Trabajo INFO 188 – Programación en paradigma funcional y paralelo:

Batalla de Sorting paralelo

Estudiantes:

- **Nicolas Yevenes**
- **Luis Berrocal**
- **Victor Silva**
- **Luciano Guzman**

Profesor: Cristobal Navarro

Introducción

El ordenamiento de datos es esencial en el mundo de la informática, especialmente con el crecimiento exponencial de los volúmenes de datos y el advenimiento de las inteligencias artificiales. Para optimizar esta tarea, por trivial que pueda parecer, los algoritmos paralelos aprovechan arquitecturas modernas como CPUs multicore y GPUs, reduciendo significativamente los tiempos de ejecución.

El presente informe compara el desempeño de algoritmos paralelos de ordenamiento implementados en OpenMP para CPU y CUDA para GPU, seleccionados según investigaciones recientes. Se analizan métricas como tiempo de ejecución, speedup y eficiencia paralela para evaluar su rendimiento en distintos tamaños de datos y configuraciones de hardware, destacando las ventajas y limitaciones de cada enfoque.

Radix Sort Paralelo: CPU

- **Optimización basada en enteros:** Radix Sort es particularmente eficiente para datos numéricos, ya que evita comparaciones y ordena en base a los bits o dígitos de los números. Esto reduce la complejidad y aprovecha mejor las capacidades de una CPU multicore, que está optimizada para operaciones de baja latencia como el acceso a memoria y operaciones matemáticas simples.
- **Escalabilidad en múltiples núcleos:** En entornos modernos, Radix Sort puede distribuir su trabajo (contar bits, reorganizar datos) entre múltiples hilos, lo que permite un aprovechamiento eficiente de los núcleos de la CPU sin depender de la latencia asociada con transferencias de memoria masivas entre dispositivos.

Merge Sort: GPU

- **Uso intensivo de memoria de acceso aleatorio:** Las GPU son extremadamente eficientes en el manejo de operaciones de acceso paralelo a memoria. Merge Sort, al trabajar en dividir y combinar bloques de datos, se adapta bien a la arquitectura paralela de los GPUs.
- **Buen rendimiento en problemas de gran tamaño:** Aunque Merge Sort consume más memoria que otros algoritmos, este costo se mitiga en la GPU debido al acceso rápido a su memoria global y compartida. Además, la naturaleza del algoritmo permite que las GPUs trabajen eficientemente con grandes lotes de datos al dividir el problema en subproblemas paralelizables.

Máquina

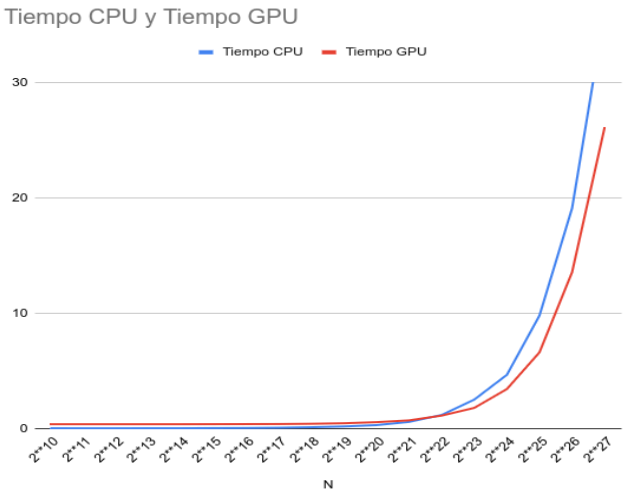
Para la implementación del código y las subsecuentes pruebas de rendimiento se hizo uso del servidor Carbón provisto por el profesor Cristóbal Navarro, pues además de ser un recurso común para el curso, está dotado de una GPU Nvidia esencial para la ejecución de código en CUDA. Las CPU y GPU del servidor son las siguientes:

