Tarea 08 - Paralelización

Isaac Rodríguez Bribiesca

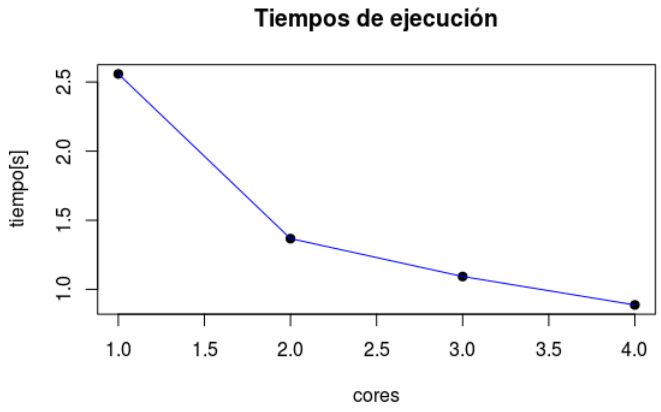
Resumen Para esta tarea se paralelizaron las siguientes operaciones: la suma de dos vectores, la multiplicación de vectores elemento a elemento, el producto punto, la mutiplicación matriz vector y la mutiplicación matriz matriz.

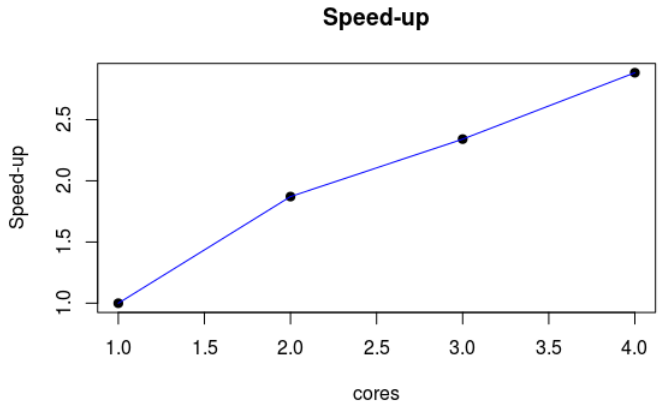
1. Suma de vectores

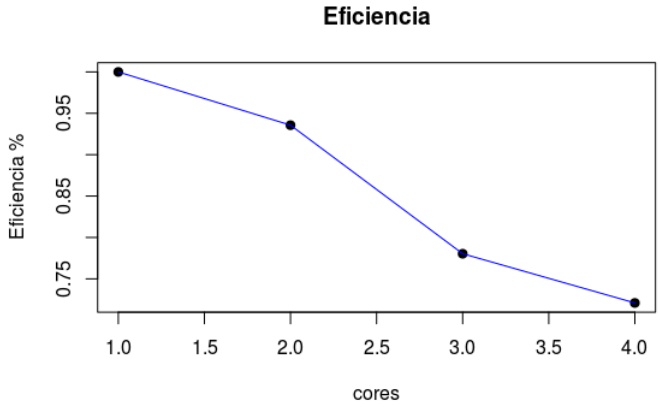
1.1. Prueba Ejecución

```
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP export OMP_NUM_THREADS=1
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP gcc -fopenmp sum_vectors.c -o run && ./run

Suma de vectores
---------------

Se ejecuta con 1 hilo(s)

Duración de la operación: 2.558507 segundos con vectores de tamaño 100000000

isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP export OMP_NUM_THREADS=2
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP gcc -fopenmp sum_vectors.c -o run && ./run

Suma de vectores
---------------

Se ejecuta con 2 hilo(s)

Duración de la operación: 1.367222 segundos con vectores de tamaño 100000000

isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP export OMP_NUM_THREADS=3
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP gcc -fopenmp sum_vectors.c -o run && ./run

Suma de vectores
---------------

Se ejecuta con 3 hilo(s)

Duración de la operación: 1.092944 segundos con vectores de tamaño 100000000

isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP export OMP_NUM_THREADS=4
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP gcc -fopenmp sum_vectors.c -o run && ./run

Suma de vectores
---------------

Se ejecuta con 4 hilo(s)

Duración de la operación: 0.887320 segundos con vectores de tamaño 100000000
```

1.2. Gráficas




Speed-up
Speed-up
1.0
1.5
2.0
2.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

