An overview of dense eigenvalue solvers for distributed memory systems

Davor Davidović[1]

Centre for Informatics and Computing, Ruđer Bošković Institute, Zagreb, Croatia,
davor.davidovic@irb.hr

MIPRO 2021
Data Science and Biomedical Engineering (DS-BE)
29 Sep - 1 Oct 2021, Opatija, Croatia

Croatian Science Foundation

Outline

Motivation

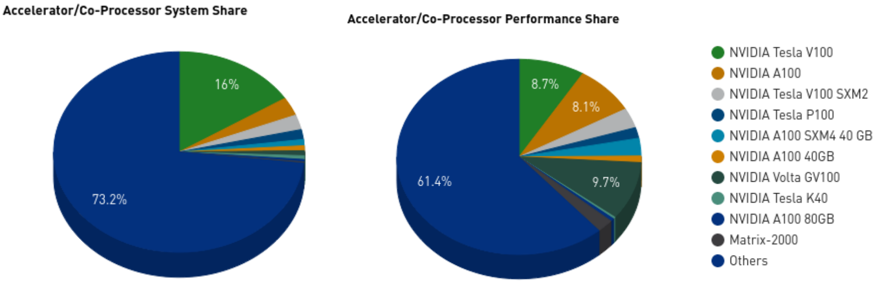
Eigenvalue problems

- Fundamental mathematical problem in many research fields
  - molecular dynamics,
  - computational quantum chemistry,
  - finite element modeling,
  - multivariate statistics.
- Solving an extremely large-scale eigenproblem with $\geq 10^{10}$ elements
- The most time-consuming part of the codes

Motivation

- Computationally demand problem with complexity $O(n^3)$
- The solution requires large machines $\rightarrow$ supercomputers and large computational clusters
- Many scientific software are using the eigensolvers from the highly-tuned libraries (e.g. MKL, ScaLAPACK, cuSOLVER)
- GPU accelerators are becoming more attractive $\rightarrow$ much higher performance compared to CPU ($> 100\times$)

Motivation



[1] Source: https://www.top500.org

Outline

Eigenvalue problem

Problem statement

- The standard eigenproblem is defined as:

$$AX = X\Lambda,$$

where $A \in \mathbb{R}^{n\times n}$ is a square matrix, $\Lambda \in \mathbb{C}^{n\times n}$ is diagonal with the sought-after eigenvalues λ_i and the columns of $X \in \mathbb{R}^{n\times n}$ contain the associated eigenvectors.

Generalized eigenproblem

$$AX = BX\Lambda \quad (1)$$

where A and B are square. If $B = I$ then it becomes the standard eigenvalue problem.

Problem solving - eigensolvers

Direct eigensolvers $\rightarrow$ dense matrices, all eigenvalues

1. Reduce dense matrix to tridiagonal/Hessenberg form (T)

   $$Q^T AQ \rightarrow T,$$

2. Compute eigenvalues of T

   $$TX = X\Lambda,$$

   Divide&Conquer, MRRR, bisection and invert iteration, QR iteration

Iterative eigensolvers $\rightarrow$ sparse matrices and/or subset of eigenvalues

- Iterative process $\rightarrow$ until convergence
- QR algorithm, Krylov subspace iteration, Jacobi, Power iteration

Problem solving - choosing the right solver

Choose the right method/implementation based on the type of the problem your try to solve

Problem types

1. **Number of eigenvalues** required $\rightarrow$ all, inner/outer spectrum, smallest/largest
2. **Shape of the matrix** (A) $\rightarrow$ rectangular/square, symmetric/unsymmetric, unitary, symmetric positive definite,. . .
3. **Structure of the matrix** $\rightarrow$ dense, sparse, band, tridiagonal, structured sparseness, Toeplitz,. . .
4. Are eigenvectors required?

Outline

Eigensolvers

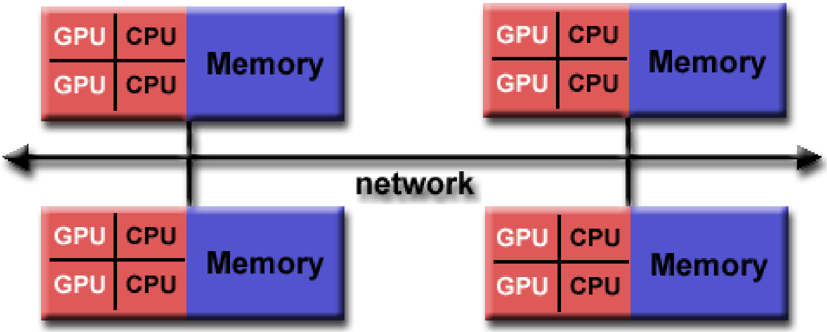
The scope of this research

- Only the maintained and open access libraries are observed
- Focus on dense eigensolvers for distributed memory systems
- GPU-accelerated

Type of solvers based on memory architectures

- Shared memory systems
- Distributed memory systems
- Hybrid memory systems

Hybrid memory systems



2

[2] Source: https://hpc.llnl.gov/training/tutorials/introduction-parallel-computing-tutorial

Shared-memory eigensolvers

- Exploit parallelism on a node level
- Multi-CPU + (optionally) GPUs
- Parallelisation models: OpenMP, POSIX threads, CUDA (OpenCL)
- **Advantages**: Easy to program and use, large number of available libraries, highly-tuned, no expensive memory copies through network
- **Disadvantages**: Low scalability, bounded by the available main/GPU memory, cannot compute large-scale problems

Numerical libraries

LAPACK, MKL, OpenBLAS, MAGMA, cuSOLVER, Eigen,. . .

