

# CUDA OPTIMIZATIONS

PPAM 2011 Tutorial

Tim C. Schroeder, NVIDIA Corporation



# Outline



- Launch configuration
- Global memory throughput
- Shared memory access
- Instruction throughput / control flow



# Launch Configuration

# Launch Configuration



- **Key to understanding:**
  - Instructions are issued in order
  - A thread stalls when one of the operands isn't ready:
    - Memory read by itself doesn't stall execution
  - Latency is hidden by switching threads
    - GMEM latency: 400-800 cycles
    - Arithmetic latency: 18-22 cycles
- **How many threads/threadblocks to launch?**
- **Conclusion:**
  - Need enough threads to hide latency

# Launch Configuration



- **Hiding arithmetic latency:**
  - Need **~18** warps (**576**) threads per SM
  - Or, latency can also be hidden with independent instructions from the same warp
    - For example, if instruction never depends on the output of preceding instruction, then only **9** warps are needed, etc.
- **Maximizing global memory throughput:**
  - Depends on the access pattern, and word size
  - Need enough memory transactions in flight to saturate the bus
    - Independent loads and stores from the same thread
    - Loads and stores from different threads
    - Larger word sizes can also help (**float2** is twice the transactions of **float**, for example)

# Launch Configuration: Summary



- **Need enough total threads to keep GPU busy**
  - Typically, you'd like **512+** threads per SM
    - More if processing one fp32 element per thread
  - Of course, exceptions exist
- **Threadblock configuration**
  - Threads per block should be a multiple of warp size (**32**)
  - SM can concurrently execute up to **8** threadblocks
    - Really small threadblocks prevent achieving good occupancy
    - Really large threadblocks are less flexible
    - I generally use **128-256 threads/block**, but use whatever is best for the application
- **For more details:**
  - Vasily Volkov's GTC2010 talk "Better Performance at Lower Occupancy" (<http://www.gputechconf.com/page/gtc-on-demand.html#session2238>)



# Global Memory Throughput

# Memory Hierarchy Review



- **Local storage**
  - Each thread has own local storage
  - Mostly registers (managed by the compiler)
- **Shared memory / L1**
  - Program configurable: 16KB shared / 48 KB L1 OR 48KB shared / 16KB L1
  - Shared memory is accessible by the threads in the same threadblock
  - Very low latency
  - Very high throughput: 1+ TB/s aggregate
- **L2**
  - All accesses to global memory go through L2, including copies to/from CPU host
- **Global memory**
  - Accessible by all threads as well as host (CPU)
  - High latency (400-800 cycles)
  - Throughput: up to 177 GB/s

# GMEM Operations



- **Two types of loads:**

- Caching

- Default mode
    - Attempts to hit in L1, then L2, then GMEM
    - Load granularity is **128-byte line**

- Non-caching

- Compile with `-Xptxas -dlcm=cg` option to nvcc
    - Attempts to hit in L2, then GMEM
      - Do not hit in L1, invalidate the line if it's in L1 already
    - Load granularity is **32-bytes**

- **Stores:**

- Invalidate L1, write-back for L2

# Load Operation



- **Memory operations are issued per warp (32 threads)**
  - Just like all other instructions
- **Operation:**
  - Threads in a warp provide memory addresses
  - Determine which lines/segments are needed
  - Request the needed lines/segments

# Caching Load



- Warp requests 32 aligned, consecutive 4-byte words
- Addresses fall within 1 cache-line
  - Warp needs 128 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 100%



