Local Memory

• Name refers to memory where registers and other thread-data is spilled
  – Usually when one runs out of SM resources
  – “Local” because each thread has its own private area

• Details:
  – Not really a “memory” - bytes are stored in global memory
  – Differences from global memory:
    • Addressing is resolved by the compiler
    • Stores are cached in L1
**LMEM Access Operation**

- **A store writes a line to L1**
  - If evicted, that line is written to L2
  - The line could also be evicted from L2, in which case it’s written to DRAM

- **A load requests the line from L1**
  - If a hit, operation is complete
  - If a miss, then requests the line from L2
    - If a miss, then requests the line from DRAM

- **A store always happens before a load**
  - Only GPU threads can access LMEM addresses
Fermi Memory Hierarchy

- **SM-0**: Registers → L1 → SMEM → L2 → Global Memory (DRAM)
- **SM-1**: Registers → L1 → SMEM → L2 → Global Memory (DRAM)
- **SM-N**: Registers → L1 → SMEM → L2 → Global Memory (DRAM)
When is Local Memory Used?

• **Register spilling**
  – Fermi hardware limit is 63 registers per thread
  – Programmer can specify lower registers/thread limits:
    • To increase occupancy (number of concurrently running threads)
    • `-maxrregcount` option to nvcc, `__launch_bounds__()` qualifier in the code
  – LMEM is used if the source code exceeds register limit

• **Arrays declared inside kernels, if compiler can’t resolve indexing**
  – Registers aren’t indexable, so have to be placed in LMEM
How Does LMEM Affect Performance?

• It could hurt performance in two ways:
  – Increased memory traffic
  – Increased instruction count

• Spilling/LMEM usage isn’t always bad
  – LMEM bytes can get contained within L1
    • Avoids memory traffic increase
  – Additional instructions don’t matter much if code is not instruction-throughput limited
General Analysis/Optimization Steps

• Check for LMEM usage
  – Compiler output
    • nvcc option: -Xptxas -v,-abi=no
    • Will print the number of lmem bytes for each kernel (only if kernel uses LMEM)
  – Profiler

• Check the impact of LMEM on performance
  – Bandwidth-limited code:
    • Check how much of L2 or DRAM traffic is due to LMEM
  – Arithmetic-limited code:
    • Check what fraction of instructions issued is due to LMEM

• Optimize:
  – Try: increasing register count, increasing L1 size, using non-caching loads
Register Spilling: Analysis

- Profiler counters:
  - \texttt{l1\_local\_load\_hit}, \texttt{l1\_local\_load\_miss}, \texttt{l1\_local\_store\_hit}, \texttt{l1\_local\_store\_miss}
  - Counted for a single SM, incremented by 1 for each 128-byte transaction

- Impact on memory
  - Any memory traffic that leaves SMs (goes to L2) is expensive
  - L2 counters of interest: read and write sector queries
    - Actual names are longer, check the profiler documentation
    - Incremented by 1 for each 32-byte transaction
  - Compare:
    - Estimated L2 transactions due to LMEM misses in all the SMs
      - \(2 \times (\text{number of SMs}) \times 4 \times \text{l1\_local\_load\_miss}\)
      - 2: load miss implies a store happened first
      - Number of SMs: \text{l1\_local\_load\_miss} counter is for a single SM
      - 4: local memory transaction is 128-bytes = 4 L2-transactions
    - Sum of L2 read and write queries (not misses)

- Impact on instructions
  - Compare the sum of all LMEM instructions to total instructions issued
Optimizations When Register Spilling is Problematic

• Try increasing the limit of registers per thread
  – Use a higher limit in `-maxrregcount`, or lower thread count for `__launch_bounds__`
  – Likely reduces occupancy, potentially reducing execution efficiency
    • may still be an overall win - fewer total bytes being accessed

• Try using non-caching loads for global memory
  – nvcc option: `-Xptxas -dlcm=cg`
  – Potentially fewer contentions with spilled registers in L1

• Increase L1 size to 48KB
  – Default is 16KB L1, larger L1 increases the chances for LMEM hits
  – Can be done per kernel or per device:
    • `cudaFuncSetCacheConfig()`, `cudaDeviceSetCacheConfig()`
Case Study

• Time Domain Finite Difference of the 3D Wave Equation
  – Simulates seismic wave propagation through Earth subsurface
  – Largely memory bandwidth-bound
  – Running more threads concurrently helps saturate memory bandwidth
    • Thus, to run 1024 threads per Fermi SM we specify 32 register maximum per thread

• Check for LMEM Use
  – Spills 44 bytes per thread when compiled down to 32 registers per thread

$ nvcc -arch=sm_20 -Xptxas -v,-abi=no,-dlcm=cg fwd_o8.cu -maxrregcount=32
ptxas info : Compiling entry function '_Z15fwd_3D_orderX2bILi4ELi9EEvPfS0_S0_iiiii' for 'sm_20'
ptxas info : Used 32 registers, 44+0 bytes lmem, 6912+0 bytes smem, 76 bytes cmem[0], …
Case Study: Analyze the Impact on Memory

