A Memory-Efficient Algorithm for Large-Scale Symmetric Tridiagonal Eigenvalue Problem on Multi-GPU Systems

Hyunsu Cho and Peter A. Yoon

Trinity College, Hartford, CT, USA

Symmetric Eigenvalue Problem

$$A\mathbf{x} = \lambda\mathbf{x}$$

where A is symmetric

Many interesting applications require eigenvectors

Divide and Conquer

Yields **full spectrum** of eigenvalues and eigenvectors

Is numerically stable

Gives rise to **independent subproblems**

Often faster than $O(n^3)$ due to deflation

Divide and Conquer

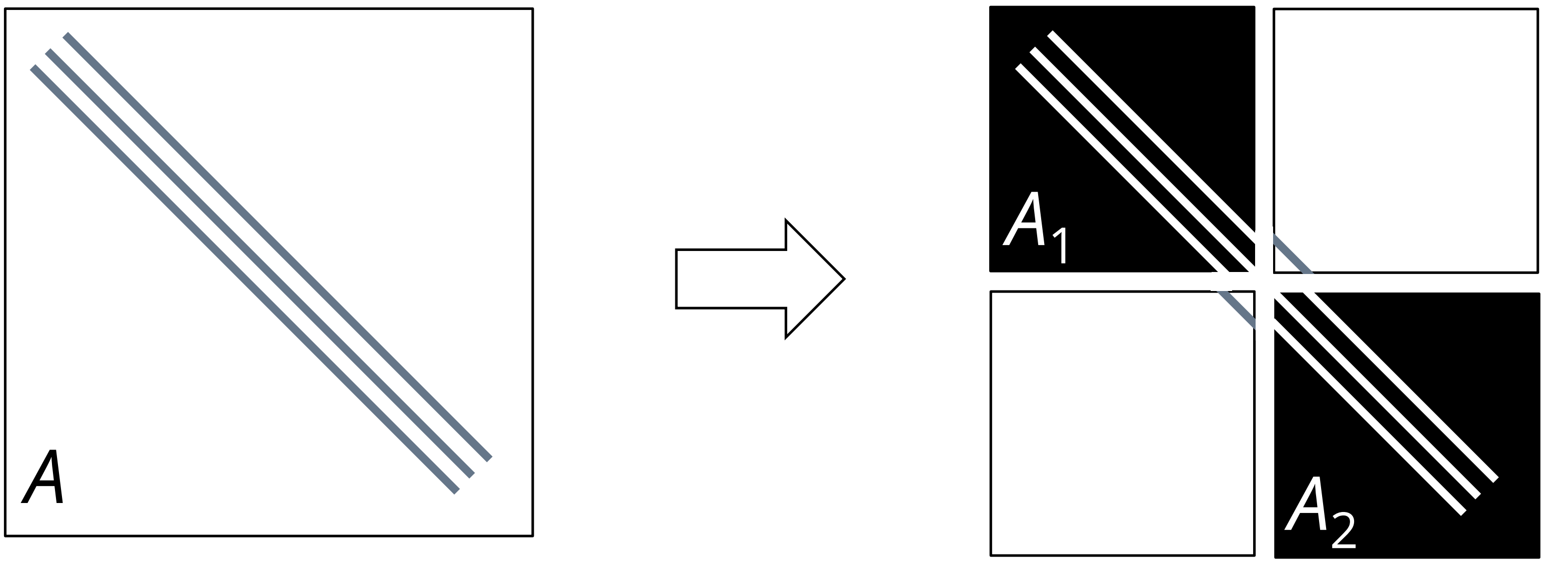
Apply **orthogonal similarity transformation** to reduce *A* to tridiagonal form

$$Q^T A Q = A'$$

where
A' is symmetric tridiagonal
and *Q* is orthogonal

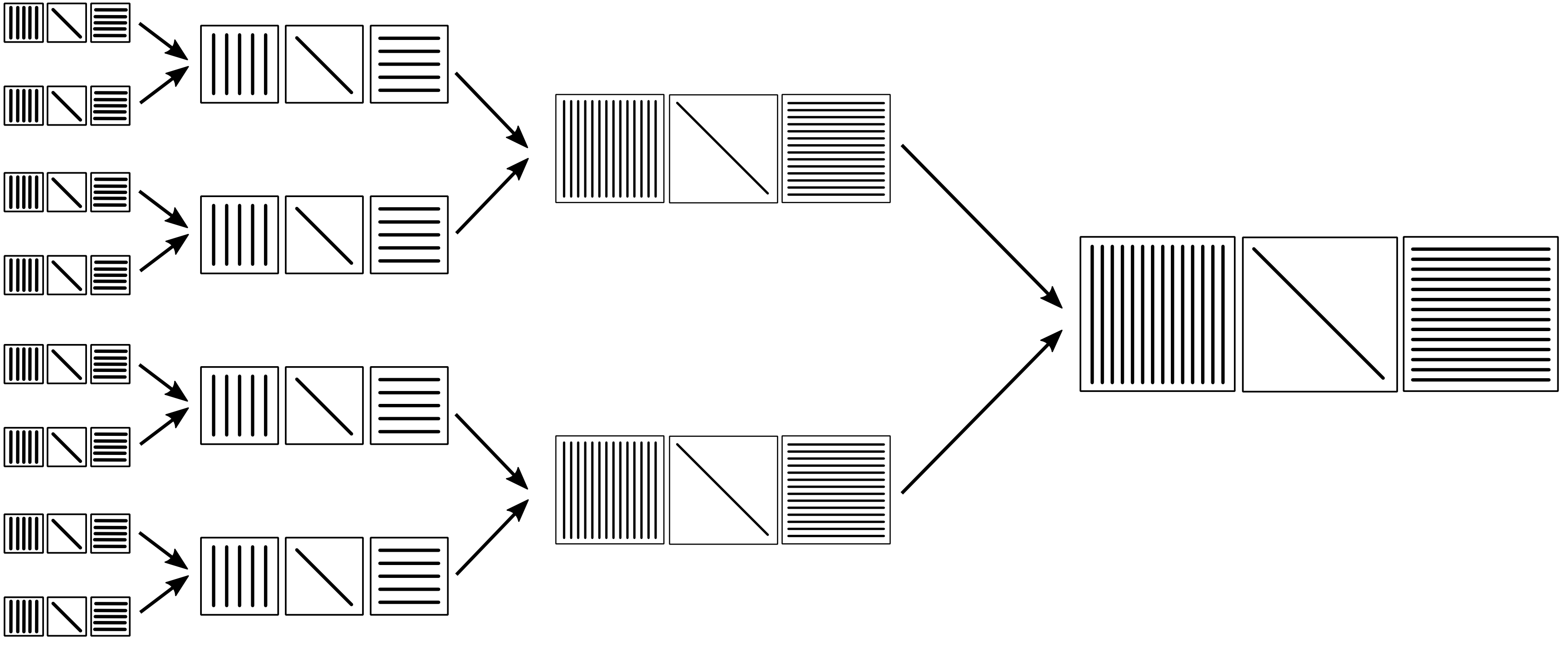
Existing work on single-node, multi-GPU:
MAGMA (UTK)

