Parallel Reduction

Prof. Naga Kandasamy
ECE Department, Drexel University

Reduction can be performed on the GPU in parallel using tree-style approaches. We will discuss two such techniques in this article.

Figure 1 shows an approach to reduction using eight data values $d_0, d_1, \ldots, d_7$. We create eight GPU threads as part of a $(8, 1, 1)$ thread block in which each thread first loads its data value from GPU global memory into shared memory. Reduction then proceeds in steps. We maintain a variable called $stride$ that is initialized to 1. During the first step, only threads whose indices are divisible by $stride = 2 \times stride = 2$, which are T_0, T_2, T_4, and T_6, take part in the reduction process. Each of these threads adds the value belonging to its neighbor located one position to the right to its value and generates a partial sum as shown in the figure. Barrier synchronization is used to ensure that all threads within the block wait for the partial sums to be calculated and stored before proceeding to the next step. During the second step, only threads whose indices are

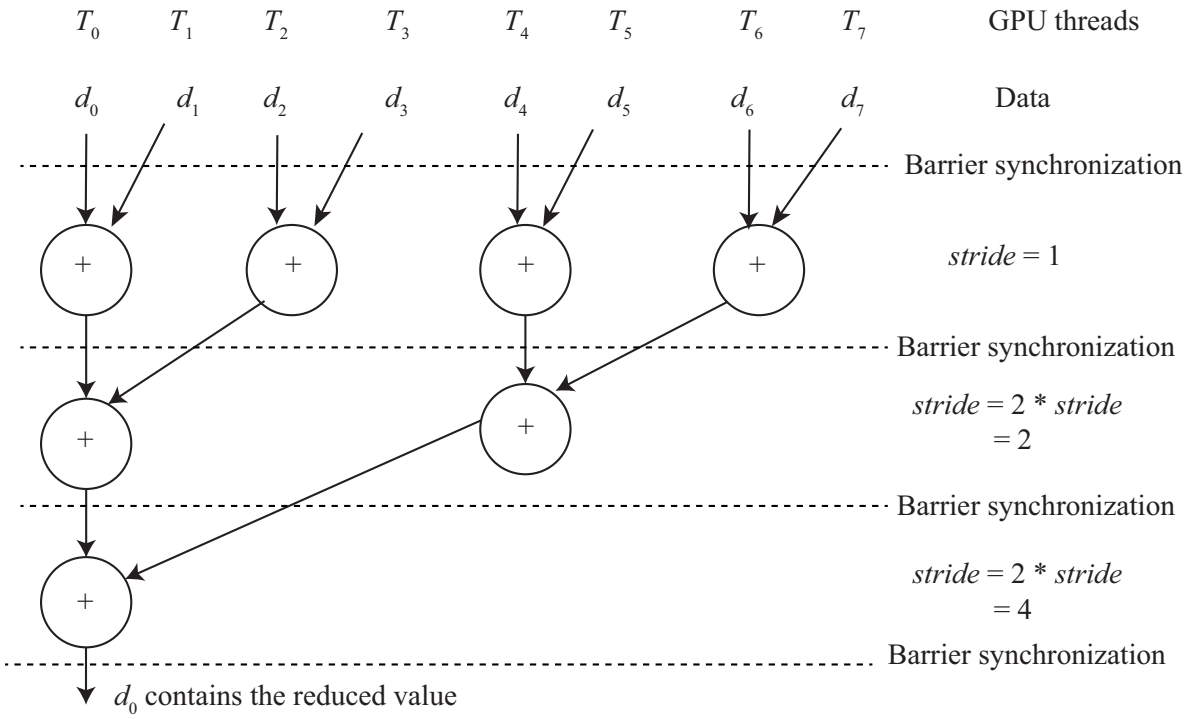

Figure 1: Tree-style reduction that exhibits branch divergence among threads within a warp.

divisible by $stride = 2 \times stride = 4$ participate in the reduction process — by adding values belonging to their neighbors located two positions to the right. The final step is performed by T_0 since its thread ID is the only one divisible by $stride = 8$. The following code snippet summarizes the reduction algorithm.

```
1
2 for (int stride = 1; stride < 8; stride *= 2) {
3     if (threadID % (2 * stride) == 0)
4         partial_sum[threadID] += partial_sum[threadID + stride];
5     __syncthreads();
6 }
```

