Demystifying GPU Microarchitecture through Microbenchmarking

Henry Wong

Henry Wong, Misel-Myrto Papadopoulou,
Maryam Sadooghi-Alvandi, Andreas Moshovos
University of Toronto

GPUs

- Graphics Processing Units
  - Increasingly programmable
- 10x arithmetic and memory bandwidth vs. CPU
  - Commodity hardware
  - Requires 1000-way parallelization
- NVIDIA CUDA
  - Write GPU thread in C variant
  - Compiled into native instructions and run in parallel on GPU

How Well Do We Know the GPU?

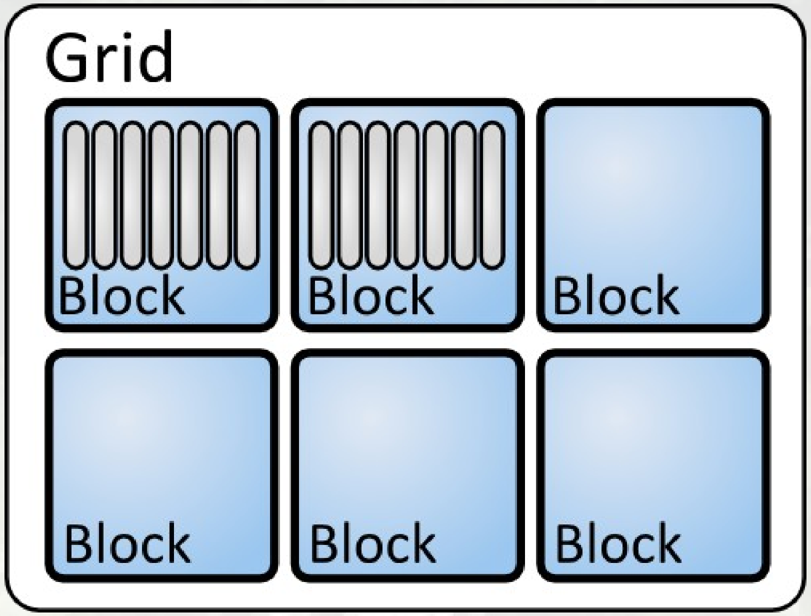
- How much do we know?
  - We know enough to write programs
  - We know basic performance tuning
    - Caches exist
    - Approximate memory speeds
    - Instruction throughput
- Why do we want to know more?
  - Detailed optimization
  - Compiler optimizations. e.g., Ocelot
  - Performance modeling. e.g., GPGPU-Sim

Outline

- Background on CUDA
- Pipeline latency and throughput
- Branch divergence
- Syncthreads barrier synchronization
- Memory Hierarchies
  - Cache and TLB organization
  - Cache sharing
- ...more in the paper

CUDA Software Model

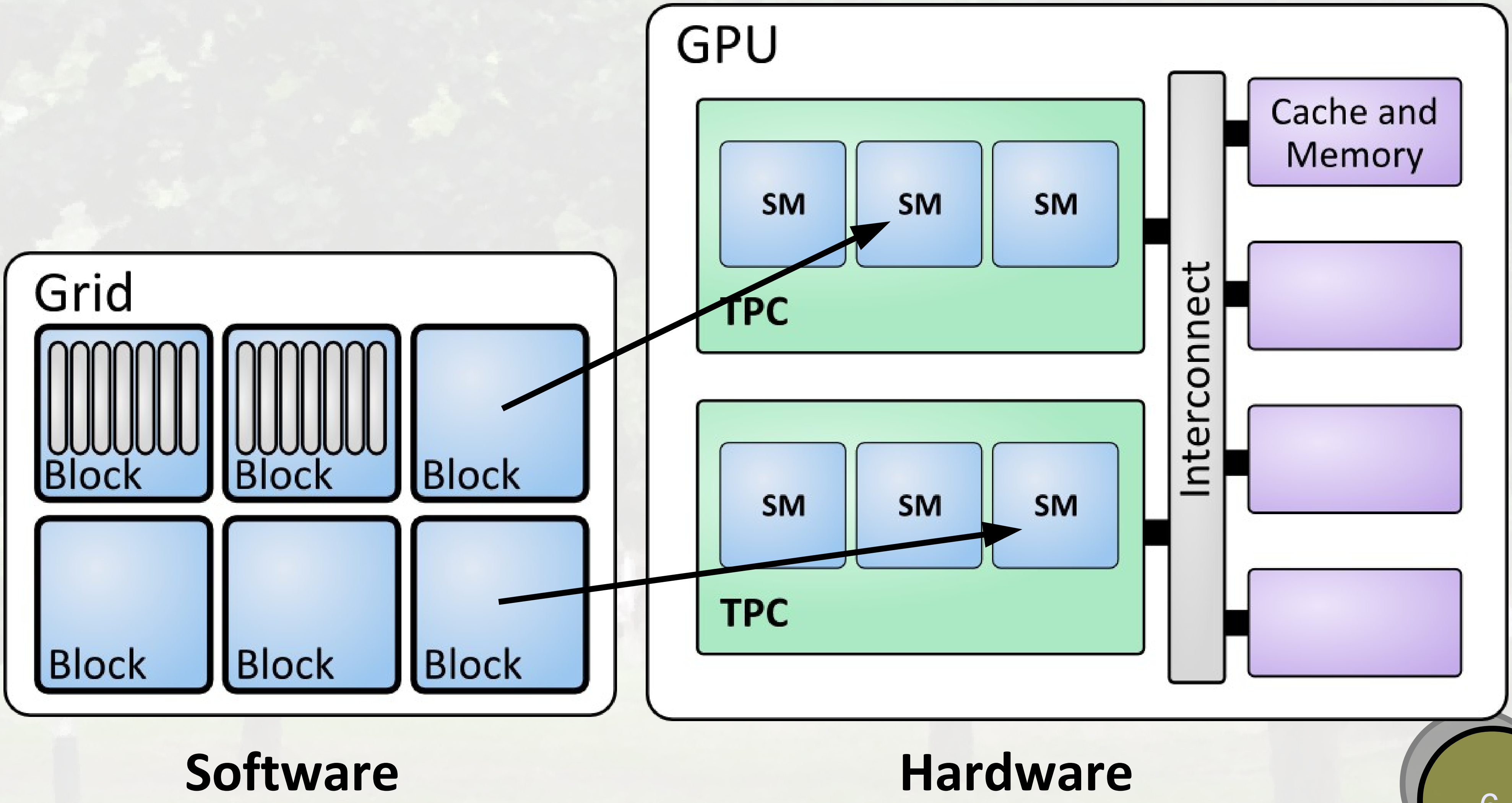
- Grid
  - Block
    - Thread



- Threads branch independently
- Threads inside Block may interact
- Blocks are independent