Divide and Conquer



- Solve subproblems
- Merge solutions
- Repair

Divide and Conquer



Merging solutions

Suppose

$$A = \left[\begin{array}{c|c} A_1 & \\ \hline & A_2 \end{array}\right] + \left[\begin{array}{c|c} \quad b_m & b_m \quad \\ \hline \quad b_m & b_m \quad \end{array}\right]$$

where

$$A_1 = Q_1 D_1 Q_1^T \qquad \text{(subproblem \#1)}$$

$$A_2 = Q_2 D_2 Q_2^T \qquad \text{(subproblem \#2)}$$

Merging solutions

Then

$$A = \quad QDQ^T \quad + \left[\begin{array}{c|c} b_m & b_m \\ \hline b_m & b_m \end{array}\right]$$

where

$$Q = \left[\begin{array}{c|c} Q_1 & \\ \hline & Q_2 \end{array}\right] \quad \text{and} \quad D = \left[\begin{array}{c|c} D_1 & \\ \hline & D_2 \end{array}\right]$$

Merging solutions

Rank-one modifier

$$H = b_m \left[\begin{array}{c} \mathbf{e}_m \\ \hline \mathbf{e}_1 \end{array}\right] \left[\begin{array}{c} \mathbf{e}_m \\ \hline \mathbf{e}_1 \end{array}\right]^T$$

Then

$$A = \quad QDQ^T \quad + \left[\begin{array}{cc|cc} & & & \\ & b_m & b_m & \\ \hline & b_m & b_m & \\ & & & \end{array}\right]$$

where

$$Q = \left[\begin{array}{c|c} Q_1 & \\ \hline & Q_2 \end{array}\right] \quad \text{and} \quad D = \left[\begin{array}{c|c} D_1 & \\ \hline & D_2 \end{array}\right]$$

Rank-one update

$$H = b_m \left[\frac{\mathbf{e}_m}{\mathbf{e}_1} \right] \left[\frac{\mathbf{e}_m}{\mathbf{e}_1} \right]^T$$

$$\begin{aligned} A &= QDQ^T + H \\ &= Q(D + b_m \mathbf{z}\mathbf{z}^T)Q^T \end{aligned}$$

where

$$\mathbf{z} = Q^T \left[\frac{\mathbf{e}_m}{\mathbf{e}_1} \right] = \begin{bmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{bmatrix}$$

Rank-one update

$$H = b_m \left[\frac{\mathbf{e}_m}{\mathbf{e}_1}\right]\left[\frac{\mathbf{e}_m}{\mathbf{e}_1}\right]^T$$

$$\begin{aligned} A &= QDQ^T + H \\ &= Q(D + b_m \mathbf{z}\mathbf{z}^T)Q^T \end{aligned}$$

where

$$\mathbf{z} = Q^T \left[\frac{\mathbf{e}_m}{\mathbf{e}_1}\right] = \begin{bmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{bmatrix}$$

Need eigen-decomposition of inner system

Decompose $D + b_m \mathbf{z}\mathbf{z}^T$

1. Sort entries in *D*; permute z likewise
2. Filter some entries in *D* and z via deflation (next slide)

Decompose $D + b_m \mathbf{z}\mathbf{z}^T$

1. Sort entries in *D*; permute z likewise
2. Filter some entries in *D* and z via deflation (next slide)
3. Compute all roots of the **secular equation** [1]

$$1 + b_m \sum_{i=1}^{n} \frac{d_i^2}{z_i - \lambda} = 0,$$

   giving the *m* eigenvalues.
4. Compute corresponding eigenvectors stably [2]

[1] Li 1994
[2] Gu & Eisenstat 1994

Decompose $D + b_m \mathbf{z}\mathbf{z}^T$

1. Sort entries in *D*; permute z likewise
2. Filter some entries in *D* and z via deflation (next slide)
3. Compute all roots of the **secular equation** [1]

$$1 + b_m \sum_{i=1}^{n} \frac{d_i^2}{z_i - \lambda} = 0,$$

   giving the *m* eigenvalues.
4. Compute corresponding eigenvectors stably [2]
5. Multiply each eigenvector by *Q*
   Recall: $A = Q(D + b_m \mathbf{z}\mathbf{z}^T)Q^T$

[1] Li 1994
[2] Gu & Eisenstat 1994

Deflation

Recall:

$$D = \left[\begin{array}{c|c} D_1 & \\ \hline & D_2 \end{array}\right]$$

Entries of *D* are eigenvalues of two subproblems

If two entries are nearly identical, we throw one away

Fewer columns when multiplying eigenvectors by *Q*

Same thing for small entries in z

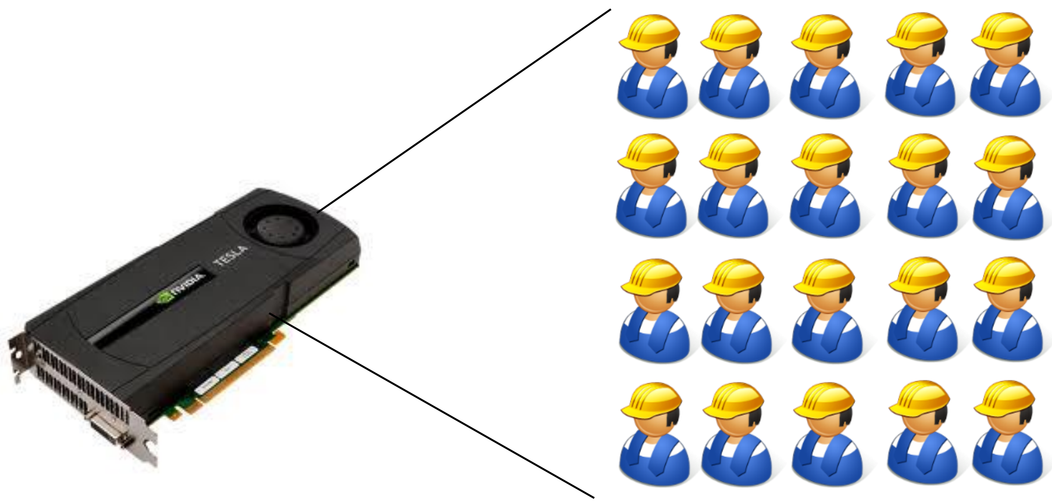
Reduce work complexity to $O(n^{2.3})$

GPU computing

General-purpose computation on GPUs

Bulk parallelism w/ many small threads

Cost effective; widely available



Mapping work to GPU

1. Sort entries in *D*; permute z likewise
2. Filter some entries in D and z via deflation
3. Compute all roots of the secular equation, giving the *m* eigenvalues.
4. Compute corresponding eigenvectors stably
5. Multiply each eigenvector by *Q*
   → Done in bulk via DGEMM

