Optimizing Square Matrix Transpose

In CUDA

Matrix Transpose

- Square Matrix
- Aij = Bji

- A=

a b c
d e f
g h i

B =

a d g
b e h
c f i

Task 1 – Naïve Transpose

- Each thread reads from (row, col) and writes to (col, row)
- Using indexing macro:
  - #define INDX(row, col, ld) (((row) * (ld)) + (col))
  - ld = leading dimension (width)
- Look for FIXME

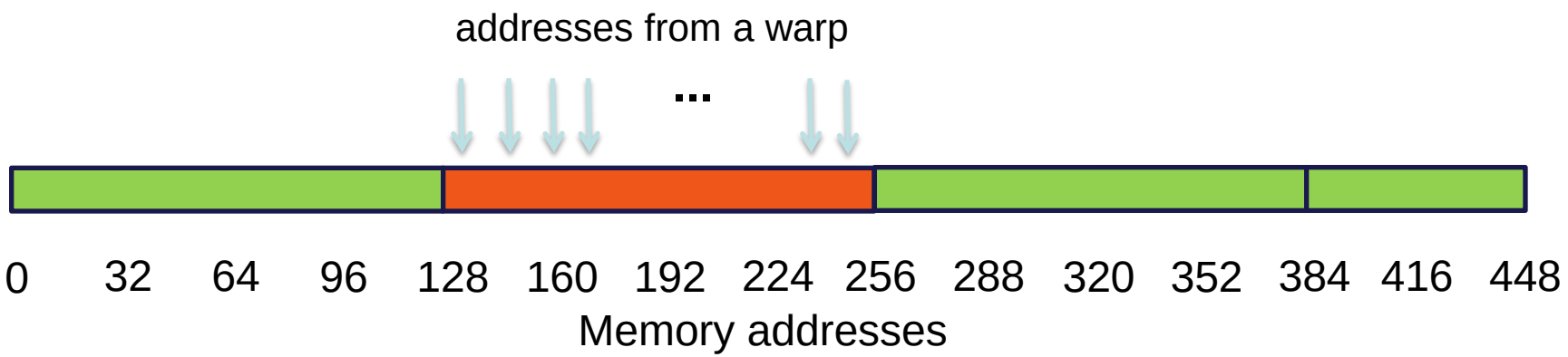
Profiling Task 1

- Generate timeline
- Analysis … examine GPU usage
- Examine Ind. Kernels … select kernel … Perform Kernel Analysis
- Perform Memory Bandwidth Analysis
- Global Load Trans = Global Store Trans ?
- Perform Additional Analysis

Review Memory Coalescing

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%



Task 2 – shared memory

- Eliminate uncoalesced global access by using shared memory to transpose the data
- Each block handles a tile
- We must transpose the data within the tile, *and* we must transpose the tiles.
- Look for FIXME

