Семинар 18
Технология CUDA
Параллельная редукция

Михаил Курносов

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2015

Редукция (reduction)

- **Задан** массив $v[0..n-1]$ из n элементов и ассоциативная операция $\otimes$
- **Необходимо** вычислить результаты редукции

$$r = v[0] \otimes v[1] \otimes \ldots \otimes v[n-1]$$

- **Пример:**

$$v[0..5] = [5, 8, 3, 12, 1, 7]$$

$$r = ((((5 + 8) + 3) + 12) + 1) + 7 = 36$$

Последовательная версия (float)

```
void reduce_cpu(float *v, int n, float *sum)
{

    float s = 0.0;
    for (int i = 0; i < n; i++)
        s += v[i];
    *sum = s;
}

int n = 10000;
for (size_t i = 0; i < n; i++)
    v[i] = i + 1.0;

reduce_cpu(v, n, &sum);
```

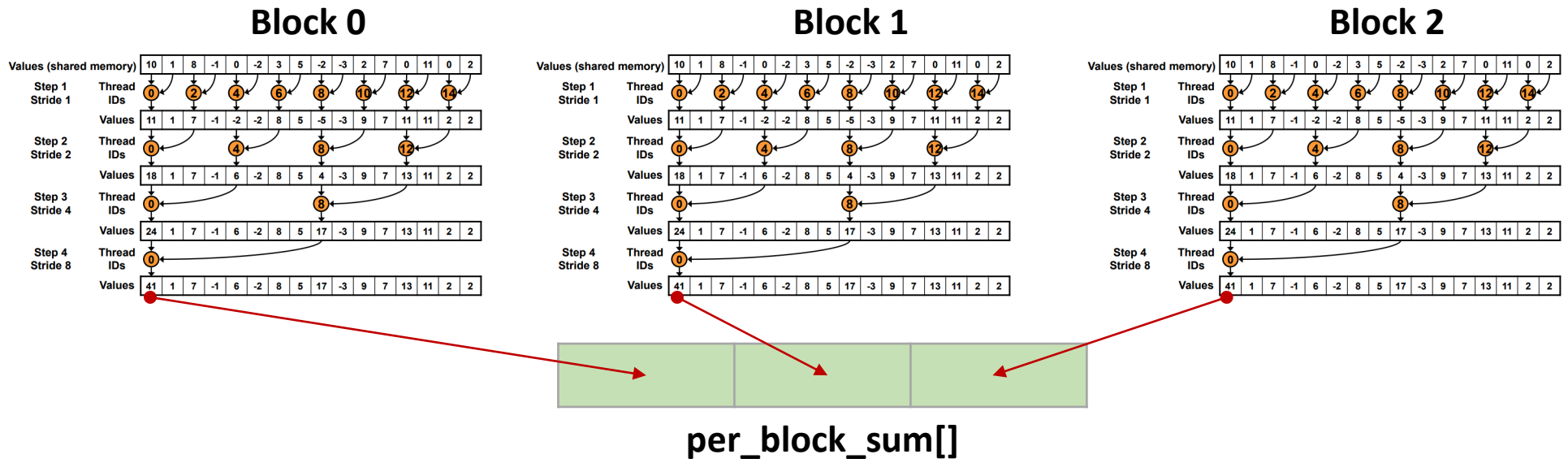
- **Floating-point Summation** // http://www.drdobbs.com/floating-point-summation/184403224
- David Goldberg. **What Every Computer Scientist Should Know About Floating-Point Arithmetic** // ACM Computing Surveys, Vol. 23, #1, March 1991, pp. 5-48.

```
# Real sum = 50005000
Sum (CPU) = 50002896.000000, err = 2104.000000
```

Последовательная версия (int)

```
void reduce_cpu(int *v, int n, int *sum)
{
    int s = 0.0;
    for (int i = 0; i < n; i++)
        s += v[i];
    *sum = s;
}
```

