Parallel Eigen Solver

Anup Mohan
Hridya Valsaraju
Ronak Vinod Panchal

Why Eigen Values ?

- Widely used in many applications like,
- Quantum mechanics
- Graph theory
- Image processing etc…

DotNetMatrix Library

- We use the DotNetMatrix library for Eigen Value and Eigen vector calculation
- Accepts only Real Matrices
- Computes EigenValues and EigenVectors for Symmetic and Non-Symmetric Matrices
- Capable of computing Imaginary EigenValues and EigenVectors

Symmetric Eigen Value Calculation

- Calculation involves two major steps
  - Tridiagonalization
    - Uses Householder transformation to convert input matrix to tridiagonal form
    - Contributes to 98% of execution time
  - QL Method for Diagonalization
    - Uses QL algorithm to convert the tridiagonal matrix to diagonal form
    - Diagonal elements of the Diagonal matrix are the EigenValues
    - Contributes to 1% of execution time

Non-Symmetric Eigen Value Calculation

- Calculation involves two major steps
  - Hessenberg Matrix Generation
    - Converts the input matrix to Hessenberg form
    - Contributes to 22% of execution time
  - Hessenberg to Real-Schur Decomposition
    - Converts the Hessenberg matrix to Real-schur form
    - Diagonal elements of Real-Schur matrix are the EigenValues
    - Contributes to 75% of execution time

Code Profiling

- Symmetric
  - tred2() : 98%
    two main loops: loop2- 31%
    loop3- 68%

- Non Symmetric
  -hqr2() : 75%
    two main loops: loop2- 87.7%
    loop5- 10.9%
  -orthes() : 22%
    two main loops: loop1- 60.1%
    loop3- 35.0%

Test Setup

- Input sizes of 800x800, 500x500, 100x100
- Intel® Core™ i7-2670QM, 4 Cores, CPU @2.20GHz

