To GPU Synchronize or Not GPU Synchronize?

Prof. Wu FENG

Departments of Computer Science and Electrical & Computer Engineering

VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

What are the Takeaways?

- At the systems software level …
  - How to support *efficient communication between SMs* via barrier synchronization ***on*** the GPU → ***GPU Synchronization***
- At the application level …
  - How to integrate the GPU synchronization capability into real applications: FFT, dynamic programming, and bitonic sort
- From a performance and correctness perspective …
  - How to "improve" the performance of *existing* GPU-optimized applications
  - How to guarantee correctness and its associated cost

Motivation

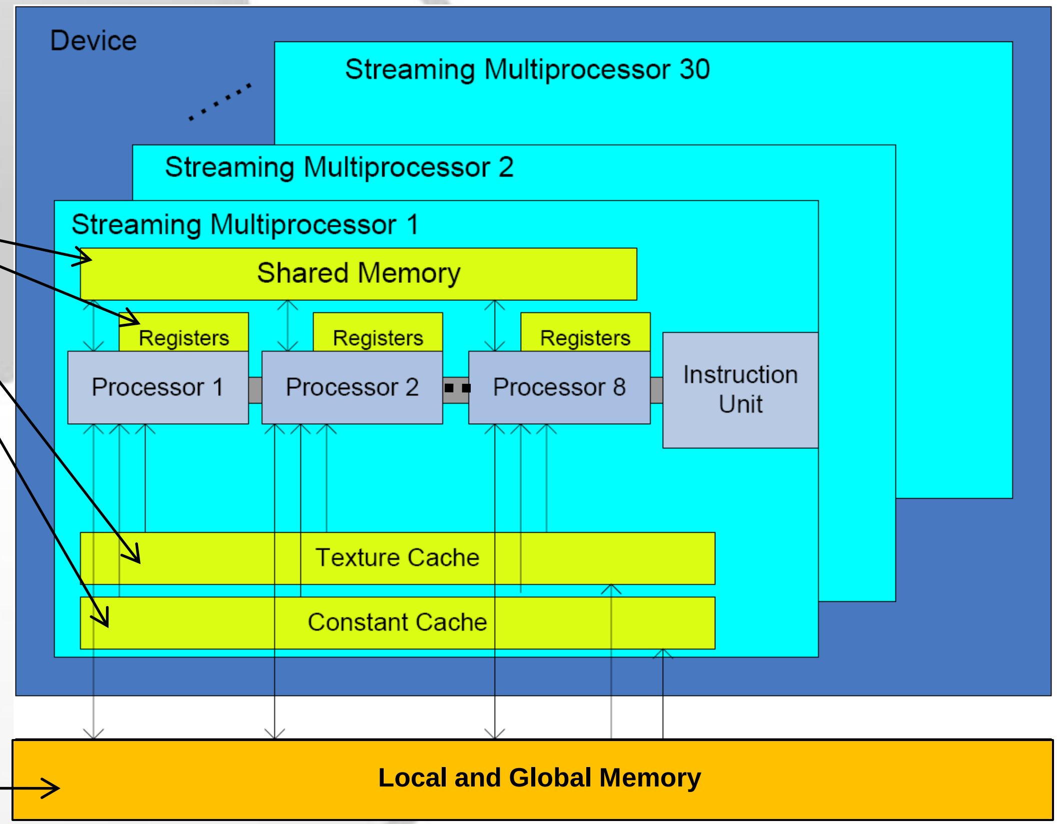
- Only task- or data-parallel algorithms map well to the GPU.
  - No explicit support for *communication between SMs,* i.e., inter-block data communication
- Consecutive kernel launches from CPU serve as an *implicit barrier synchronization* for inter-block communication.
  - How expensive is CPU implicit barrier synchronization?
  - Would barrier synchronization on the GPU be better in support of “more general-purpose computation”?
- To GPU synchronize or not GPU synchronize?

Outline

- Motivation
- Background
  - GTX 280 and CUDA Programming Model
- GPU Synchronization
  - GPU Lock-Based Synchronization
  - GPU Lock-Free Synchronization
- Experimental Results
- Conclusion & Future Work

GTX 280 Architecture

On-chip memory
- Small sizes
- Fast access



Off-chip memory
- Large size
- High access latency

CUDA Programming Model

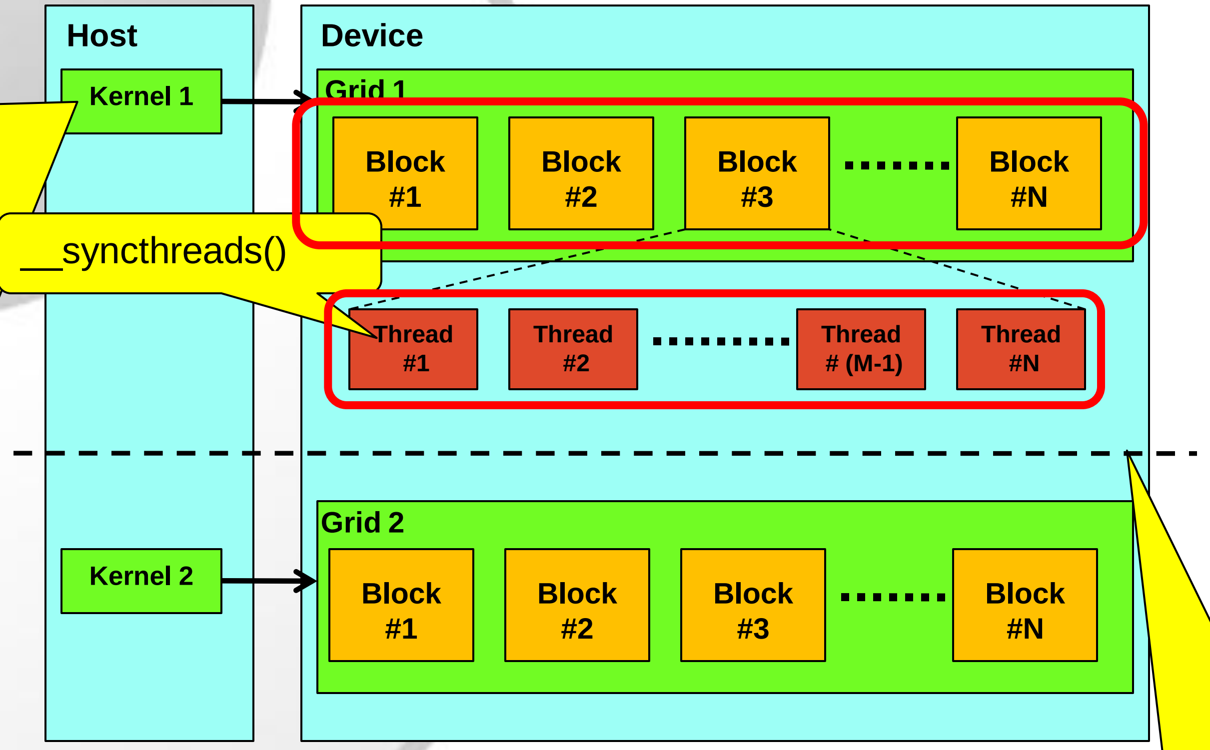
- CUDA: An extension of the C programming language

