Advanced CUDA: Fundamental CUDA Programming Abstractions

John E. Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

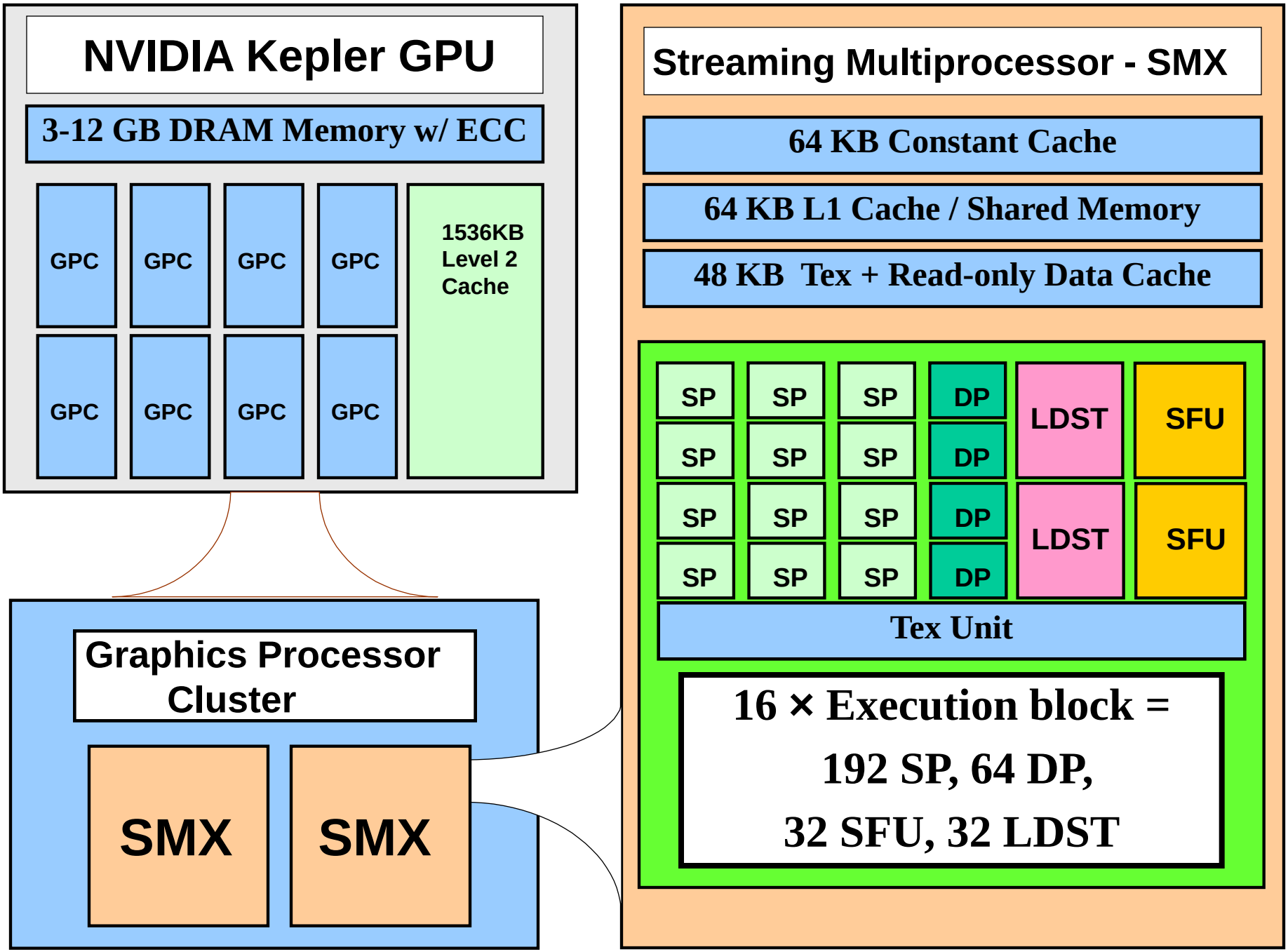
http://www.ks.uiuc.edu/Research/gpu/

GPGPU2: Advanced Methods for Computing with CUDA,

University of Cape Town, April 2014

CUDA Work Abstraction

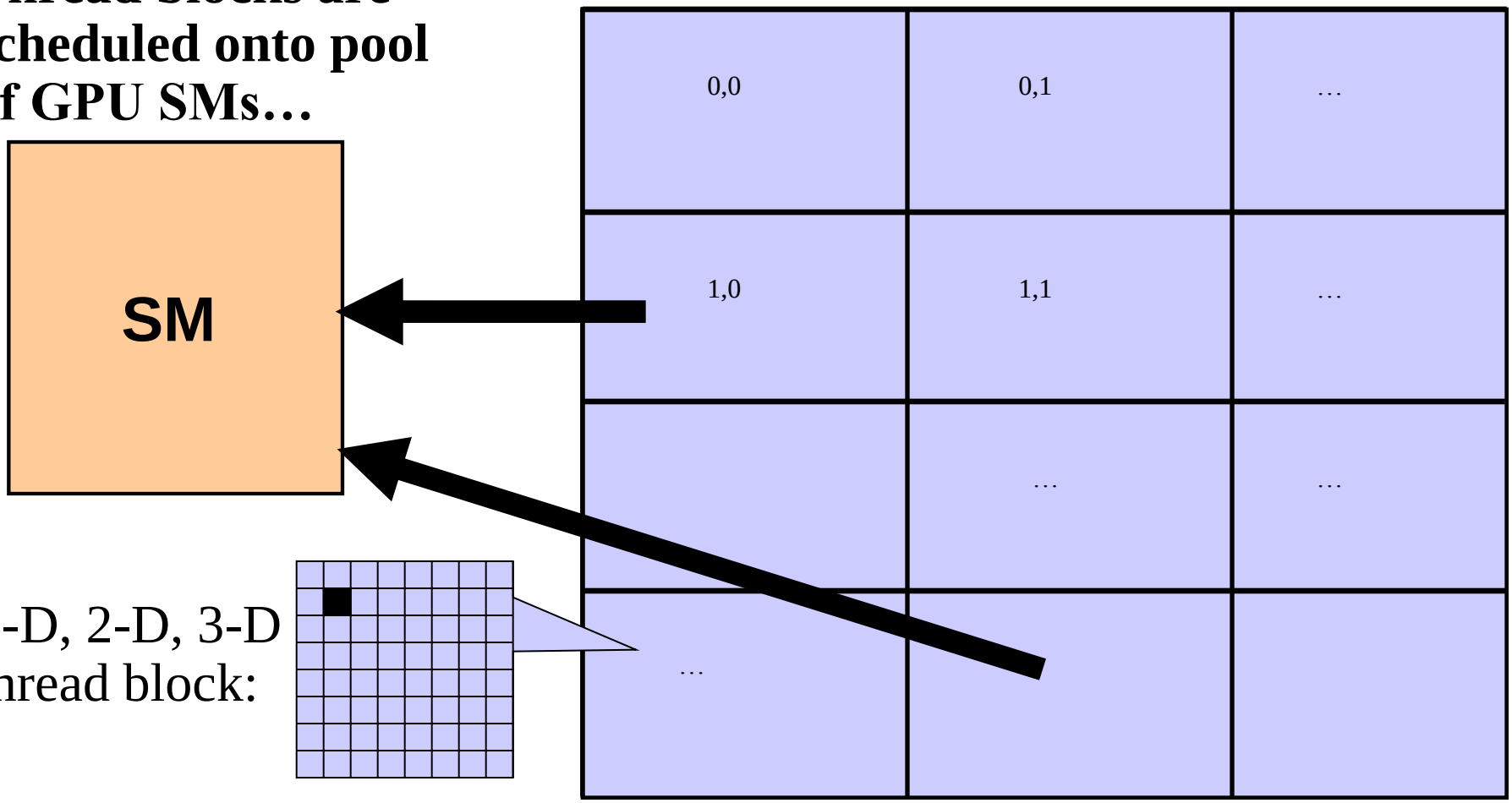
- Work is expressed as a multidimensional array of independent work items called **“threads”** – not the same thing as a CPU thread
- CUDA Kernels can be thought of as telling a GPU to compute **all iterations** of a set of nested loops **concurrently**
- Threads are dynamically scheduled onto hardware according to a hierarchy of thread groupings


