CSED490C Heterogeneous Parallel Computing

Lab6 Report

20232241 Dookyung Kang

This is my version of template.

```
// Kernel
__global__ void scan(float *input, float* aux, int len) {
  __shared__ float intm[BLOCK_SIZE];

  int tid = blockIdx.x * blockDim.x + threadIdx.x;
  if (tid < len)
    intm[threadIdx.x] = input[tid];
  for (unsigned int stride = 1; stride <= blockDim.x; stride *= 2) {
    __syncthreads();
    int index = (threadIdx.x + 1) * 2 * stride - 1;
    if (index < blockDim.x) {
      intm[index] += intm[index - stride];
    }
  }
  for (int stride = blockDim.x / 4; stride > 0; stride /= 2) {
    __syncthreads();
    int index = (threadIdx.x + 1) * 2 * stride - 1;
    if (index + stride < blockDim.x) {
      intm[index + stride] += intm[index];
    }
  }
  __syncthreads();
  if (tid < len)
    input[tid] = intm[threadIdx.x];
  if (aux && threadIdx.x == 0 && blockIdx.x + 1 < gridDim.x) {
    aux[blockIdx.x + 1] = intm[blockDim.x - 1];
  }
}

__global__ void sum_scan(float *input, float *output, float* aux, int len) {
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
  if (tid < len)
    output[tid] = input[tid] + aux[blockIdx.x];
}


// Host
// (skipped)
  gpuTKTime_start(GPU, "Allocating GPU memory.");
  float *deviceAux;
  int gridSize = (numElements + BLOCK_SIZE - 1) / BLOCK_SIZE;
  gpuTKCheck(cudaMalloc((void **)&deviceInput, numElements * sizeof(float)));
  gpuTKCheck(cudaMalloc((void **)&deviceAux, gridSize * sizeof(float)));
  gpuTKCheck(cudaMalloc((void **)&deviceOutput, numElements * sizeof(float)));
  gpuTKTime_stop(GPU, "Allocating GPU memory.");

  gpuTKTime_start(GPU, "Clearing output memory.");
  gpuTKCheck(cudaMemset(deviceOutput, 0, numElements * sizeof(float)));
  gpuTKCheck(cudaMemset(deviceAux, 0, gridSize * sizeof(float)));
  gpuTKTime_stop(GPU, "Clearing output memory.");

  gpuTKTime_start(GPU, "Copying input memory to the GPU.");
  gpuTKCheck(cudaMemcpy(deviceInput, hostInput, numElements * sizeof(float),
                     cudaMemcpyHostToDevice));
  gpuTKTime_stop(GPU, "Copying input memory to the GPU.");

  //@@ Initialize the grid and block dimensions here

  gpuTKTime_start(Compute, "Performing CUDA computation");
  //@@ Modify this to complete the functionality of the scan
  //@@ on the deivce
```

```
① scan<<<gridSize, BLOCK_SIZE>>>(deviceInput, deviceAux, numElements);
② scan<<<gridSize, BLOCK_SIZE>>>(deviceAux, NULL, gridSize);
③ sum_scan<<<gridSize, BLOCK_SIZE>>>(deviceInput, deviceOutput, deviceAux, numElements);

cudaDeviceSynchronize();
gpuTKTime_stop(Compute, "Performing CUDA computation");

gpuTKTime_start(Copy, "Copying output memory to the CPU");
gpuTKCheck(cudaMemcpy(hostOutput, deviceOutput, numElements * sizeof(float),
                cudaMemcpyDeviceToHost));
gpuTKTime_stop(Copy, "Copying output memory to the CPU");
```

Note. I intentionally violated "single kernel" constraint in assignment. To parallelize scan within thread block, hierarchical parallel scan is required. However, I've failed to find a way to synchronize within running thread blocks in single kernel, after calculating partial scan sum per thread block. Alternatively, I decided to use multiple kernels launch to achieve the assignment goal.

Note. Let L = inputLength, T = BLOCK_SIZE, tid = blockIdx * BLOCK_SIZE + threadIdx, which is the global thread index.

1. How many global memory reads are being performed by your kernel?

In ① kernel call: Global memory deviceInput is read per thread if tid < L, which means the global memory reads are occurred L times.

In ② kernel call: Global memory deviceAux is called per thread if tid < ceil(L/T) in this case.

In ③ kernel call: Both input and aux is called per thread if tid < len. In this case, global memory read is occurred 2L times.

Therefore, global memory read is performed **3L + ceil(L/T)** times.

In this case, access global aux in kernel call ③ can be alternated into shared memory, but it causes additional sync overhead.

2. How many global memory writes are being performed by your kernel?

In ① kernel call: Global memory deviceInput is written per thread if tid < L. Then, global memory deviceAux is written if running thread is first of thread block, and thread block index is not final. In this case, deviceInput is written L times and deviceAux is written ceil(L / T) – 1 times.

In ② kernel call: Global memory deviceAux is written per thread if tid < ceil(L/T) times.

In ③ kernel call: Global write to "output" is performed L times.

Therefore, global memory write is performed **2L + 2ceil(L/T) – 1** times.

3. How many times does a single thread block synchronize to reduce its portion of the array to a single value?

In ① kernel call: first __syncthreads is called exactly $\lceil \log_2 T \rceil + 1$ times, second __syncthreads is called $\lceil \log_2 T \rceil - 1$ times, and third __syncthreads is called once per thread. Therefore, thread block synchronization is performed $(2 \times \lfloor \log_2 T \rfloor + 1) \left\lceil \frac{L}{T} \right\rceil T$ times.

4. Suppose that you want to scan using a binary operator that is not commutative. Can you use a parallel scan for that?

For this algorithm, all partial binary operation is called along the right sequence with objective reduction formula. Hence, not commutative binary operator is available in this scan algorithm.

5. Is it possible to get different results from running the serial version and parallel version of scan? Explain.

It might be different, but it's wrong. It should NOT be different. Isn't it?

6. Your version of template.cpp.

Refer to the code embedded previously.

7. The result as a table/graph of kernel execution times for different input data, with the system information where you performed your evaluation. Run your implementation with the input generated by the provided dataset generator. For time measurement, use gpuTKTime start and gpuTKTime stop functions (You can find details in libgputk/README.md).

L	**64**	**112**	**1120**	**9921**	**4098**	**16656**	**30000**	**96000**	**120000**	**262144**
Alloc	0.146294	0.157691	0.145786	0.159631	0.145186	0.16219	0.15067	0.158302	0.157739	0.303996
Clear	0.022585	0.027053	0.032016	0.038249	0.033788	0.0374	0.03376	0.034248	0.032317	0.03507
H->D	0.01952	0.025029	0.020398	0.033606	0.023786	0.034279	0.035539	0.065932	0.079026	0.171282
Kernel	0.02963	0.036476	0.023109	0.025497	0.024706	0.024072	0.023755	0.028699	0.031954	0.046603
D->H	0.014871	0.015678	0.017979	0.025562	0.018014	0.029186	0.035393	0.075231	0.093219	0.193765
Free	0.010652	0.012631	0.011009	0.016273	0.011376	0.01592	0.013309	0.011109	0.012472	0.161862