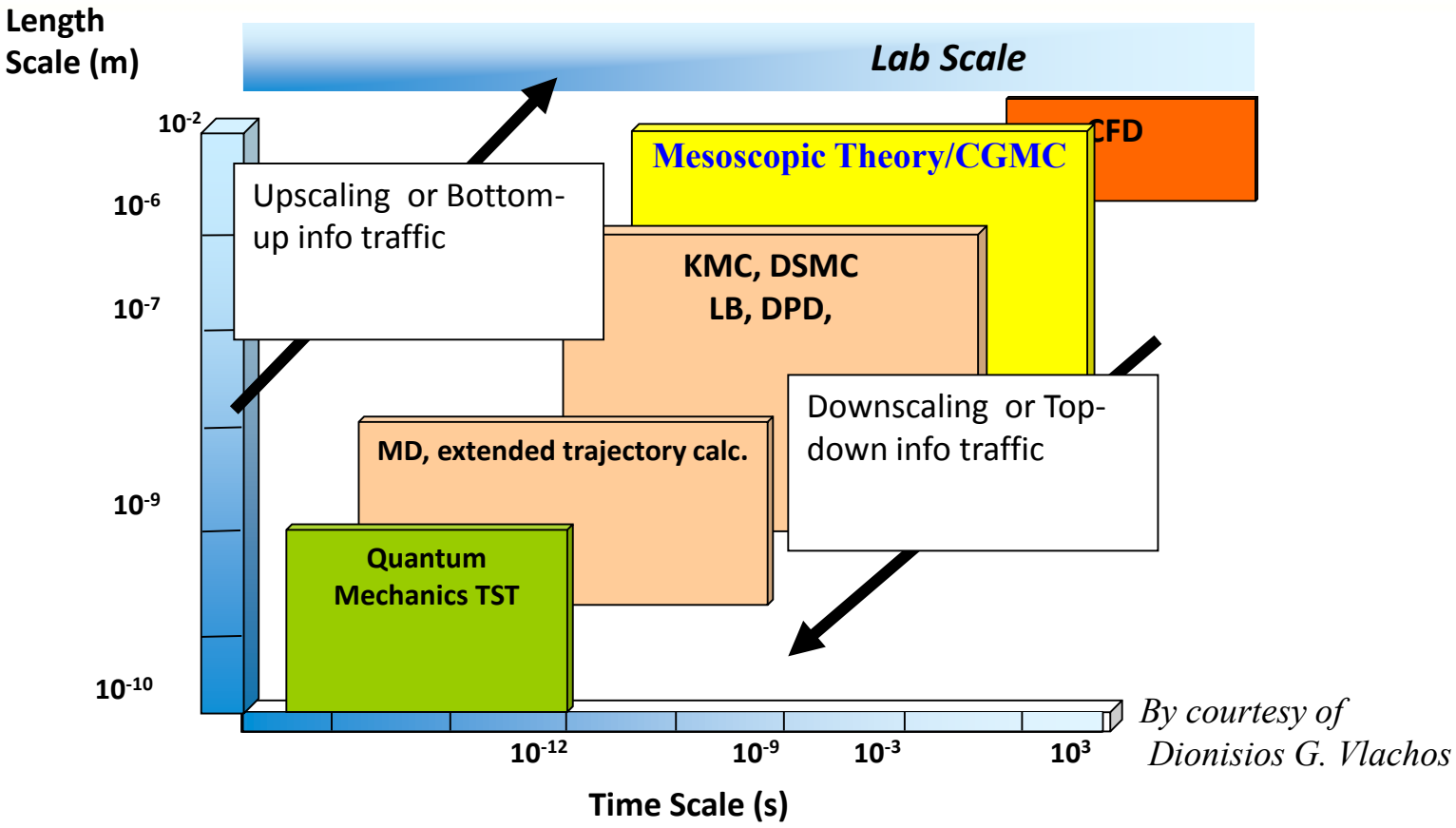
Parallelization of Tau-Leap Coarse-Grained Monte Carlo Simulations on GPUs

Lifan Xu, Michela Taufer,
Stuart Collins, Dionisios G. Vlachos

Global Computing Lab
University of Delaware

Multiscale Modeling: Challenges



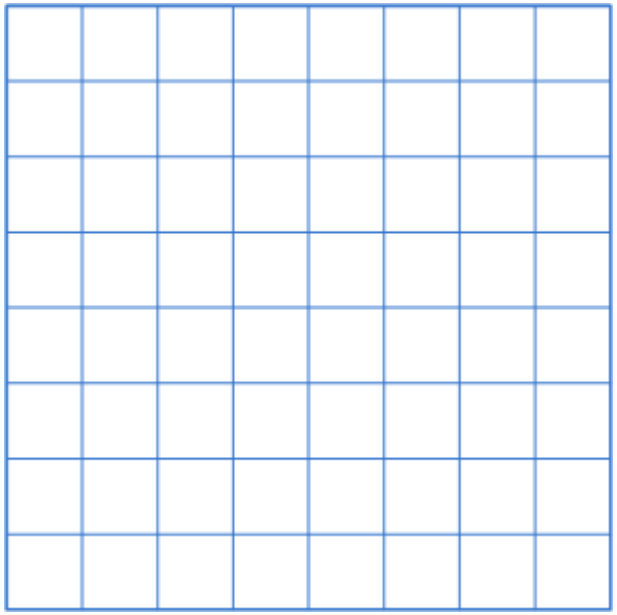
Our Goal: increase time and length scale for the CGMC by using GPUs

Outline

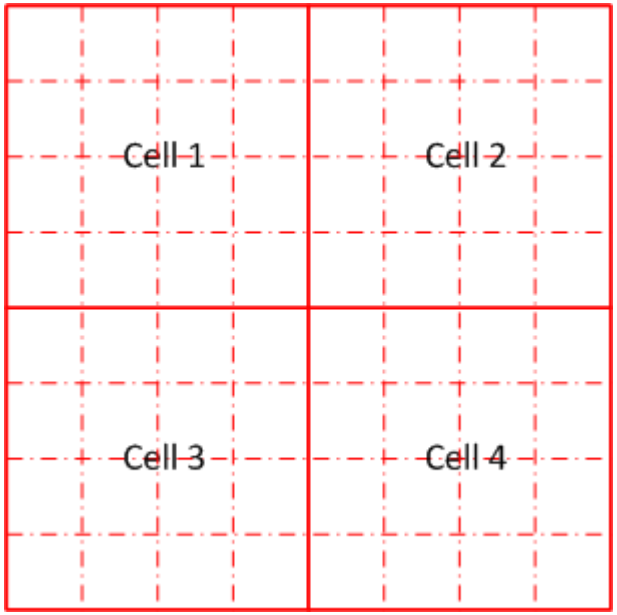
- Background
  - CGMC
  - GPU
- CGMC implementation on single GPU
  - Optimize memory use
  - Optimize algorithm
- CGMC implementation on multiple GPUs
  - Combine OpenMP and CUDA
- Accuracy
- Conclusions
- Future work

Coarse Grained Kinetic Monte Carlo (CGMC)

- Study phenomena such as catalysis, crystal growth, and surface diffusion
- Group neighboring microscopic sites (molecules) together into "coarse-grained" cells



