# Benchmark Evaluation of Radar Processing Algorithms on GPUs

## HPEC 2010

## 15 September 2010

Lockheed Martin MS2
199 Borton Landing Road
Moorestown, NJ 08057

LOCKHEED MARTIN

**Scott Sawyer**
**R. Bennett, R. Pancoast, M. Iaquinto,**
**R. Putatunda, N. Doss, J. Broadbent**

# Agenda

- Background
- GeForce GTX285 Architecture
- Pulse Compression Implementation
- Benchmark Analysis
- Shared Memory Study
- Path Forward

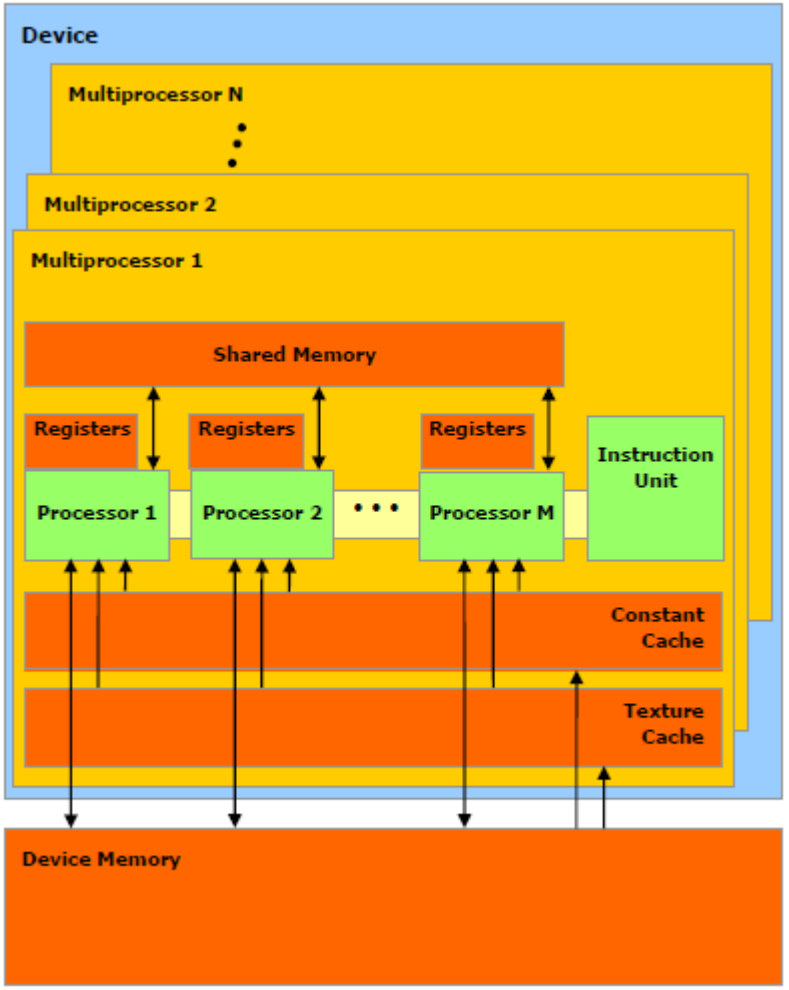
# Background

- Next-generation radar architectures will need significantly more processing capacity
- Architecture options include multi-core CPUs, FPGAs and GPUs
- Algorithms of interest include:
  - Pulse compression
  - Pulse integration
  - Adaptive beam forming
- Pulse compression benchmarked on GeForce GTX285