Kernel: *A global function called from host and executed on device*

•Consists of multiple blocks with each block consisting of multiple threads

•Intra-block sync is implemented with __syncthreads()

•Inter-block sync is implemented via kernel launches



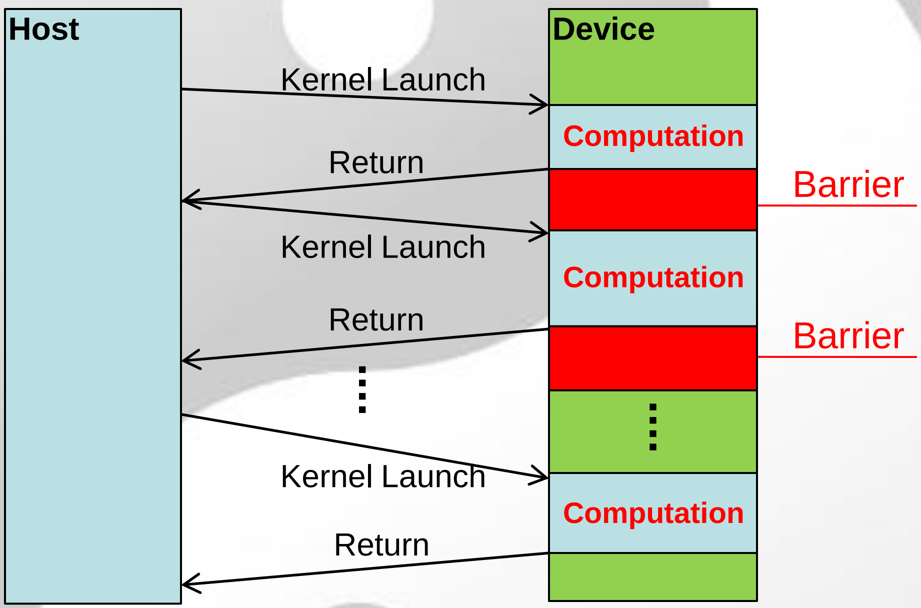
Barrier between two kernel launches

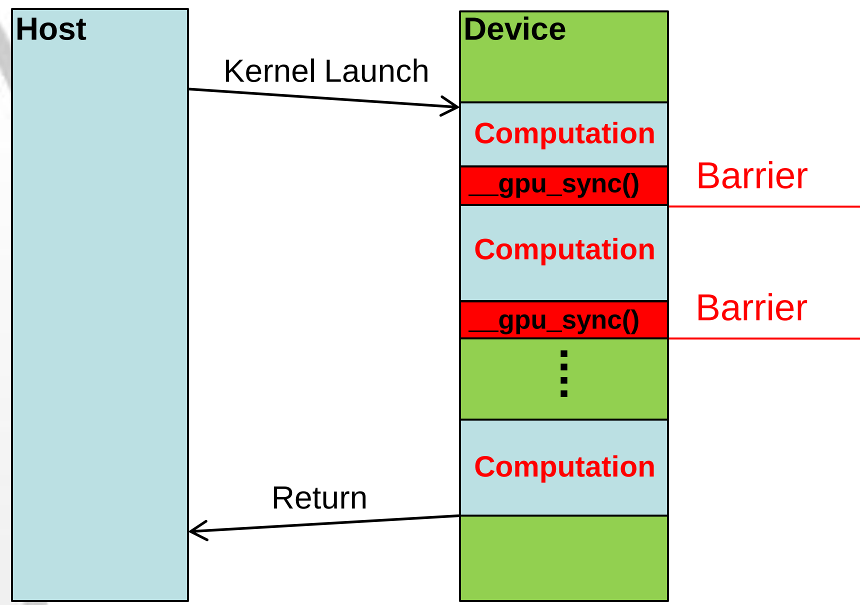
Outline

- Motivation
- Background
  - GTX 280 and CUDA Programming Model
- GPU Synchronization
  - GPU Lock-Based Synchronization
  - GPU Lock-Free Synchronization
- Experimental Results
- Conclusion & Future Work