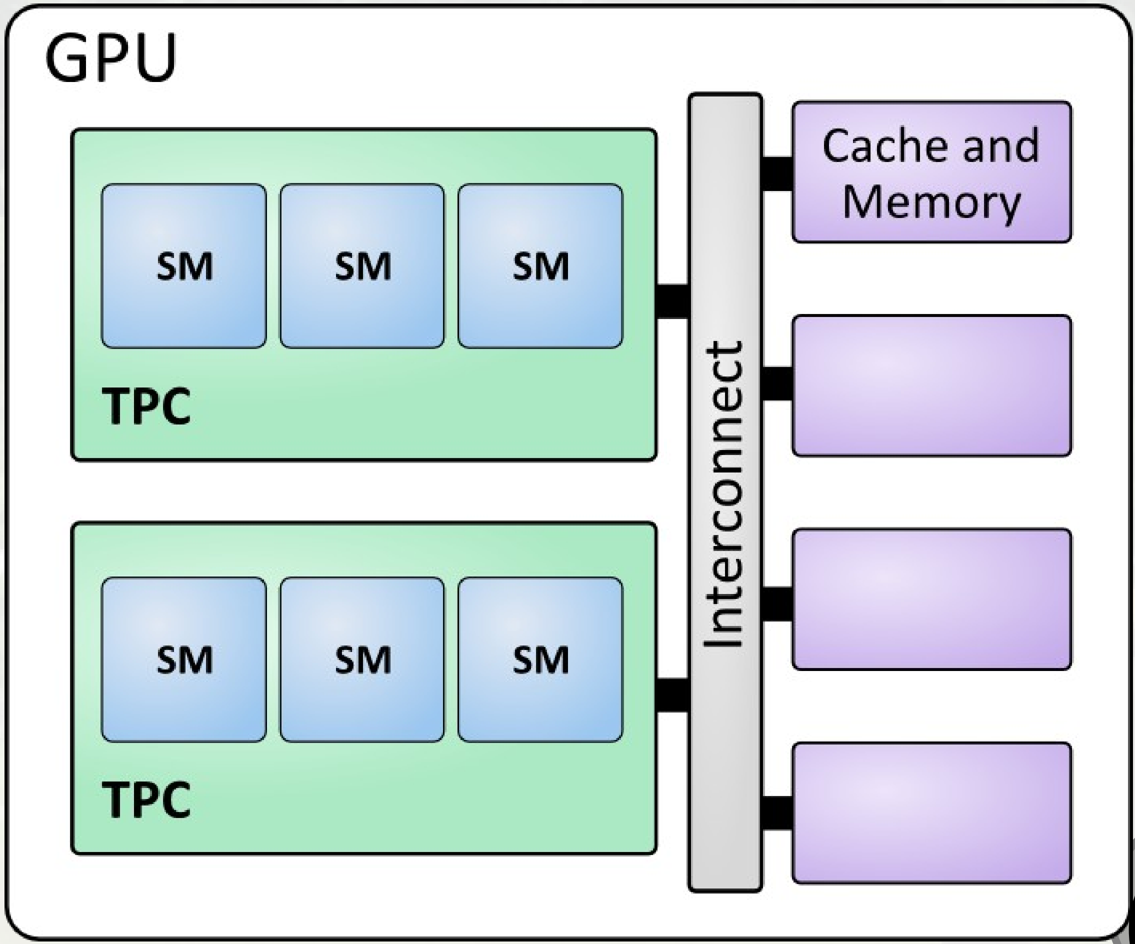
CUDA Blocks and SMs

- Blocks are assigned to SMs



CUDA Hardware Model

- SM: Streaming Multiprocessor
  - Warp: Groups of 32 threads like a vector
- TPC: Thread Processing Cluster
- Interconnect
- Memory Hierarchy



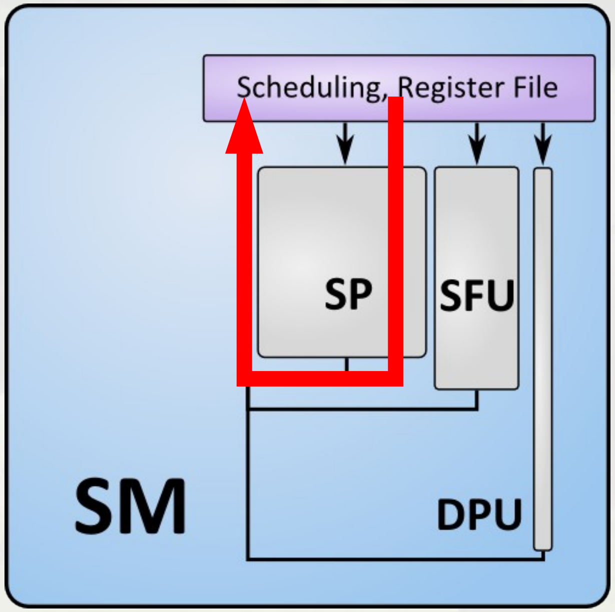
Goal and Methodology

- Aim to discover microarchitecture beyond documentation
- Microbenchmarks
  - Measure code timing and behaviour
  - Infer microarchitecture
- Measure time using clock() function
    - two clock cycle resolution