Параллельная редукция v1

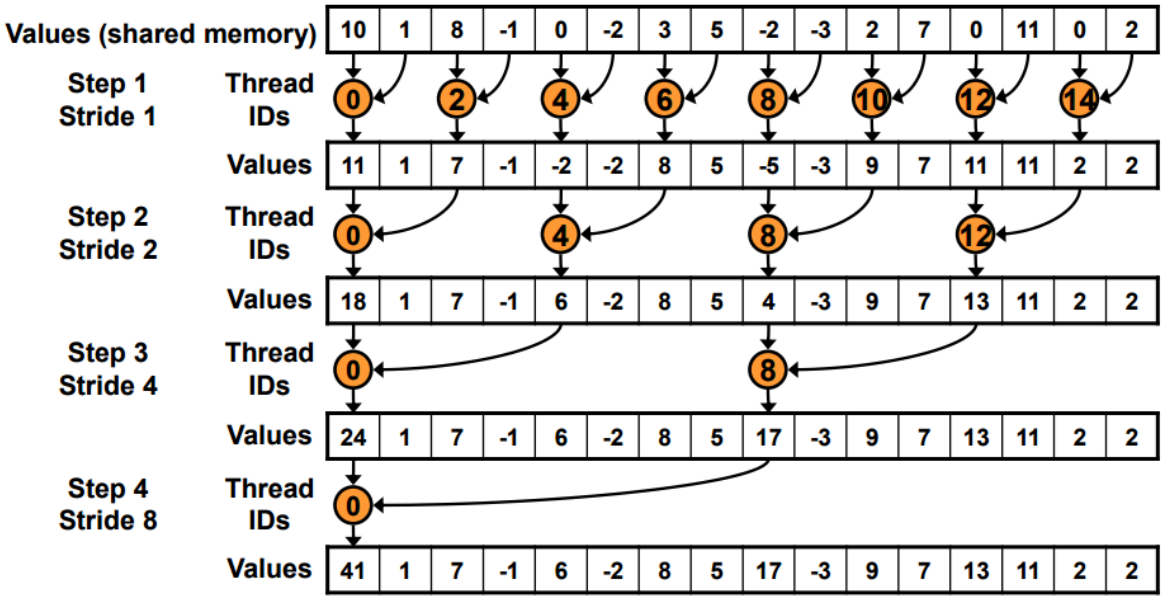


- Каждый блок потоков обрабатывает часть массива *v*[0..*n* - 1]
- Поток 0 каждого блока сохраняет результат редукции в массив per_block_sum[0..blocks - 1]
- Для выполнения редукции массива per_block_sum[] нужен еще один шаг на GPU или CPU:

```
cudaMemcpy(sums, per_block_sum, sizeof(int) * blocks, cudaMemcpyDeviceToHost);
int sum_gpu = 0;
for (int i = 0; i < blocks; i++)
    sum_gpu += sums[i];
```

https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf

Параллельная редукция v1




```
for (int s = 1; s < blockDim.x; s *= 2) {
    if (tid % (2 * s) == 0)
        sdata[tid] += sdata[tid + s];
}
```

- Каждый поток по своему номеру tid определяет с кем ему взаимодействовать
- На шаге 1 каждый 2-й поток взаимодействует с соседом справа на расстоянии s = 1
- На шаге 2 каждый 4-й поток взаимодействует с соседом справа на расстоянии s = 2
- На шаге 3 каждый 8-й поток взаимодействует с соседом справа на расстоянии s = 4
- …
- Всего шагов $O(\log n)$