NVIDIA Kepler GPU
3-12 GB DRAM Memory w/ ECC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
1536KB Level 2 Cache
Graphics Processor Cluster
SMX
SMX
Streaming Multiprocessor - SMX
64 KB Constant Cache
64 KB L1 Cache / Shared Memory
48 KB Tex + Read-only Data Cache
SP
DP
LDST
SFU
Tex Unit
16 × Execution block =
192 SP, 64 DP,
32 SFU, 32 LDST

CUDA Work Abstractions: Grids, Thread Blocks, Threads

1-D, 2-D, or 3-D (SM >= 2.x)
Grid of thread blocks:

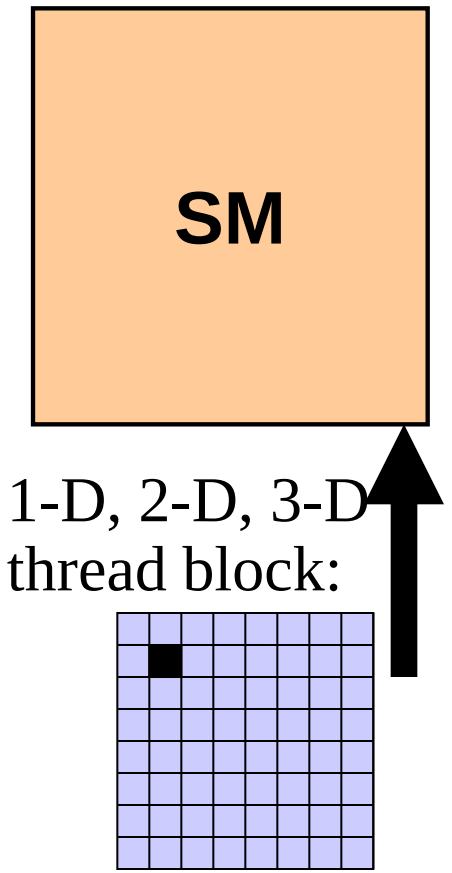
Thread blocks are scheduled onto pool of GPU SMs…



GPU Thread Block Execution

- Thread blocks are decomposed onto hardware in **32-thread "warps"**
- Hardware execution is scheduled in units of warps – an SM can execute warps from several thread blocks
- Warps run in SIMD-style execution:
  - **All threads execute the same instruction in lock-step**
  - **If one thread stalls, the entire warp stalls…**
  - **A branch taken by a thread has to be taken by all threads... (divergence is bad!)**

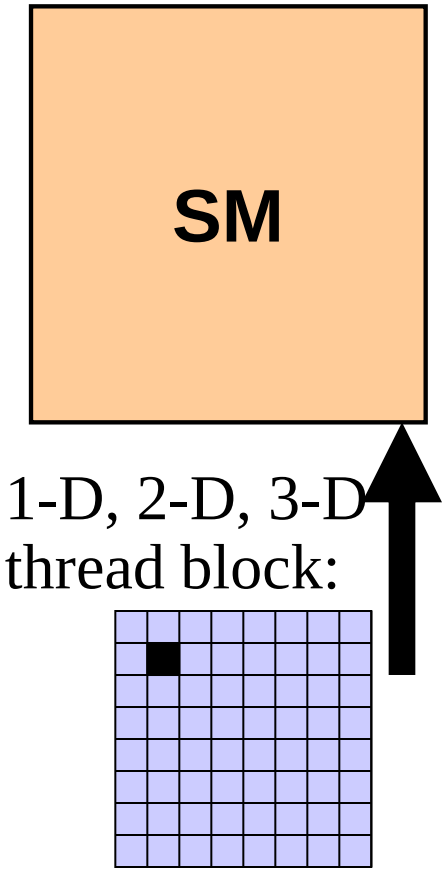
Thread blocks are multiplexed onto pool of GPU SMs…