# Non-caching Load



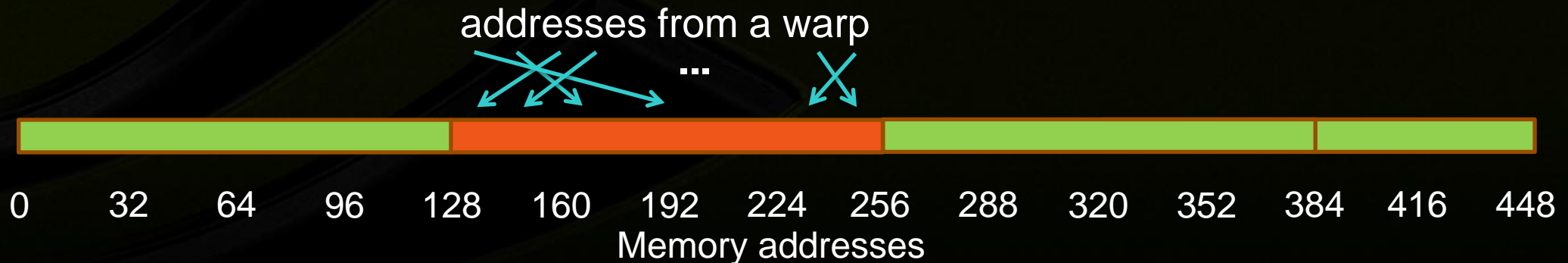
- Warp requests 32 aligned, consecutive 4-byte words
- Addresses fall within 4 segments
  - Warp needs 128 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 100%



# Caching Load



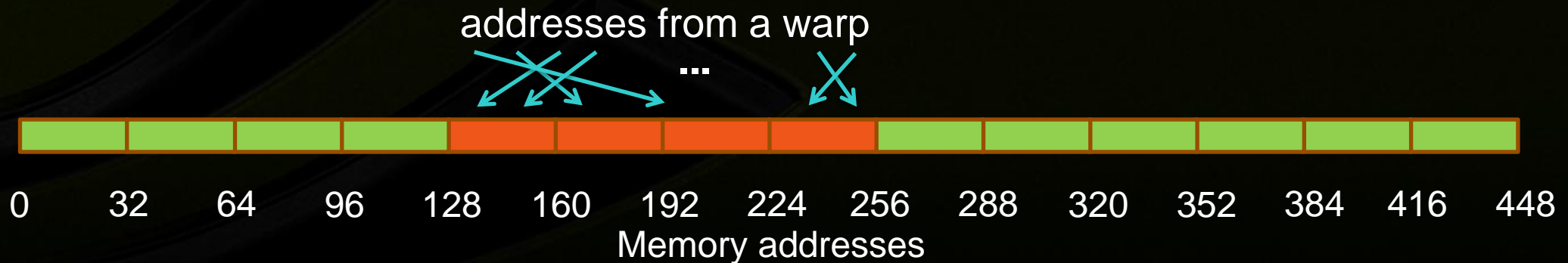
- Warp requests 32 aligned, permuted 4-byte words
- Addresses fall within 1 cache-line
  - Warp needs 128 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 100%



# Non-caching Load



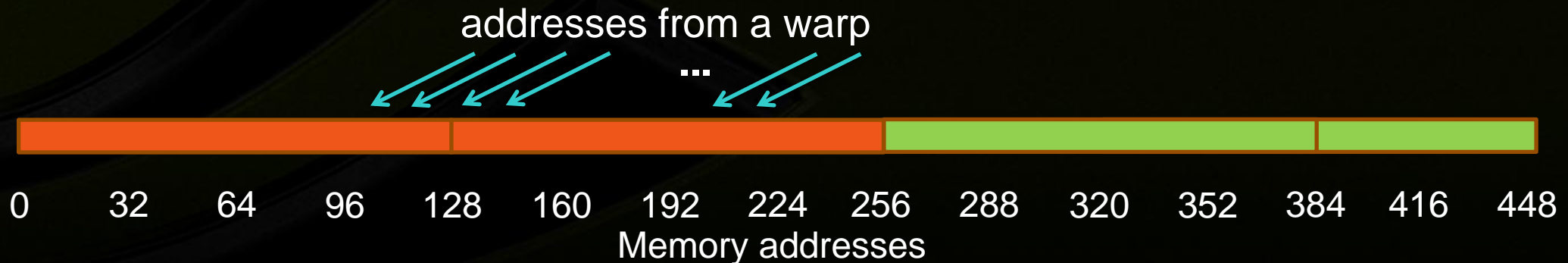
- Warp requests 32 aligned, permuted 4-byte words
- Addresses fall within 4 segments
  - Warp needs 128 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 100%