Types of Barrier Synchronization

CPU Synchronization vs. GPU Synchronization

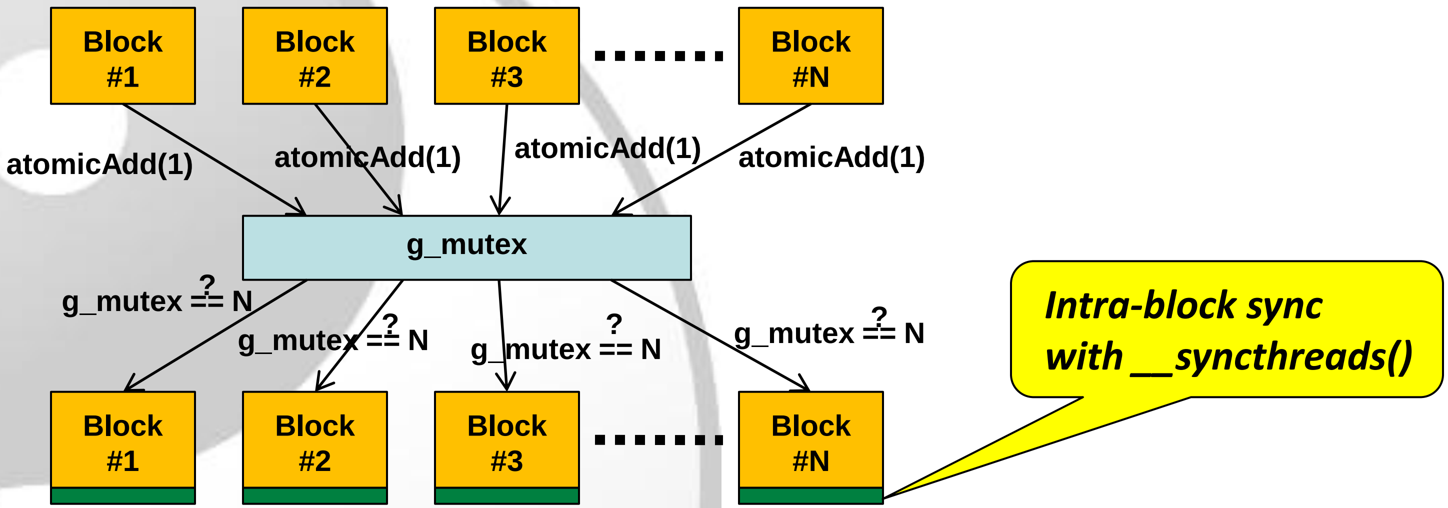




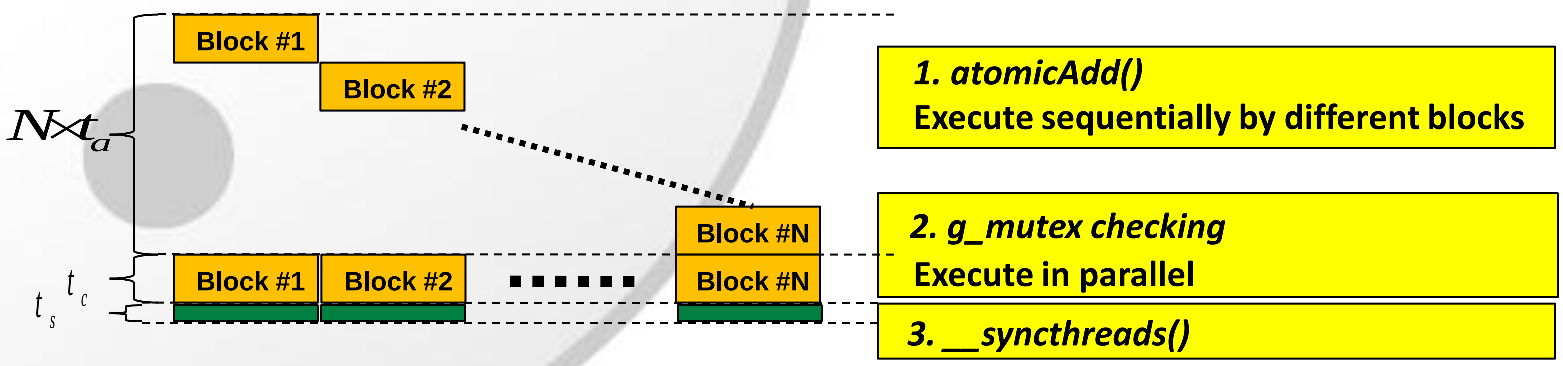

```
for() {
  __kernel_func<<<grid, block>>>();
}
```

```
__global__ void __kernel_func()
{
  for () {
      __device_func();
      __gpu_sync();
  }
}
```