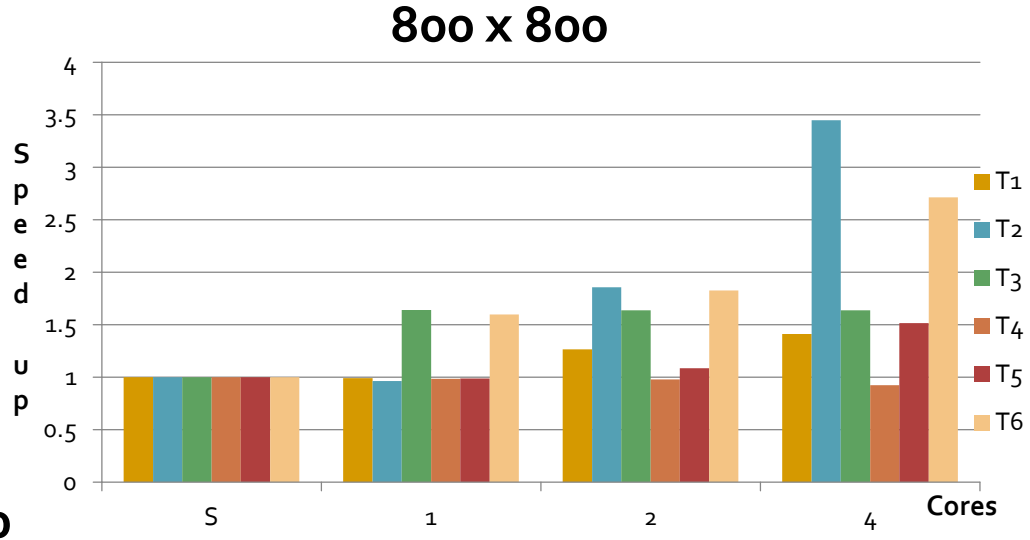
Non-Symmetric – orthes() - Observations

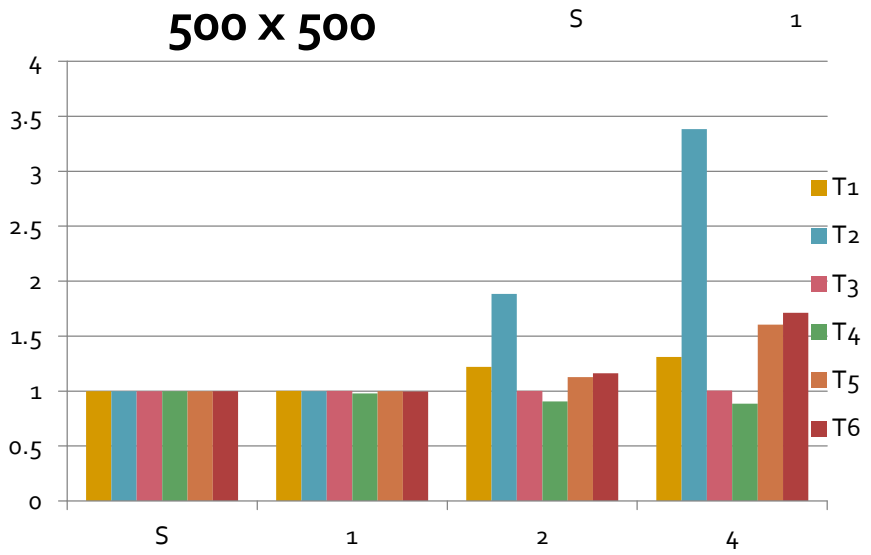
- Contains two main loops

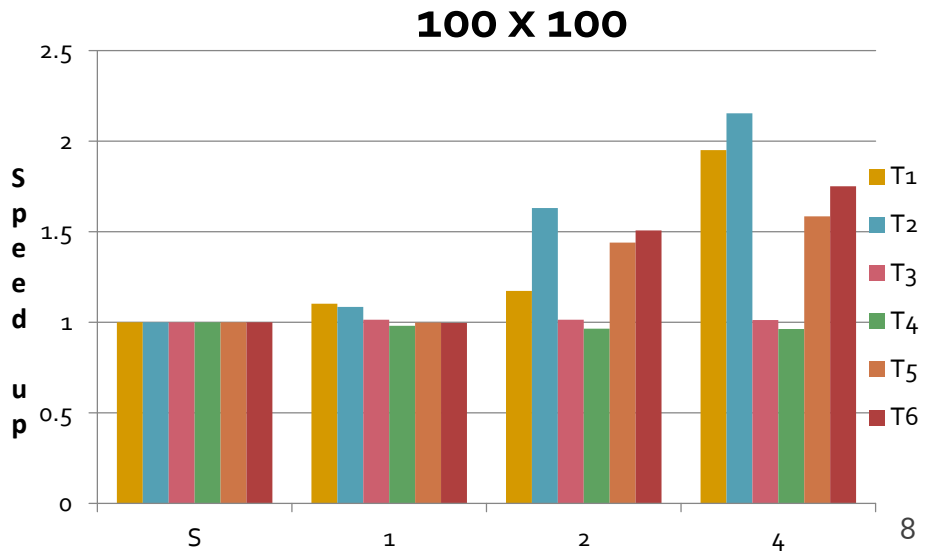
Transformations performed:

- Loop interchange
