Computational Science on Many-Core Architectures

360.252

Karl Rupp

Institute for Microelectronics, TU Wien
http://www.iue.tuwien.ac.at/

Zoom Channel 941 8518 8102
Q&A on Wednesday, October 12, 2022

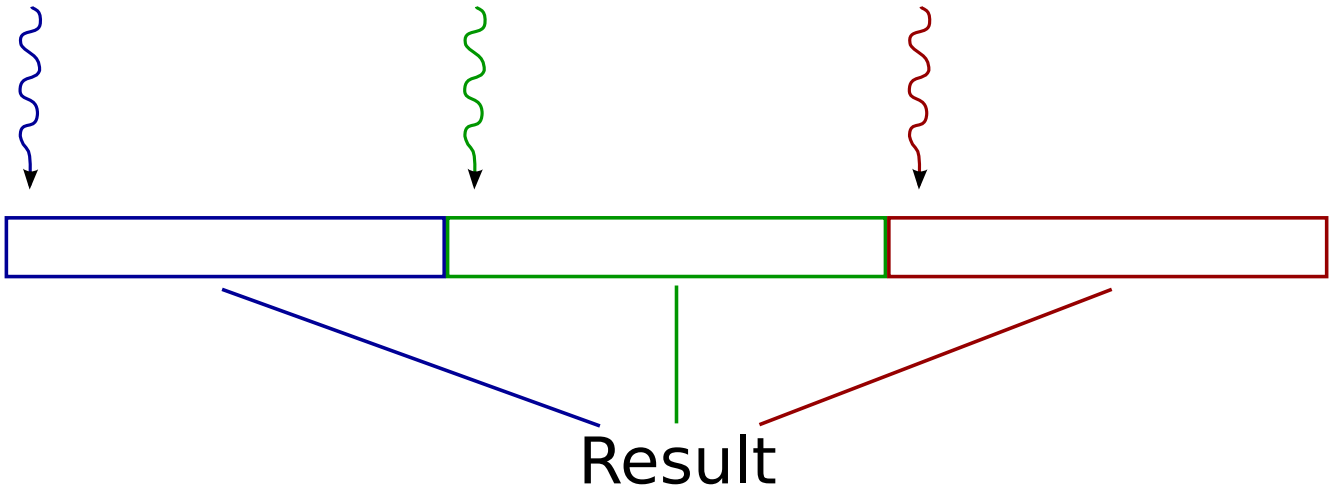
Parallel Primitives

Reductions

- Use N values to compute 1 result value
- Examples: Dot-products, vector norms, etc.

Reductions with Few Threads

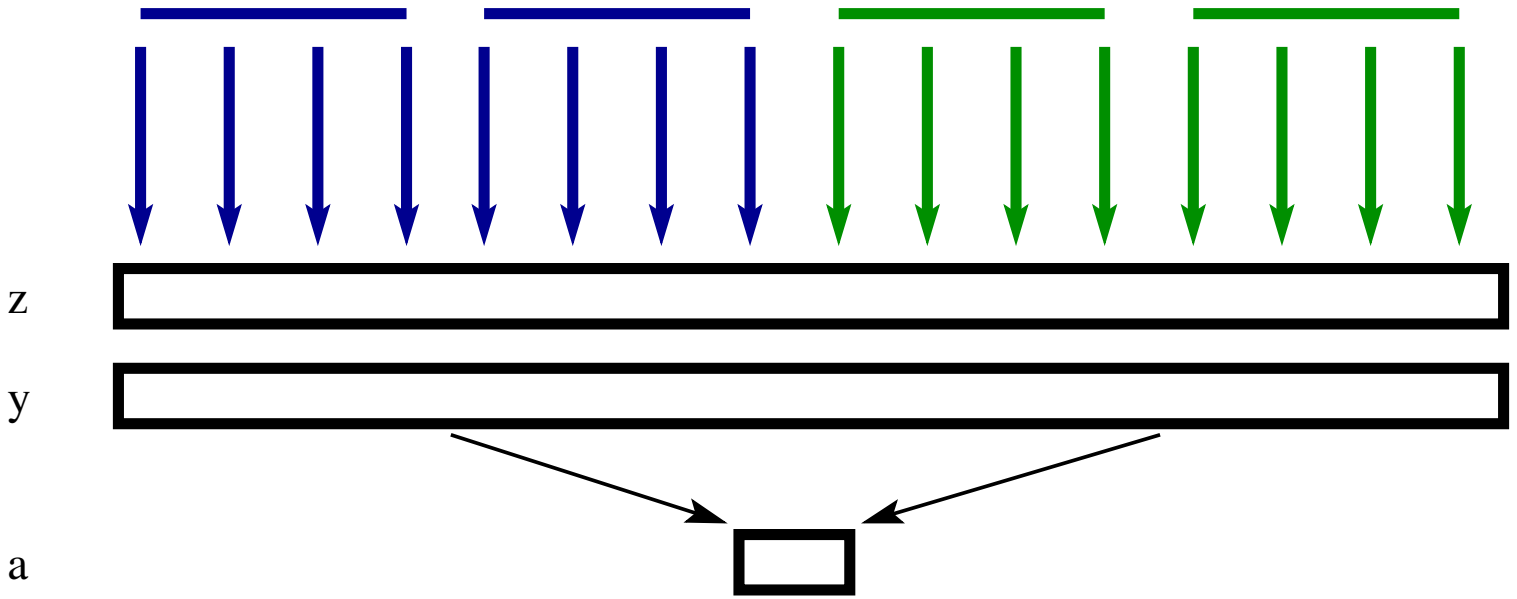
- Decompose N into chunks for each thread
- Compute chunks in parallel
- Merge results with single thread