https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf

Параллельная редукция v1

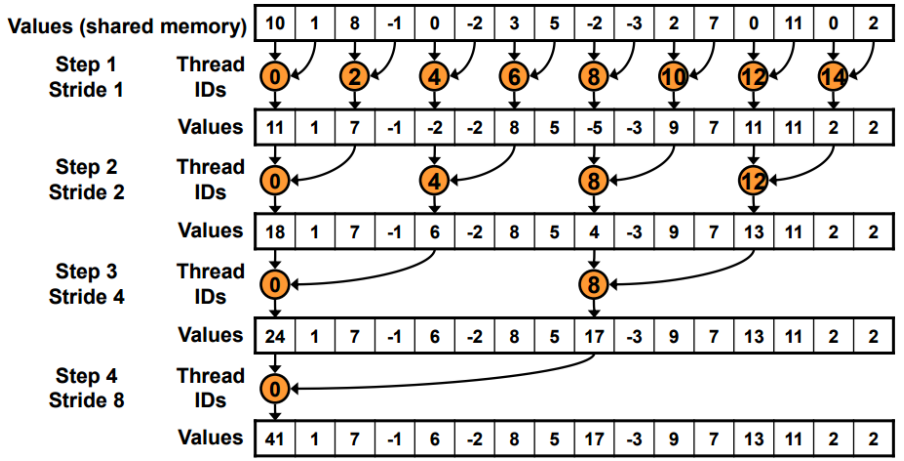
```
const int block_size = 1024;
const int n = 4 * (1 << 20);

int main()
{
    size_t size = sizeof(int) * n;
    int *v = (int *)malloc(size);
    for (size_t i = 0; i < n; i++)
        v[i] = i + 1;

    int sum;
    reduce_cpu(v, n, &sum);

    /* Allocate on device */
    int threads_per_block = block_size;
    int blocks = (n + threads_per_block - 1) / threads_per_block;
    int *dv, *per_block_sum;
    int *sums = (int *)malloc(sizeof(int) * blocks);
    tmem = -wtime();
    cudaMalloc((void **)&per_block_sum, sizeof(int) * blocks);
    cudaMalloc((void **)&dv, size);
    cudaMemcpy(dv, v, size, cudaMemcpyHostToDevice);
    tmem += wtime();
```

Параллельная редукция v1

```
/* Compute per block sum: stage 1 */
tgpu = -wtime();
reduce_per_block<<<blocks, threads_per_block>>>(dv, n, per_block_sum);
cudaDeviceSynchronize();
tgpu += wtime();

tmem = -wtime();
cudaMemcpy(sums, per_block_sum, sizeof(int) * blocks, cudaMemcpyDeviceToHost);
tmem += wtime();

/* Compute block sum: stage 2 */
tgpu -= wtime();
int sum_gpu = 0;
for (int i = 0; i < blocks; i++)
    sum_gpu += sums[i];
tgpu += wtime();

printf("CPU version (sec.): %.6f\n", tcpu);
printf("GPU version (sec.): %.6f\n", tgpu);
printf("GPU bandwidth (GiB/s): %.2f\n", 1.0e-9 * size / (tgpu + tmem));
printf("Speedup: %.2f\n", tcpu / tgpu);
printf("Speedup (with mem ops.): %.2f\n", tcpu / (tgpu + tmem));
```

Параллельная редукция v1

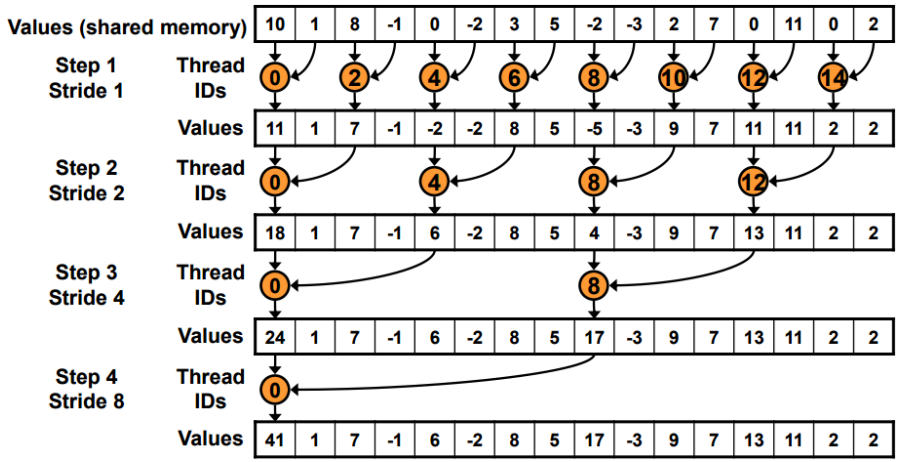
```
__global__ void reduce_per_block(int *v, int n, int *per_block_sum)
{
    __shared__ int sdata[block_size];
    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) {
        sdata[tid] = v[i];
        __syncthreads();

        for (int s = 1; s < blockDim.x; s *= 2) {
            if (tid % (2 * s) == 0)
                sdata[tid] += sdata[tid + s];
            __syncthreads();
        }
        if (tid == 0)
            per_block_sum[blockIdx.x] = sdata[0];
    }
}
```




```
CUDA kernel launch with 4096 blocks of 1024 threads
Sum (CPU) = 2097152
Sum (GPU) = 2097152
CPU version (sec.): 0.005956
GPU version (sec.): 0.002198
GPU bandwidth (GiB/s): 7.56
Speedup: 2.71
Speedup (with mem ops.): 2.68
```

