CMPE 214
GPU Architecture & Programming

Lecture 5.
Advanced GPU Programming (2)

Haonan Wang

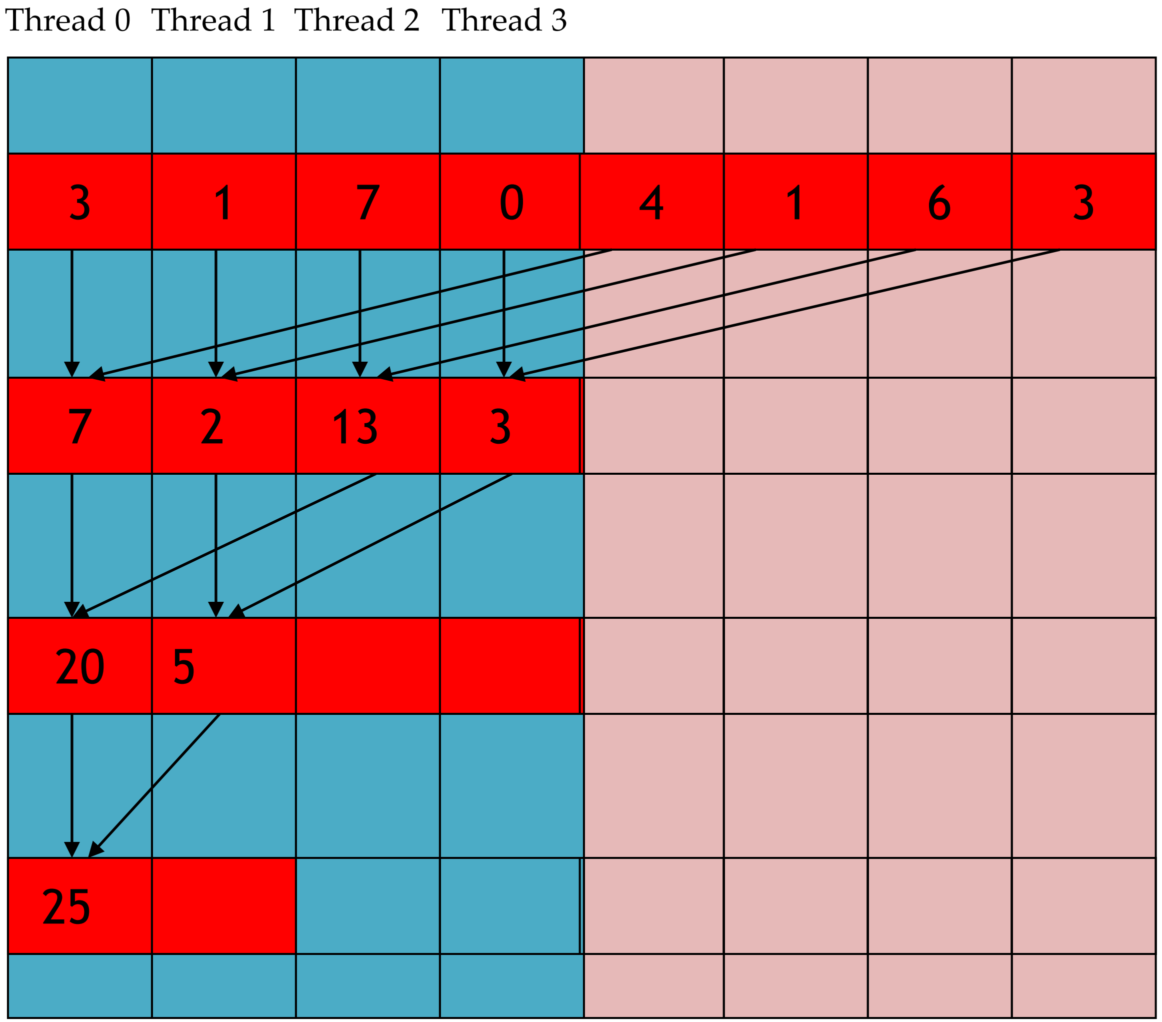
SJSU SAN JOSÉ STATE UNIVERSITY

Naïve Reduction Kernel

```
…

__shared__ float partialSum[2*BLOCK_SIZE];

unsigned int t = threadIdx.x;
unsigned int start = 2 * blockIdx.x * blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim + t] = input[start + blockDim.x + t];

for (unsigned int stride = 1; stride <= blockDim.x; stride *= 2){
   __syncthreads();

   if (t % stride == 0)
     partialSum[2 * t]+= partialSum[2 * t + stride];
}
…
```

How to sum up the results from different blocks?