Arithmetic Pipeline: Methodology

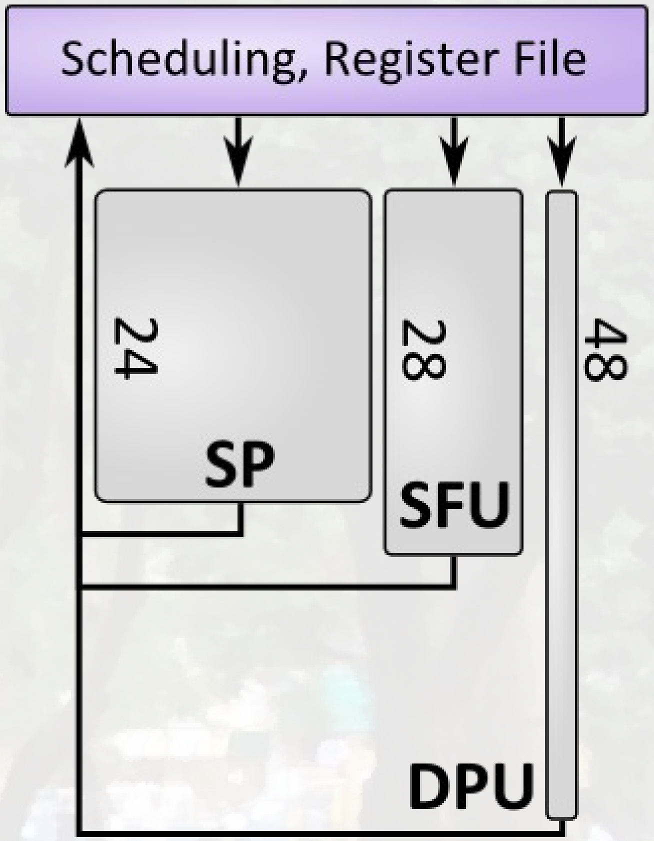
- Objective: Measure instruction latency and throughput
  - Latency: One thread
  - Throughput: Many threads
- Measure runtime of microbenchmark core
  - Avoid compiler optimizing away code
  - Discard first iteration: Cold instruction cache

```
for (i=0;i<2;i++) {
     start_time = clock();
          t1 += t2;
          t2 += t1;
          ...
          t1 += t2;
     stop_time = clock();
}
```



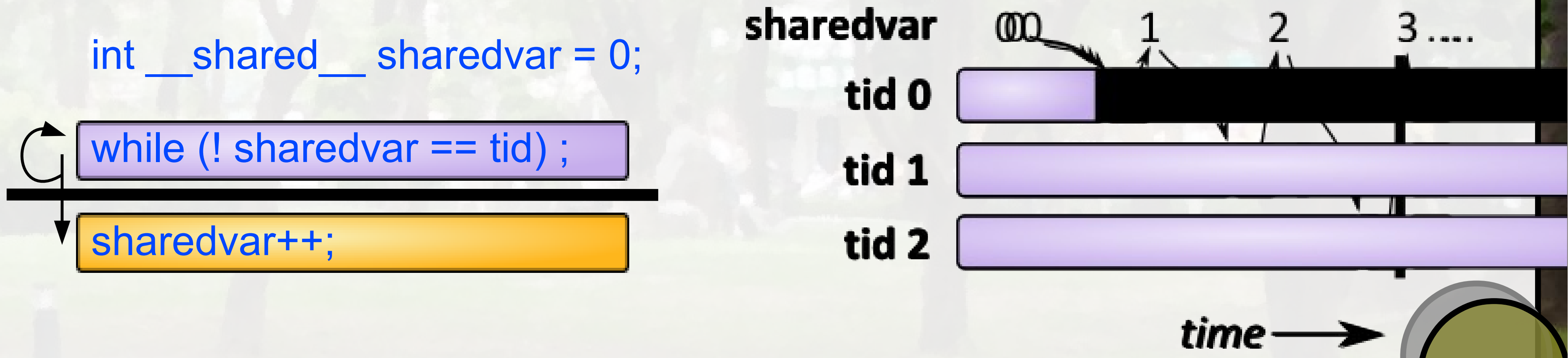
Arithmetic Pipeline: Results

- Three types of arithmetic units
  - SP: 24 cycles, 8 ops/clk
  - SFU: 28 cycles, 2 ops/clk
  - DPU: 48 cycles, 1 op/clk
- Peak SM throughput
  - 11.2 ops/clk: MUL or MAD + MUL



SIMT Control Flow

- Warps run in lock-step but threads can branch independently
- Branch divergence
  - Taken and fall-through paths are serialized
  - Taken path (usually "else") executed first
  - Fall-through path pushed onto a stack

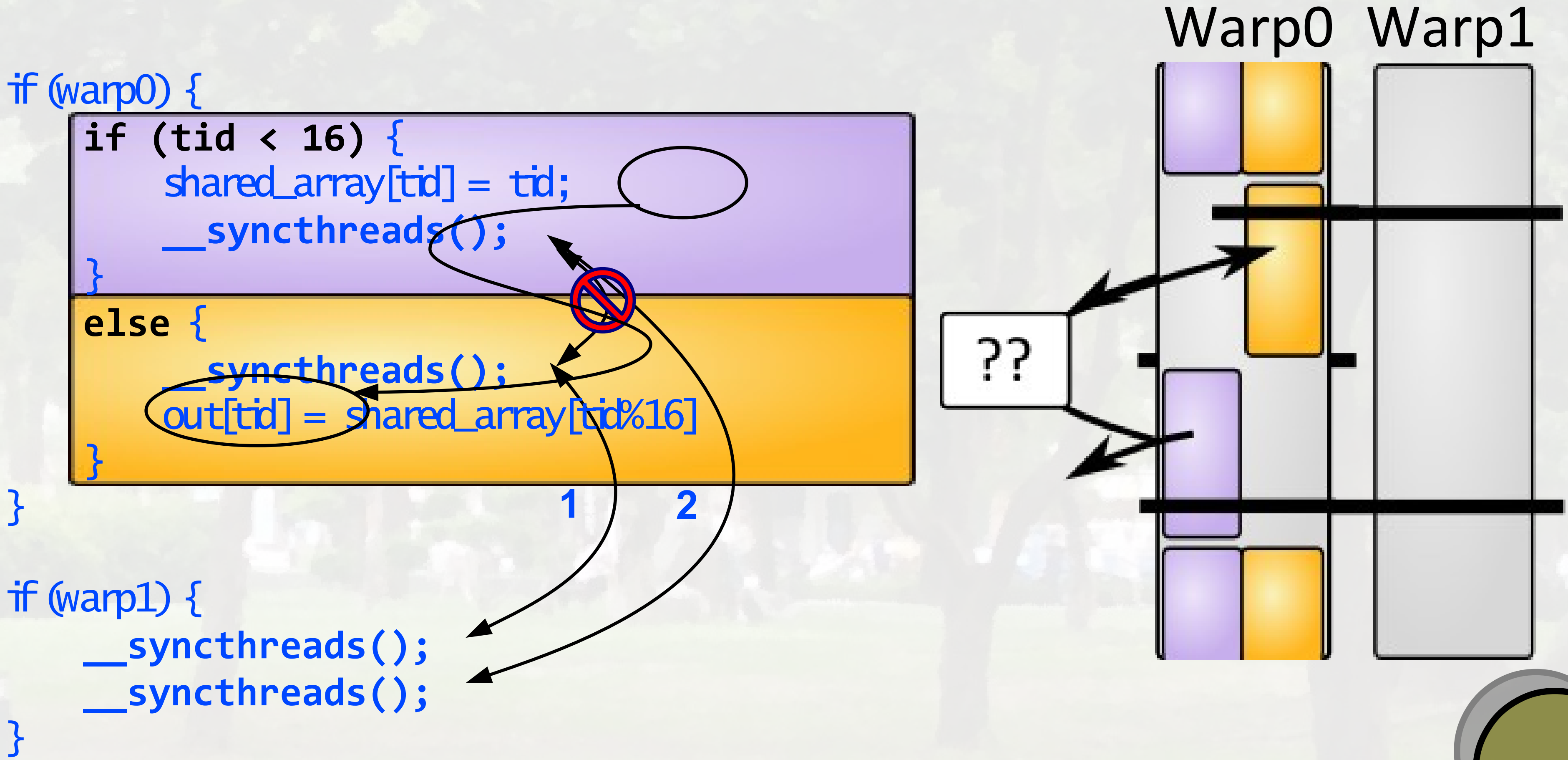


```
int __shared__ sharedvar = 0;

while (! sharedvar == tid) ;
sharedvar++;
```