# Caching Load



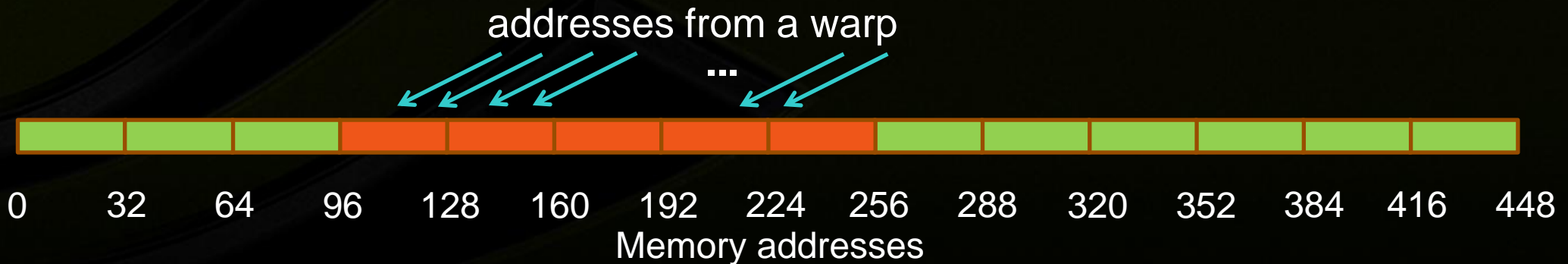
- Warp requests 32 misaligned, consecutive 4-byte words
- Addresses fall within 2 cache-lines
  - Warp needs 128 bytes
  - 256 bytes move across the bus on misses
  - Bus utilization: 50%



# Non-caching Load



- Warp requests 32 misaligned, consecutive 4-byte words
- Addresses fall within at most 5 segments
  - Warp needs 128 bytes
  - 160 bytes move across the bus on misses
  - Bus utilization: at **least 80%**
    - Some misaligned patterns will fall within 4 segments, so 100% utilization



# Caching Load



- All threads in a warp request the same 4-byte word
- Addresses fall within a single cache-line
  - Warp needs 4 bytes
  - 128 bytes move across the bus on a miss
  - Bus utilization: 3.125%



# Non-caching Load



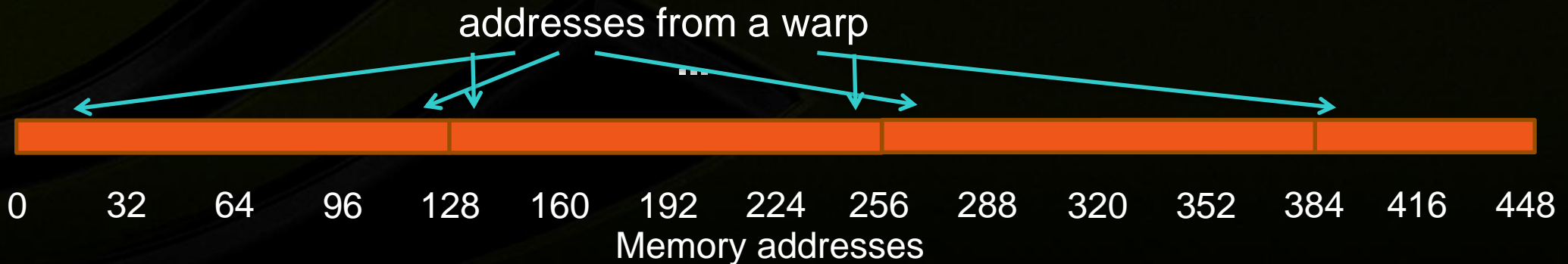
- All threads in a warp request the same 4-byte word
- Addresses fall within a single segment
  - Warp needs 4 bytes
  - 32 bytes move across the bus on a miss
  - Bus utilization: 12.5%