• Using profiler counters:
  – SM counters:
    • l1_local_load_miss: 564,332
    • l1_local_load_hit: 91,520
    • l1_local_store_miss: 269,215
    • l1_local_store_hit: 13,477
    • inst_issued: 20,412,251
  – L2 query counts:
    • Read: 99,435,608
    • Write: 33,385,908
    • Total: 132,821,516

• This was on a 16-SM GPU

To get the counters use any of:
• Visual Profiler
• Command-line profiler
• NSight
Case Study: Analyze the Impact on Memory

• Using profiler counters:
  – SM counters:
    • l1_local_load_miss: 564,332
    • l1_local_load_hit: 91,520
    • l1_local_store_miss: 269,215
    • l1_local_store_hit: 13,477
    • inst_issued: 20,412,251
  – L2 query counts:
    • Read: 99,435,608
    • Write: 33,385,908
    • Total: 132,821,516

Load L1 hit rate: 13.95%
Estimated L2 queries per SM due to LMEM:
\[ 2 \times 4 \times 564,332 = 4,514,656 \]
Estimated L2 queries due to LMEM of all 16 SMs:
\[ 16 \times 4,514,656 = 72,234,496 \]
Percentage of all L2 queries due to LMEM:
\[ \frac{72,234,496}{132,821,516} = 53.38\% \]

• This was on a 16-SM GPU
Case Study: Analyze the Impact on Memory

- Using profiler counters:
  - SM counters:
    - L1 local load miss: 564,332
    - L1 local load hit: 91,520
    - L1 local store miss: 269,215
    - L1 local store hit: 13,477
    - Inst issued: 20,412,251
  - L2 query counts:
    - Read: 99,435,608
    - Write: 33,385,908
    - Total: 132,821,516

This was on a 16-SM GPU

53.38% of memory traffic between the SMs and L2/DRAM is due to LMEM (not useful from the application’s point of view).

Since application is bandwidth-limited, reducing spilling could help performance.

Load L1 hit rate: 13.95%
Estimated L2 queries per SM due to LMEM: 
\[ 2 \times 4 \times 564,332 = 4,514,656 \]

Estimated L2 queries due to LMEM of all 16 SMs: 
\[ 16 \times 4,514,656 = 72,234,496 \]
Percentage of all L2 queries due to LMEM: 
\[ 72,234,496 / 132,821,516 = 53.38\% \]
Case Study: Analyze the Impact on Instructions

• Using profiler counters:
  – SM counters:
    • l1_local_load_miss: 564,332
    • l1_local_load_hit: 91,520
    • l1_local_store_miss: 269,215
    • l1_local_store_hit: 13,477
    • inst_issued: 20,412,251
  – L2 query counts:
    • Read: 99,435,608
    • Write: 33,385,908
    • Total: 132,821,516

Total instructions due to LMEM: 938,944

Percentage of instructions due to LMEM: 938,944 / 20,412,251 = 4.60%

• This was on a 16-SM GPU
Case Study: Analyze the Impact on Instructions

• Using profiler counters:
  – SM counters:
    • l1_local_load_miss: 564,332
    • l1_local_load_hit: 91,520
    • l1_local_store_miss: 269,215
    • l1_local_store_hit: 13,477
    • inst_issued: 20,412,251
  – L2 query counts:
    • Read: 99,435,608
    • Write: 33,385,908
    • Total: 132,821,516

4.6% is not significant enough to worry about
(Removing spilling completely cannot improve performance by more than 4.6%, and then only if kernel is instruction-limited)

Total instructions due to LMEM: 938,944
Percentage of instructions due to LMEM: 938,944 / 20,412,251 = 4.60%

• This was on a 16-SM GPU
Case Study: Optimizations

• Try increasing register count
  – Remove the -maxrregcount=32 compiler option
    • 46 registers per thread, no spilling
  – Performance improved by 1.22x

• Increase L1 cache size
  – Keeping the 32 register maximum and spilling 44 bytes
  – Add cudaDeviceSetCacheConfig( cudaFuncCachePreferL1 ); call
  – L1 LMEM load hit rate improved to 98.32%
  – Estimated 1.63% of all requests to L2 were due to LMEM
    • way too small to worry about
    • 1.63 was computed as on slide 12 (not by 100% - 98.32%)
  – performance improved by 1.45x

• Application was already using non-caching loads for other reasons
Register Spilling: Summary

• Doesn’t always decrease performance, but when it does it’s because of:
  – Increased pressure on the memory bus
  – Increased instruction count

• Use the profiler to determine:
  – Bandwidth-limited codes: LMEM L1 miss impact on memory bus (to L2) for
  – Arithmetic-limited codes: LMEM instruction count as percentage of all instructions

• Optimize by
  – Increasing register count per thread
  – Increasing L1 size
  – Using non-caching GMEM loads
Questions?