Google Summer of Code Program 2022
Project Proposal

Memory-Distributed Singular value decomposition.

Apr 18, 2022

1. Personal Details.

Name: V. Mano Sai Suraj.
Email: manosaisuraj12@gmail.com
University: Indian Institute of Technology Tirupati.
City: Vijayawada
Country of Residence: India
Timezone: IST (GMT + 05:30)
Phone Number: **9441128115**
Emergency Contact: **9014047971**
Primary Language: English
Links: LinkedIn, GitHub.

I am a second-year student pursuing a B.Tech. in Civil engineering at the Indian Institute of Technology, Tirupati. My semester will complete in The 2nd week of May exactly 2 to 3 days before the start of GSoC 2022. After that, I will be having holidays up to the 1st week of August. If I am selected, I shall be able to work around **40 hrs a week** on the project, though am open to putting in more effort if the work requires it.

2. Why this project? And Inspiration for this project.

I am a civil engineer but I always liked learning to code and solving problems on platforms like HackerRank and CodeChef.

I was seeking to do a challenging project this summer, and the google summer of code 2022 provides the perfect opportunity. After going through the organization list of GSoC 2022, the Heat projects list offered a perfect match.

I really like working with Data. That is the reason I learned Data Science and all the related areas(Numpy, pandas, PyTorch). When I saw your projects list I was 100% sure that I will work on one of these but not on any other projects for GSoC 2022, because it is the best way I can contribute to an open-source project and Also, at the same time learn so much in the areas which I liked the most. Once I made this decision, after reaching out to the Heat community, it provided to be a good fit for reaching my project goals. Becoming a part of the Heat community not only for GSoC but, in the future as well, is planned.

While being very well versed with the applications of python, Numpy, PyTorch, and Linear Algebra. This project will give me a head start on the learning curve. I hope to have an impact on Heat's mission to fill the gap between data analytics and machine learning libraries, through this project.

3. Project

The project Memory Distributed Singular value Decomposition intrigues me, here's an idea on how to go about this project:

Synopsis/ Project Abstract

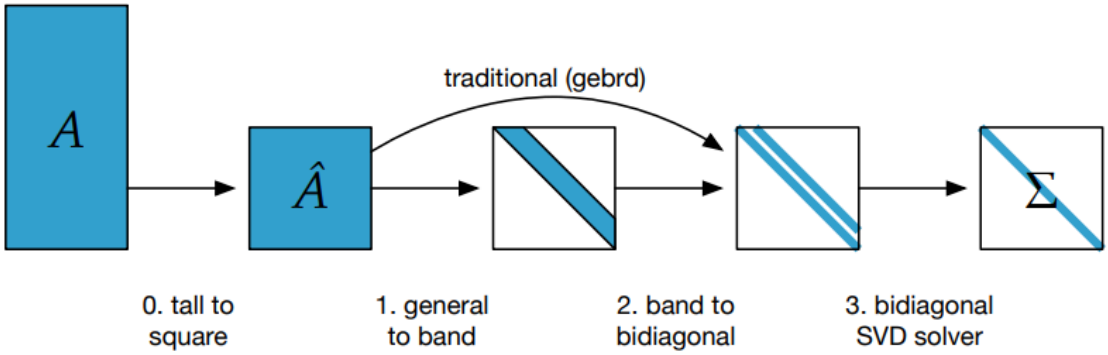
The major goal of the project is to develop a distributed SVD algorithm that is both efficient and numerically stable in Heat. This will be a major boost as the number of applications of the SVD algorithm is high, In most of the applications, the basic principle of **Dimensionality Reduction** is used. It means You want to reduce a high-rank matrix to a low-rank matrix while preserving most of the important information present in the matrix. This makes data visualization, and data analysis easy for a given input of very large DataSets, SVD algorithm does this by removing unnecessary data(**Denoising data**).

The most important applications are Image Compression, For recognition of faces, Removing Background from Videos, and Finally, the SVD algorithm is also the backbone of recommender systems such as Amazon, YouTube, Netflix, and many others. So, if we effectively implement this algorithm in Heat. It will be a great benefit for Heat and its users.

What it means to accomplish

The most important steps of this project are:

- Converting the given input matrix to a product of 2 matrices (Q&R) using QR factorization, where matrix R is an upper triangular matrix(band matrix).
- The reduction of the matrix R to a bidiagonal real matrix, This step uses the bulge chasing algorithm.
- Reducing the matrix from a bidiagonal to a diagonal matrix. Which will give us the matrix Σ(sigma) Its diagonal contains all the singular values. We can do this by the QR iteration technique.



Technical Details

Below I have explained the significant steps of the algorithm.

