Benchmarking the Memory Hierarchy of Modern GPUs

In 11th IFIP International Conference on
Network and Parallel Computing

Xinxin Mei, Kaiyong Zhao, Chengjian Liu, Xiaowen Chu

CS Department, Hong Kong Baptist University

September 19, 2014

Outline

Outline

GPU ARCHITECTURE

GPU is a SIMD parallel many-core architecture

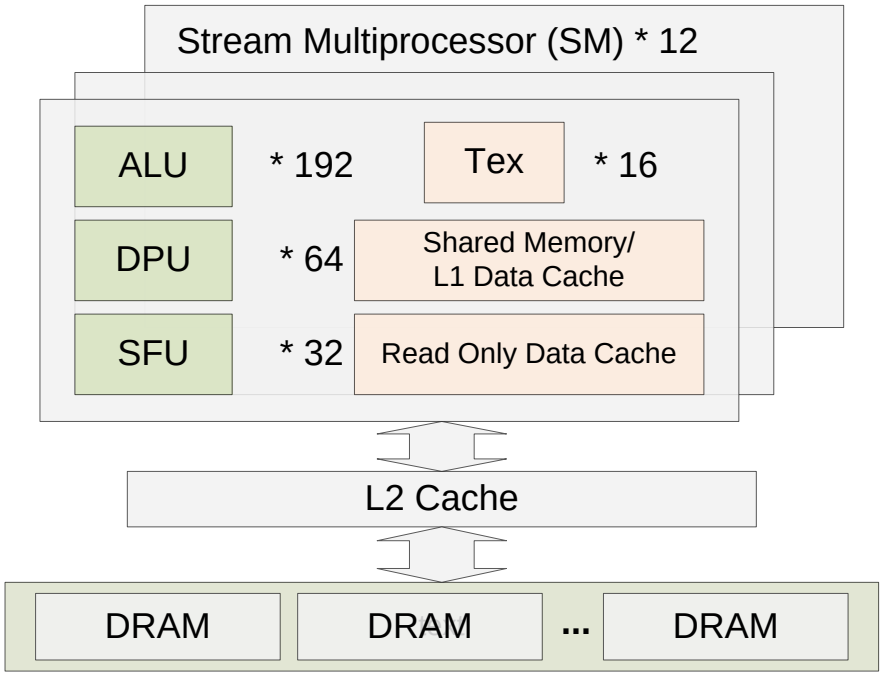


Figure: Block Diagram of GeForce 780

GPU Memory Hierarchy

Table: GPU Various Memory Spaces[1]

Memory	Type	Location	Cached	Lifetime
Register	R/W	on-chip	no	per-thread
Shared Memory	R/W	on-chip	no	per-block
Constant Memory	R	off-chip	yes	host allocation
Texture Memory	R	off-chip	yes	host allocation
Local Memory	R/W	off-chip	yes	per-thread
Global Memory	R/W	off-chip	yes[2]	host allocation

[1] Sorted by their normal accessing time in ascending order

[2] Cached local/global memory accesses are for devices of compute capacity 2.0 above only

GPU Memory Hierarchy

Table: GPU Various Memory Spaces[1]

Memory	Type	Location	Cached	Lifetime
Register	R/W	on-chip	no	per-thread
Shared Memory	R/W	on-chip	no	per-block
Constant Memory	R	off-chip	yes	host allocation
Texture Memory	R	off-chip	yes	host allocation
Local Memory	R/W	off-chip	yes	per-thread
Global Memory	R/W	off-chip	yes[2]	host allocation

Memory accesses have long been the bottleneck of further performance enhancement.

[1] Sorted by their normal accessing time in ascending order

[2] Cached local/global memory accesses are for devices of compute capacity 2.0 above only

Target Structure

Study GPU Memory hierarchy

The most popular three:

1. Shared memory
2. Global memory
3. Texture memory

Characteristics including:

