En esta batería de pruebas ejecuto una multiplicación de matrices de tamaño 1000x1000.

La finalidad es comparar la ejecución en serie y en paralelo mediante el calculo del speedUp.

Tras probar diferentes formas de paralelizar los bucles he concluido que lo mas optimo era paralelizar el bucle mas externo colapsando primer bucle interno. Esto me da un speedUp mejor que paralelizando ambos bucles. Supongo que esto es así porque al llegar al segundo bucle con cuatro threads, creaba 4 por cada uno de ellos, teniendo en total 16 thread. Dicho numero de threads podría haber aportado una mejora en un ordenador con mas núcleos pero en mi portátil (de dos núcleos físicos y dos lógicos) la carga de computación asociada a crear 16 threads no valía la pena ya que solo puedo ejecutar 4 a la vez.

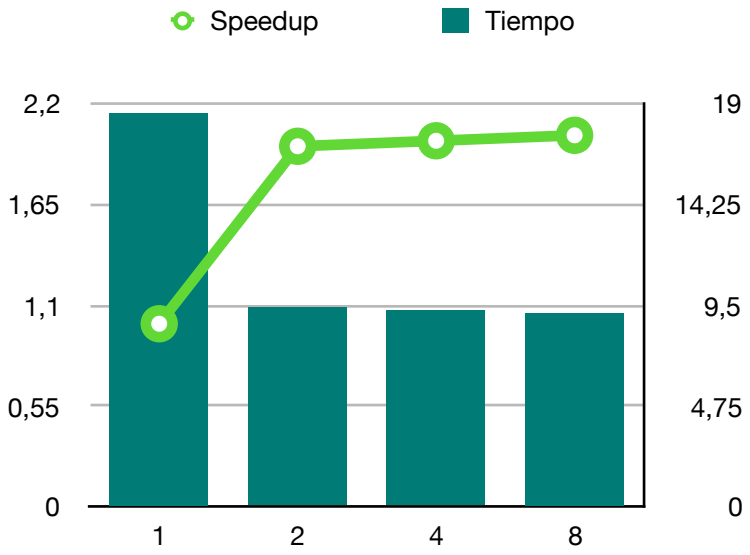
```
Creando Matrices...
Inicializando Matrices...
Multiplicando en Serie...
Tiempo en serie 18.572261

Multiplicando en Paralelo Schedule Static

2 threads --

LS= 1 -> tm= 9.340021, sp= 1.988460 (1)
LS= 51 -> tm= 9.134725, sp= 2.033149 (1)
LS= 101 -> tm= 9.384790, sp= 1.978975 (1)
LS= 151 -> tm= 9.240651, sp= 2.009843 (1)

4 threads --

LS= 1 -> tm= 9.916193, sp= 1.872922 (1)
LS= 51 -> tm= 9.569995, sp= 1.940676 (1)
LS= 101 -> tm= 9.269633, sp= 2.003559 (1)
LS= 151 -> tm= 9.315445, sp= 1.993706 (1)

6 threads --

LS= 1 -> tm= 9.129985, sp= 2.034205 (1)
LS= 51 -> tm= 9.234566, sp= 2.011168 (1)
LS= 101 -> tm= 9.431292, sp= 1.969217 (1)
LS= 151 -> tm= 9.345275, sp= 1.987342 (1)

8 threads --

LS= 1 -> tm= 9.138016, sp= 2.032417 (1)
LS= 51 -> tm= 9.342399, sp= 1.987954 (1)
LS= 101 -> tm= 9.211198, sp= 2.016270 (1)
LS= 151 -> tm= 9.279533, sp= 2.001422 (1)


Multiplicando en Paralelo Schedule Dynamic

2 threads --

LS= 1 -> tm= 9.353977, sp= 1.985494 (1)
LS= 51 -> tm= 9.216839, sp= 2.015036 (1)
LS= 101 -> tm= 9.318269, sp= 1.993102 (1)
LS= 151 -> tm= 9.234650, sp= 2.011149 (1)

4 threads --

LS= 1 -> tm= 9.953428, sp= 1.865916 (1)
LS= 51 -> tm= 9.493472, sp= 1.956319 (1)
LS= 101 -> tm= 9.661407, sp= 1.922314 (1)
LS= 151 -> tm= 10.047054, sp= 1.848528 (1)

6 threads --

LS= 1 -> tm= 9.334041, sp= 1.989734 (1)
LS= 51 -> tm= 9.827343, sp= 1.889856 (1)
LS= 101 -> tm= 10.051504, sp= 1.847710 (1)
LS= 151 -> tm= 9.346205, sp= 1.987145 (1)

8 threads --

LS= 1 -> tm= 9.677291, sp= 1.919159 (1)
LS= 51 -> tm= 10.016531, sp= 1.854161 (1)
LS= 101 -> tm= 9.468939, sp= 1.961388 (1)
LS= 151 -> tm= 9.919572, sp= 1.872285 (1)


Multiplicando en Paralelo Schedule Guided

2 threads --

LS= 1 -> tm= 9.407464, sp= 1.974205 (1)
LS= 51 -> tm= 9.371168, sp= 1.981851 (1)
LS= 101 -> tm= 9.174092, sp= 2.024425 (1)
```

```
LS= 151 -> tm= 9.409878, sp= 1.973698 (1)

4 threads --

LS= 1 -> tm= 9.554644, sp= 1.943794 (1)
LS= 51 -> tm= 9.272561, sp= 2.002927 (1)
LS= 101 -> tm= 9.349686, sp= 1.986405 (1)
LS= 151 -> tm= 9.215718, sp= 2.015281 (1)

6 threads --

LS= 1 -> tm= 9.313362, sp= 1.994152 (1)
LS= 51 -> tm= 9.285775, sp= 2.000077 (1)
LS= 101 -> tm= 9.505499, sp= 1.953844 (1)
LS= 151 -> tm= 9.348487, sp= 1.986660 (1)

8 threads --

LS= 1 -> tm= 9.309971, sp= 1.994879 (1)
LS= 51 -> tm= 9.265304, sp= 2.004496 (1)
LS= 101 -> tm= 9.552558, sp= 1.944219 (1)
LS= 151 -> tm= 9.677630, sp= 1.919092 (1)
```