GPU Warp Branch Divergence

- Branch divergence: when not all threads take the same branch, the entire warp has to **execute both sides of the branch**
- GPU blocks memory writes from disabled threads in the "if then" branch, then inverts all thread enable states and runs the "else" branch
- GPU hardware detects warp reconvergence and then runs normally...
- Not unique to GPUs, an attribute of all SIMD hardware designs…
- In the case of the GPU, we are at least benefiting from a completely **hardware-based** implementation…

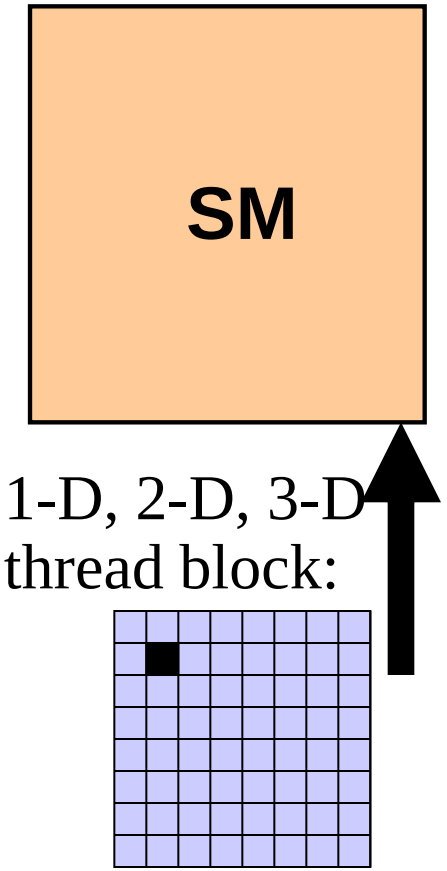
Thread blocks are multiplexed onto pool of GPU SMs…



GPU Thread Block Collective Operations

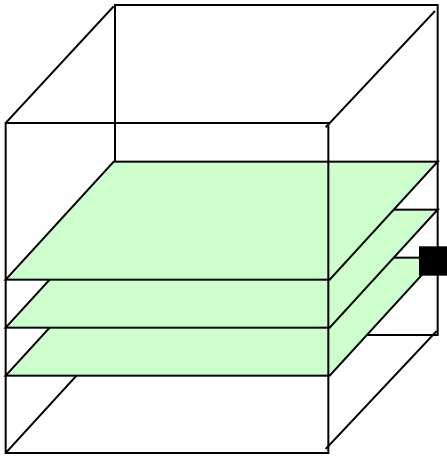
- Threads within the same thread block can communicate with each other in fast on-chip **shared memory**
- Once scheduled on an SM, **thread blocks run until completion**
- Because the order of thread block execution is arbitrary and they can't be stopped, **they cannot communicate or synchronize with other thread blocks**
- **Atomic memory ops are an exception wrt/ communication**