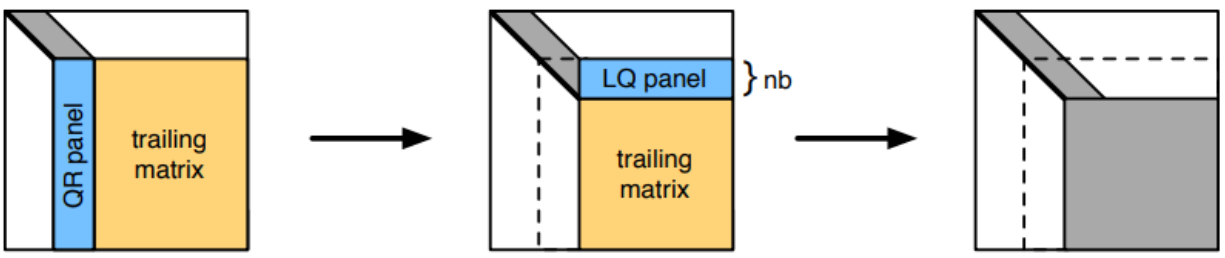
- We will assume the matrix taken as input be **A** and has **m** rows and **n** columns.
- If m >> n (or in some articles, it is given as If m > 5/3*n) then we will reduce the matrix A into a product of 2 matrices Q&R using the QR factorization. Here R will be an upper triangular matrix.
- When m > n (tall matrix), as R is upper triangular, its last m - n rows are zero. In this case, we **can drop the last m - n columns** of Q to form the reduced QR decomposition:

$$A = QR \qquad Q \in \mathbb{K}^{m \times n}, R \in \mathbb{K}^{n \times n}$$

- The function is already implemented. https://github.com/helmholtz-analytics/heat/blob/main/heat/core/linalg/qr.py
- During the initial period, I will try to optimize the already implemented code and try to make it as better as possible.

- We will also try to add some more parameters like mode = reduced, complete, R, etc which will give output specifically based on the value of parameters like in Numpy or PyTorch.
- Note: The QR decomposition is only unique up to the sign of the diagonal of R when the first k = min(m, n) columns of A are linearly independent. We have to take care of this as different outcomes are possible for the same matrix A given as input and this will produce different answers for **U,Σ, and V.**
- The matrix Q will be orthogonal(A real square matrix whose columns and rows are orthonormal vectors.) in the real case and unitary(A matrix whose inverse equals its conjugate transpose) in the complex case.
- The next step will be to convert the matrix R(a square matrix) to a band matrix(A matrix whose only nonzero elements lie on diagonal bands above and/or below the main diagonal).
- This process is done by computing a QR factorization of a block column to annihilate entries below the diagonal then computes an LQ factorization of a block row to annihilate entries right of the upper bandwidth, and updates the trailing matrix. This process is repeated until the entire matrix is brought to band form.

This picture shows how we will gradually reduce the given matrix into a band matrix.



- The next step is to reduce the band matrix to a real bidiagonal matrix. In this step, we will use the bulge chasing algorithm.

- The second stage proceeds in a series of sweeps, each sweep bringing one row to bidiagonal and chasing the created fill-in elements down to the bottom right of the matrix using successive orthogonal transformations.

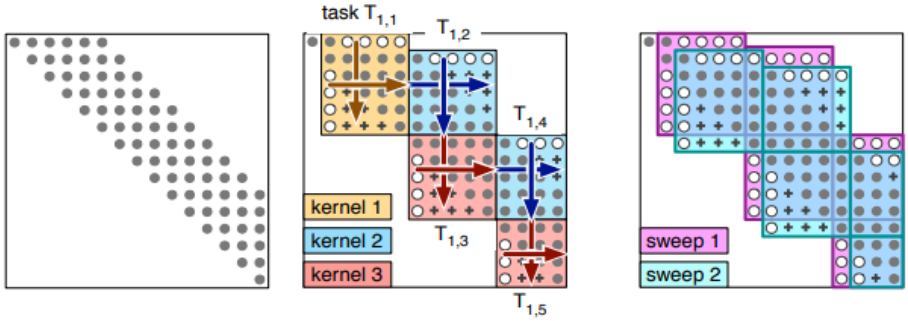
- In this process, we use 3 kernels, kernel 1, kernel 2, and kernel 3. Kernel 1 applies a Householder reflector from the right to eliminate a row right off the **super diagonal**(The diagonal of a matrix that lies directly above and to the right of the main diagonal), which also creates a bulge of fill-in beneath the diagonal.

- Then it applies a Householder reflector from the left to eliminate the first column of the bulge below the diagonal and applies the update to the first block column only.

- Kernel 2 continues to apply the left Householder reflector from kernel 1 to the next block column, creating a bulge above the upper bandwidth. It then applies a right Householder reflector to eliminate the first row of the bulge right of the upper bandwidth, updating only the first block row.

- Kernel 3 continues to apply the right Householder reflector from kernel 2, creating a bulge below the main diagonal. As in kernel 1, it then applies a left Householder reflector to eliminate the first column of the bulge below the diagonal and updates just the current block column.
- After kernel 3, kernel 2 is called again to continue the application of the left Householder reflector in the next block column.

- So, finally, we can say in short that, A sweep consists of calling kernel 1 to bring a row to bidiagonal, followed by repeated calls to kernels 2 and 3 to eliminate the first column or row of the resulting bulges, until the bulges are chased off the bottom-right of the matrix.



(a) initial band matrix **(b)** tasks in sweep 1 **(c)** overlap of sweeps

Figure 2.3: Bulge-chasing algorithm. "o" indicates eliminated elements; "+" indicates fill. Arrows show application of Householder reflector on left (→), which update a block row, and on right (↓), which update a block column.

- At this point, we converted the given input matrix into a bidiagonal matrix.
- Let the matrix obtained be D2(which is characterized by dual diagonality) let, $D2 = L1 \cdot R(Transpose) \cdot R1$. where R is the matrix obtained in the 1st step.
- Now, we will perform the last step, which is to convert the given matrix from a bidiagonal to a diagonal matrix.
- Bringing the Matrix D2 to a Diagonal Form: To make it diagonal, we will gradually assign a zero value to elements that are not on the main matrix diagonal. When operations are performed with elements below the main diagonal, the matrix is multiplied by the left rotation matrix. When processing the elements located above the main diagonal, multiplication by the right rotation matrix is performed.

- After iterative processing of the matrix D2, it is gradually reduced to a diagonal form. Then the resulting diagonal matrix is = D1.

  $D1 = L2 \cdot D2 \cdot R2$ as a result we get,

$$RT = L1T \cdot D2 \cdot R1T,$$
$$D2 = L\,T\,2 \cdot D1 \cdot R\,T\,2,$$
$$R\,T\,2 = L\,T\,1 \cdot L\,T\,2 \cdot D1 \cdot R\,T\,2 \cdot R\,T\,1,$$
$$R = R1 \cdot R2 \cdot D\,T\,1 \cdot L1,$$
$$R = R1 \cdot R2 \cdot D1 \cdot L1 \cdot L2,$$
$$A = Q \cdot R1 \cdot R2 \cdot D1 \cdot L2 \cdot L1,$$

Hence finally we will compute the values of U,Σ, and V.

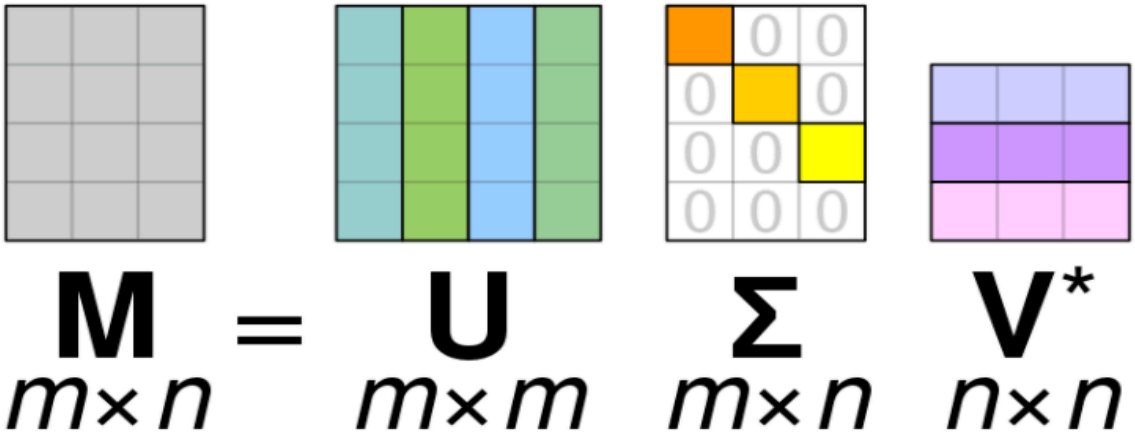
$U = Q \cdot R1 \cdot R2$

$\sum = D1$

$VT = L2 \cdot L1$

$$A = U \cdot \sum \cdot V\,T$$

Here, in the above equations, T represents the transpose of the matrix that is present at the left to it.



An alternative way:

- Similar to the above process If m >> n (or in some articles, it is given as If m > 5/3*n) then we will reduce the matrix A into a product of 2 matrices Q&R using the QR factorization. Here R will be an upper triangular matrix.

$$A = Q \cdot R$$

- Now, we will apply the **Golub-Kahan algorithm** to R, by which we will get $R = U \cdot B \cdot V\,T$

  Together this results in $A = Q \cdot U \cdot B \cdot V\,T$.

- In **Golub-Kahan Bidiagonalization**, we use two different sets of Householder reflectors to get a bidiagonal (instead of an upper Hessenberg) matrix.
- The difference between the previous method and this method is that in the previous method we go from a full matrix to a band matrix and then we will convert the band matrix to a bidiagonal matrix using the bulge chasing algorithm.

- But here in this method, We will convert the given input matrix **directly from full to bidiagonal matrix.**

This procedure is just like applying two separate QR factorizations, alternatingly applied to $A\ and\ AT$

The operations count is that for two QR factorizations, i.e., approximately

$2 \cdot m \cdot n\text{^}2 + 2 \cdot n\text{^}3\ which\ is\ more\ efficient\ if\ m > 5/3 \cdot n$

We can also use **Golub-Kahan -Lanczos Bidiagonalization Procedure.** Which also converts the given matrix into a bidiagonal matrix directly but its algorithm/ process is different.

I read many articles about these 2 methods, they are also Improved over the years, and in one of the latest methods - the algorithm is like this:

ALGORITHM 6.27: Golub–Kahan–Lanczos Bidiagonalization Procedure

(1) *choose* $v_1 = $ *unit 2-norm vector and set* $\beta_0 = 0$
(2) **for** $k = 1, 2, \ldots,$
(3) $\quad u_k = Av_k - \beta_{k-1}u_{k-1}$
(4) $\quad \alpha_k = \|u_k\|_2$
(5) $\quad u_k = u_k/\alpha_k$
(6) $\quad v_{k+1} = A^*u_k - \alpha_k v_k$
(7) $\quad \beta_k = \|v_{k+1}\|_2$
(8) $\quad v_{k+1} = v_{k+1}/\beta_k$
(9) **end**

Collecting the computed quantities from the first k steps of the algorithm, we have the following important relations:

$$AV_k = U_k B_k,$$
$$A^*U_k = V_k B_k^* + \beta_k v_{k+1} e_k^*,$$

<u>One example showing Golub-Kahan Bidiagonalization:</u>(I am attaching the photos because it is not possible to write these methods and algorithms directly.)

Example Consider

$$A = \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix}.$$

Householder reflectors applied alternatingly from the left and the right will be used to zero parts of the matrix as follows:

$$U_1^*A = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} & \mathbf{x} \end{bmatrix} \longrightarrow U_1^*AV_1 = \begin{bmatrix} x & \mathbf{x} & \mathbf{0} & \mathbf{0} \\ 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \end{bmatrix}$$

$$\longrightarrow U_2^*U_1^*AV_1 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{0} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{0} & \mathbf{x} & \mathbf{x} \\ 0 & \mathbf{0} & \mathbf{x} & \mathbf{x} \end{bmatrix} \longrightarrow U_2^*U_1^*AV_1V_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & \mathbf{x} & \mathbf{0} \\ 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & \mathbf{x} & \mathbf{x} \end{bmatrix}$$

$$\longrightarrow U_3^*U_2^*U_1^*AV_1V_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & \mathbf{x} & \mathbf{x} \\ 0 & 0 & \mathbf{0} & \mathbf{x} \\ 0 & 0 & \mathbf{0} & \mathbf{x} \end{bmatrix} \longrightarrow U_4^*U_3^*U_2^*U_1^*AV_1V_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & \mathbf{x} \\ 0 & 0 & 0 & \mathbf{0} \end{bmatrix} = B.$$

We can discuss and decide on the most efficient method to implement the SVD algorithm in Heat after observing all these methods based on their Time complexity, accuracy for large Data sets, and how well they will parallelize.

As most of this is math, it is difficult to explain some steps directly here. So, I tried to explain as many details as possible and used photos so that every small step is understandable.

4. Applying the algorithm to the memory-distributed data

With increasing data dimensions, there is a need to process data not on a single computing device, but in a distributed computing system.

It is possible to modify the Singular Value Decomposition method, in which the same type of operations, such as adding data or multiplying matrices, are divided into smaller parts and processed simultaneously by different devices, through this approach, it is possible to speed up the running time of the method.

I read a research article in which they divided the existing SVD algorithm into two parts:

1. Operations that could be performed in parallel by different computing devices.

2. Operations that were too difficult to perform in parallel, so it was decided to use traditional sequential data processing.

They distributed some operations that were of the same type, such as the multiplication of matrices and exponentiation among several computational processes, which speed up the algorithm.