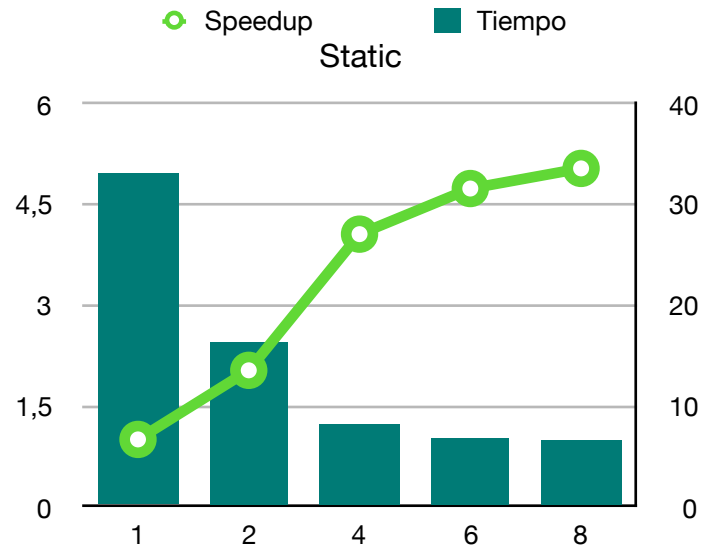
Como se puede ver para cada numero de threads diferente (2,4,6,8) he hecho el calculo con el tamaño de trozo (1,51,101 y 151) para cada una de las planificaciones diferentes de gestión de paralelizacion de bucles.

