PORTABLE PERFORMANCE FOR MONTE CARLO SIMULATION OF PHOTON MIGRATION IN 3D TURBID MEDIA FOR SINGLE AND MULTIPLE GPUS

Fanny Nina-Paravecino

Leiming Yu Qianqian Fang* David Kaeli

Department of Electrical and Computer Engineering

Department of Bioengineering*

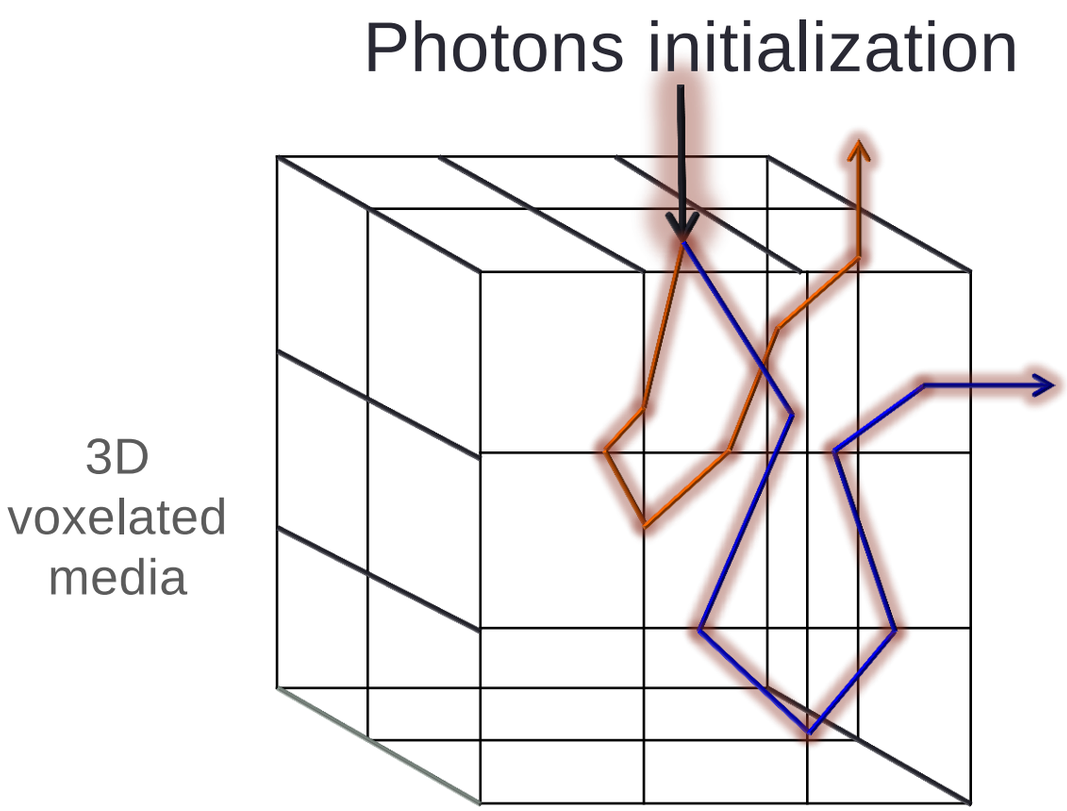
Northeastern University

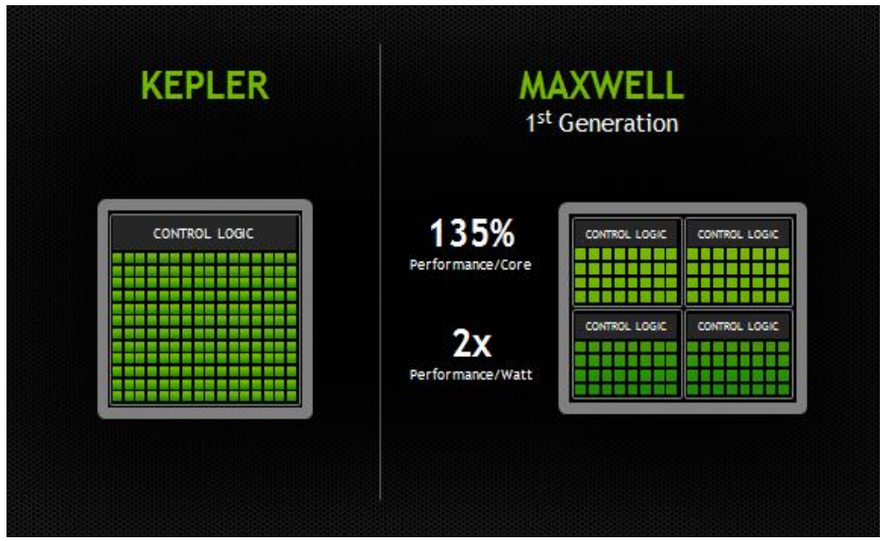
Boston, MA

Outline

- Portable Performance Monte Carlo Extreme (MCX)
  - MCX in CUDA
  - Persistent Threads in MCX
  - Portable Performance MCX
- MCX on multiple GPUs
  - Linear Performance
  - Linear Programming Model
  - Performance Results

PORTABLE PERFORMANCE MCX

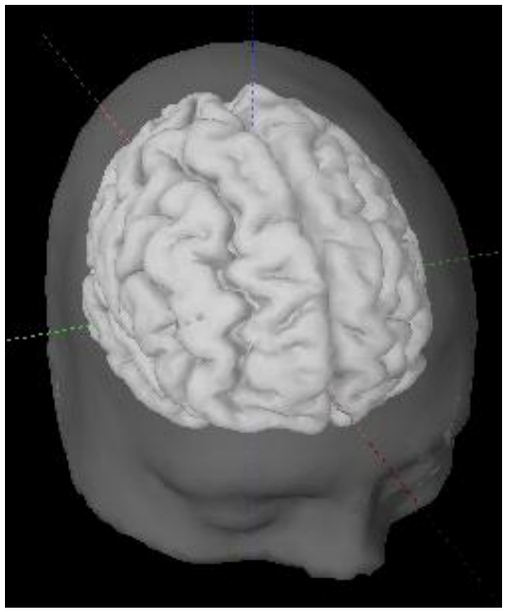