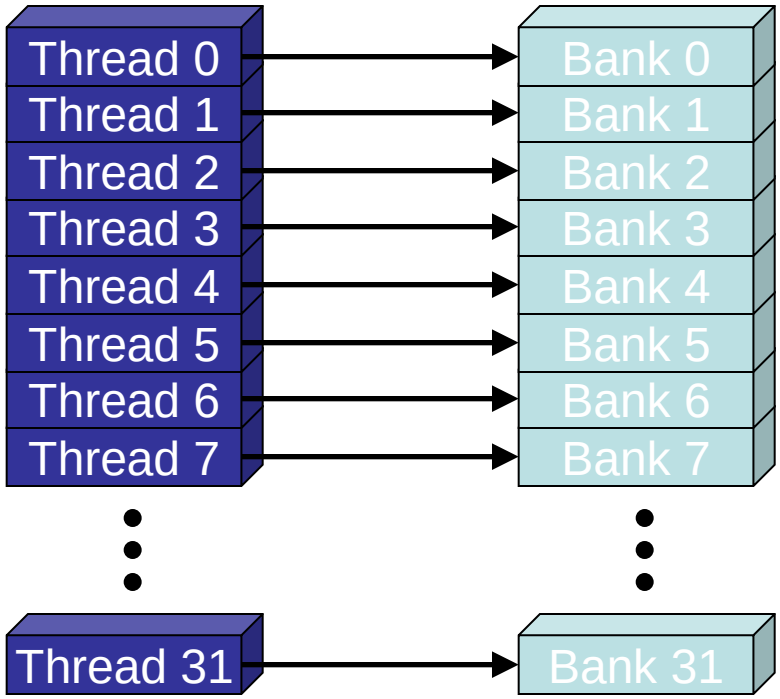
Profiling Task 2

- What happened?
- Memory Bandwidth analysis – are global loads and stores the same?
- How about shared loads and stores ?
- Click on analysis to see code line (requires compiling with -lineinfo)

