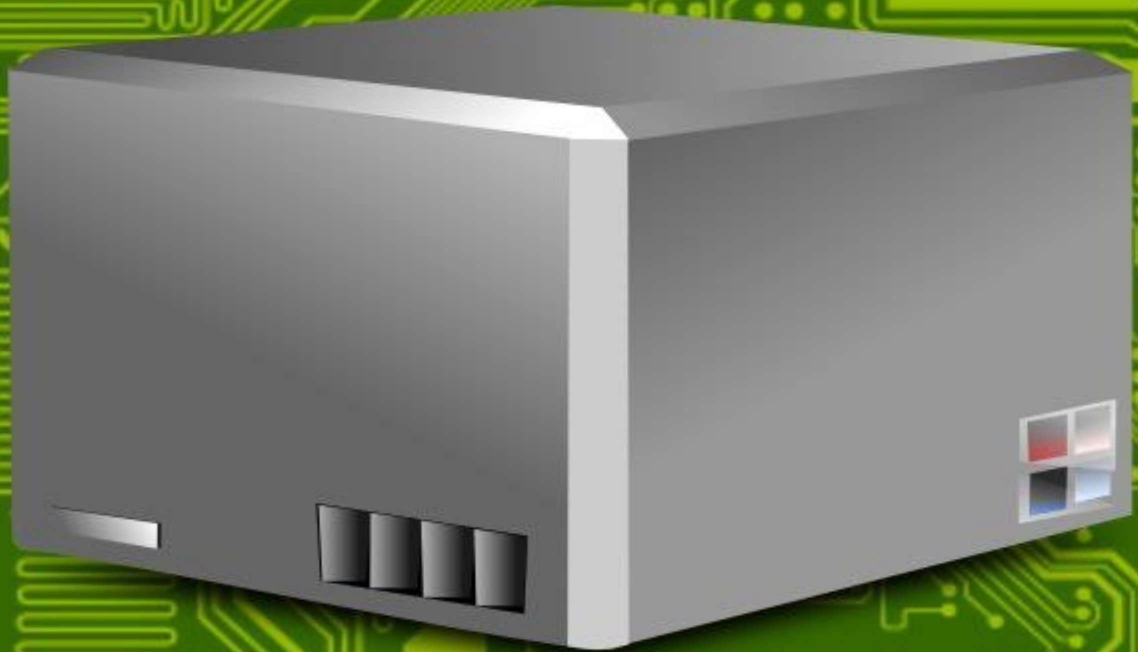
GPU Computing



规约算法

Hui Liu
Email: hui.sc.liu@gmail.com

Reduction #6: Complete Unrolling

- If we knew the number of iterations at compile time, we could completely unroll the reduction
  - Block size is limited to 512
  - We can restrict our attention to powers-of-two block sizes
- We can easily unroll for a fixed block size
  - But we need to be generic
  - How can we unroll for block sizes we don’t know at complile time?
- C++ Templates
  - CUDA supports C++ templates on device and host functions

Unrolling Using Templates

- Specify block size as a function template parameter:

```
template <unsigned int blockSize>
   __global__ void reduce5(int *g_data, int *g_odata)
```

Reduction #6: Completely Unrolled

```
if (blockSize >= 512) {
        if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads();
}
if (blockSize >= 256) {
        if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads();
}
if (blockSize >= 128) {
        if (tid < 64) { sdata[tid] += sdata[tid + 64]; } __syncthreads();
}
if (tid < 32) {
        if (blockSize >= 64) sdata[tid] += sdata[tid + 32];
        if (blockSize >= 32) sdata[tid] += sdata[tid + 16];
        if (blockSize >= 16) sdata[tid] += sdata[tid + 8];
        if (blockSize >= 8) sdata[tid] += sdata[tid + 4];
        if (blockSize >= 4) sdata[tid] += sdata[tid + 2];
        if (blockSize >= 2) sdata[tid] += sdata[tid + 1];
}
```

- Note: all code in **RED** will be evaluated at compile time.
- Results in a very efficient inner loop

What if block size is not known at compile time?

- There are “only” 10 possibilities:

```
switch (threads)
{
case 512:
    reduce5<512><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 256:
    reduce5<256><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 128:
    reduce5<128><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 64:
    reduce5< 64><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 32:
    reduce5< 32><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 16:
    reduce5< 16><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 8:
    reduce5< 8><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 4:
    reduce5< 4><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 2:
    reduce5< 2><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
case 1:
    reduce5< 1><<< dimGrid, dimBlock, smemSize >>>(d_idata, d_odata); break;
}
```

Performance for 4M element reduction

	Time (2^{22} ints)	Bandwidth	Step Speedup	Cumulative Speedup
Kernel 1: interleaved addressing with divergent branching	**8.054 ms**	**2.083 GB/s**		
Kernel 2: interleaved addressing non-divergent branching	**3.456 ms**	**4.854 GB/s**	**2.33x**	**2.33x**
Kernel 3: sequential addressing	**1.722 ms**	**9.741 GB/s**	**2.01x**	**4.68x**
Kernel 4: first step during global load	**0.965 ms**	**17.377 GB/s**	**1.78x**	**8.34x**
Kernel 5: Unroll last warp	**0.536 ms**	**31.289 GB/s**	**1.8x**	**15.01x**
Kernel 6: Complete Unroll	**0.381 ms**	**43.996 GB/s**	**1.41x**	**21.16x**

THANK YOU