Barrier Synchronization

- Syncthreads: Synchronizes Warps, not threads
  - Does not resync diverged warps
  - Will sync with other warps

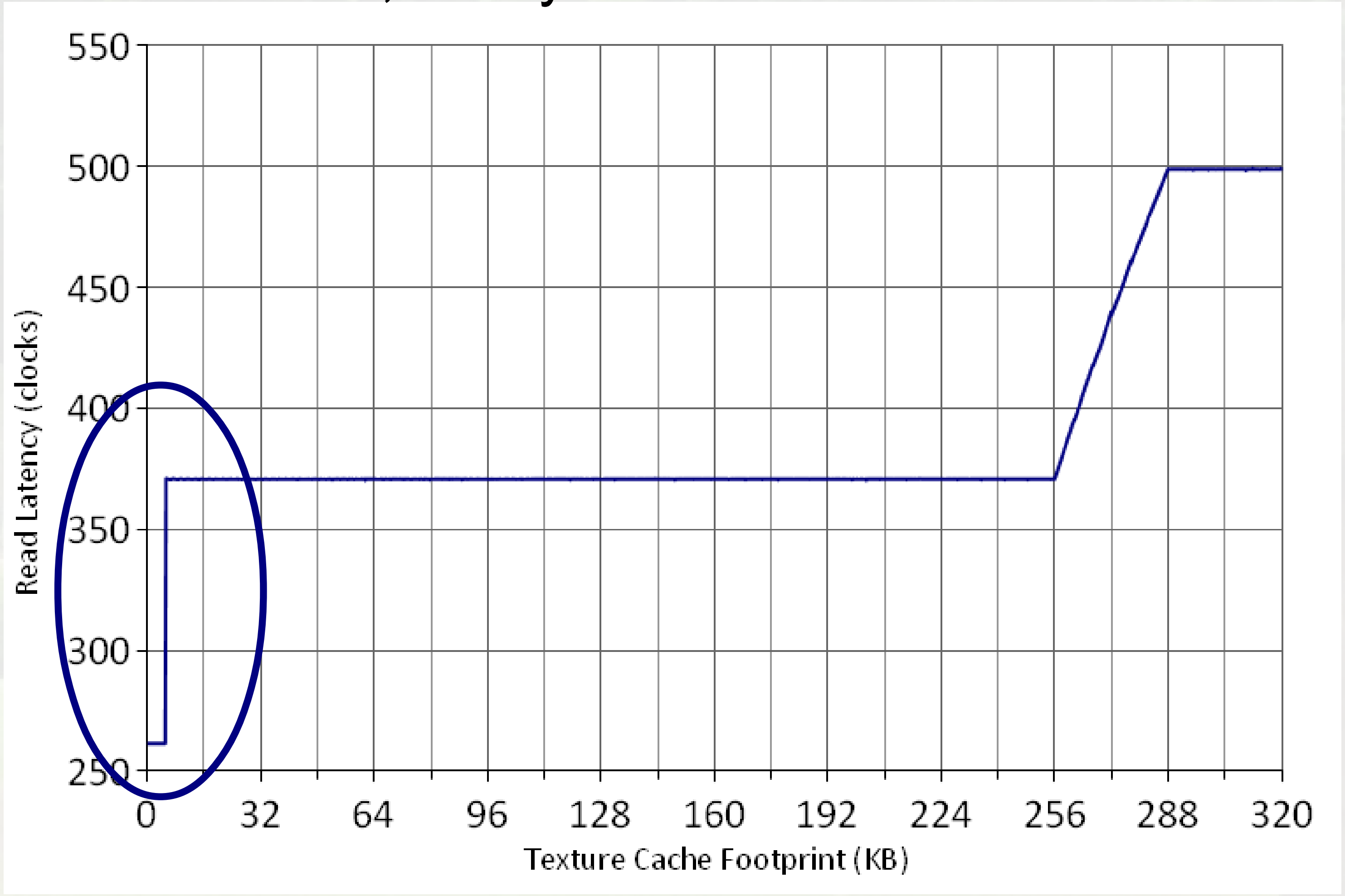


```
if (warp0) {
    if (tid < 16) {
        shared_array[tid] = tid;
        __syncthreads();
    }
    else {
        __syncthreads();
        out[tid] = shared_array[tid%16]
    }
}

if (warp1) {
    __syncthreads();
    __syncthreads();
}
```

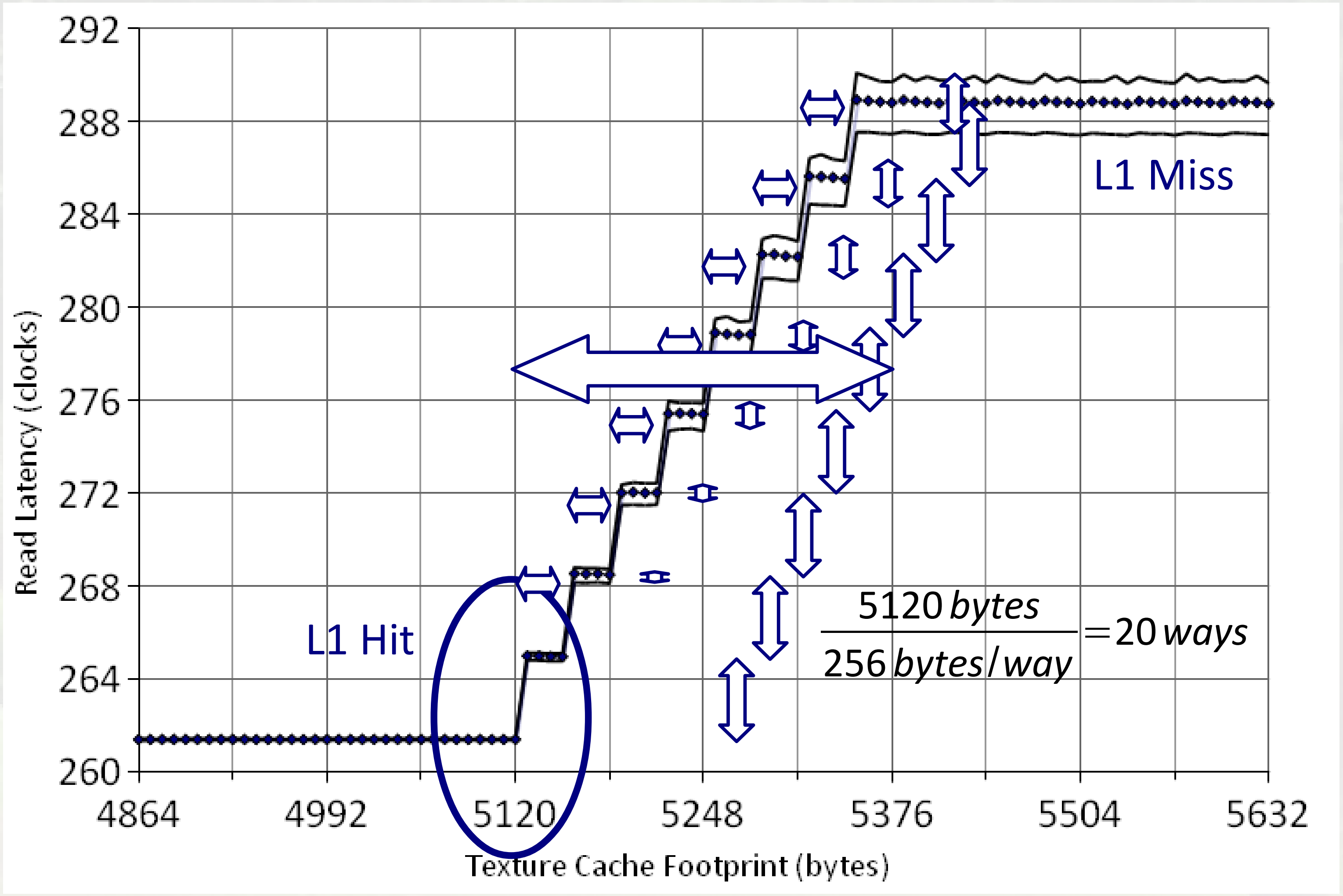
Texture Memory

- Average latency vs. memory footprint, stride accesses
  - 5 KB L1 cache, 20-way
  - 256 KB L2 cache, 8-way



Texture L1

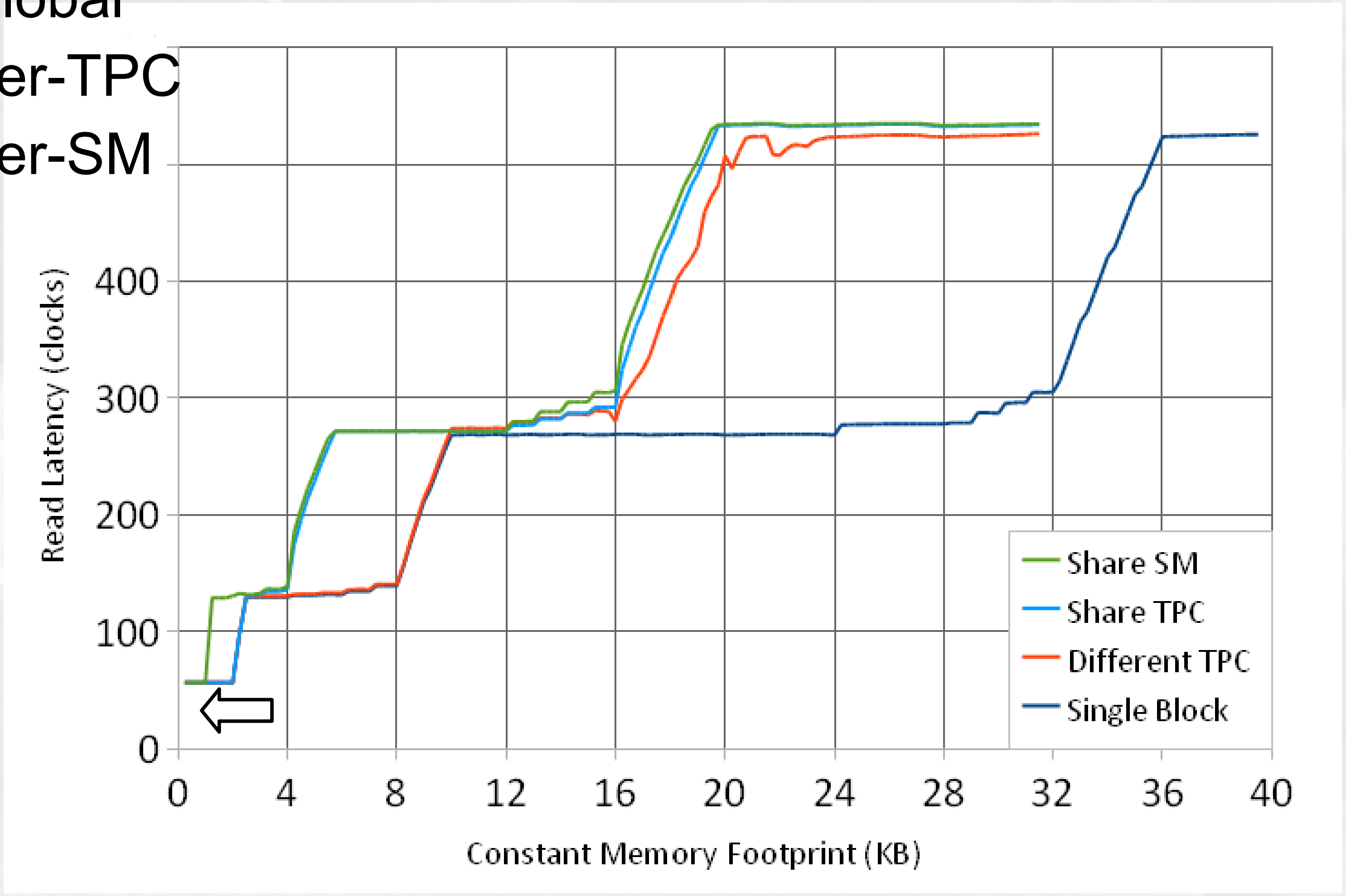
- 5 KB, 32-byte lines, 8 cache sets, 20-way set associative
- L2 is located across a non-uniform interconnect



