Parallel Computation Patterns

A Work-inefficient Scan Kernel

Wen-mei Hwu - University of Illinois at Urbana-Champaign
Objective

• To learn to write and analyze a high-performance scan kernel
  • Interleaved reduction trees
  • Thread index to data mapping
  • Barrier Synchronization
  • Work efficiency analysis
A Better Parallel Scan Algorithm

1. Read input from device global memory to shared memory

2. Iterate log(n) times; stride from 1 to n-1:
   - double stride each iteration
   - Active threads stride to n-1 (n-stride threads)
   - Thread $j$ adds elements $j$ and $j$-stride from shared memory and writes result into element $j$ in shared memory
   - Requires barrier synchronization, once before read and once before write
A Better Parallel Scan Algorithm

1. Read input from device to shared memory
2. Iterate $\log(n)$ times; stride from 1 to $n-1$:
double stride each iteration.

<table>
<thead>
<tr>
<th>XY</th>
<th>3</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>4</th>
<th>1</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRIDE 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XY</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>STRIDE 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XY</td>
<td>3</td>
<td>4</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>

ITERATION = 2
STRIDE = 2
A Better Parallel Scan Algorithm

1. Read input from device to shared memory
2. Iterate log(n) times; stride from 1 to n-1:
   double stride each iteration
3. Write output from shared memory to device memory

<table>
<thead>
<tr>
<th>XY</th>
<th>3</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>4</th>
<th>1</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
</table>

**STRIDE 1**

<table>
<thead>
<tr>
<th>XY</th>
<th>3</th>
<th>4</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
</table>

**STRIDE 2**

<table>
<thead>
<tr>
<th>XY</th>
<th>3</th>
<th>4</th>
<th>11</th>
<th>11</th>
<th>12</th>
<th>12</th>
<th>11</th>
<th>14</th>
</tr>
</thead>
</table>

**STRIDE 4**

<table>
<thead>
<tr>
<th>XY</th>
<th>3</th>
<th>4</th>
<th>11</th>
<th>11</th>
<th>15</th>
<th>16</th>
<th>22</th>
<th>25</th>
</tr>
</thead>
</table>

**ITERATION = 3**

**STRIDE = 4**
Handling Dependencies

• During every iteration, each thread can overwrite the input of another thread
  • Barrier synchronization to ensure all inputs have been properly generated
  • All threads secure input operand that can be overwritten by another thread
  • Barrier synchronization to ensure that all threads have secured their inputs
  • All threads perform Addition and write output
A Work-Inefficient Scan Kernel

1. `__global__ void
   work_inefficient_scan_kernel(float *X, float *Y, int InputSize) {
2.       __shared__ float XY[SECTION_SIZE];
3.       int i = blockIdx.x*blockDim.x + threadIdx.x;
4.       if (i < InputSize) {XY[threadIdx.x] = X[i];}

   // the code below performs iterative
   // scan on XY
5.       for (unsigned int stride = 1; stride <= threadIdx.x; stride *= 2) {
6.           __syncthreads();
7.           float inl = XY[threadIdx.x-stride];
8.           __syncthreads();
9.           XY[threadIdx.x] += inl;
10.      }
11.      __syncthreads();
}
Work Efficiency Considerations

• This Scan executes log(n) parallel iterations
  – The steps do (n-1), (n-2), (n-4),...(n- n/2) adds each
  – Total adds: $n \times \log(n) - (n-1) \rightarrow O(n \times \log(n))$ work

• This scan algorithm is not work efficient
  – Sequential scan algorithm does $n$ adds
  – A factor of $\log(n)$ can hurt: 10x for 1024 elements!

• A parallel algorithm can be slower than a sequential one when execution resources are saturated from low work efficiency
To learn more, read Sections 9.2-9.3