CPU: Intel(R) Core(TM) i7-6950X

GPU: NVIDIA RTX 3090

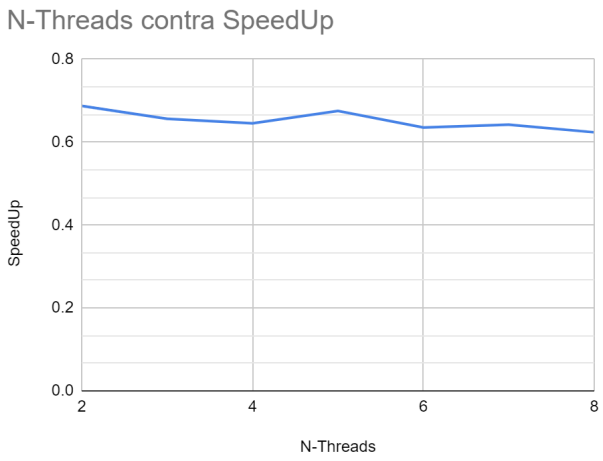
Benchmarking

Tiempo vs n:



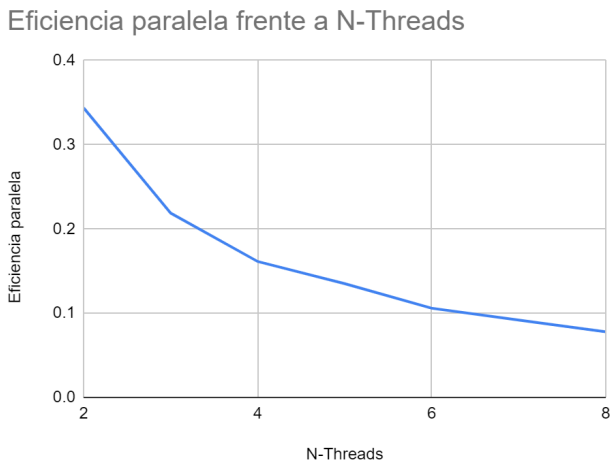
En arreglos más pequeños CPU es más eficiente, pero al ir creciendo, la eficiencia de GPU se vuelve aparente.

SpeedUp [CPU] vs N-Threads:



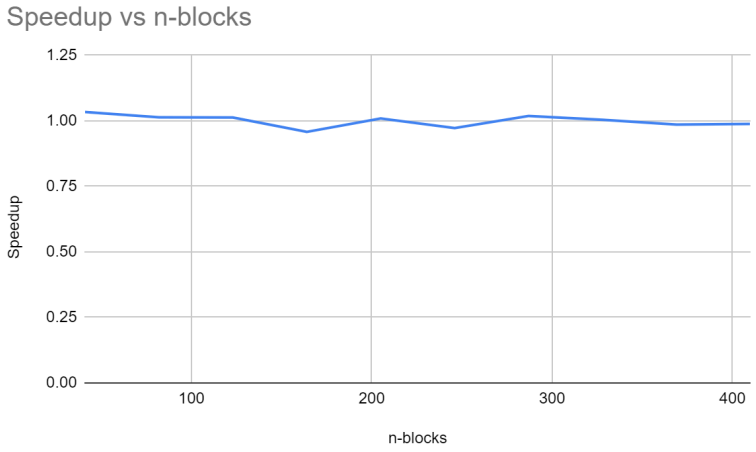
El speedup fluctúa en un rango de 0.6 a 0.8, lo cual no muestra beneficio alguno con el aumento de threads.

Eficiencia Paralela [CPU] vs N-Threads



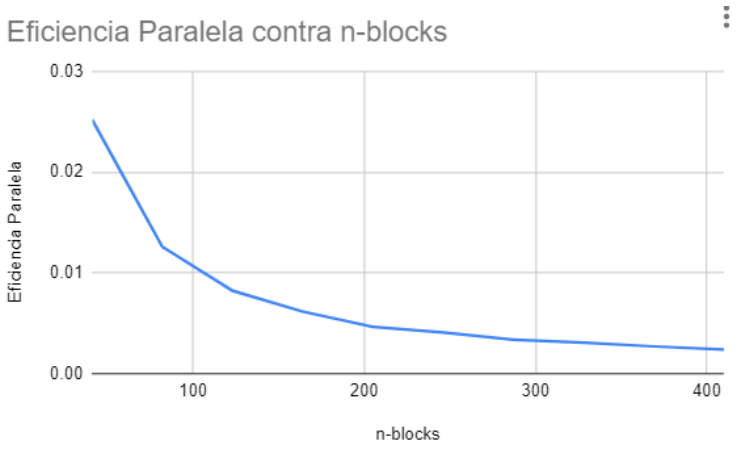
Claramente, no se puede apreciar una buena eficiencia en el algoritmo paralelizado.

