Wandering Salesman Problem en MPI

Patricio López Juri

Al ejecutar el programa con:

```
./run.sh
```

Se obtienen los resultados en consola. Se correrán 3 veces cada experimento y se usarán los valores promedio.

Disclamer

La implementación de la solución no es la mejor (es mala). Pues no usa *branch-and-bound*, sino que solo reparte las tareas en un `MPI_Scatter` para luego tomar la mejor de cada proceso.

Baseline

Con MPI usando solo un nodo `N = 1` se obtienen en promedio los siguientes resultados:

```
real 1.528s
user 0.196s
sys  0.208s
```

Experimentos

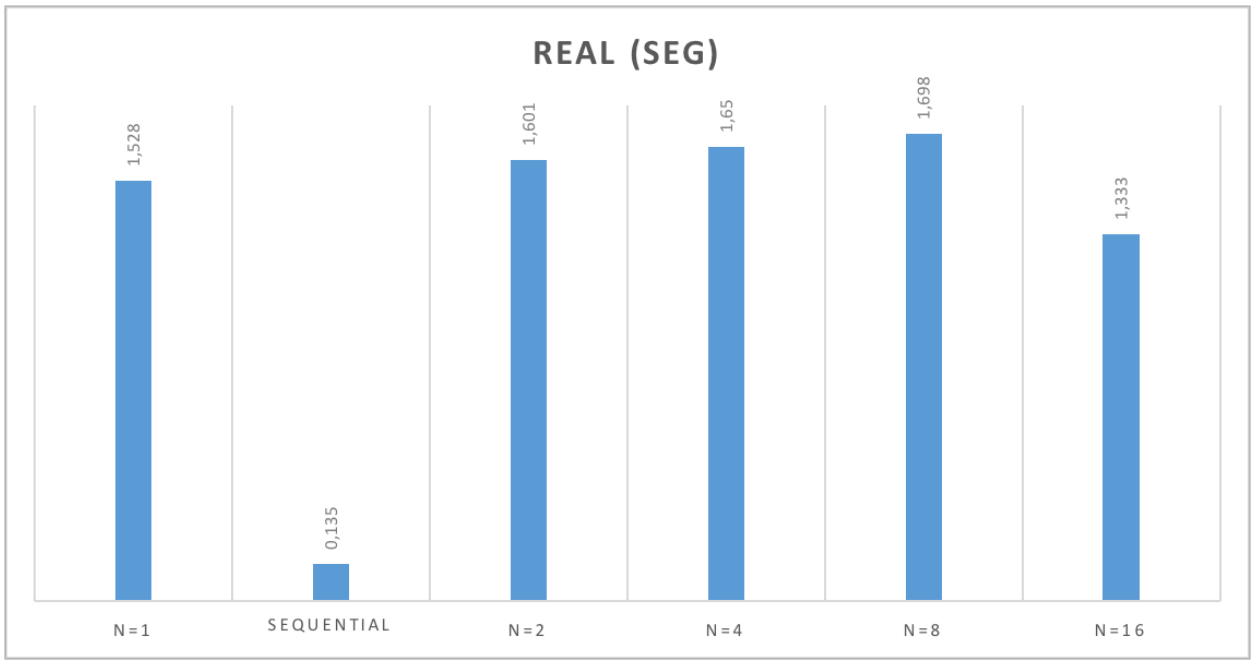
Se usarán los siguientes casos donde `N` es el número de procesos:

- *Secuencial* (sin MPI)
- `N=2`
- `N=4` (ideal)
- `N=8`
- `N=16`

Los resultados promedio son los siguientes:

	N=1	Sequential	N=2	N=4	N=8	N=16
real	**1,528**	**0,135**	**1,601**	**1,65**	**1,698**	**1,333**
user	0,196	0,104	0,248	0,52	1,28	0,028
sys	0,208	0,032	0,24	0,5	0,832	0,048
Speedup	**1,000**	**11,319**	**0,954**	**0,926**	**0,900**	**1,146**
Eficiencia	**1,000**	**11,319**	**0,477**	**0,232**	**0,112**	**0,072**

Gráficamente:



Análisis

A pesar de la mala implementación, se puede notar que realizar la tarea en modo *Secuencial* obtiene los mejores resultados. Esto debe ser porque no existe un overhead de sincronización entre procesos.

El mismo *overhead* se puede notar en los experimentos hasta `N=8`. Algo interesante es el caso de `N=16` donde incluso el entorno MPI en la terminal arroja una alerta de estar usando más recursos de los disponibles:

```
A request was made to bind to that would result in binding more
processes than cpus on a resource:

   Bind to:     NONE:IF-SUPPORTED
   Node:        trauco
```

```
   #processes:  5
   #cpus:       4

You can override this protection by adding the "overload-allowed"
option to your binding directive.
```

En este caso extremo y para posterior estudio: el Speedup es `> 1`.
Sin embargo tiene la peor eficiencia dada la gran cantidad de procesos.