8 X 8 microscopic sites

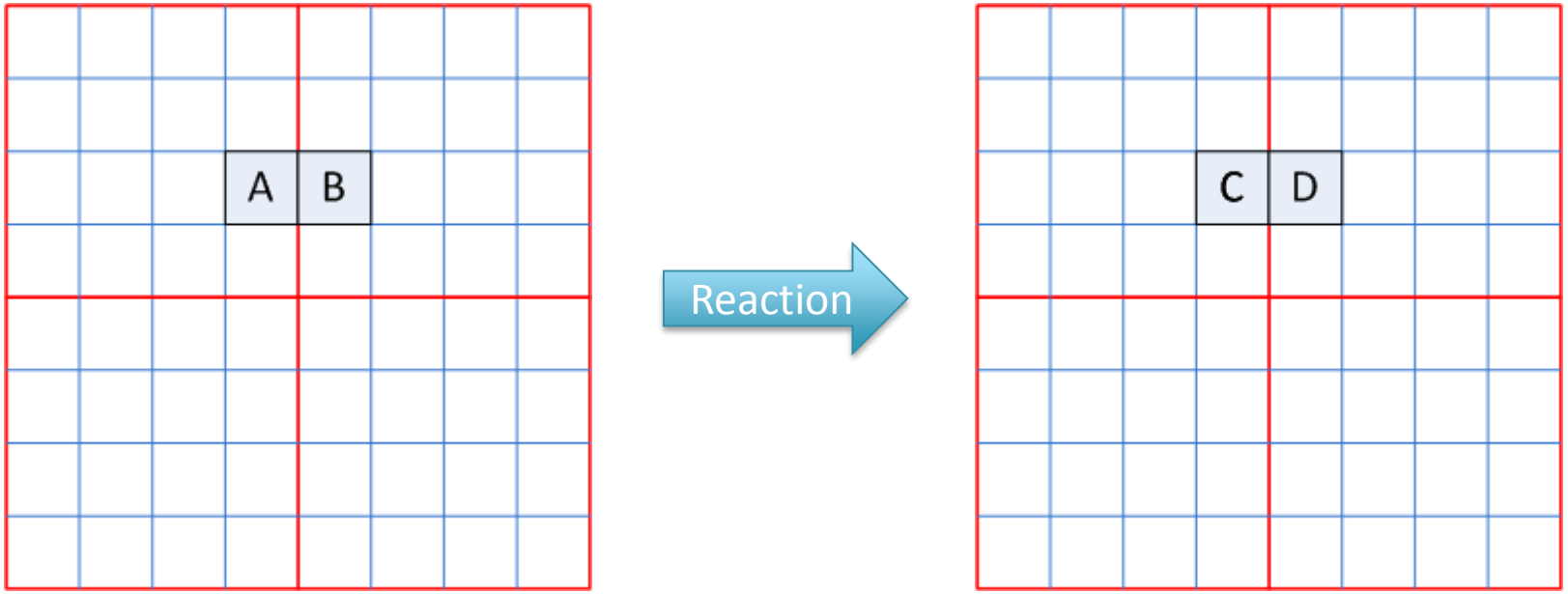


2 X 2 cells

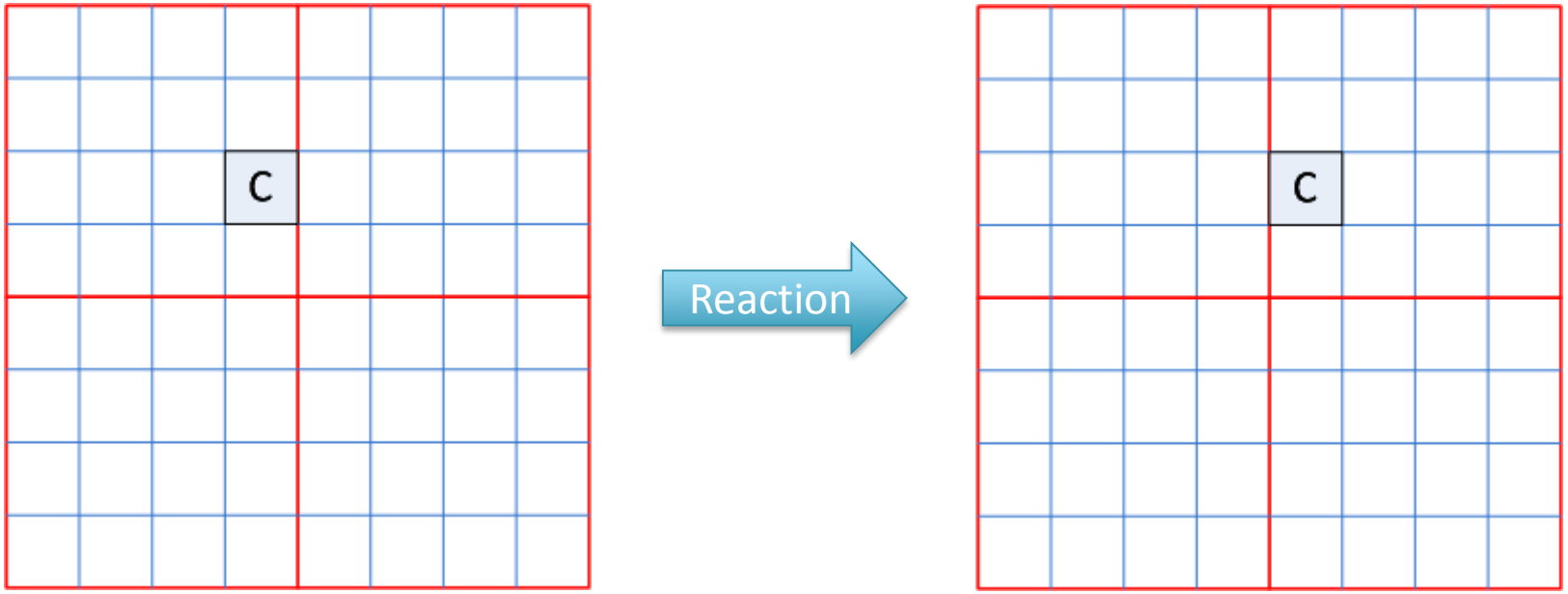
Terminology

- Coverage
  - A 2D matrix contains number of different species of molecules for each cell
- Events
  - Reaction
  - Diffusion
- Probability
  - A list of probabilities of all possible events on every cell
  - Probability is calculated based on neighboring cell's coverage
  - Probability has to be updated if the coverage of its neighbor changed
  - Events are selected based on probabilities