$$\frac{5120\,bytes}{256\,bytes/way} = 20\,ways$$

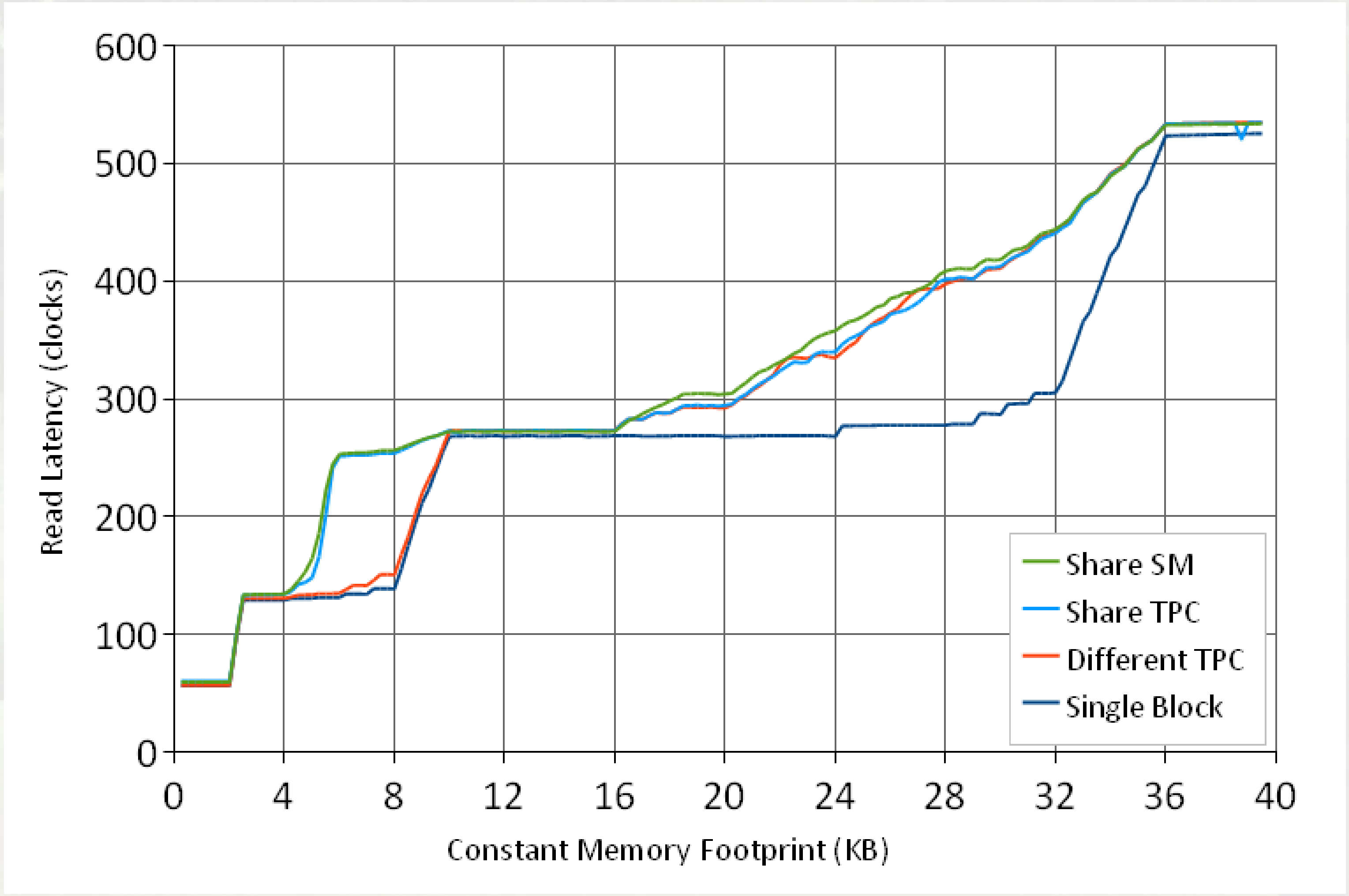
Constant Cache Sharing

- Three levels of constant cache
- Two Blocks accessing constant memory
  - L3: Global
  - L2: Per-TPC
  - L1: Per-SM