1. Memory access latency
2. Unknown **cache mechanism**

REVIEW: CACHE STRUCTURE

Cache: fast back-up memory space
Set-associative, LRU

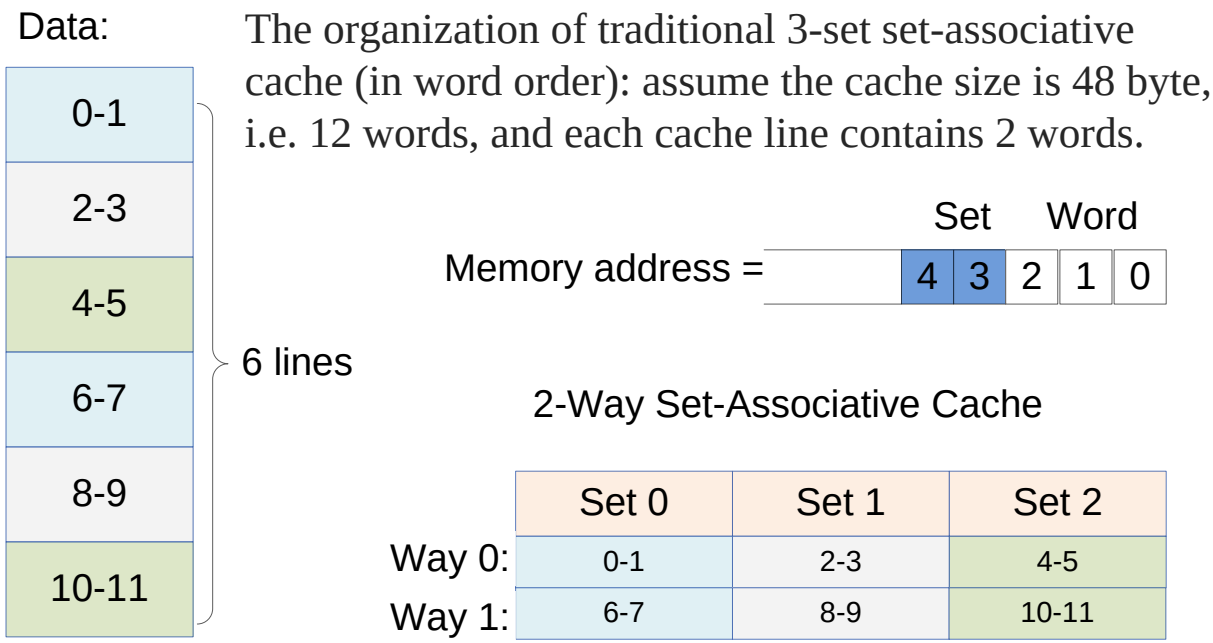


Figure: Typical Set-Associative CPU Cache Addressing

P-CHASE BENCHMARK

P-Chase: stride memory access

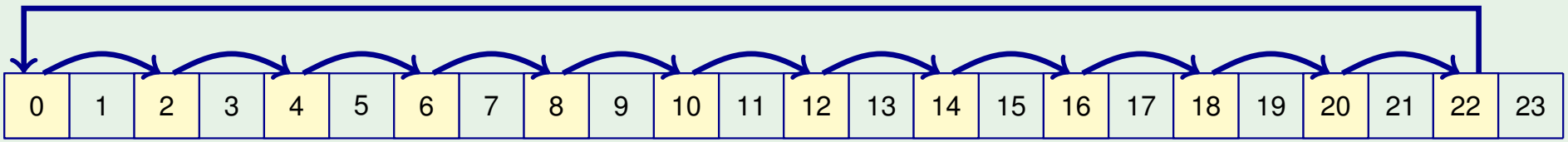
Pseudo code

```
for(i=0;i<iteration;i++)
    i=A[i];
```

Initialization

```
for(i=0;i<array_size;i++)
    A[i]=(i+stride)% array_size;
```

Memory access process



array: A, array_size: 24, stride: 2, iteration: 12

- cache miss: stride $\geq$ cache line size; array_size $>$ cache size
- cache hit: stride $<$ cache line size; array_size $\leq$ cache size

P-CHASE BENCHMARK

P-Chase: stride memory access

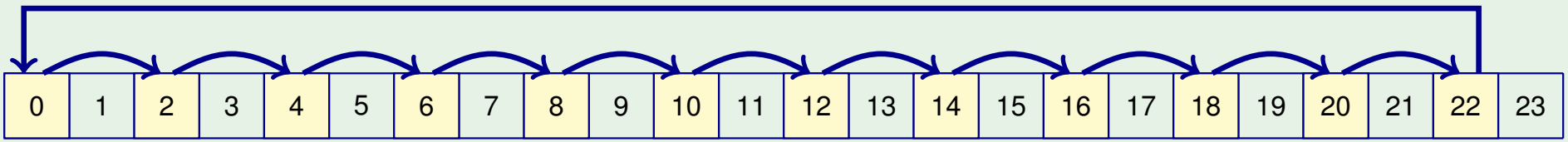
Pseudo code

```
for(i=0;i<iteration;i++)
    i=A[i];
```

Initialization

```
for(i=0;i<array_size;i++)
    A[i]=(i+stride)% array_size;
```

Memory access process



array: A, array_size: 24, stride: 2, iteration: 12

- cache miss: stride $\geq$ cache line size; array_size $>$ cache size
- cache hit: stride $<$ cache line size; array_size $\leq$ cache size

Average memory access time reflects the cache structure!