# Caching Load



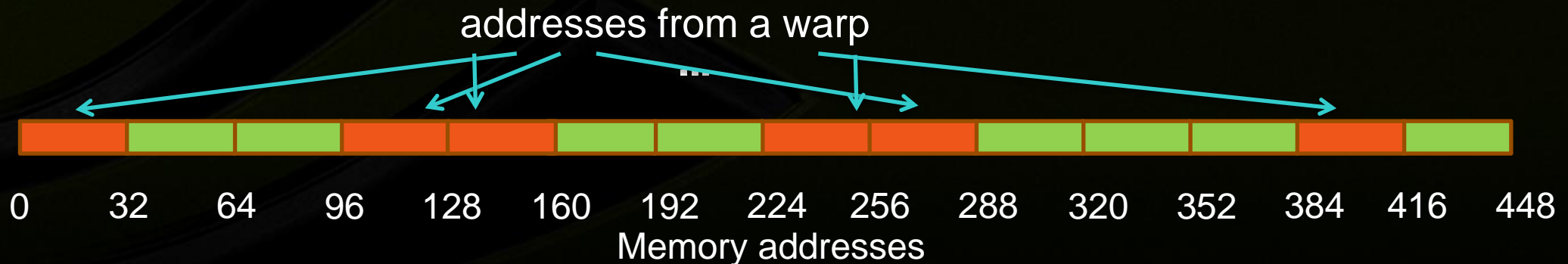
- Warp requests 32 scattered 4-byte words
- Addresses fall within  $N$  cache-lines
  - Warp needs 128 bytes
  - $N*128$  bytes move across the bus on a miss
  - Bus utilization:  $128 / (N*128)$



# Non-caching Load



- Warp requests 32 scattered 4-byte words
- Addresses fall within  $N$  segments
  - Warp needs 128 bytes
  - $N*32$  bytes move across the bus on a miss
  - Bus utilization:  $128 / (N*32)$



# GMEM Optimization Guidelines



- **Strive for perfect coalescing**
  - Align starting address (may require padding)
  - A warp should access within a contiguous region
- **Have enough concurrent accesses to saturate the bus**
  - Process several elements per thread
    - Multiple loads get pipelined
    - Indexing calculations can often be reused
  - Launch enough threads to maximize throughput
    - Latency is hidden by switching threads (warps)
- **Try L1 and caching configurations to see which one works best**
  - Caching vs non-caching loads (compiler option)
  - 16KB vs 48KB L1 (CUDA call)