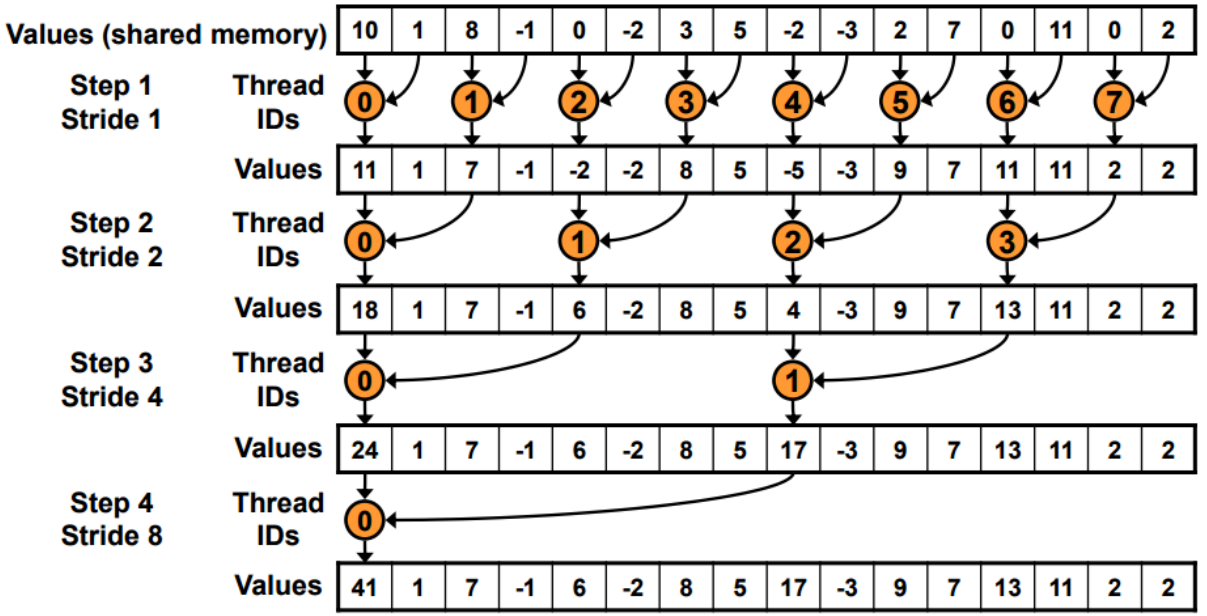
Параллельная редукция v1

```
__global__ void reduce_per_block(int *v, int n, int *per_block_sum)
{
    __shared__ int sdata[block_size];
    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) {
        sdata[tid] = v[i];
        __syncthreads();

        for (int s = 1; s < blockDim.x; s *= 2) {
            if (tid % (2 * s) == 0)
                sdata[tid] += sdata[tid + s];
            __syncthreads();
        }
        if (tid == 0)
            per_block_sum[blockIdx.x] = sdata[0];
    }
}
```



Результат условия зависит от номера потока – часть потоков деактивируется (control flow divergence)

Параллельная редукция v2



Control flow divergence!