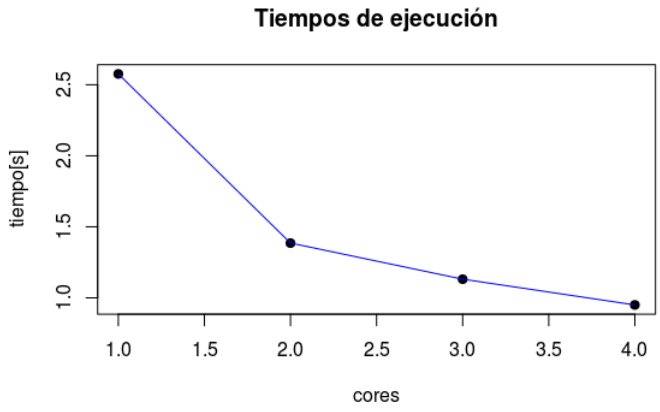

Eficiencia
Eficiencia %
0.75
0.85
0.95
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

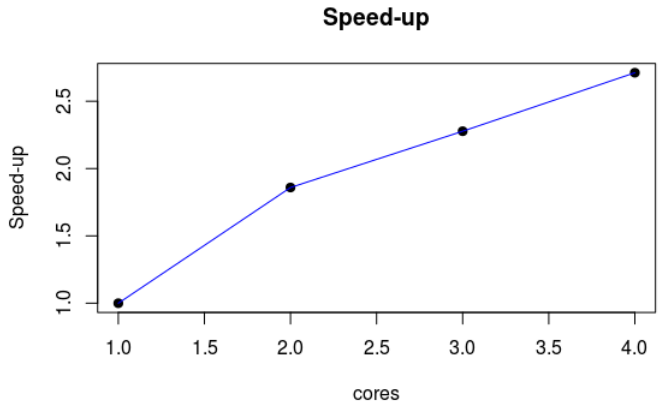
2. Multiplicación vectores elemento a elemento

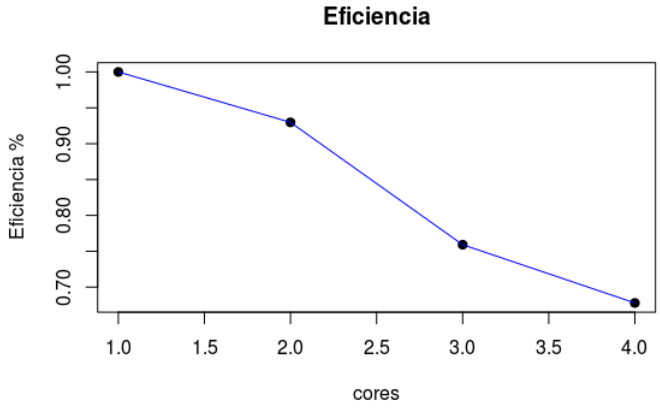
2.1. Prueba Ejecución

```
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/multiplica vectores gcc -fopenmp multiply_vectors.c -o run && ./run 1

Multiplicación de vectores elemento a elemento
--------------------

Se ejecuta con 1 hilo(s)

Duración de la operación: 2.575933 segundos con vectores de tamaño 100000000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/multiplica vectores gcc -fopenmp multiply_vectors.c -o run && ./run 2

Multiplicación de vectores elemento a elemento
--------------------

Se ejecuta con 2 hilo(s)

Duración de la operación: 1.385368 segundos con vectores de tamaño 100000000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/multiplica vectores gcc -fopenmp multiply_vectors.c -o run && ./run 3

Multiplicación de vectores elemento a elemento
--------------------

Se ejecuta con 3 hilo(s)

Duración de la operación: 1.131123 segundos con vectores de tamaño 100000000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/multiplica vectores gcc -fopenmp multiply_vectors.c -o run && ./run 4

Multiplicación de vectores elemento a elemento
--------------------

Se ejecuta con 4 hilo(s)

Duración de la operación: 0.949925 segundos con vectores de tamaño 100000000
```

2.2. Gráficas




Speed-up
Speed-up
1.0
1.5
2.0
2.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