Parallel Primitives

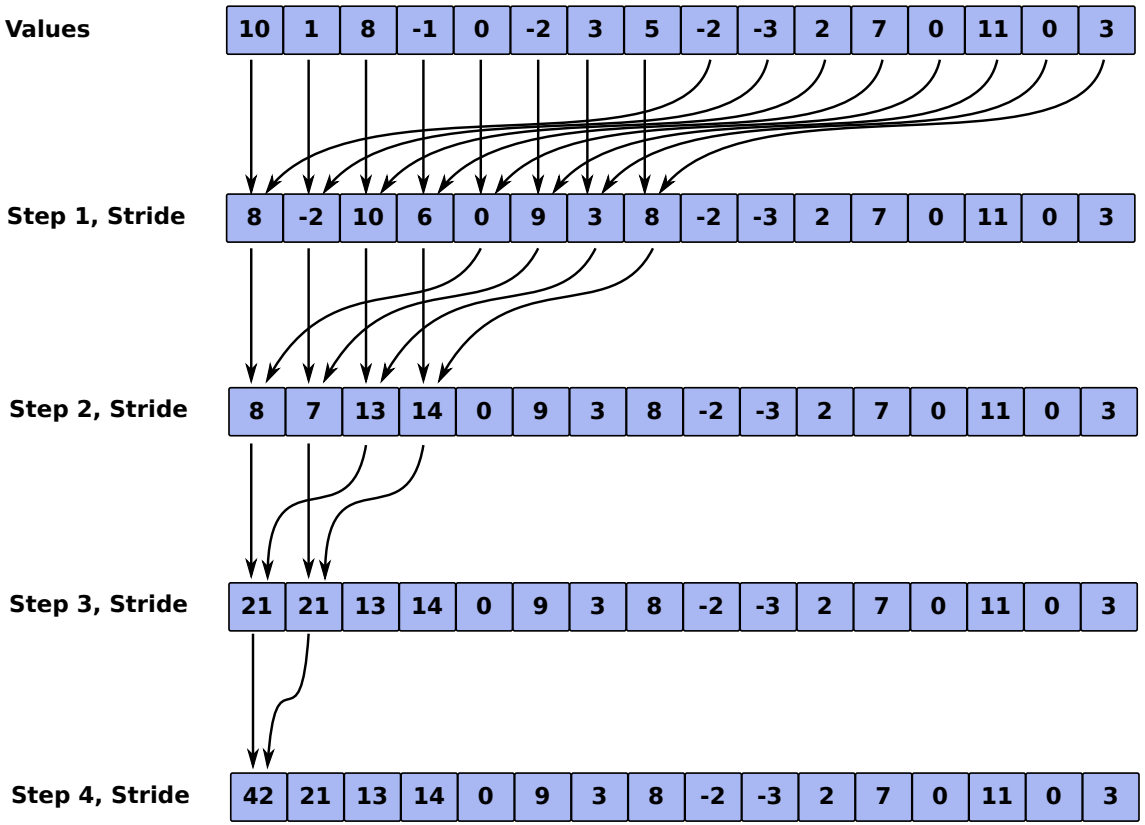
Reductions with Many Threads

- Decompose N into chunks for each workgroup
- Use fast on-chip synchronization within each workgroup
- Sum result for each workgroup separately



Parallel Primitives

Reductions with Many Threads




```
shared_m[threadIdx.x] = thread_sum;
for (int stride = blockDim.x/2; stride>0; stride/=2) {
  __syncthreads();
  if (threadIdx.x < stride)
    shared_m[threadIdx.x] += shared_m[threadIdx.x+stride];
}
```

Parallel Primitives

```
__global__ void sum_vector(const double * x, unsigned int N,
                           double * partial_results) {
  __shared__ double shared_m[256]; // shared memory for each
      thread block

  double thread_sum = 0;  // local variable for each thread
  unsigned int total_threads = blockDim.x * gridDim.x;
  int thread_id = blockIdx.x*blockDim.x + threadIdx.x;
  for (unsigned int i = thread_id; i<N; i += total_threads)
      thread_sum += x[i];

  // reduction within each block in shared memory
  shared_m[threadIdx.x] = thread_sum;
  for (unsigned int stride = blockDim.x/2; stride>0; stride/=2) {
    __syncthreads();  // synchronize threads within thread block
    if (threadIdx.x < stride)
        shared_m[threadIdx.x] += shared_m[threadIdx.x + stride];
  }

  // only first thread of block writes result
  if (threadIdx.x == 0) {
    partial_results[blockIdx.x] = shared_m[0];
    // alternative: atomicAdd(result, shared_m[0]);
} }
```