A better Thread Organization



Modified Reduction Kernel

```
…

__shared__ float partialSum[2*BLOCK_SIZE];

unsigned int t = threadIdx.x;
unsigned int start = 2 * blockIdx.x * blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim + t] = input[start + blockDim.x + t];

for (unsigned int stride = blockDim.x/2; stride > 0; stride /= 2){
    __syncthreads();

    if (tid < stride)
        partialSum[tid] += partialSum[tid + stride];
}
…
```

How to further optimize?

Loop Unrolling Within A Warp

```
…

for (unsigned int stride = blockDim.x/2; stride > 32; stride /= 2){
    if (tid < stride)
        partialSum[tid] += partialSum[tid + stride];
    __syncthreads();
}

if( tid < 32 ) partialSum[t] += partialSum[t + 32];
if( tid < 16 ) partialSum[t] += partialSum[t + 16];
if( tid < 8 ) partialSum[t] += partialSum[t + 8];
if( tid < 4 ) partialSum[t] += partialSum[t + 4];
if( tid < 2 ) partialSum[t] += partialSum[t + 2];
if( tid < 1 ) partialSum[t] += partialSum[t + 1];

…
```

What is good about removing the for statement?

Already in lockstep within a warp

Loop Unrolling in General

```
#pragma unroll
for (unsigned intstride = 32; stride > 0; stride /= 2){
    if (tid < stride) partialSum[tid] += partialSum[tid + stride];
}
```

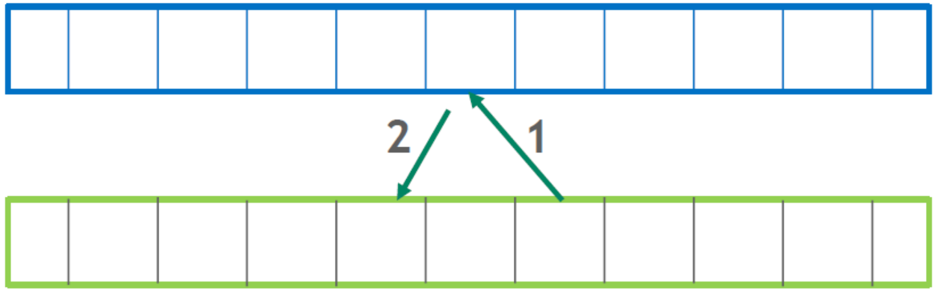
Is the same as:

```
if( tid < 32 ) partialSum[t] += partialSum[t + 32];
if( tid < 16 ) partialSum[t] += partialSum[t + 16];
if( tid < 8 ) partialSum[t] += partialSum[t + 8];
if( tid < 4 ) partialSum[t] += partialSum[t + 4];
if( tid < 2 ) partialSum[t] += partialSum[t + 2];
if( tid < 1 ) partialSum[t] += partialSum[t + 1];
```

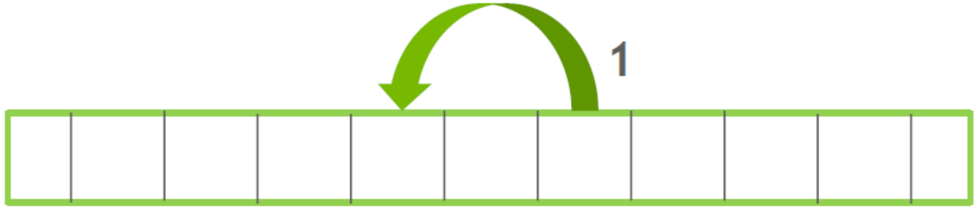
Unroll factor can also be specified: #pragma unroll <UnrollFactor>

Warp Shuffle Instructions (1)

Threads communicate via shared memory:



Treads communicate directly?



- **T __shfl_sync(unsigned mask, T var, int srcLane, int width=warpSize);**
  - Copy from lane ID (arbitrary pattern)
- **T __shfl_up_sync(unsigned mask, T var, unsigned int delta, int width=warpSize);**
  - Copy from delta/offset lower lane
- **T __shfl_down_sync(unsigned mask, T var, unsigned int delta, int width=warpSize);**
  - Copy from delta/offset higher lane
- **T __shfl_xor_sync(unsigned mask, T var, int laneMask, int width=warpSize);**
  - Copy from calculated lane ID (calculated pattern)

