A Study of Persistent Threads Style Programming Model for GPU Computing

GTC 2012 | San Jose

Kshitij Gupta [/shi/ /tij/]
UC Davis

A Study of Persistent Threads Style GPU Programming for GPGPU Workloads

InPar 2012 | San Jose

Kshitij Gupta, Jeff A. Stuart, John D. Owens
UC Davis

Outline

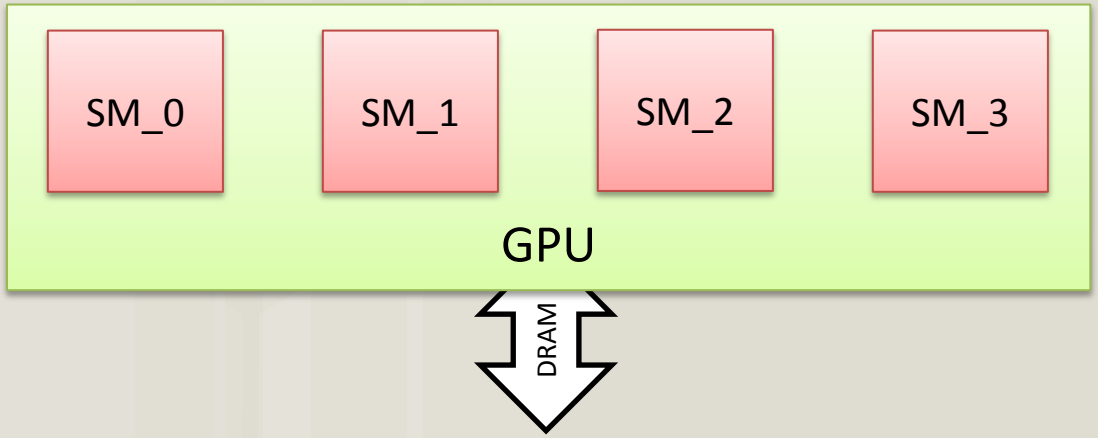
- GPGPU Programming (“nonPT”)
  - Limitations
- Introduction to “PT”
- Use Cases
- Observations/Discussion

NOTE:
(CUDA | OpenCL) Terminology

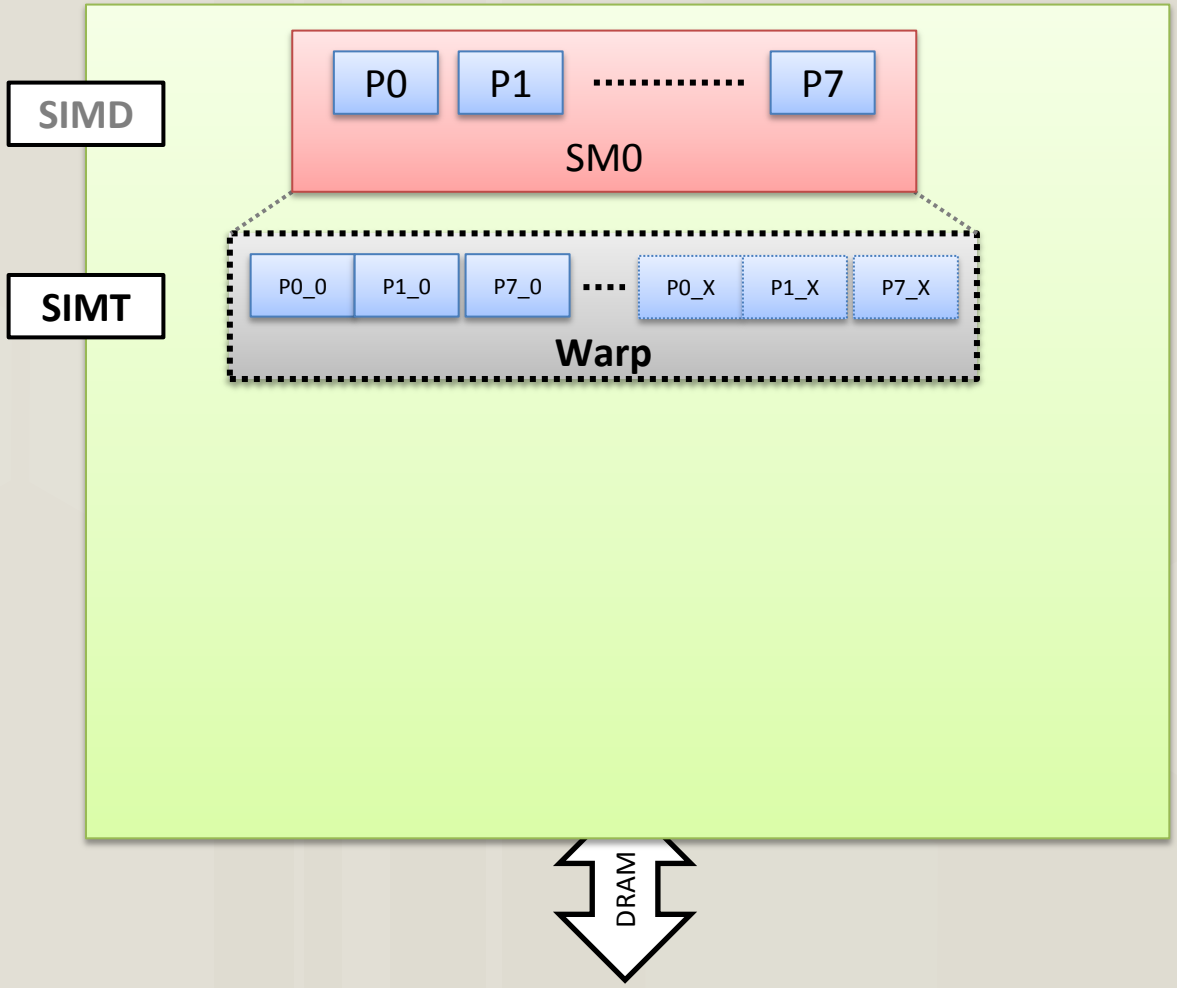
CUDA	OpenCL
Thread	Work item
Warp	--
Thread block	Work group
Grid	Index space
Local memory	Private memory
Shared memory	Local memory
Global memory	Global memory
Scalar core	Processing element
Multi-processor (SM)	Compute unit