GPU Lock-Based Synchronization

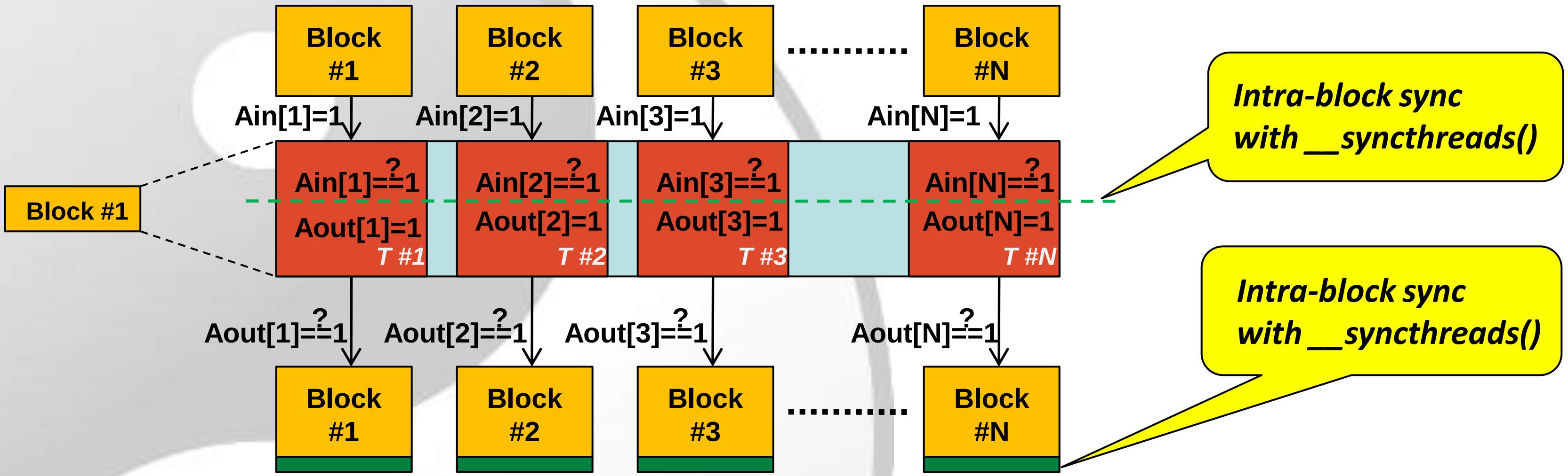


- Time Profile

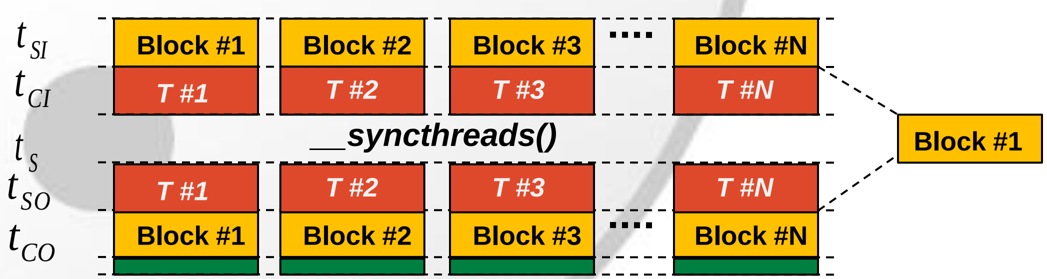


Total synchronization time: $N \times t_a + t_c + t_s$

GPU Lock-Free Synchronization



- Time Profile

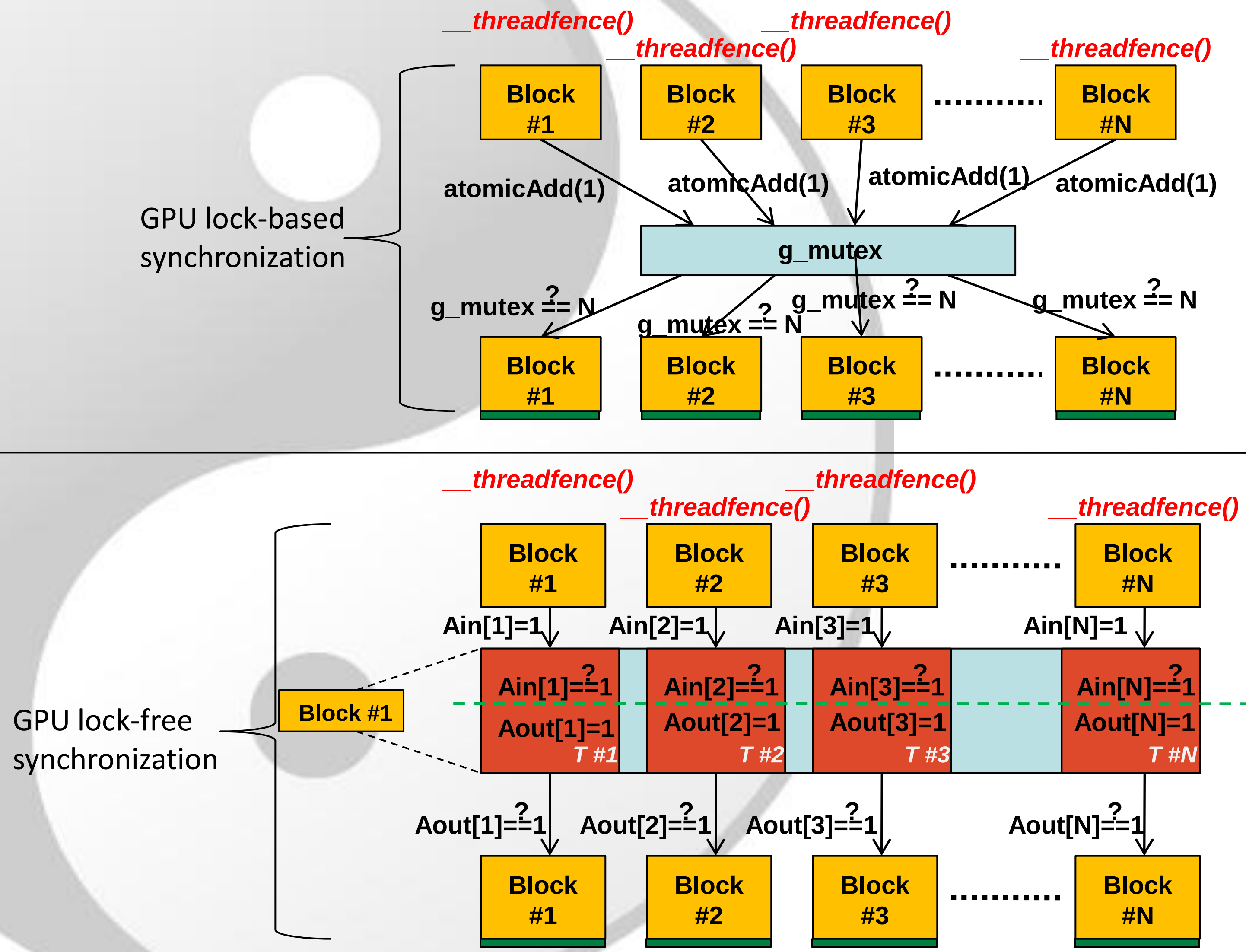


Total synchronization time: [illegible]

1. *Set Ain*
2. *Check Ain*
3. *Intra-block sync*
4. *Set Aout*
5. *Check Aout*
6. *Intra-block sync*