- Privatization
- Scheduling
- Reduction

Non-Symmetric – orthes() - Observations







Non-Symmetric – hqr2() - Observations

- Contains two main loops

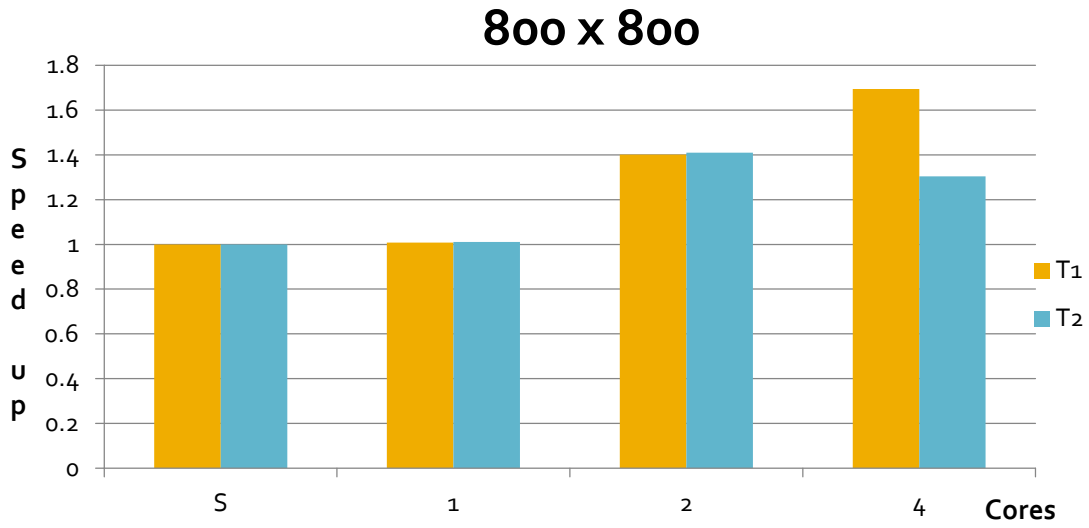
Transformations performed:

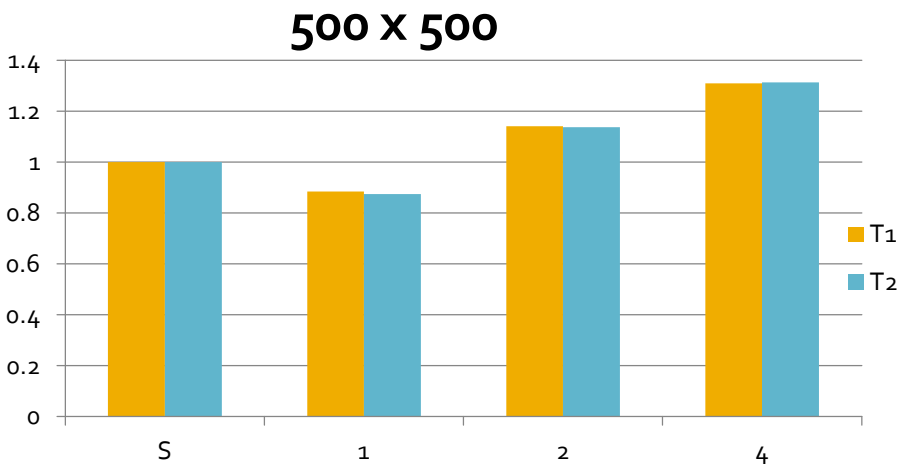
- Reduction
- Privatization

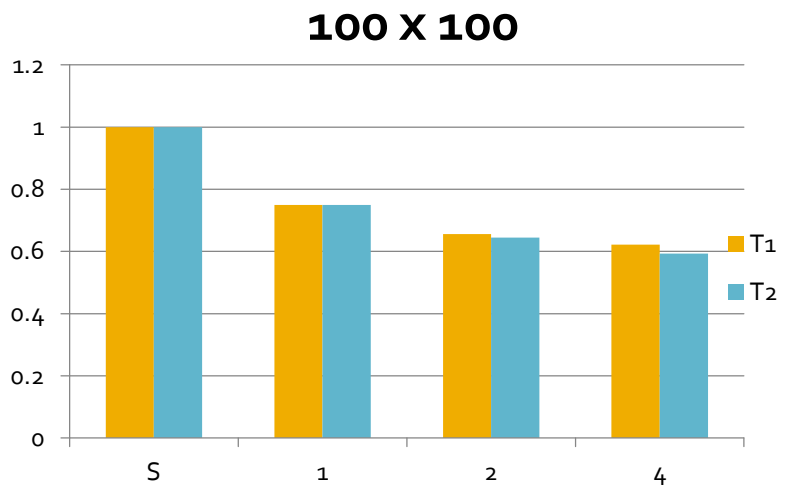
Factors preventing parallelization

- True Dependence
- Break Statements inside conditionals

Non-Symmetric – hqr2() - Observations







Symmetric – tred2() - Observations

- Contains two main loops

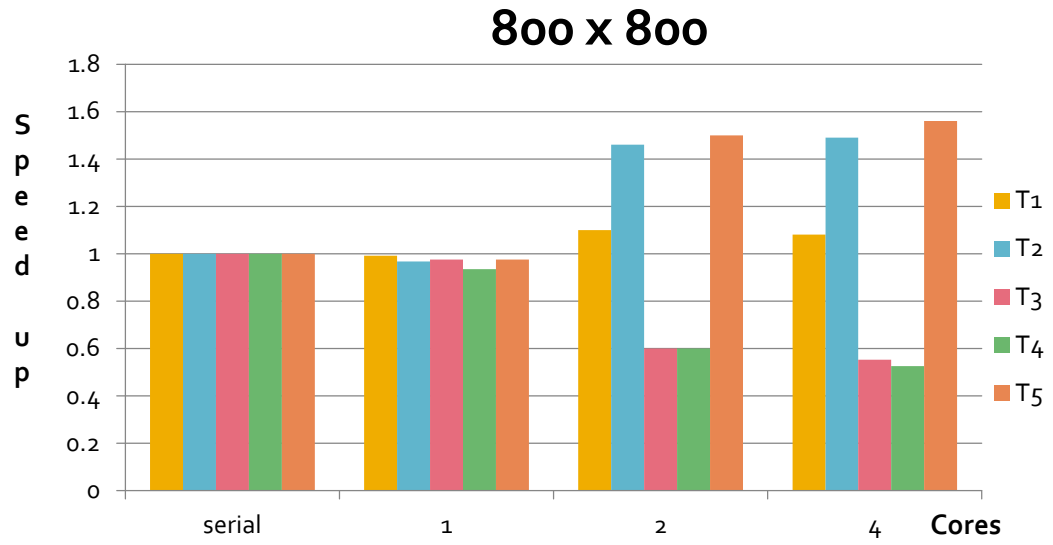
Transformations performed:

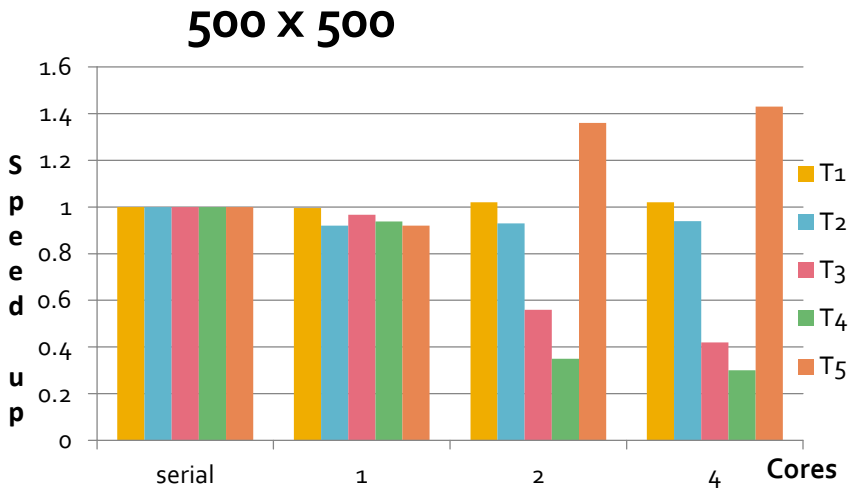
- Reduction
- Privatization
- Scheduling