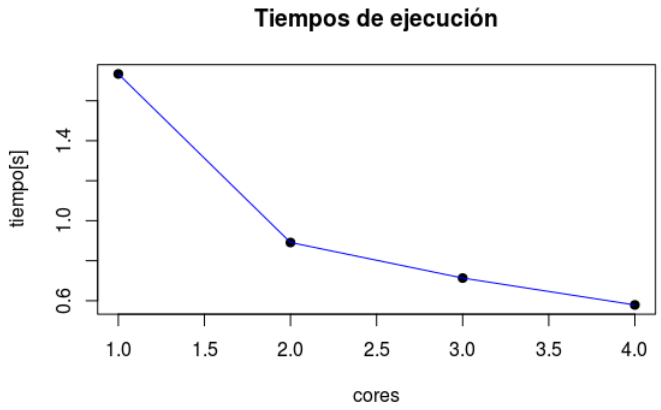

Eficiencia
Eficiencia %
0.70
0.80
0.90
1.00
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

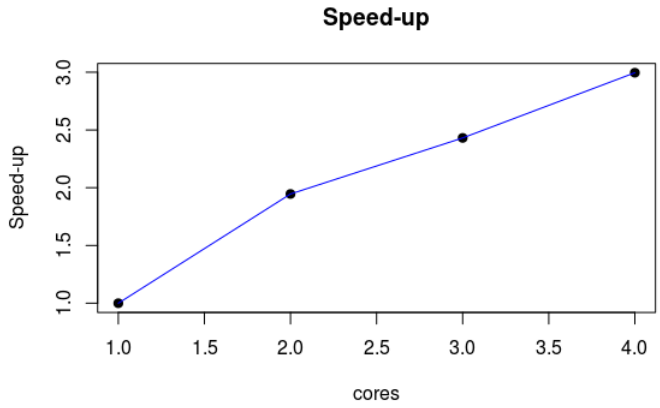
3. Producto punto

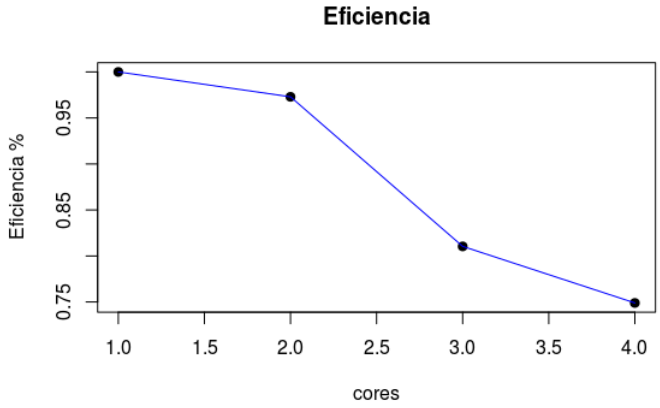
3.1. Prueba Ejecución

```
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/producto punto gcc -fopenmp dot_product.c -o run && ./run 1
Producto punto de vectores
--------------------
Se ejecuta con 1 hilo(s)
Duración de la operación: 1.733983 segundos con vectores de tamaño 100000000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/producto punto gcc -fopenmp dot_product.c -o run && ./run 2
Producto punto de vectores
--------------------
Se ejecuta con 2 hilo(s)
Duración de la operación: 0.891083 segundos con vectores de tamaño 100000000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/producto punto gcc -fopenmp dot_product.c -o run && ./run 3
Producto punto de vectores
--------------------
Se ejecuta con 3 hilo(s)
Duración de la operación: 0.713234 segundos con vectores de tamaño 100000000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/producto punto gcc -fopenmp dot_product.c -o run && ./run 4
Producto punto de vectores
--------------------
Se ejecuta con 4 hilo(s)
Duración de la operación: 0.578833 segundos con vectores de tamaño 100000000
```

3.2. Gráficas




Speed-up
Speed-up
1.0
1.5
2.0
2.5
3.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

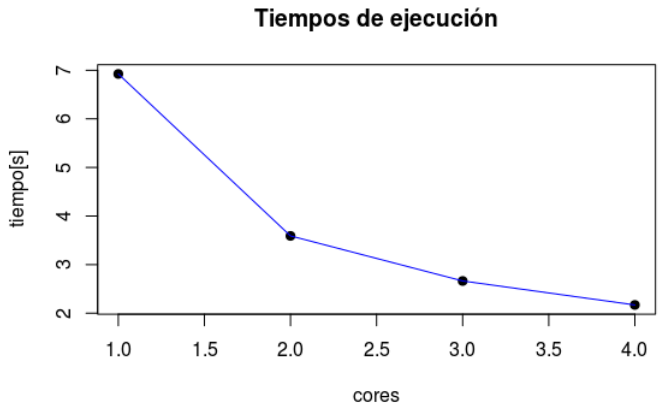

Eficiencia
Eficiencia %
0.75
0.85
0.95
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

