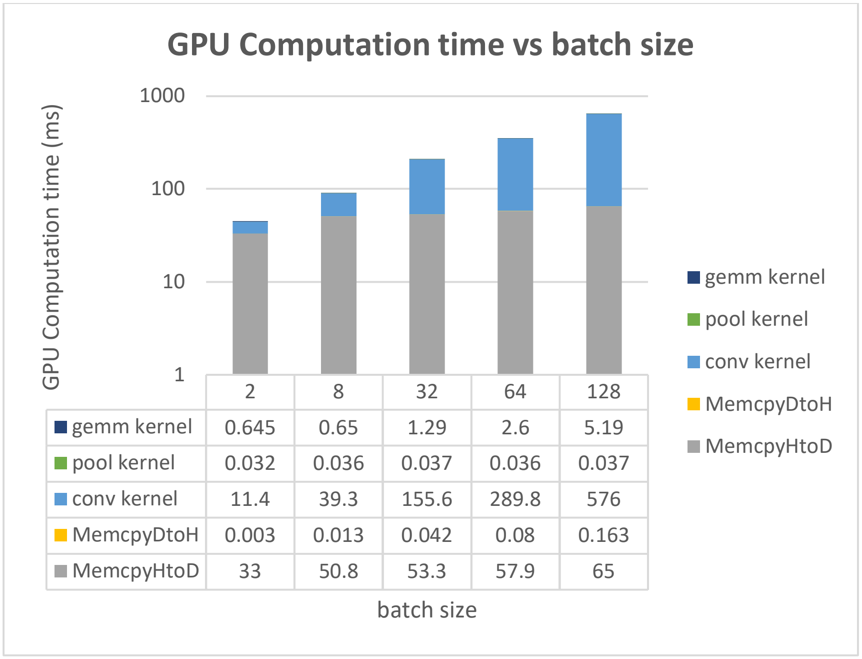
ECE695 Assignment: Part 4 Bonus (done by: Christin David Bose)

Design considerations

- The kernels for convolution, pooling and GEMM operations are reused from previous assignments and stitched together to run the whole AlexNet network on the GPU.
- As observed in part3, unmanaged memory seems to result in a more efficient kernel implementation in terms of speedups (at the expense of manual host-device memory allocation) and hence, cudamalloc(), cudamemcopy() functions are used to store and transfer the necessary filter and intermediate values. It is to be noted that intermediate activations are stored on the GPU and are not sent back to the CPU until the processing for all layers have been completed. This is to avoid unnecessary transfer of data which is time expensive.
- The verification of the layer outputs is done at a layer wise level and then integrated to run the whole network.
- Runtimes for batch sizes of 2, 8, 32, 64 and 128 are reported.

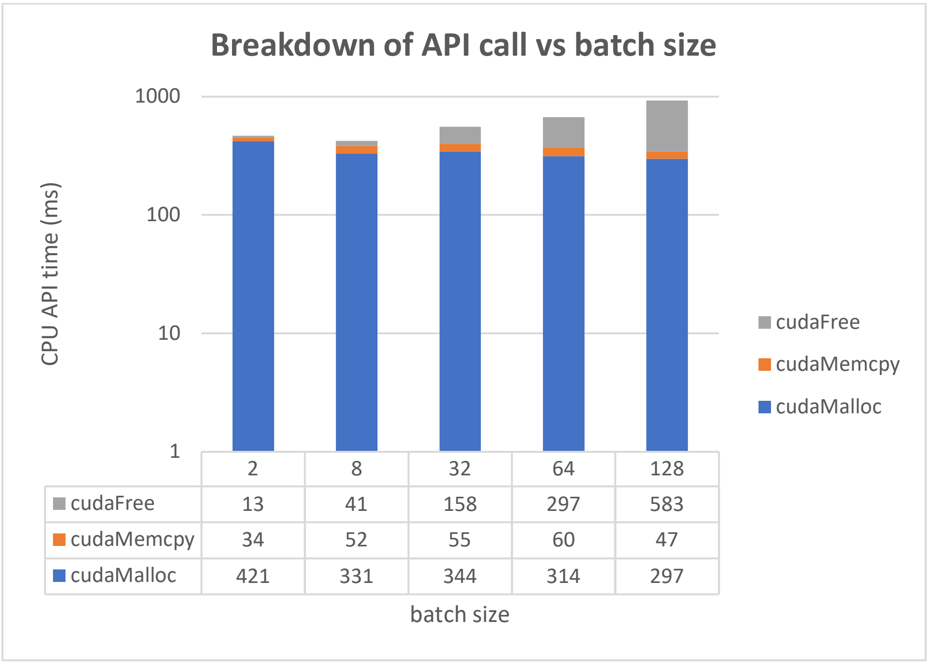
Figure below shows the computation time on GPU for end to end running of Alexnet.



	2	8	32	64	128
gemm kernel	0.645	0.65	1.29	2.6	5.19
pool kernel	0.032	0.036	0.037	0.036	0.037
conv kernel	11.4	39.3	155.6	289.8	576
MemcpyDtoH	0.003	0.013	0.042	0.08	0.163
MemcpyHtoD	33	50.8	53.3	57.9	65

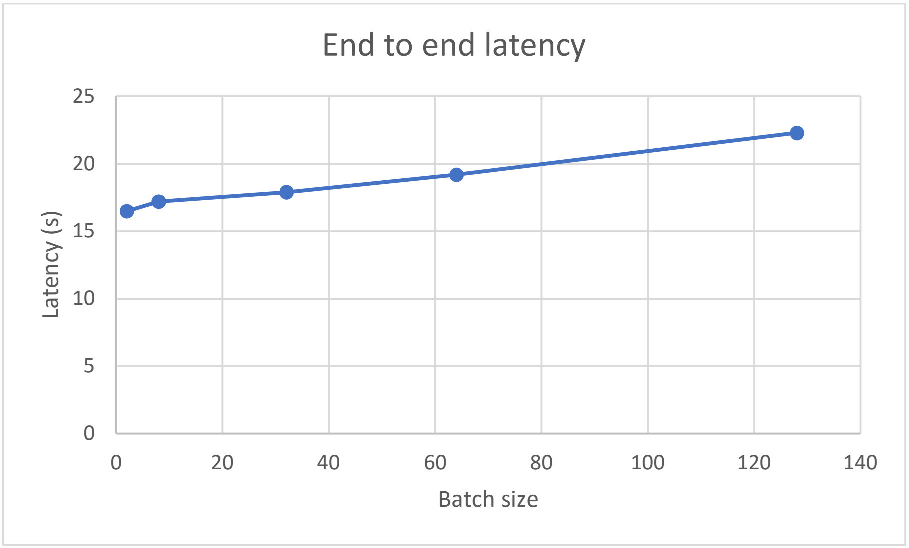
- It is observed that the time of the kernel is very small and much of the time is spent in the memcpy operation which is a function of the number of inputs and outputs of a layer.
- AlexNet has 5 convolution layers and 2 fully connected layers. As convolution layers are compute-bound, we see increasing time spent on conv layers as the batch size is increased. As the GEMM operation is memory bound, it’s less sensitive to batch size. For conv layers, the kernel execution time is nearly linear with respect to the batch size.

Figure below shows the CPU times of various API calls. The memory allocation on the cuda device is a significant portion of the total GPU compute time and remains the bottleneck for overall speedup.



	2	8	32	64	128
▨ cudaFree	13	41	158	297	583
🟧 cudaMemcpy	34	52	55	60	47
🟦 cudaMalloc	421	331	344	314	297

Figure below shows the overall compute time (end to end latency) for the GPU for various batch sizes. This is measured using the std::chrono::high_resolution_clock::now() function after removing any print statements.


End to end latency
Latency (s)
Batch size