# NVIDIA GeForce GTX285

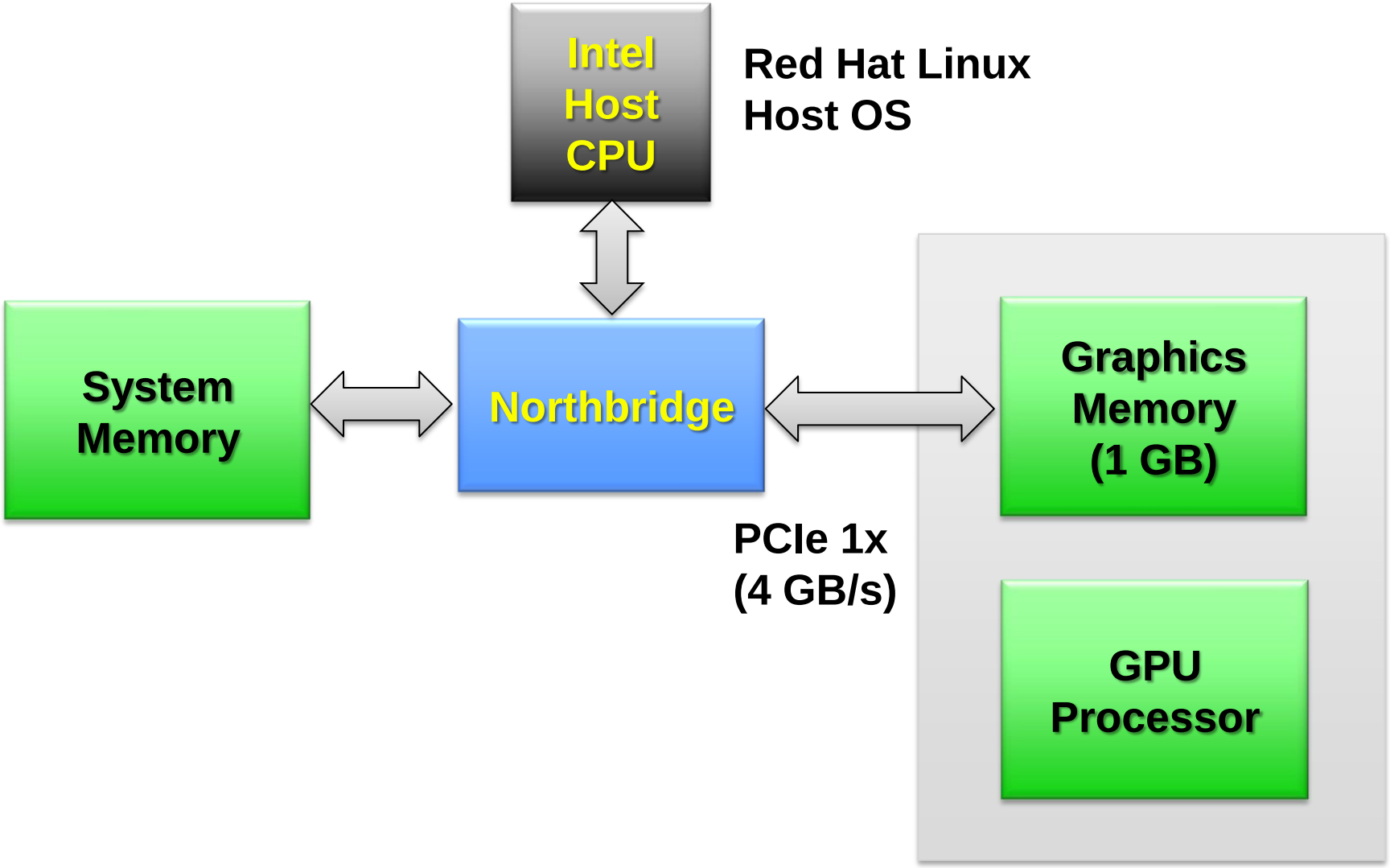
- CUDA Compute Level 1.3
- Multiprocessors: 30
- Total cores: 240
- Global Memory: 1 GB
- Shared Memory: 16 KB per multiprocessor
- Processor Clock: 1476 MHz

