An Out-of-core Implementation of Block Cholesky Decomposition on A Multi-GPU System

Lin Cheng, Hyunsu Cho, Peter Yoon, Jiajia Zhao
Trinity College, Hartford, CT

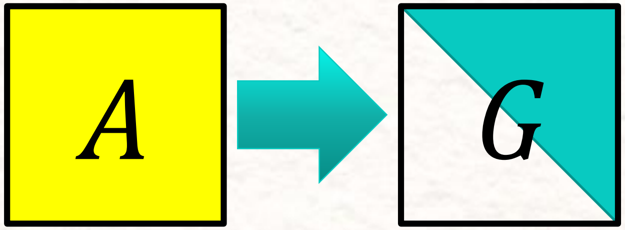
Outline

- Block Cholesky decomposition
- Multi-GPU system
- Out-of-core implementation
- Inter-device communication
- Extension to disk I/O
- Performance

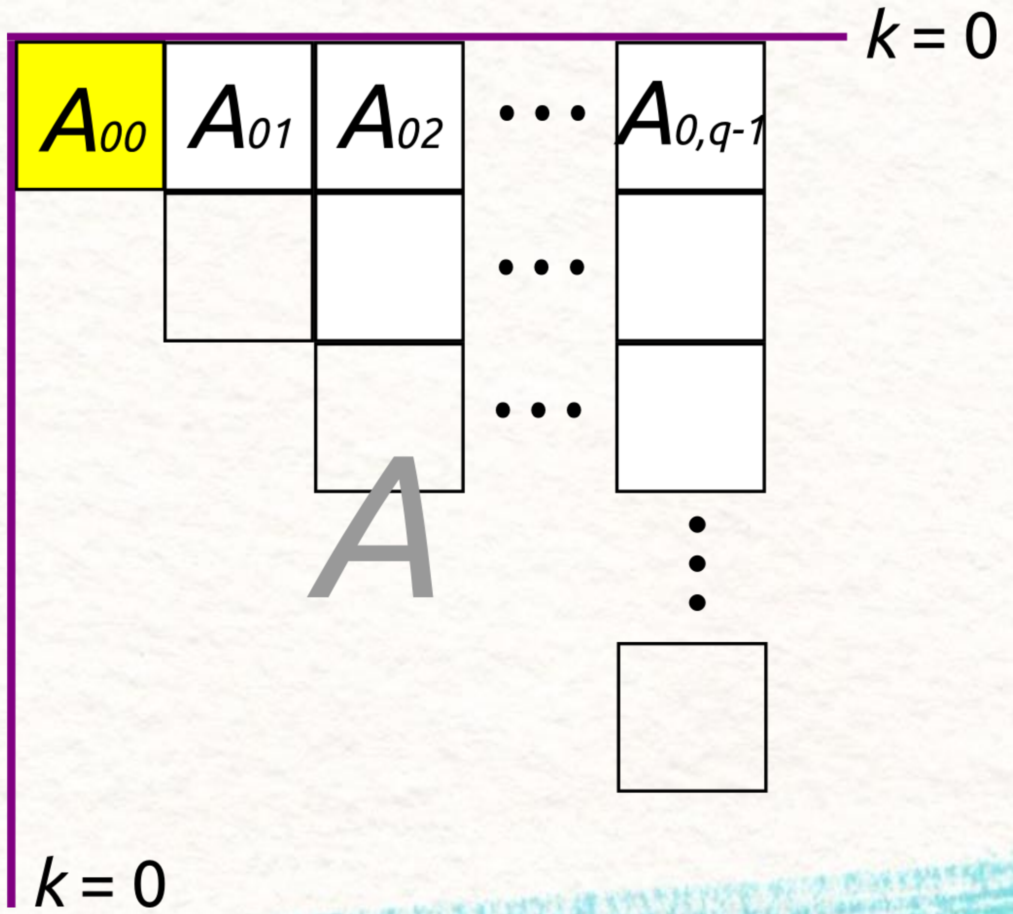
Block Cholesky decomposition

- Widely used matrix factorization
- Dense linear algebra routine
- Diagonally dominant, numerically stable
- Block version
  - Cache-aware block size