LITERATURE REVIEW

cache line size: 32 bytes

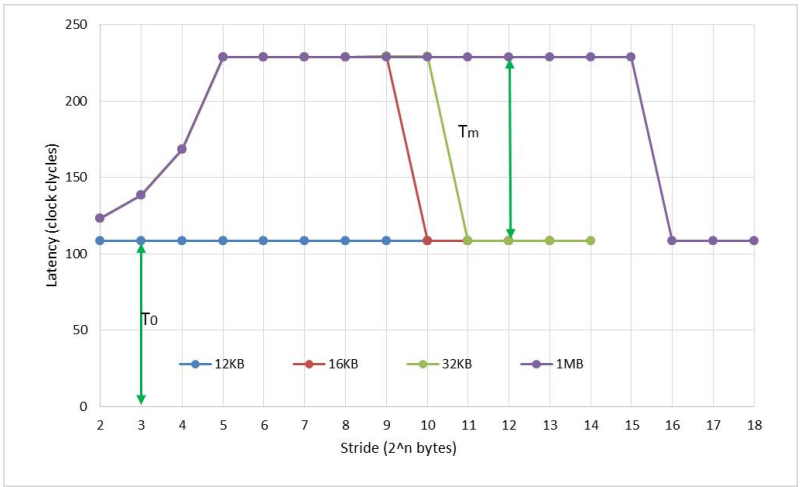


Figure: Kepler Texture L1 Cache (result of Saavedraet1992)

cache line size: 128 bytes

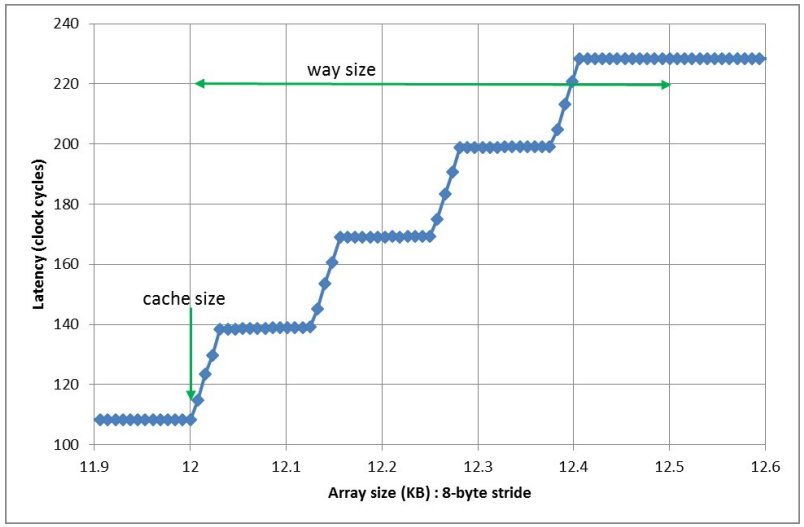


Figure: Kepler Texture L1 Cache (result of Wong2010)

Cache line sizes are **contradictory!**

Average memory latency hides some details

Motivation

- Hardware is upgraded
  - Global memory was not cached
  - Memory access time was much longer
  - ...
- Traditional P-Chase bases on CPU cache model
  - GPU cache could be different
  - Observe every latency rather than average one