SpeedUp[GPU] vs N-Blocks



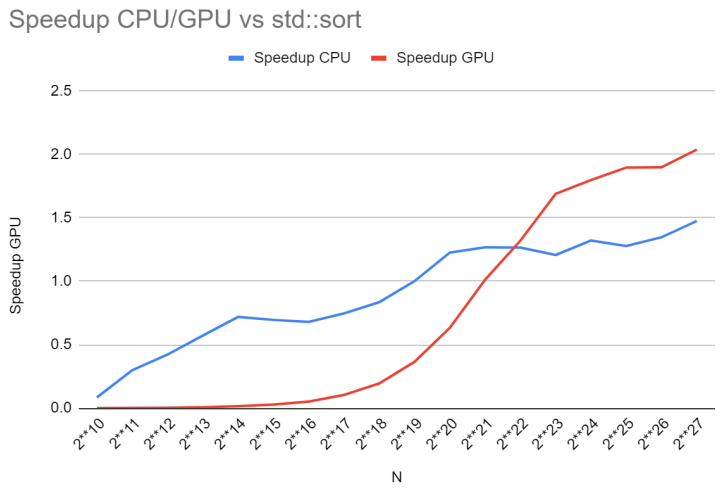
Como la razón del speedup se mantiene constante, el incremento de performance no varía.

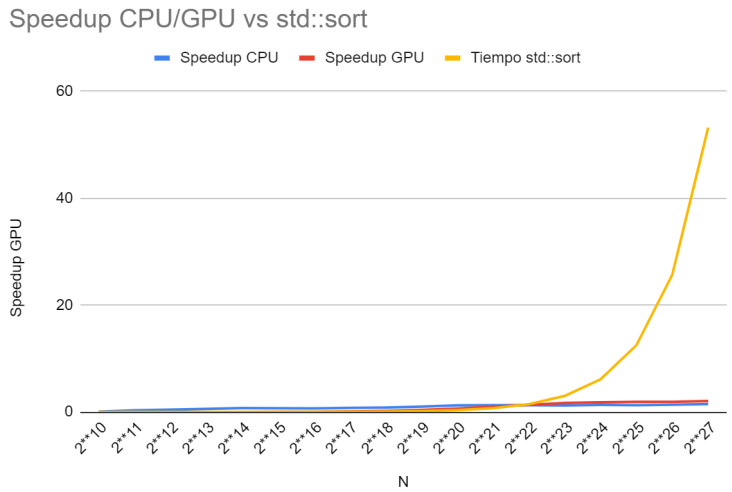
Eficiencia Paralela vs N-Blocks



Claramente, no se puede apreciar una buena eficiencia en el algoritmo paralelizado.

Speedup[GPU y CPU] vs Sort(Librería STL)





En los gráficos se aprecia que ambos enfoques (GPU y CPU) no lograron superar de manera consistente el rendimiento de la librería std::sort hasta una cantidad de valores muy elevada, probablemente debido a desafíos en la implementación, como organización o distribución de tareas.

Conclusión:

A partir de los gráficos se puede observar que las mejoras de los algoritmos no fueron las esperadas, esto puede deberse a que no se hizo una buena organización y/ o distribución

de las tareas en los threads, lo que pudo provocar que el aumentar el número de threads no afectará significativamente en el rendimiento de los algoritmos implementados.

Bibliografía:

- NVIDIA, "Improved GPU Sorting," disponible en el sitio oficial.
- Tanasic et al., "Sorting with GPUs: A Survey," arXiv.
- IEEE, "Comprehensive Comparison of Parallel Sorting Techniques" IEEE Xplore.