*** We will use CUDA terminology, but the same† discussion can be extended to OpenCL**

Preliminaries:
GPGPU Programming Hierarchy

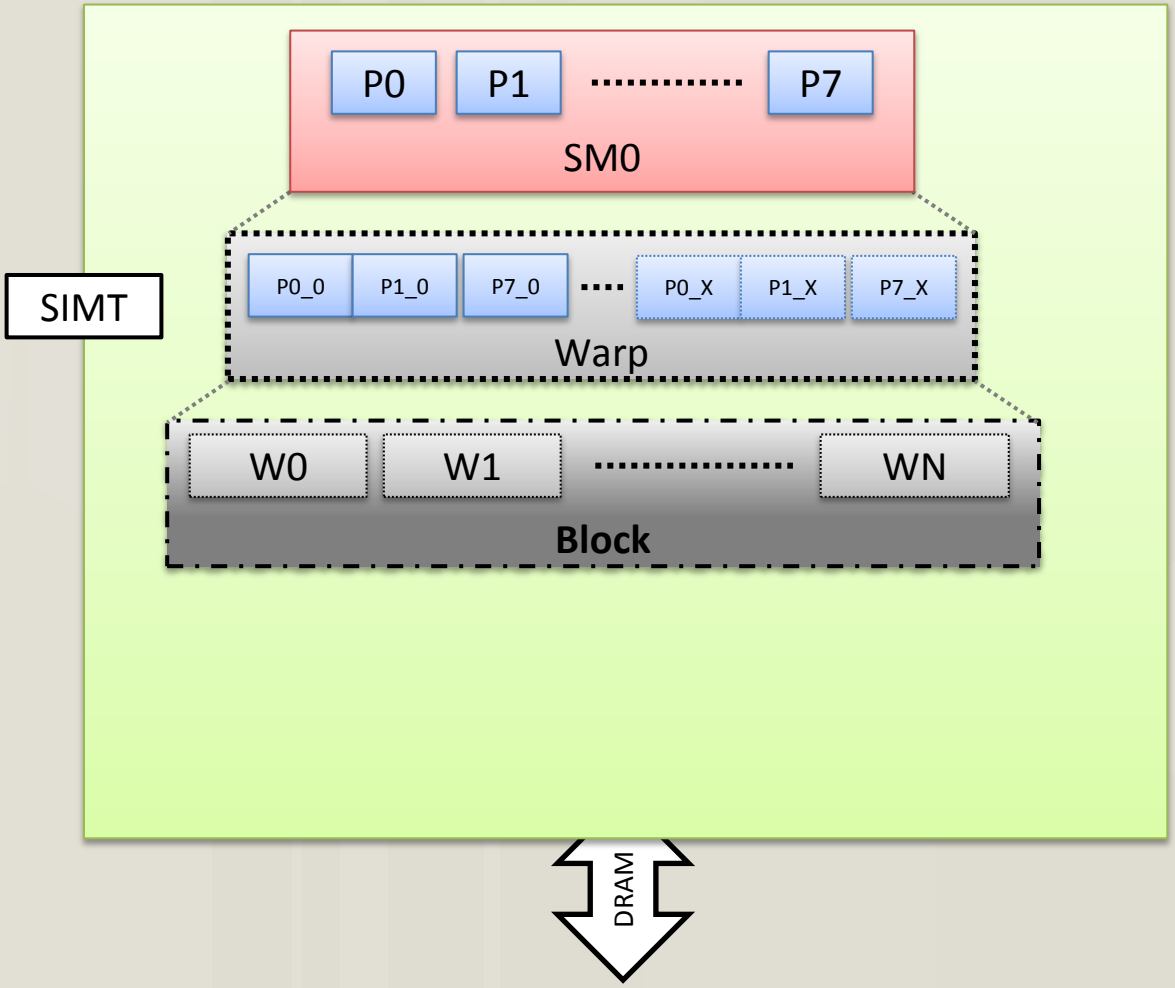


Preliminaries:
GPGPU Programming Hierarchy



Virtualize

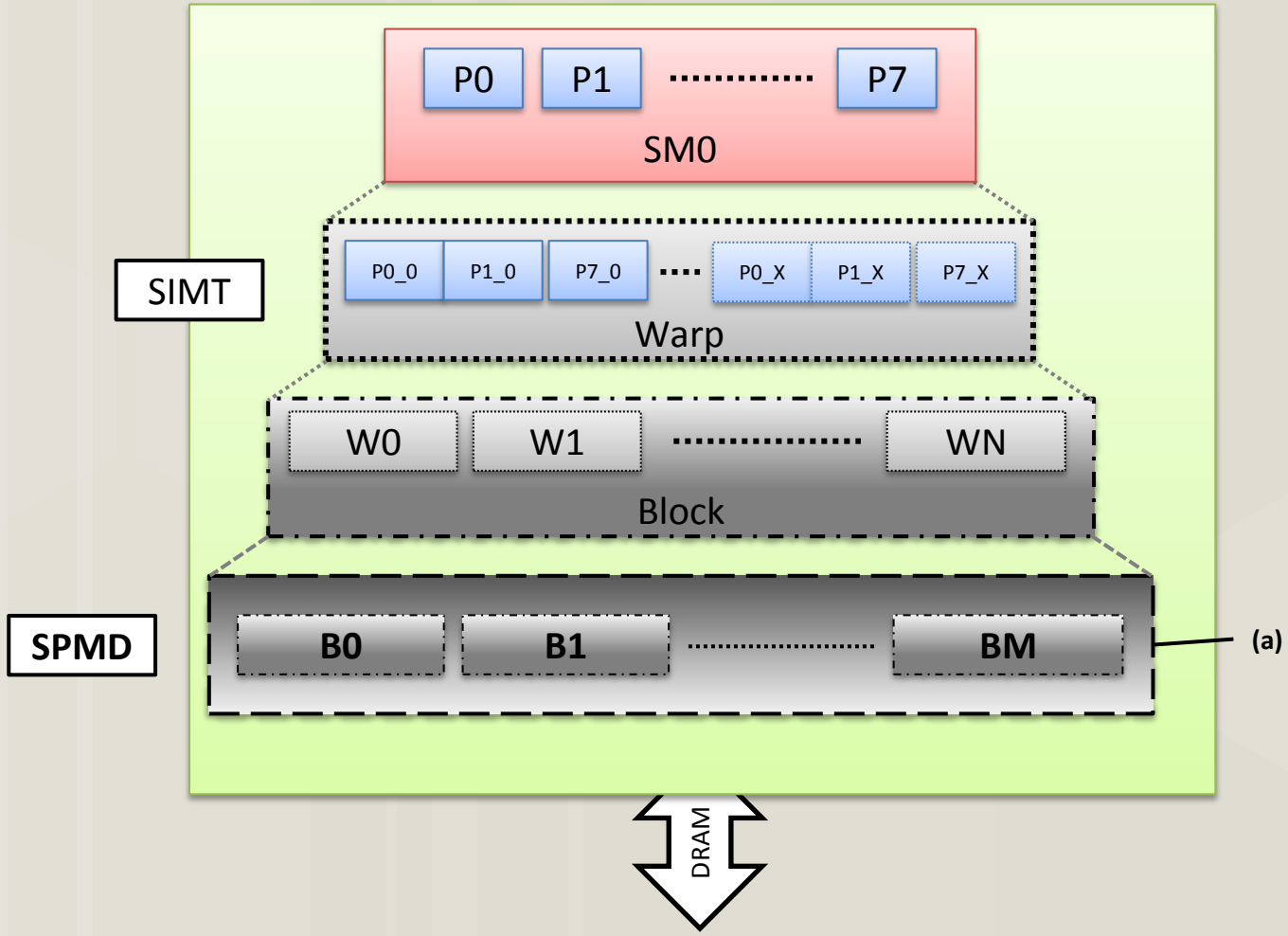
Preliminaries:
GPGPU Programming Hierarchy