Fine-grained benchmark

Record time consumption of every array element's access time

Outline

DESIGN

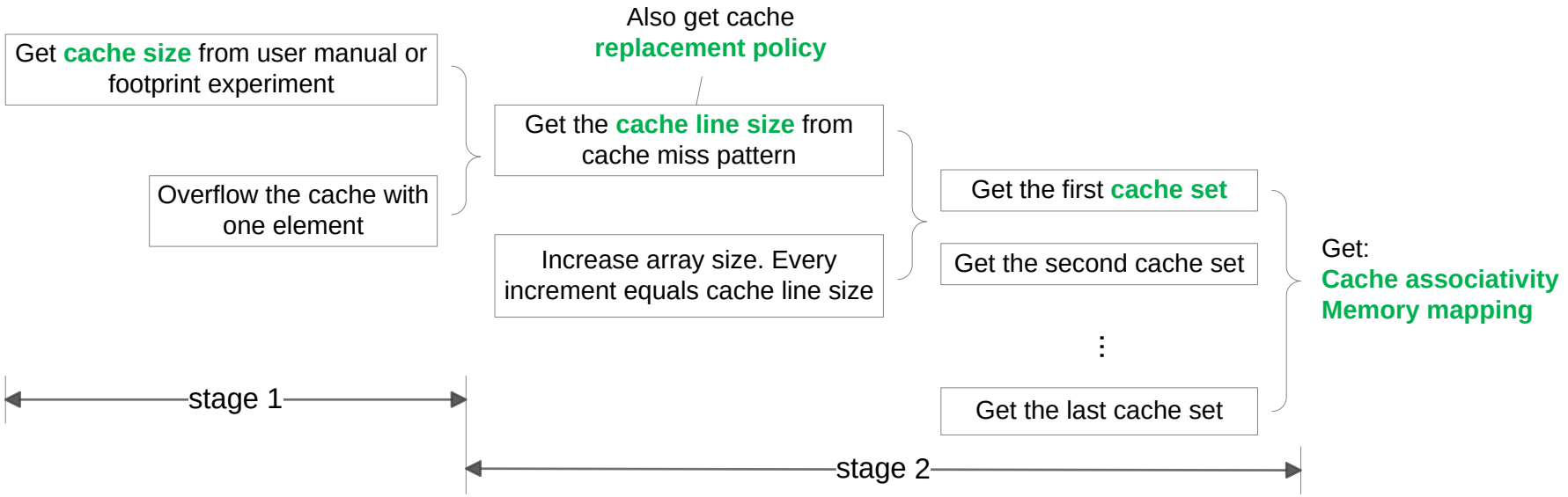
- Storage: shared memory
  - On-chip, write is prompt
  - 48 KB per SM
    Declare two spaces
    1. s_tvalue: current element access time
    2. s_index: index for next memory access
- Timing: clock()
  Store the value after it is used!

Pseudo Code

```
__global__ void KernelFunction(){
__shared__ unsigned int s_tvalue [ ] ;
__shared__ unsigned int s_index [ ] ;
for ( k = 0 ; k < iterations ; k++) {
        start time = clock( ) ;
        j = my_array [ j ] ;
        // store the element index
        s_index [ k ]= j ;
        end_time = clock ( ) ;
        // store the element access latency
        s_ tvalue [ k ] = end_time-start_time ;
       }
}
```

DIRECTIONS

Flowchart of applying our fine-grained benchmark:



Two stages

1. stride = 1 element, array_size = cache size + 1 element
2. stride = 1 cache line, array_size = cache size + 1:n cache lines

Outline

Bank Conflict I

- Normal memory latency: $\sim$ 50 clock cycles
- Shared memory is organized as memory banks

Bank conflict:

- Two or more threads in the same warp visit memory spaces belong to the same shared memory bank
- Stride memory access