# Shared Memory

# Shared Memory

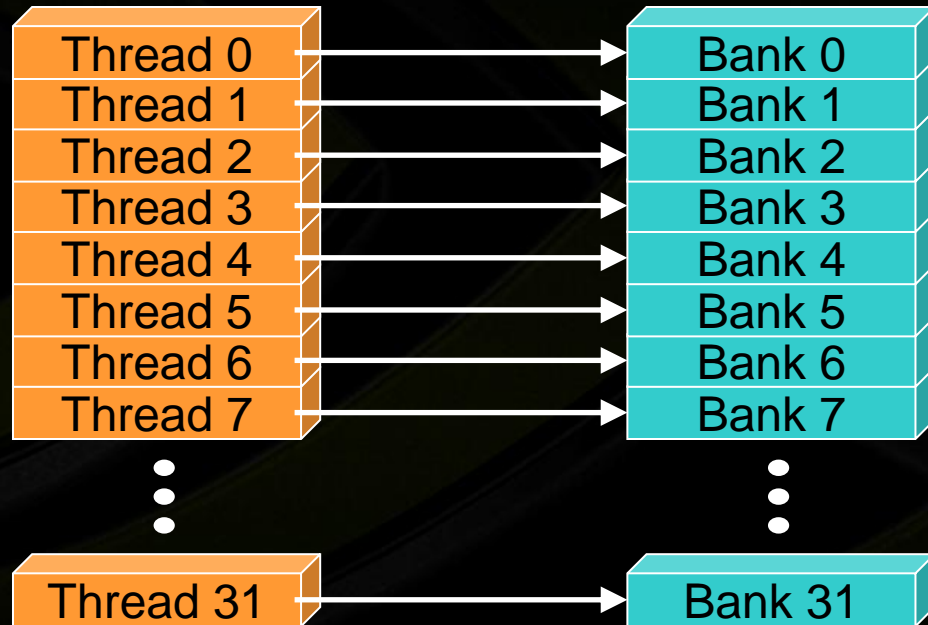


- **Uses:**
  - Inter-thread communication within a block
  - Cache data to reduce redundant global memory accesses
  - Use it to improve global memory access patterns
- **Organization:**
  - 32 banks, 4-byte wide banks
  - Successive 4-byte words belong to different banks
- **Performance:**
  - 4 bytes per bank per 2 clocks per multiprocessor
  - smem accesses are issued per 32 threads (warp)
  - **serialization:** if  $n$  threads of 32 access different 4-byte words in the same bank,  $n$  accesses are executed serially
  - **multicast:**  $n$  threads access the same word in one fetch
    - Could be different bytes within the same word