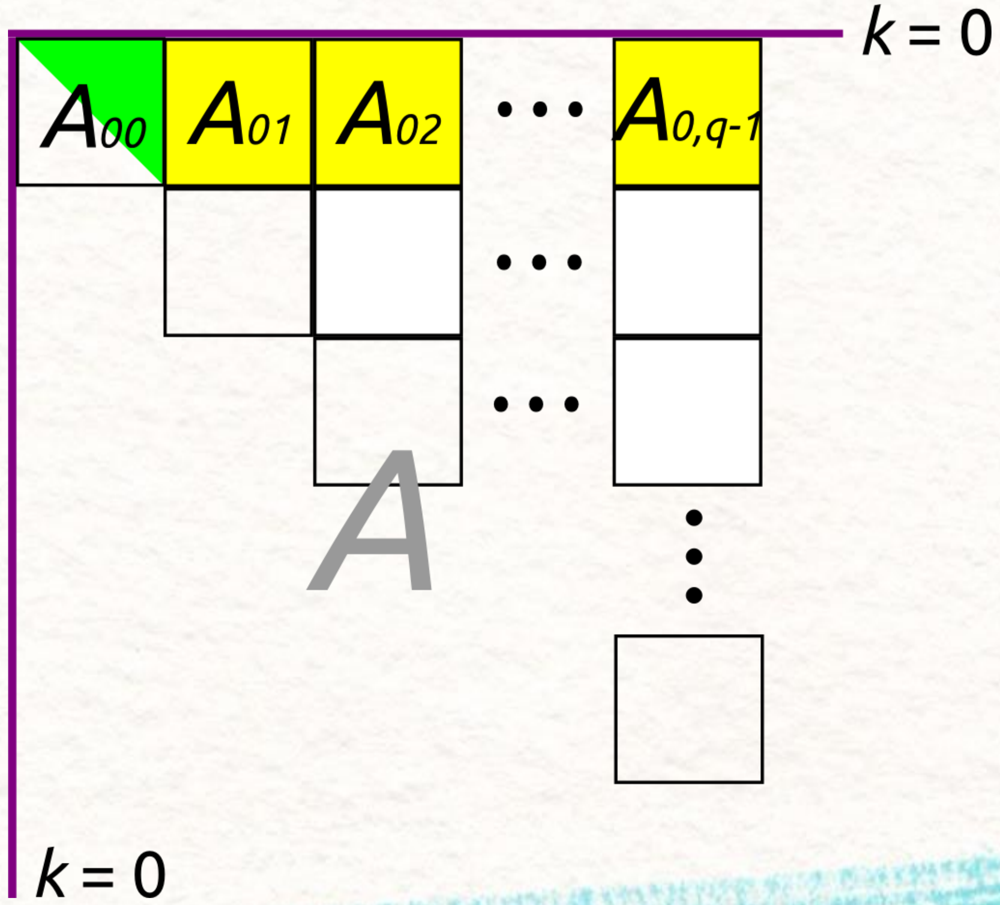
$$A = G^T G$$



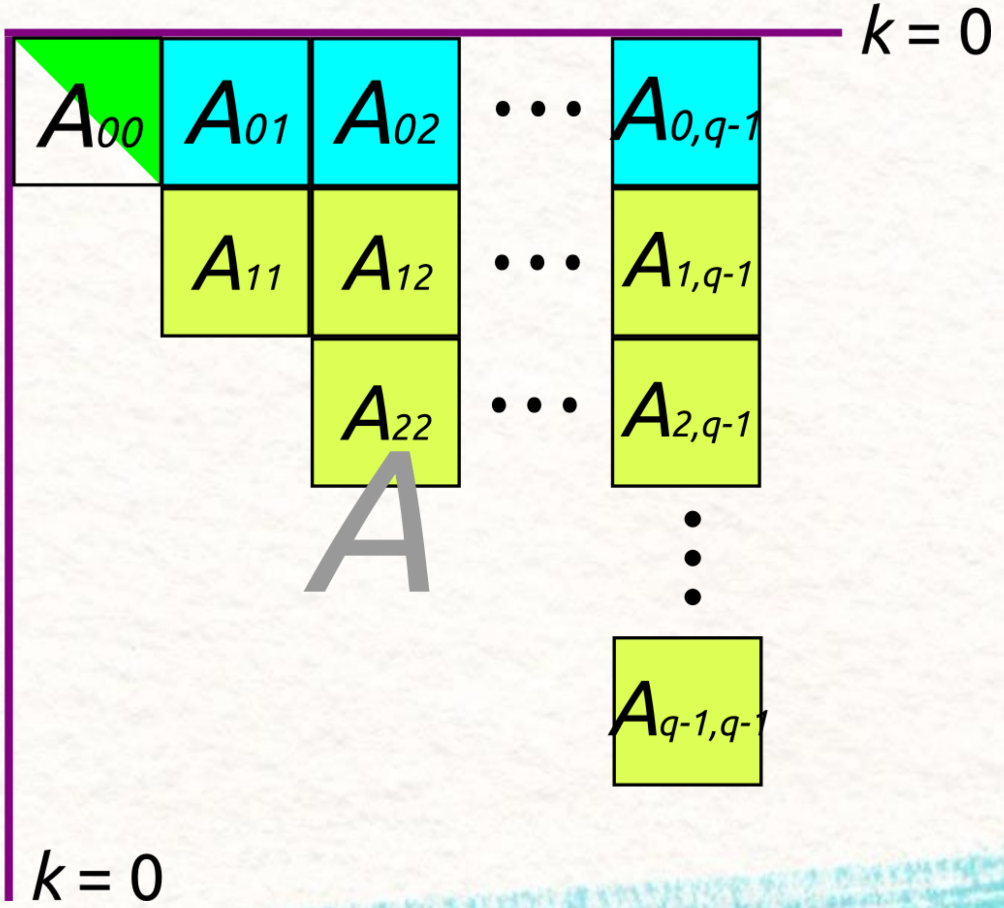