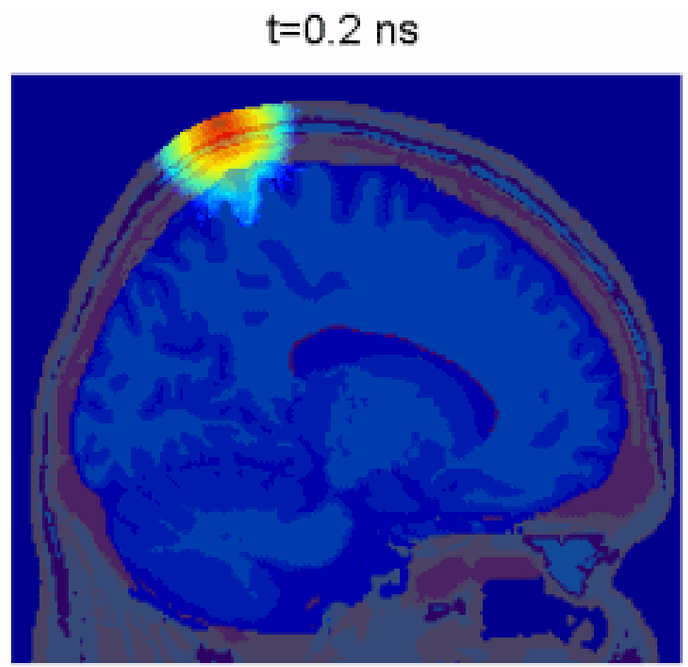


Monte Carlo Extreme (MCX) in CUDA

- Estimates the 3D light (fluence) distribution by simulating a large number of independent photons
- Most accurate algorithm for a wide ranges of optical properties, including low-scattering/voids, high absorption and short source-detector separation
- Computationally intensive, so a great target for GPU acceleration
- Widely adopted for bio-optical imaging applications:
  - Optical brain functional imaging
  - Fluorescence imaging of small animals for drug development
  - Gold stand for validating new optical imaging instrumentation designs and algorithms