A performance issue with the above-discussed process is the *branch divergence* exhibited by threads belonging to a single warp, leading to serialization of their execution. For example, if the warp size is 32, then during the very first step, 16 of the threads participate in the reduction process — by executing line 3 — whereas 16 threads do not and are blocked at the barrier sync in line 4.
Figure 2 shows another approach to reduction. The data array is divided in half during each step and only threads having IDs corresponding to the lower half participate in the process. This approach drastically reduces the amount of branch divergence within warps. For example, if one starts with 1024 elements to reduce, then the first 512 threads —- 16 thread warps — all participate in the reduction whereas the rest, also 16 warps, do not. Therefore, there is no branch divergence during

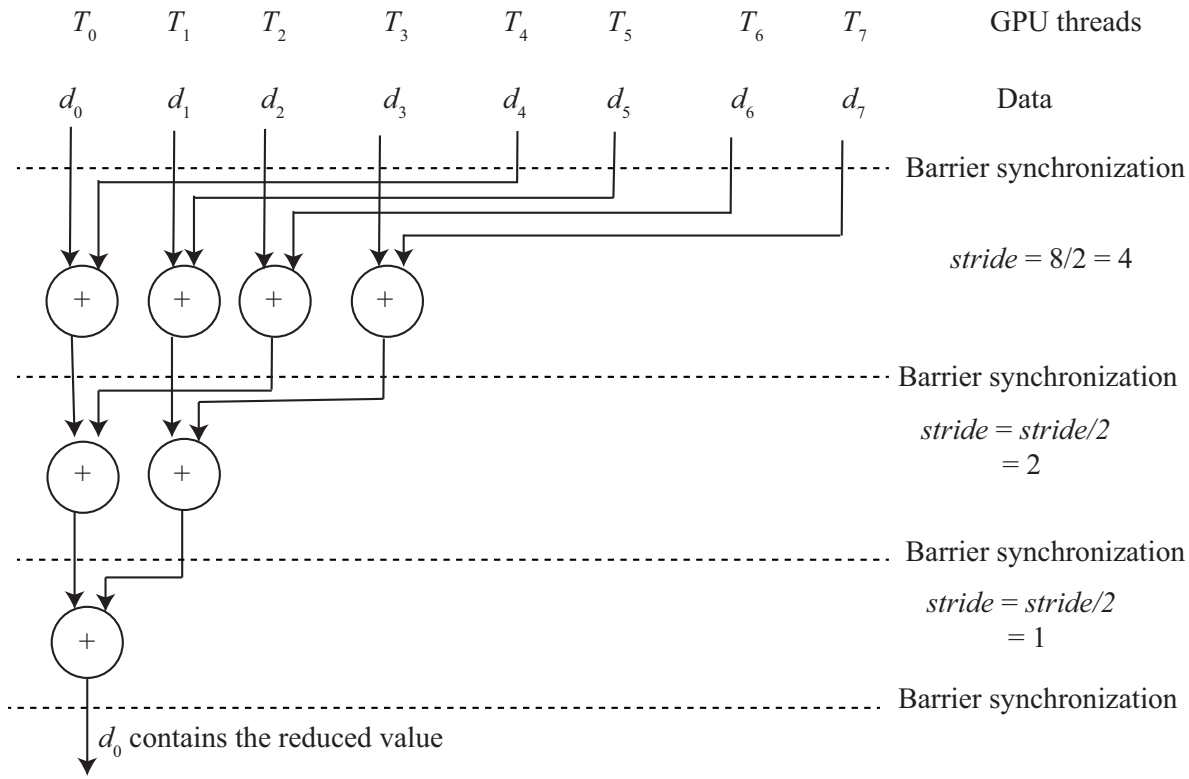

Figure 2: Tree-style reduction that exhibits minimal branch divergence among threads in a warp.

this step between threads belonging to a single warp. Similarly, during the second step all 256 lower-numbered threads or 8 warps participate in the reduction whereas the rest do not, and so on. We will experience branch divergence towards the end once the number of data items to process becomes 32. The following code snippet shows the algorithm.

```
for (int stride = 4; stride > 0; stride /= stride) {
    if(threadID < stride)
        partial_sum[threadID] += partial_sum[threadID + stride];
    __syncthreads();
}
```

Note the following important points with respect to the above-described reduction algorithms:

- The maximum size of the data array is limited to the maximum number of threads that can be accommodated within a thread block. Reducing arrays of arbitrary sizes will be discussed during lecture.
- The size of the array to be reduced should be a power of 2. If the size is not a power of 2, the array must be padded with zeros such that the size becomes a power of 2.
- The reduction is done in place, by overwriting the contents of the original array.