Distributed-memory eigensolvers

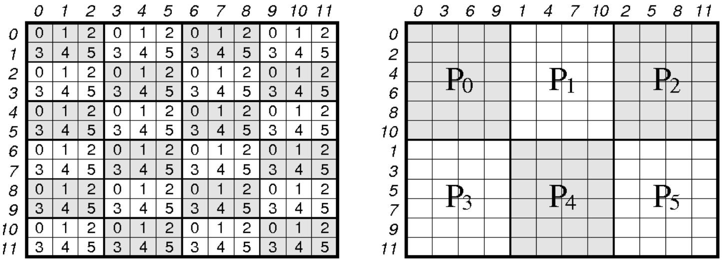
- Run across numerous computer nodes
- Hybrid models: MPI, OpenMP, CUDA
- **Advantages**: Scalable, capable to solve extremely large eigenproblems, high efficiency
- **Disadvantages**: Expensive memory transfers through network, hard to program, small number of GPU-based libraries

Numerical libraries

ScaLAPACK, ELPA, Elemental, EigenEXA, FEAST, SLATE, PARPACK,. . .

ScaLAPACK

- Standard in distributed memory Linear Algebra
- CPU-only, based on MPI parallel model
- `PxSYEV` (QR alg.) and `PxSYEVD` (Divide&Conquer) routines for all eigenpairs of symmetric/Hermitian matrix
- Approach based on reducing the matrix A into tridiagonal form
- Parallelisation: 2D Block Cyclic Distribution of input matrix A

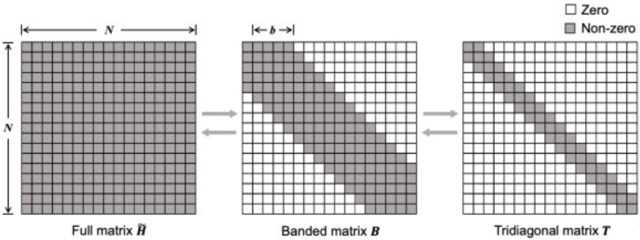


(a) block distribution over 2 x 3 grid. (b) data distribution from processor point-of-view. [3]

[3] Source: http://www.netlib.org/utk/papers/factor/node3.html

ELPA

- Compute eigenvalues and eigenvectors of large dense symmetric matrices
- **ELPA1**: One stage approach → reduce from dense A to tridiagonal form
- **ELPA2**: Two stage approach → reduce from A to band from then to tridiagonal
- Support distributed GPU in ELPA2 → back-transformation part in computing the eigenvectors



[4]

[4] Source: GPU-acceleration of the ELPA2 distributed eigensolver for dense symmetric and hermitian eigenproblems, https://www.sciencedirect.com/science/article/abs/pii/S0010465520304021

SLATE

- Software for Linear Algebra Targeting Exascale - under development
- Aims to replace ScaLAPACK but with support for different accelerators and hybrid computational model (MPI+OpenMP+GPU)
- Currently supports only symmetric eigenproblems
- Based on the two-stage reduction to tridiagonal/bidiagonal form

5

[5] Source: https://icl.bitbucket.io/slate/

An overview of the libraries

Library	Distributed	GPU	Hybrid	Parallel model	Sparsity	Eigenproblem
LAPACK	×	×	×	OpenMP/pthreads	d/b	std/gen nsym/sym
MAGMA	×	yes (multi-GPU)	yes	OpenMP/pthreads/CUDA	d/s/b	std/gen nsym/sym
cuSolver	×	yes (multi-GPU)	×	CUDA	d/s	std/gen sym
EIGEN	×	×	×	OpenMP	d	std/gen nsym/sym
ScaLAPACK	yes	×	×	MPI/BLASC	d	std/gen sym
ELPA	yes	yes (GPU)	×	MPI/OpenMP/CUDA	d	std/gen sym
EigenEXA	yes	×	×	MPI/OpenMP	d	std sym
FEAST	yes	×	×	MPI	d/s/b	std/gen nsym/sym
Intel MKL	yes	yes (Intel GPU)	×	MPI/OpenMP/pthreads	d/b/s	std/gen nsym/sym
Elemental/Hydrogen	yes	yes (Hydrogen)	yes (Hydrogen)	MPI/OpenMP/(CUDA)	d	std sym
SLATE	yes	yes	yes	MPI/OpenMP/CUDA	d	std sym
P_ARPACK	yes	×	×	MPI/BLACS	s	std/gen nysm/sym
LIS	yes	×	×	MPI/OpenMP	d/s	? ?

Legend

Sparsity: $d \rightarrow$ dense, $s \rightarrow$ sparse, $b \rightarrow$ band

Eigenproblem: *std* $\rightarrow$ standard, *gen* $\rightarrow$ generalized, *nsym* $\rightarrow$ non-sysmmetric, *sym* $\rightarrow$ symmetric

Outline

Conclusion

- Large number of distributed memory eigensolvers but only **ELPA** and **SLATE** support GPUs,
- Support only a limited types of eigenproblems,
- The performance is usually compared only with ScaLAPACK $\rightarrow$ does not give a good comparison with other GPU-based libraries.

Future work

- Benchmark of the modern distributed eigensolvers (side-by-side),
- Analyze the scalability and the performance challenges on the current eigensolvers, and predict their behavior on future computing architectures systems,
- Extend and optimized the existing distributed eigensolvers for multi-GPU platforms and for specific eigenproblem types.

Thank you for your attention!

HRZZ
Croatian Science Foundation