- Theoretical peak throughput: 1063 GFLOPS



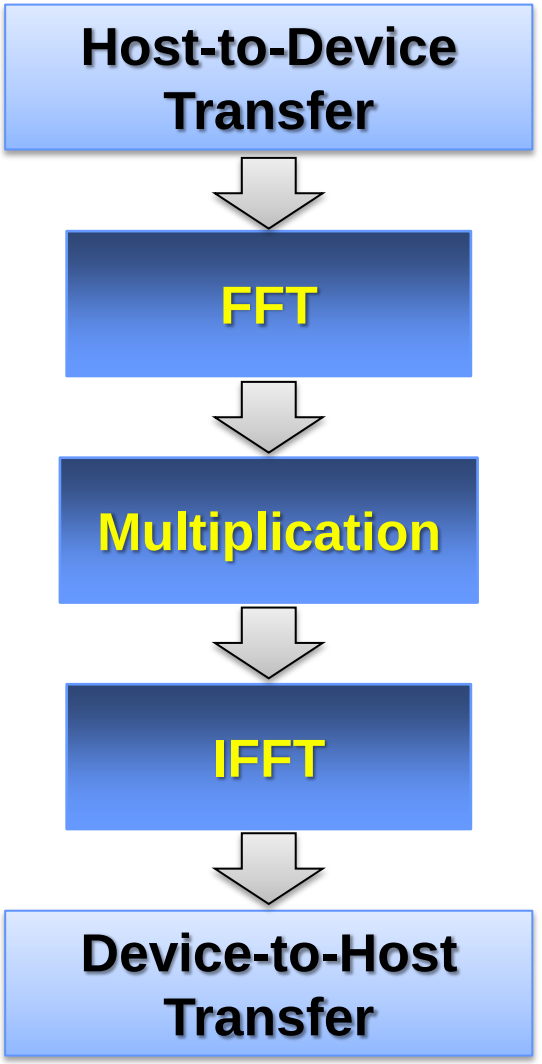

NVIDIA CUDA C Programming Guide, Version 3.1, 5/28/2010

# Benchmark System Architecture



# Pulse Compression Implementation

- C for CUDA
  - CUFFT for FFT and IFFT
  - Custom kernel for point-by-point multiplication
- MATLAB reference implementation
  - Input vectors
  - Result verification
- Assumptions
  - Pulsed radar
  - LFM Waveform
  - Frequency domain matched filter
- Best of ‘N’ runs taken
  - Assume least host OS interference

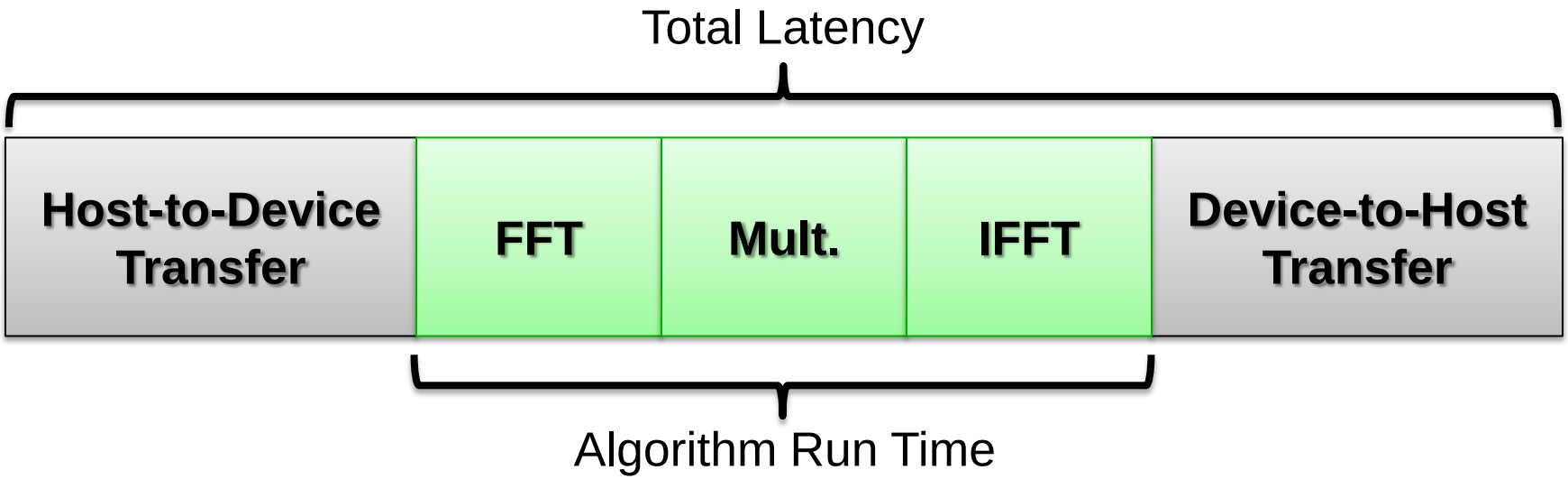


# Input Vectors

- Simulated radar returns
  - Complex signal (interleaved real/imaginary)
  - Single-precision floating point
  - Constant delay/range
  - AWGN
- Variable pulse interval (FFT size)
  - 1024, 2048, 4096, …, 65536 samples
- Variable pulse count
  - 1, 2, 4, …, 64 pulses