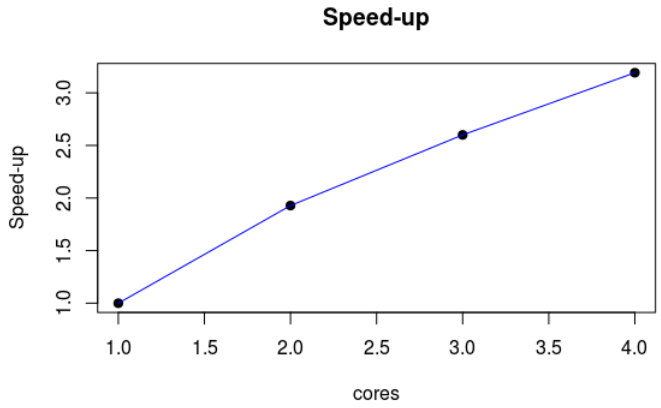
4. Multiplicacion matriz - vector

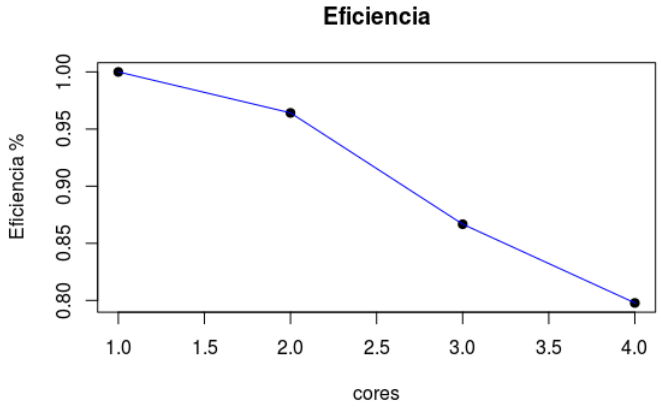
4.1. Prueba Ejecución

```
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult matriz vector gcc -fopenmp multiply_matrix_vector.c -o run && ./run 1

Multiplicación de matriz por vector
-------------------

Se ejecuta con 1 hilo(s)

Duración de la operación: 6.924456 segundos con vectores de tamaño 22000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult matriz vector gcc -fopenmp multiply_matrix_vector.c -o run && ./run 2

Multiplicación de matriz por vector
-------------------

Se ejecuta con 2 hilo(s)

Duración de la operación: 3.590970 segundos con vectores de tamaño 22000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult matriz vector gcc -fopenmp multiply_matrix_vector.c -o run && ./run 3

Multiplicación de matriz por vector
-------------------

Se ejecuta con 3 hilo(s)

Duración de la operación: 2.663149 segundos con vectores de tamaño 22000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult matriz vector gcc -fopenmp multiply_matrix_vector.c -o run && ./run 4

Multiplicación de matriz por vector
-------------------

Se ejecuta con 4 hilo(s)

Duración de la operación: 2.169955 segundos con vectores de tamaño 22000
```

4.2. Gráficas




Speed-up
Speed-up
1.0
1.5
2.0
2.5
3.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

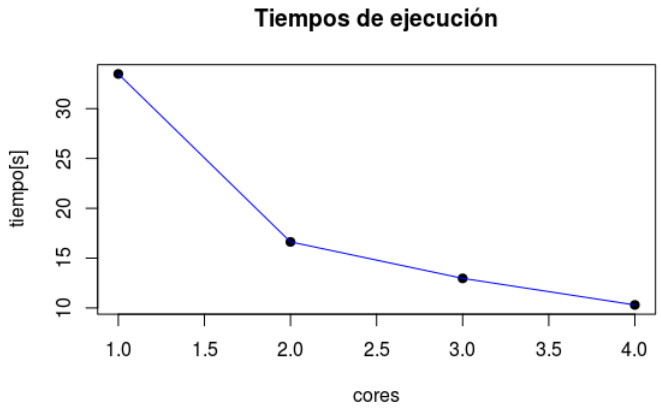

Eficiencia
Eficiencia %
0.80
0.85
0.90
0.95
1.00
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cores