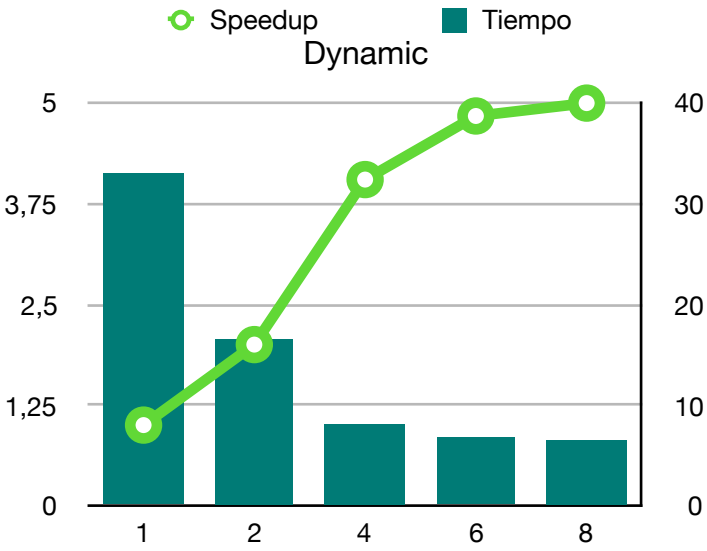
Como se puede ver en la salida, las planificaciones dynamic y guided no aportan mejoras al speedUp, de hecho los resultados son mejores con la planificación static. Esto era de esperar ya que estas planificaciones aportan mejoras significativas en el caso de que cada vuelta del bucle pueda tener una carga de trabajo distinta, y por ello el tiempo de ejecución de cada vuelta sea variable.
Estos algoritmos de planificación tienen mas lógica y son mas complejos que el tipo static por eso en este caso, en el que cada ejecución del bucle siempre va a hacer el mismo numero de operaciones, veamos una perdida de eficiencia en los tipos schedule dynamic y guided. Por tanto la mejor planificación para este caso es static.

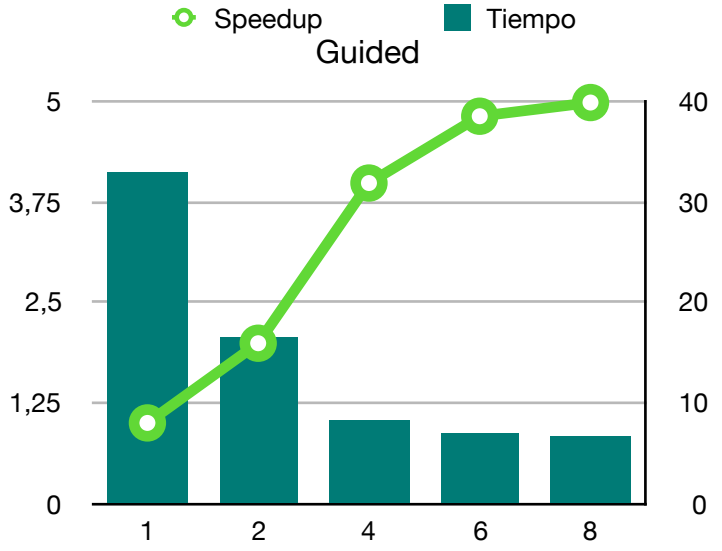
Prueba en maquina de dos núcleos: (STATIC)