Virtualize

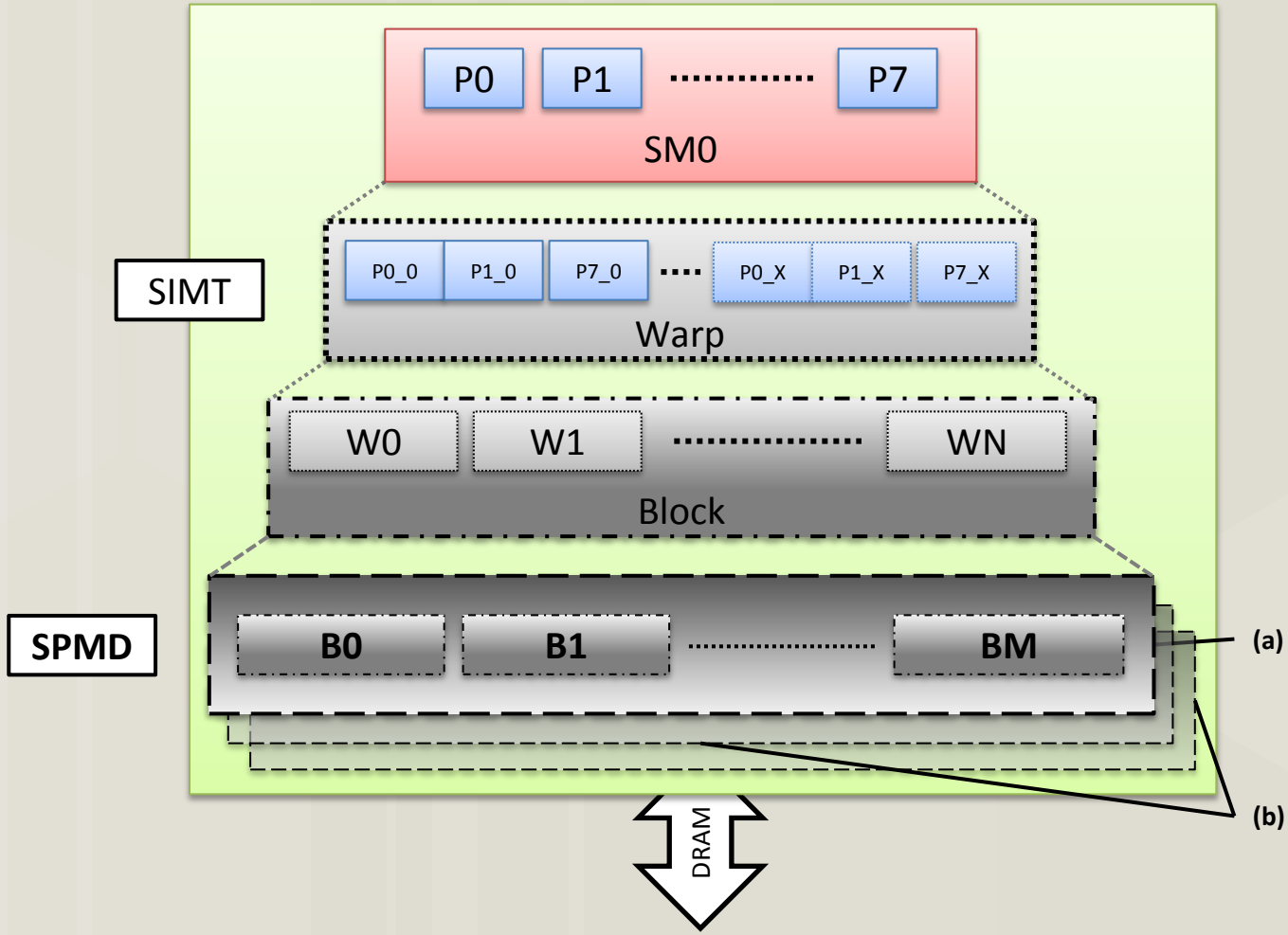
VirtualizE

Preliminaries:
GPGPU Programming Hierarchy



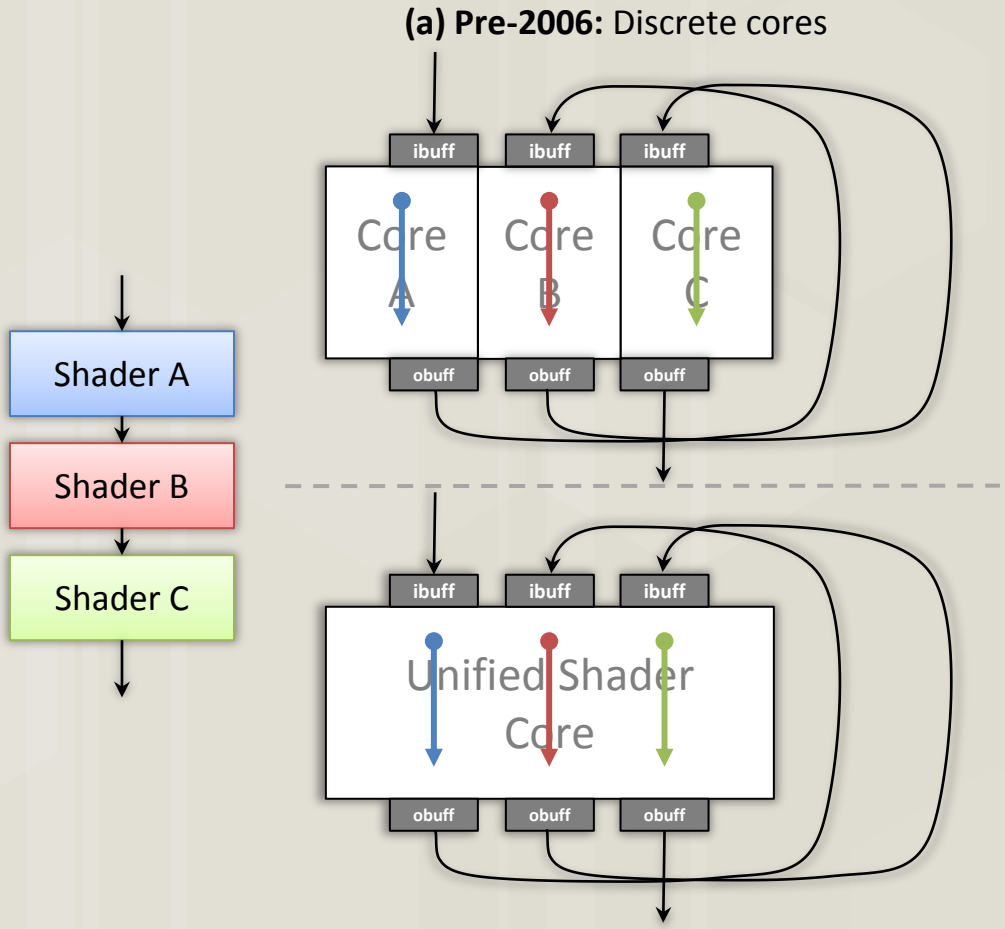
Virtualize | VirtualizE | ViRtUaLiZe

Preliminaries:
GPGPU Programming Hierarchy



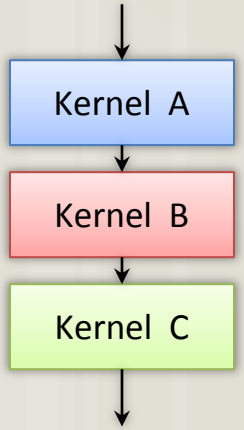
Virtualize | VirtualizE | ViRtUaLiZe | **VIRTUALIZE!!**

Preliminaries: Workload Evolution

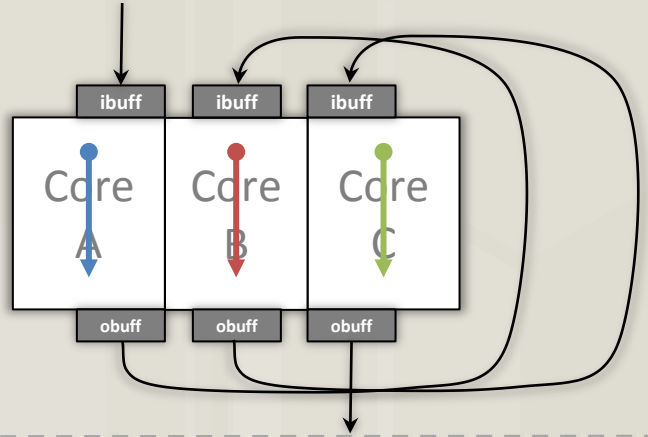


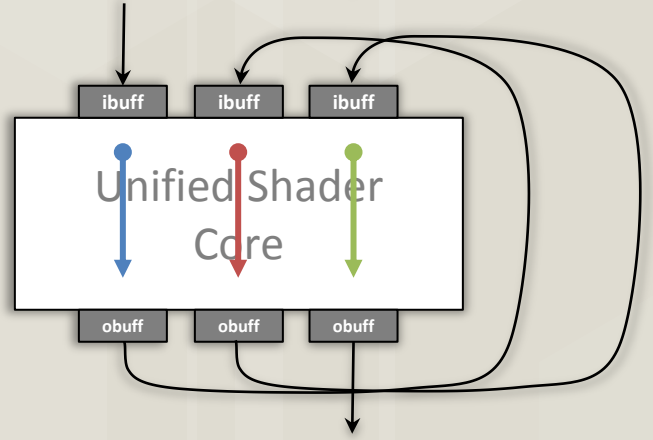
(b) 2006: Stream programming – CUDA architecture with unified cores; along with 'C for CUDA'

Preliminaries: Workload Evolution



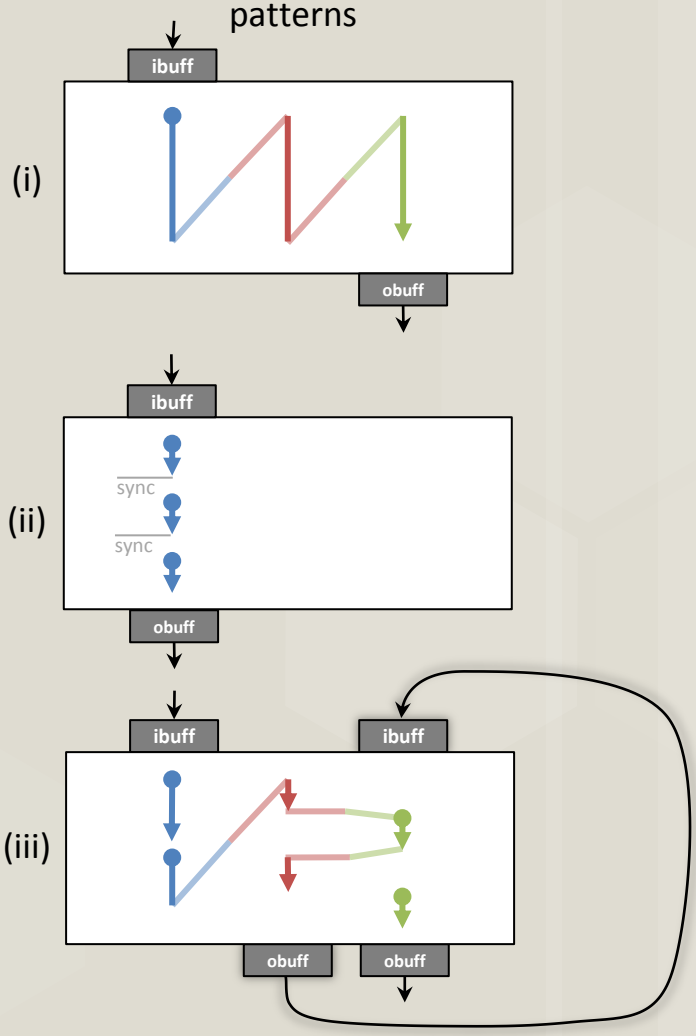
(a) Pre-2006: Discrete cores





(b) 2006: Stream programming – CUDA architecture with unified cores; along with 'C for CUDA'