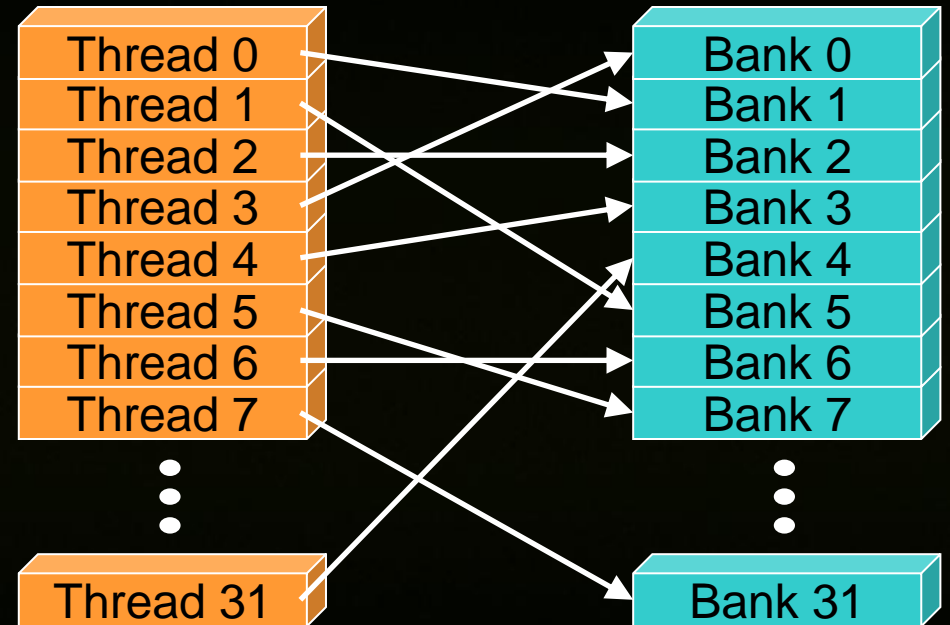
# Bank Addressing Examples



- No Bank Conflicts



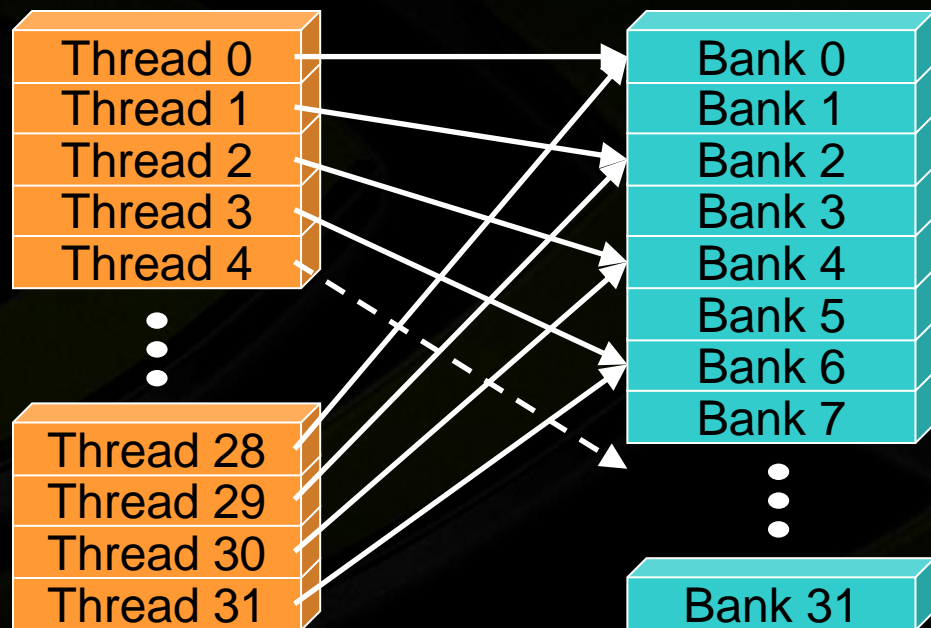
- No Bank Conflicts



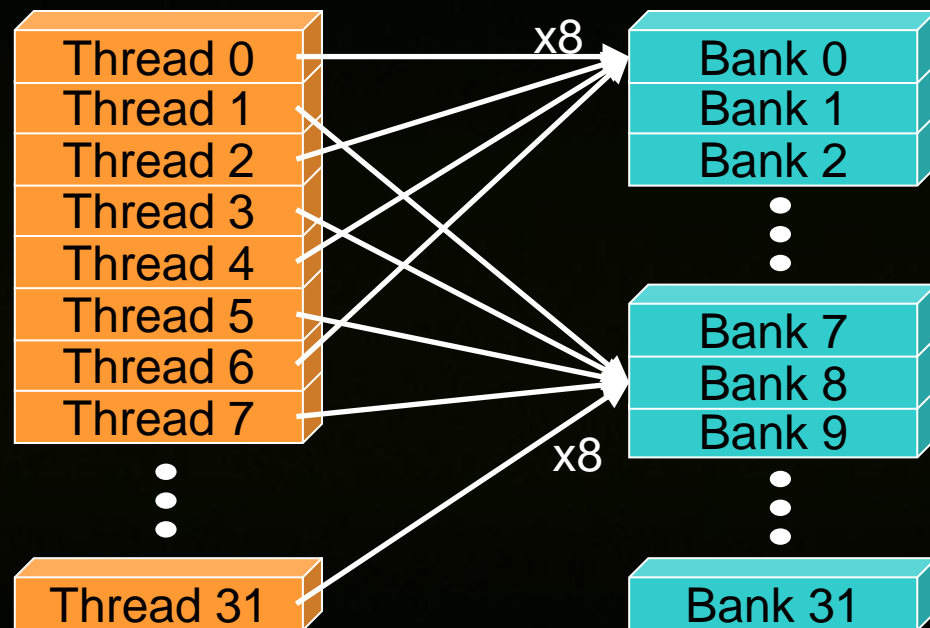
# Bank Addressing Examples



## 2-way Bank Conflicts



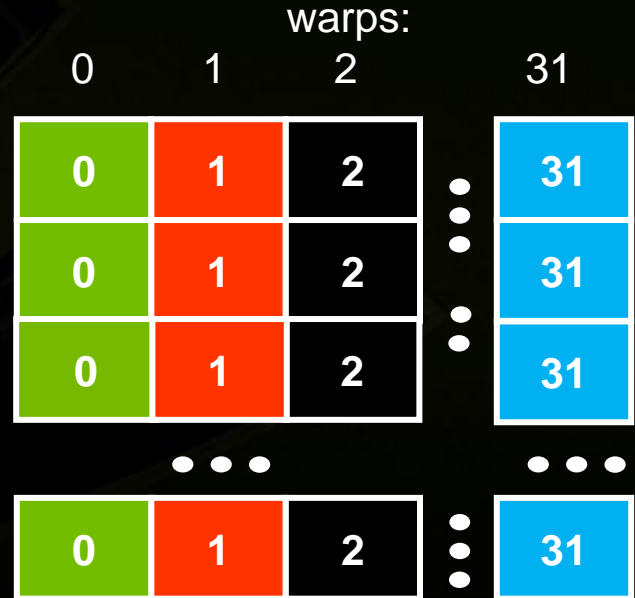
## 8-way Bank Conflicts



# Shared Memory: Avoiding Bank Conflicts

- 32x32 SMEM array
- Warp accesses a column:
  - 32-way bank conflicts (threads in a warp access the same bank)

