Hardware/Software Co-Design

Parallel Scan & Review

Miaoqing Huang
University of Arkansas
Fall 2011
Outline

1. Parallel Prefix Sum (Scan)
2. Review
Outline

1. Parallel Prefix Sum (Scan)

2. Review
What is all-prefix-sums?

Definition
The all-prefix-sums operation takes a binary associative operator $\oplus$, and an array of $n$ elements 
\[ [a_0, a_1, \ldots, a_{n-1}] \]
and returns the array
\[ [a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus a_2 \oplus \cdots \oplus a_{n-2} \oplus a_{n-1})] \].

Example
If $\oplus$ is addition, then the all-prefix-sums operation on the array 
\[ [3, 1, 7, 0, 4, 1, 6, 3] \]
would return 
\[ [3, 4, 11, 11, 15, 16, 22, 25] \].

Pseudo code
```plaintext
out[0] = in[0];
for (j=1; j<n; j++) {
    out[j] = out[j-1] operator in[j];
}
```
Parallel Scan in CUDA

A naive solution

\[
\text{for } d = 1 \text{ to } \log(n) \text{ do} \\
\quad \text{forall } k \text{ in parallel do} \\
\quad \quad \text{if } (k \geq 2^d) \text{ then } x[k] = x[k-2^{(d-1)}] + x[k]
\]

- \( d = 0 \)
  - \( x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \)

- \( d = 1 \)
  - \( \Sigma(x_0..x_0) \quad \Sigma(x_0..x_1) \quad \Sigma(x_1..x_2) \quad \Sigma(x_2..x_3) \quad \Sigma(x_3..x_4) \quad \Sigma(x_4..x_5) \quad \Sigma(x_5..x_6) \quad \Sigma(x_6..x_7) \)

- \( d = 2 \)
  - \( \Sigma(x_0..x_0) \quad \Sigma(x_0..x_1) \quad \Sigma(x_0..x_2) \quad \Sigma(x_0..x_3) \quad \Sigma(x_1..x_4) \quad \Sigma(x_2..x_5) \quad \Sigma(x_3..x_6) \quad \Sigma(x_4..x_7) \)

- \( d = 3 \)
  - \( \Sigma(x_0..x_0) \quad \Sigma(x_0..x_1) \quad \Sigma(x_0..x_2) \quad \Sigma(x_0..x_3) \quad \Sigma(x_0..x_4) \quad \Sigma(x_0..x_5) \quad \Sigma(x_0..x_6) \quad \Sigma(x_0..x_7) \)
Outline

1. Parallel Prefix Sum (Scan)
2. Review
The example

- Compute 1,024 dot products, each of which is calculated from a pair of 256-element vectors

```
1 #define VECTOR_N 1024
2 #define ELEMENT_N 256
3 const int DATA_N = VECTOR_N * ELEMENT_N;
4 const int DATA_SZ = DATA_N * sizeof(float);
5 const int RESULT_SZ = VECTOR_N * sizeof(float);
......
6 float *d_A, *d_B, *d_C;
......
7 cudaMalloc((void **)d_A, DATA_SZ);
8 cudaMalloc((void **)d_B, DATA_SZ);
9 cudaMalloc((void **)d_C, RESULT_SZ);
......
10 scalarProd<<<VECTOR_N, ELEMENT_N>>>(d_C, d_A, d_B, ELEMENT_N);
```
The code (cont.)

```c
__global__ void scalarPar(float * d_C, float *d_A, float *d_B, int ElementN) {
    __shared__ float accumResult[ELEMENT_N];
    //Current vectors bases
    float *A = d_A + ElementN * blockIdx.x;
    float *B = d_B + ElementN * blockIdx.x;
    int tx = threadIdx.x;


    for (int stride = ElementN / 2; stride >0; stride >>= 1) {
        __syncthreads();
        if (tx < stride)
            accumResult[tx] += accumResult[stride + tx];
    }

d_C[blockIdx.x] = accumResult[0];
```
How many threads are there in total?

$$\text{VECTOR}_N \times \text{ELEMENT}_N$$

How many threads are there in a warp?

32

How many threads are there in a block?

\(\text{ELEMENT}_N\)

How many global memory loads and stores are done for each thread?

Every thread performs two global memory loads on line 21 (one each from A and B). Every thread also writes one value on line 29. Each thread performs 3 global memory accesses in total.
How many threads are there in total?

VECTOR_N * ELEMENT_N
1. How many threads are there in total?
   - $\text{VECTOR}_N \times \text{ELEMENT}_N$

2. How many threads are there in a warp?
Problems

1. How many threads are there in total?
   - $\text{VECTOR}_N \times \text{ELEMENT}_N$

2. How many threads are there in a warp?
   - 32
Problems

1. How many threads are there in total?
   - VECTOR_N * ELEMENT_N

2. How many threads are there in a warp?
   - 32

3. How many threads are there in a block?
Problems

1. How many threads are there in total?
   - VECTOR_N * ELEMENT_N
2. How many threads are there in a warp?
   - 32
3. How many threads are there in a block?
   - ELEMENT_N
Problems

1. How many threads are there in total?
   - \( \text{VECTOR}_N \times \text{ELEMENT}_N \)

2. How many threads are there in a warp?
   - 32

3. How many threads are there in a block?
   - \( \text{ELEMENT}_N \)

4. How many global memory loads and stores are done for each thread?
   - Every thread performs two global memory loads on line 21 (one each from A and B). Every thread also writes one value on line 29. Each thread performs 3 global memory accesses in total.
**Problems**

1. How many threads are there in total?
   - \( \text{VECTOR}_N \times \text{ELEMENT}_N \)

2. How many threads are there in a warp?
   - 32

3. How many threads are there in a block?
   - \( \text{ELEMENT}_N \)

4. How many global memory loads and stores are done for each thread?
   - Every thread performs two global memory loads on line 21 (one each from A and B). Every thread also writes one value on line 29. Each thread performs 3 global memory accesses in total.
How many accesses to shared memory are done for each block? (Note: do not consider memory coalescing during the calculation)

On line 21, 256 writes (one for each thread) to shared memory occur. When a thread executes line 27, it performs two shared memory reads, adds the two values together, and the performs a shared memory write to store the result. Given the loop structure, on the first iteration, 128 threads will execute the statement, then 64 on the following statement, and so on. Finally, each thread reads element 0 on line 29, accounting for another 256 accesses. Therefore, the total number of accesses is 256 + (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1)*3 + 256 = 256 + 255*3 + 256 = 1277 accesses.

List the source code lines, if any, that cause shared memory bank conflicts.

This kernel will not cause any shared memory bank conflicts. Every access uses an index of base + tx, which will never generate a conflict if the base is uniform among threads, as is the case here.
How many accesses to shared memory are done for each block? (Note: do not consider memory coalescing during the calculation)

On line 21, 256 writes (one for each thread) to shared memory occur. When a thread executes line 27, it performs two shared memory reads, adds the two values together, and the performs a shared memory write to store the result. Given the loop structure, on the first iteration, 128 threads will execute the statement, then 64 on the following statement, and so on. Finally, each thread reads element 0 on line 29, accounting for another 256 accesses. Therefore, the total number of accesses is $256 + (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1) \times 3 + 256 = 256 + 255 \times 3 + 256 = 1277$ accesses.
How many accesses to shared memory are done for each block? (Note: do not consider memory coalescing during the calculation)

- On line 21, 256 writes (one for each thread) to shared memory occur. When a thread executes line 27, it performs two shared memory reads, adds the two values together, and then performs a shared memory write to store the result. Given the loop structure, on the first iteration, 128 threads will execute the statement, then 64 on the following statement, and so on. Finally, each thread reads element 0 on line 29, accounting for another 256 accesses. Therefore, the total number of accesses is $256 + (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1) \times 3 + 256 = 256 + 255 \times 3 + 256 = 1277$ accesses.

List the source code lines, if any, that cause shared memory bank conflicts.

This kernel will not cause any shared memory bank conflicts. Every access uses an index of base + tx, which will never generate a conflict if the base is uniform among threads, as is the case here.
How many accesses to shared memory are done for each block? (Note: do not consider memory coalescing during the calculation)

- On line 21, 256 writes (one for each thread) to shared memory occur. When a thread executes line 27, it performs two shared memory reads, adds the two values together, and the performs a shared memory write to store the result. Given the loop structure, on the first iteration, 128 threads will execute the statement, then 64 on the following statement, and so on. Finally, each thread reads element 0 on line 29, accounting for another 256 accesses. Therefore, the total number of accesses is 256 + (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1)*3 + 256 = 256 + 255*3 + 256 = 1277 accesses.

List the source code lines, if any, that cause shared memory bank conflicts.

- This kernel will not cause any shared memory bank conflicts. Every access uses an index of base + tx, which will never generate a conflict if the base is uniform among threads, as is the case here.
Identify an opportunity to significantly reduce the bandwidth requirement on the global memory. How would you achieve this? How many accesses can you eliminate?
Identify an opportunity to significantly reduce the bandwidth requirement on the global memory. How would you achieve this? How many accesses can you eliminate?

- Without any conditional statement guarding it, every thread will execute line 29. This means that every thread in the block will be redundantly writing the same final result to the same final location. These writes get serialized in the memory system, and take significantly more bandwidth than is necessary. This can be avoided by selecting one thread only to perform the copy from shared to global memory. An example code addition could look like:

```c
if (tx == 0)
    d_C[blockIdx.x] = accumResult[0];
```