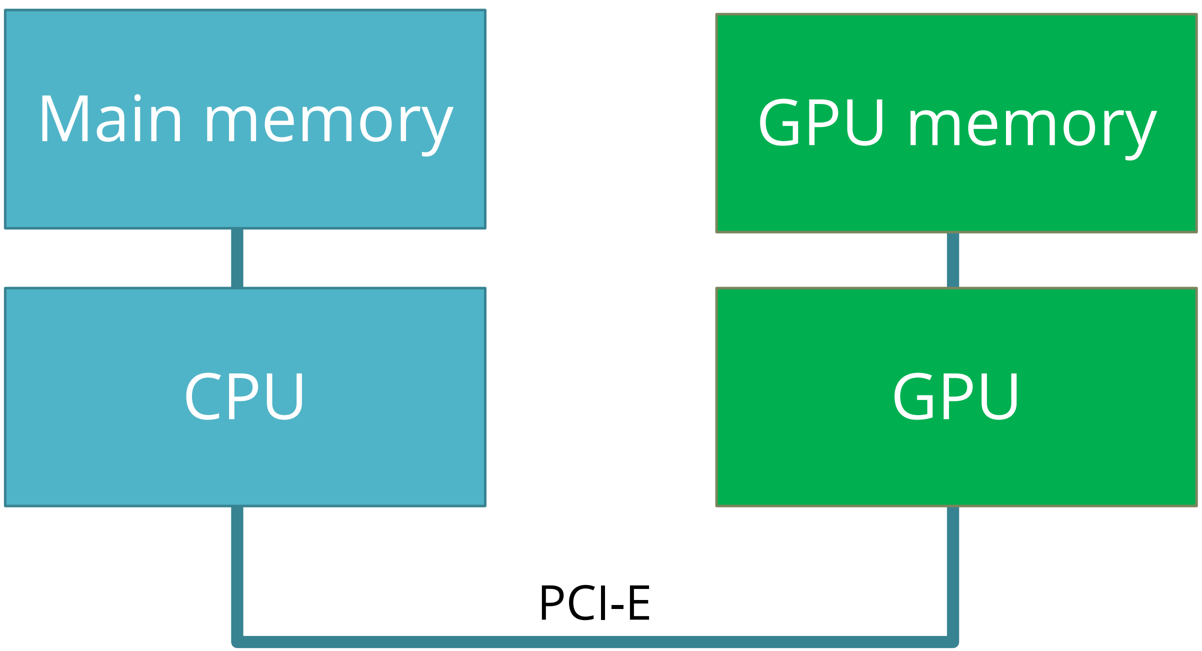
Parallel but
not as work-intense

GPU memory

High-bandwidth dedicated memory

Separate from main memory

Limited in size



Memory requirement

Eigenvectors are dense
$\longrightarrow O(n^2)$ storage

Intermediate workspace: eigenvectors of inner system

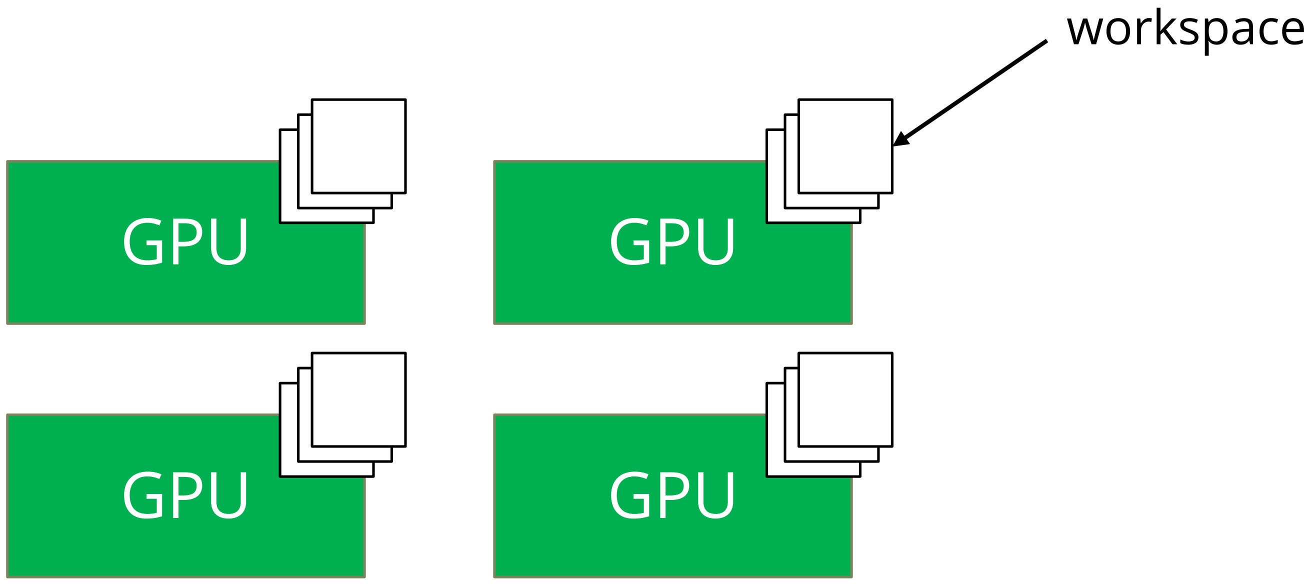
Matrix dimension	Memory required
8192	1.5 GB
16384	5.8 GB
32768	23.4 GB
36000	28.2 GB
50000	54.4 GB

