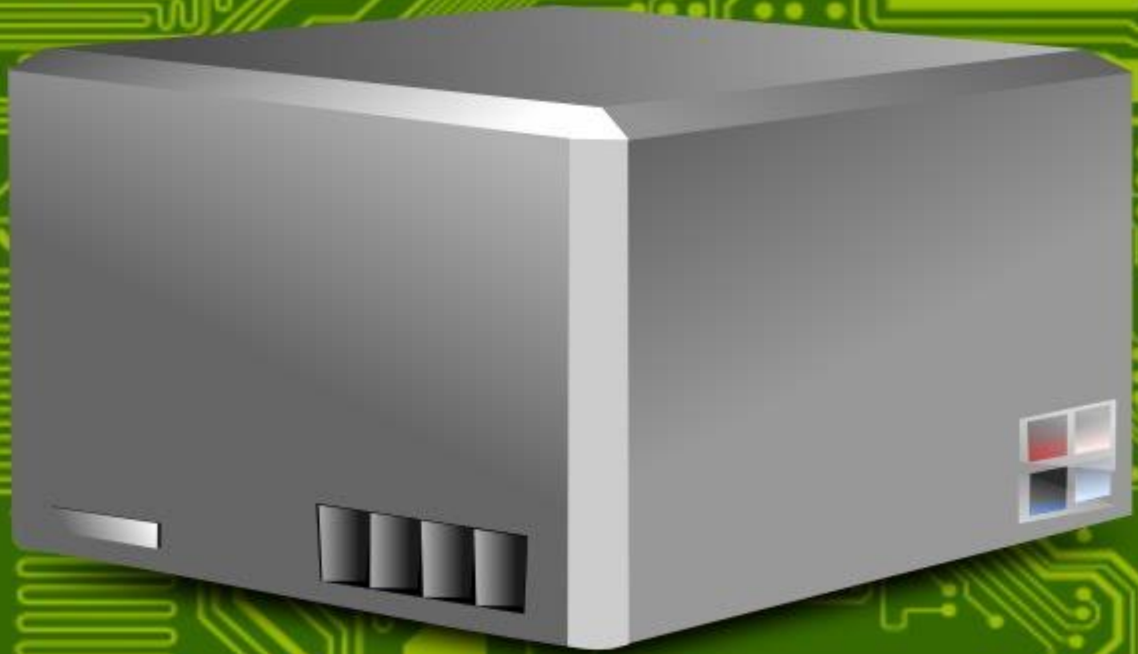
GPU Computing



规约算法: 循环展开注解

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

Reduction #5: Unrolling the last 6 iterations

```
// do reduction in shared mem
for (unsigned int s = blockDim.x/2; s > 32; s /= 2) {

        if (tid < s) {
                sdata[tid] += sdata[tid + s];
        }
        __syncthreads();
}
```

```
if (tid <32)
{
        sdata[tid] += sdata[tid + 32];
        sdata[tid] += sdata[tid + 16];
        sdata[tid] += sdata[tid + 8];
        sdata[tid] += sdata[tid + 4];
        sdata[tid] += sdata[tid + 2];
        sdata[tid] += sdata[tid + 1];
}
```