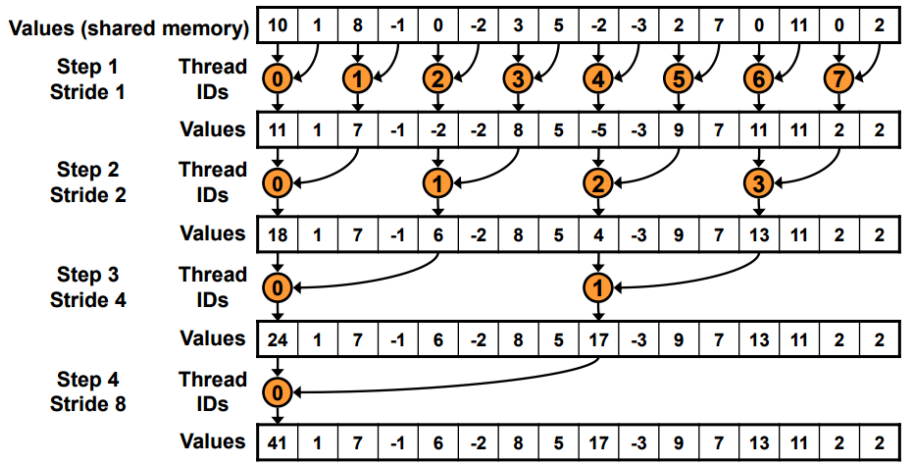
```
for (int s = 1; s < blockDim.x; s *= 2) {
    if (tid % (2 * s) == 0)
        sdata[tid] += sdata[tid + s];
    __syncthreads();
}
```

```
for (int s = 1; s < blockDim.x; s *= 2) {
    int index = 2 * s * tid;
    if (index < blockDim.x)
        sdata[index] += sdata[index + s];
        __syncthreads();
}
```

https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf

Параллельная редукция v2

```
__global__ void reduce_per_block(int *v, int n, int *per_block_sum)
{
    __shared__ int sdata[block_size];
    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) {
        sdata[tid] = v[i];
        __syncthreads();

        for (int s = 1; s < blockDim.x; s *= 2) {
            int index = 2 * s * tid;

            if (index < blockDim.x)
                sdata[index] += sdata[index + s];
            __syncthreads();
        }
        if (tid == 0)
            per_block_sum[blockIdx.x] = sdata[0];
    }
}
```




```
CUDA kernel launch with 4096 blocks of 1024 threads
Sum (CPU) = 2097152
Sum (GPU) = 2097152
CPU version (sec.): 0.006481
GPU version (sec.): 0.001968
GPU bandwidth (GiB/s): 8.44
Speedup: 3.29
Speedup (with mem ops.): 3.26
```

Параллельная редукция v2

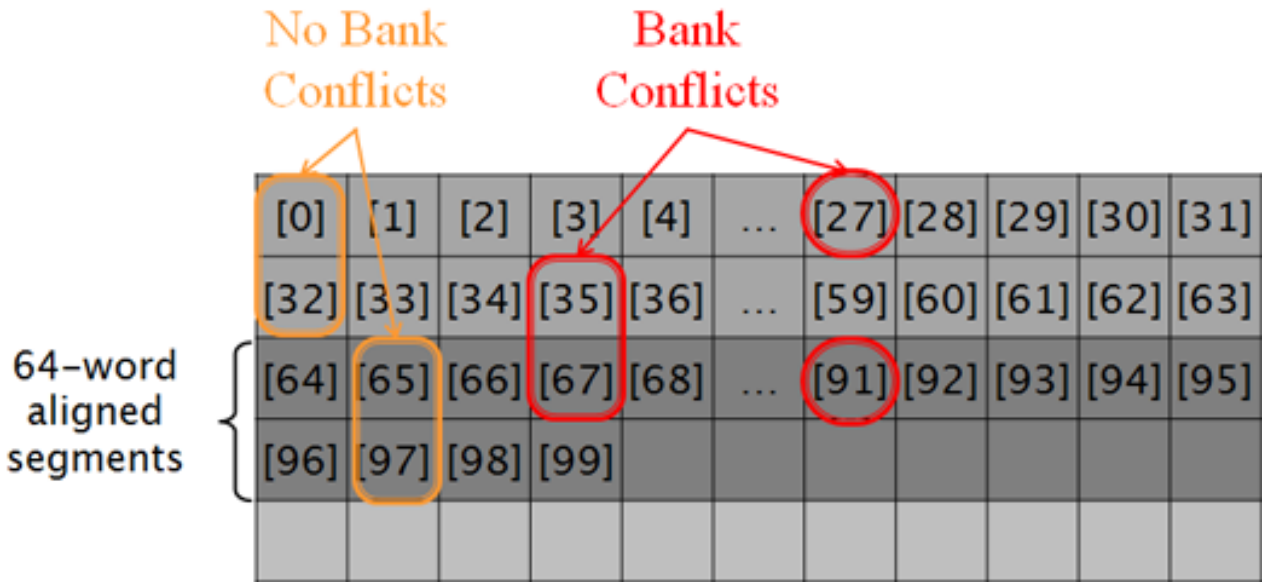
```
__global__ void reduce_per_block(int *v, int n, int *per_block_sum)
{
    __shared__ int sdata[block_size];
    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) {
        sdata[tid] = v[i];
        __syncthreads();

        for (int s = 1; s < blockDim.x; s *= 2) {
            int index = 2 * s * tid;

            if (index < blockDim.x)
                sdata[index] += sdata[index + s];
            __syncthreads();
        }
        if (tid == 0)
            per_block_sum[blockIdx.x] = sdata[0];
    }
}
```