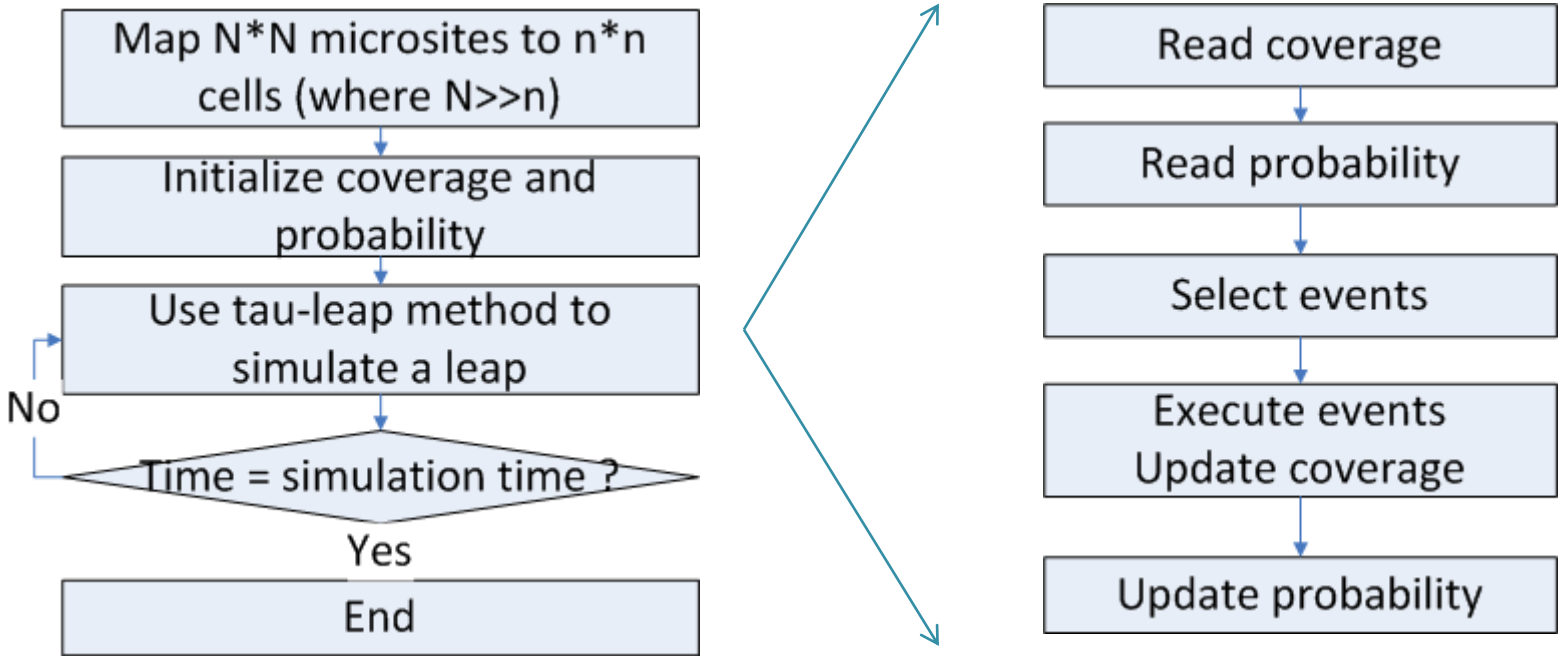
Reaction



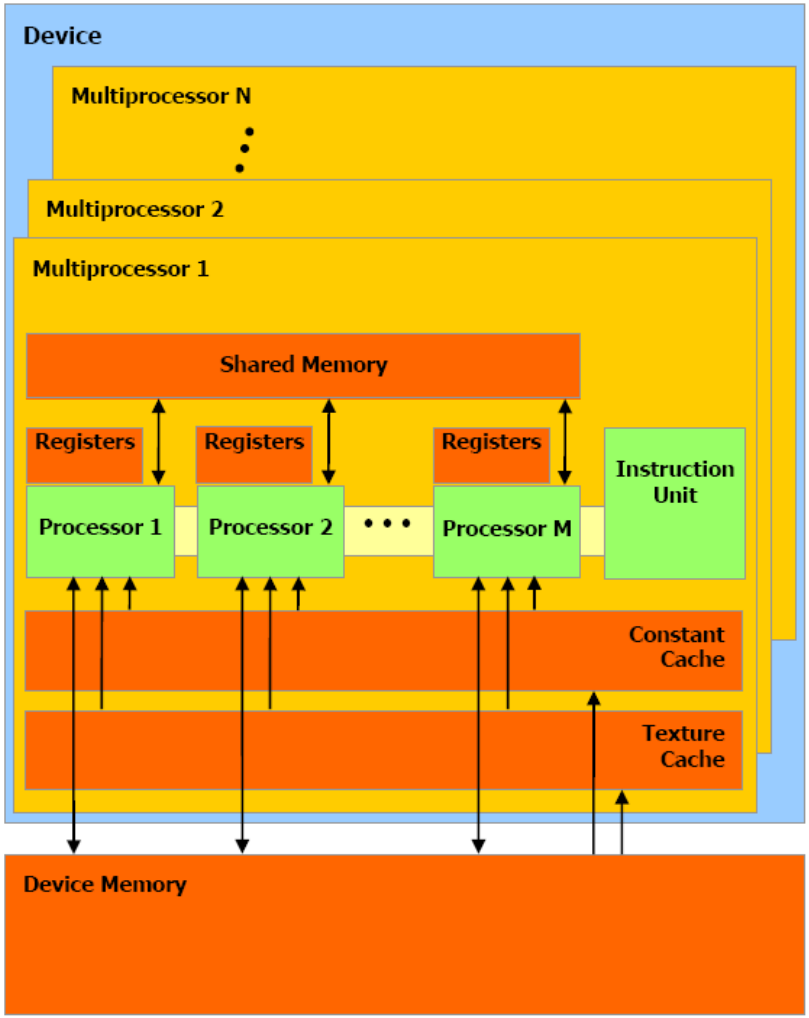
Diffusion



CGMC Algorithm



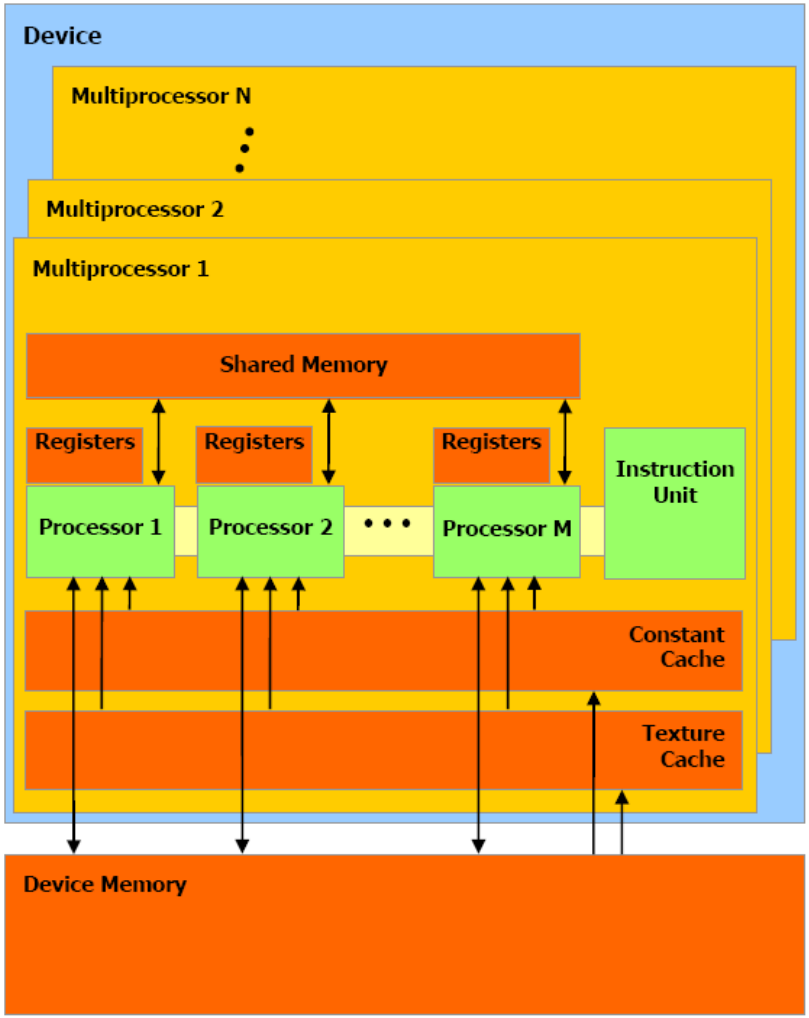
GPU Overview (I)