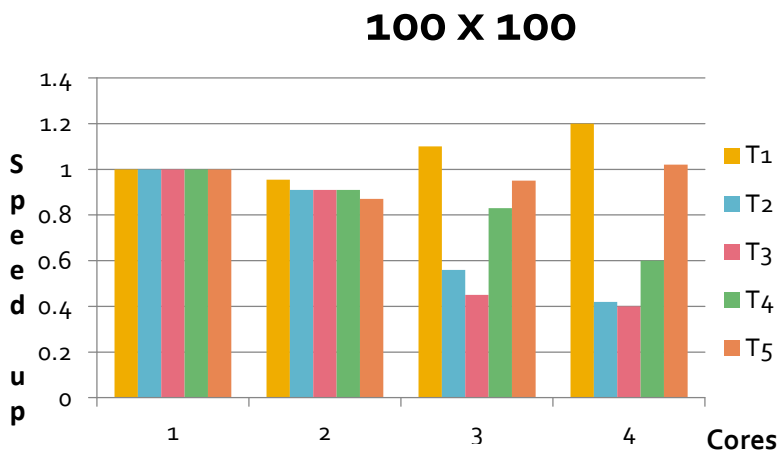
Factors preventing parallelization

- Multiple True Dependencies

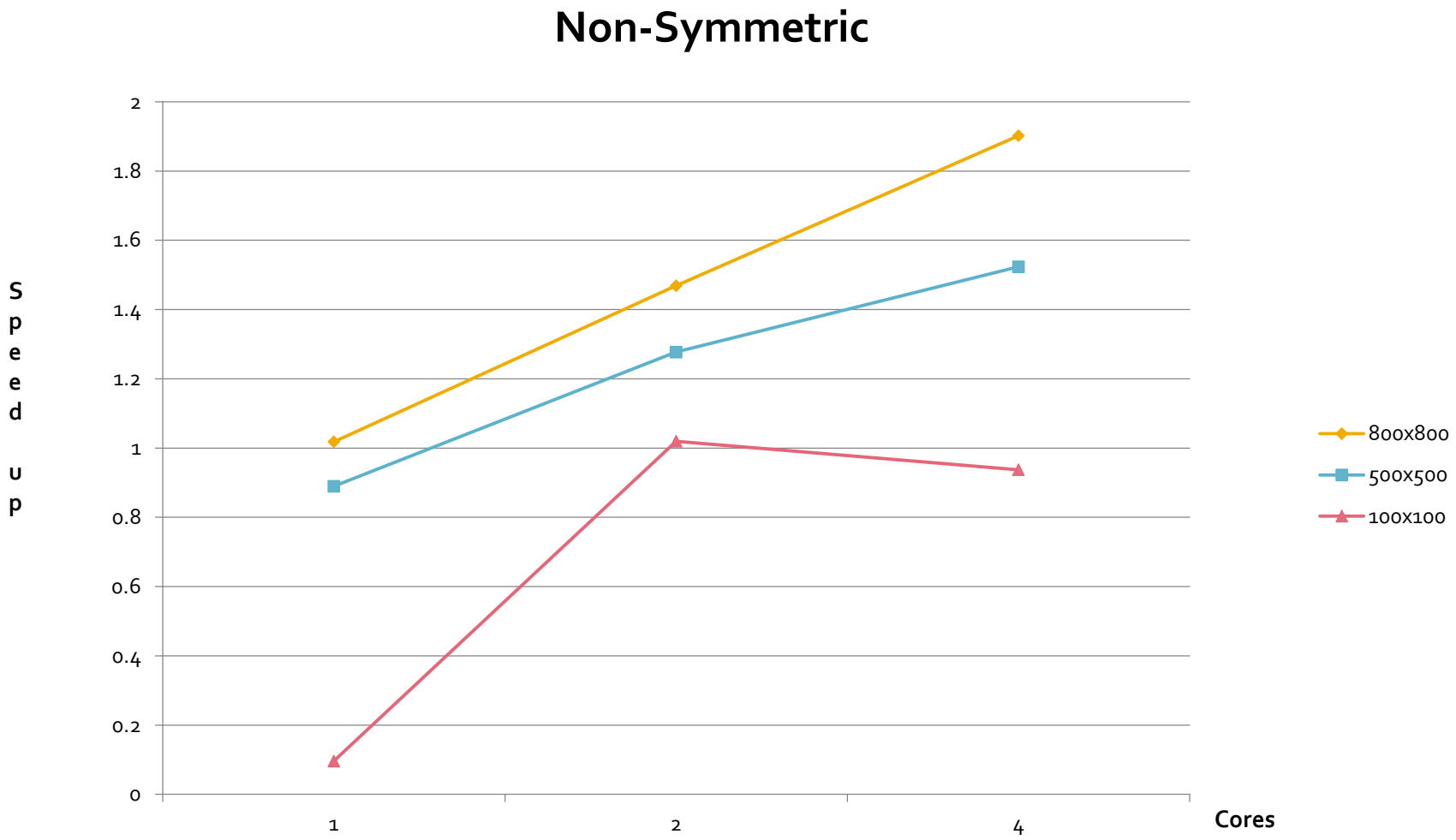
Symmetric – tred2() - Observations



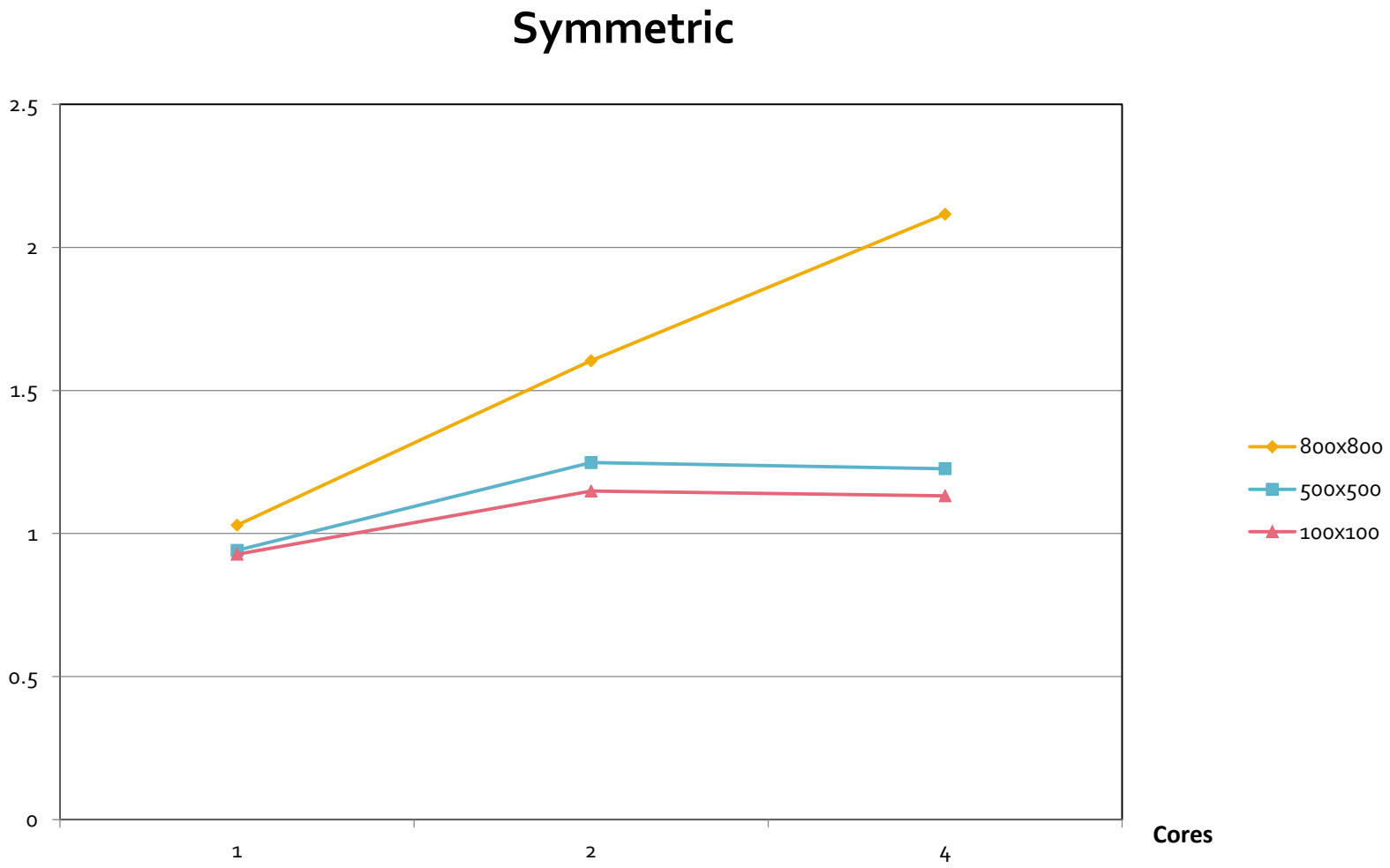




Overall Performance



Overall Performance



Using Cetus

- Used the Cetus 1.4.0 binary release
- Helpful in finding a few reductions which resulted in improved performance
- A few optimizations suggested by Cetus worsened the performance due to over optimization of inner loops

Future Work

- Take data on 8,16 core machines
- Compare the results with Intel MKL benchmarks
- Modifying the algorithm to reduce the dependencies