Procedure: Phase I



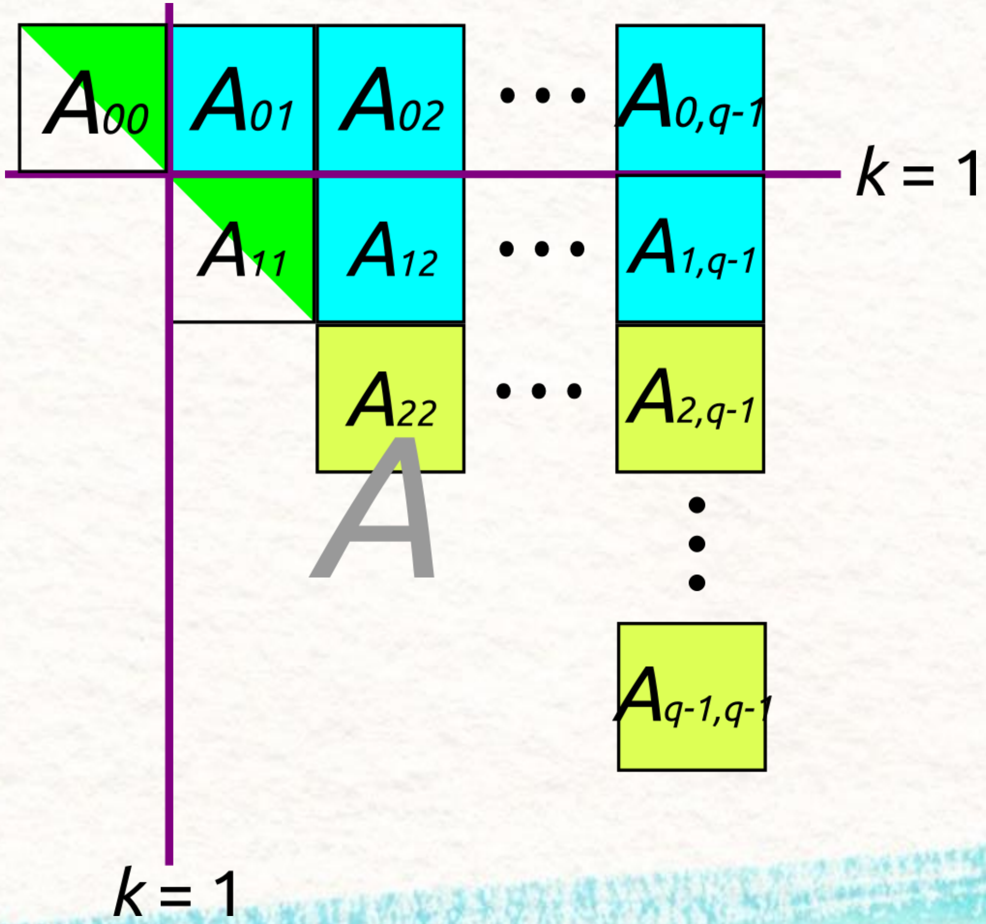
Procedure: Phase II



Procedure: Phase III

