High-Performance Computing Fall 2022 Assignment 3

Amin Alimoradi - Sudent ID 100884445

December 14 2022

Contents

0.1 Introduction

In this assignment, the objective is implementing the Manager Worker MPI model for computing maximum Eigenvalues of random matrices and finding the distribution of the Eigenvalues for a large number of matrices. For a symmetric matrix A[$n \times n$] with elements normally distributed. An Eigenvector v lies along the same line as Av: **Av =** λ**v**, the Eigenvalue is λ, The above equation can also be written as: (A - λ) = 0 In a random matrix, λmax is the maximum Eigenvalue, and the distribution of such maximum Eigenvalues can be determined.

What is Eigenvalue?
The eigenvalue is explained to be a scalar associated with a linear set of equations which, when multiplied by a nonzero vector, equals to the vector obtained by transformation operating on the vector.

The hardware that we run our tests on is:
CPU Information Model name:
CPUs is 8 - Intel(R) Core(TM)
Cache size is 6144 kb
CPU Max MHZ = 3900
CPU Min MHZ = 1200
Threads per core is 2
Core(s) per socket: 4

RAM Information
Memtotal is 16310212 kb
Cached is 1797260 kb

0.2 Method

In this assignment, we investigate the speed up and efficiency of parallelization by computing the former distribution of λmax for large and small matrices using the Manager-Worker MPI model. In addition, we want to show whether the distribution of λmax is normal or not. In this study, I run the code on SHARCnet to show the distribution of large matrices.

0.2.1 Implementation

For the present study, we implemented a Manager-Worker system with the manager keeping track of the number of computations, receiving and collecting the outputs of each worker, and giving instructions to the ideal worker. In each Worker, random matrices are generated, Eigenvalues are computed, and results are sent back to the Master.

0.3 Psuedocode

0.3.1 Worker Pseudocode

Data: Boolean (conv= converges), seed for PRNG or pseudorandom number generator, recvd_tags: Receiving tag it means communication between worker and manager, Tag: sending tag, Status: fail-safe, Eig = Eigenvalues, A = Matix, ierr: For errors
MPI_Bcast = give – integer and receive from manager to all worker (One to all)
Allcoate Matrix
Loop MPI_reev = Receive the seed (for generating random number) from the manager (integer) and any worker can receive
recv_tag status = checks that worker received seed
Makes matrix and calculates eigenvalues
If no convergence then "bad end"
If convergence then give tag = 0 otherwise tag =1
MPI_send = Send back maximum of Eigenvalues and also Eigenvalues should not be an integer
End loop
Deallocate matrix (because we are done with memory)
End subroutine worker
Subroutine making matrix (Use lapack, Lapack is Linear Algebra Package)
Call DGEEV (DGEEV computes for an N-by-N real nonsymmetric matrix A, the eigenvalues and, optionally, the left and/or right eigenvectors.)
Subroutine compute Eigenvalues (use lapack)

0.3.2 Manager Pseudocode

Data: num_procs (number of threads), Seed (PRNG), ierr (for errors), recv = 0 (counter for Eigenvalues), Worker (worker that we communicate with), exit tags, sent tag, ndat (number of Eigenvalues(λ)), P (loop counter), failed,
Initialize: set matrix and number of eigenvalues
B_cast - send matrix
Allocate eigs (number of eigenvalues)
Initializing tags
Write to file eigs
Loop
If a worker is ready, make seed
If recvd, number of eigenvalues, n_procs 2
Otherwise stop
Because every matrix is random and has its own seed
Give seed, check for done
Worker is busy
Once exit tags are sent
Finish
MPI_Recv (receive buf (which is not a singular integer))
Can receive from anybody
Find out which worker sent out eigenvalues
Find out if convergent

If tag=0 then
recvd = recevd +1
Add eigenvalues to list
Write data
Otherwise, increase the failed counter
Print results
Now make worker available
End loop
Deallocate list
Initialization data
n = size of the matrix
ndat = number of eigenvalues
End subroutine manager

0.3.3 Main Pseudocode

Module global: because for MKL & MPI must be global variables
MKL_VSL_TYPE Routines for quasi-random number generation(Quasi-random sequences are functions from the positive integers to the unit hypercube.)
Data: n (number of matrix size must be an integer), exit tag, void = 0, proc_num,num_procs (must be an integer) matrix splitting parameters,
End global
Get random seed, use the hard drive and iostat=istat(Must be an integer) (An IOSTAT value is a value assigned to the variable for the IOSTAT= specifier.), end
Get a random number and use MKL, called every time a matrix is filled (with open random stream)
If ierr is not equal to 0 then print returns flags
End if
Deallocate pseudo-random number stream
Pseudo-random number stream
Main: Wtime is the time and variable ok (everything is okay with code ?)
call MPI (make sure everything is set up property)
If proc_num is equal to 0 then use MPI library clock (wtime)
If everything is good and proc_num is equal to 0 call manage and worker.
Print walltime
call finalize
End main
Initialize MPI (check everybody is okay)
If ierr is not equal to 0 then
print MPI comm size failed
End IF
Mpi comm rank: returns the calling process's rank in the specified communicator. It's often necessary for a process to know its own rank.
IF ierr is not equal 0 then
Print mpi comm rank failed
Finalizes all code, Call MPI Barrier (It is Communicator), Call MPI Initialized (The functions MPI_INIT and MPI_FINALIZE are used to initiate and shut down an MPI computation,).