Our contribution

Overcome limitation in GPU memory
while retaining adequate performance

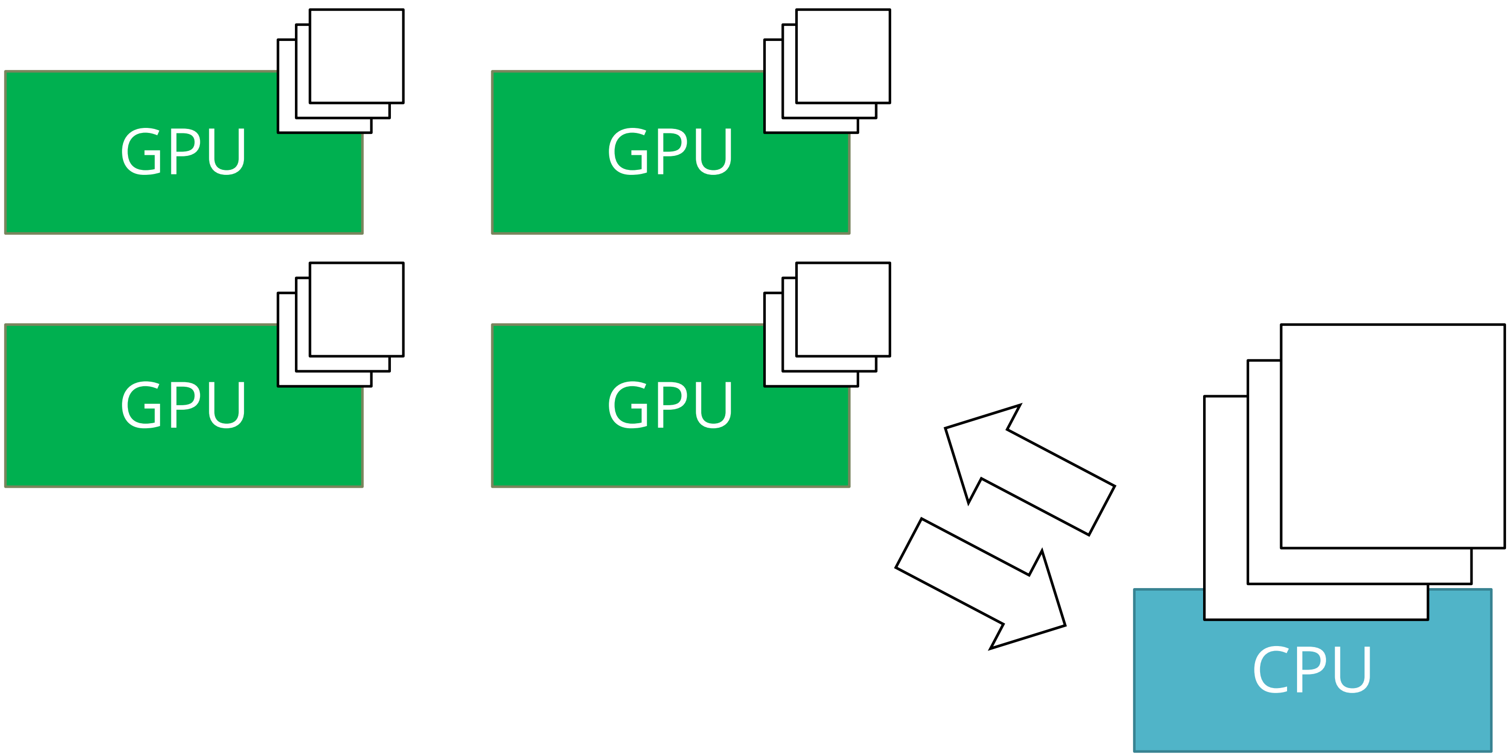
Strategies

1. Use multiple GPUs



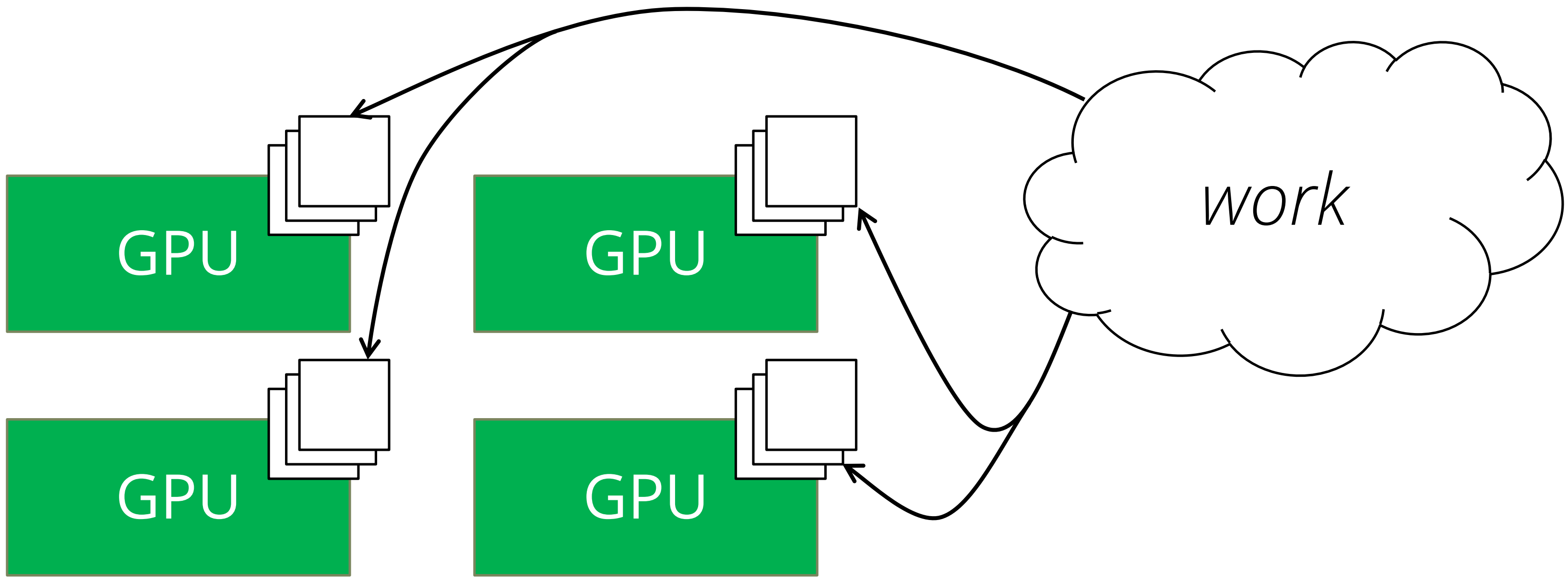
Strategies