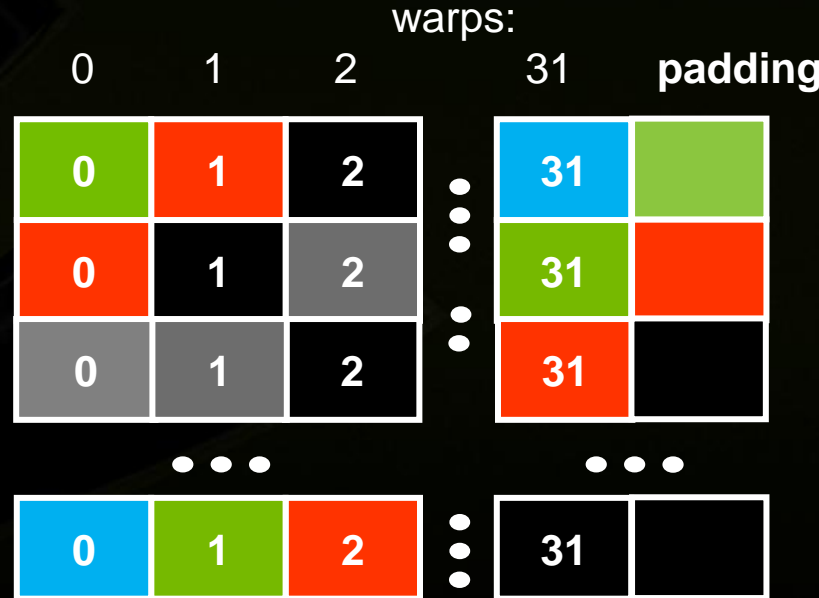
Bank 0  
Bank 1  
...  
Bank 31



# Shared Memory: Avoiding Bank Conflicts

- Add a column for padding:
  - 32x33 SMEM array
- Warp accesses a column:
  - 32 different banks, no bank conflicts

Bank 0  
Bank 1  
...  
Bank 31





# Instruction Throughput / Control Flow

# Runtime Math Library and Intrinsics



- **Two types of runtime math library functions**
  - `__func()`: many map directly to hardware ISA
    - Fast but lower accuracy (see [CUDA Programming Guide](#) for full details)
    - Examples: `__sinf(x)`, `__expf(x)`, `__powf(x, y)`
  - `func()`: compile to multiple instructions
    - Slower but higher accuracy (5 ulp or less)
    - Examples: `sin(x)`, `exp(x)`, `pow(x, y)`
- **A number of additional intrinsics:**
  - `__sincosf()`, `__frcp_rz()`, ...
  - Explicit IEEE rounding modes (rz, rn, ru, rd)

# Control Flow



- **Instructions are issued per 32 threads (warp)**
- **Divergent branches:**
  - Threads within a single warp take different paths
    - `if-else, ...`
  - Different execution paths within a warp are serialized
- **Different warps can execute different code with no impact on performance**
- **Avoid diverging within a warp**
  - Example with divergence:
    - `if (threadIdx.x > 2) {...} else {...}`
    - Branch granularity < warp size
  - Example without divergence:
    - `if (threadIdx.x / WARP_SIZE > 2) {...} else {...}`
    - Branch granularity is a whole multiple of warp size

# Summary



- **Kernel Launch Configuration:**
  - Launch enough threads per SM to hide latency
  - Launch enough threadblocks to load the GPU
- **Global memory:**
  - Maximize throughput (GPU has lots of bandwidth, use it effectively)
- **Use shared memory when applicable (over 1 TB/s bandwidth)**
- **GPU-CPU interaction:**
  - Minimize CPU/GPU idling, maximize PCIe throughput
- **Use analysis/profiling when optimizing:**
  - “Analysis-driven Optimization” GTC 2010 Tutorial



Questions?