From: CUDA Programming Guide, NVIDIA

- NVIDIA Tesla C1060:
  - 30 Streaming Multiprocessor (1-N)
  - 8 Scalar Processors/SM (1-M)
  - 30, 8-way SIMD cores = 240 PEs
- Massively parallel multithreaded
  - Up to 30720 active threads handled by thread execution manager
- Processing power
  - 933 GigaFLOPS(single precision)
  - 78 GigaFLOPS(double precision)

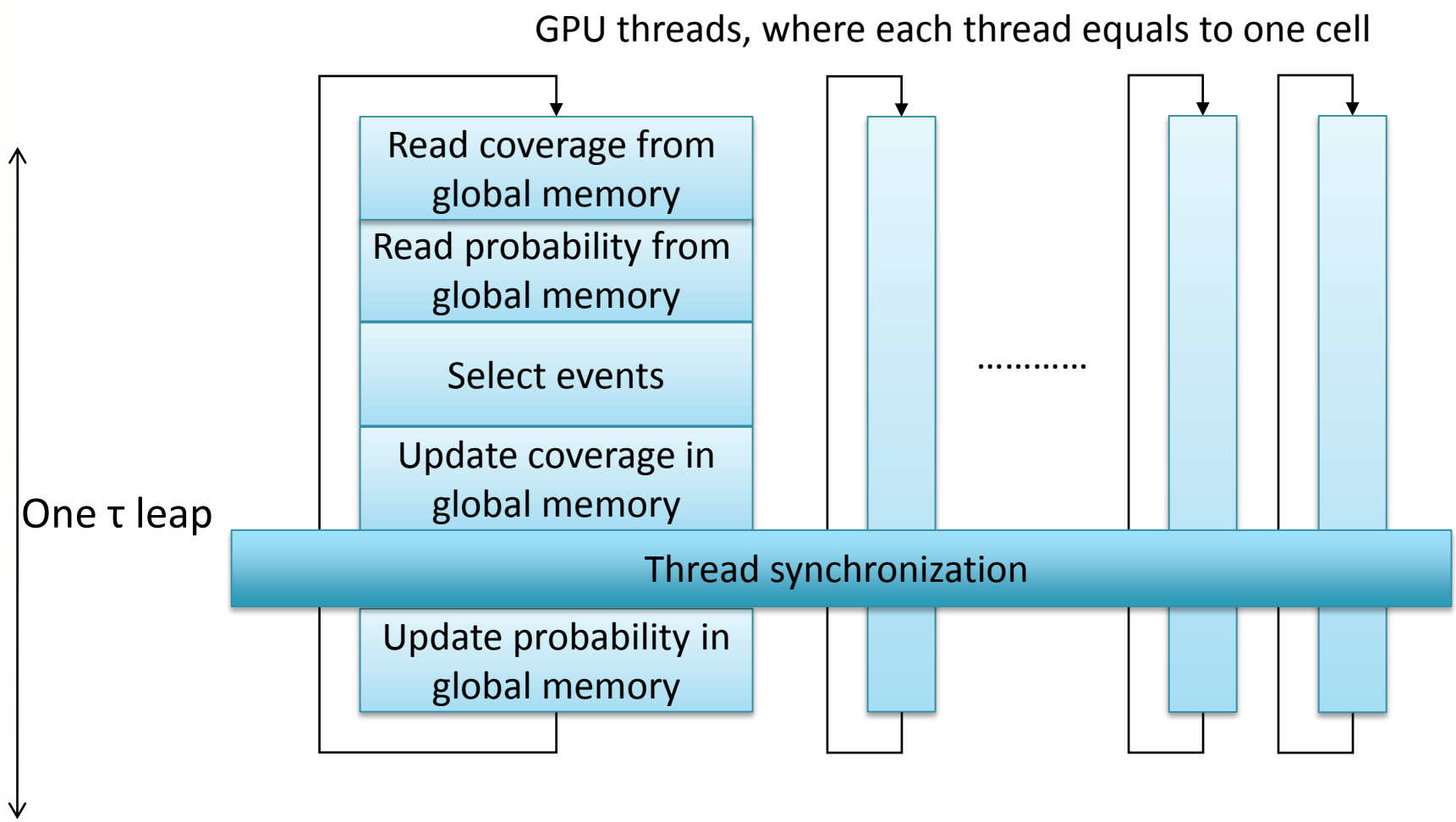
GPU Overview (II)