When a 4-processor architecture of a distributed system has been used the results showed that with increasing the number of distributed operations, the **duration of calculations first decreased and then**

increased. The reason behind this is that distributed computations of small operations take more time than their simple sequential execution.

This graph shows the time taken vs the Number of operations performed by distribution.

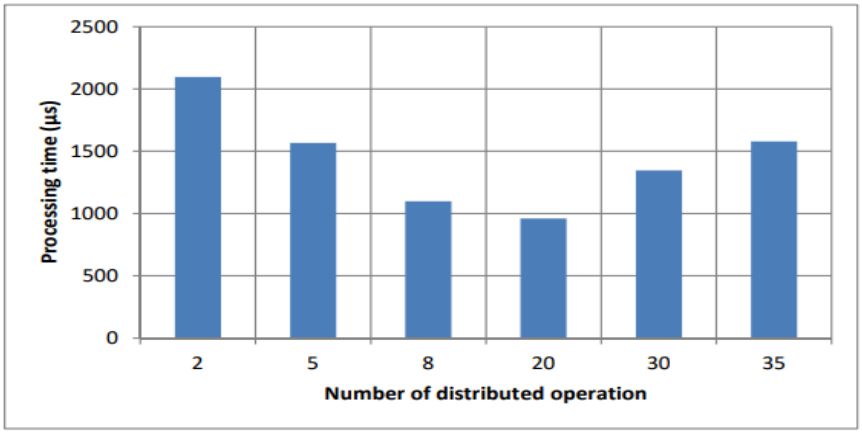


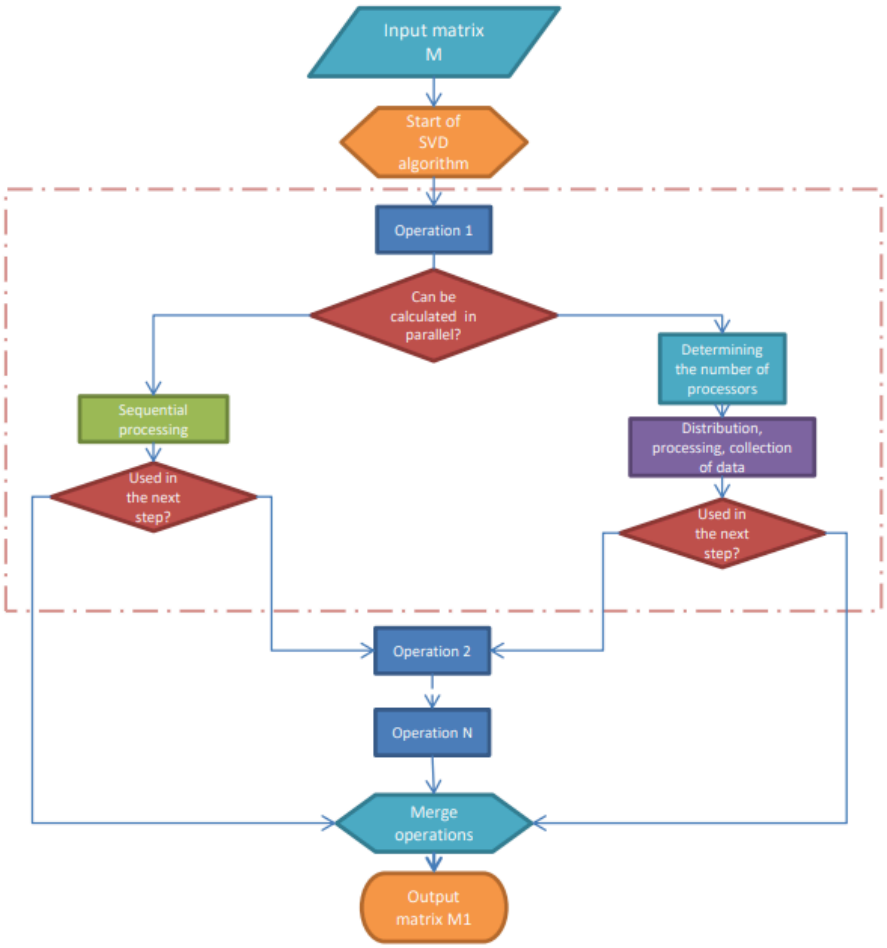
Figure 4. Dependence of SVD processing time on the number of distributed operations.

To find the number of distributed operations and the number of operations performed sequentially, an algorithm was proposed. This proposed algorithm allowed us to determine the required degree of modification of the SVD algorithm for different architectures of distributed systems with the highest efficiency.

In the algorithm, for every operation(at each step where it was necessary) to perform a certain action on the data, the possibility of its slave division for processing by different processors was checked. When modeling the operation of the algorithm, the researchers used a 4-processor architecture of a distributed system.