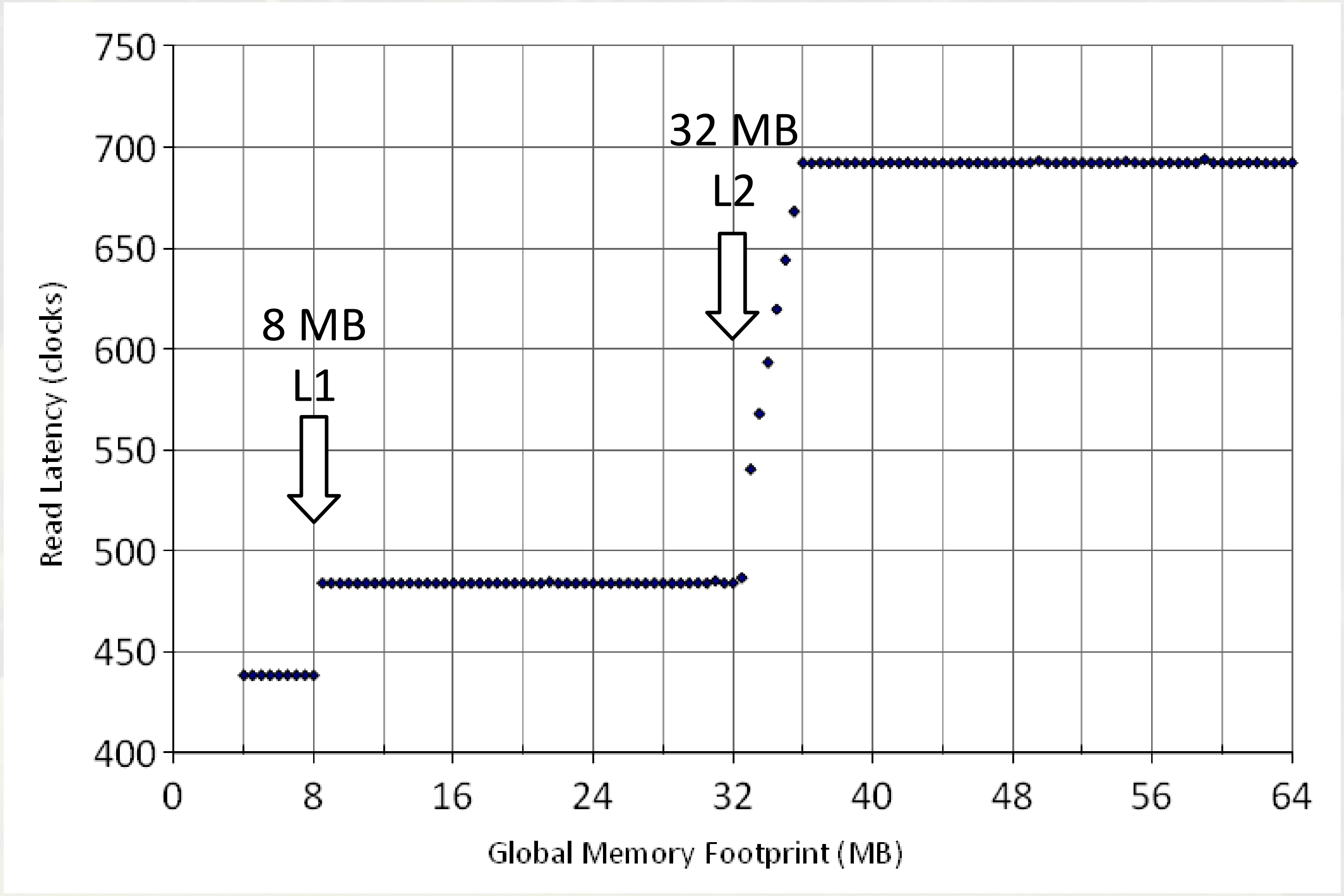
Constant and Instruction Cache Sharing

- Two different types of accesses
- L2 and L3 are Constant *and* Instruction caches



Global Memory TLBs

- 8 MB L1, 512 KB line, 16-way
- 32 MB L2, 4 KB line, 8-way
  - L2 non-trivial to measure