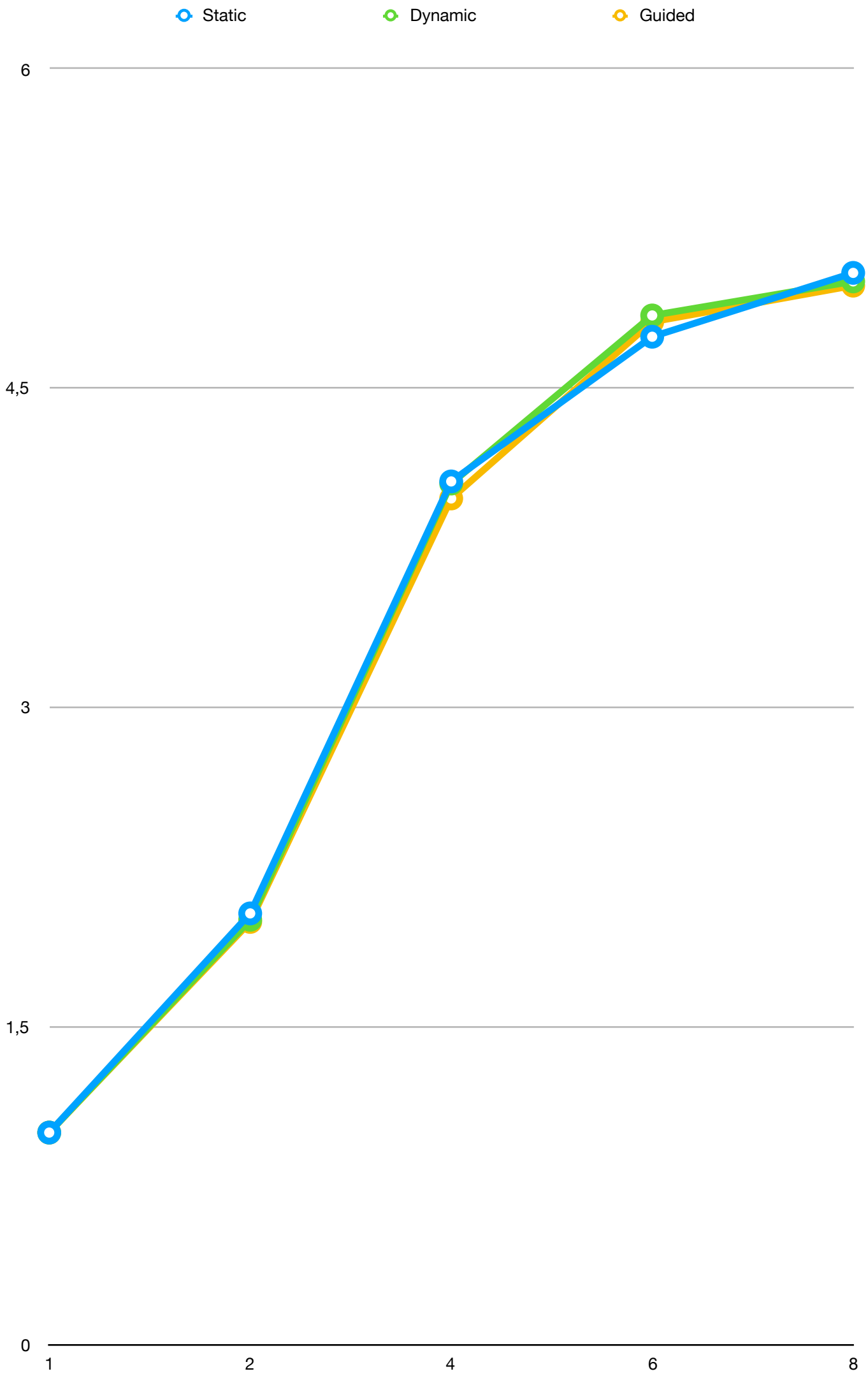


Prueba en maquina de cuatro núcleos:








Static
Dynamic
Guided
6
4,5
3
1,5
0
1
2
4
6
8

La salida del programa correspondiente a las anteriores graficas es:

```
Creando Matrices...
Inicializando Matrices...
Multiplicando en Serie...
Tiempo en serie 33.038687


Multiplicando en Paralelo Schedule Static

2 threads --

LS= 1 -> tm= 16.271335, sp= 2.030484 (1)
LS= 61 -> tm= 16.614949, sp= 1.988492 (1)
LS= 121 -> tm= 16.960586, sp= 1.947968 (1)
LS= 181 -> tm= 16.781812, sp= 1.968720 (1)

4 threads --

LS= 1 -> tm= 8.127140, sp= 4.065229 (1)
LS= 61 -> tm= 8.685584, sp= 3.803853 (1)
LS= 121 -> tm= 8.922015, sp= 3.703052 (1)
LS= 181 -> tm= 8.899222, sp= 3.712537 (1)

6 threads --

LS= 1 -> tm= 6.961692, sp= 4.745784 (1)
LS= 61 -> tm= 7.382261, sp= 4.475416 (1)
LS= 121 -> tm= 7.652253, sp= 4.317511 (1)
LS= 181 -> tm= 8.945618, sp= 3.693282 (1)

8 threads --

LS= 1 -> tm= 6.588604, sp= 5.014520 (1)
LS= 61 -> tm= 7.225769, sp= 4.572342 (1)
LS= 121 -> tm= 7.076222, sp= 4.668973 (1)
LS= 181 -> tm= 7.504451, sp= 4.402545 (1)


Multiplicando en Paralelo Schedule Dynamic

2 threads --

LS= 1 -> tm= 16.496378, sp= 2.002784 (1)
LS= 61 -> tm= 16.928809, sp= 1.951625 (1)
LS= 121 -> tm= 16.972242, sp= 1.946631 (1)
LS= 181 -> tm= 16.918812, sp= 1.952778 (1)

4 threads --

LS= 1 -> tm= 8.153369, sp= 4.052152 (1)
LS= 61 -> tm= 8.667338, sp= 3.811861 (1)
LS= 121 -> tm= 8.894838, sp= 3.714366 (1)
LS= 181 -> tm= 8.923099, sp= 3.702602 (1)

6 threads --

LS= 1 -> tm= 6.816403, sp= 4.846939 (1)
LS= 61 -> tm= 7.571556, sp= 4.363527 (1)
LS= 121 -> tm= 7.806467, sp= 4.232220 (1)
LS= 181 -> tm= 8.956754, sp= 3.688690 (1)

8 threads --

LS= 1 -> tm= 6.600067, sp= 5.005811 (1)
LS= 61 -> tm= 7.297277, sp= 4.527536 (1)
```

```
LS= 121 -> tm= 7.212985, sp= 4.580446 (1)
LS= 181 -> tm= 7.595247, sp= 4.349916 (1)


Multiplicando en Paralelo Schedule Guided

2 threads --

LS= 1 -> tm= 16.592530, sp= 1.991178 (1)
LS= 61 -> tm= 16.716126, sp= 1.976456 (1)
LS= 121 -> tm= 16.363390, sp= 2.019061 (1)
LS= 181 -> tm= 16.493275, sp= 2.003161 (1)

4 threads --

LS= 1 -> tm= 8.292692, sp= 3.984072 (1)
LS= 61 -> tm= 8.817829, sp= 3.746805 (1)
LS= 121 -> tm= 8.687863, sp= 3.802855 (1)
LS= 181 -> tm= 8.562646, sp= 3.858467 (1)

6 threads --

LS= 1 -> tm= 6.866680, sp= 4.811450 (1)
LS= 61 -> tm= 7.055422, sp= 4.682737 (1)
LS= 121 -> tm= 7.507094, sp= 4.400996 (1)
LS= 181 -> tm= 8.804072, sp= 3.752660 (1)

8 threads --

LS= 1 -> tm= 6.631692, sp= 4.981939 (1)
LS= 61 -> tm= 6.926059, sp= 4.770200 (1)
LS= 121 -> tm= 7.271922, sp= 4.543322 (1)
LS= 181 -> tm= 7.452645, sp= 4.433149 (1)
```

Como puede verse en la salida del programa, el tamaño optimo de bloque de trabajo es el mas alto para la planificación static. Para las otras dos planificaciones a mas tamaño de bloque peor tiempo de ejecución.

Concluyo por tanto que la mejor configuración para este problema es static, con bloques de trabajo de tamaño 180.
En caso de utilizar OMP 3.0 utilizar solo un bucle for colapsando en el segundo nivel.