(c) Today: A sample of irregular workload patterns

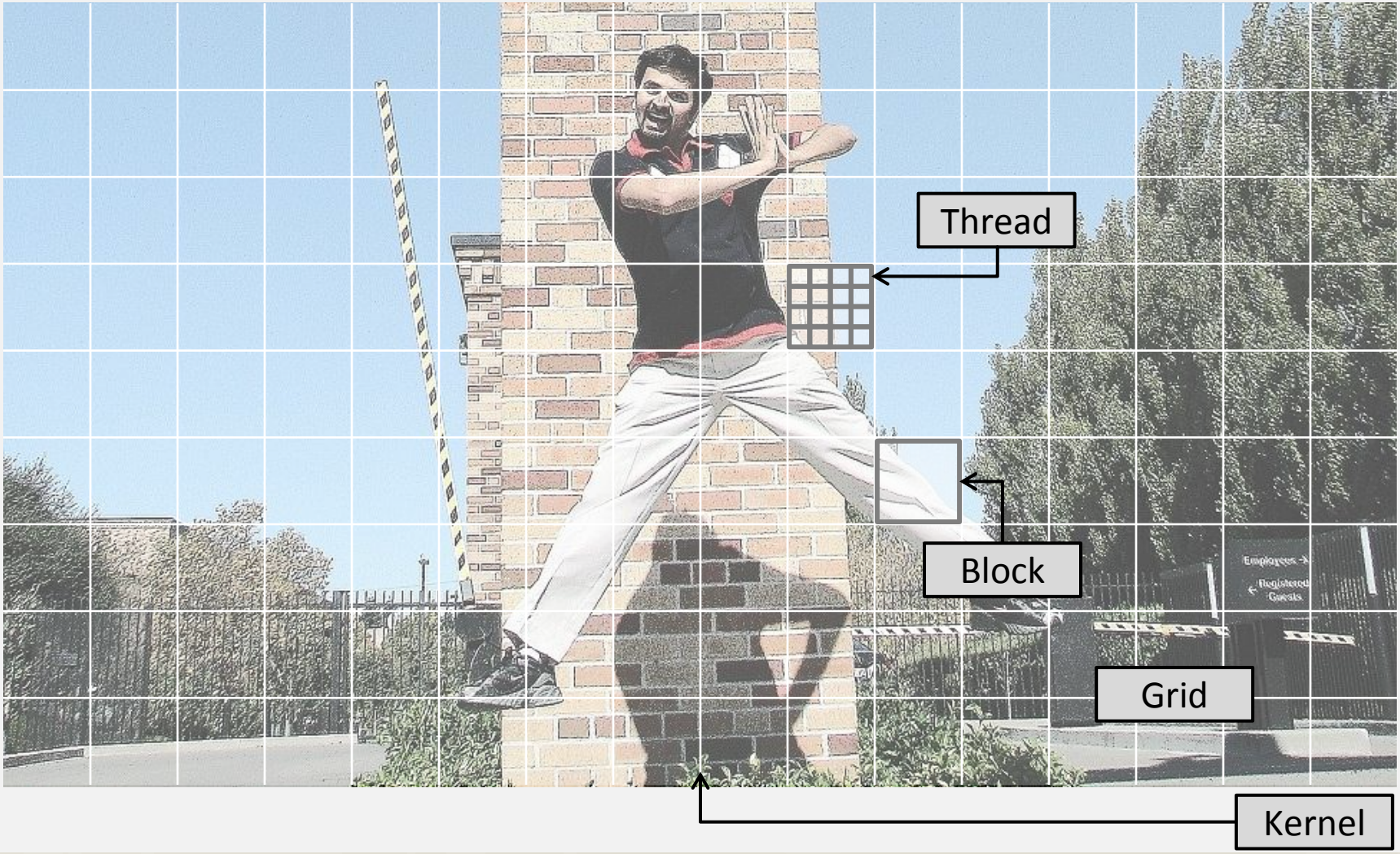


Core GPGPU Programming Characteristics (& Limitations)

1. Host-Device Interface
   - Master-slave processing
   - Kernel size
2. Device-side Properties
   - Lifetime of a Block
   - Hardware Scheduler
   - Block State
3. Memory Consistency
   - Intra-block
   - Inter-block
4. Kernel Invocations
   - Producer-consumer
   - Spawning kernels

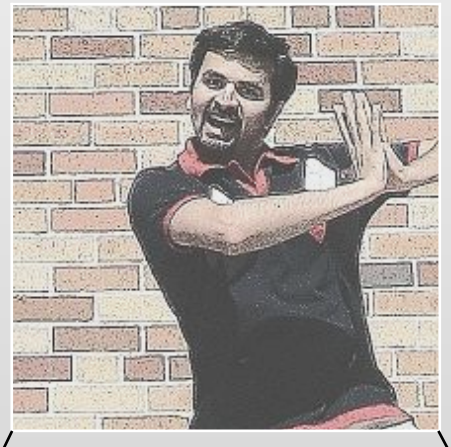
Irregular workloads

Preliminaries:
Example – Image Processing (outside Pixar HQ)

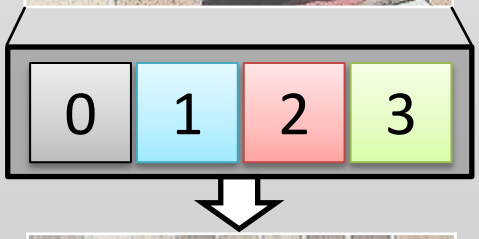


A sample image of 128x128 pixels divided into 16 blocks

Input Image



GPU



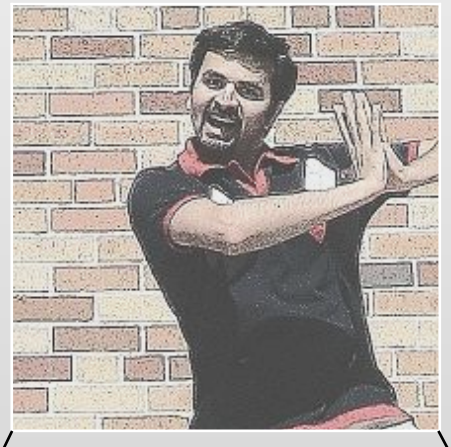
Output Image

vertical motion blur;
processed on a 4-SM GPU

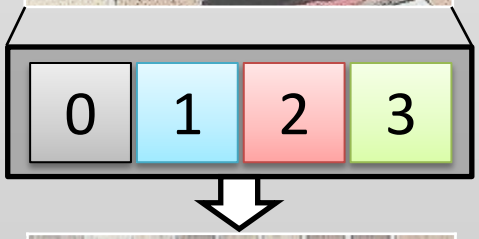
A sample image of 128x128 pixels divided into 16 blocks

nonPT illustration

Input Image



GPU



Output Image

vertical motion blur; processed on a 4-SM GPU

16 blocks mapped to the 4 SMs in random order

Software view

Hardware view

Persistent Threads: Properties

1. Maximal Launch – *A kernel uses only as many threads as can be concurrently scheduled on the SMs*

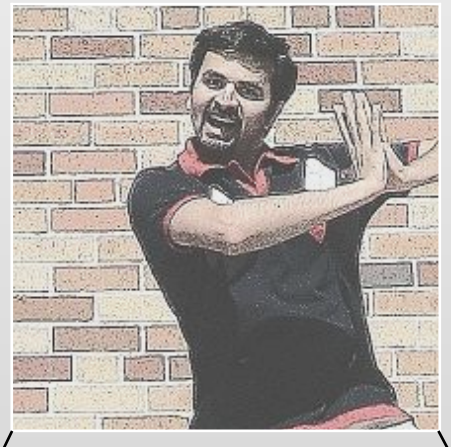
2. Software, not hardware, schedules work

A sample image of 128x128 pixels divided into 16 blocks

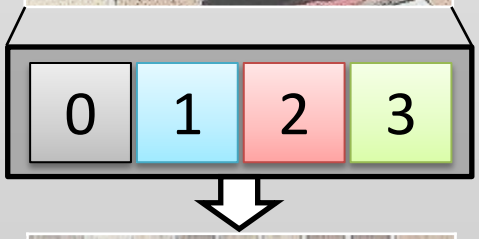
nonPT illustration

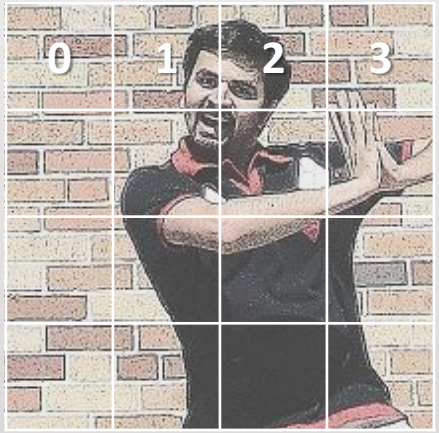
PT illustration

Input Image



GPU



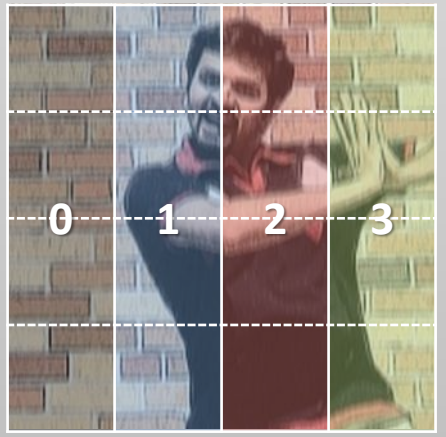


Software view

Output Image

vertical motion blur; processed on a 4-SM GPU

16 blocks mapped to the 4 SMs in random order



4 SMs → 4 thread groups

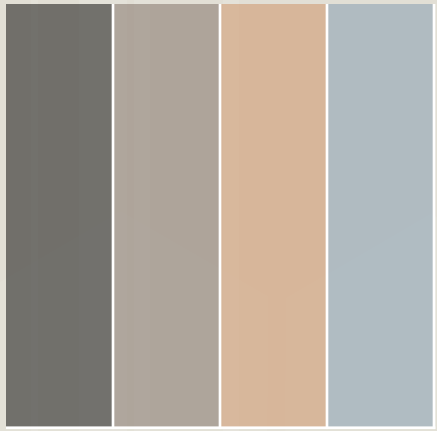
Hardware view

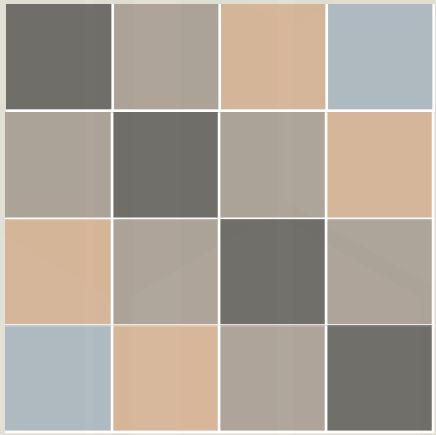
Persistent Threads: Properties

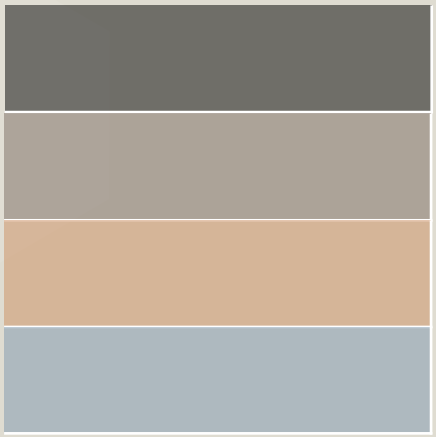
1. Maximal Launch – *A kernel uses only as many threads as can be concurrently scheduled on the SMs*
   - **Thread-group** (v/s thread-block)
     - Upper-bound: maximal launch
     - Lower-bound: 1

