Group number: B
Group members:
Burr, Maximilian
Gausepohl, Dominik
Valek, Vladislav

GPU Algorithm Design, Winter Term 2024
Exercise 4

4.1 Histogram with more bins

Question :
Make a performance plot with three curves, one for `Wc=1`, one for `Wc=2` one for `Wc=4`, where on the x-axis we have `binNum=256, ..., 8192`.

Answer:
As the previous implementation used shared memory to allocate sub histogram for each warp, the measurement was not possible for the largest required amount of bins because the shared memory space on the Nvidia RTX 4080 would not suffice. This would effectively reduce the maximum amount of bins to 2048 when 4 warps are engaging in the binning to the histogram. The improved implementation for the uint datatype and the introduction processing of large amounts of bins provides a compelling performance graph as seen in Fig. 1. Each thread block now has only one sub histogram in its shared memory for its warps. Although the amount of threads increases exponentially, the throughput increases only linearly, where the baseline is set by the workgroup of 32 active threads that do not exceed 200 GBps throughput and steadily declines, reaching just under 100 GBps for 8192 bins. Increasing to the active threads, a speedup of approx. 2x is reached. Increasing to 4 active workgroups (128 threads) provides a 3x improvement, sometimes even slightly more, compared with the baseline. This biggest workgroup shows a similar drop in performance as the baseline, which is more than 50 %.

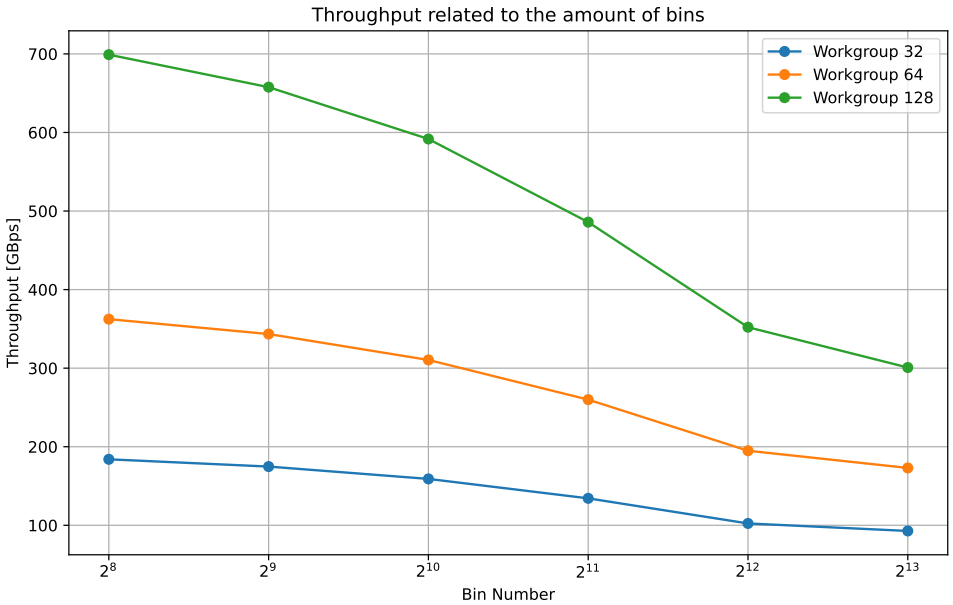

Figure 1: Figure 1

4.2 Bitonic sort

Question :
Answer:

- Why is there a `break` after `bitonicMergeShared()`?
- What happens if we omit the `break`, do we still get the correct results?
- Express the code more clearly without using a `break`?

Answer:

- Context: We find the `break` inside of the `else` branch inside of the nested loop, directly after the call to `bitonicMergeShared()`. The outer loop increases the size of the sorting block by powers of two, while the inner loop reduces the stride size by half in each iteration.

  When the stride becomes smaller than `SHARED_SIZE_LIMIT`, the `bitonicMergeShared()` kernel is called. This kernel performs the merge operation using shared memory. Once the merge is complete, there's no need to continue iterating through smaller strides, as the whole problem fitted into shared memory and is solved completely.

- Yes, we still get the correct results, but we do the same operation multiple times, wasting time and resources. Specifically, we get a throughput of 511 MElements/s instead of 1922 MElements/s (measured over 1k iterations on an Nvidia RTX 4080).

- Old code:

```
for (uint size = 2 * SHARED_SIZE_LIMIT; size <= arrayLength; size <<= 1)
    for (unsigned stride = size / 2; stride > 0; stride >>= 1)
        if (stride >= SHARED_SIZE_LIMIT) {
          bitonicMergeGlobal<<<(batchSize * arrayLength) / 512, 256>>>(
              d_DstKey, d_DstVal, d_DstKey, d_DstVal, arrayLength, size, stride,
              dir);
        } else {
          bitonicMergeShared<<<blockCount, threadCount>>>(
              d_DstKey, d_DstVal, d_DstKey, d_DstVal, arrayLength, size, dir);
          break;
        }
```

  New Code:

```
for (uint size = 2 * SHARED_SIZE_LIMIT; size <= arrayLength; size <<= 1) {
  bool sizeMerged = false;
  for (unsigned stride = (size / 2); ((stride > 0) && (!sizeMerged)); stride >>= 1) {
    if (stride >= SHARED_SIZE_LIMIT) {
      bitonicMergeGlobal<<<(batchSize * arrayLength) / 512, 256>>>(
          d_DstKey, d_DstVal, d_DstKey, d_DstVal, arrayLength, size, stride,
          dir);
    } else {
      bitonicMergeShared<<<blockCount, threadCount>>>(
          d_DstKey, d_DstVal, d_DstKey, d_DstVal, arrayLength, size, dir);
      sizeMerged = true;  // Set the flag to stop further iterations
    }
  }
}
```

4.3 Extra: scan with transpose access

Disclaimer: We tried our best but we could not fix the last issues.

Question :
Answer:

- Depending on warpNum, how many additions are performed in the original and in the new version?
- Does the new version produce bank conflicts, can we avoid them?
- What limits performance in the new version?

Answer:
The original kernel scanned 65536 elements in 13.45 ms, while the modified kernel took nearly the same performance with a slight better performance with 13.22 ms.

- Which Kernel Performs More Additions? For Small warpNum: The modified kernel may perform fewer additions for small warpNum due to the simplicity of row-wise summation.

  For Large warpNum: The original kernel is more efficient due to its logarithmic scaling in both warp-level and inter-warp scans, whereas the modified kernel scales quadratically with increasing warpNum.
- Yes, because we access the shared memory with a stride we can avoid them by transposing before the calculation. Another possiblilty is to maybe add padding.
- Major limit in our case is the quadratic scaling, more synchronisation needed and the underutilization of threads. Because our row-wise summation in step 2 currently uses only 1 thread. But we can accomplish that also with an warp row summation.