Australian National University

Reducing

Dr Eric McCreath

Research School of Computer Science

The Australian National University

Introduction

The "map" operation on a GPU is straight forward, just use a lot of threads to work in parallel on different parts of the data. However with the "reduce" operation you need to be a bit more strategic to optimise for performance.

When dealing with millions of elements that need to be reduced if you end up using a serial approach it can easily dominate performance. As such you need to think how the reduction can be done in parallel.

If the reduction operator is associative, like addition with (a+b)+c=a+(b+c), then you will generally have the option of doing a tree based reduction.

Note that with floating point calculations changing the order can slightly change the final result due to the way rounding occurs. This can throw off your testing but generally will not effect correctness.

Tree Based Reduction

The basic approach is to get teams of threads within a block to reduce down to a single value. This can be done in shared memory and the results stored in global memory. Once done other kernels can be launched to further reduce the result. So a basic approach to sum a list of integers in blocks with 1024 threads is:

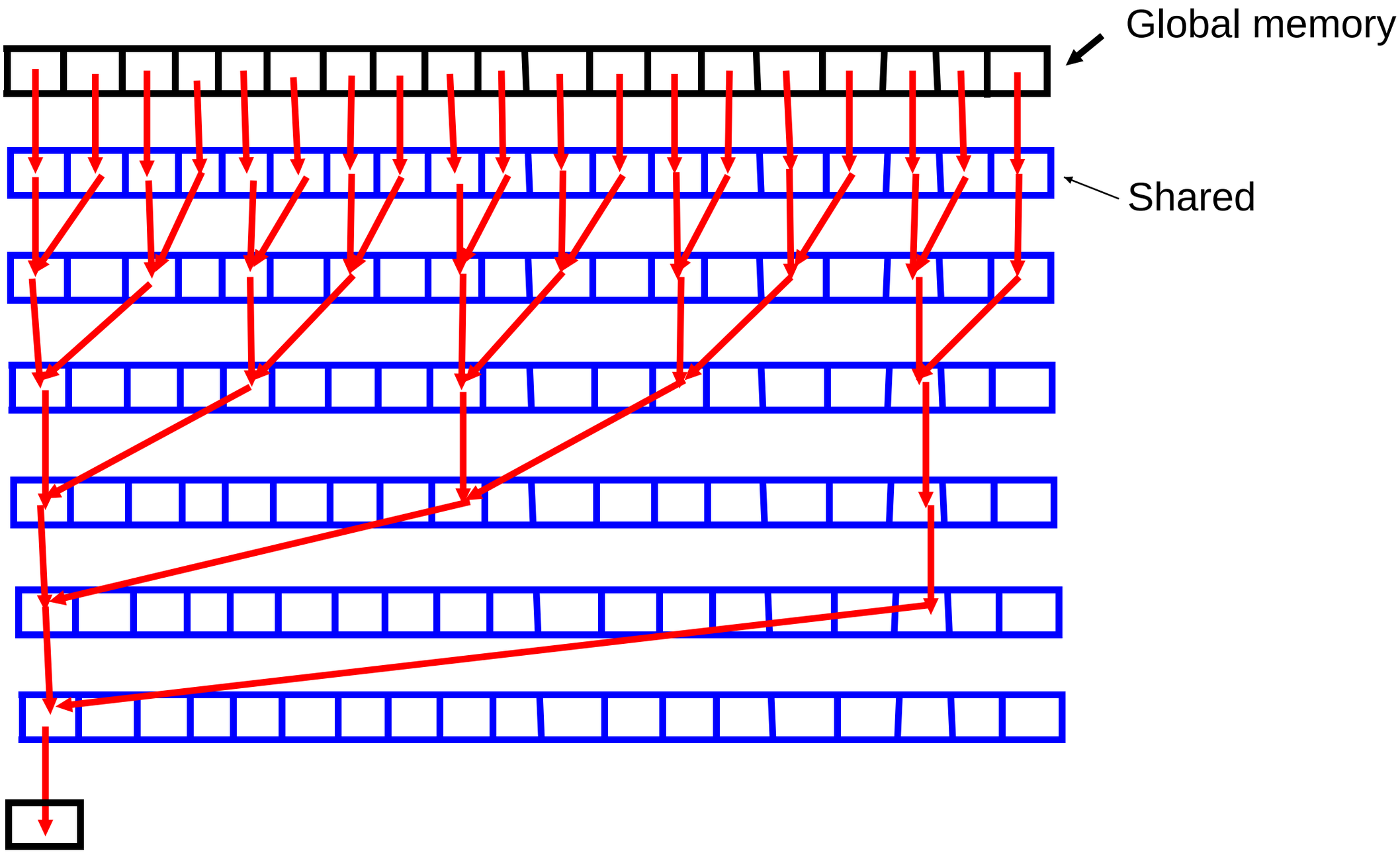
```
__global__ void reduceV1(int *list, int n, int *res) {
    __shared__ int tmp[1024];
    int i = threadIdx.x;
    tmp[i] = list[blockIdx.x*blockDim.x + threadIdx.x];
    __syncthreads();

    for(int s = 1; s < 1024; s <<= 1) {
        if (i % (s<<1) == 0 && (i+s) < 1024) tmp[i] = tmp[i] + tmp[i+s];
        __syncthreads();
    }

    if (i == 0) res[blockIdx.x] = tmp[0];
}
```