Note: Each step can be executed in parallel by different blocks

Guaranteeing Correctness



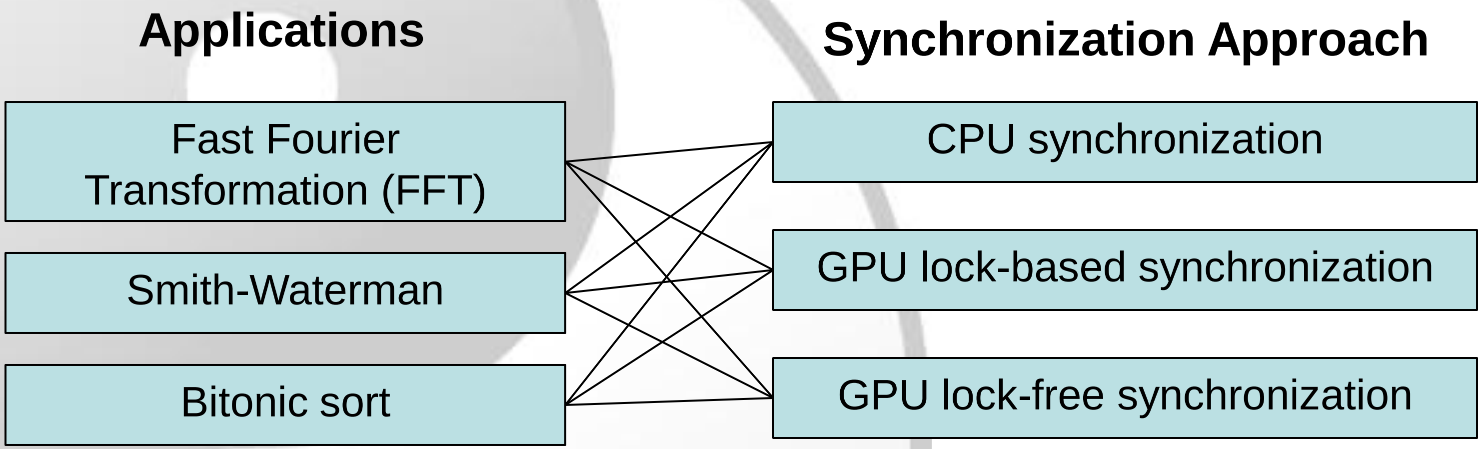
Outline

- Motivation
- Background
  - GTX 280 and CUDA Programming Model
- GPU Synchronization
  - GPU Lock-Based Synchronization
  - GPU Lock-Free Synchronization
- Experimental Results
- Conclusion & Future Work

Experimental Set-Up

- Hardware
  - Host
    - 2.2-GHz Intel Core 2 Duo CPU
    - 2 X 2GB of DDR2 SDRAM
  - Device
    - GTX 280 video card
    - 1024 MB device memory
- Software
  - 64-bit Ubuntu GNU/Linux 8.04
  - NVIDIA CUDA 2.3 SDK toolkit

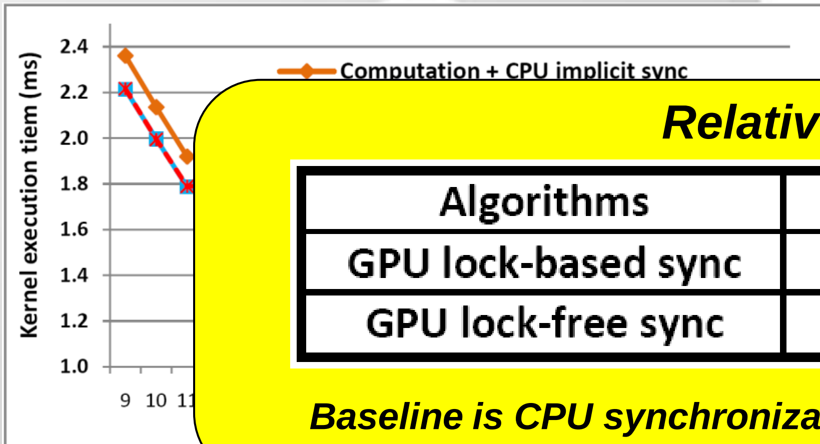
Measurements

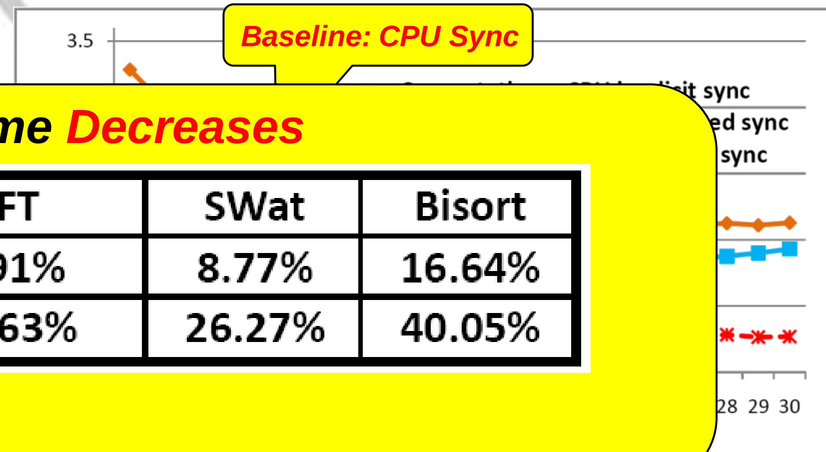


- Execution time **without** __*threadfence()*
- Execution time **with** __*threadfence()*
- Synchronization time percentage (without __threadfence())

Execution Time ***without*** __threadfence()

Kernel Execution Time vs. Number of Blocks in the Kernel





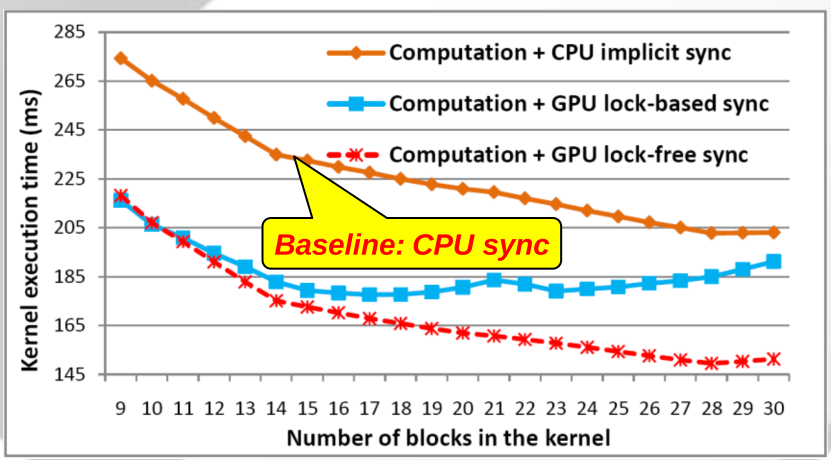
Relative Time Decreases

Algorithms	FFT	SWat	Bisort
GPU lock-based sync	6.91%	8.77%	16.64%
GPU lock-free sync	10.63%	26.27%	40.05%