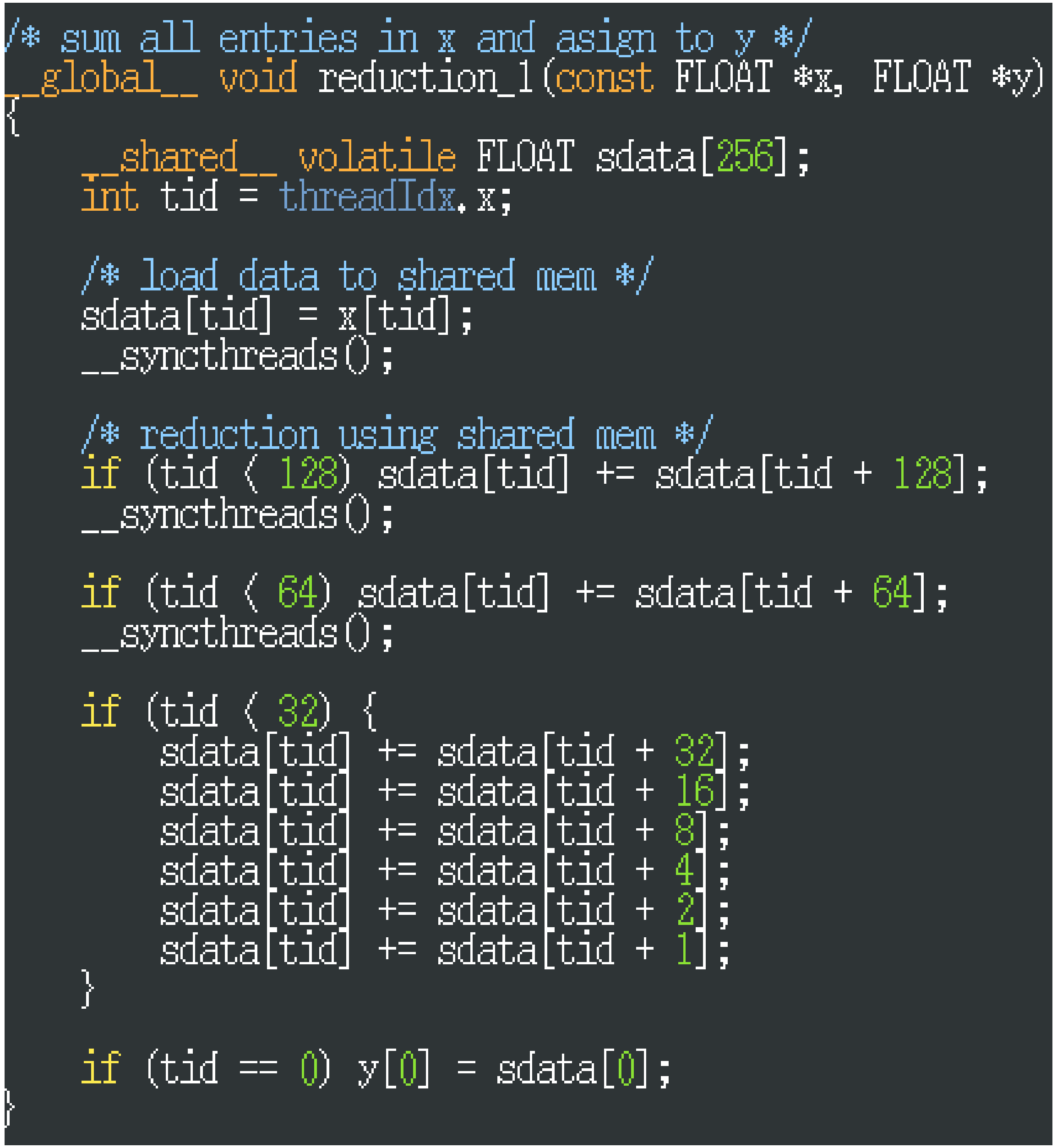
- This saves work in all warps not just the last one
  - Without unrolling all warps execute the for loop and if statement