Tree based reduction

Diagrammaticly the approach is:



Tree Based Reduction

The previous approach is good although the below may be just a little better in terms of divergence.

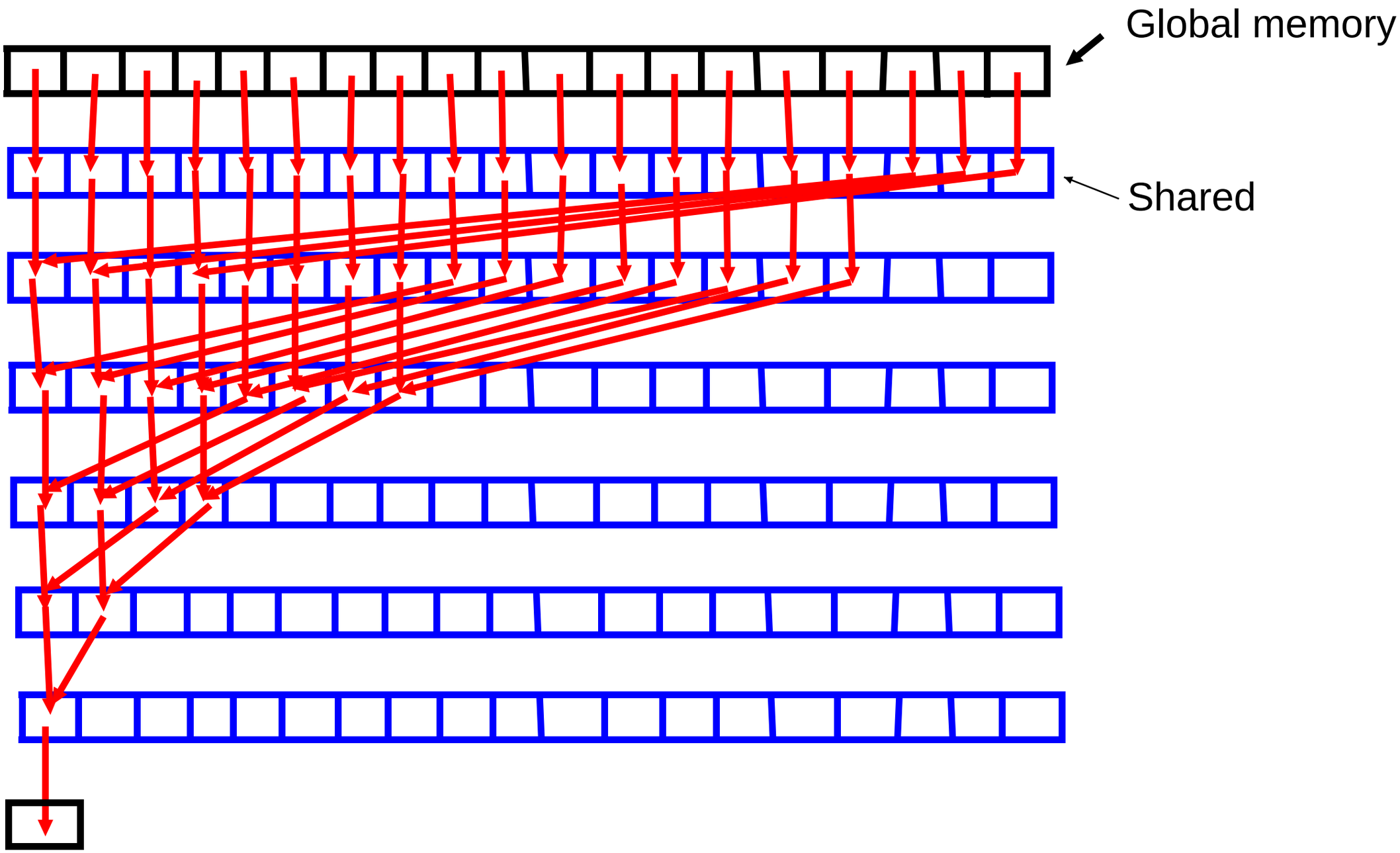
```
__global__ void reduceV2(int *list, int n, int *res) {
    __shared__ int tmp[1024];
    int i = threadIdx.x;
    tmp[i] = list[blockIdx.x*blockDim.x + threadIdx.x];
    __syncthreads();

    for(int s = (1024>>1); s > 0; s >>= 1) {
            if (i < s && i+s < 1024) tmp[i] = tmp[i] + tmp[i+s];
            __syncthreads();
    }

    if (i == 0) res[blockIdx.x] = tmp[0];
}
```

