Parallel Computation of Eigenvalues

Eignvector and eignvalues which is a concept often studied in Linear Algebra, has a lot of engineering applications in fields like physical problems, quantum mechanics and even in computer science. An eigenvector or characteristic vector of a [linear transformation](https://en.wikipedia.org/wiki/Linear_map) is a non-zero [vector](https://en.wikipedia.org/wiki/Vector_space) that only changes by an overall scale when that linear transformation is applied to it. Geometrically an eigenvalue tells whether the special vector x is stretched or shrunk or reversed or left unchanged—when it is multiplied by λ. More formally, if T is a linear transformation from a [vector space](https://en.wikipedia.org/wiki/Vector_space) V over a [field](https://en.wikipedia.org/wiki/Field_(mathematics)) F into itself and v is a vector in V that is not the [zero vector](https://en.wikipedia.org/wiki/Zero_vector), then v is an eigenvector of T if T(v) is a scalar multiple of v. This can be expressed in form of equation as:

|  |
| --- |
| T(v)= *λ*v |

where λ is a [scalar](https://en.wikipedia.org/wiki/Scalar_(mathematics)) in the field F, known as the eigenvalue, characteristic value, or characteristic root associated with the eigenvector v. There is a correspondence between n by n square matrices and linear transformations from an n-dimensional vector space to itself.

A variety of algorithms are available to calculate eigenvalues and eigenpairs and the objective of this report is to compute the eign values of given n\*n matrix using parallelization. It includes using various type of algorithms and parallelization strategy and optimization of code wherever possible. The server to be used for developing and testing the code is xsede and for parallelization both openMP and MPI to be used along with GPU.

Please revise the proposals addressing the following points

1. I suggest that you focus on one method for dense matrices. The simplest algorithm to implement in my opinion is Jacobi’s method, though it is not the most efficient it is easy to parallelize. The QR algorithm is more efficient but includes several steps to find the eigenvalues and eigenvectors. But you may have some other method in mind. Please choose one method as your primary and if time permits what would be your second choice.

I am implementing Jacobi’s method and QR method. Also I have considered QR as primary method.

1. Also, you should choose a primary target for which to make your code efficient, either distributed memory using MPI or shared memory/single node using OpenMP. Including GPU is fine, but only after you have good performance on a node without accelerator. So, select your primary target and if time permits you can include GPUs

As primary

1. You need to come up with a plan to assess how efficiently your code uses the platform. How would you know if your code could run in half the time? 10% of the time, or even less?
2. The plan need to include how you will test scalability in regards to number of threads used in case of an OpenMP implementation, and the number of processes in case of an MPI implementation, and how scalability depends on matrix size.