2. Keep most of workspace in main memory (**out-of-core** approach)



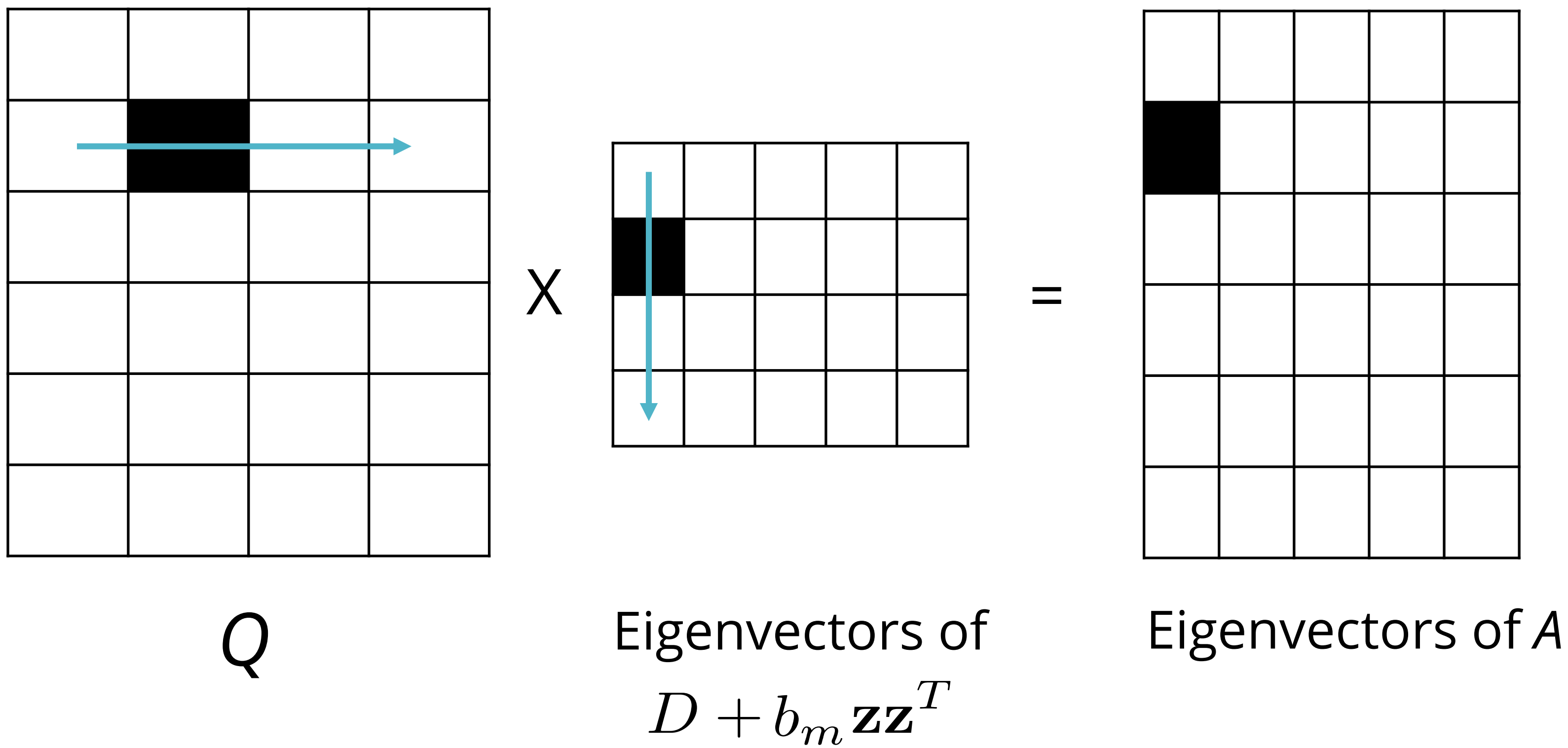
Strategies

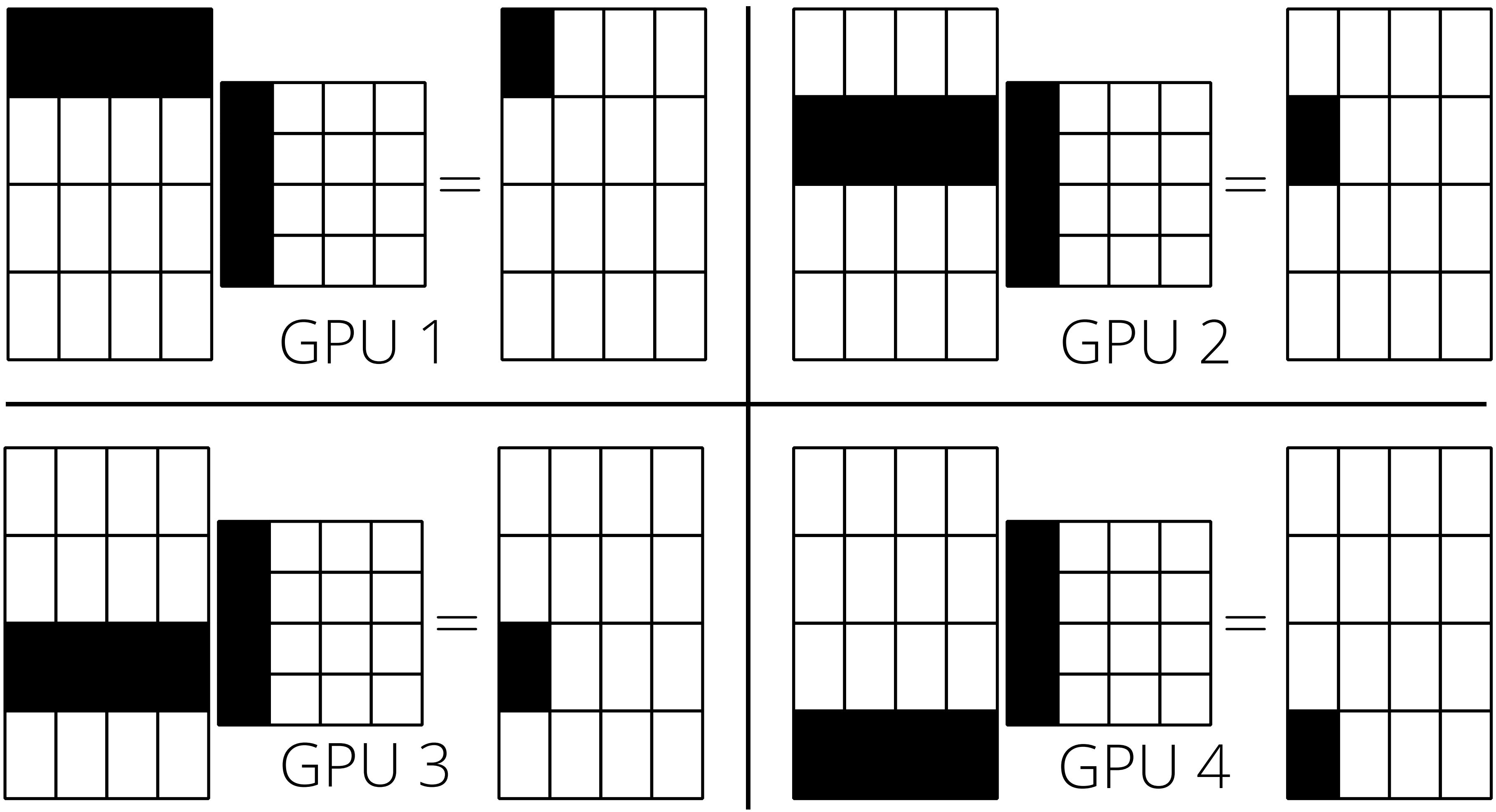
3. **Shape work** to fit GPU workspaces



Block matrix multiplication

Use a fine partition to fit submatrices into GPU memory



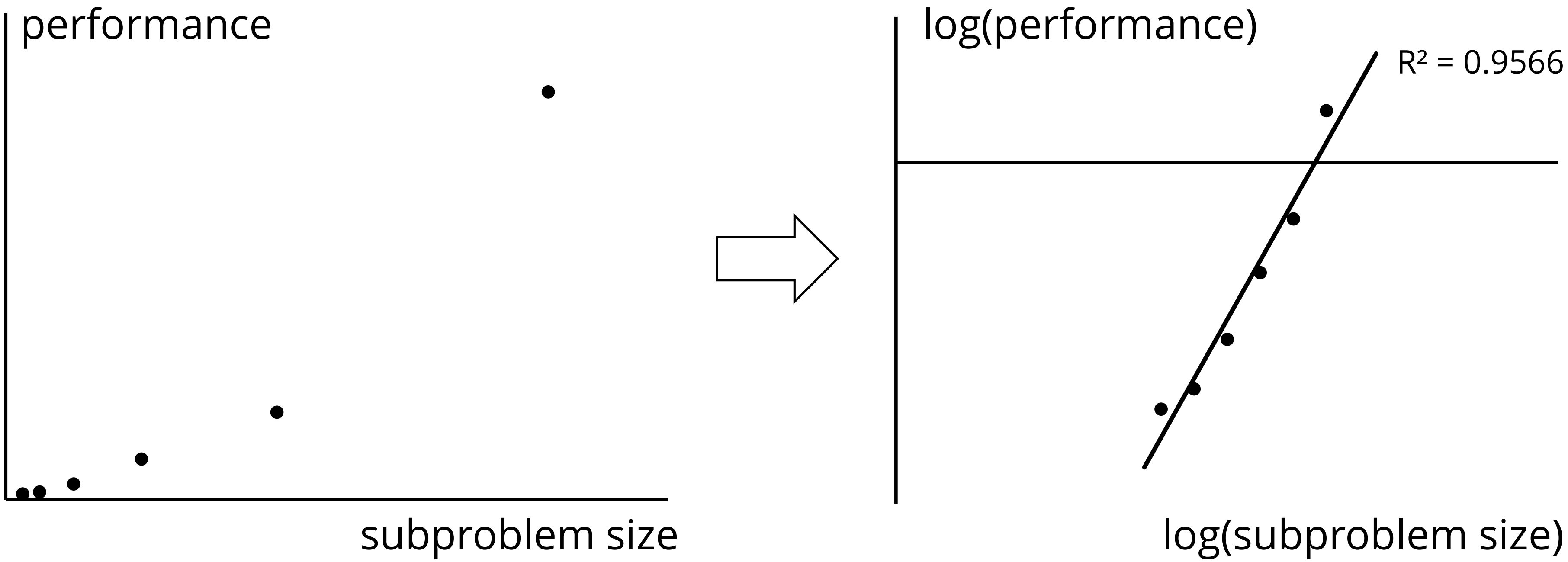

GPU 1
GPU 2
GPU 3
GPU 4

Hybrid computation

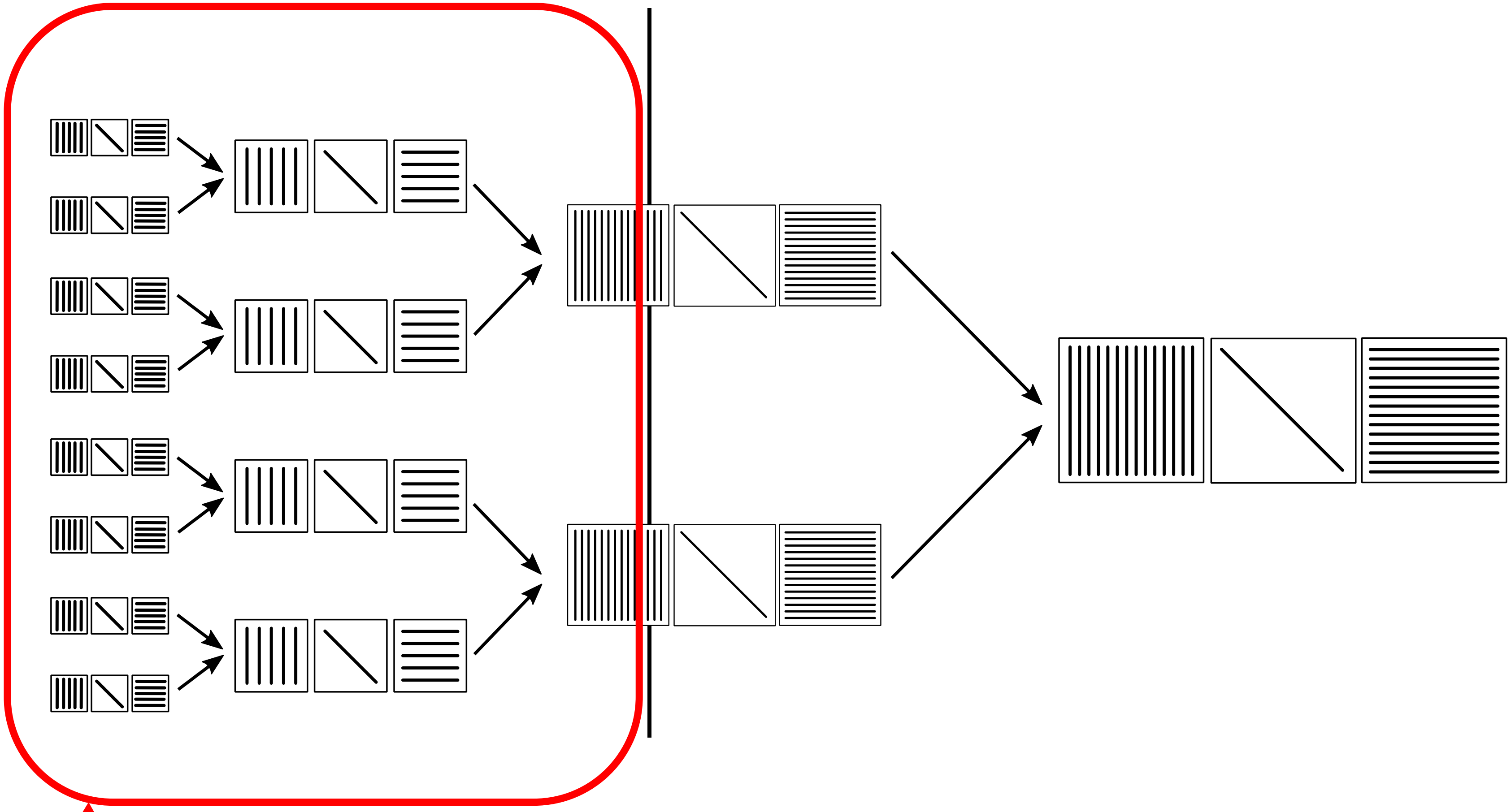
Allocate subproblems to both GPUs and CPUs