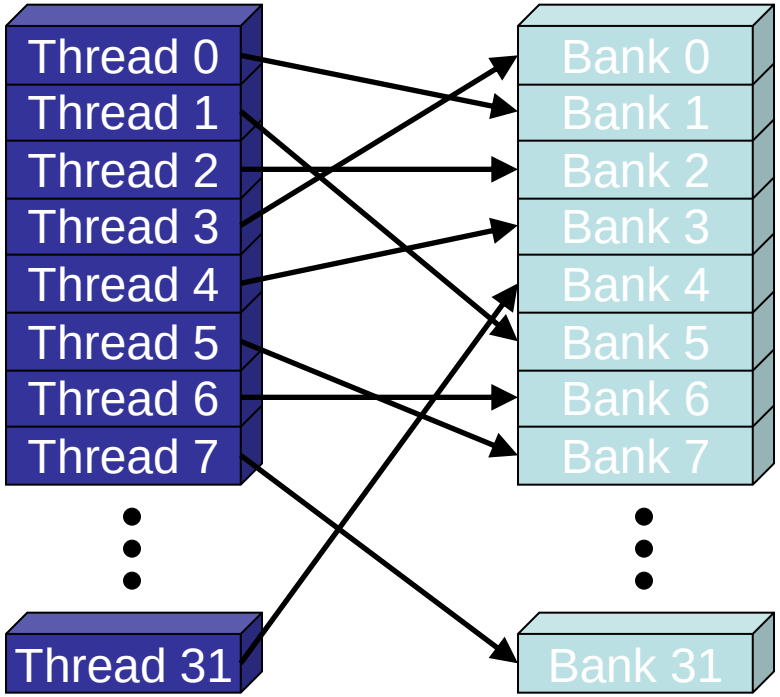
Review Shared Memory and Bank Conflicts

Bank Addressing Examples

- No Bank Conflicts

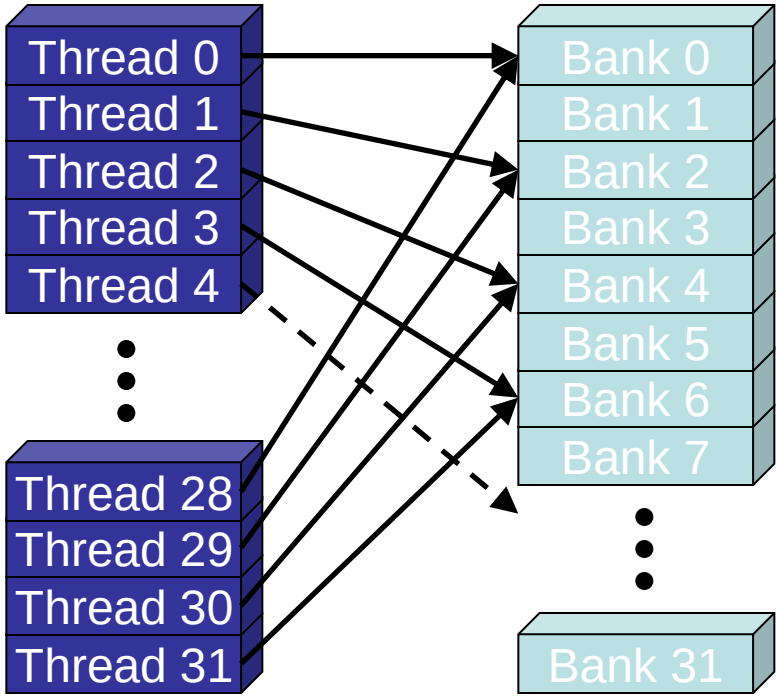


- No Bank Conflicts

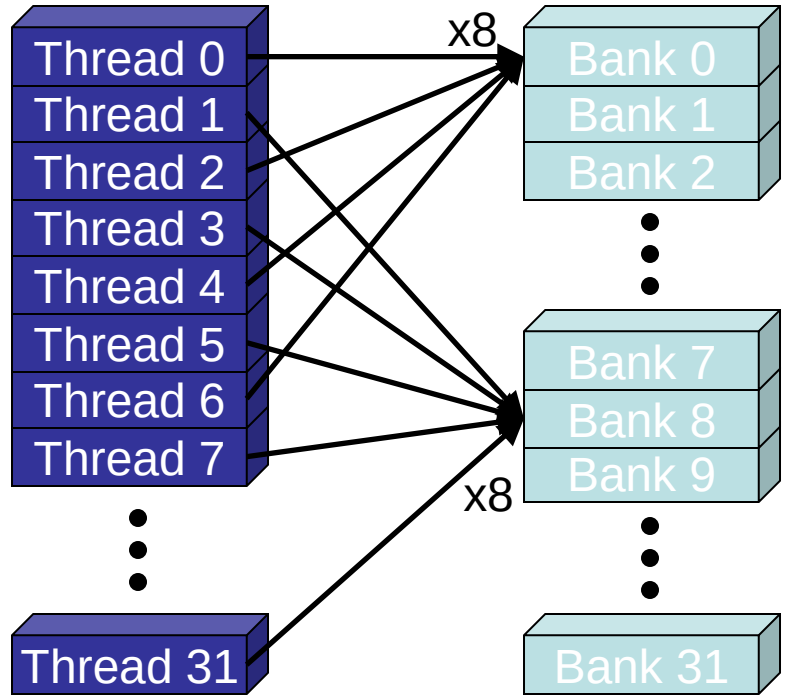


Bank Addressing Examples

- 2-way Bank Conflicts



- 8-way Bank Conflicts

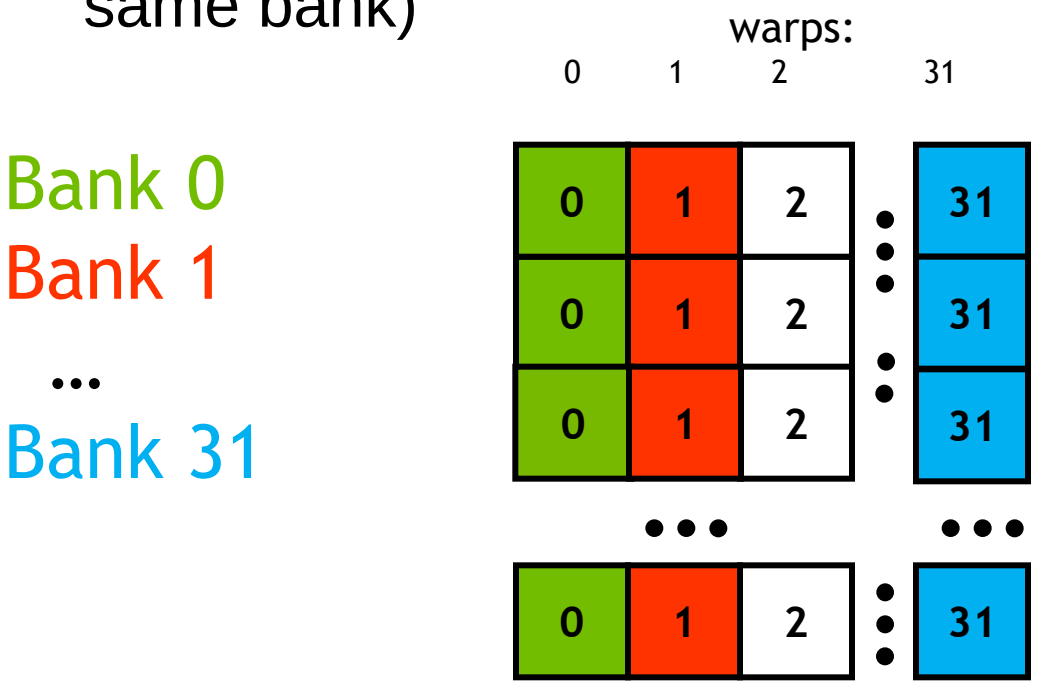


Task 3 – Fix the bank conflicts

- Add extra column to shared memory array

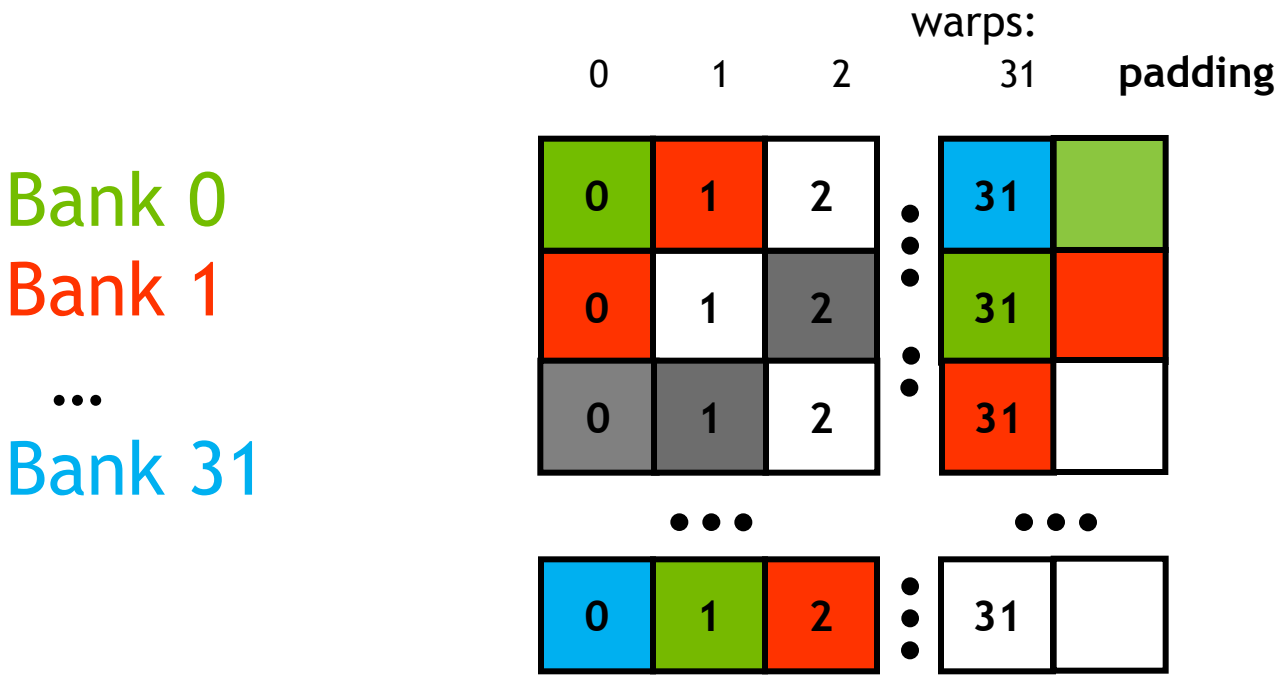
Shared Memory: Avoiding Bank Conflicts

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



Shared Memory: Avoiding Bank Conflicts

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



Is there more???????

- Is the kernel bandwidth limited or latency limited?

Task 4 – address latency

- Increase the work per thread
- Integrate final solution from: http://devblogs.nvidia.com/parallelforall/efficient-matrix-transpose-cuda-cc/