Similarly, we will divide the algorithm into steps and at each step, we will observe whether the particular operation is fast when performed by multiple processors or when performed sequentially based on that we can implement the complete algorithm (by merging all the individual steps) with as much efficiency as possible.

The flowchart below shows the algorithm for modifying the SVD method in distributed systems.



The following figure below shows the operation of an unmodified Singular Value Decomposition method, in which all data is processed by one processor, and a modified one, in which the task is distributed to several processors. Distributed processing can increase the efficiency of the computing system. **Elements of the modified method are highlighted in red.**

I found this figure in one of the articles. It gives a very good picture of how the SVD algorithm is implemented in a distributed system.

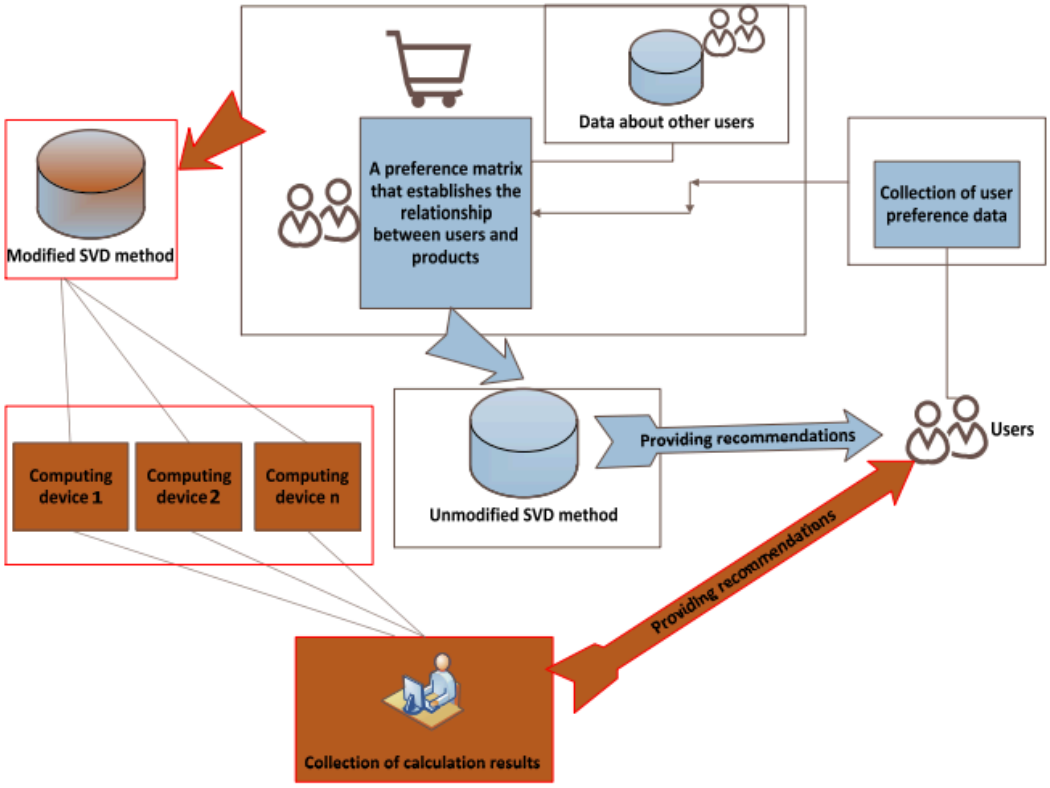


Figure 5. Implementation of the SVD method in a distributed system.

The figure shows a real-life use case of the SVD algorithm which is the **recommendation systems** in Amazon, YouTube, Netflix, and many others(which I already mentioned in the project abstract section) which helps all users get the best recommendations possible from these companies. The modified SVD method uses distributed system in which data is distributed to several processors because of which required results are calculated fast. In the end, all the obtained results are collected and used.

I searched many articles related to memory distributed SVD, but I found only a few explaining all the details properly, and I understood as much as I can from them. I am not an expert in parallel computing but I am very good at linear algebra and coding, so I hope to get a little support in the areas related to parallel/distributed computing.

5. Timeline

Before the community bonding period

- Discuss different possible approaches for the project with the mentors.
- Learn more about parallel programming and linear algebra.
- Contribute in general to Heat.
- Remain in constant touch with my mentors through Mattermost or GitHub.

Community Bonding Period(May 20 - June 12)

- Get acquainted with the code base of Heat and the procedure that needs to be followed to submit the code and get it reviewed.
- Discuss with the team what exactly needs to be done, That is discussing the effective implementation of the algorithm (including minute details, like the kind of dataset that is failing to give the correct output now, what should be algorithmically decided based on input data type, etc).
- Try to fix bugs to get further understanding.
- Completely understood the already present here, https://github.com/helmholtz-analytics/heat/tree/main/heat/core/linalg (Code related to the implementation of the svd algorithm).