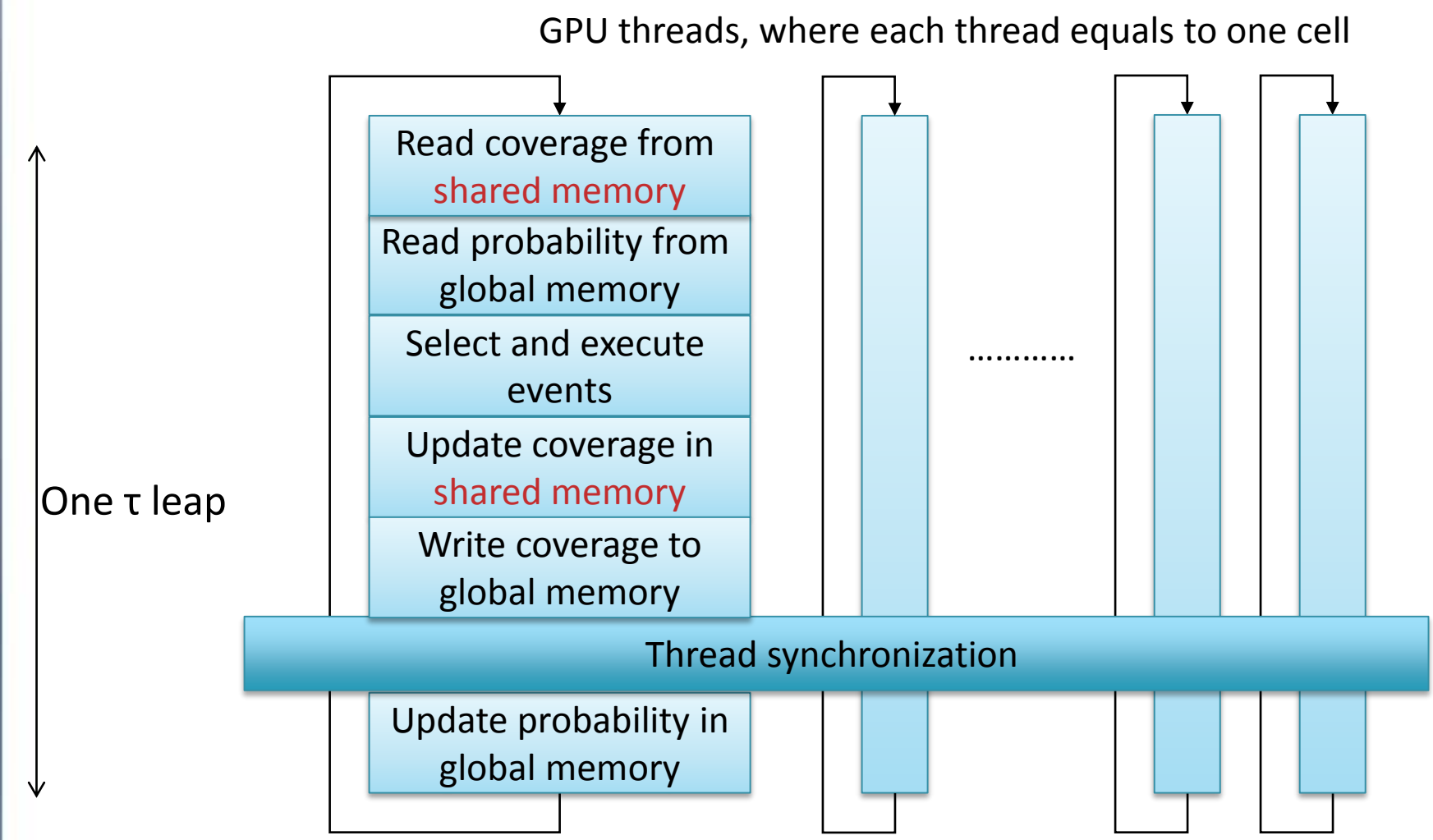
From CUDA Programming Guide, NVIDIA

- Memory types:
  - Read/write per thread
    - Registers
    - Local memory
  - Read/write per block
    - Shared memory
  - Read/write per grid
    - Global memory
  - Read-only per grid
    - Constant memory
    - Texture memory
- Communication among devices and with CPU
  - Through PCI Express bus

Multi-thread Approach



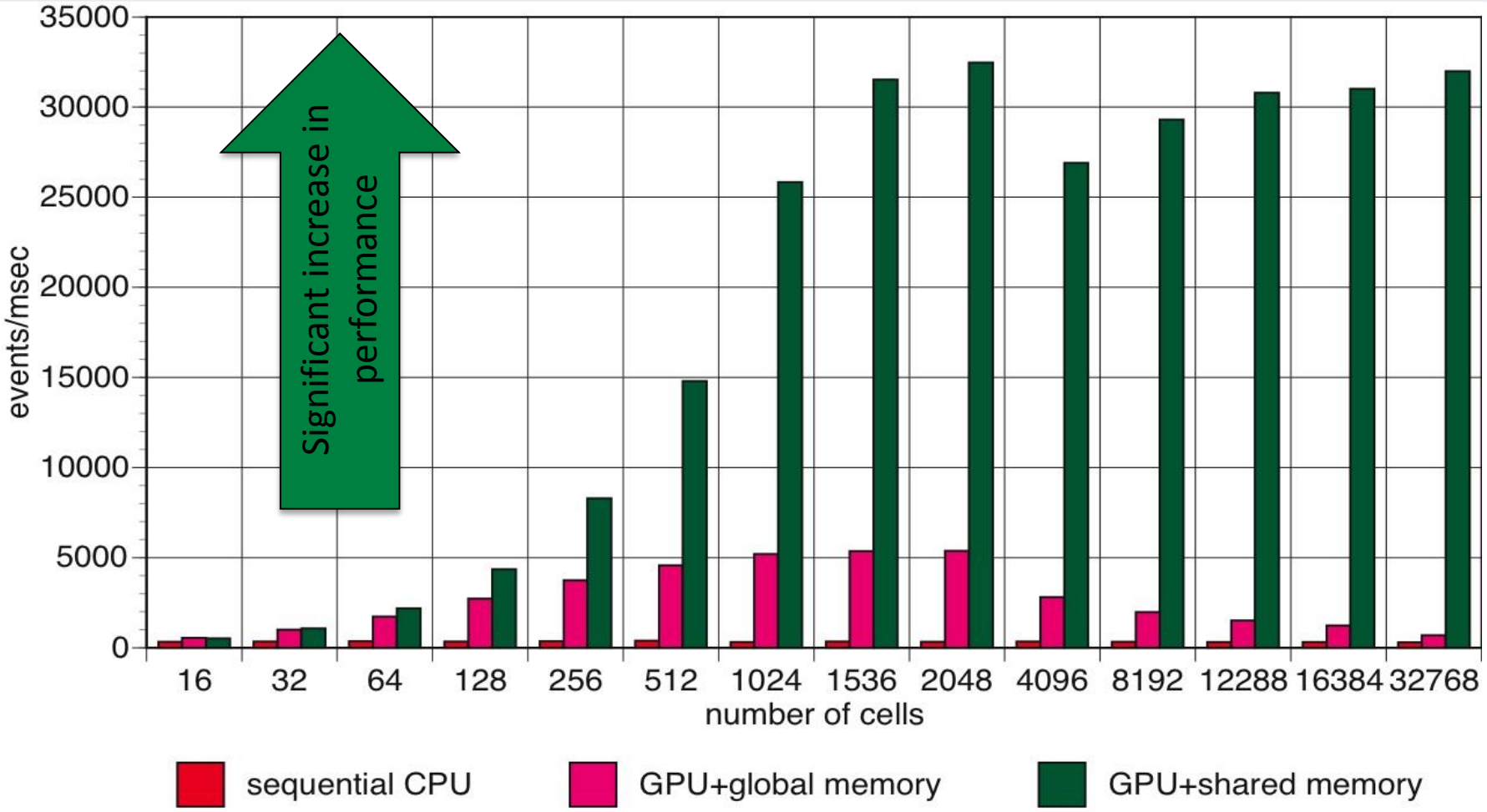
Memory Optimization



Performance

Platforms	
GPU –Tesla C1060	Intel(R) Xeon(R) CPU X5450(CPU)
Number of cores 240	Number of cores 4
Global memory 4GB	Memory 4GB
Clock rate 1.44 GHz	Clock rate 3 GHz

Performance



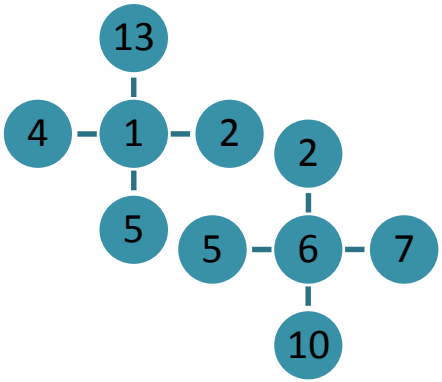
Time scale: CGMC simulations can be 100X faster than a CPU implementation

Rethinking the CGMC Algorithm for GPUs

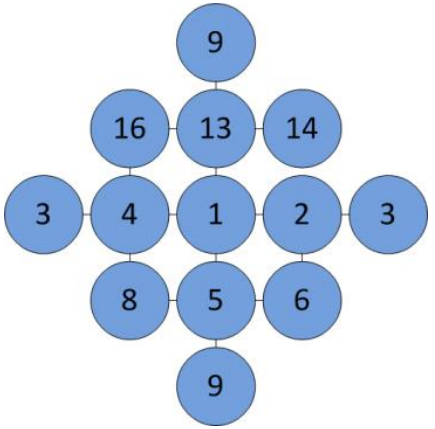
- Redesign of cell structures:
  - Macro-cell = set of cells
  - Number of cells per macro-cell is flexible
- Redesign of event execution:
  - One thread per macro-cell (coarse-grained parallelism) or one block per macro-cell (fine-grained parallelism)
  - Events replicated across macro-cells