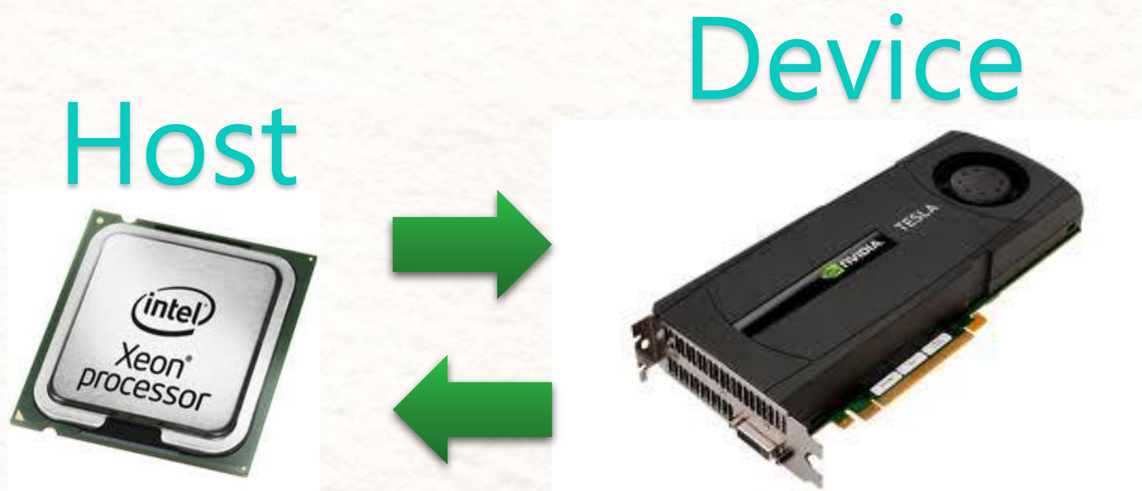


Procedure: Repeat



General Purpose GPU (GPGPU)