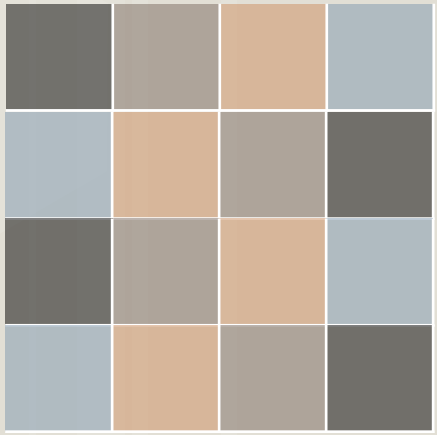
2. Software, not hardware, schedules work
   - **Work-queues**
     - Several optimizations possible
     - In his paper – single global FIFO

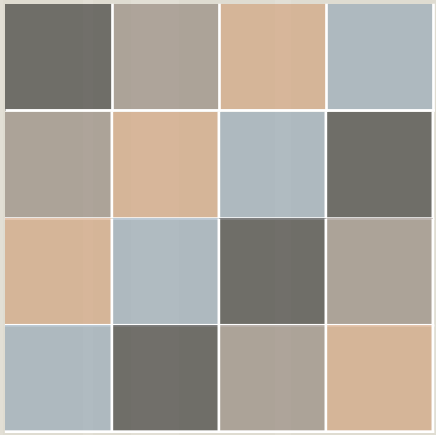
Common Communication Patterns

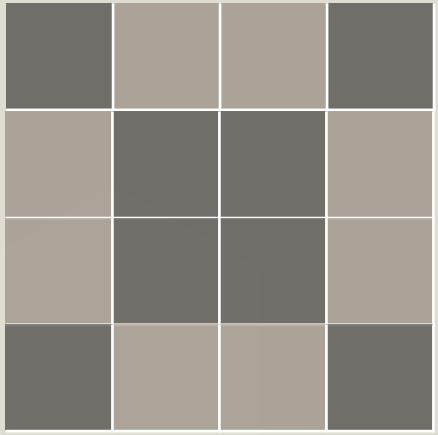












- Linear
- Diagonal
- Zig-Zag
- Scanline
- Wavefront
- Pinwheel
- Checker
- .
- .
- .

Use Cases

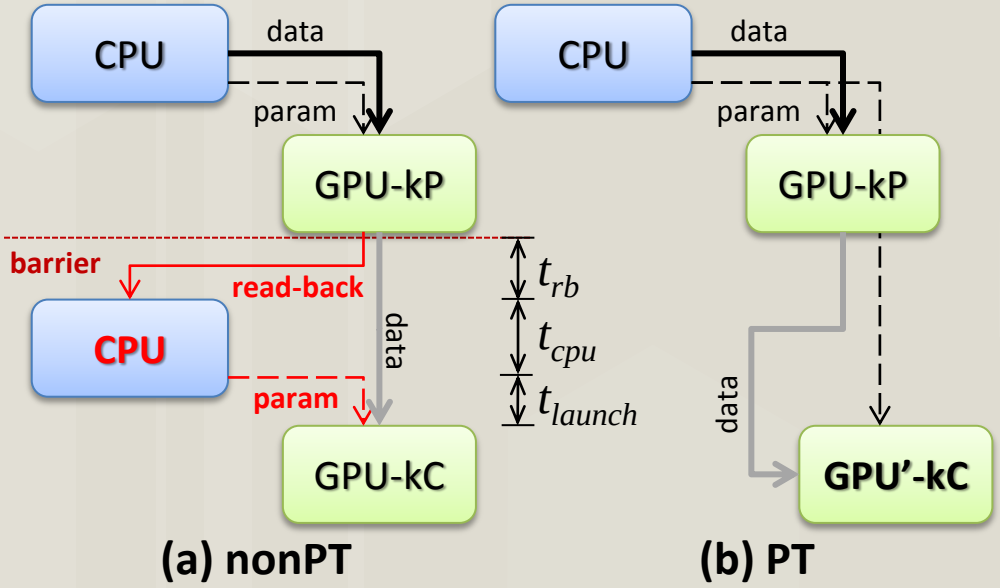
- μ-kernel benchmarks
- Workload comprises of FMAs
- Nvidia GeForce GTX295

PT Use Cases

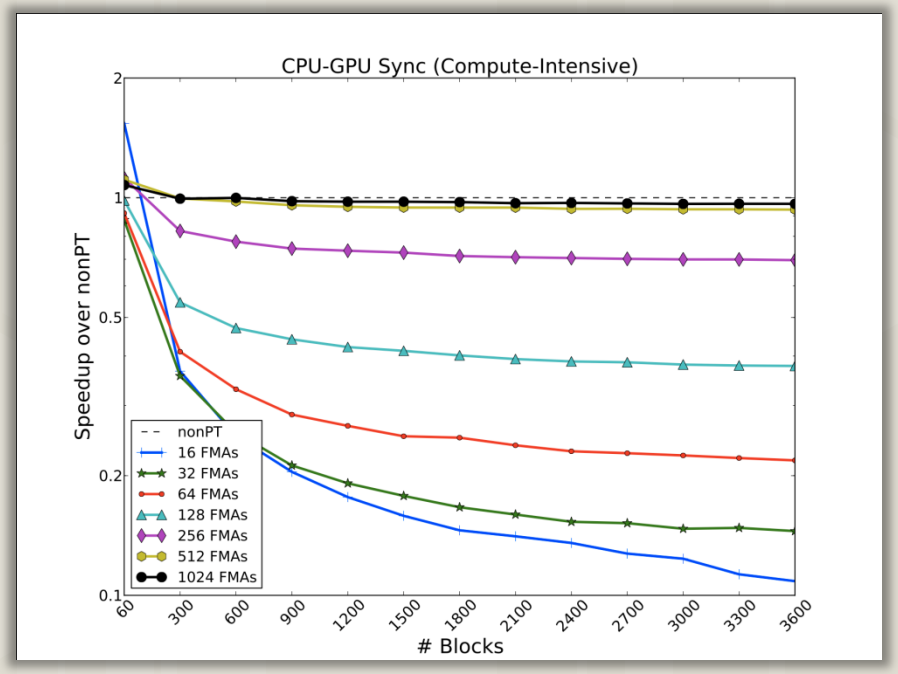
#	Use Case	Scenario	Advantage of Persistent Threads
1	**CPU-GPU Synchronization**	Kernel A produces a variable amount of data that must be consumed by Kernel B	nonPT implementations require a round-trip communication to the host to launch Kernel B with the exact number of blocks corresponding to work items produced by Kernel A.
2	**Load Balancing**	Traversing an irregularly-structured, hierarchical data structure	PT implementations build an efficient queue to allow a single kernel to produce a variable amount of output per thread and load balance those outputs onto threads for further processing.
3	**Maintaining Active State**	A kernel accumulates a single value across a large number of threads, or Kernel A wants to pass data to Kernel B through shared memory or registers	Because a PT kernel processes many more items per block than a nonPT kernel, it can effectively leverage shared memory across a larger block size for an application like a global reduction.
4	**Global Synchronization**	Global synchronization within a kernel across workgroups	In a nonPT kernel, synchronizing across blocks within a kernel is not possible because blocks run to completion and cannot wait for blocks that have not yet been scheduled. The PT model ensures that all blocks are resident and thus allows global synchronization.

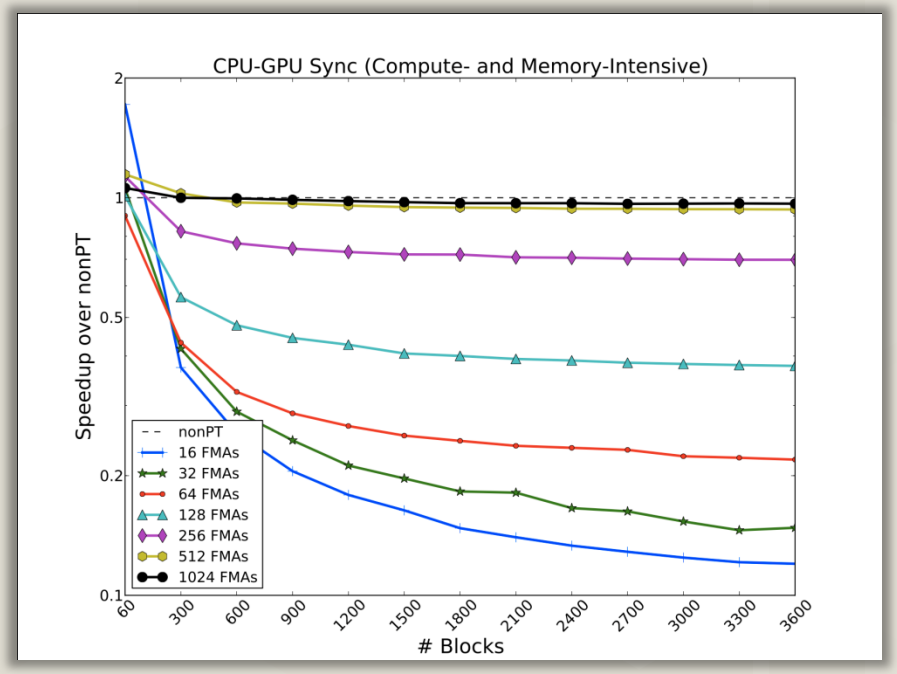
Use Case #1: CPU-GPU Synchronization

Scenario	Advantage of Persistent Threads
Kernel A produces a variable amount of data that must be consumed by Kernel B	nonPT implementations require a round-trip communication to the host to launch Kernel B with the exact number of blocks corresponding to work items produced by Kernel A.



