Reporte | Carlos Castro Correa 103531

Para este primer reporte leí el paper “Singular Value Decomposition on GPU using CUDA” .

A manera de introducción, puedo comentar que esta publicación trata sobre la descomposición en valores singulares de una matriz, además de documentar y comparar con otros método y software, propone un nuevo método basado en el uso de la GPU y el cómputo en paralelo, algo interesante es que este método supera el rendimiento que tienen otros competidores como MATLAB e Intel.

Antes de iniciar con los comentarios respecto al paper, me gustaría poner en contexto el contenido y la relevancia del tema: como matemático aplicado soy testigo de la importancia que la descomposición matricial tiene, no solamente en proyectos académicos y de investigación sino en nuestro desarrollo profesional, basta recordar que la mayoría de los métodos de estadística multivariada están basados en proyecciones y descomposiciones matriciales, por ejemplo, uno de las más utilizadas: la aplicación de componentes principales para encontrar una rotación óptima o la creación de índices.

Respecto al documento que leí puedo comentar que el desarrollo está centrado en el uso de la GPU y diferentes librerías como CUBLAS para calcular la descomposición espectral, por medio de procesos de bi-diagonalización y diagonalización (un tipo de descomposición matricial). Como vimos en clase, el uso de la librería CUBLAS se debe a que es una de las implementaciones para trabajar problemas de álgebra lineal en NVIDIA desde C, en otras palabras, esta librería facilita la explotación de los recursos y el poder de la tarjeta gráfica.

Antes de mostrar la comparación que hace respecto a otros métodos, el autor enlista una serie de avances en otras áreas, esta parte es muy interesante porque no solo indica que existen esfuerzos por trata el problema sino que nos muestra los distintos enfoques y herramientas que han sido implementadas para resolver el mismo problema. El autor también menciona e incluye el pseudocódigo de los algoritmos antes mencionados (diagonalización y bi-diagonalización) recalcando la implementación de estos en el GPU.

Para la comparación de método el autor siguió el proceso:

- Generar una muestra de matrices generadas de forma aleatoria para probar el rendimiento.
- Generar muestras de matrices de distintas dimensiones.
- Aplicar los algoritmos y medir el tiempo que tardan en implementar la descomposición espectral.

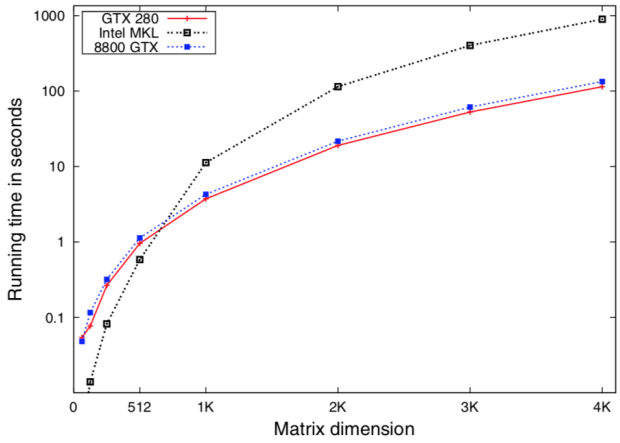
La tabla de tamaño y tiempos para cada algoritmo es la siguiente:

SIZE	SVD MATLAB	SVD MKL	SVD GTX 280	SVD 8800	Speedup MKL/280
64×64	0.01	0.003	0.054	0.048	0.05
128×128	0.03	0.014	0.077	0.116	0.18
256×256	0.210	0.082	0.265	0.319	0.31
512×512	3.19	0.584	0.958	1.129	0.61
1K×1K	72	11.255	3.725	4.28	3.02
2K×2K	758.6	114.625	19.6	21.656	5.84
3K×3K	2940	402.7	52.8	61.31	7.62
4K×4K	6780	898.23	114.32	133.68	7.85
1K×512	5.070	2.27	1.523	3.749	1.48
2K×512	10.74	12.8	3.118	4.072	4.11
4K×512	34.33	54.7	8.311	12.418	6.58
8K×32	24.310	17.112	3.506	-	4.88
8K×64	47.87	33.7	5.016	-	6.72
8K×256	107.57	103.8	13.96	-	7.4
8K×512	137.98	215	26.33	-	8.16
8K×1K	254.26	417	50.364	-	8.2
8K×2K	1371.9	808	111.3	-	7.25

Table 1. Total computation time for SVD (in seconds) for different matrices

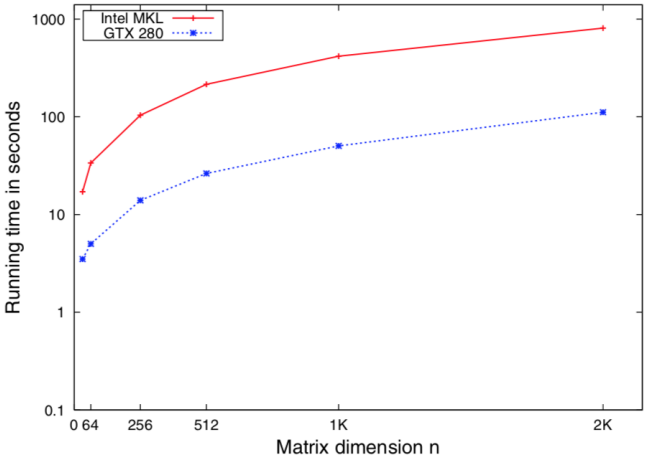
En la siguiente gráfica podemos observar el tiempo que tardo cada algoritmo de acuerdo al tamaño de la **matriz cuadradas** a factorizar:

- Podemos observar que el algoritmo propuesto por los autores supera el rendimiento de Intel a partir de cierto tamaño de matriz (cercano a 512), **después el algoritmo propuesto funciona mucho mejor.**
- **Recordemos que el método propuesto por los autores se basa en paralelización de tareas, en consecuencia, me parece lógico que el método de Intel sea más rápido para matrices pequeñas.**
- Es necesario considerar que las tareas realizadas en paralelo requieren pasar por el nodo maestro quien se encarga de repartir y decidir, generalmente tareas de dimensión pequeña corren más rápido en secuencial por este hecho. Aunque no se si Intel no haga algo en paralelo, pero me parece una posible explicación lógica.



También podemos observar los resultados utilizando **matrices rectangulares** (con diferente dimensión para filas y columnas):

- En este caso, el rendimiento del método propuesto por los autores supera desde el inicio a al de Intel.
- Una observación interesante es que parece que la diferencia entre el tiempo que le lleva a cada método es más o menos constante, obviamente entre mayor sea la matriz, más grande es el tiempo pero la diferencia entre ambas implementaciones parece ser la misma.



Finalmente, el autor menciona que el error resultante de esta implementación es pequeño por lo que en términos prácticos, la propuesta resulta bastante aceptable para utilizar en problemas reales.

- Como conclusión, podemos observar que las herramientas vistas en clase son una útil herramienta para resolver problemas de gran escala, esto es un recurso adicional pues durante la carrera solo utilizamos Matlab y tendemos a pensar que no hay nada mejor para problemas numéricos o de alta

precisión que utilizar Matlab o algo propietario.

- Si bien, en la mayoría de los problemas no nos vamos a enfrentar a una matrices de altísimas dimensiones, sí es importante que la solución o aproximación que encontremos a nuestros problemas se alcance en poco tiempo; en este sentido, el computo en paralelo, las GPU y las exntesiones de C son una ventaja comparativa y económica para nuestra carrera profesional.