- Cost-efficient parallel platform
- Many-core approach

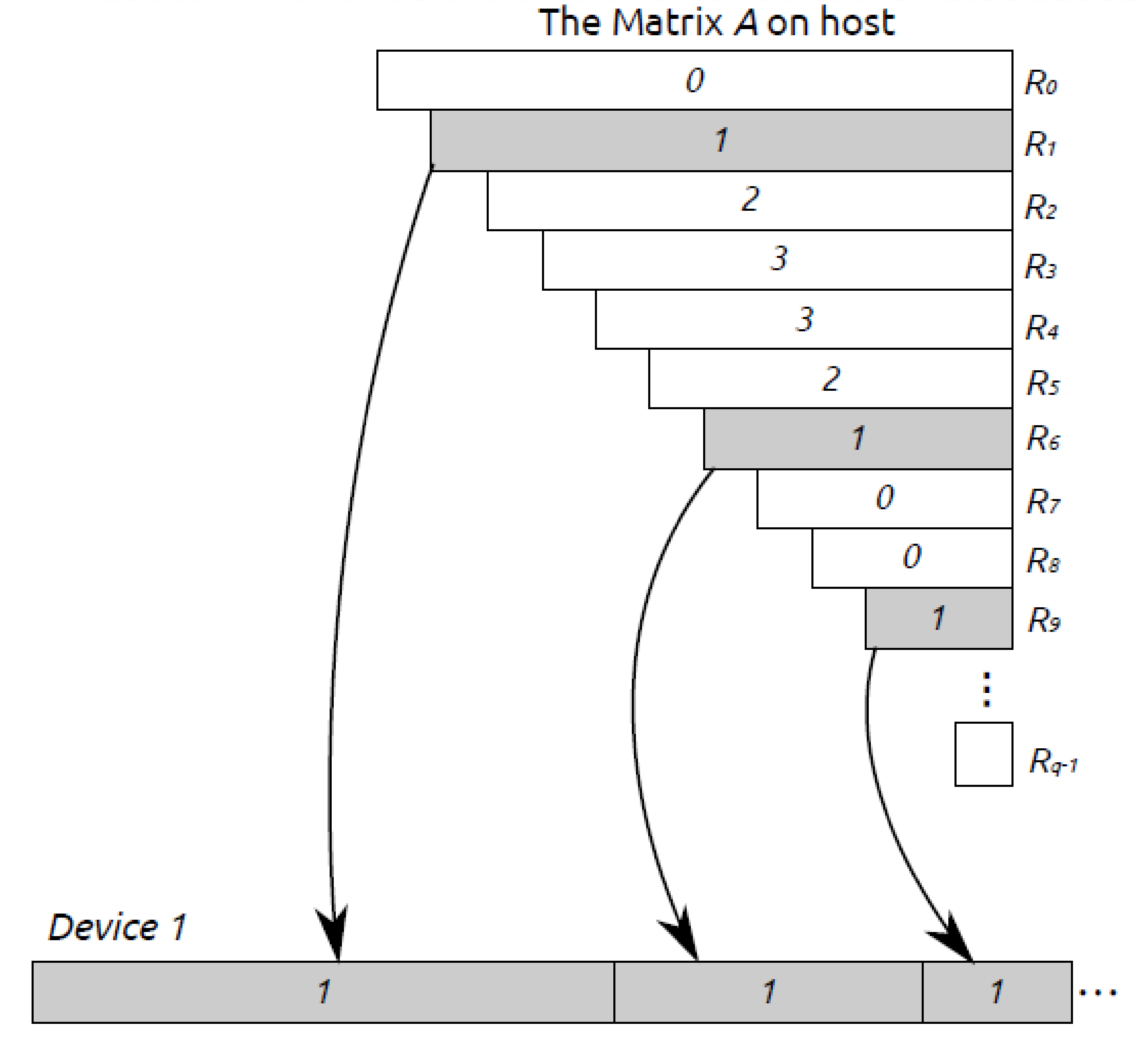


Multi-GPU system

- GPU devices maintain separate memory & kernel invocations
- Coordination becomes a significant issue
- Memory transfer between devices is costly
- Load-balancing is necessary to achieve high performance