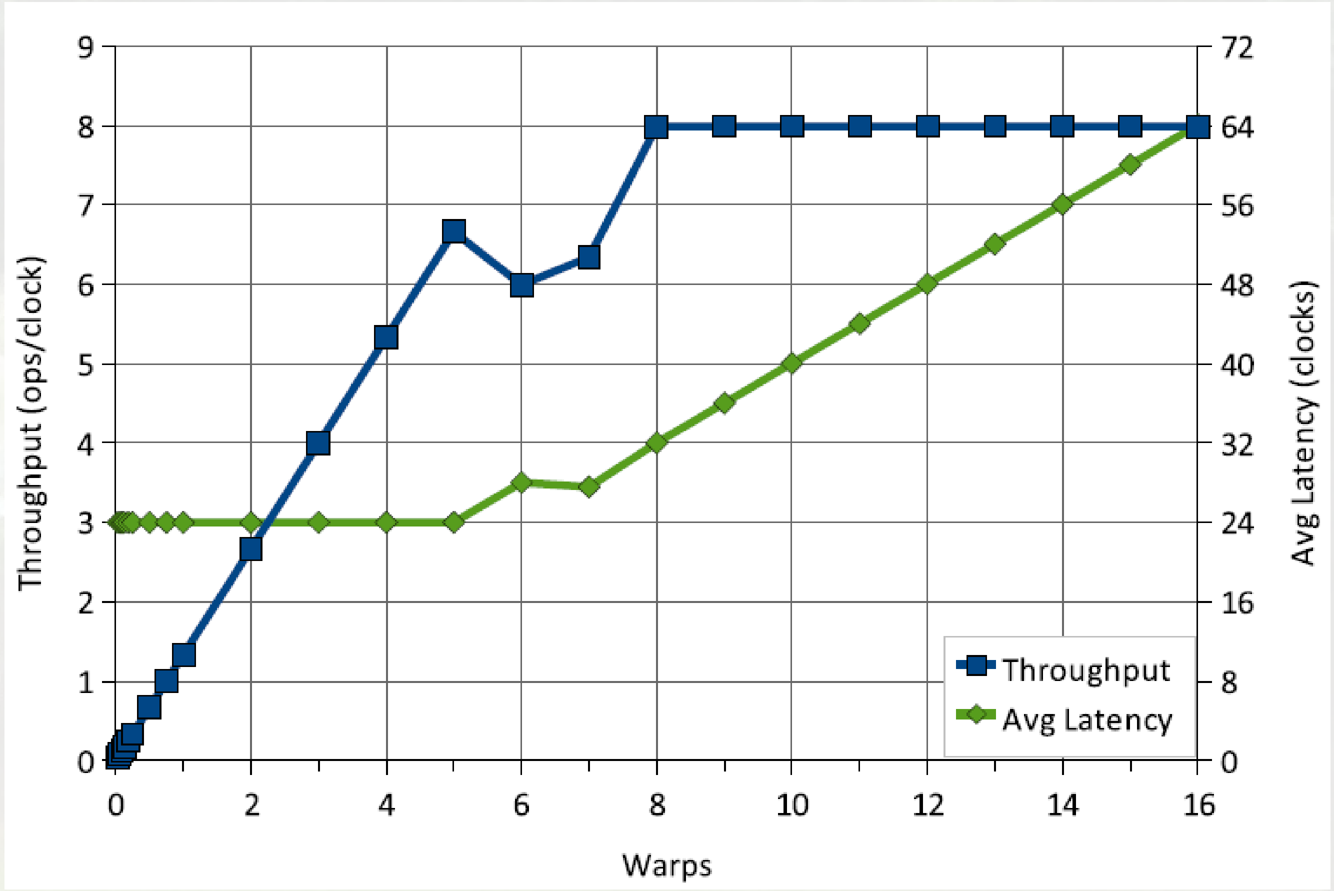
Conclusions

- Microbenchmarking reveals undocumented microarchitecture features
  - Three arithmetic unit types: latency and throughput
  - Branch divergence using a stack
  - Barrier synchronization on warps
  - Memory Hierarchy
    - Cache organization and sharing
    - TLBs
- Applications
  - Measuring other architectures and microarchitectural features
  - For GPU code optimization and performance modeling

Questions...

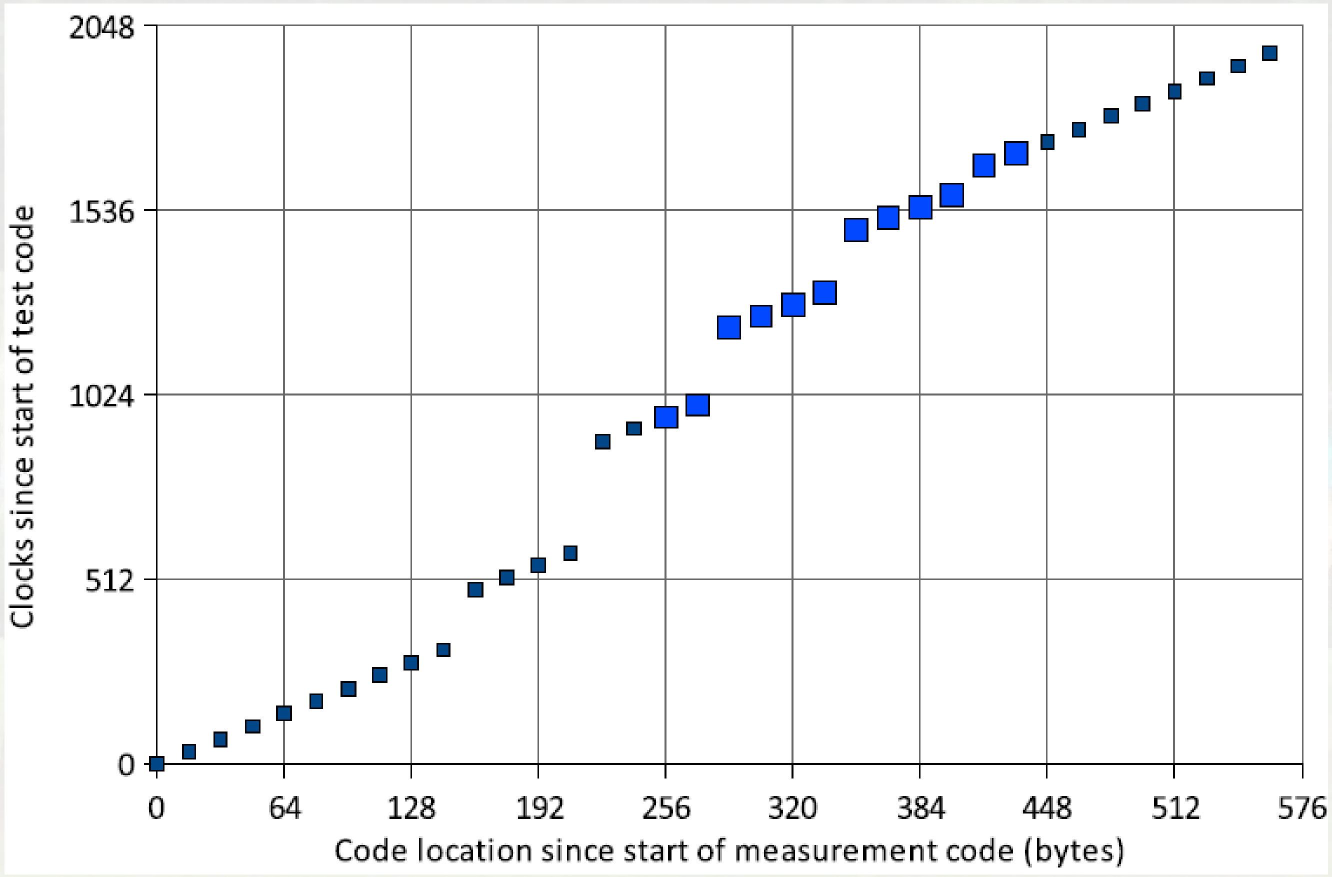
Filling the SP Pipeline

- 6 warps (24 clocks, 192 “threads”) should fill pipeline
  - 2 warps if sufficient instruction-level independence (not shown)
- 24-cycle SP pipeline latency
- Fair scheduling on average



Instruction Fetch?

- Capture burst of timing at each iteration, *then* average
  - One thread measures timing
  - Other warps thrash one line of L1 instruction cache
- Instructions are fetched from L1 in groups of 64 bytes


Clocks since start of test code
Code location since start of measurement code (bytes)