- Обращение к разделяемой памяти выполняется через 32 параллельно функционирующих банка
- Каждый банк 4 байта (настраивается)
- Номер банка = адрес % 32
- Одновременный доступ к одному банку сериализуется!

TID	S = 1	S = 2	S = 4
0	0-1 // banks 0, 1	0-2 **// banks 0, 1**	0-4 **// banks 0, 1**
1	2-3 // banks 2, 3	4-6 **// banks 2, 3**	
2	4-5 // banks 4, 5	8-10 **// banks 4, 5**	
3	6-7 // banks 6, 7	12-14 **// banks 6, 7**	24-28 // banks 24, 28
4	8-9 // banks 8, 9	16-18 // banks 8, 9	32-36 **// banks 0, 2**
…			
8		32-34 **// banks 0, 2**	
9		36-38 **// banks 4, 6**	

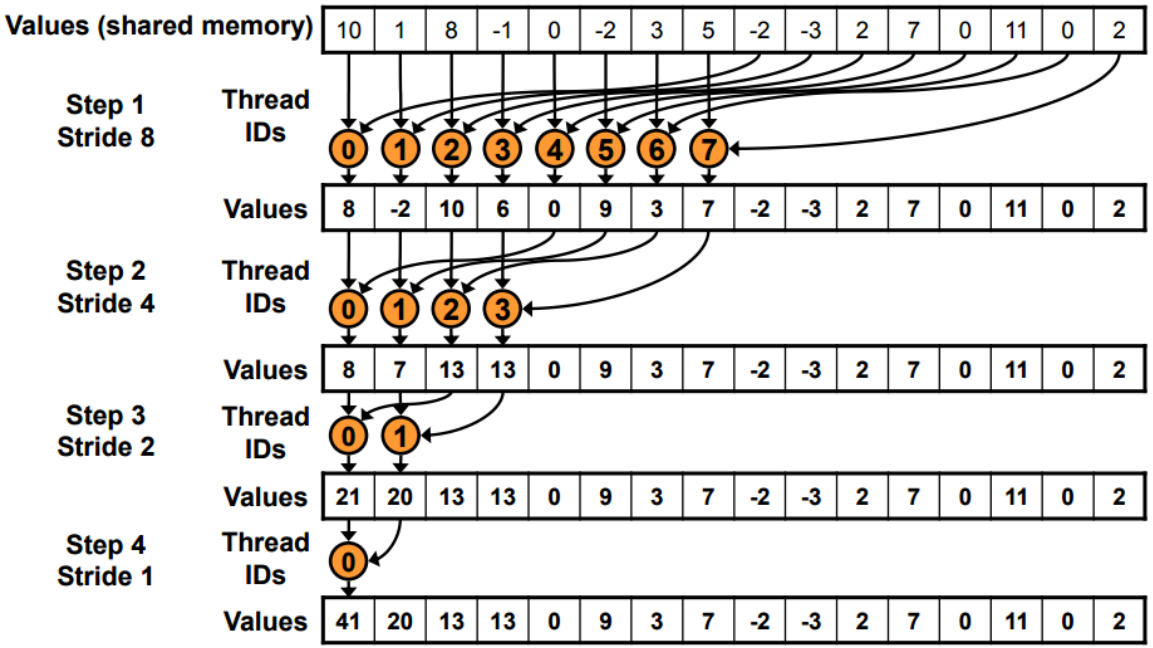
http://gpgpu.org/wp/wp-content/uploads/2013/09/08-opti-smem-instr.pdf