Baseline is CPU synchronization

FFT

Bitonic sort



Smith-Waterman

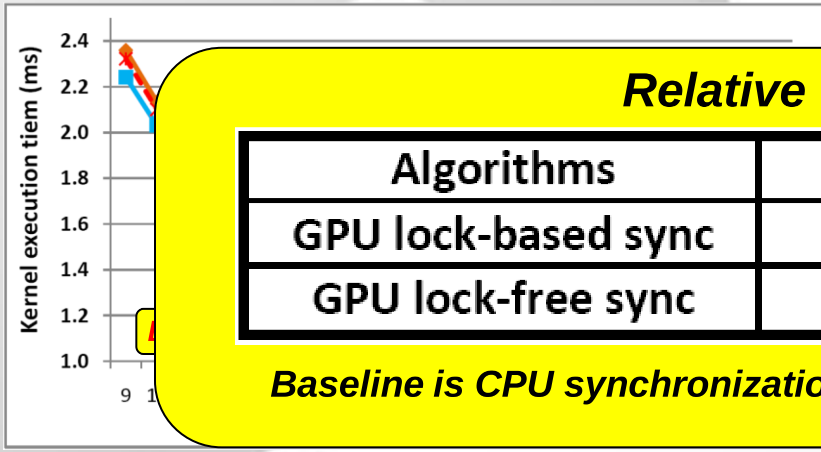
- ***Less time is needed with more blocks in the kernel***
- ***Matrix filling time difference in FFT is smaller than the other two with different sync approaches used***
- ***GPU sync has a better performance than CPU implicit sync, BUT …***

Performance & Correctness: Execution Time w/o __threadfence()

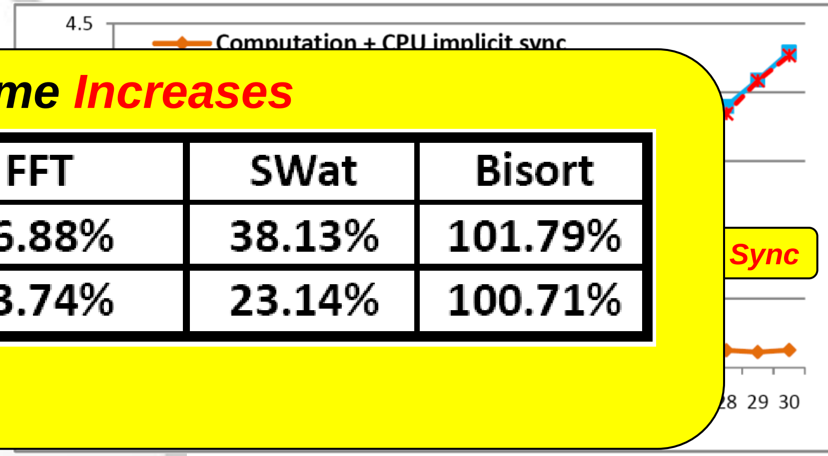
- Performance Improvement (relative to existing GPU)
  - FFT: 10% | Dynamic Programming: 26% | Bitonic Sort: 40%
- Overall Performance Improvement (relative to CPU serial)
  - FFT: 70x | Dynamic Programming: 13x | Bitonic Sort: 24x
- But …
  - Our GPU barrier synchronizations run the risk that writes performed ***before*** our gpu sync() barrier are ***not*** completed by the time the GPU is released from the barrier.
  - syncthreads() can only guarantee writes to shared memory and global memory visible to threads of the ***same*** block, it ***cannot*** do so for threads across different blocks.
  - In practice, highly unlikely the above will ever happen given the amount of time spent spinning at the barrier, but still possible. So, …

Execution Time *with* __threadfence()

Kernel Execution Time vs. Number of Blocks in the Kernel



FFT

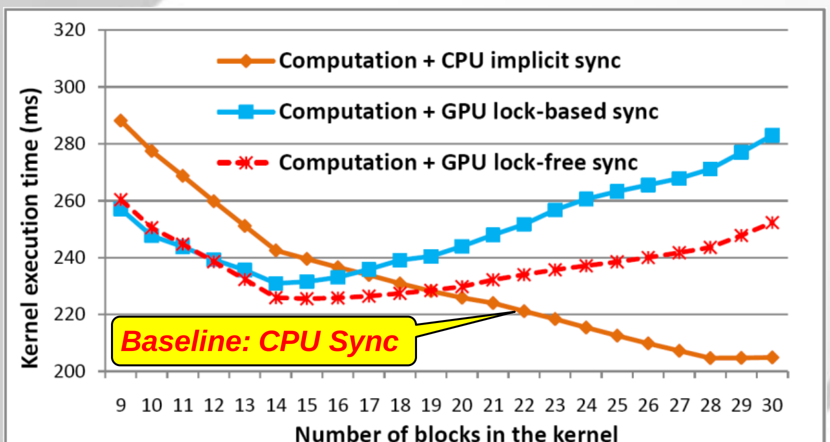


Bitonic sort

Relative Time Increases

Algorithms	FFT	SWat	Bisort
GPU lock-based sync	16.88%	38.13%	101.79%
GPU lock-free sync	13.74%	23.14%	100.71%