Official Coding Period(June 13 - September 4)

Week 1 (June 13 to 20)

- Understand the relevant parts of the Heat's codebase (https://github.com/helmholtz-analytics/heat/blob/main/heat/core/linalg/basics.py and https://github.com/helmholtz-analytics/heat/blob/main/heat/core/linalg/qr.py) and try to figure out what the final product should look like, and start implementing the algorithm.

Week 2-3 (June 20 to 30)

- Begin optimizing the 1st step of the algorithm, that is QR factorization based on the dimensions of the input matrix(i.e based on the number of rows m and number of columns n) and taking care of special cases like complex numbers in the input matrix.
- Make sure that the matrix R is an upper triangular matrix for any type of input.
- This step is very important as it is necessary for both the ways discussed(Golub Kahan Bidiagonalization or bulge chasing algorithm in both of these the 1st step is QR factorization only).
- Workload has been kept less this week so that I can practice other crucial techniques such as adding test cases and understanding code review workflow.

Week 4 (July 1 to 7)

- By this time It would be completely clear regarding which algorithm we will use to convert the matrix R from upper triangular to Bidiagonal matrix. So I will be implementing the bulge chasing algorithm or Golub-Kahan Bidiagonalization.
- I will also test the QR algorithm this week and make sure that it is perfectly working by running some tests.
- If any issues need to be fixed then 1st priority will be to fix them before moving on to the 2nd step of the algorithm.

Week 5-6 (July 8 to 20)

- Implementing the bulge chasing algorithm or its alternative**.** Checking the outcomes for different datasets(input matrices) and fixing the bugs related to wrong outcomes.
- This implementation is the toughest part of the SVD algorithm, so I will be focusing on implementing this efficiently for most of July.

Note: Before July 29th(Phase 1 evaluation)

Week 7-8 (July 20 to 31)

- Evaluation of code by mentors and taking their insights for making it much better.
- Optimizing both the algorithms and code written until now, Also fixing issues if there are any.
- During this period I will again discuss with mentors regarding the best possible ways to optimize the code without affecting any other functions and try to make the whole code neat and organized.

Week 9 (August 1 to 7)

- Now, I will implement the last step, which is reducing the bidiagonal matrix into a diagonal matrix. This will be done using either the QR iteration method or the Divide & Conquer method, one provides more accuracy whereas the other provides results fast.
- Implementing one of these 2 algorithms.

Week 10 (August 8 to 15)

- Implementing the algorithm further and testing it.

Week 11 (August 15 to 21)

- Buffer time for testing the algorithm and fixing bugs or issues present.
- Using all the implemented techniques to finally get U,Σ, and V(transpose).

Week 12 (August 22 to 30)

- Making the algorithm better for some special cases.

- Take feedback from the community and iterate on all the 3 steps again and improvise on use cases. Ensure code quality by adding more test cases and working with more input Data Sets.
- Work to make documents, blogs, or videos to help increase the user base for this product(Subject to developer community approval).

Week 13 (September 1 to 5)

- Spare week in case of some work getting delayed, in case of any emergency or otherwise.

Week 14 (September 5 to 12)

- **Final week:** Final Evaluation of code by mentors and taking their insights for making it much better.
- Once again try to Optimize the code further and make the whole code much more understandable and organized.

September 12 - September 19

- Mentors submit final GSoC contributor evaluations

By this time the algorithm will be implemented, but if we want to extend the project further to 22 weeks. I am completely fine with it but I may not be able to work 35hrs a week as my new semester will start again. I can only spend around 20hrs a week after September 12th.

I hope to finish all the requirements of the project and successfully implement the algorithm by the 2nd week of September.

6. Technical Knowledge. (Also, proof that I am not just saying things but I have good knowledge in development and coding).

I am a 2nd-year undergraduate student at IIT Tirupati. I am enrolled in a 4-year B.Tech course. My major is Civil Engineering(it's more of Maths and Physics though). But as I am studying at an IIT we have a good amount of free time to learn multiple things.

My current **CGPA = 8.41/10**

These are some of the programming-related things I learned and achieved **apart from my college courses**.

- C++ (I did so much competitive programming during the last 2-3 months, C++, Data structures, and Algorithms are the main tools for it and I reached a 3-star rating.).

https://www.codechef.com/users/saisuraj27 (link to my CodeChef profile where I did most of my competitive programming with C++).

CE20B031 SAI SURAJ - ce20b031 | HackerRank(My Hacker rank profile where I learned python, and C++ and solved many problems).