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Cells in a 4x4 system

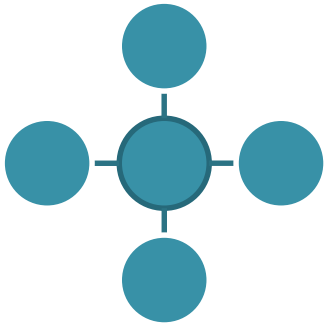


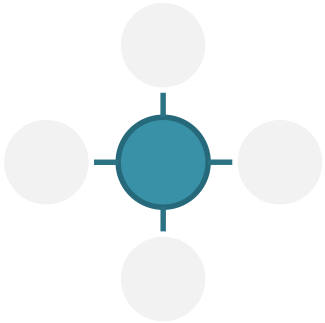
2-layer macro-cells



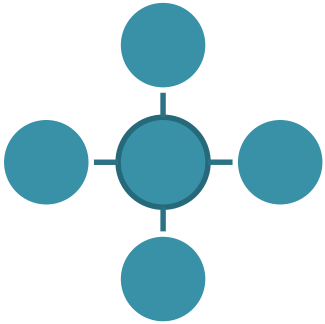
3-layer macro-cell

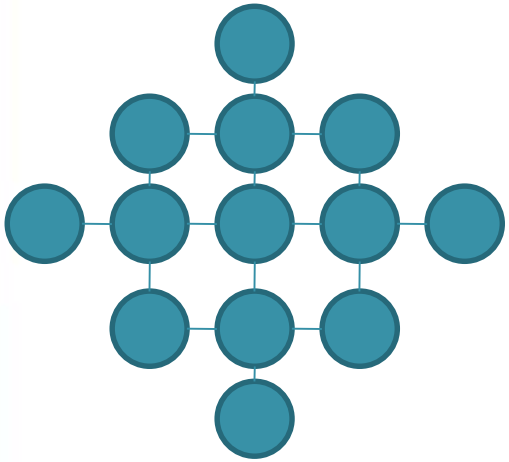
Synchronization Frequency

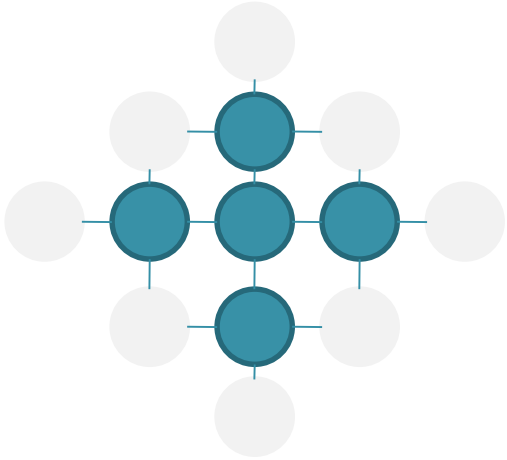


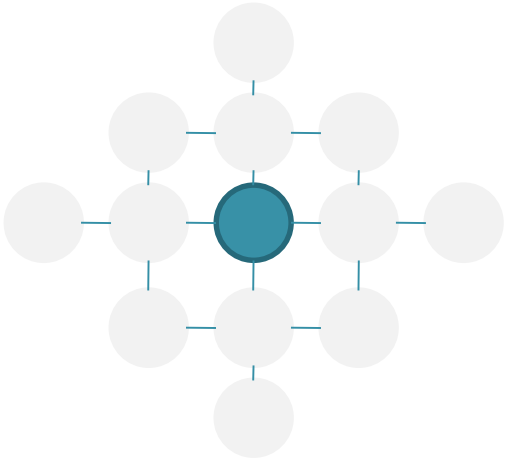


Synchro nization





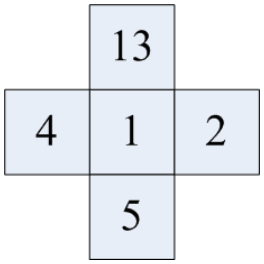




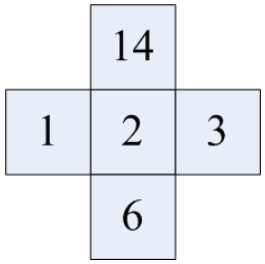
Synchro nization

Leap 1 Leap 2 Leap 3 Leap 4

Event Replication



Macro-cell 1



Macro-cell 2

E1: A[cell 1] -> A[cell 2]