Model performance as a power function

Profiler fits parameters using least-squares

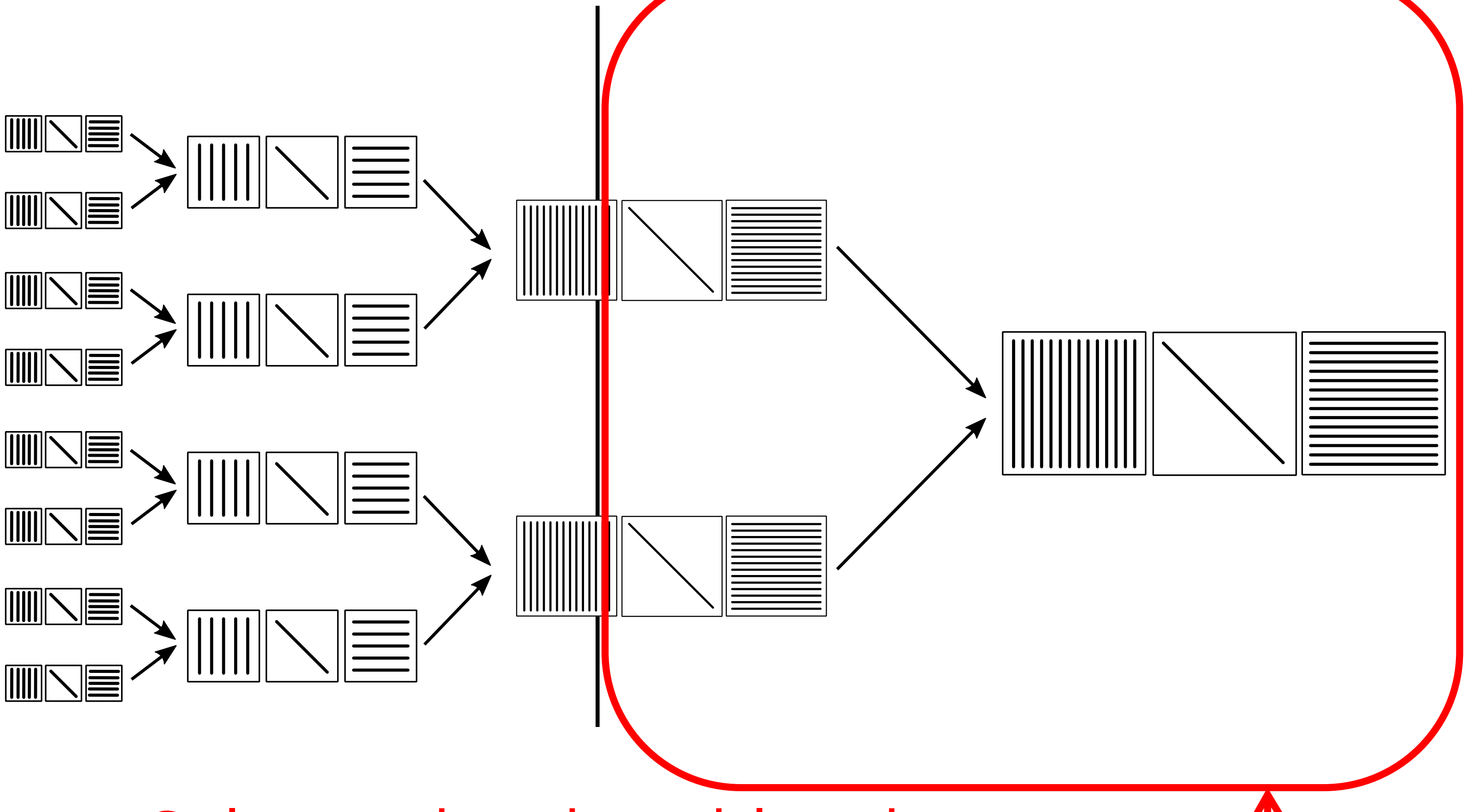


Hybrid computation



Solve many subproblems in parallel

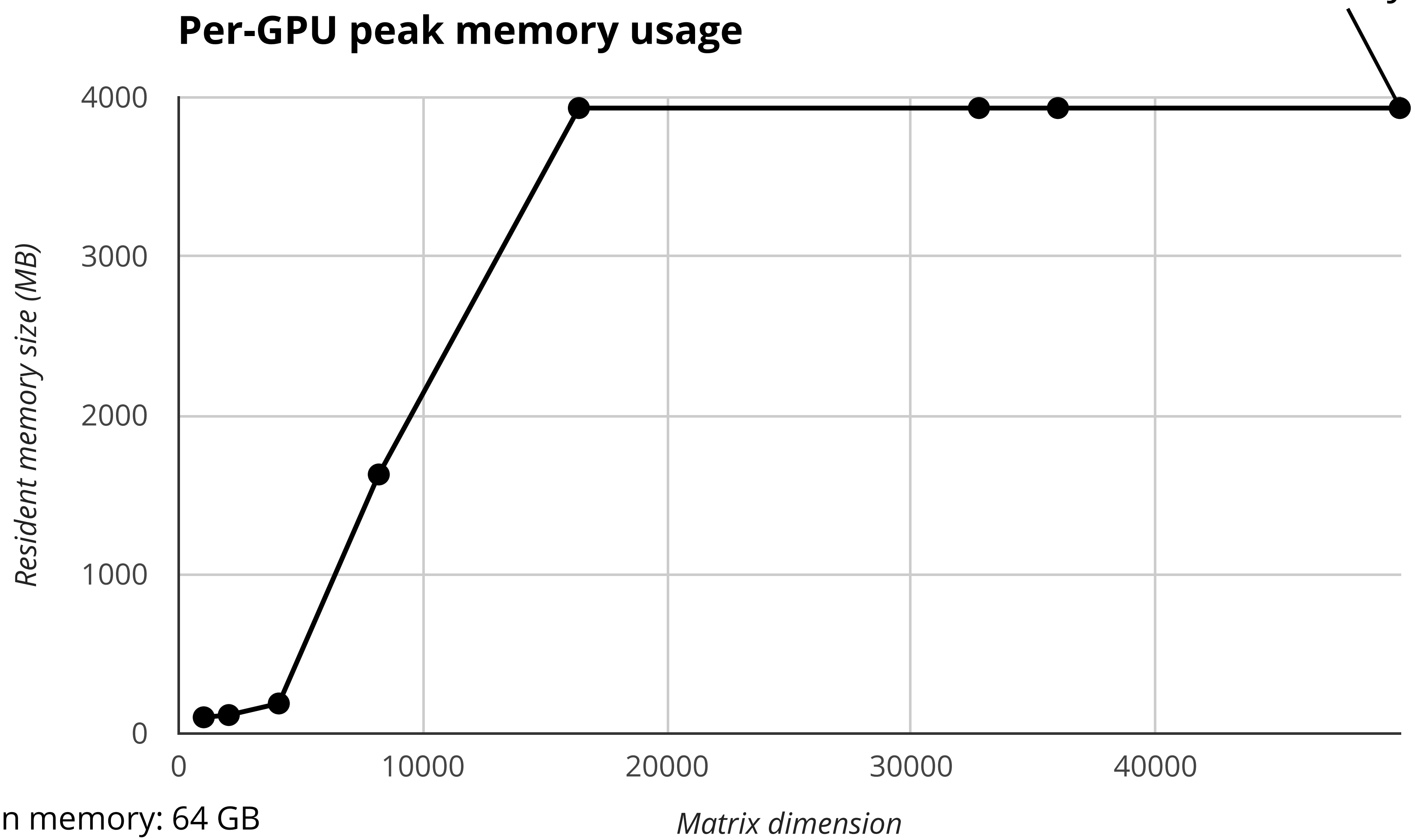
Hybrid computation



Solve each subproblem by parts

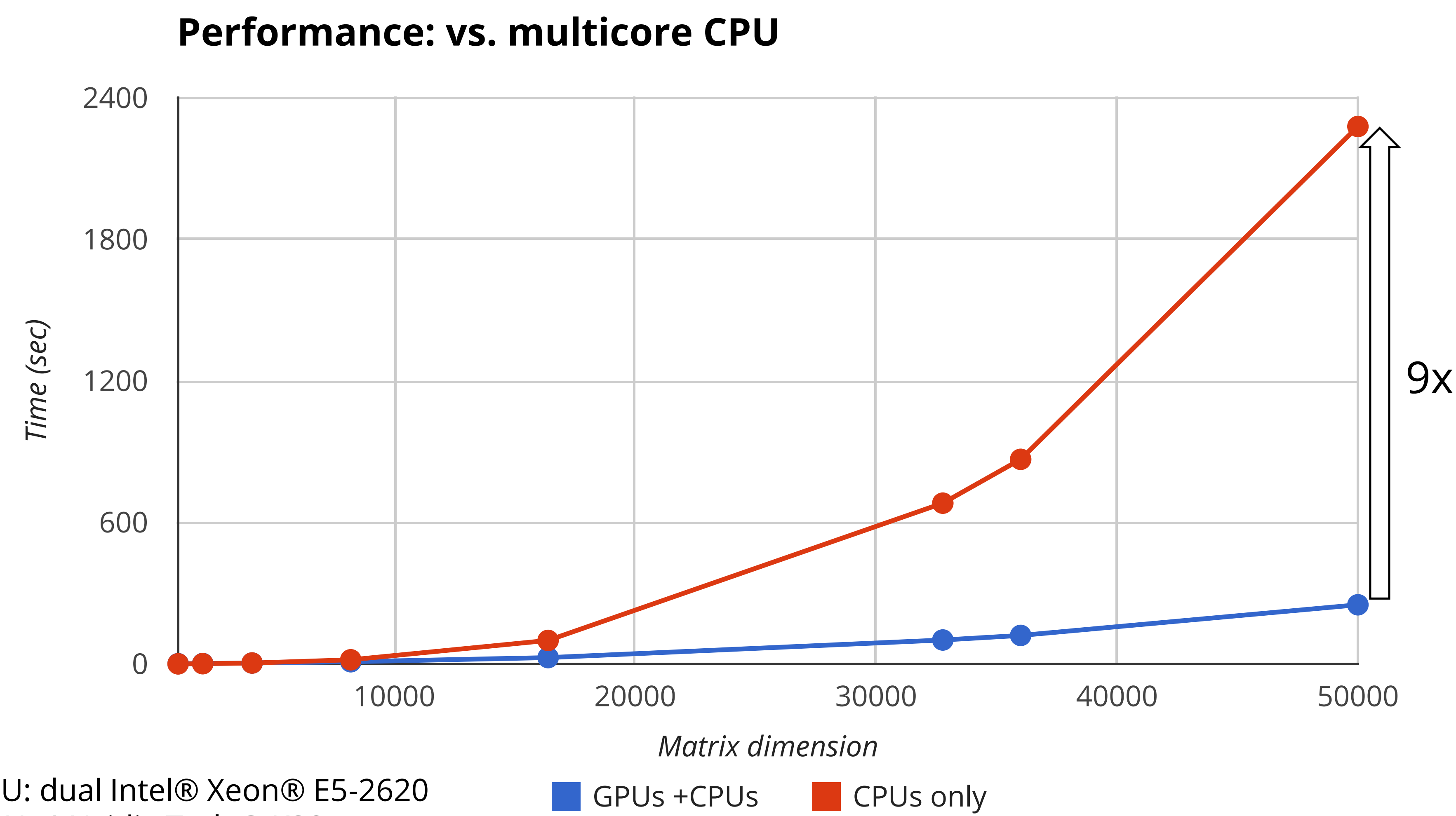
Results

Scales to 50k * 50k matrix
With 4 GB of GPU memory



Main memory: 64 GB
GPU memory: 5 GB per GPU

Results



CPU: dual Intel® Xeon® E5-2620
GPU: 4 Nvidia Tesla® K20c

Conclusion

Out-of-core approach overcomes memory limitation on the GPU

Hybrid computation with profiling delivers reasonable performance

Acknowledgment

Trinity College, Student Research Program

Nvidia Corporation, CUDA Teaching Center Program

Any questions?