# Performance Metrics

## Algorithm Timing

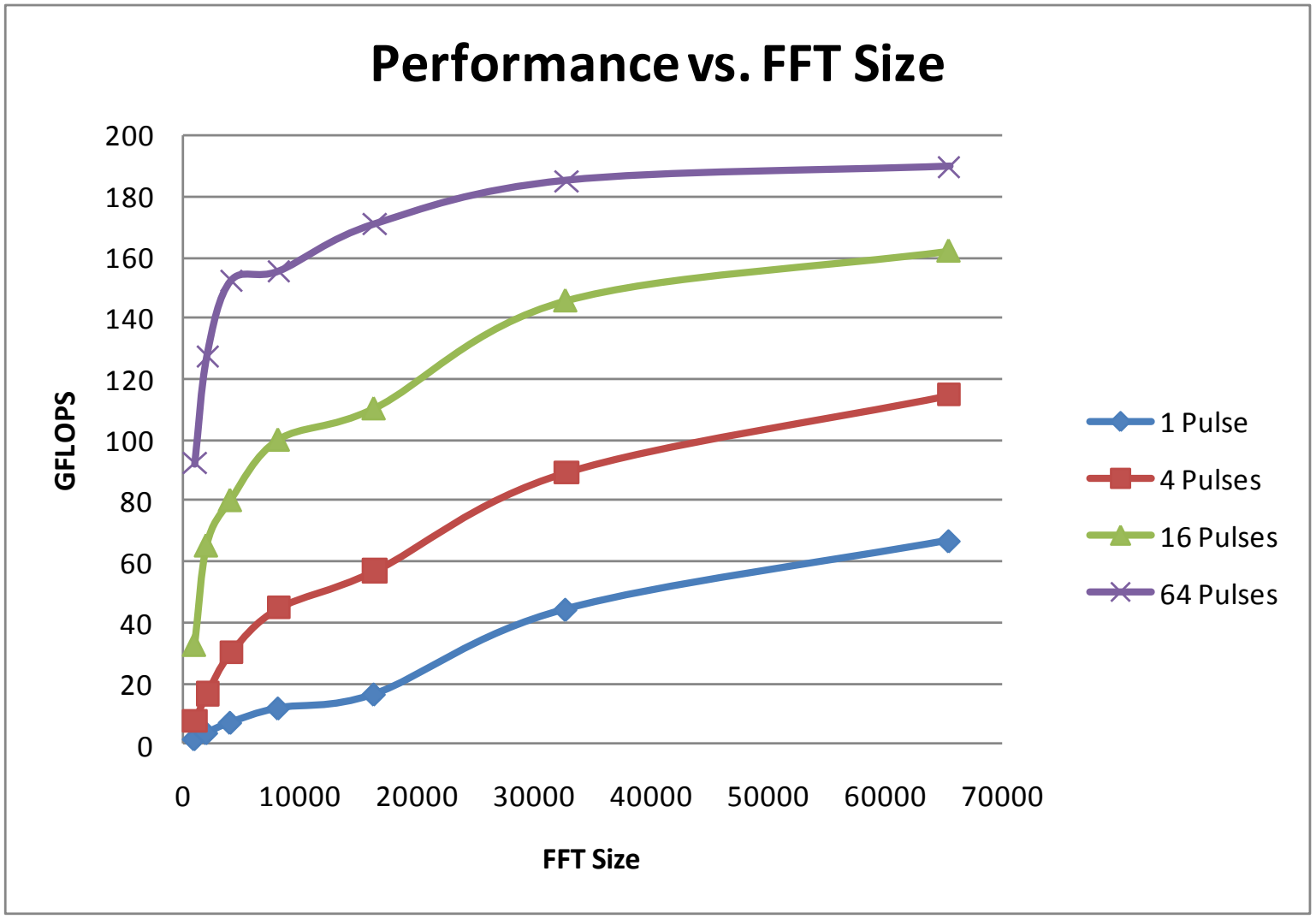



- Timing does not include file I/O
- Used CUDA event API to time asynchronous kernel calls

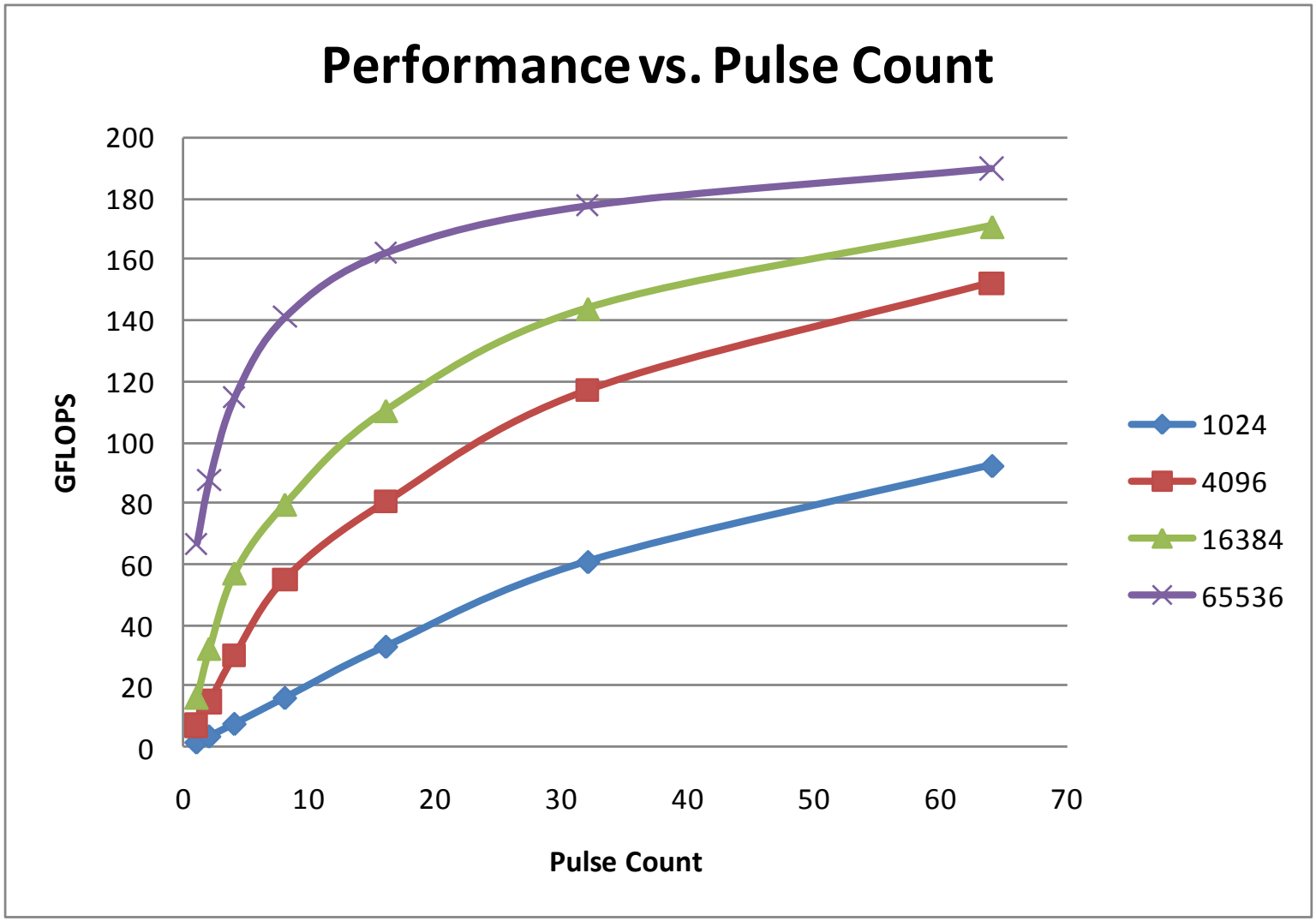
## Operation Counts

- Assume radix-2 FFTs
- Count all floating point adds and multiplies in custom kernel

# Performance vs. FFT Size

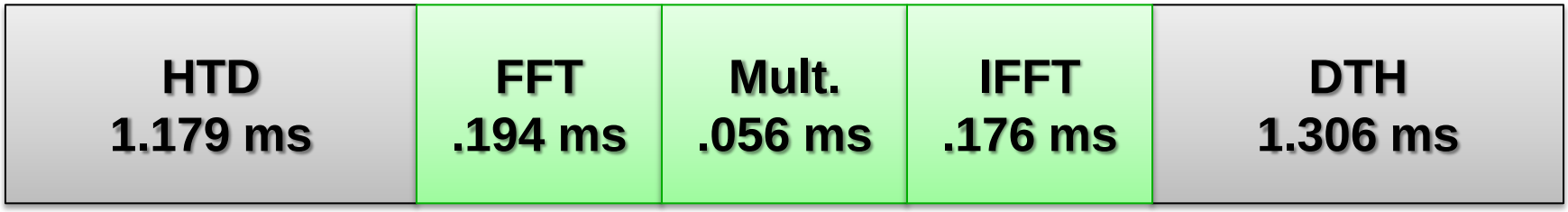


# Performance vs. Pulse Count



# Timing Breakdown

Example: 16K, 16 Pulses



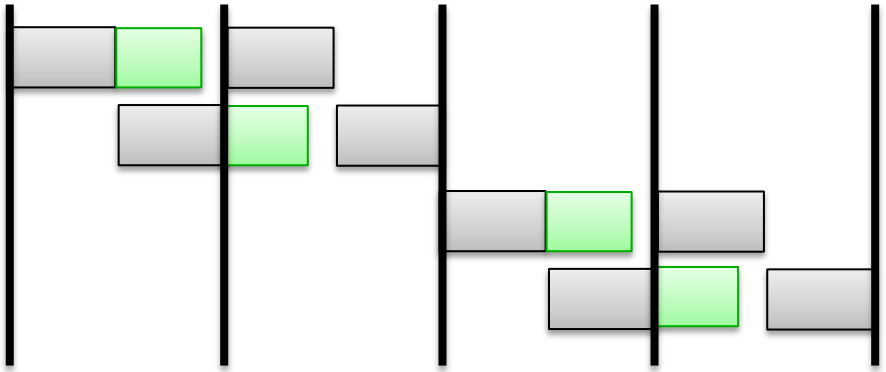

- 15% run time, 85% transfer time
- 1696 MB/s PCIe (1x) transfer rate

All times in ms

| | 4096, 4 Pulses | 4096, 16 Pulses | 16K, 4 Pulses | 16K, 16 Pulses |
|---|---|---|---|---|
| **Host-to-Device Copy** | 0.089 | 0.319 | 0.314 | 1.179 |
| **FFT** | 0.029 | 0.050 | 0.077 | 0.194 |
| **Multiply** | 0.011 | 0.023 | 0.023 | 0.056 |
| **IFFT** | 0.028 | 0.049 | 0.077 | 0.176 |
| **Device-to-Host Copy** | 0.116 | 0.399 | 0.400 | 1.306 |
| | | | | |
| **Run Time** | 0.068 | 0.123 | 0.177 | 0.427 |
| **Latency** | 0.273 | 0.840 | 0.891 | 2.912 |

# Real Time Operation

## GTX285, Single DMA Channel



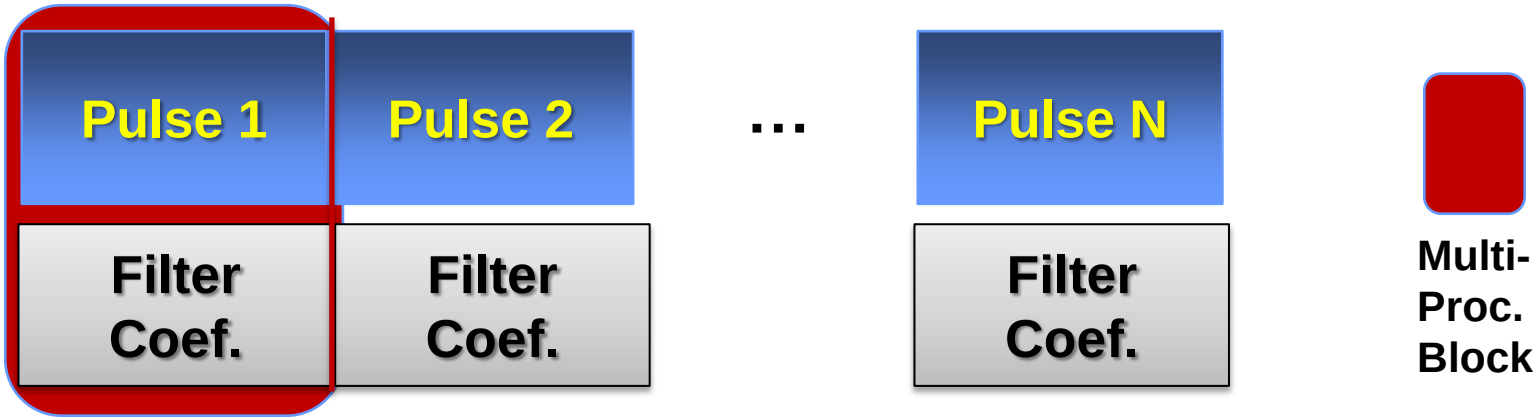
For run time less than one-way transfer, real-time operation requires HTD+DTH time less than pulse interval.

## Fermi, Dual DMA Channels

Separate HTD and DTH channels allow tighter timelines. Real-time operation requires HTD + algorithm run time less than pulse interval.

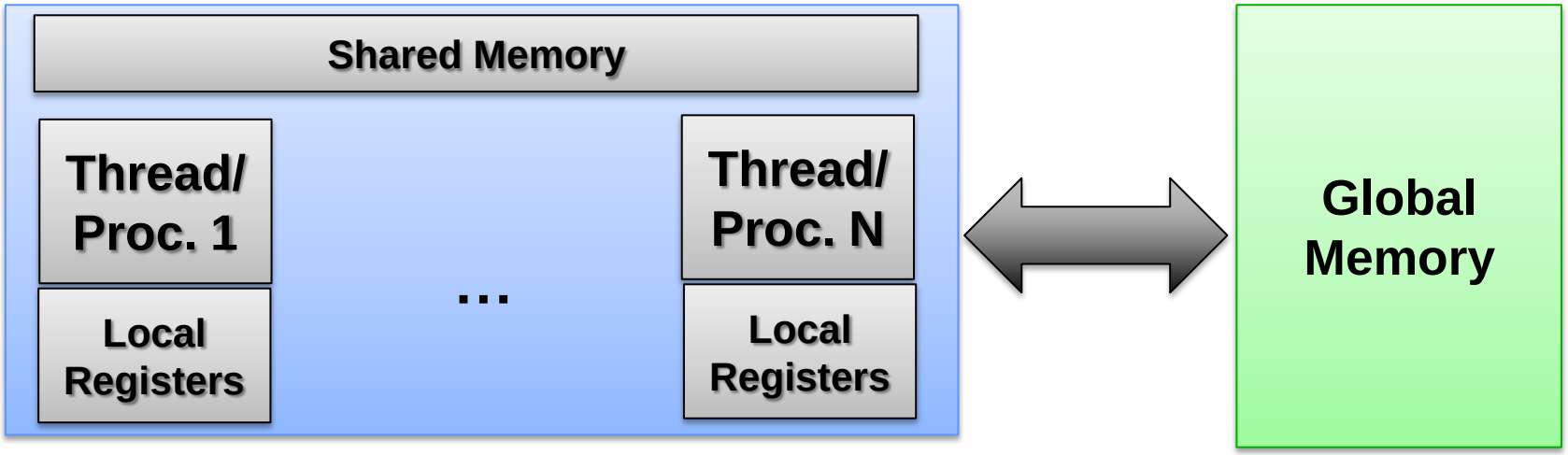
# Shared Memory Study

**Investigated optimizing multiplication step by using shared memory.**

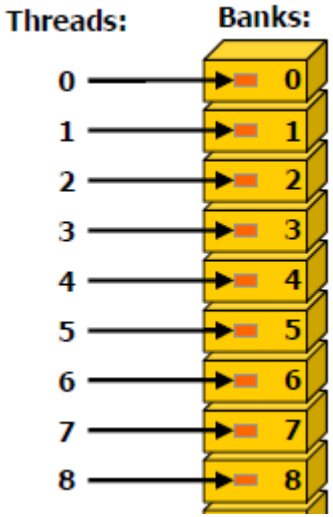



**Reduce reads from global to on-chip registers by sharing filter coefficients:**

# Coalesced Reads

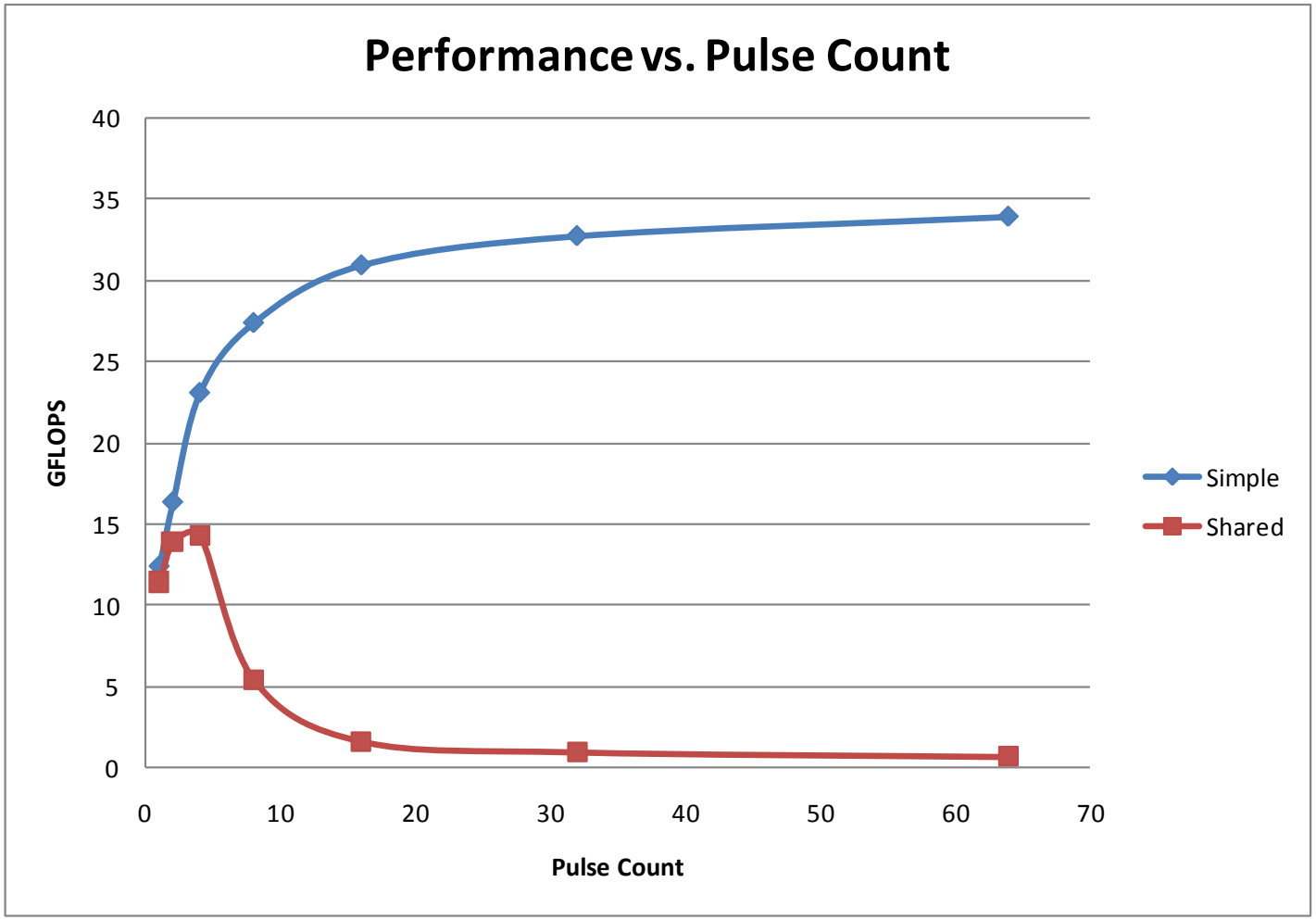



- Global memory accessed via 32-, 64- or 128-byte reads
- Threads accessing aligned memory may use coalesced reads (strict rules based on compute capability)




NVIDIA CUDA C Programming Guide, Version 3.1, 5/28/2010

# Shared Memory Results

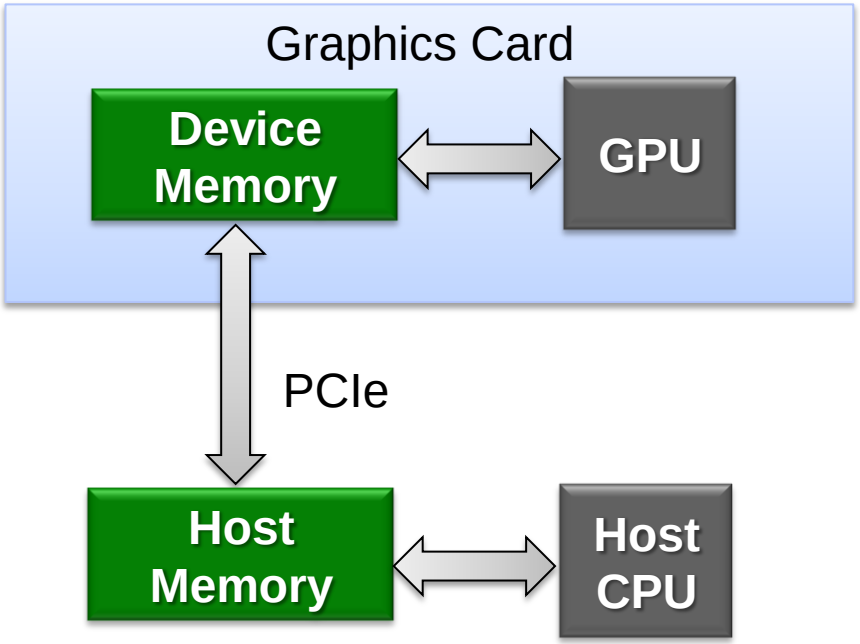



- **Shared memory implementation lowers multiplication block performance.**

# Path Forward for Tactical Applications

- Update algorithms for Fermi architecture
- Benchmark adaptive beam forming and other algorithms

**Desktop System**




**Future Tactical Systems**