0.4 Experiment

In this experiment, we investigate the distribution of the maximum Eigenvalues at different matrix sizes and different Eigenvalue numbers. We compare wall time and then used PYTHON For plotting the data. The results are as follows:

Matrix size	Number of Eigenvalues	Walltime
100×100	600	0.47 sec
400×400	15000	336 sec
2000×2000	600	2052 sec
2000×2000	10000	SHARCnet(Goerges data)
5000×5000	1000	SHARCnet(Goerges data)
6000×6000	1100	91413 sec

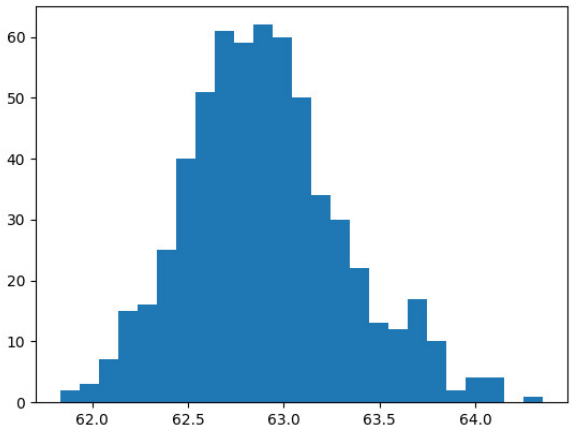

Figure 1: $\lambda = 600$, *1000x1000-sized matrices*

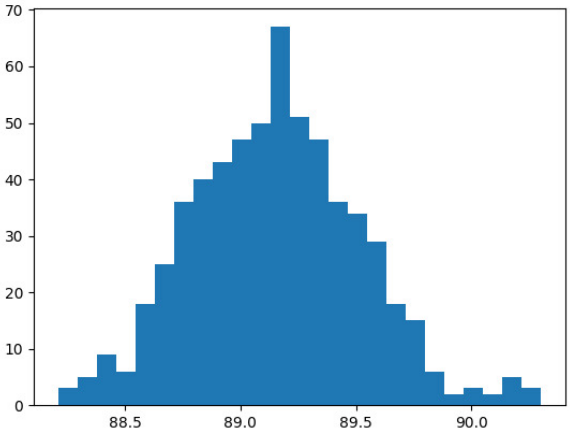

Figure 2: $\lambda = 600$, *2000x2000-sized matrices*

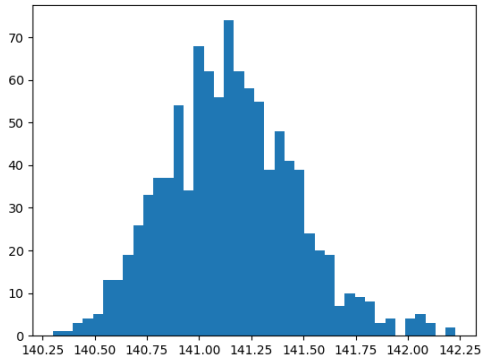

Figure 3: $\lambda = 1000$, *5000x5000-sized matrices*

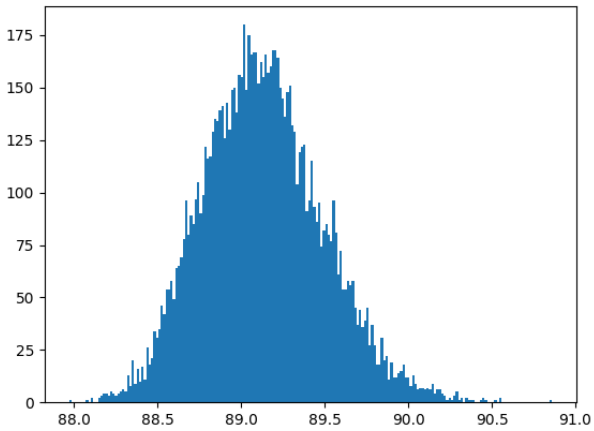

Figure 4: $\lambda = 10000$, *2000x2000-sized matrices*

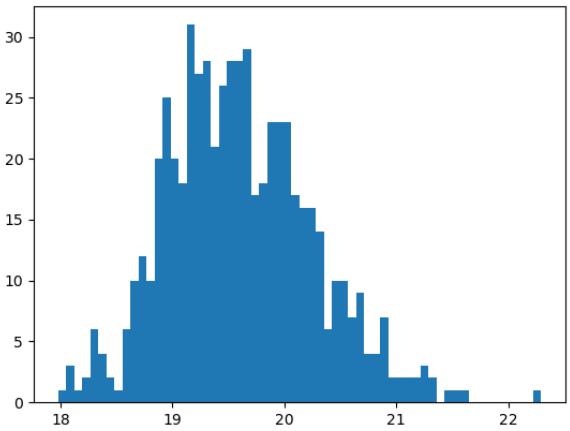

Figure 5: $\lambda = 600$, *100x100-sized matrices*

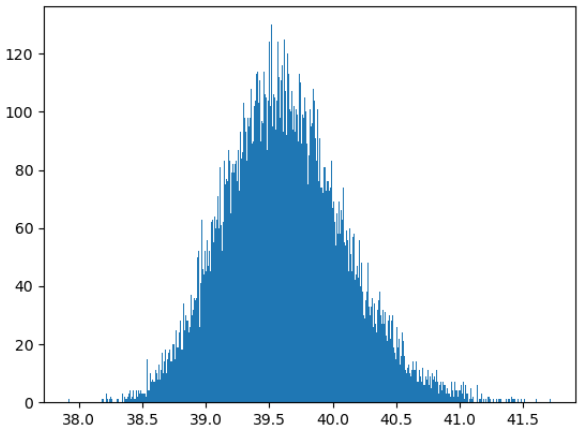

Figure 6: $\lambda = 15000$, *400x400-sized matrices*

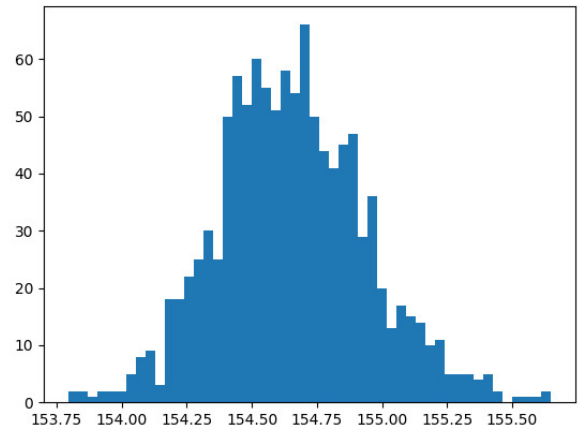

Figure 7: $\lambda = 1100$, *6000x6000-sized matrices*

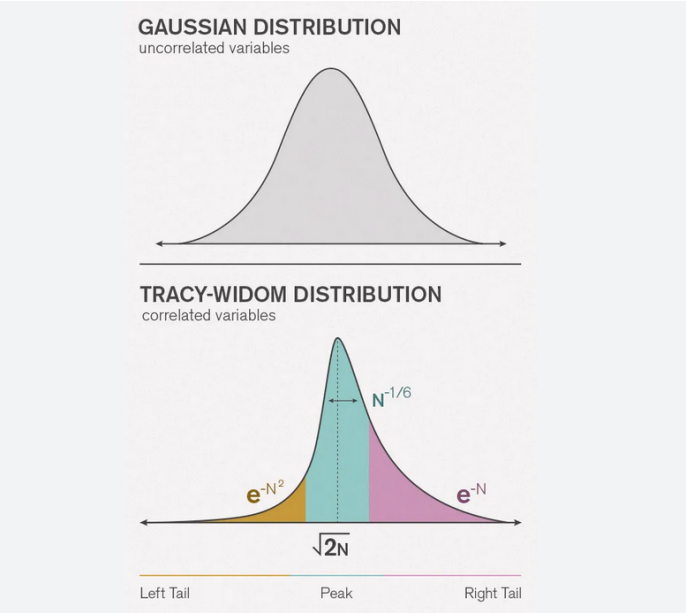

Figure 8: *Tracy-Widom distribution VS gaussian distribution*
https://www.wired.com/2014/10/tracy-widom-mysterious-statistical-law/

