Performance Issue

- The first performance issue was that every thread in a block was calculating the sum of the elements present in that block. There is no need for each thread to do the computation. We can let only one of the threads to do the computation.
- The second issue is the sum variable which is constant for a block was not placed in a shared memory. All the threads are accessing the same variable sum. Hence, placing it in a shared memory increases our memory performance by reducing the memory bandwidth.
- All the threads in a block are performing on a unique data. i.e data in global memory is conceptually divided and given to blocks and there is no sort of interaction between the blocks. Threads in a particular block are performing computation over same set of data elements. This increases memory bandwidth because the data is stored in global memory. If the data is stored on shared memory we can reduce the memory bandwidth.

Optimization

- The first optimization deals with calculating the multiplication in[start + i * width + j] * mul[j] . This is done with the help of other threads in a block. The value of the multiplication is store in a 2D array name 'value'. This essentially gets rid of the nested for loop that every threads need to go through. This belongs to the category 4 which is reducing the number of instructions a thread needs to do. The expected performance for this optimization was that the execution time of the kernel would reduce by atleast 50%. The observed performance exceeded our expected performance. The original program had 0.016 s execution time on gtx780 and the optimized version had 0.005s as the kernel execution time.
- The second optimization deals with the calculating sum using nested for loop. In original program, every thread is calculating the sum of elements which are present in a block, this is wastage of computing power. Because the sum variable is constant to a particular block. So allowing only one thread to do the computation of sum will reduce the computation load on other threads. The rest of the threads can access sum variable by placing it in a shared memory. This optimization belongs to category 4 which is reducing the number of instructions a thread needs to do. We expected similar performance as in the first optimization. The observed performance matched our expected performance. The original program performed in 0.016s while the optimized version performed in 0.009s.
- The third optimization deals with the thread divergence. In the original program the we have if else ladder which was responsible for causing the thread divergence. This was solved by rewriting the code such that using the value of tx%2 and ty%2 we got a variable named factor with possible values being 0,1,-1 and 2. This factor was then multiplied the in[tx * width +ty] /sum and stored to out[tx * width + ty]. This makes all the threads in a warp do the same set of instructions thus eliminating the thread divergence issue. We did not expect to see huge improvement in the execution time. The observed running time of the optimized version only differed by 0.001s, reducing execution time from 0.016s to 0.015s on gtx780.

Experimentation

The optimization which deals with better data placement was also performed. We put array data[] and mul[] into the shared memory. As a standalone optimization it gave very good performance result. It reduced the kernel execution time on gtx780 to 0.004s. But when it was combined with other optimization it did not reduce the kernel execution time by huge factor. Possibly because in other optimization we reduced the work load on the threads and there by needing less access to the array data[].

Performance

File	GTX 480	GTX 780
norm	0.012472 s	0.016307 s
opt1.cu	0.007510 s	0.005655 s
opt2.cu	0.006591 s	0.009430 s
opt3.cu	0.011476 s	0.015523 s
optAll.cu	0.003141 s	0.002781 s

Experience on tool

The tool has been very useful in finding the different optimization technique. The optimization mentioned in the experimentation has been done based on the information provided by the advising tool.