MCX in CUDA

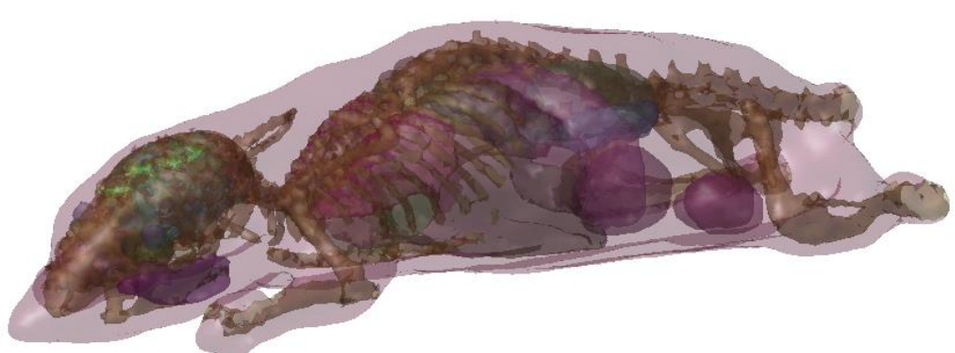


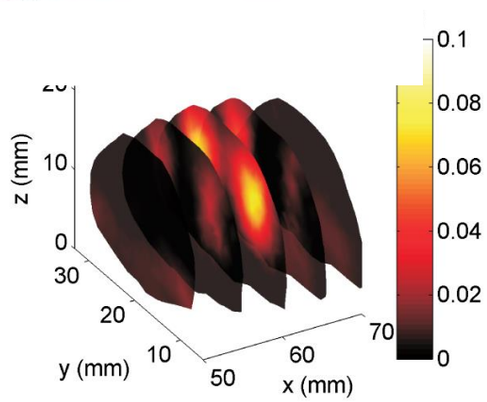


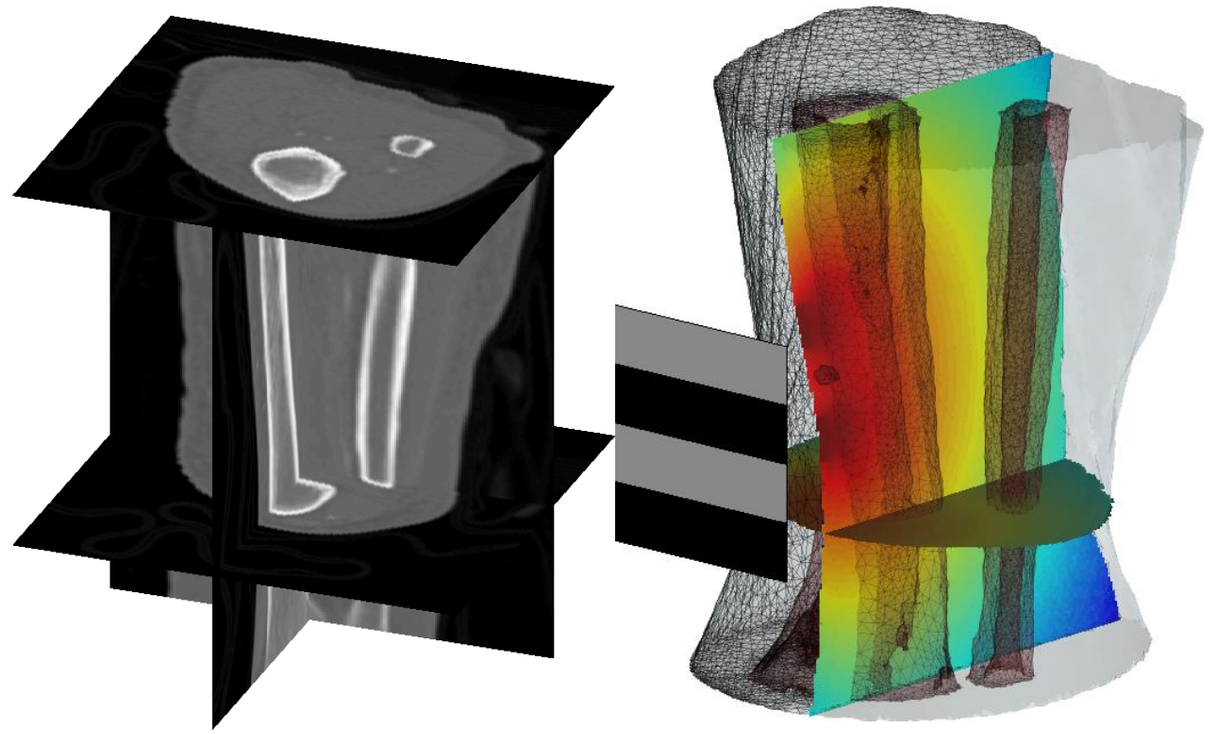
Simulation of photon transport inside human brain

Imaging of bone marrow in the tibia

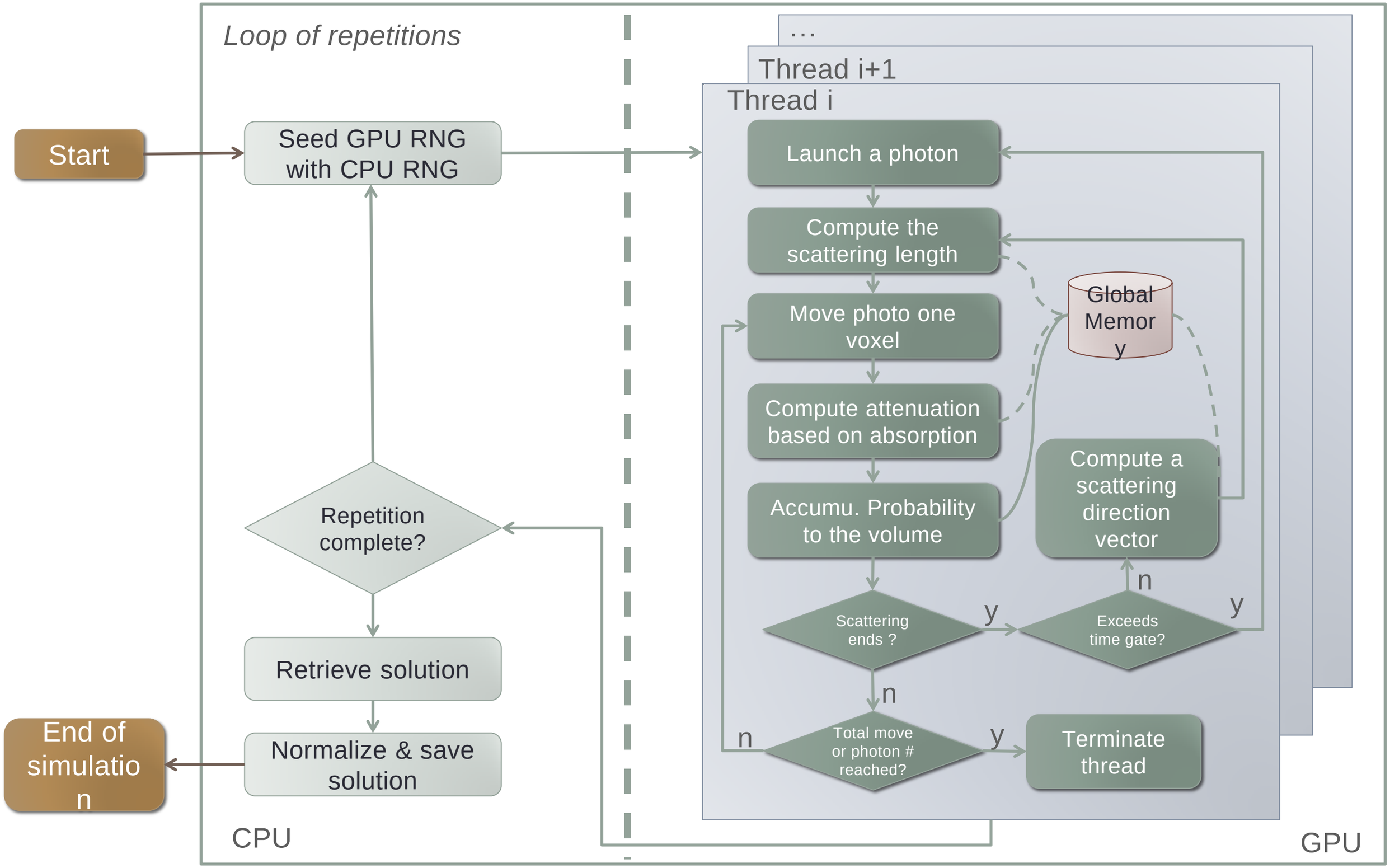
Imaging of a complex mouse model using Monte Carlo simulations







MCX in CUDA [1]

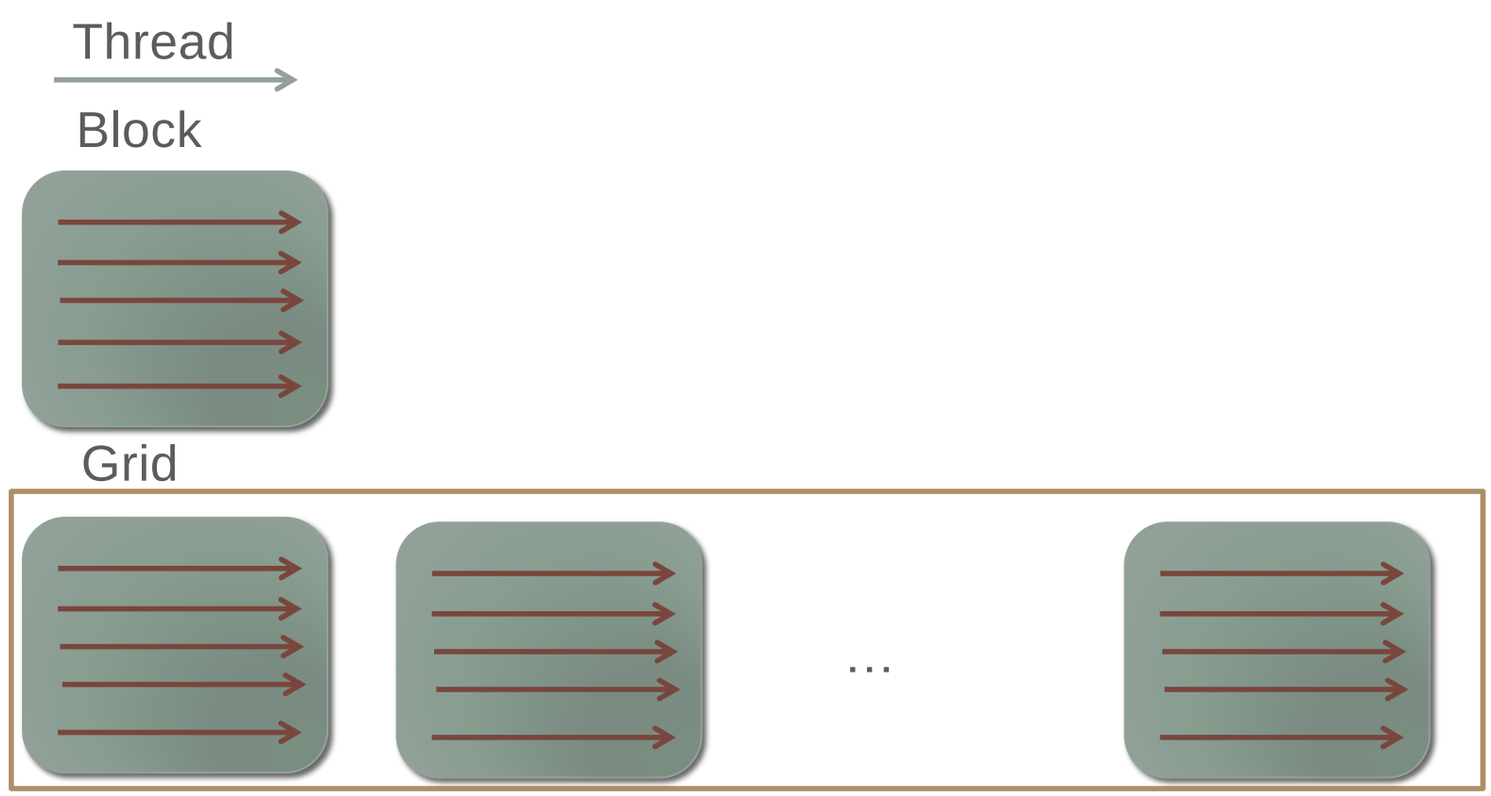


[1] Q. Fang and D. A. Boas. "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units." Optics express 17.22 (2009): 20178-20190.

Persistent Threads (PT) in MCX

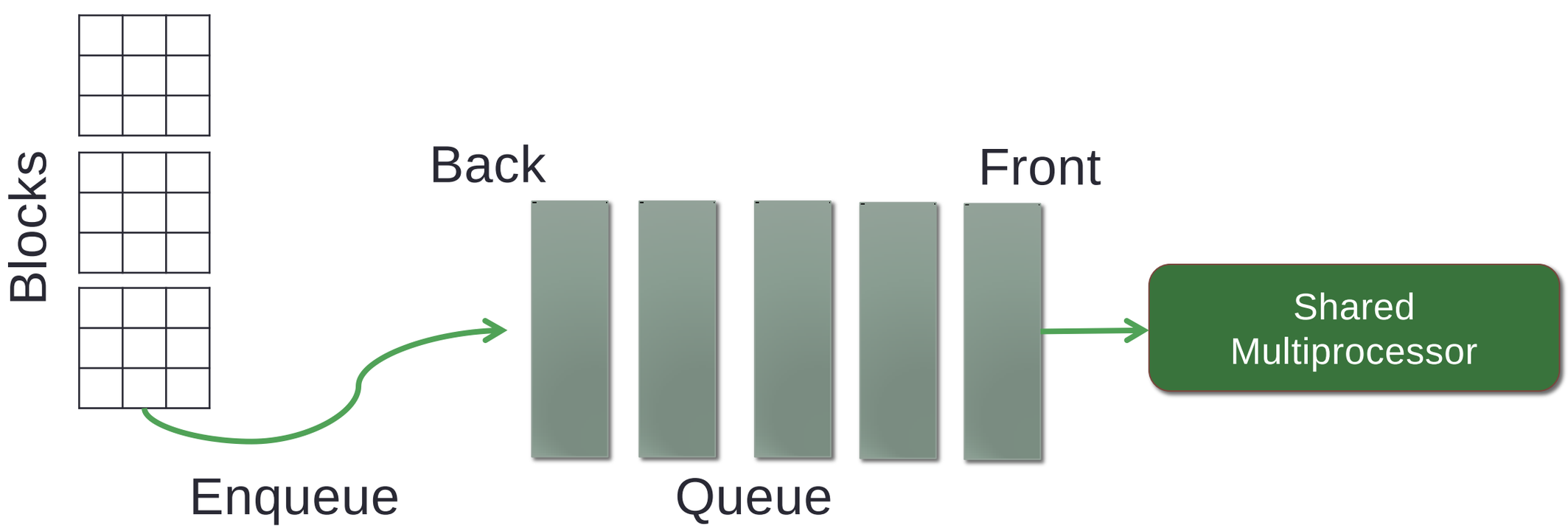
- PT kernels alter the notion of a virtual thread lifetime, treating those threads as physical hardware threads
- PT kernels provide a view that threads are active for the entire duration of the kernel
  - We schedule only as many threads as the GPU SMs can concurrently run
  - The threads remain active until end of kernel execution

CUDA Grid Structure



Persistent Threads (PT) in MCX

- A PT kernel bypasses the hardware scheduler, relying on a work queue to schedule blocks
- A PT kernel checks the queue for more work and continues doing so until no work is left
- PT MCX works on a FIFO blocking queue



Portable Performance for MCX

	Fermi	Kepler	Maxwell
MaxThreadBlocks /Multiprocessor	8	16	32
MaxThreads/Multi processor	1536	2048	2058
Multiprocessors (MP)	16	14	22
CUDA cores / MP	32	192	128

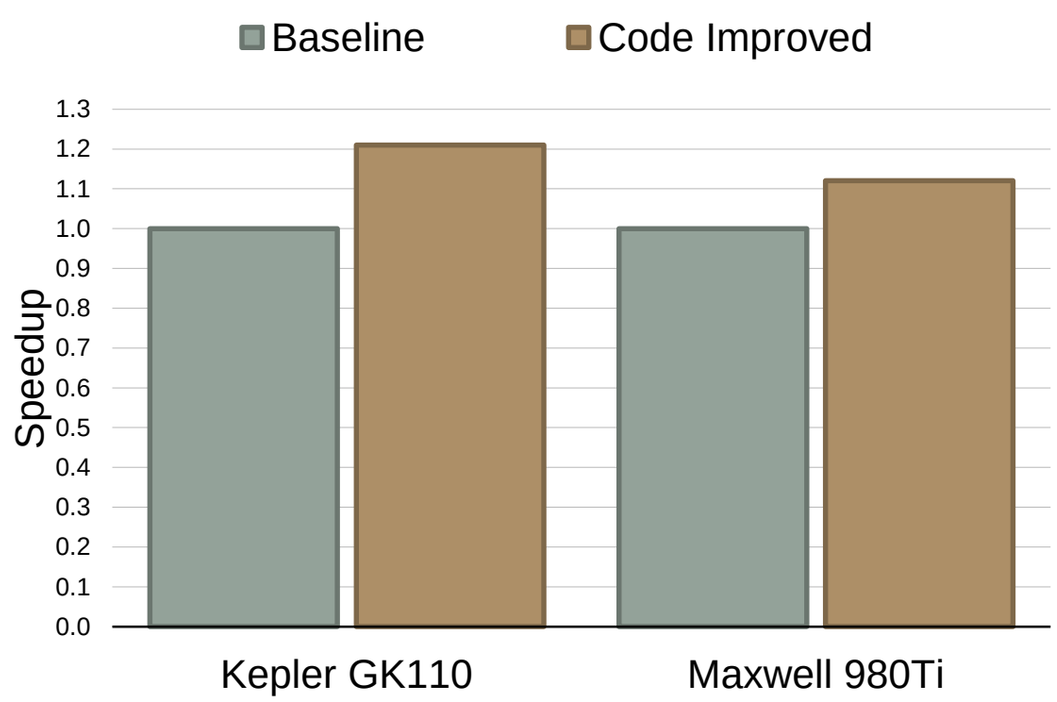
threadsPerBlock = (MaxThread/MP)/(MaxThreadBlocks/MP)

blocks = # threadsPerBlock * (MaxThreadBlocks/MP) * MP

Portable Performance MCX - Results

Kepler GK110	Baseline	Improved Code
ThreadsPerBlock	32	128
# Total Threads	86,016	28,672
# Blocks	2688	224
Performance (Photons/ms)	2383	2887
Speedup	**1.0**	**1.21**

Maxwell 980Ti	Baseline	Improved Code
ThreadsPerBlock	32	128
# Total Threads	90,112	45,056
# Blocks	2816	352
Performance (Photons/ms)	13,369	15,015
Speedup	**1.0**	**1.12**



MCX ON MULTIPLE GPUS

Linear Programming Model

- Given *n* devices: D_1, D_2, … D_n
- Given linear performance for each device
- Given the performance for 10 Million photons and 100 Millions for each device
- We can obtain the linear equation for each device as follow:

Device 1 f1: $$y_1 = b_1 + (x_1 - 1)a_1 + C_1$$

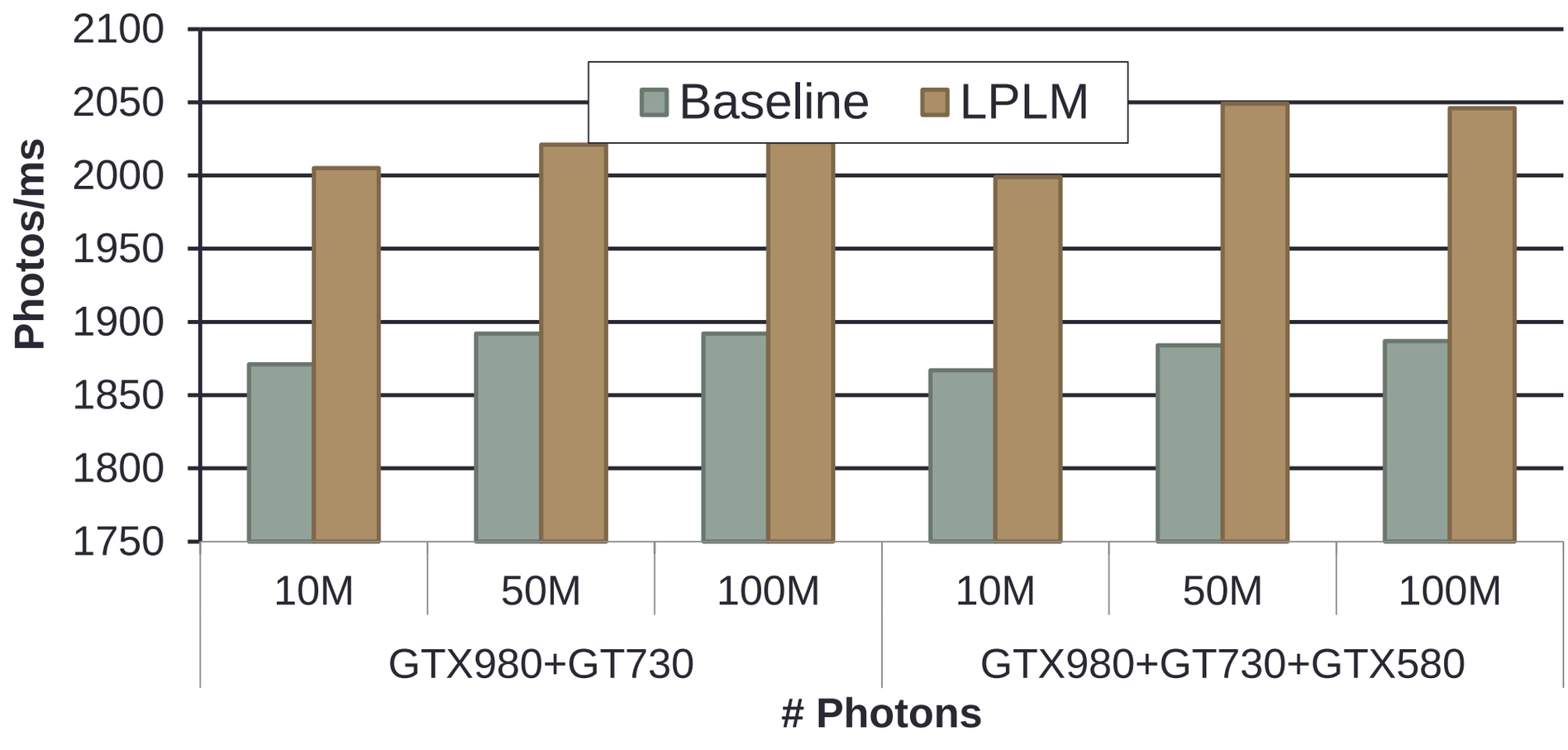
Device 2 f2: $$y_2 = b_2 + (x_2 - 1)a_2 + C_2$$

$$\vdots \qquad \vdots \qquad \vdots$$

Device n f3: $$y_n = b_n + (x_n - 1)a_n + C_n$$

Performance Results

- We evaluated our *Linear Programming on Linear Model* (**LPLM**) scheme for two different configurations of NVIDIA devices
- The resulting partition of the workload achieves an average **8%** speedup over the baseline



Summary

- We have improved the performance of MCX across a range of NVIDIA GPU architectures
- We have showed how to exploit Persistent Thread kernel to automatically tune MCX kernel
- We developed a linear programming model to find the best partition to run MCX on multiple GPUs
- We improved performance of MCX run on multiple NVIDIA GPUs, including Kepler and Maxwell
- We obtained an 8% speedup when using automatic partitioning

Future Work

- PT MCX
  - The queue of blocks can either can be static (know at compile time) or dynamic (generated at runtime), and can be used to control the order, location, and the timing of each block
- Instrumentation of MCX
  - Leverage SASSI to instrument MCX and better characterize the behavior of a kernel to guide auto-tuning
- MCX on Multiple GPUs
  - Evaluate our partitioning optimization for multiple devices

THANK YOU!

Questions?

NVIDIA

Northeastern