Reduction #6: Completely Unrolled

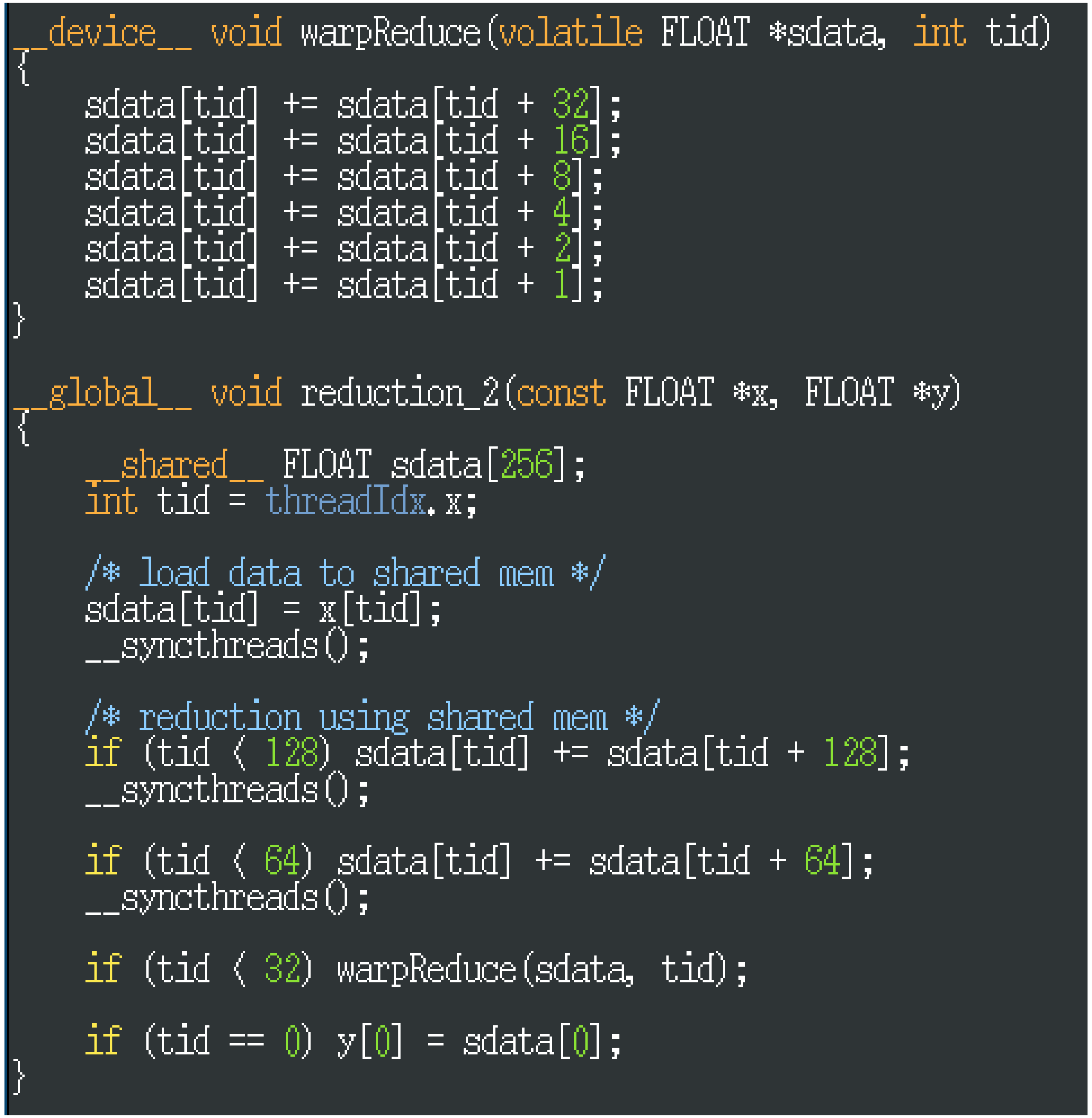
```
if (blockSize >= 512) {
        if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads();
}
if (blockSize >= 256) {
        if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads();
}
if (blockSize >= 128) {
        if (tid < 64) { sdata[tid] += sdata[tid + 64]; } __syncthreads();
}
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

Volatile 修饰符



```
/* sum all entries in x and asign to y */
__global__ void reduction_1(const FLOAT *x, FLOAT *y)
{
    __shared__ volatile FLOAT sdata[256];
    int tid = threadIdx.x;

    /* load data to shared mem */
    sdata[tid] = x[tid];
    __syncthreads();

    /* reduction using shared mem */
    if (tid < 128) sdata[tid] += sdata[tid + 128];
    __syncthreads();

    if (tid < 64) sdata[tid] += sdata[tid + 64];
    __syncthreads();

    if (tid < 32) {
        sdata[tid] += sdata[tid + 32];
        sdata[tid] += sdata[tid + 16];
        sdata[tid] += sdata[tid + 8];
        sdata[tid] += sdata[tid + 4];
        sdata[tid] += sdata[tid + 2];
        sdata[tid] += sdata[tid + 1];
    }

    if (tid == 0) y[0] = sdata[0];
}
```



```
__device__ void warpReduce(volatile FLOAT *sdata, int tid)
{
    sdata[tid] += sdata[tid + 32];
    sdata[tid] += sdata[tid + 16];
    sdata[tid] += sdata[tid + 8];
    sdata[tid] += sdata[tid + 4];
    sdata[tid] += sdata[tid + 2];
    sdata[tid] += sdata[tid + 1];
}

__global__ void reduction_2(const FLOAT *x, FLOAT *y)
{
    __shared__ FLOAT sdata[256];
    int tid = threadIdx.x;

    /* load data to shared mem */
    sdata[tid] = x[tid];
    __syncthreads();

    /* reduction using shared mem */
    if (tid < 128) sdata[tid] += sdata[tid + 128];
    __syncthreads();

    if (tid < 64) sdata[tid] += sdata[tid + 64];
    __syncthreads();

    if (tid < 32) warpReduce(sdata, tid);

    if (tid == 0) y[0] = sdata[0];
}
```

THANK YOU