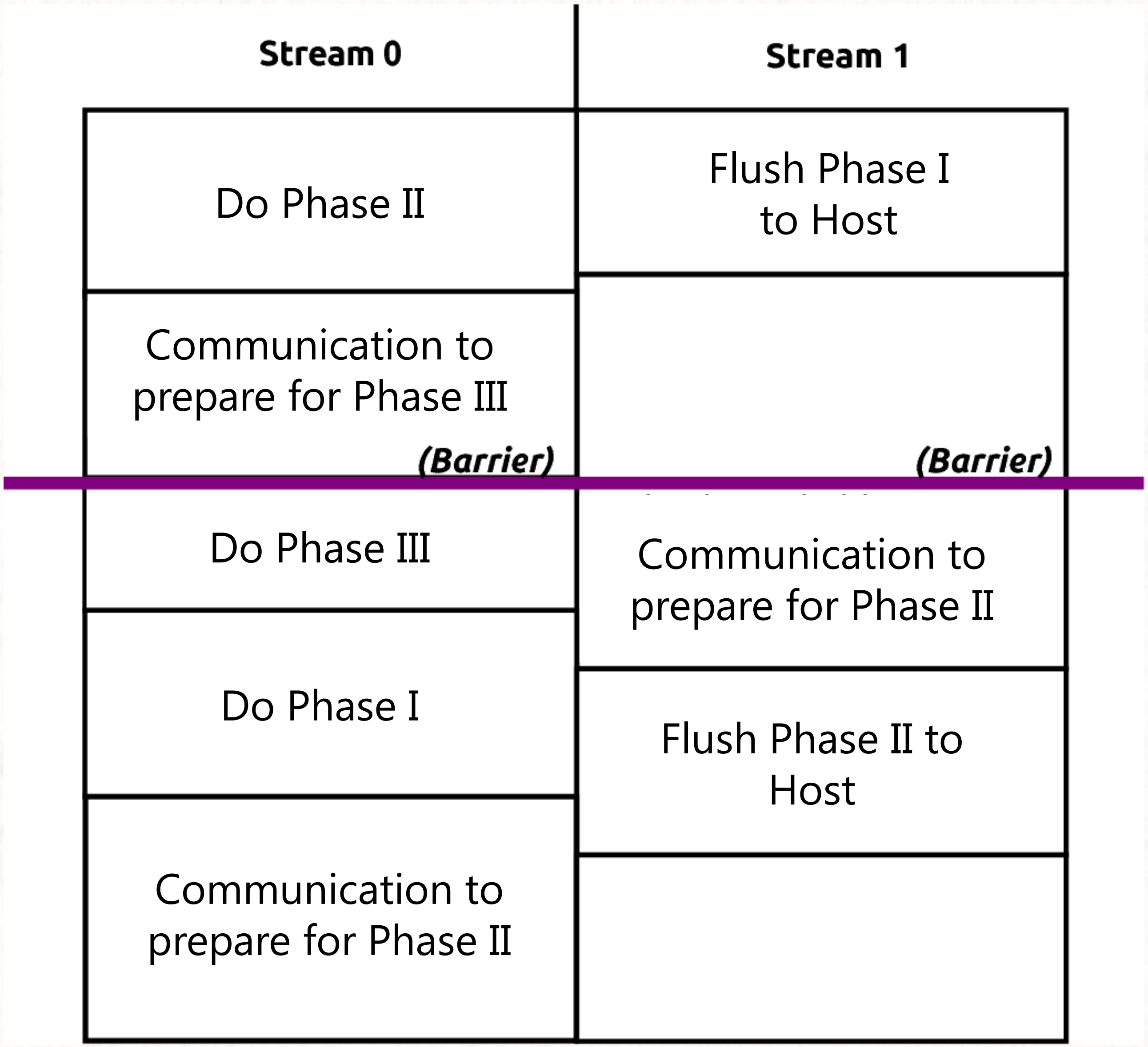
Out-of-core implementation

- GPU memory as **cache** for main CPU memory
- Roughly 1/N of matrix were loaded to each device
  - Balanced load
  - Minimal communication w/ the host
  - Write back to main memory only finished parts
- Submatrix size
  - small enough to load several of them at once
  - large enough to reduce latency


The Matrix A on host
0
1
2
3
3
2
1
0
0
1
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
Rq-1
Device 1
1
1
1

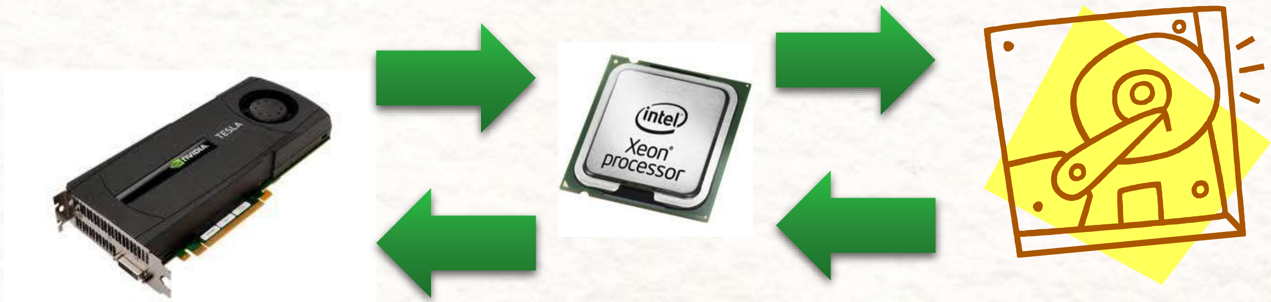
Inter-device communication

- Happens whenever we transition from one phase to another
- Data transfer can be costly
- Possible solutions
  - Peer-to-peer: 2x fast
  - **Overlapping** of computation and data transfer
- Synchronization is critical.
  - CPU threads control GPU devices.
  - Between Phases II and III.


Stream 0
Stream 1
Do Phase II
Flush Phase I to Host
Communication to prepare for Phase III
(Barrier)
(Barrier)
Do Phase III
Communication to prepare for Phase II
Do Phase I
Flush Phase II to Host
Communication to prepare for Phase II

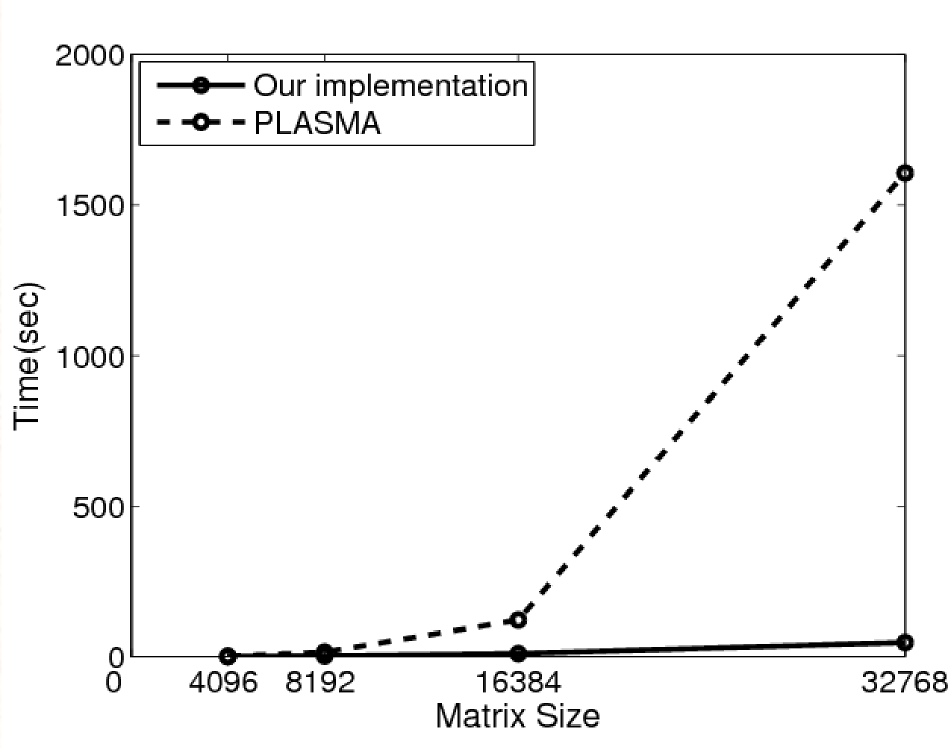
Extension to disk I/O

- For larger matrices, use main memory as cache for disks
- Prefetching, delayed write: exploit locality



Performance

- CPU: dual 2.4 GHz Intel® Xeon® quad-core
- Main memory: 16GB
- GPU: four Tesla C2050 graphics cards with 3GB memory
- CUDA 4.2 Runtime
- 33x compared to PLASMA, a numerical linear algebra library for multicore CPU
- Scalable to larger systems
  - 65,000 x 65,000 matrix amounts to 32GB



Conclusion

- Our implementation is scalable to very large systems.
- We streamlined operation across three memory layers.
- We were able to apply it to image segmentation.