GPU Project

Singular Value Decomposition

Aakash Kumar Dhal
(862464601)

Link to the Video Presentation

Part 1
https://www.loom.com/share/5a614de7794c48778edfb2b42ed89596?sid=0f4a64ff-cb82-40ad-94ec-fc0aecbab511

Part 2
https://www.loom.com/share/63eb27270a824deab71bfab837146b78?sid=b1836dcd-ee17-4bd9-8d4a-c4ee086fbf53

Project Idea

The goal of this project is to implement and optimize Singular Value Decomposition (SVD) on a GPU (Graphics Processing Unit) for efficient and high-performance computation. SVD is a fundamental linear algebra operation widely used in various fields, including machine learning, signal processing, and data analysis. By leveraging the parallel processing capabilities of GPUs, the project aims to accelerate SVD computations, particularly for large-scale datasets.

What is Singular Value Decomposition?

Singular Value Decomposition (SVD) is a powerful matrix factorization technique used in various mathematical and computational applications. Given a matrix A(m x n), SVD expresses it as the product of three matrices: U, Σ, and V^T, where U(m x m) and V(n x n) are orthogonal matrices, and Σ(m x n) is a diagonal matrix with non-negative singular values. These singular values in Σ provide crucial insights into the matrix's structure and significance. SVD is fundamental in linear algebra and data analysis. The matrices U and V capture the left and right singular vectors, respectively, which form an orthonormal basis for the matrix A. The singular values in Σ indicate the importance of each basis vector, allowing for dimensionality reduction. This property is particularly valuable in applications like image compression, where SVD can be employed to retain the most significant features while discarding less critical information. In signal processing, SVD is used for noise reduction and feature extraction. Additionally, SVD finds extensive use in machine learning tasks such as collaborative filtering and recommendation systems. By decomposing matrices into their singular values and vectors, SVD facilitates efficient computation and reveals the underlying structure of complex data, making it a versatile tool across various disciplines.

How is the GPU used to accelerate the application?

Singular Value Decomposition is a very compute expensive job. We need to optimize heavily to save on compute time and resources. Otherwise it is going to be a major overhead on CPU resources. We have implemented the entire program using python's NUMBA library. This code demonstrates a matrix multiplication operation using CUDA and Numba, a JIT (Just-In-Time) compiler for Python that translates Python functions to optimized machine code at runtime, especially for CUDA-compatible GPUs. `@cuda.jit` decorator indicates that the function `matmul` will be compiled for GPU execution using CUDA. `sA` and `sB` are shared memory arrays allocated for each block (`TPB x TPB` size), declared with `cuda.shared.array`. Shared memory is faster than global memory but is shared within a block. `x, y = cuda.grid(2)` fetches the thread's global coordinates in a 2D grid. `tx = cuda.threadIdx.x` and `ty = cuda.threadIdx.y` obtain the thread's local coordinates within its block. `bpg = cuda.gridDim.x` calculates the number of blocks per grid along the x-axis. The conditional statement `if x >= C.shape[0] and y >= C.shape[1]` checks if the current thread is outside the boundaries of the output matrix `C`. If so, the thread exits early. The matrix multiplication operation is done by loading chunks of data into shared memory (`sA` and `sB`). Each thread within a block loads a portion of matrices `A` and `B` into shared memory. `cuda.syncthreads()` synchronizes all threads within a block to ensure that the shared memory is fully populated before performing computations. Then, the code proceeds to perform a matrix multiplication in chunks using the shared memory. Each thread computes a part of the dot product by iterating through the shared memory arrays `sA` and `sB`. After the partial products are computed, `cuda.syncthreads()` is used again to ensure synchronization before proceeding with the next iteration. Finally, the result `tmp` (partial product) is stored in the output matrix `C`.

It was very tricky to find the ideal number of thread and thread blocks. We need to meticulously run through it.

I have parallelized wherever it could be parallelized. For example I have parallelized all of the matrix multiplications in this code base. This is the major overhead for the CPU.

I used the library numba from numpy and time libraries for performing all of the tasks.

Running the code:

1) Open Singularity shell from bender terminal using the following command:
   singularity shell --nv /singularity/cs217/cs217.sif
2) Install all of the dependencies from requirements.txt using:
   pip install -r requirements.txt

3) Run the following command:
   python test.py

And you are good to go!!!

Results

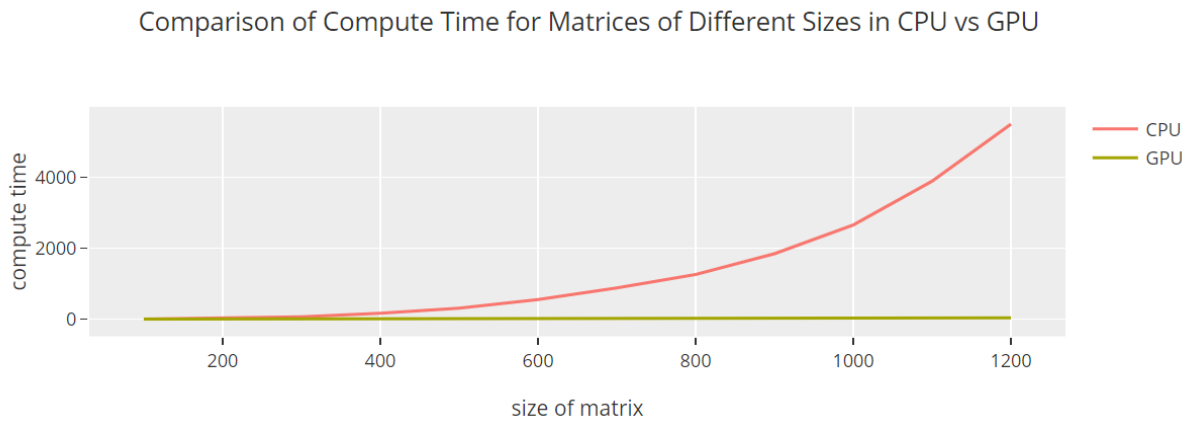
This is the result that we get when we run it.

```
U:  [[ 1.         -0.99587776 -0.07747295 -0.57047163 -0.64535165]
 [ 0.          0.04729557  0.29210472  0.          0.76388563]
 [ 0.          0.04124575 -0.11382586  0.82131731  0.        ]
 [ 0.          0.05864334 -0.42139448  0.          0.        ]
 [ 0.          0.02916097  0.84743326  0.          0.        ]]

S: [ 9.42330533 19.03158734  0.          0.         20.14944168]

Vt [[-2.87819181e-01 -3.12176729e-01 -4.68640987e-01 -5.65837634e-01
  -5.29064464e-01]
 [-4.68224331e-01 -4.36389790e-02  8.56976681e-02 -3.42855617e-01
   8.08682602e-01]
 [-9.61881814e-01  1.53567270e-01  2.02117509e-01 -5.95030927e-03
   1.01555779e-01]
 [ 4.08248290e-01 -8.16496581e-01  4.08248290e-01  8.02046653e-17
   6.78654861e-16]
 [-5.35131816e-01  3.99199005e-01  4.67625199e-01 -1.20878931e-01
  -5.66558957e-01]]
```

Running Time comparison

We can clearly see from the running time plots that the time taken to compute in case of CPU grows exponentially as the matrix size increases. Although there is an increase in GPU time as

well, the delta is not very high, which means it almost stays flat in comparison.



Tasks Breakdown

I did the entire thing on my own solely. Hope I am rewarded for that.