- Python (I did the following 3 major projects in python along with solving many problems on the platforms mentioned above).
    - Rock, paper, Scissors.
    - Sudoku solver.
    - Shortest Path-finder.
- Data Science(Numpy, pandas) Apart from learning it from my pure interest, I also participated in a Boot Camp organized by Geeks for Geeks, and there **I analyzed 2 Datasets and achieved a Top 10 rank in the assignment conducted after the Bootcamp** where more than 2000 people participated.
    - https://www.kaggle.com/code/saisuraj27/geeks-for-geeks-articles-analysis-by-suraj
    - https://www.kaggle.com/code/saisuraj27/ted-talk-analysis-suraj

- I also completed Web Development Training from Internshala where I learned **HTML & CSS, Bootstrap, SQL, and PHP** and I scored 84% marks. In the Final Exam.
  https://trainings.internshala.com/s/v/815163/f8bd30c4

- **Linear Algebra:** I have a good idea in this field as I am studying this course for the 2nd time, 1st - during my 1st year in college as a part of the Basic engineering mathematics course, https://nptel.ac.in/courses/111105121.

  And now, for the 2nd time, in my current semester, as a math elective, I am studying this for the last 1 month and will be studying up to May 1st week.
  https://www.iittp.ac.in/pdfs/syllabus/MA2021.pdf

- Parallel computing: I have less prior understanding of this concept but I watched most of the tutorials and read the articles provided on the Heats GitHub page, Which increased my understanding of the project, and I am reading it continuously and will surely get good at it by the end of April. I am trying to get a complete understanding of it. **Edit:** Now I got a better understanding of this.

- **PyTorch:** I have pretty good knowledge of PyTorch, I didn't do any complete projects in this, but I learned this as suggested by a few of my seniors and this helped me to understand Heat's goals and this project.

- **Git:** I have also learned about version-controlled systems and I also read so much about Git for this project, I also downloaded Git Bash and practiced different commands, created different repositories in GitHub, and also created repositories directly in my local system with Vs code and committed the changes to my GitHub repositories. This gave me a good idea of Git.

7. Expectations from Mentor.

- Help me understand the existing code of Heat whenever I am incapable of doing so on my own.
- Suggest me some study material to have a clearer view of how things are done ideally, especially if I face a problem related to Parallel computing.
- Help me to come to a decision when I have more than one way of doing things and tell me why that is the best option.
- Take time to review my work and provide your timely insight and feedback.

8. Commitments

My university exams will be completed by May 17th, just 2 to 3 days before the announcement of accepted proposals. After this, I will have holidays up to the 1st week of August. Up to August 10th, I will be available for at least 40hrs a week through online platforms and am ready to extend whenever needed. I don't have any other commitments except learning and practicing problems related to data structures and algorithms for my future interviews, although My 1st preference would always be working for the GSOC project.

9. After GSoC

I would love to keep contributing to Heat even after GSOC and will be available to resolve issues. I would like to help with different projects by suggesting new ideas and participating in discussions. I will do whatever I can to help.

10. References

- https://www.research.manchester.ac.uk/portal/files/82231456/17m1117732.pdf
- https://www.researchgate.net/publication/350978772_Distributed_Singular_Value_Decomposition_Method_for_Fast_Data_Processing_in_Recommendation_Systems
- https://towardsdatascience.com/simple-svd-algorithms-13291ad2eef2
- https://wwwmayr.in.tum.de/konferenzen/Jass09/courses/2/Kleine_Albers_paper.pdf
- http://www.netlib.org/utk/people/JackDongarra/etemplates/node198.html
- https://github.com/helmholtz-analytics/heat/tree/main/heat/core/linalg

11. What would you consider to be successful participation in GSoC? What would make it a valuable experience from your point of view?

I would consider implementing each step in the algorithm a success. Contributing to an open-source project during my summer will be a wonderful experience for me.

From my point of view discussing ideas with the mentors, implementing algorithms from scratch, and making a good impact on the project will be valuable experiences for me.

12. GSoC participation and TimeZone:

- This is my first time applying for the Google Summer of Code. I did not submit a proposal to any other organization.
- The difference between the TimeZone of the mentors and mine is just **3.5 hours**. So our working hours are 90%(almost) the same. So the difference in times will not at all be a problem.

Conclusion

I thank all the mentors for spending their valuable time answering all my doubts and also giving suggestions related to my proposal.

I would be happy to receive any kind of suggestions or feedback. Also, you can contact me any time through mattermost or WhatsApp.

My WhatsApp number is: **+91 9441128115**

Finally, I would love to get the opportunity to work with Heat during my summer, I want to discuss various ideas with the mentors and tackle the problems together.

Thank you,

Yours sincerely

V. Sai Suraj.