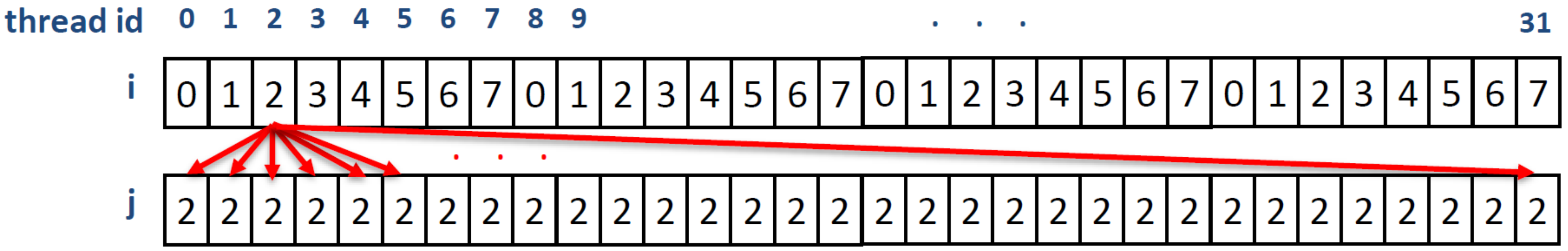
Warp Shuffle Instructions (2)

- **T __shfl_sync(unsigned mask, T var, int srcLane, int width=warpSize);**
  - Copy from lane ID (arbitrary pattern)

- **Example 1:**

  Int i = threadIdx.x % 8;

  Int j = __shfl_sync(0xffffffff, i, 2);

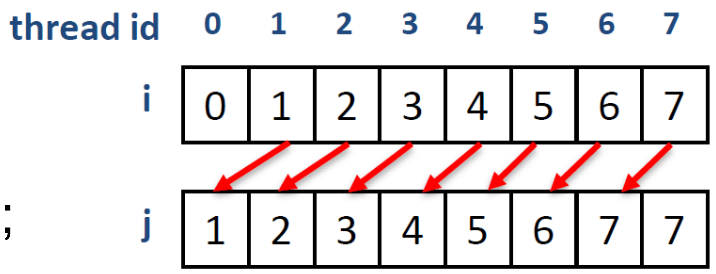


thread id	0	1	2	3	4	5	6	7	8	9								.	.	.												31
i	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
j	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

- **Example 2:**

  int i = threadIdx.x % 32;

  int j = __shfl_sync(0xffffffff, i, (threadIdx.x+1)%32, 8);

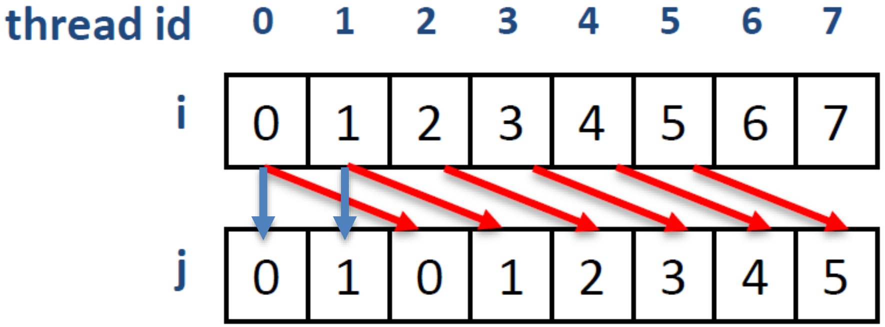


thread id	0	1	2	3	4	5	6	7
i	0	1	2	3	4	5	6	7
j	1	2	3	4	5	6	7	7

Warp Shuffle Instructions (3)

- **T __shfl_up_sync(unsigned mask, T var, unsigned int delta, int width=warpSize);**
  - Copy from delta/offset lower lane
- **Example:**

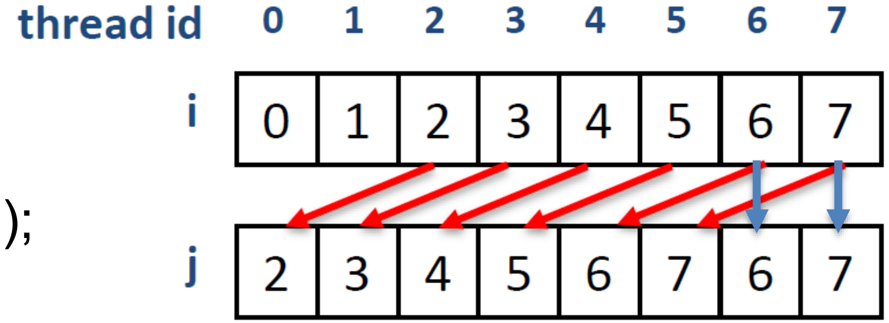
```
int i = threadIdx.x % 32;
int j = __shfl_up_sync(0xffffffff, i, 2, 8);
```



