UNIVERSIDAD DE GRANADA

Arquitectura y Computación de Altas Prestaciones — Uso de PThreads

Daniel Pedrosa Montes

https://github.com/moshidev

abril del 2023

Esta obra está bajo una licencia Creative Commons "Atribución-NoComercial-CompartirIgual 4.0 Internacional".

Índice general

Capítulo 1:
Resolución de los ejercicios propuestos.

§1.1: Búsqueda del máximo de un vector sin uso de mecanismos de exclusión mutua explícita.

Resolvemos el problema dividiendo los trozos del vector entre las distintas hebras. Cada hebra almacena el máximo encontrado en su trozo asignado. Después de esperar a que todas las hebras terminen buscamos el máximo de entre estos valores devueltos.

Ejecutamos con `make run`.

§1.2: Búsqueda del máximo de un vector usando mecanismos de exclusión mutua explícita.

Para ello modificamos el primer ejercicio de forma que ahora en vez de encontrar el máximo de entre los valores devueltos por las hebras, inmediatamente antes de terminar de ejecutarse la hebra esta actualiza atómicamente el mayor valor encontrado, mediante el uso de `pthread_mutex_lock`.

Ejecutamos con `make run`.

§1.3: Cálculo del Índice de Jaccard.

Conociendo la intersección de los dos conjuntos podemos conocer este índice. La mejor forma que encontramos para calcular la intersección de dos conjuntos es la de utilizar un *hashset*. Escogemos la implementación *khash*.

Paralelizamos la carga de los datos en dos hebras, una por conjunto. De no ser así ocurrirían condiciones de carrera al no ser la implementación escogida *thread-safe*.

Una vez tenemos los dos conjuntos en memoria, paralelizamos el tamaño de la intersección entre tantas hebras como queramos. Podemos hacer esto porque la lectura no modifica la estructura de datos.

Ejecutamos con `make run`.

1.3.1: Trabajo futuro.

Podríamos paralelizar la carga de datos si permitiésemos que la estructura de datos fuese reentrante. Para ello tal vez podríamos encontrar una solución que usase un cerrojo de espera ocupada por cada *bucket* junto a un *mutex* que sincronizase el redimensionado de la estructura de datos cuando fuese necesario (similar al problema de los lectores-escritores).

§1.4: Cálculo de Pi mediante MPI y PThreads.

Paralelizamos dividiendo la carga de trabajo primero entre procesos y luego en cada proceso entre hebras, de forma equitativa.

Ejecutamos con `ppi.sh`.

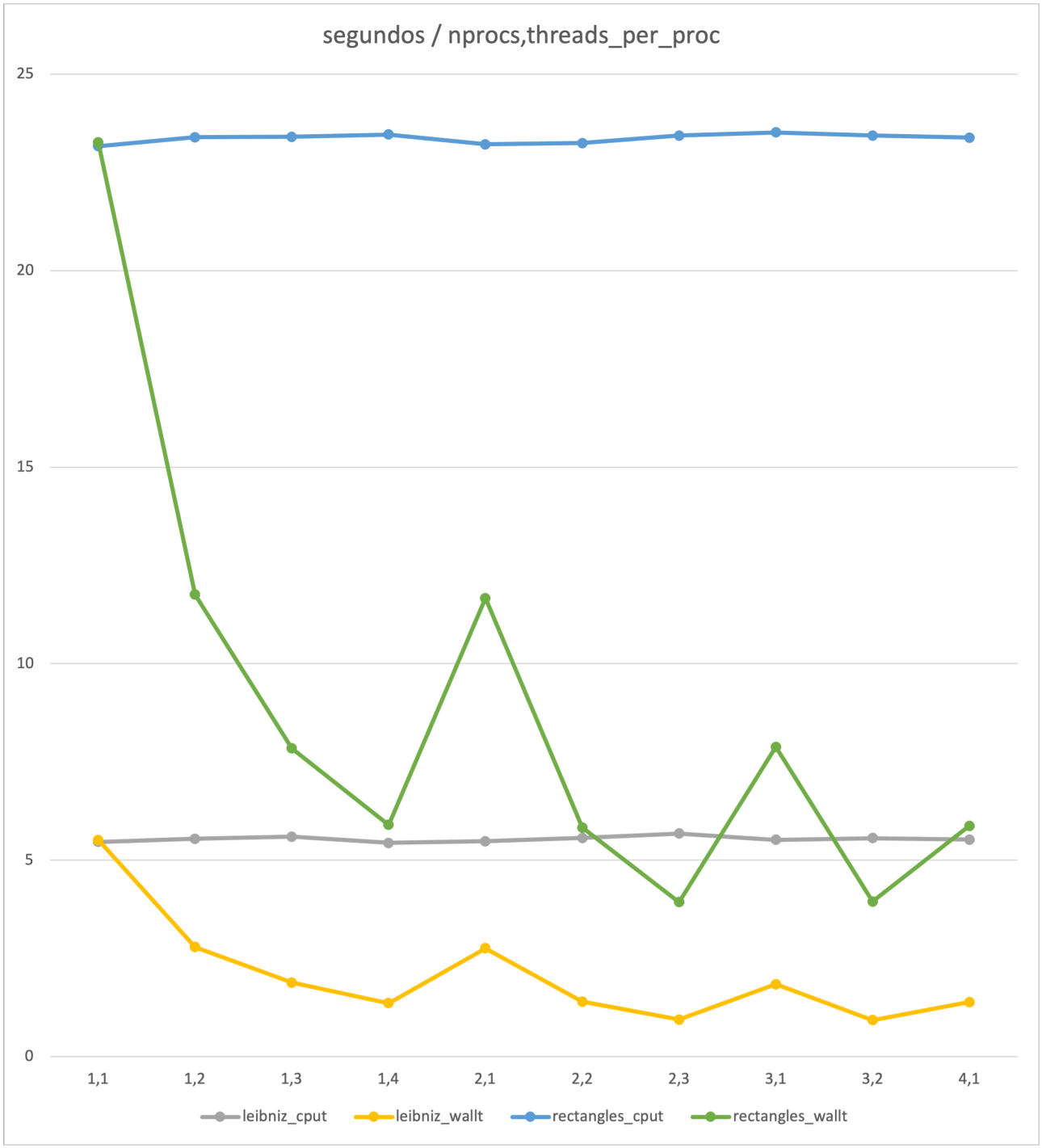

Figura 1.1: Relación segundos / número de procesos,hebras por proceso. Podemos observar cómo son resultados similares cuando tenemos un mismo número de unidades de ejecución, independientemente de su naturaleza. Esto se debe a que el número de mensajes intercambiados es mínimo y nuestros algoritmos no se ven afectados por el tipo de acceso a memoria.