Use Case #1: CPU-GPU Synchronization

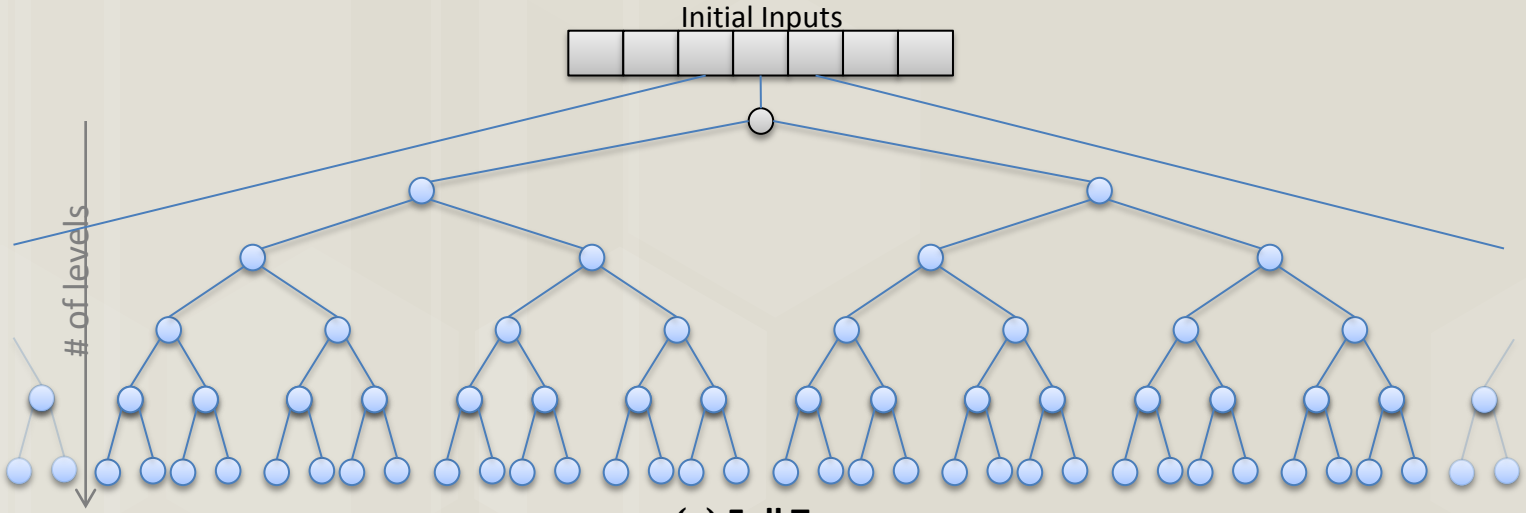




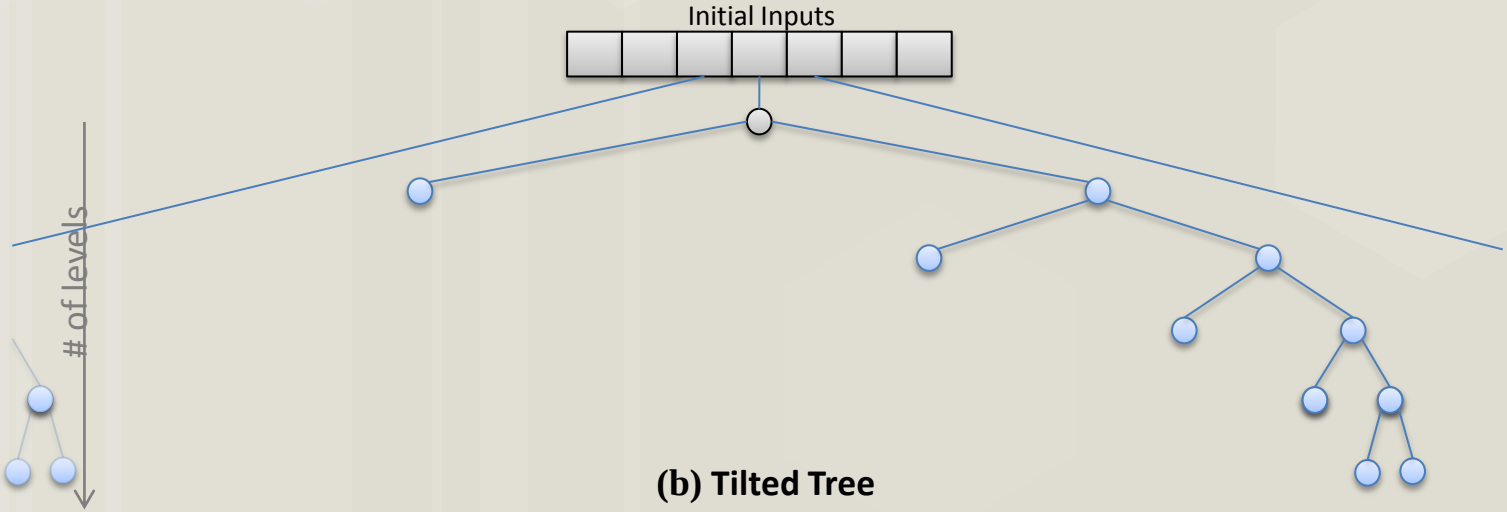
Use Case #2: Load Balancing/Irregular Parallelism

Scenario	Advantage of Persistent Threads
Traversing an irregularly-structured, hierarchical data structure	PT implementations build an efficient queue to allow a single kernel to produce a variable amount of output per thread and load balance those outputs onto threads for further processing.

Use Case #2: Workload Illustration – Tree(s)