Baseline is CPU synchronization

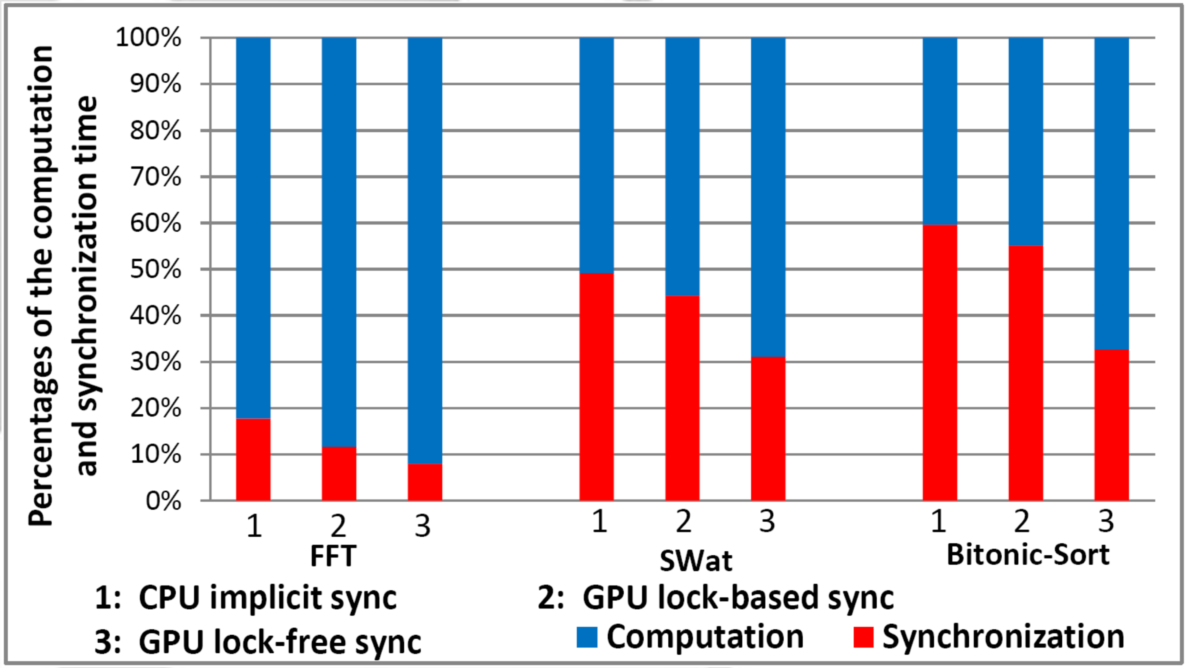


Smith-Waterman

- ***Matrix filling time difference in FFT is smaller than the other two with different sync approaches used***
- ***With __threadfence() called, performance of GPU sync is worse than CPU implicit sync***

Percentage of Time Spent Synchronizing

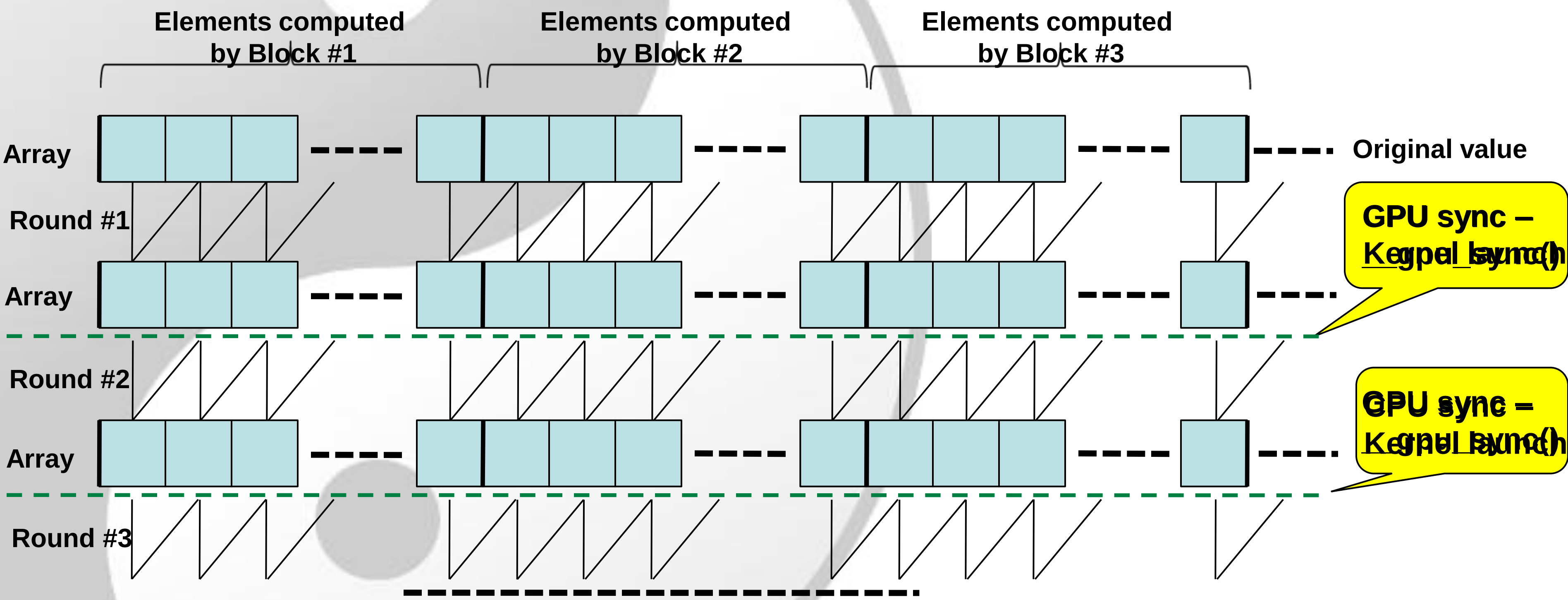
Synchronization Time Percentages (without __threadfence())



- ***% time to sync in FFT is lower than the other two algorithms***
- ***Sync time percentages of SWat and bitonic sort are more than 50% with CPU sync***
- ***% time to GPU sync is lower than that of CPU implicit sync***

Profiling the Synchronization Time

- Micro-benchmark – Compute average of two floats 10,000 times



Decomposing Synchronization Time

- Ways to compute each time component
  - Only kernel execution time can be recorded
  - Indirect method
  - Use GPU lock-based synchronization as the example
  - Synchronization time can be represented as $N \times t_a + t_c + t_s$
  - Times that can be recorded directly