Thread blocks are multiplexed onto pool of GPU SMs…

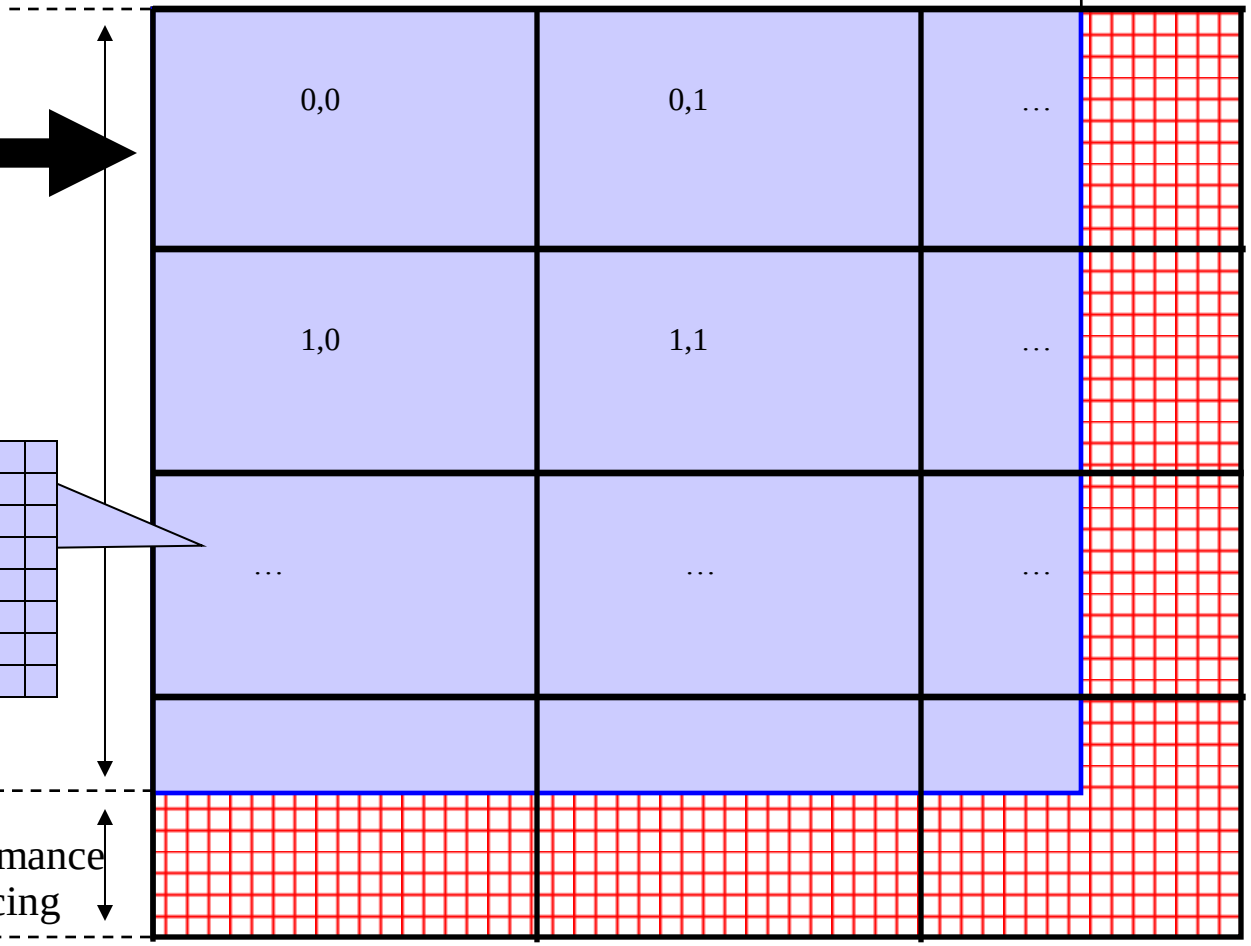


CUDA Grid/Block/Thread Decomposition

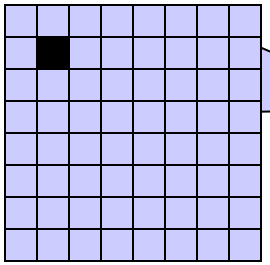
1-D, 2-D, or 3-D Computational Domain



1-D, 2-D, or 3-D (SM >= 2.x) Grid of thread blocks:



1-D, 2-D, 3-D thread block:



Padding arrays out to full blocks optimizes global memory performance by guaranteeing memory coalescing

Indexing Work

- Within a CUDA kernel:
  - Grid: gridDim.[xyz]
  - Block: blockDim.[xyz] and blockIdx.[xyz]
  - Thread: threadIdx.[xyz]
- Example CUDA kernel with 1-D Indexing:

```
__global__ void cuda_add(float *c, float *a, float *b) {
  int idx = (blockIdx.x * blockDim.x) + threadIdx.x;
  c[idx] = a[idx] + b[idx];
}
```

Running a GPU kernel:

```
int sz = N * sizeof(float);
…
cudaMalloc((void**) &a_gpu, sz);
cudaMemcpy(a_gpu, a, sz, cudaMemcpyHostToDevice);
… // do the same for ‘b_gpu’, allocate ‘c_gpu’
int Bsz = 256; // 1-D thread block size
cuda_add<<<N/Bsz, Bsz>>>(c, a, b);
cudaDeviceSynchronize(); // make CPU wait for completion
...
cudaMemcpy(c, c_gpu, sz, cudaMemcpyDeviceToHost);
cudaFree(a_gpu);
… // free ‘b_gpu’, and ‘c_gpu’…
```

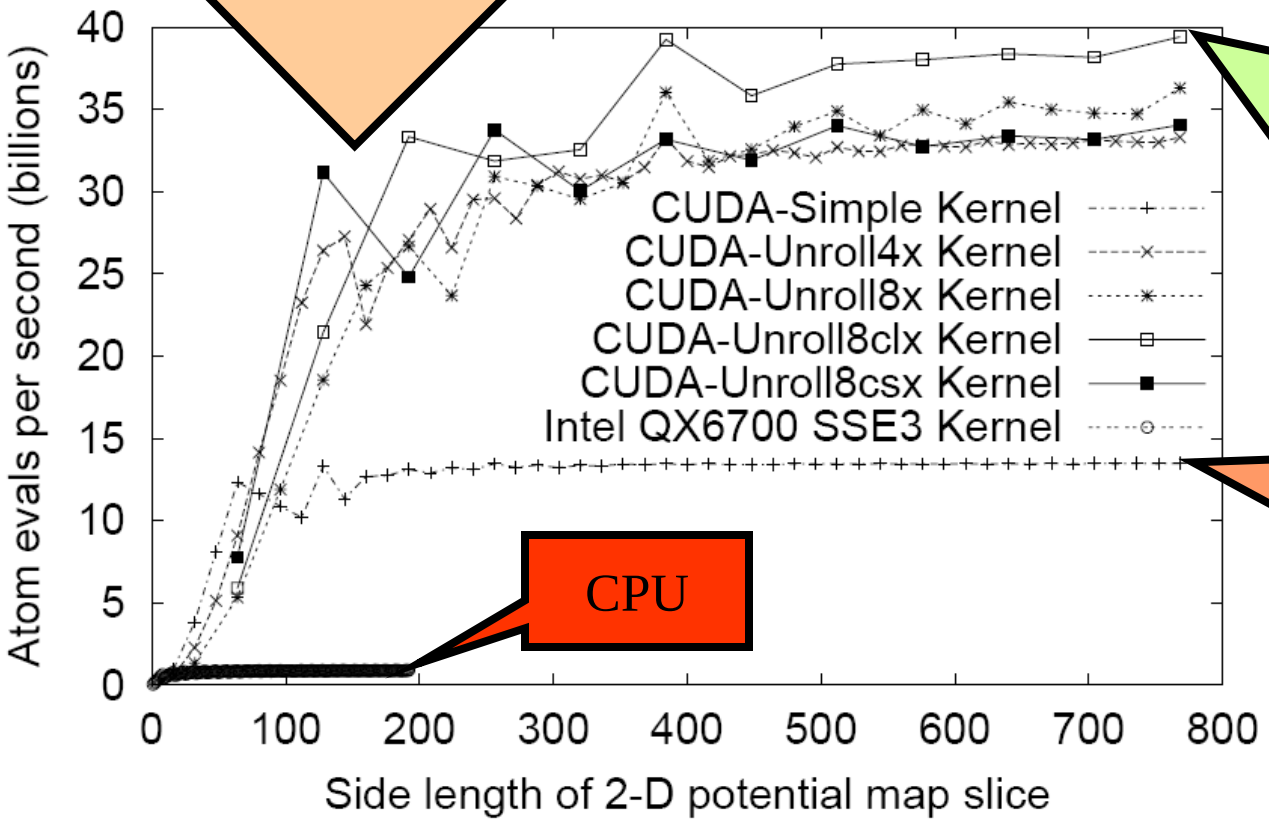
What if Work Size Isn't an Integer Multiple of the Thread Block Size?