thread id	0	1	2	3	4	5	6	7
i	0	1	2	3	4	5	6	7
j	0	1	0	1	2	3	4	5

- **T __shfl_down_sync(unsigned mask, T var, unsigned int delta, int width=warpSize);**
  - Copy from delta/offset higher lane:
- **Example:**

```
int i = threadIdx.x % 32;
int j = __shfl_down_sync(0xffffffff, i, 2, 8);
```



thread id	0	1	2	3	4	5	6	7
i	0	1	2	3	4	5	6	7
j	2	3	4	5	6	7	6	7

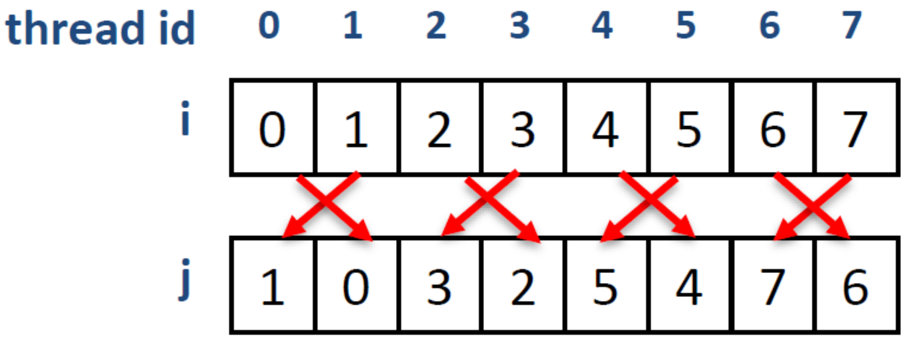
Warp Shuffle Instructions (4)

- **T __shfl_xor_sync(unsigned mask, T var, int laneMask, int width=warpSize);**
  - Copy from calculated lane ID

- **Example:**

  Int i = threadIdx.x % 32;

  Int j = __shfl_xor_sync(0xffffffff, i, 1, 8);



thread id	0	1	2	3	4	5	6	7
i	0	1	2	3	4	5	6	7
j	1	0	3	2	5	4	7	6

- **An XOR (exclusive or) operation is performed between laneMask and the calling thread's laneID to determine the lane from which to copy the value.**

- **Be careful with conditional code!**
  - Threads may only read data from another active thread.
  - If the target thread is inactive, the retrieved value is undefined.

Using Warp Shuffle Instructions

```
…

for (unsigned int stride = blockDim.x/2; stride > 16; stride /= 2){
    if (tid < stride)
        partialSum[tid] += partialSum[tid + stride];
    __syncthreads();
}

float shuffle_sum = 0;
if( tid < 32 ){
    shuffle_sum = partialSum[tid];
    for (int offset = 16; offset > 0; offset /= 2){
         shuffle_sum += __shfl_down_sync(0xffffffff, shuffle_sum, offset);
    }
}
…
```

Spinlocks with Atomic

```
int compare_and_swap(int *value, int expected, int new_value)
{
    int original_value = *value;
    if (*value == expected)
        *value = new_value;
    return original_value;
}
```

```
while (true){
    while (compare_and_swap(&lock, 0, 1) != 0)
    {
        // busy wait
    }

    critical_region();
    lock = 0;
    noncritical_region();
}
```

Locks in CUDA (1)

```
__global__ void myKernel(Lock lock, int *A){

    lock.lock();
    // Serial part (critical region)
    lock.unlock();
}

Int main(){
…

    Lock lock;
    myKernel<<<m, n>>> (lock, A);

…
}
```

Locks in CUDA (2)

```
class Lock {
    Int *mutex; // lock value

public:
    Lock () {
        int state = 0;
        cudaMalloc((void**) &mutex, sizeof(int));
        cudaMemcpy(mutex, &state, sizeof(int), cudaMemcpyHostToDevice));
    }

    ~Lock () {
        cudaFree(mutex);
    }

…
}
```

Locks in CUDA (3)

```
class Lock {
…

    __device__ void lock (){
        while (atomicCAS(mutex, 0, 1)! = 0);
    }

    __device__ void unlock (){
        *mutex= 0;
    }

}
```

Locks in CUDA (4)

```
__global__ void myKernel(Lock lock, int *A){

    lock.lock();
    // Serial part (critical region)
    lock.unlock();
}
```

Any issue?

```
__global__ void myKernel(Lock lock, int *A){

    if((threadIdx.x% 32) == 0){
        lock.lock();
        // Serial part (critical region)
        lock.unlock();
    }
}
```

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY