Assignment 3

Project Description

In this assignment, a convolution neural network was implemented. This CNN has three layers – convolution layer, and two fully connected layers. The CNN takes an image of size 28x28 and tries to identify the number based on the trained weighted neural network. The convolution layer takes an image of 28x28 as input and performs convolution across 8 channels each of size 10x10. A bias is then applied and ReLu activation function is applied, which essentially removes all negative values. This is now reshaped in a channel first-row major format and provided as an input to the first fully connected layer. Here, first a matrix multiplication is performed with a weighted matrix and then a bias is applied. ReLu activation function is applied on this output and provided to the next fully connected layer. This layer further performs another matrix multiplication followed by a bias addition. The output of the CNN is a weighted probability matrix of 1x10, each value corresponding to [0,9]. The element of the result that has the maximum probability is the number detected by the CNN.

Implementation

- Optimization techniques
  1. Multiple kernels

     As each of these stages had different filter and input matrix sizes, the three layers were invoked as separate kernels so as to prevent wastage of threads and optimizing thread block sizes based on the needs of each of the operations.
  2. Constant memory

     The input image and other smaller biases and weights were stored in the constant memory as deemed appropriate. The idea is to keep most frequently accessed data structures within the constant memory to reduce timing.
  3. Loop unrolling

     Loop unrolling was performed using `#pragma unroll`. This provided with more concurrency among and within threads.
  4. Optimized thread block size

     For the first fully connected layer, for each of the matrix multiplication, the weight matrix is accessed only once by each thread for an output stationary approach. Thus, if multiple threads are accessing consecutive locations, there will be memory coalescing leading to improved execution times. The output matrix size in this case was 1x512. Thus, for an output stationary approach, a total of 512 threads will be required where each thread is responsible for computing a single output, and there won't be any element in the weight matrix that is being accessed twice across or even within threads. However, by keeping a single thread block of size 512 gave very high execution times. It was seen that at a thread block size of 32, which is also the warp size, the execution time was minimum. It was suspected that this was the case probably due to faster and more efficient memory coalescing within a warp and this in turn reduced the total number of global memory references. It is also suspected that there might be a cap to the total number of global memory requests sent by a single thread block, and at larger thread block sizes, the requests essentially get serialized, thereby increasing the execution times.

Result Analysis and Observations

The timings of all the cases discussed above have been tabulated below. The unoptimized version uses the constant memory, but doesn't include loop unrolling and optimized thread block sizes. Overall, it was observed that loop unrolling had a very small improvement, as can be seen for the convolution and fully connected 2 kernels. However, the thread block size optimization had a massive improvement for the fully connected 1 kernel. All times reported are in microseconds (us). The results below are demonstrated for images xaa, xab, …..xal. For all these images, the CNN passed in detecting the number. The trend in the net improvement is also illustrated in the graph.

Unoptimized (us)				Optimized (us)			
convolution	fully connected 1	fully connected 2	total	convolution	fully connected 1	fully connected 2	total
71.743	1047	18.72	1137.463	71.711	318.97	18.112	408.793
71.903	1046	18.656	1136.559	71.647	318.11	18.015	407.772
71.743	1045.3	18.528	1135.571	75.711	317.44	18.432	411.583
71.807	1043.4	18.751	1133.958	76.511	317.15	18.752	412.413
71.647	1045.2	18.432	1135.279	77.215	318.08	18.592	413.887
71.711	1046.3	18.496	1136.507	71.519	317.47	18.272	407.261
71.647	1046.7	18.463	1136.81	78.495	319.61	17.952	416.057
71.423	1039.8	18.848	1130.071	71.423	318.65	18.336	408.409
71.391	1039.2	18.528	1129.119	71.519	316.57	18.4	406.489
71.327	1038.4	19.2	1128.927	71.903	316.32	18.528	406.751
77.279	1036	18.016	1131.295	71.647	318.01	18.528	408.185
71.231	1041.2	18.496	1130.927	71.679	318.24	18.208	408.127