CUDA Tool

Daniel Cohen, Mohamed Zahran
Kernel Performance

- Data Use
- Coalescing
- Bank Conflicts (Global and Shared)
- Branch Divergence
- Occupancy
Data Use

• Measure ratio of global to shared uses.

• Goal is to move away from a 1:1 (load:use) for global data
Coalescing

- Make the *best* use of memory bandwidth.
Bank Conflicts

- Occur in both global and shared memory.

- Global Memory:
  - Matrix Transpose

![Input Data](image1)

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Cartesian</th>
<th>Output Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td></td>
<td>0 64 128 129 129 130</td>
</tr>
<tr>
<td>64 65 66 67 68 69</td>
<td></td>
<td>2 66 130</td>
</tr>
<tr>
<td>128 129 130</td>
<td></td>
<td>3 67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 68</td>
</tr>
</tbody>
</table>

(a) Transpose Coalesced

![Diagonal](image2)

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Output Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 5</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>64 65 128 129 130</td>
<td>64 1 128 65 2</td>
</tr>
<tr>
<td>1 65 129</td>
<td>65 2 129</td>
</tr>
<tr>
<td>2 66 130</td>
<td>66 3 130</td>
</tr>
<tr>
<td>3 67</td>
<td>67 130</td>
</tr>
<tr>
<td>4 68</td>
<td>68 4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

(b) Transpose Diagonal
Bank Conflicts

• Shared Memory:
  - Stride length and array dimensions
Occupancy

• Measures:

  - Percentage of SMs used in entire GPU.

  - Efficiency of SMs. Calculates warps:max_warps for each SM based on shared memory, register use, and block dimensions.
Setup

• Needs gpgpu-sim:

• Installer for virtual machine available.
  - Ubuntu 10.04

• [https://www.dropbox.com/sh/5y5gqd2vt2sm659/AADbTOahU0oY-IDeeZSUJ4pka?dl=0](https://www.dropbox.com/sh/5y5gqd2vt2sm659/AADbTOahU0oY-IDeeZSUJ4pka?dl=0)
How to Use

- user$ python gpu_parse.py [source code] [executable]
- Returns advice based on deficiencies in kernel.
  - Order of output based on bottlenecks
Sample Kernel

```c
#include <stdio.h>
#include <cuda.h>
#define TILE_DIM 32
#define BLOCK_ROWS 8

// Number of repetitions used for timing.
#define NUM_REPS 1

__global__ void transposeNaive(float *odata, float *idata, int width, int height, int nreps)
{
    int xIndex = blockIdx.x*TILE_DIM + threadIdx.x;
    int yIndex = blockIdx.y*TILE_DIM + threadIdx.y;
    int index_in = xIndex + width * yIndex;
    int index_out = yIndex + height * xIndex;
    for (int r=0; r<nreps; r++) {
        for (int i=0; i<TILE_DIM; i+=BLOCK_ROWS) {
            odata[index_out+i] = idata[index_in+i*width];
        }
    }
}

int main(int argc, char** argv)
{
    int i;
    int nreps = NUM_REPS;
    const int size_x = 512, size_y = 512;
    dim3 grid(size_x/TILE_DIM, size_y/TILE_DIM),
    threads(TILE_DIM, BLOCK_ROWS);
    const int mem_size = sizeof(float) * size_x*size_y;
    float *h_idata = (float*) malloc(mem_size);
    float *h_odata = (float*) malloc(mem_size);
    for(i = 0; i < size_x * size_y; i++) {
        h_idata[i] = (float) i;
    }
    float *d_idata, *d_odata;
    cudaMalloc((void**) &d_idata, mem_size);
    cudaMalloc((void**) &d_odata, mem_size);
    cudaMemcpy(d_idata, h_idata, mem_size, cudaMemcpyHostToDevice);
    transposeNaive<<<grid,threads>>>(d_idata, d_odata, size_y, size_x, nreps);
    cudaMemcpy(h_odata, d_odata, mem_size, cudaMemcpyDeviceToHost);
}
```
Output
setup environment succeeded

What type of compute capability? 1.3  Enter compute capability when prompted

Kernel information below, alternate thread/block configurations are shown at the bottom:

Uncoalesced accesses occur in 3.0 or 7.0 global memory accesses 3/7 global memory accesses are uncoalesced

The following lines of code contain the source code of these accesses (check for variables stored in global memory):

```c
// 16  int index_in = xIndex + width * yIndex;
// 17  int index_out = yIndex + height * xIndex;
// 18  for (int r=0; r < nreps; r++) {
// 19    for (int i=0; i<TILE_DIM; i+=BLOCK_ROWS) {
// 20      odata[index_out+i] = idata[index_in+i*width];
// 16  int index_in = xIndex + width * yIndex;
// 17  int index_out = yIndex + height * xIndex;
// 18  for (int r=0; r < nreps; r++) {
// 19    for (int i=0; i<TILE_DIM; i+=BLOCK_ROWS) {
// 20      odata[index_out+i] = idata[index_in+i*width];
// 16  int index_in = xIndex + width * yIndex;
// 17  int index_out = yIndex + height * xIndex;
// 18  for (int r=0; r < nreps; r++) {
// 19    for (int i=0; i<TILE_DIM; i+=BLOCK_ROWS) {
// 20      odata[index_out+i] = idata[index_in+i*width];
```

These 3 accesses come from the same section of code (16-20)

Data Use ratio is 1.0  Data use is 1:1, as there is no use of shared memory

Bank conflicts are present in your code. There are 0 conflicts out of 0 shared memory accesses. No shared memory accesses so 0 bank conflicts.

The following lines contain the source of the bank conflicts:

The access patterns for your global memory results in excessive simultaneous accesses to the same bank. Reorganization of data layout needed.

Listed below are possible configurations of shared memory, block dimensions, and register use. The format is [current, x/100]: Option

Kernel call #0 has an SM occupancy score of 100.0

Shared memory usage:
current: [0, 1.0]
other options: [[512, 1.0], [1024, 1.0], [1536, 1.0], [2048, 1.0], [2560, 1.0], [3072, 1.0], [3584, 1.0], [4096, 1.0], [4608, 1.0], [5120, 1.0], [9216, 1.0], [9728, 1.0], [10240, 1.0], [10752, 1.0], [11264, 1.0], [11776, 1.0], [12288, 1.0]]

Block dimensions:
current: 256
other options: [[128, 1.0], [256, 1.0], [512, 1.0], [1024, 1.0]]
Contact

• dsc381@nyu.edu

• Please email me if you have any problems:
  - Setting up
  - Understanding the output
  - Errors

• Or if it helps you