$N \times t_a$: Kernel consisting of only atomicAdd

t_{com} : Kernel consisting of only computation

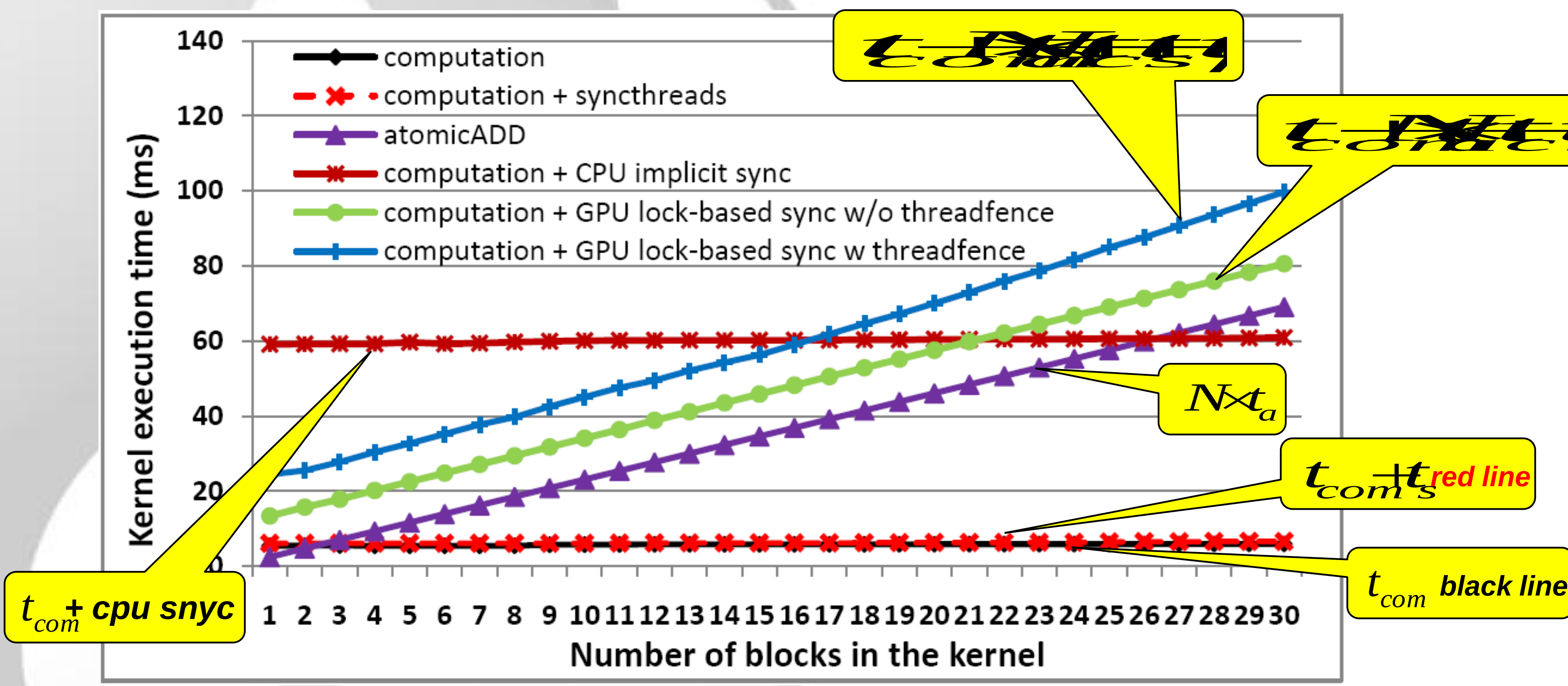
$t_{com} + t_s$: Kernel with computation and __syncthreads()

$t_{com} + N \times t_a + t_c + t_s$: Kernel with GPU lock-based synchronization

$t_{com} + N \times t_a + t_c + t_s + t_f$: Kernel with GPU lock-based synchronization (__threadfence())

Profile of Synchronization Time

- Results



for 10,000 times execution

Outline

- Motivation
- Background
  - GTX 280 and CUDA Programming Model
- GPU Synchronization
  - GPU Lock-Based Synchronization
  - GPU Lock-Free Synchronization
- Experimental Results
- Conclusion & Future Work

Conclusion

- At the systems software level …
  - How to support *efficient communication between SMs* via barrier synchronization ***on*** the GPU → ***GPU Synchronization***
- At the application level …
  - How to integrate the GPU synchronization capability into real applications: FFT, dynamic programming, and bitonic sort
- From a performance and correctness perspective …
  - How to "improve" the performance of *existing* GPU-optimized applications
  - How to guarantee correctness and its associated cost
- *__threadfence guarantees correctness but needs to be optimized to support GPU synch. Is Fermi the answer?!*

Conclusion

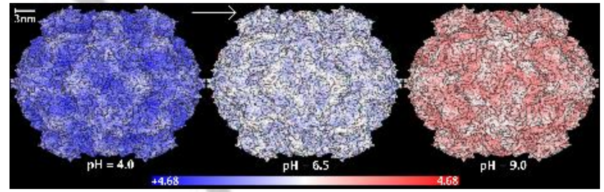
- To GPU synchronize or not GPU synchronize?
  - GTX 280 / Tesla C1060 / Tesla S1080: Do *NOT* GPU synchronize.
  - Fermi? Likely GPU synchronize?
- Next steps?
  - Efficient inter-block synchronization via NVIDIA Fermi
  - Efficient inter-block synchronization in OpenCL
  - Automated tool to transform CPU sync to GPU sync
- For more information
  - "On the Robust Mapping of Dynamic Programming onto a Graphics Processing Unit," *15th Int'l Conf. on Parallel & Distributed Systems*, 12/2009.
  - "Inter-Block GPU Communication via Fast Barrier Synchronization," Technical Report TR-09-19, Computer Science, Virginia Tech, 10/2009.

Yawn …

- Where are the massive speed-ups and cool pictures?

Electrostatic Potential for Molecular Dynamics

Viral Capsid



Processor + Optimization	Execution Time (seconds)	Speed-Up
CPU Serial	36,360	-
GPU + Kernel Split + Multi-Level HCP	0.37	98,270x

- Visit the Supermicro booth, i.e., behind you, or go to http://www.youtube.com/watch?v=zPBFenYg2Zk

Contact: Prof. Alexey Onufriev for info on the science!

Wu FENG, Ph.D.

feng@cs.vt.edu

SyNeRGy Laboratory

http://synergy.cs.vt.edu/

SUPERCOMPUTING in SMALL SPACES

http://sss.cs.vt.edu/

THE GREEN 500

http://www.green500.org/

mpiBLAST

http://www.mpiblast.org/