Pseudo code

```
data = threadIdx.x * stride;
```

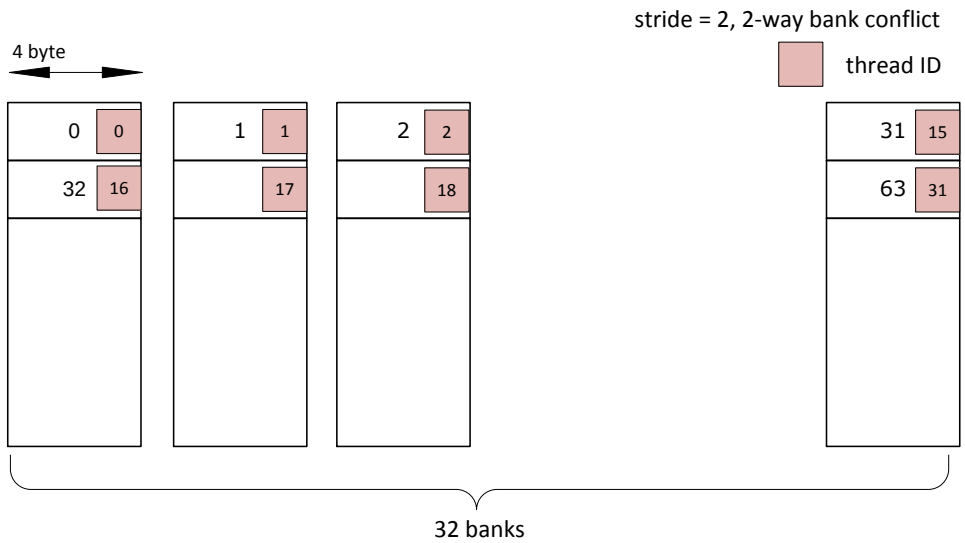


Figure: 2-way Shared Memory Bank Conflict Caused By Stride Memory Access

Bank Conflict II

Bank conflict latency is much longer

Memory requests are sequentially executed!

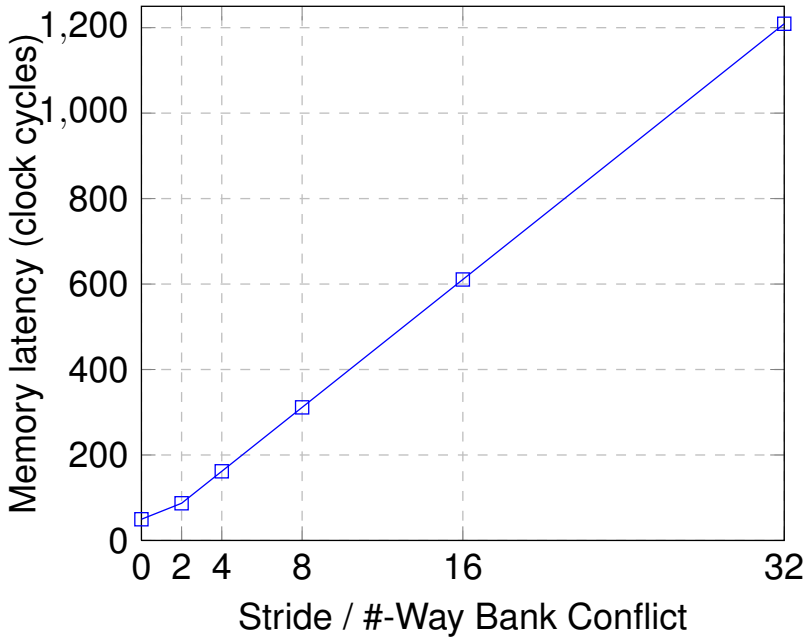


Figure: Bank Conflict Memory Latency of Fermi

BANK CONFLICT III

- **Kepler** outperforms **Fermi** in terms of avoiding shared memory bank conflict by introducing 8-byte mode shared memory bank