- Threads must check if they are "in bounds":

```
__global__ void cuda_add(float *c, float *a, float *b, int N) {
  int idx = (blockIdx.x * blockDim.x) + threadIdx.x;
  if (idx < N) {
    c[idx] = a[idx] + b[idx];
  }
}
```

Direct Coulomb Summation Performance

Number of thread blocks modulo number of SMs results in significant performance variation for small workloads



CUDA-Unroll8clx: fastest GPU kernel, 44x faster than CPU, 291 GFLOPS on GeForce 8800GTX

CUDA-Simple: 14.8x faster, 33% of fastest GPU kernel

GPU computing. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.

An Approach to Writing CUDA Kernels

- Find an algorithm that can expose **substantial parallelism**, we'll ultimately need thousands of independent threads…
- Identify **appropriate** GPU memory or texture subsystems used to store data used by kernel
- Are there trade-offs that can be made to exchange computation for **more parallelism**?
  - Though counterintuitive, past successes resulted from this strategy
  - "Brute force" methods that expose significant parallelism do surprisingly well on GPUs
- Analyze the real-world use case for the problem and select a specialized kernel for the problem sizes that will be heavily used

Getting Performance From GPUs

- Don't worry (much) about counting arithmetic operations…at least until you have nothing else left to do
- GPUs provide tremendous memory bandwidth, but even so, **memory bandwidth often ends up being the performance limiter**
- Keep/reuse data in **registers** as long as possible
- The main consideration when programming GPUs is **accessing memory efficiently**, and storing operands in the **most appropriate memory system** according to data size and access pattern

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- NVIDIA CUDA team
- NVIDIA OptiX team
- NCSA Blue Waters Team
- Funding:
  - NSF OCI 07-25070
  - NSF PRAC “The Computational Microscope”
  - NIH support: 9P41GM104601, 5R01GM098243-02


NIH BTRC for Macromolecular Modeling and Bioinformatics
1990-2017
Beckman Institute
University of Illinois at
Urbana-Champaign

GPU Computing Publications

http://www.ks.uiuc.edu/Research/gpu/

- **Runtime and Architecture Support for Efficient Data Exchange in Multi-Accelerator Applications** Javier Cabezas, Isaac Gelado, John E. Stone, Nacho Navarro, David B. Kirk, and Wen-mei Hwu. IEEE Transactions on Parallel and Distributed Systems, 2014. (Accepted)
- **Unlocking the Full Potential of the Cray XK7 Accelerator** Mark Klein and John E. Stone. Cray Users Group, 2014. (In press)
- **Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations** Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten. Journal of Parallel Computing, 2014. (In press)
- **GPU-Accelerated Analysis and Visualization of Large Structures Solved by Molecular Dynamics Flexible Fitting** John E. Stone, Ryan McGreevy, Barry Isralewitz, and Klaus Schulten. Faraday Discussion 169, 2014. (In press)
- **GPU-Accelerated Molecular Visualization on Petascale Supercomputing Platforms.** J. Stone, K. L. Vandivort, and K. Schulten. UltraVis'13: Proceedings of the 8th International Workshop on Ultrascale Visualization, pp. 6:1-6:8, 2013.
- **Early Experiences Scaling VMD Molecular Visualization and Analysis Jobs on Blue Waters.** J. E. Stone, B. Isralewitz, and K. Schulten. In proceedings, Extreme Scaling Workshop, 2013.
- **Lattice Microbes: High-performance stochastic simulation method for the reaction-diffusion master equation.** E. Roberts, J. E. Stone, and Z. Luthey-Schulten. J. Computational Chemistry 34 (3), 245-255, 2013**.**