In order to implement the code, Mersenne Twister pseudorandom number generation from the Intel Math Kernel Library (MKL) was used to seed the random numbers. LAPACK was implemented to calculate the maximum eigenvalues. The LAPACK subroutine is O(n^3).
In large matrices, Lapack will use multi-threading automatically unless you tell it otherwise. Following that, we don't use multi-threading because it will affect on Wall times
generally, these results are presented in figures to show that the Eigenvalues follow a Tracy-Wisdom distribution with a Gaussian distribution as one limit for small sizes.
In these two figures 4 and 6 we can see that the distribution function which can be fitted to the matrices with small sizes would be similar to the Tracy-Widom distribution function presented in figure 8.
We can conclude that The fitted distribution function would be more similar to Tracy-Widom when the number of Max Eigenvalues increases.

The memory requirement is:
$100 \times$ N/MB

which N = Cpu size,
M = matrix sizes - (for example 1000×1000)
B = 8 bytes.
This will show how much of the CPU is going to use. (percentage)

0.5 Conclusion

As a result of the computational experiments, we are able to conclude that the study's expected outcomes have been verified. When the number of Workers increased, MPI implementation of the Manager and Worker model resulted in a reduction in wallclock times. Well, it should model a Tracy-Widom distribution and not a Gaussian so we would reject the zero hypotheses.