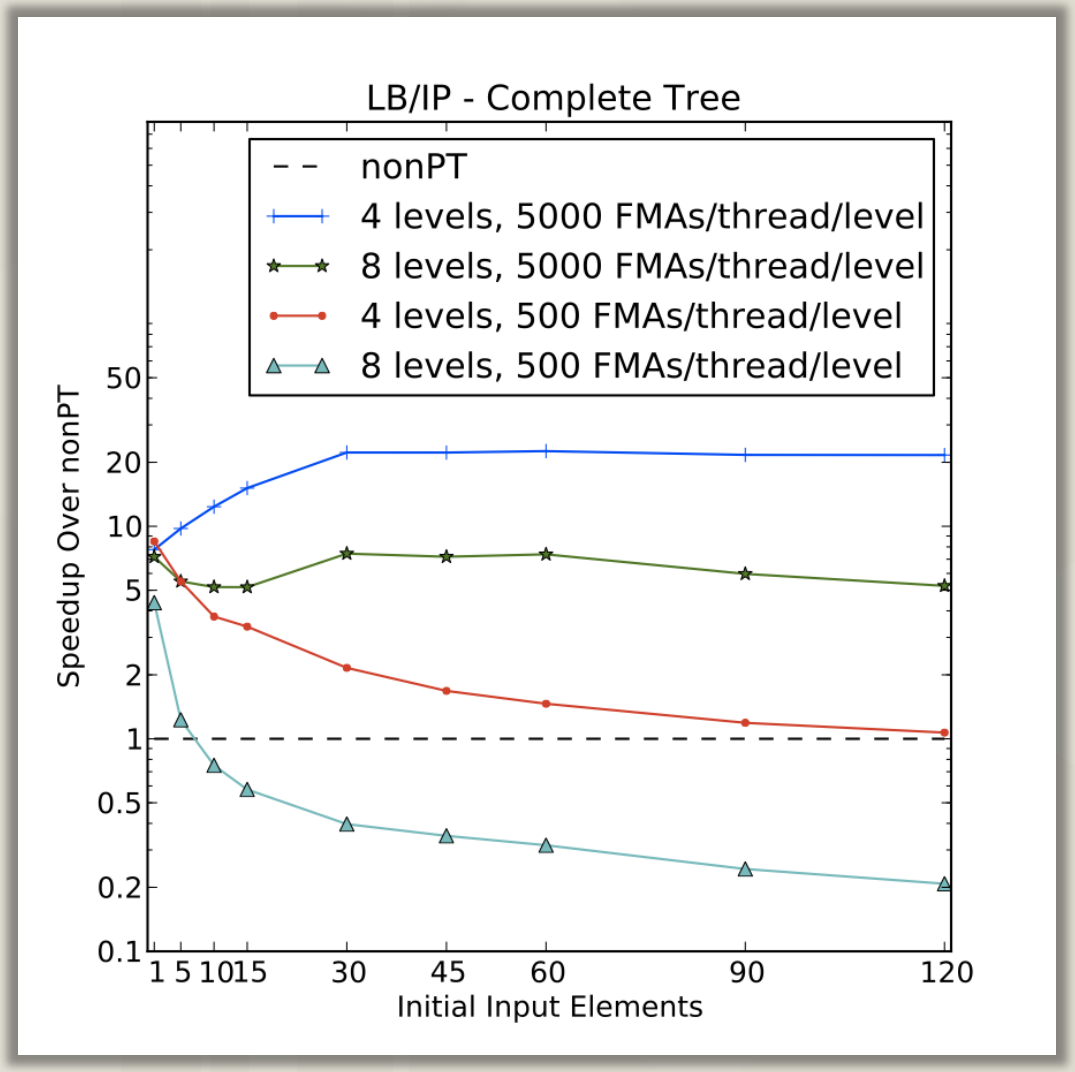


(a) Full Tree

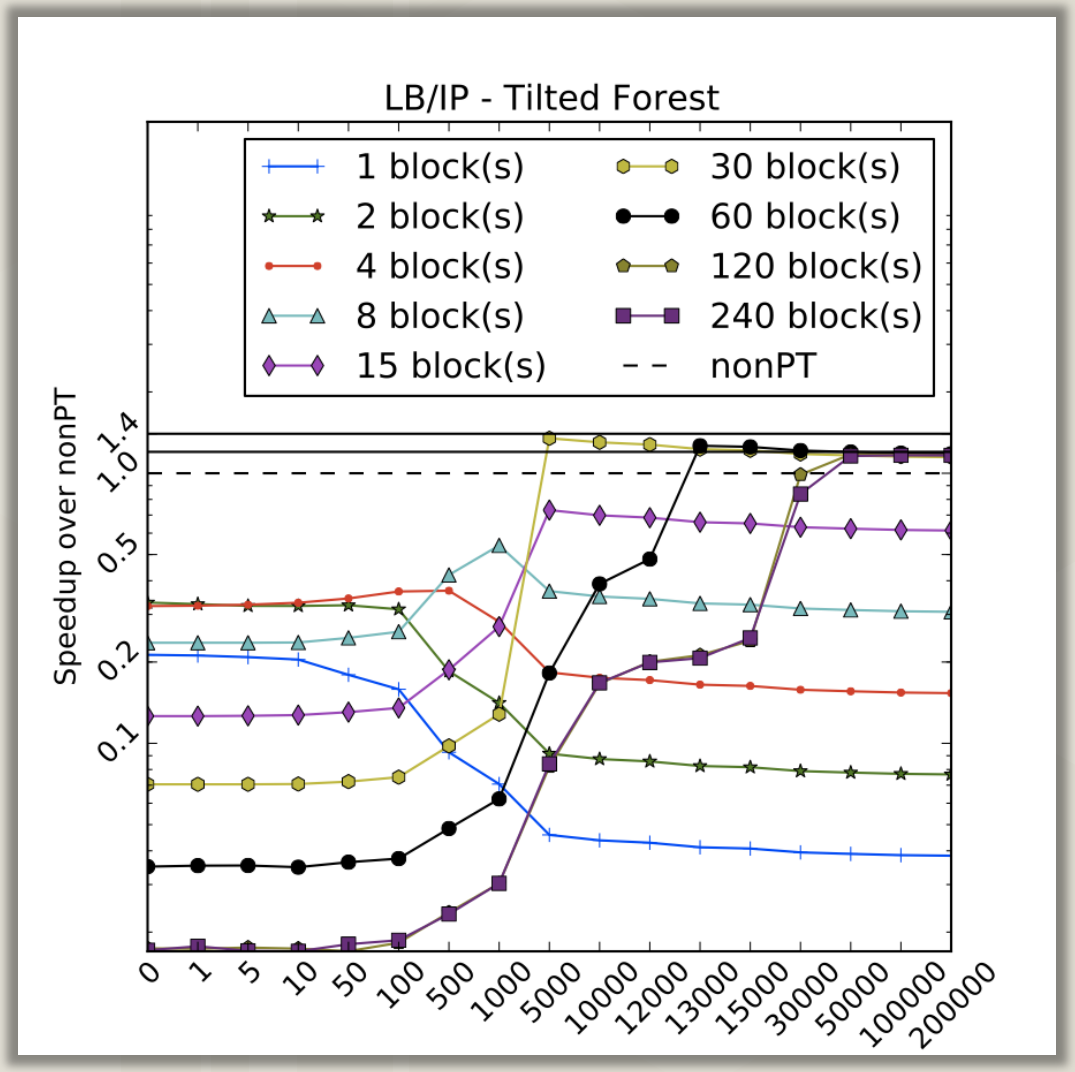


(b) Tilted Tree

Use Case #2:
Workload – Complete Tree

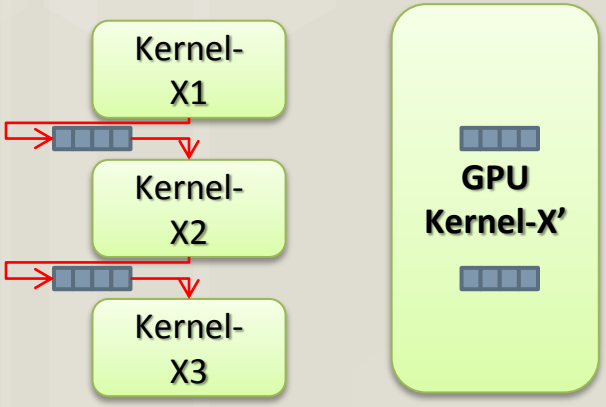


Use Case #2:
Workload – Tilted Tree



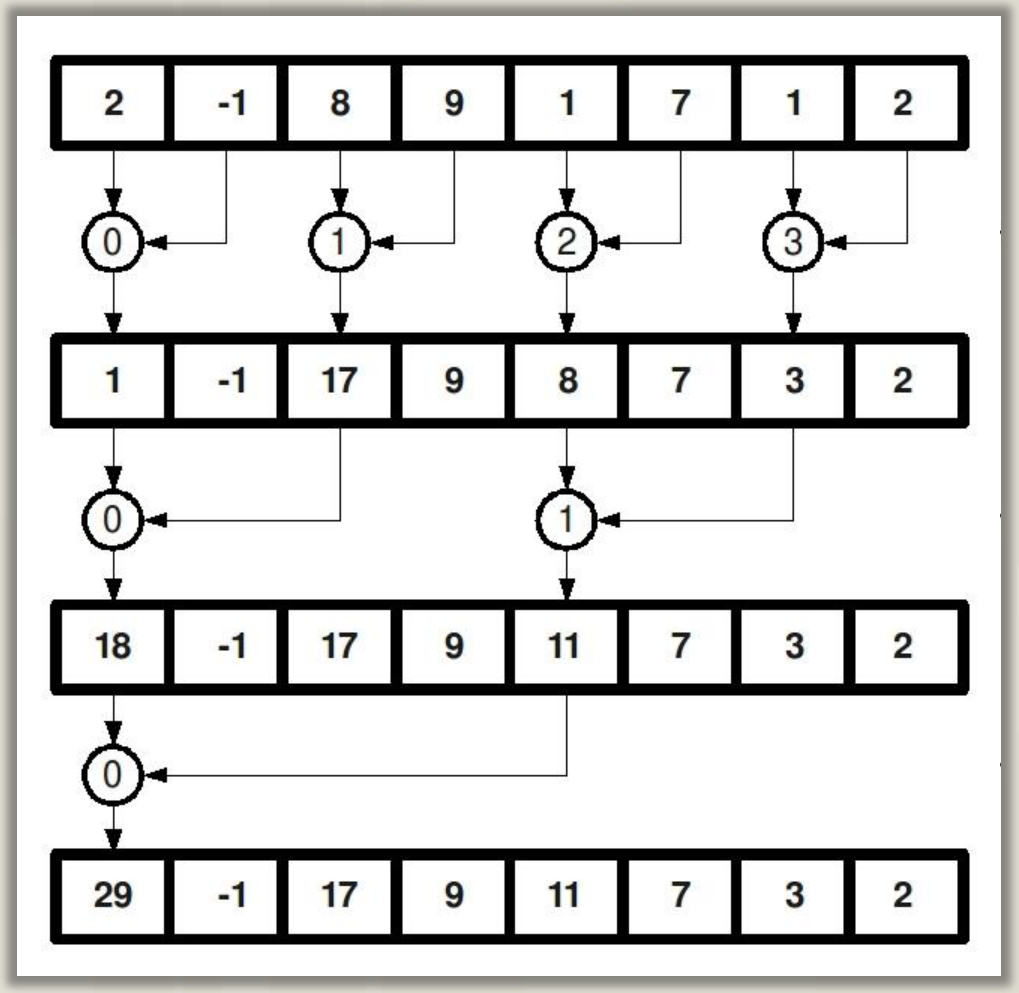
Use Case #3: Maintaining Active State

Scenario	Advantage of Persistent Threads
A kernel accumulates a single value across a large number of threads, or Kernel A wants to pass data to Kernel B through shared memory or registers	Because a PT kernel processes many more items per block than a nonPT kernel, it can effectively leverage shared memory across a larger block size for an application like a global reduction.

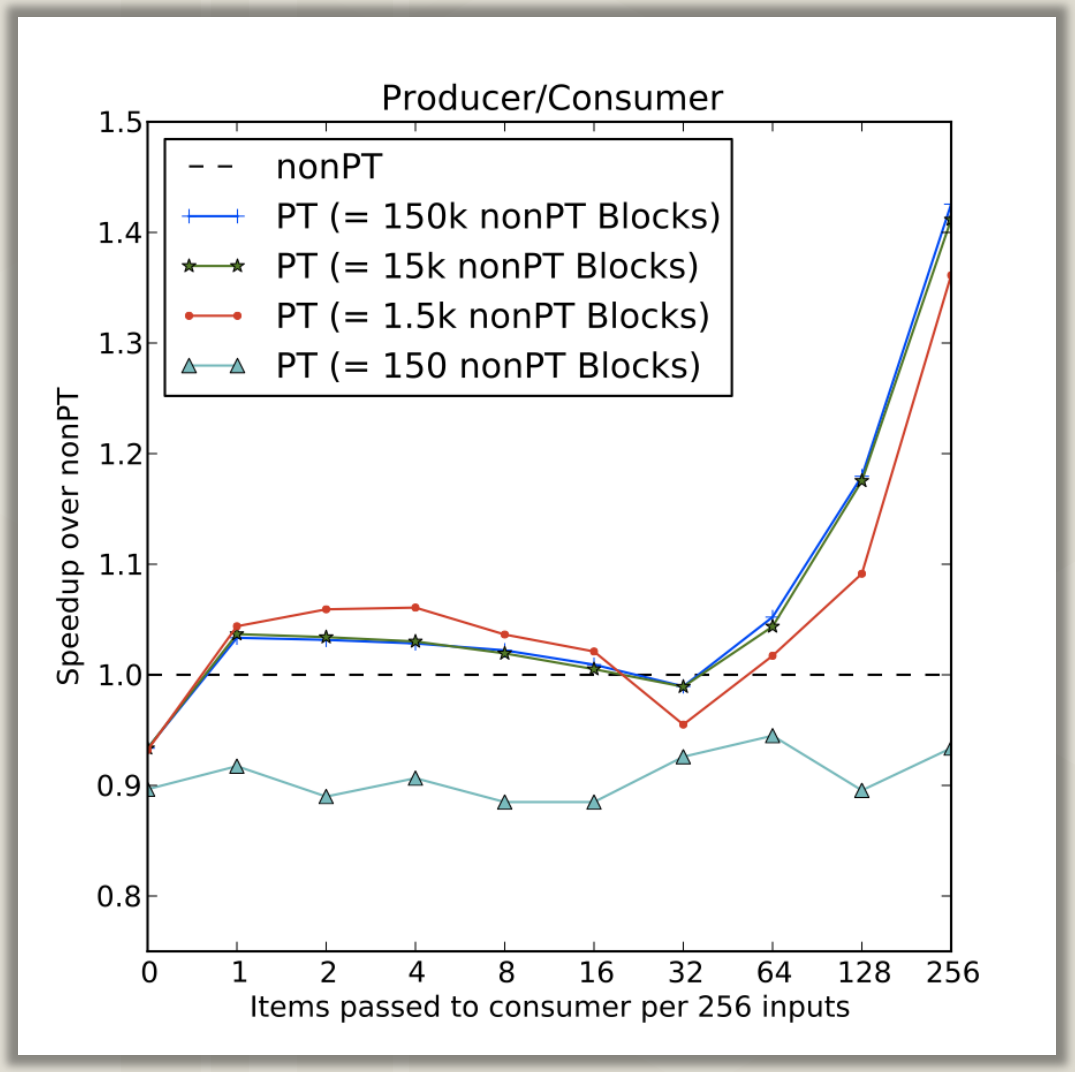


(a) nonPT (b) PT

Use Case #3:
Workload Illustration – Reduction

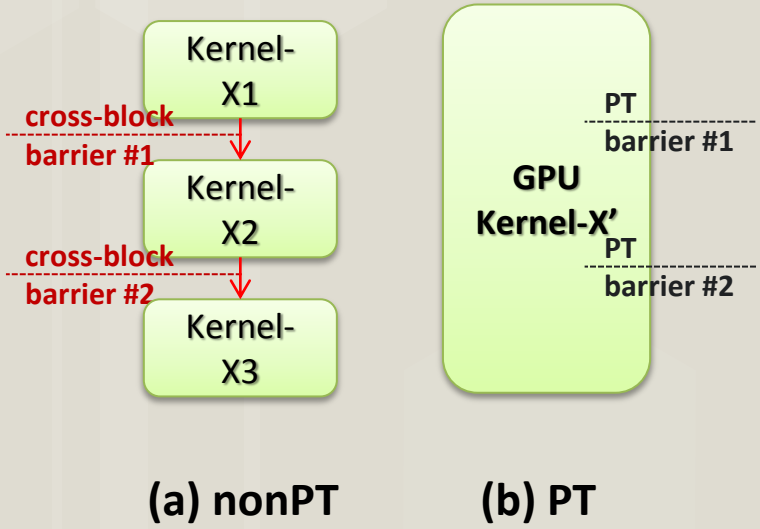


Use Case #3: Workload – Reduction

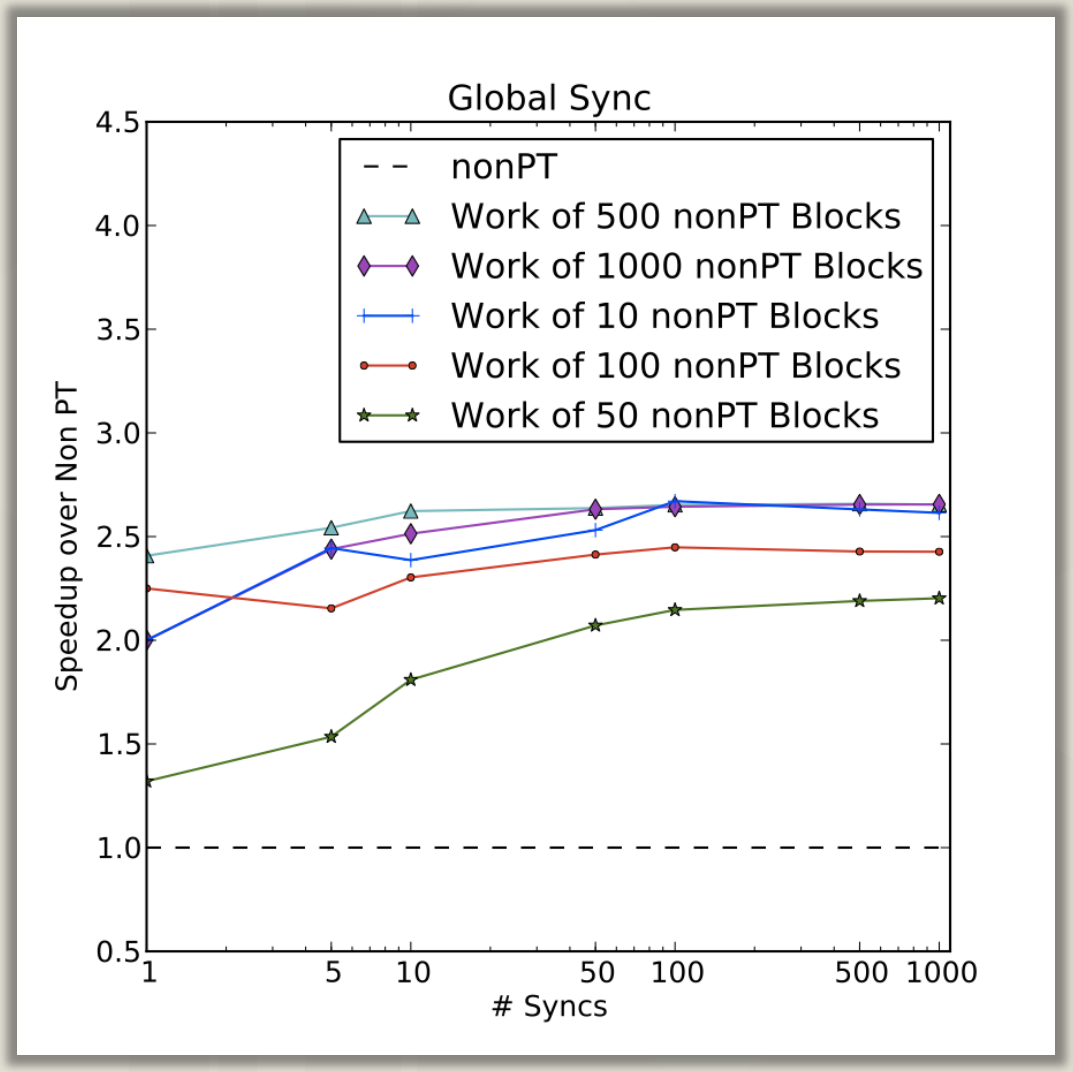


Use Case #4: Global Synchronization

Scenario	Advantage of Persistent Threads
Global synchronization within a kernel across workgroups	In a nonPT kernel, synchronizing across blocks within a kernel is not possible because blocks run to completion and cannot wait for blocks that have not yet been scheduled. The PT model ensures that all blocks are resident and thus allows global synchronization.



Use Case #4: Global Synchronization



Portability & Usability

#	Use Case	Occupancy	Scheduling	Comments
1	**CPU-GPU Synchronization**	-----	-----	indirect; CPU-GPU workload partitioning
2	**Load Balancing/Irregular Parallelism**	-----	●	non-trivial when sophisticated queuing structures (local + global) and work stealing/donation optimizations are used
3	**Maintaining Active State**	●	●	different kernel organization and partitioning strategies
4	**Global Synchronization**	●	-----	hard to debug as occupancy changes

Looking Ahead…

#	Use Case	Disucssion
1	**CPU-GPU Synchronization**	• Less of an issue on future consumer systems; but HPC still a problem • We expect this to be addressed in the future
2	**Load Balancing/Irregular Parallelism**	• Provide support for queues
3	**Maintaining Active State**	• Very hard to solve!
4	**Global Synchronization**	• Kernel launch might be cheaper than synchronizing across an entire chip in future-generation hardware

Looking Ahead…

- Return-on-Investment
- Power?
- Modifications?
  - Native support would not require a complete re-making of the underlying hardware
  - Small changes could lead to reasonable gains
  - Augment existing APIs?

Thank You!

Kshitij Gupta

www.kshitijgupta.com