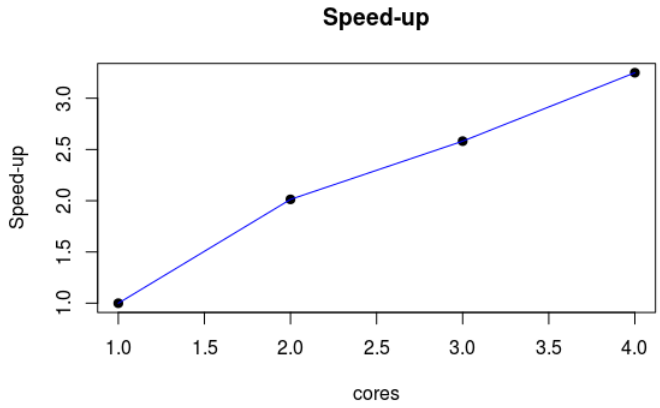
5. Multiplicacion matriz - matriz

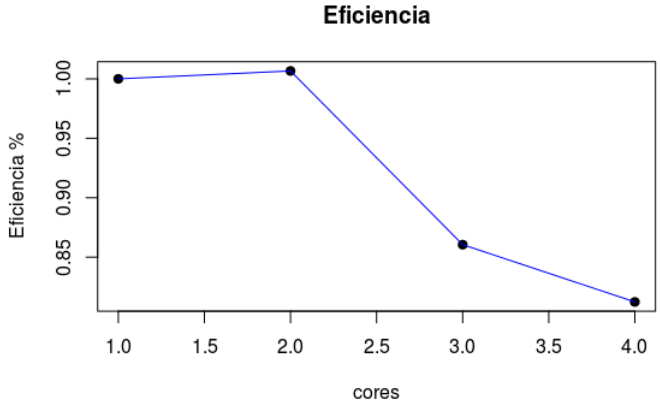
5.1. Prueba Ejecución

```
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult_matriz_matriz gcc -fopenmp multiply_matrix_matrix.c -o run && ./run 1
Multiplicación de matriz por matriz
--------------------
Se ejecuta con 1 hilo(s)
Duración de la operación: 33.478608 segundos con vectores de tamaño 1000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult_matriz_matriz gcc -fopenmp multiply_matrix_matrix.c -o run && ./run 2
Multiplicación de matriz por matriz
--------------------
Se ejecuta con 2 hilo(s)
Duración de la operación: 16.628712 segundos con vectores de tamaño 1000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult_matriz_matriz gcc -fopenmp multiply_matrix_matrix.c -o run && ./run 3
Multiplicación de matriz por matriz
--------------------
Se ejecuta con 3 hilo(s)
Duración de la operación: 12.969060 segundos con vectores de tamaño 1000
isaac@irb ~/Documents/CIMAT/Métodos Numéricos/OPENMP/Tarea 8/mult_matriz_matriz gcc -fopenmp multiply_matrix_matrix.c -o run && ./run 4
Multiplicación de matriz por matriz
--------------------
Se ejecuta con 4 hilo(s)
Duración de la operación: 10.303944 segundos con vectores de tamaño 1000
```

5.2. Gráficas







6. Conclusiones

A partir de las gráficas de tiempos de ejecución, en todos los casos se pudo observar que al aumentar el número de cores usados en la paralelización, hubo una disminución en el tiempo de ejecución. Esto considerando que los tamaños de las matrices y vectores fueron relativamente grandes, ya que si se manejan pocas dimensiones las diferencias en tiempos de ejecución no se podrían apreciar lo suficiente.

Observando las gráficas de Speed-up, se concluye que conforme se aumenta el número de cores, no necesariamente se disminuye el tiempo de ejecución en la misma proporción. Al principio, cuando se incrementa de 1 a 2 cores, la reducción de tiempo de ejecución es aproximadamente de la mitad y la gráfica de Speed-up sigue la linea recta, pero al aumentar a 3 y 4 cores, la reducción en tiempo de ejecución ya no es de 1/3 ni de 1/4 respectivamente, y por lo tanto la curva de Speed-up empieza a disminuir su pendiente.

Finalmente, de las gráficas de eficiencia, también observamos que lo que aporta cada core en la reducción de tiempos de ejecución, va disminuyendo, lo cual concuerda con la métrica de Speed-up. Si quisiéramos tener un Speed up lineal, se necesitaría que la eficiencia se mantuviera en el 100 %, de forma que n cores ejecuten la tarea n veces más rápido.

Es importante tener en cuenta estas métricas siempre que se paraleliza un programa ya que esto nos da una idea de qué tan bien lo hemos hecho y de qué podemos esperar de nuestro programa conforme aumentamos el tamaño de la entrada.