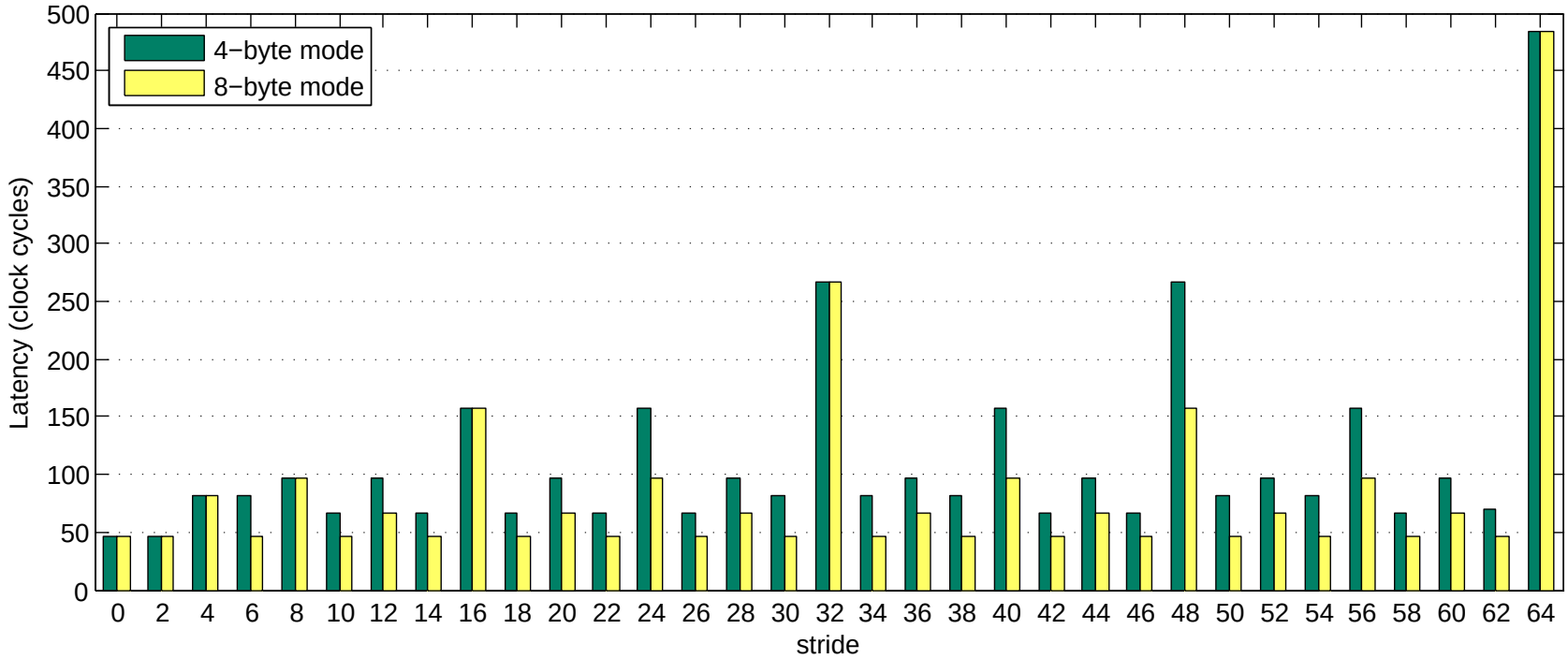


Figure: Memory Latency of Kepler Shared Memory

Global Memory: An Overview I

- Global memory access
  - Kepler: cached in L2 data cache
  - Fermi: cached in L1 and L2 data cache, L1 can be disabled
  - Two levels of TLB
- Memory latency exhibition
  Use our fine-grained benchmark with specialized initialization

GLOBAL MEMORY: AN OVERVIEW II

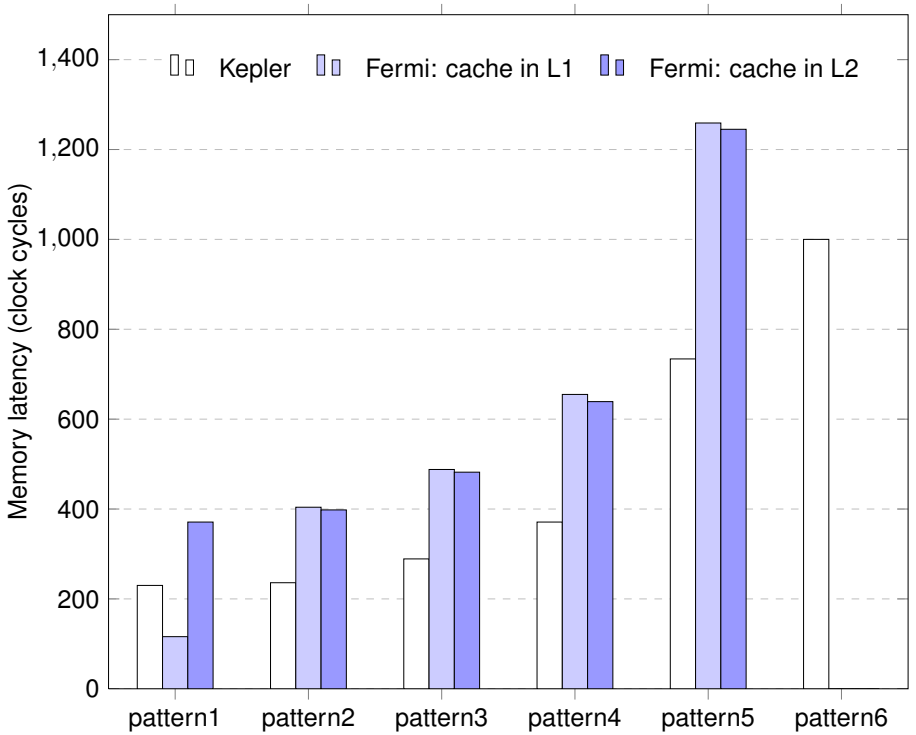


Figure: Global Memory Access Latencies

Table: Global Memory Access Patterns

Pattern	Data cache	TLB
1	hit	L1 hit
2	hit	L1 miss, L2 hit
3	hit	L2 miss
4	miss	L1 hit
5	miss	L2 miss
6	miss	L2 miss [a]

[a] page table "miss": switch between tables

GLOBAL MEMORY: AN OVERVIEW III

Observations

1. Big gap between pattern1 and pattern2 of Fermi: cached in L1
   $\Longrightarrow$ Both L1 TLB and L2 TLB are off-chip
2. Differences between enable/disable L1 of Fermi
   $\Longrightarrow$ Cached in L1 brings some extra time consumption
3. Kepler outperforms Fermi in terms of
   i. cache miss penalty
   ii. L1/L2 TLB miss penalty
   iii. memory latency (when they both cache in L2)
4. Kepler page table needs context switch
   (only 512 MB of page entries are activated)

FERMI L1 DATA CACHE

- Fermi L1 data cache structure [3]
  - cache size: 16 KB
  - cache line size: 128 byte
  - set associative: 4-way, 32-set

- Cache addressing: non-conventional
  - One "hot" cache way is more frequently replaced
  - Replacement probability: $(\frac{1}{6}, \frac{1}{2}, \frac{1}{6}, \frac{1}{6})$

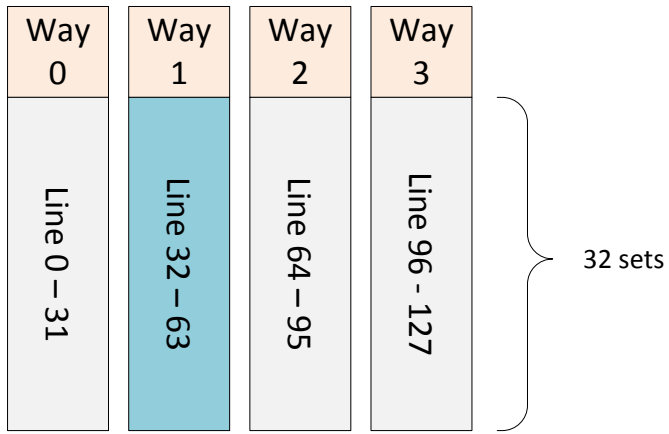


Figure: Fermi L1 Data Cache Structure

[3] This is the default setting. The Fermi L1 data cache can be configured as 48 KB, and the corresponding way number is 6.

TLB

- The page size of both Fermi and Kepler are 2 MB (by brute force experiments)
- The TLB structure of Fermi and Kepler is the same

1. L1 TLB: 16 entries, fully-associative
2. L2 TLB: 65 entries, set-associative

Non-uniform sets

1 "big" set: 17 entries
6 normal sets: 8 entries

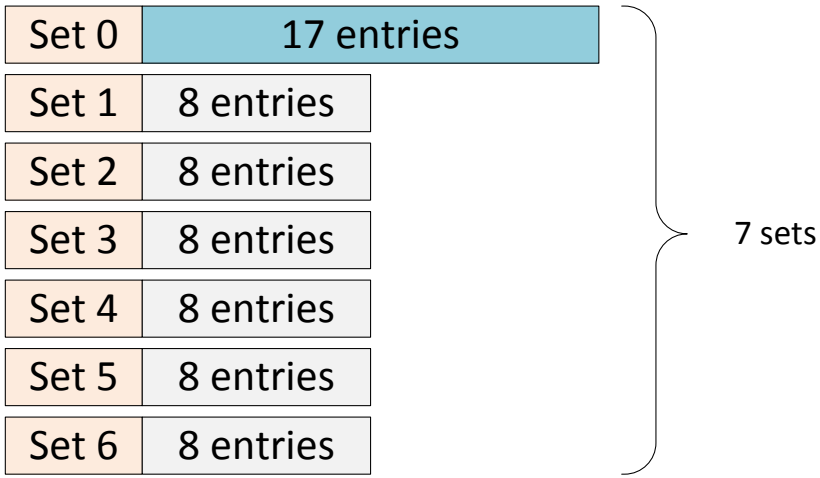


Figure: TLB Structure of Fermi & Kepler

TEXTURE MEMORY: AN OVERVIEW

- Memory latency

Table: Texture Memory Access Latency of Fermi & Kepler

Device	Texture cache		Global cache	
	L1 hit	L1 miss, L2 hit	L1 hit	L1 miss, L2 hit
Fermi	240	470	116	404
Kepler	110	220	–	230

$\Longrightarrow$ Fermi texture memory management is expensive!

- The same L2 cache, TLB with global memory
- Different L1 cache: 12 KB, 32-byte cache line, 4 sets

TEXTURE L1 CACHE

2D spacial locality optimized texture L1 cache addressing

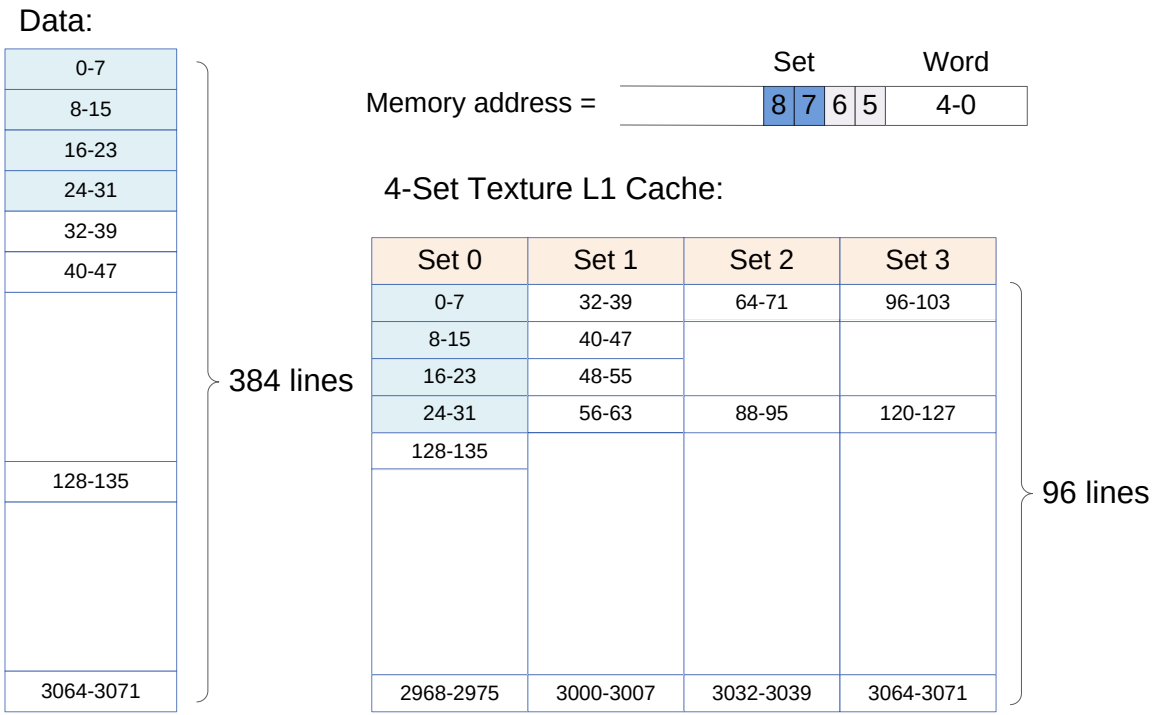


Figure: Fermi & Kepler Texture L1 Cache Addressing

For the typical set-associative mapping, 5-6th bits define the cache set, but here 7-8th bits instead. It looks like a 128-byte, 24-way conventional set-associative cache.

Outline

SUMMARY

1. Study memory hierarchy of current GPU: ***Fermi*** and ***Kepler***
2. Expose detailed GPU memory features [4]
   - Latency of shared memory bank conflict
   - Latency of Fermi/Kepler global memory accesses
   - Structure of Fermi L1 data cache
   - Structure of Fermi/Kepler TLBs
   - Latency of Fermi/Kepler texture memory accesses
   - Structure of Fermi/Kepler texture L1 cache
   - Structure of Kepler read-only data cache

[4] This is an open-source project. The testing files are at `http://www.comp.hkbu.edu.hk/~chxw/gpu_benchmark.html`.
More technical details can be found in our submitted paper.

Conclusions

1. GPU cache design is much different from CPU's
2. Fermi and Kepler outperform old architecture

Contributions

1. Design a benchmark with some novelty
2. Unveil unknown GPU cache characteristics


Q&A