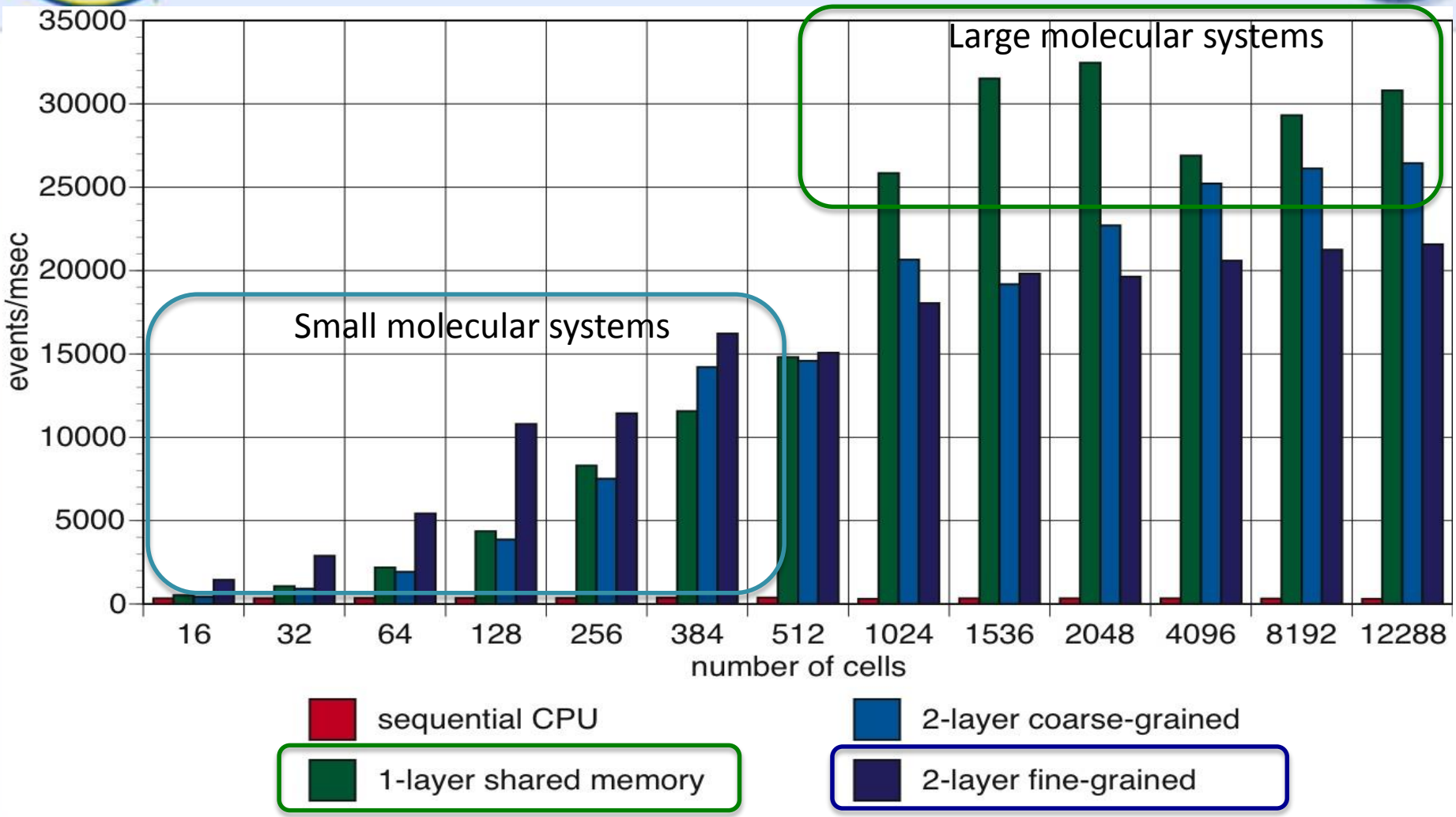
A is a molecule specie

	Macro-cell 1 (block 0)			**Macro-cell 2 (block 1)**			
	Cell 1 (thread 0)	Cell 2 (thread 1)	…	**Cell 2 (thread 5)**	Cell 1 (thread 6)	Cell 3 (thread 7)	…
Leap 1	E1(A--)	E1(A++)		E1(A++)	E1(A--)		
	E1(A--)	E1(A++)		E1(A++)	E1(A--)		
Leap 2	E1(A--)			E1(A++)			

Coarse-Grained vs. Fine-Grained

- 2-layer coarse-grained
  - Each thread is in charge of one macro-cell
  - Each thread simulates five cells in every first leap, one central cell in every second leap
- 2-layer fine-grained
  - Each block is in charge of one macro-cell
  - Each block has five threads
  - Each thread simulates one cell in every first leap
  - Five thread simulate the central cell together in every second leap

Performance



Time scale: CGMC simulations can be 100X faster than a CPU implementation

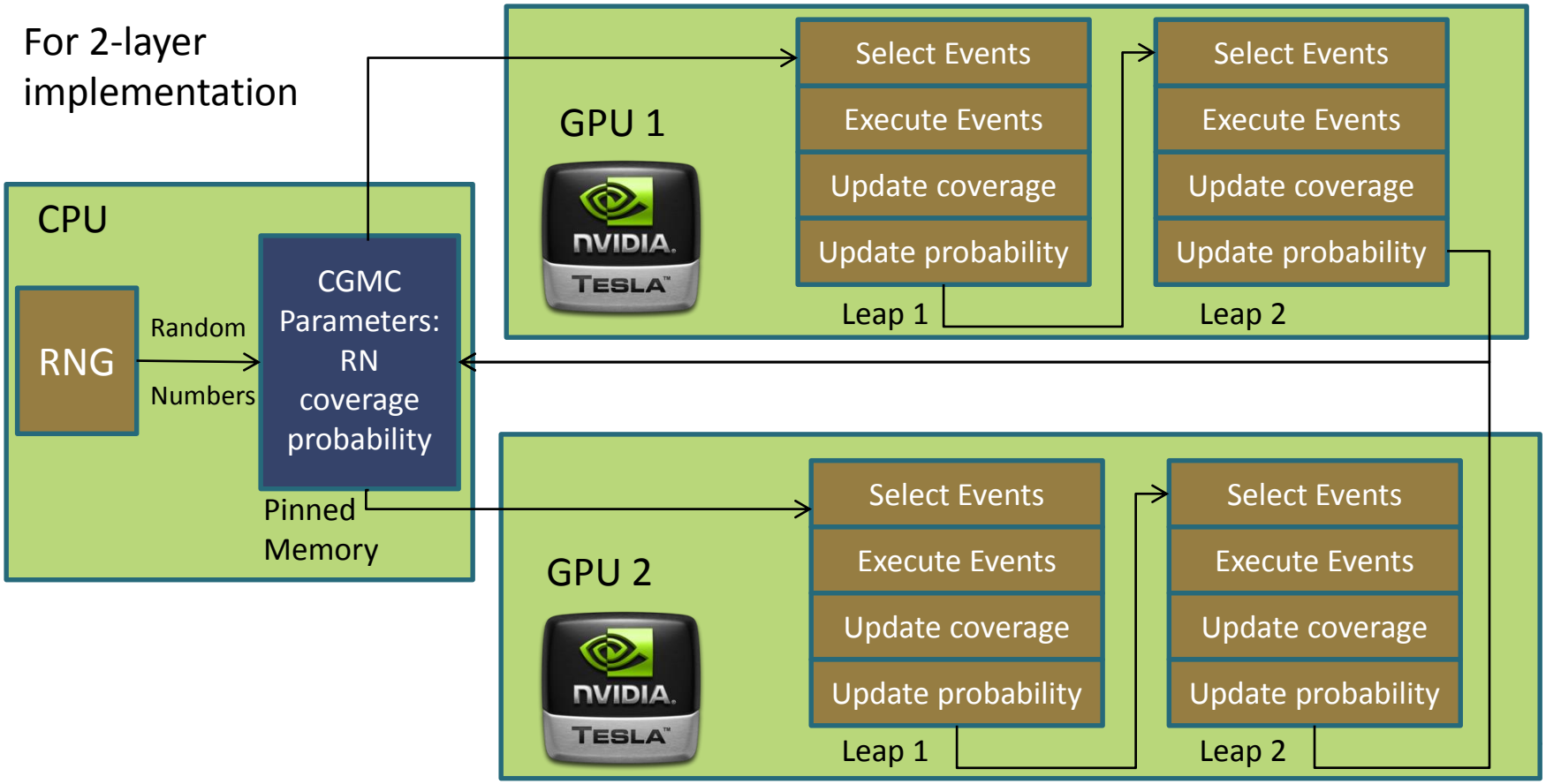
Multi-GPU Implementation

- Limit in number of cells for a single GPU implementation
  - Use multiple GPUs
- Synchronization between multiple GPUs is costly
  - Take advantage of 2-layer fine-grained parallelism
- Combine OpenMP with CUDA for multi-GPU programming:
  - Use portable pinned memory for communication between CPU threads
  - Use mapped pinned memory for data copy between CPU and GPU(zero copy)
  - Use write-combined memory for fast data access