Tree based reduction

Diagrammaticly the approach is:

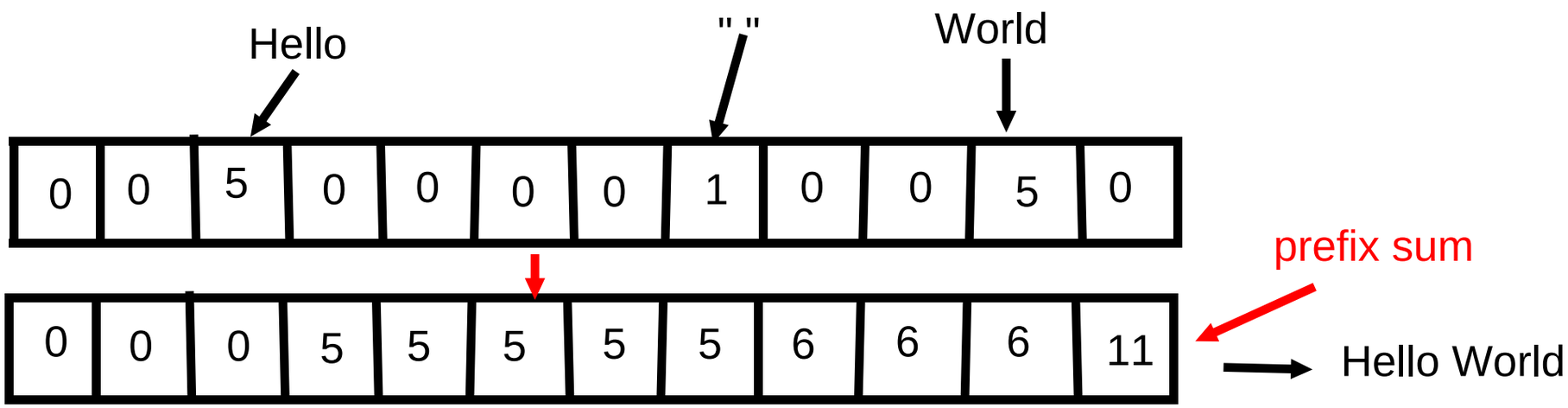


Prefix Sum or Scan Operator

If we have a variable sized output we need to work out a way of concurrently returning the result without threads writing to the same location, and also without having to use a large amount of memory and then have to filter results on the CPU.

We can use the atomicAdd or atomicInc, although if we are returning a lot of result they will serialize this process.

The operator to use is a parallel prefix sum or scan. These work in similar way to the tree based reductions and they can be work efficient.



References

- Timcheck, Stephen W., "Efficient Implementation of Reductions on GPU Architectures" (2017). Honors Research Projects. . http://ideaexchange.uakron.edu/honors_research_projects/479