GPU Computing Publications

http://www.ks.uiuc.edu/Research/gpu/

- **Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.** M. Krone, J. E. Stone, T. Ertl, and K. Schulten. *EuroVis Short Papers*, pp. 67-71, 2012.
- **Fast Analysis of Molecular Dynamics Trajectories with Graphics Processing Units – Radial Distribution Functions.** B. Levine, J. Stone, and A. Kohlmeyer. *J. Comp. Physics*, 230(9):3556-3569, 2011.
- **Immersive Out-of-Core Visualization of Large-Size and Long-Timescale Molecular Dynamics Trajectories.** J. Stone, K. Vandivort, and K. Schulten. G. Bebis et al. (Eds.): *7th International Symposium on Visual Computing (ISVC 2011)*, LNCS 6939, pp. 1-12, 2011.
- **Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters.** J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J Phillips. *International Conference on Green Computing*, pp. 317-324, 2010.
- **GPU-accelerated molecular modeling coming of age. J. Stone, D. Hardy, I. Ufimtsev, K. Schulten.** *J. Molecular Graphics and Modeling*, 29:116-125, 2010.
- **OpenCL: A Parallel Programming Standard for Heterogeneous Computing. J. Stone, D. Gohara, G. Shi.** *Computing in Science and Engineering*, 12(3):66-73, 2010.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **An Asymmetric Distributed Shared Memory Model for Heterogeneous Computing Systems**. I. Gelado, J. Stone, J. Cabezas, S. Patel, N. Navarro, W. Hwu. *ASPLOS '10: Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 347-358, 2010.
- **GPU Clusters for High Performance Computing**. V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu. *Workshop on Parallel Programming on Accelerator Clusters (PPAC)*, In Proceedings IEEE Cluster 2009, pp. 1-8, Aug. 2009.
- **Long time-scale simulations of in vivo diffusion using GPU hardware**. E. Roberts, J. Stone, L. Sepulveda, W. Hwu, Z. Luthey-Schulten. In *IPDPS'09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Computing*, pp. 1-8, 2009.
- **High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs**. J. Stone, J. Saam, D. Hardy, K. Vandivort, W. Hwu, K. Schulten, *2nd Workshop on General-Purpose Computation on Graphics Pricessing Units (GPGPU-2), ACM International Conference Proceeding Series*, volume 383, pp. 9-18, 2009.
- **Probing Biomolecular Machines with Graphics Processors**. J. Phillips, J. Stone. *Communications of the ACM*, 52(10):34-41, 2009.
- **Multilevel summation of electrostatic potentials using graphics processing units**. D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

GPU Computing Publications

http://www.ks.uiuc.edu/Research/gpu/

- **Adapting a message-driven parallel application to GPU-accelerated clusters**. J. Phillips, J. Stone, K. Schulten. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, 2008.
- **GPU acceleration of cutoff pair potentials for molecular modeling applications**. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, and W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.
- **GPU computing**. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.
- **Accelerating molecular modeling applications with graphics processors**. J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.
- **Continuous fluorescence microphotolysis and correlation spectroscopy**. A. Arkhipov, J. Hüve, M. Kahms, R. Peters, K. Schulten. *Biophysical Journal*, 93:4006-4017, 2007.