Shared memory bank conflicts



http://acceleware.com/blog/maximizing-shared-memory-bandwidth-nvidia-kepler-gpus

Параллельная редукция v3



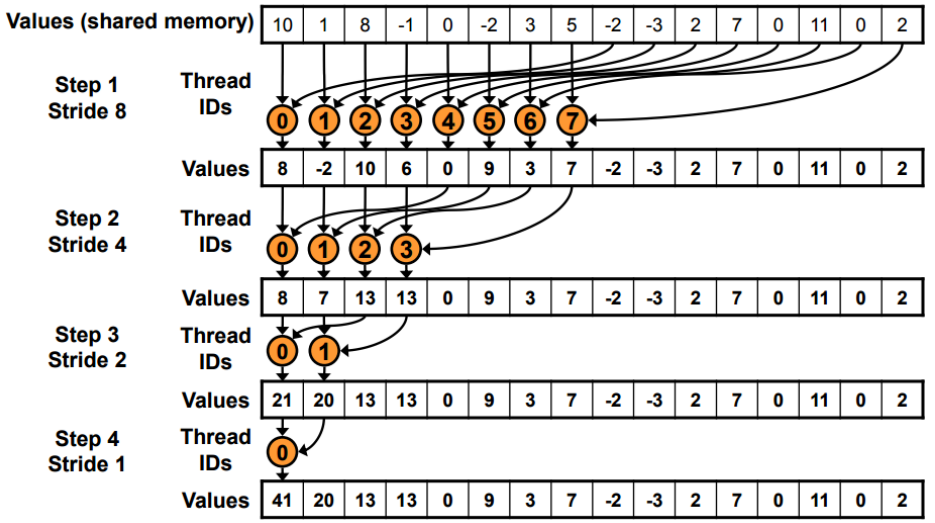
Bank conflicts

```
for (int s = 1; s < blockDim.x; s *= 2) {
    int index = 2 * s * tid;
    if (index < blockDim.x)
        sdata[index] += sdata[index + s];
        __syncthreads();
}
```

```
for (int s = blockDim.x / 2; s > 0; s >>= 1) {
    if (tid < s)
        sdata[tid] += sdata[tid + s];
    __syncthreads();
}
```

Параллельная редукция v3

```
__global__ void reduce_per_block(int *v, int n, int *per_block_sum)
{
    __shared__ int sdata[block_size];
    int tid = threadIdx.x;
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) {
        sdata[tid] = v[i];
        __syncthreads();

        for (int s = blockDim.x / 2; s > 0; s >>= 1) {
            if (tid < s)
                sdata[tid] += sdata[tid + s];
            __syncthreads();
        }
        if (tid == 0)
            per_block_sum[blockIdx.x] = sdata[0];
    }
}
```




```
CUDA kernel launch with 4096 blocks of 1024 threads
Sum (CPU) = 2097152
Sum (GPU) = 2097152
CPU version (sec.): 0.004661
GPU version (sec.): 0.001163
GPU bandwidth (GiB/s): 14.17
Speedup: 4.01
Speedup (with mem ops.): 3.94
```

Задание

- Реализовать второй шаг редукции на GPU – редукция по массиву per_block_sum[]
  https://code.google.com/p/stanford-cs193g-sp2010/source/browse/trunk/tutorials/sum_reduction.cu

- Реализовать первое сложение при загрузке данных в разделяемую память
  и разворачивание цикла (loop unrolling):
  https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf

- **SDK Samples**/6_Advanced/reduction
- **Faster Parallel Reductions on Kepler** //
  http://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/