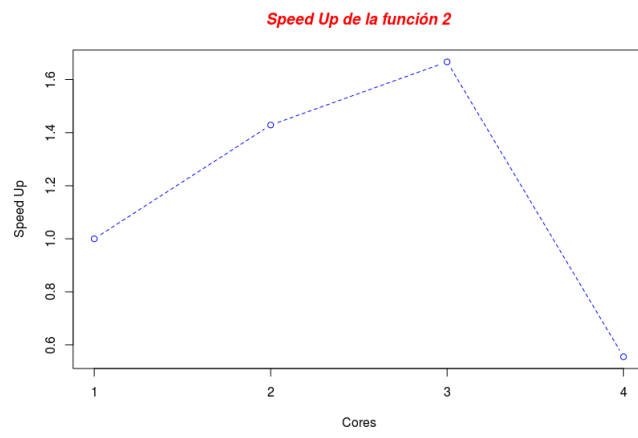
(k) Figura 11.

De la misma forma se mostrarán las gráficas relacionadas al "Speed Up" de cada función:

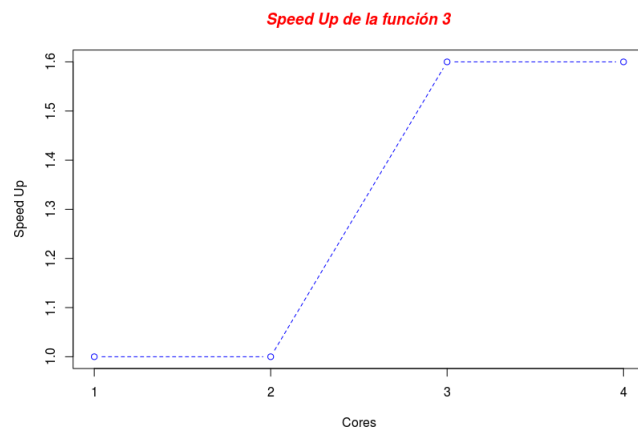
Speed Up de la función 1



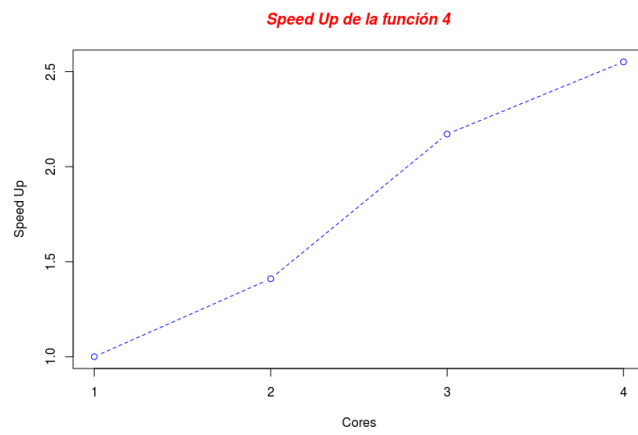
(l) Figura 12.



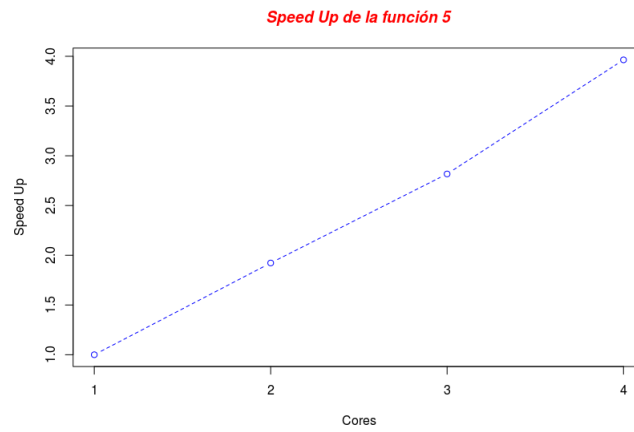
(m) Figura 13.



(n) Figura 14.



(ñ) Figura 15.



(o) Figura 16.

4. Conclusiones

Para concluir podemos argumentar sobre los tiempos de ejecución que generó el programa al ejecutarse con un conjunto de mil datos, en las primeras dos funciones podemos observar que la paralelización resultó contraproducente, esto debido a la relativamente poca cantidad de datos con los que estamos trabajando, para convencernos de lo anterior podemos ver las gráficas de los "Speed Ups" de las funciones, ya que idealmente la gráfica debería ser creciente, lo cual indica mejora en la paralelización al ir aumentando el número de cores. Hay que recordar que la paralelización tiene un costo computacional, esto debido a la creación de los threads y al almacenamiento de la memoria en la cache, cosa que no ocurre en el procesamiento en serie. Por lo tanto es más conveniente realizar el paralelismo con cantidades de datos muy grandes.