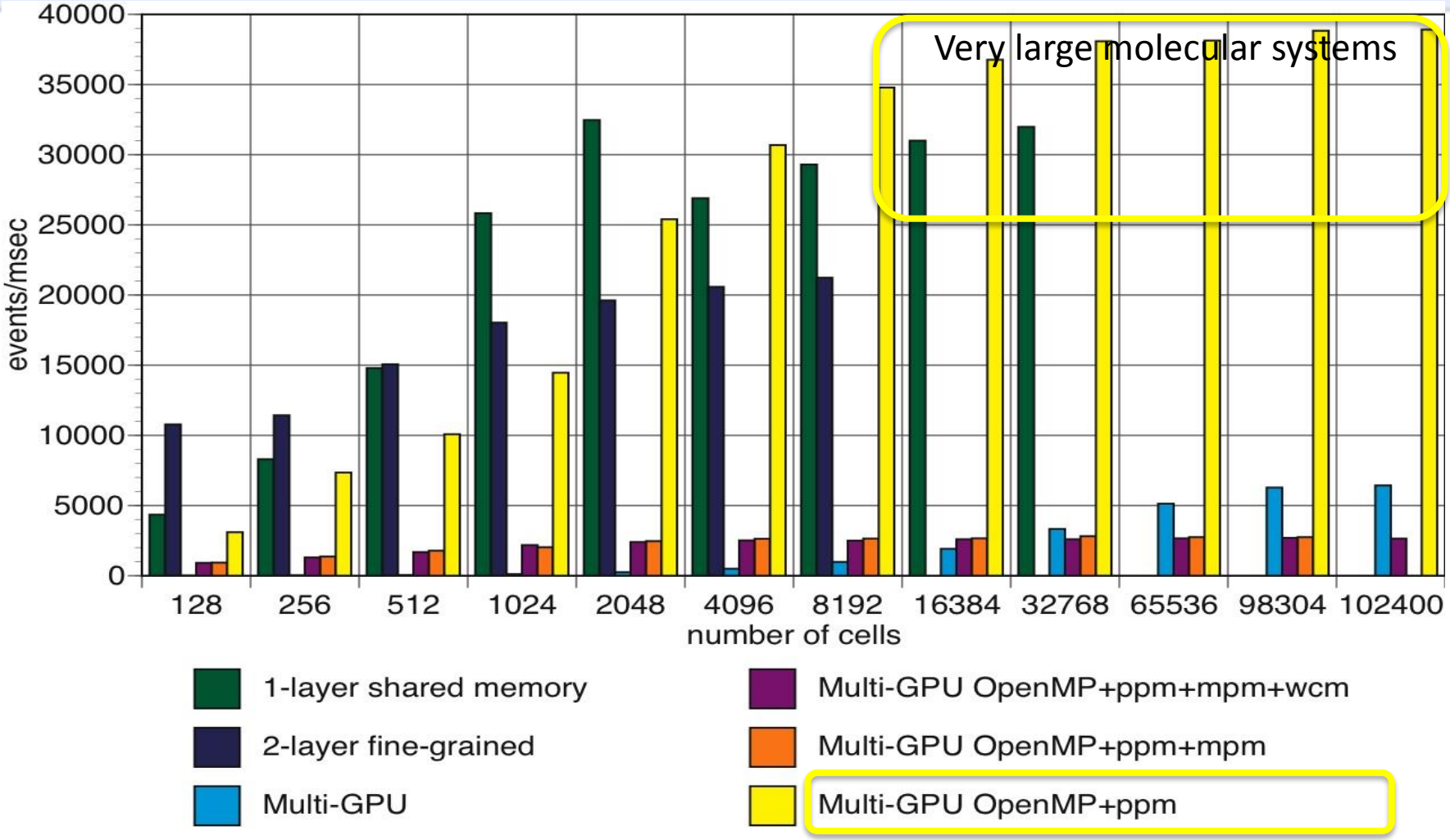
Pinned Memory

- Portable pinned memory(PPM)
  - Available for all host threads
  - Can be freed by any host thread
- Mapped pinned memory(MPM)
  - Allocate memory on host side
  - Memory can be mapped to device's address space
  - Two addresses: one in host memory and one in device memory
  - No explicit memory copy between CPU and GPU
- Write-combined pinned memory(WCM)
  - Transfers across the PCI Express bus,
  - Buffer is not snooped
  - High transfer performance but no data coherency guarantee

Multi-GPU Implementation



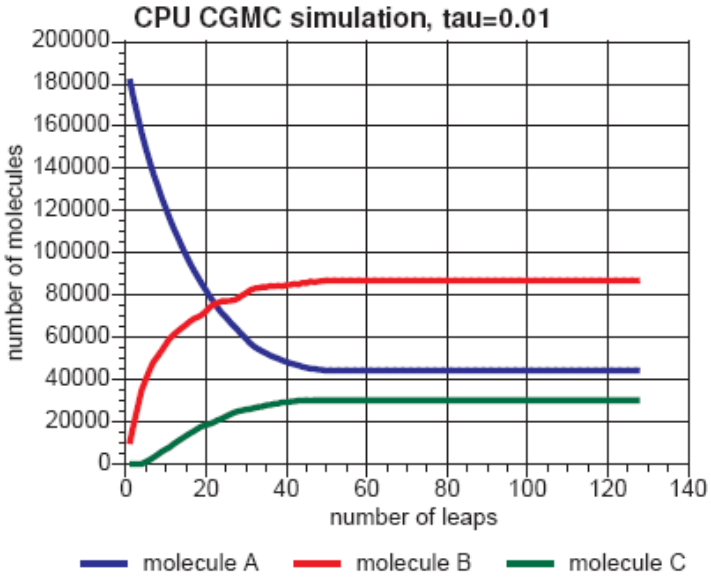
Performance

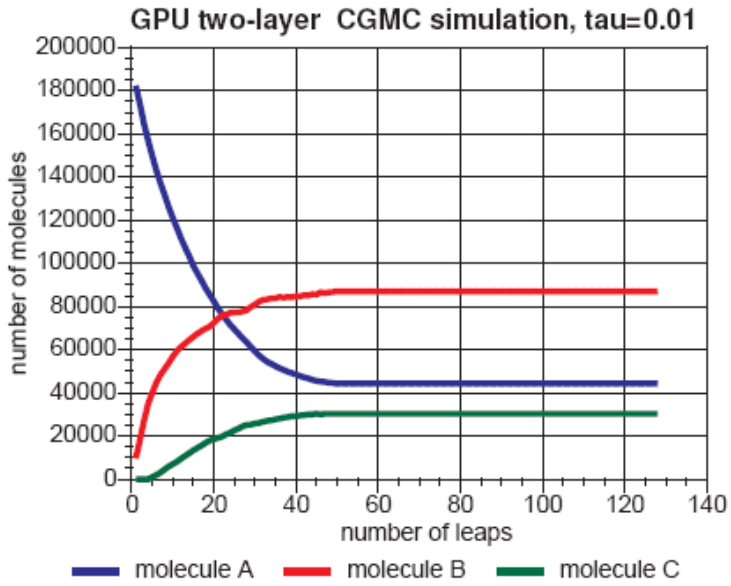


Length scale: 120X faster with molecular systems larger than 13,000 cells

Accuracy

- Simulations of three different species of molecules, A, B, and C
  - A change to B and vice-versa
  - 2 B molecules change to C and vice-versa
  - A, B, and C can diffuse
- Number of molecules in system reaching the equilibrium and in equilibrium





Conclusion

- We present a parallel implementation of the tau-leap method for CGMC simulations on single and multi-GPUs
- We identify the most efficient parallelism granularity for the simulations given a molecular system with a certain size
  - 42X speedup for small systems using 2-layer fine-grained thread parallelism
  - 100X speedup for average systems using 1-layer parallelism
  - 120X speedup for large systems using 2-layer fine-grained thread parallelism across multiple GPUs
- We increase both time scale (up to 120 times) and length scale (up to 100,000)

Future work

- Study molecular systems with heterogeneous distributions
- Identify the most efficient level of parallelism for a given system for:
  - Different system sizes
  - Different site distributions
- Combine MPI, OpenMP, and CUDA to use multiple GPUS across different machines
- Link phenomena and models across scales i.e., from QM to MD and to CGMC

Acknowledgements

GCL Members:

Trilce Estrada
Boyu Zhang
Abel Licon
Narayan Ganesan
Lifan Xu
Philip Saponaro
Maria Ruiz
Michela Taufer

Collaborators:

Dionisios G. Vlachos and Stuart Collins (UDel)

GCL members in Spring 2010

More questions: xulifan@udel.edu

Sponsors: