Understanding of GPGPU Performance: Towards a New Optimization Tool

Adi Fuchs, Noam Shalev and Avi Mendelson – Technion , Israel Institute of Technology

This work was supported in part by the Metro450 consortium

Today's Topics

- Background
- Vision

- Our benchmarks

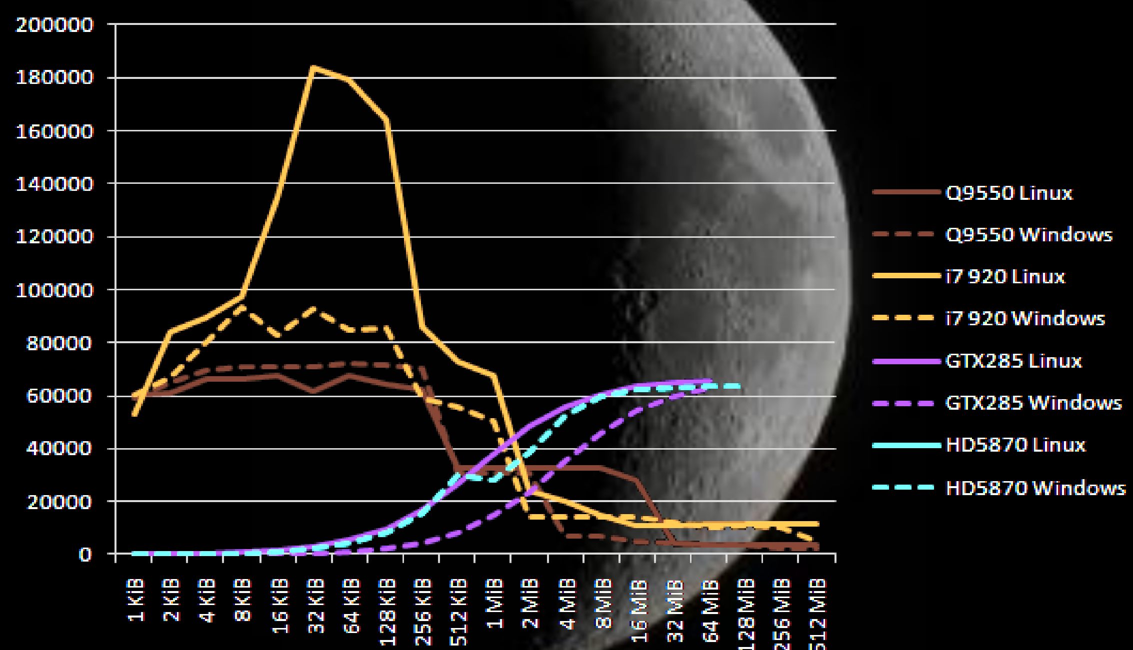
Adi Fuchs, Noam Shalev and Avi Mendelson – Technion , Israel Institute of Technology

- Conclusions + Future Work

This work was supported in part by the Metro450 consortium

Background

- **GPU provide significant performance or power efficiency for parallel workloads**
- **However, even simple workloads are microarchitecture and platform sensitive**



Bandwidth (in MB/s) for memory copy on two CPU, two GPU, and two 64-bit systems.

- **Why do applications behave the way they do?**

Background

Existing tools and work – Industry + Academia:

- **GPGPU Profiling tools:**

  - complex and not conclusive

  - mainly based on companies' work (don't expose undocumented behavior)

- **Academic work**

  - some works suggest the use of targeted benchmarks

  - some target specific structures or aspects

  - many are based on "common knowledge"

Background

Goals:

- **Unveil GPU microarchitecture characterizations**

- **…Including undocumented behavior!**

- **Auto-match applications to HW spec + HW/SW optimizations**

Today's Topics

- Background
- Vision
- Our benchmarks
- Conclusions + Future Work

Vision

Current work

- We have a series of CUDA benchmarks that explore different NVIDIA cards
- Each micro-benchmark pinpoints a different phenomena
- We focus on the memory system – has a huge impact on performance and power
- Benchmarks executed on 4 different NVIDIA systems

Vision

Long term vision...

- We wish to construct an application + HW characteristics database

- Based on this database we would like to construct a matching tool:

1. Given a workload – what type of hardware should be used?
2. Given workload + hardware – what optimizations to apply?

Today's Topics

- Background
- Vision
- Our benchmarks
- Conclusions + Future Work

Our Benchmarks

- Common microbenchmarks often target hierarchy (e.g. cache levels)
- Targeting hierarchy adds to the code's complexity
- Targeting hierarchy harms portability! (machine dependent code)
- Our micro-benchmarks target behavior, not hierarchy

Our Benchmarks

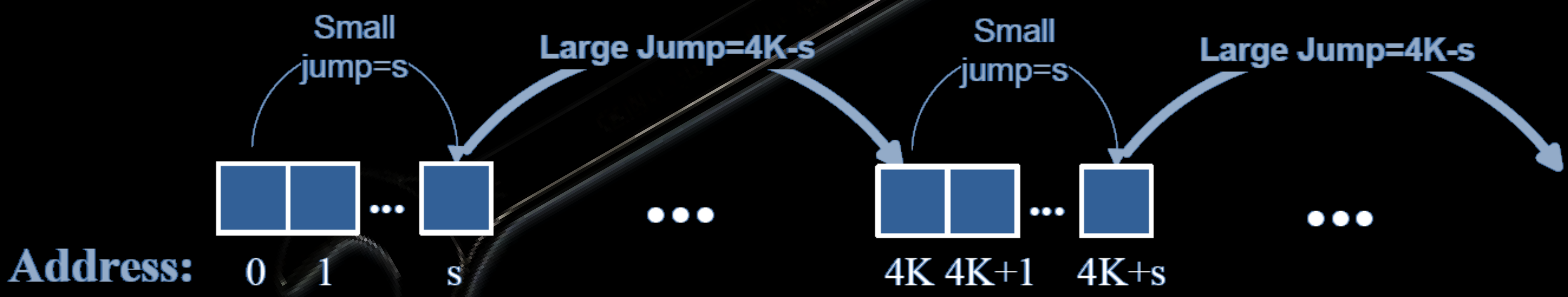
4 systems tested:

	C2070	Quadro 2000	GTX680	K20
Device Name	Tesla C2070	Quadro 2000	GeForce GTX 680	Tesla K20m
GPU Architecture	Tesla	Fermi	Kepler	Tesla
CUDA Driver / Runtime Version	5.0 /5.0	5.0 /5.0	5.0 /5.0	5.0 /5.0
CUDA Capability	2.0	2.1	3.0	3.5
Global memory size	6144 MBytes	1024 MBytes	4096 MBytes	4800 MBytes
Multiprocessors	14	4	8	13
CUDA Cores/MP	32	48	192	192
Total number of cores	448	192	1536	2496
GPU Clock rate	1.15 GHz	1.25 GHz	1.06 GHz	0.71GHz
Memory Clock rate	1.5 GHz	1.3 GHz	3 GHz	2.6 GHz
Memory Bus Width	384-bit	128-bit	256-bit	320-bit
L2 Cache Size	786432 bytes	262144 bytes	524288 bytes	1310720 bytes
Constant memory size	65536 bytes	65536 bytes	65536 bytes	65536 bytes
Shared memory per block	49152 bytes	49152 bytes	49152 bytes	49152 bytes
Max registers per block	32768	32768	65536	65536
Warp size	32	32	32	32
Max threads / MP	1536	1536	2048	2048
Threads per block	1024	1024	1024	1024
Linux kernel version	3.2.0-32-generic	3.2.0-32-generic	3.2.0-38-generic	3.2.0-38-generic

Our Benchmarks

Micro-benchmark #1: Locality

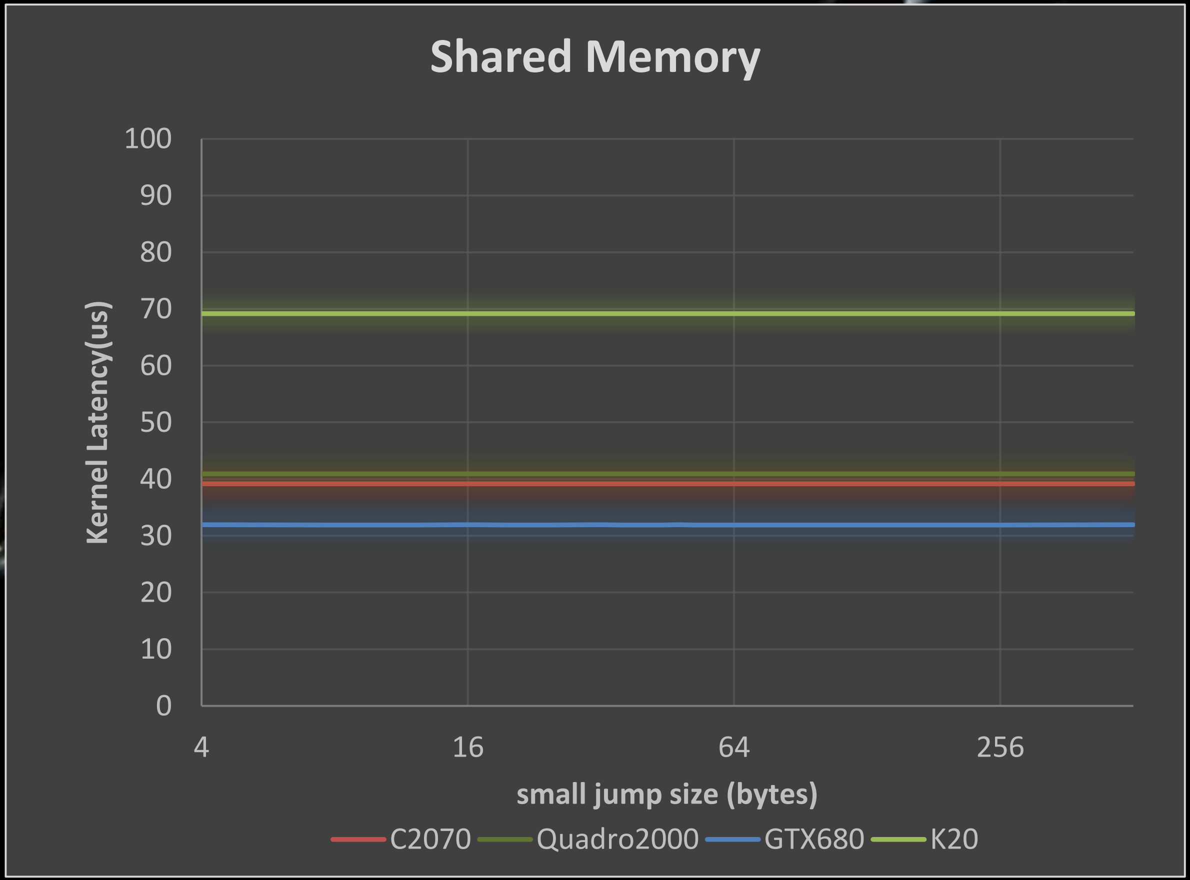
- Explore sizes of cacheline/prefetch using small jumps of varying size



Our Benchmarks

Micro-benchmark #1: Locality

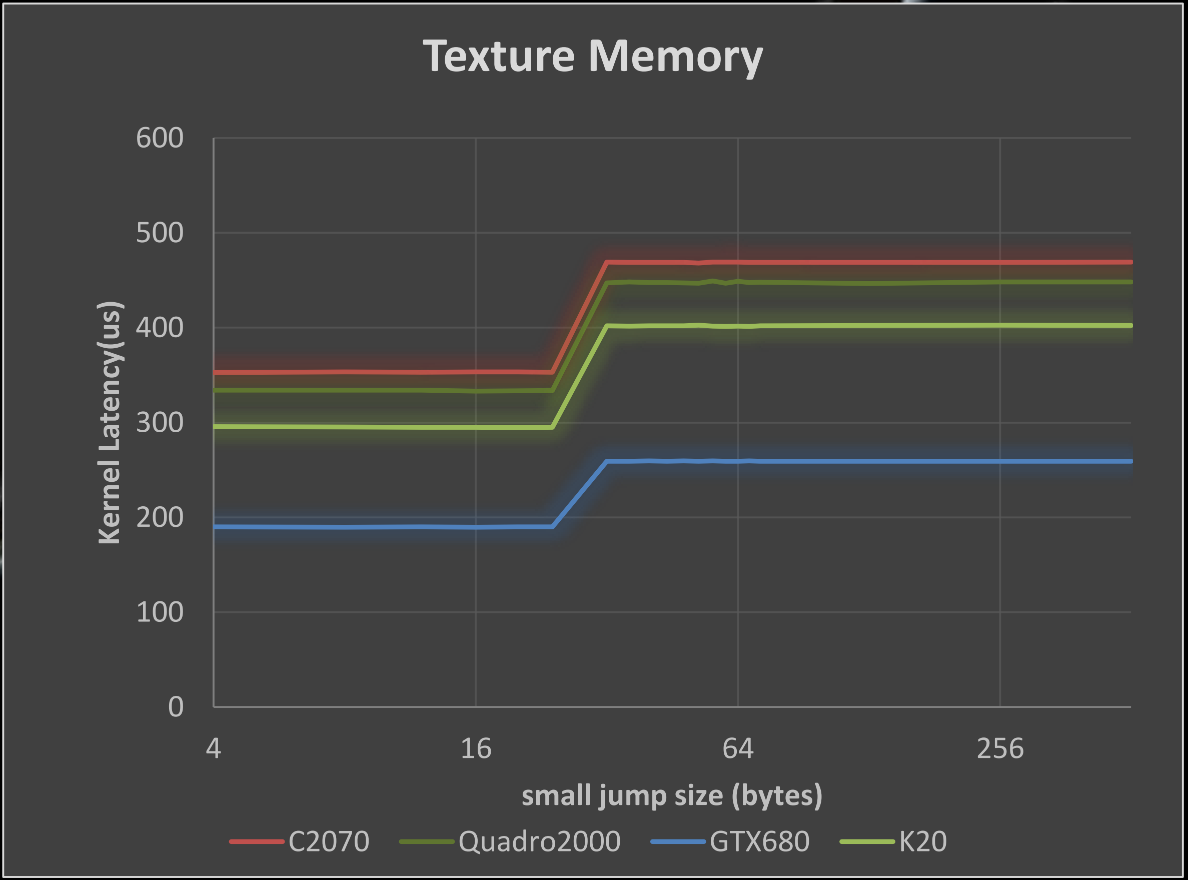
- In all systems tested shared memory is latency is fixed → no caching/prefetching



Our Benchmarks

Micro-benchmark #1: Locality

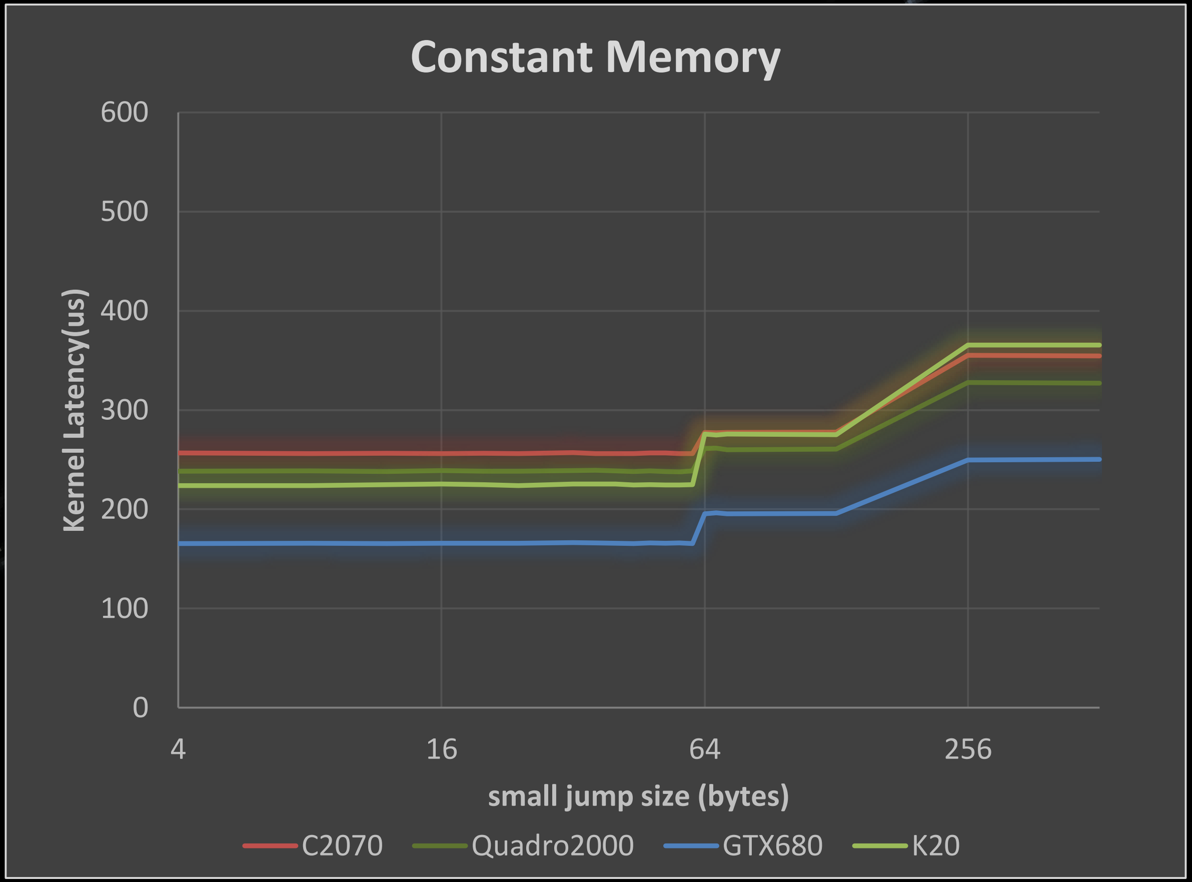
- Texture memory caching is 32 bytes of size = 4 double precision coordinates



Our Benchmarks

Micro-benchmark #1: Locality

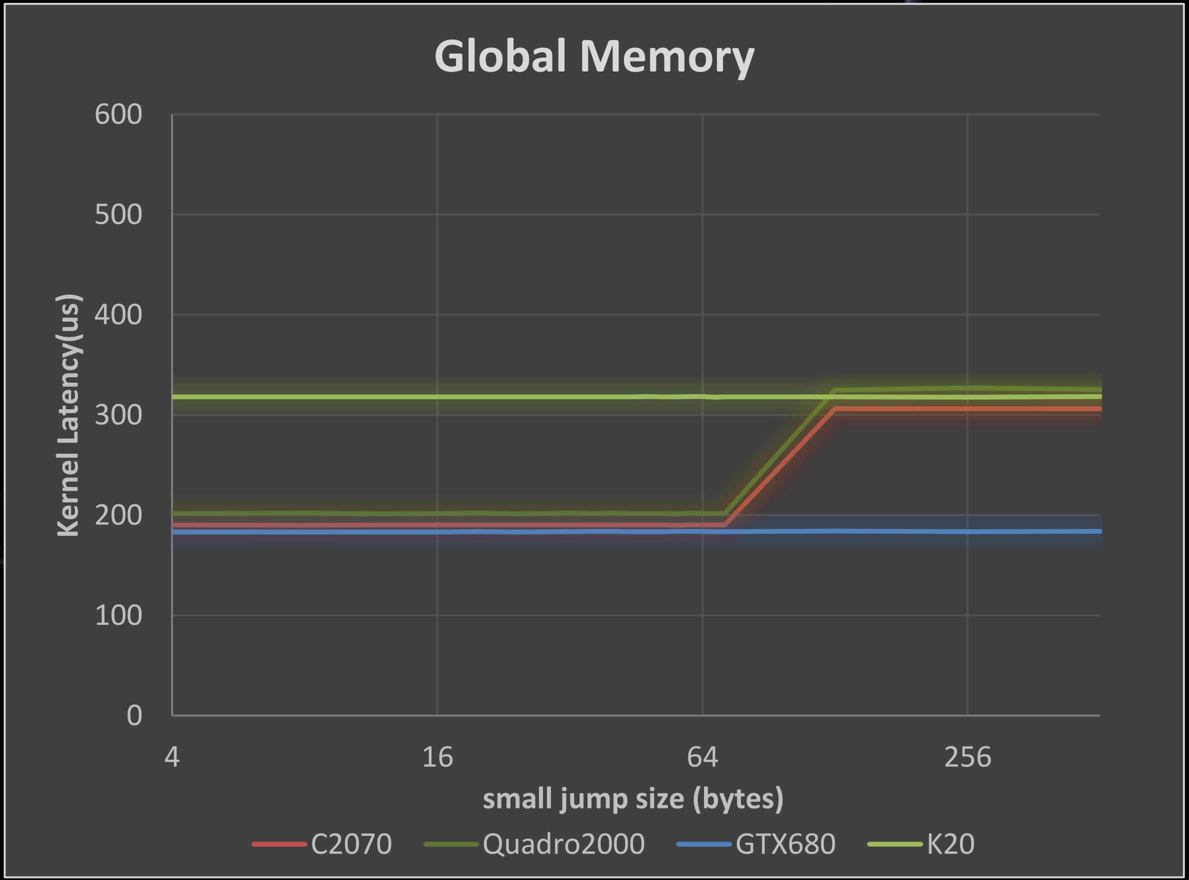
- Constant memory has a 2-level hierarchy for 64 and 256 byte segments



Our Benchmarks

Micro-benchmark #1: Locality

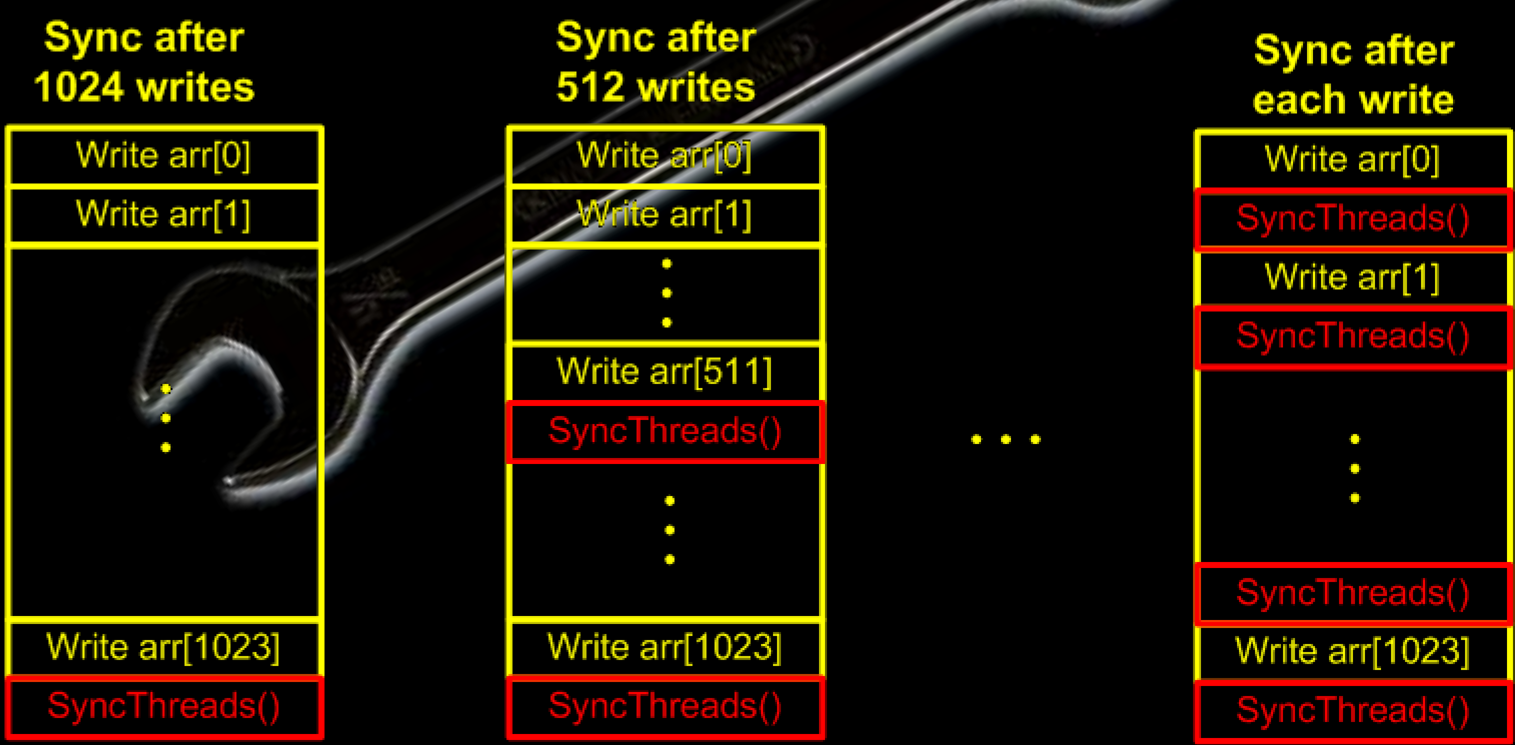
- Global memory – CUDA 2.x systems support caching / prefetching



Our Benchmarks

Micro-benchmark #2: Synchronization

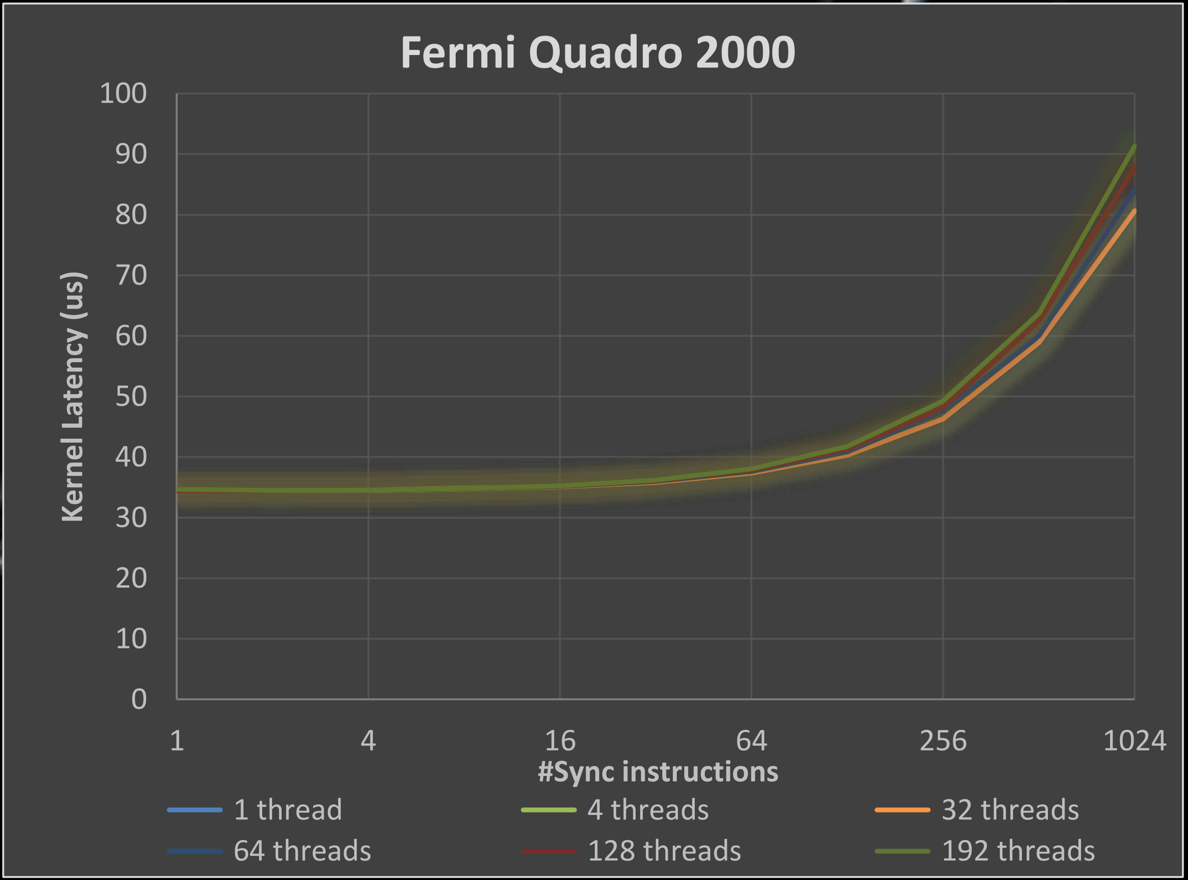
- Examine the effects of varying synchronization granularity for memory writes
- Number of thread changes as well - each thread executes the same kernel:



Our Benchmarks

Micro-benchmark #2: Synchronization

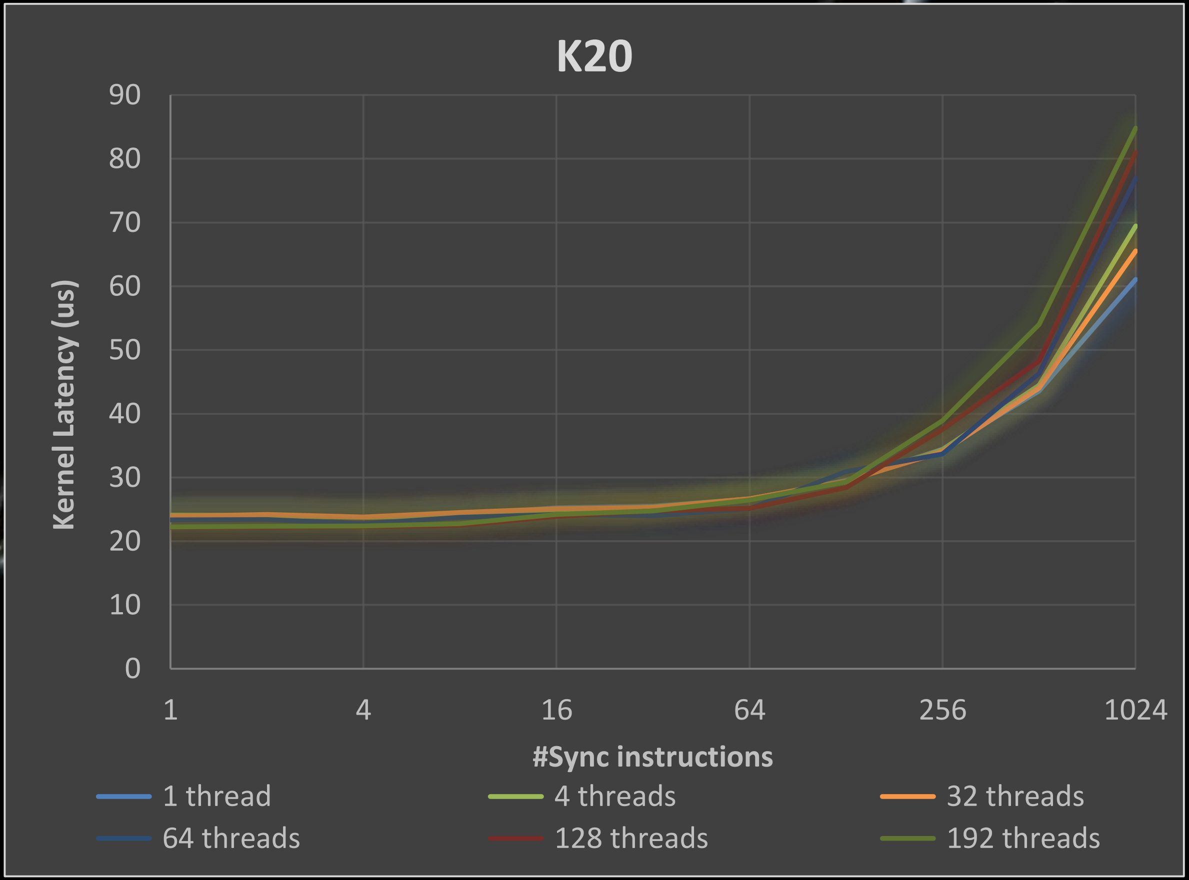
- Fine-grained sync increase latency by 163%. 192 threads increase latency by 13%



Our Benchmarks

Micro-benchmark #2: Synchronization

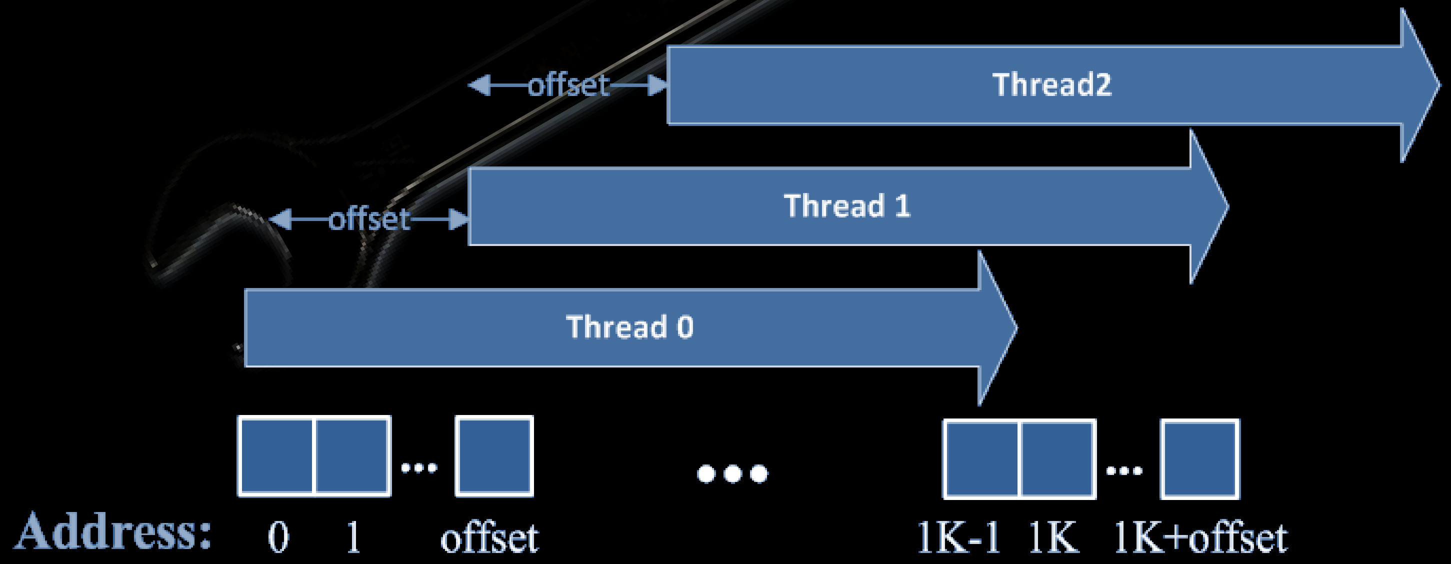
- Fine-grained sync increase latency by 281%. 192 threads increase latency by 38%



Our Benchmarks

Micro-benchmark #3: Memory Coalescing

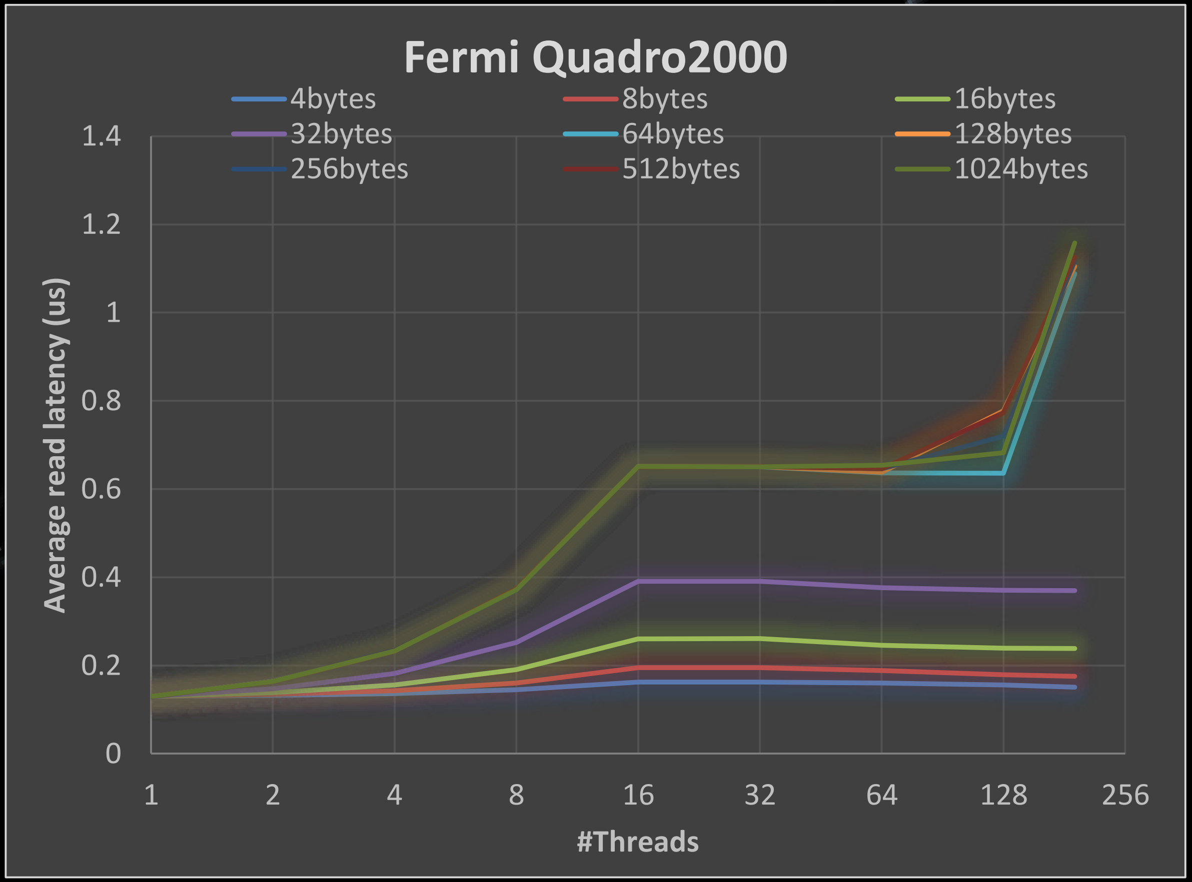
- Target: the ability of grouping memory accesses from different threads
- …And what happens when it's impossible.
- Each thread reads 1K lines starting from a different offset.



Our Benchmarks

Micro-benchmark #3: Memory Coalescing

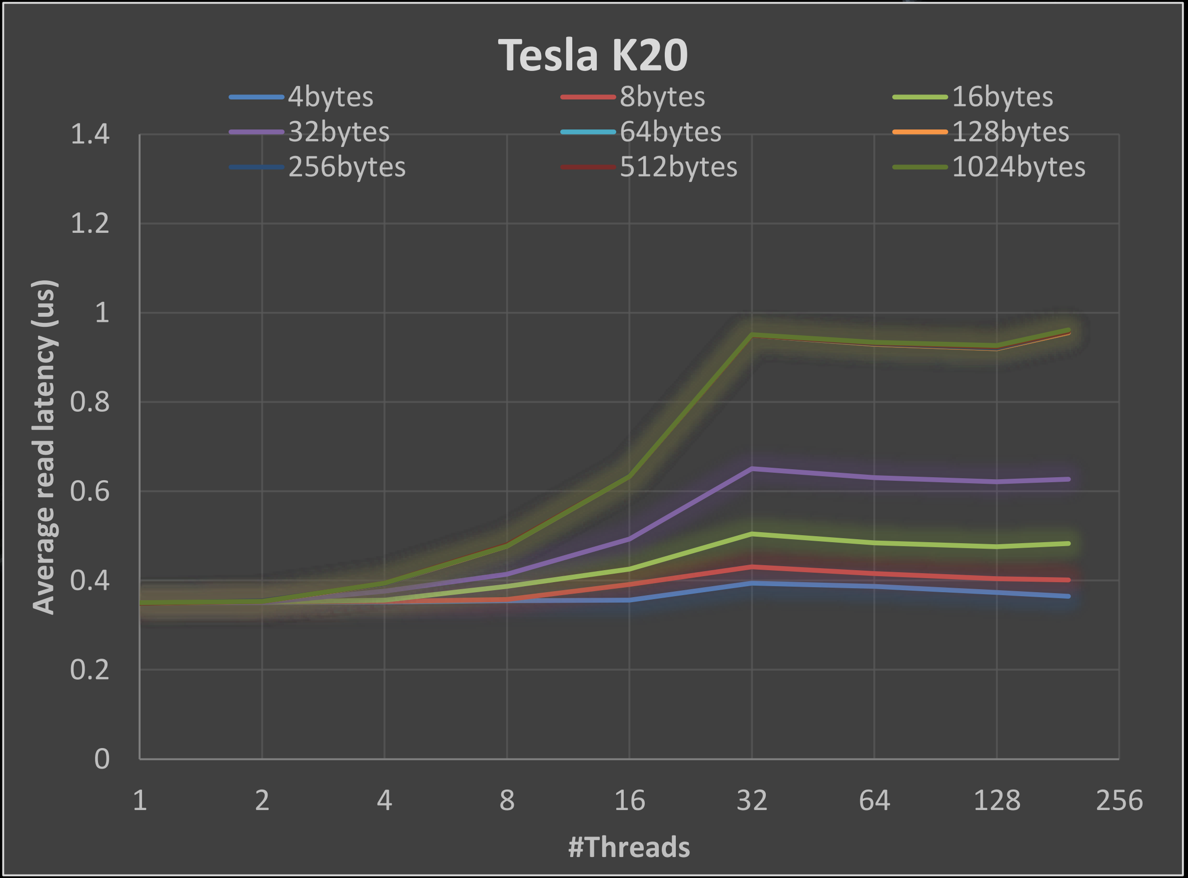
- Large offset = loss of locality. 192 threads+ Large offset = scheduler competition!



Our Benchmarks

Micro-benchmark #3: Memory Coalescing

➢ **No competition – however, overall latency is larger.**



Our Benchmarks

Other benchmarks...

Today's Topics

- Background
- Vision
- Our benchmarks
- Conclusions + Future Work

Conclusions

- Understanding GPUs performance + power = understanding microarchitecture!
- ... However microarchitecture is usually kept secret.
- Memory access patterns must be taken under considerations
- Loss of locality, resource competition , synchronizations →significant side-effects
- Side-effects differ between GPU platforms (newer is not always better!)

Future Work

- Extend the focused benchmarks to other GPU's aspects.

- Extend the work to analyze programs' behavior and correlate them with HW characterizations

- Extend the work to other platforms such as Xeon Phi

Future Work

- Extend the focused benchmarks to other GPU's aspects.
- Extend the work to analyze programs' behavior and correlate them with HW characterizations
- Extend the work to other platforms such as Xeon Phi

QUESTIONS