408: Applied Parallel Programming

Fall 2018 – Midterm Exam 2

December 4th, 2018

1. This is a closed book exam except for 1 sheet of hand-written notes
2. You may not use any personal electronic devices except for a calculator
3. **Please write legibly!! We are using OCR to help grade your exam**
4. Absolutely no interaction between students is allowed
5. Illegible answers will likely be graded as incorrect

Good Luck!

Name:_____SOLUTION_____

NetID:_____

Exam Room:_____

Question 1 (20 points): _____

Question 2 (20 points): _____

Question 3 (15 points): _____

Question 4 (20 points): _____

Question 5 (20 points): _____

Question 6 (5 points): _____

Total Score (100 points): _____

Name: _____

Problem 1 (20 points): Multiple Choice

Choose the proper response, and if multiple responses are correct, choose all. No partial credit will be provided if the answer is partially correct, or wrong.

Part 1a (3 points) For the following reduction kernel fragment, if the block size is 512 and warp size is 32, how many warps in a block will have control divergence during the iteration where stride is equal to 64?

```
1.  __shared__ float partialSum[2 * blockDim.x];
2.  unsigned int t = threadIdx.x;
3.  unsigned int start = 2 * blockIdx.x * blockDim.x;
4.  partialSum[t] = input[start + t];
5.  partialSum[blockDim.x + t] = input[start + blockDim.x + t];
6.  for (unsigned int stride = 1; stride <= blockDim.x; stride *= 2)
7.  {
8.      __syncthreads();
9.      if (t % stride == 0)
10.         partialSum[2 * t] += partialSum[2 * t + stride];
11. }
```

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 8
- ☐ 16

Part 1b (3 points) For the Brent-Kung scan kernel based on reduction trees and inverse reduction trees, assume that we have 2048 elements in each section and warp size is 32, how many warps in each block will have control divergence during the reduction tree phase iteration where stride is 64?

- ☐ 0
- ☐ 1
- ☐ 16
- ☐ 32

Name: _____

Part 1c (3 points) Suppose we need to run Brent-Kung scan algorithm on a very large input consisting of 2^{30} elements. For our CUDA device, the maximum number of threads in a block is 2^{10} and the maximum number of blocks in the x-dimension of the grid is 2^{11}. Further, suppose that we are using a one-dimensional grid along the x dimension. If we choose to use hierarchical parallel scan to process the input, what is the minimum number of times we need to launch the scan kernel?

- ☐ 2
- ☐ $2^8 + 2$
- ☐ $2^9 + 1$
- ☐ $2^9 + 2$
- ☐ $2^{19} + 2$

Part 1d (3 points) Suppose a processor supports atomic operations in L2 cache, assume that each atomic operation takes 5ns to complete in L2 cache and 180ns to complete in DRAM. Assume that 80% of the atomic operations hit the L2 cache. Further, assume that the kernel performs 10 floating-point operations per atomic operation. What is the floating-point throughput of the kernel execution as limited by the throughput of the atomic operations?

- ☐ 0.0025 GFLOPS
- ☐ 0.054 GFLOPS
- ☐ 0.25 GFLOPS
- ☐ 18.52 GFLOPS

Part 1e (3 points) To transfer data from the device to host (or vice versa), we use cudaMemcpy which essentially requires a pointer to a data block and the size of that block. To transfer data from nodes in an MPI system, we use MPI_Send (and MPI_Recv). One key difference to CUDA is that MPI_Send requires a pointer to the data block, the number of elements in the block, and the datatype of these elements. Which of the following accounts for this difference?

- ☐ MPI_Send is non-blocking
- ☐ cudaMemcpy only works on floating point data
- ☐ MPI needs to work on heterogeneous systems with different endian-ness (byte ordering).
- ☐ MPI_Send can work on non-contiguous data blocks, whereas cudaMemcpy moves a single contiguous block

Name: _____

Part 1f (3 points) We need to calculate the histogram of an array with 10^9 elements. The histogram has four bins. Assume that each atomic operation in the global memory has a constant total latency of 100ns and each atomic operation in a shared memory has a constant total latency of 1ns. Further, assume that when we launch the kernel, blockDim = 10^3 and gridDim = 10^5, thus thread is responsible for 10 elements. If we only consider the latencies caused by atomic operations, what is the theoretical minimum runtime if privatization is implemented and the elements of the array has a distribution of (50%, 30%, 10%, 10%)? Suppose that there is only one global histogram and one shared memory histogram in each block.

- ☐ $(10^7 + 1000)$ ns
- ☐ $(10^7 + 5000)$ ns
- ☐ $(10^7 + 10^4)$ ns
- ☐ $(4 * 10^7 + 1000)$ ns
- ☐ $(4 * 10^7 + 5000)$ ns
- ☐ $(4 * 10^7 + 10^4)$ ns
- ☐ None of the above

Part 1g (2 points) Given a sparse matrix of integers with m rows, n non-zero elements in the row with the largest number of non-zeros, and k non-zeros. How many integers are needed to represent the matrix in JDS-T? Recall that JDS-T has a transposed representation. Remember that we need to keep track of the number of non-zeros in each row, to assist the SpMV calculation.

- ☐ 2m+1
- ☐ m+n+k
- ☐ 2k+m+n+1
- ☐ 2k+m+1

Solution:

1a. D, 8
1b. B, 1
1c. B, 2^8 + 2
1d. C, 0.25 GFLOPS
1e. C, need to work on systems with different endian-ness
1f. B, (10^7 + 5000) ns
1g. C, 2k+m+n+1

Name: _____

Problem 2 (20 points): Histogramming

For this question, we'll consider a histogram with 2^{18} bins (much larger than shared memory), and an input data stream (with values in the range $[0\ ,\ 2^{18} - 1]$) that needs to be binned. We want to take advantage of privatization, and observe that the input stream only has 256 bins that contain a significant fraction of the data (hot bins). Say we are provided a vector that's also 2^{18} in length like the histogram, but contains a -1 for bins that are infrequently used, and a unique number between 0 and 255 for each of the hot bins. Our algorithm uses this unique value to create a private histogram in shared memory.

The following kernel will use shared memory for the hot bins and global memory for the not-so-hot bins. Once the work for a thread block is completed, the histogram will be updated in global memory.

Part 2a (12 Points) Complete the following kernel to implement the hot bin histogram. Note that each thread block will have its own privatized histogram of the hot bins in the shared memory with name **`histo_private`**. The vector **`data`** contains the input data stream of length **`size`**, and the vector **`freq`** contains the hot bin vector.

For this problem, pay close attention to the array **`global_index`**. It serves as a mapping back from the private histogram bins in **`histo_private`** to the global histogram bins in **`histo`**.

Name: _____

```
 1  // histogram is launched with the following parameters.
 2  dim3 gridDim(1000);
 3  dim3 blockDim(256);
 4  histo_kernel<<<gridDim,blockDim>>>(data, freq, length, histo)
 5
 6  // Kernel Code
 7  __global__
 8  void histo_kernel(unsigned int *data, int *freq, long size, unsigned int *histo)
 9  {
10     __shared__ unsigned int histo_private[256];
11     __shared__ unsigned int global_index[256];
12
13     //reset histogram
14     histo_private[threadIdx.x] = 0;
15     global_index[threadIdx.x] = 0;
16
17     __syncthreads();
18     unsigned int i = threadIdx.x + blockIdx.x * blockDim.x;
19     unsigned int stride = blockDim.x * gridDim.x;

20     while(i < _________size__________){

22        if (__________freq[data[i]]____________ > 0) {

23           atomicAdd(&histo_private[_____________freq[data[i]]_________________],1);

24           global_index[___________freq[data[i]]____________] = data[i];
25        }
26        else

27           atomicAdd(*histo[___________data[i]_____________________], 1);
28        i += stride;
29     }
30     __syncthreads();
31     //contribute to global histogram

32     atomicAdd(_______&histo[global_index[threadIdx.x]]__, histo_private[threadIdx.x]);
33  }
```

Part 2b (4 Points) How many **shared-memory atomic operations and global-memory atomic operations** are being performed by all the threads in the kernel if the stream contains 10^7 data items and 80% of the elements are in the "hot" bins? Write down the expression for your answer. Pay attention to the code structure.

shared-memory atomics: ____0.8 * 10^7_____

global-memory atomics: _____0.2 * 10^7 + 256*1000____

Name: _____

Part 2c (4 Points) For this part, we will use a very simple performance model with the following assumptions:

- the input stream **`data`** contains 32 data items and 25 (approx 80%) of the elements are in the "hot" bins
- There is 1 thread block with 32 threads (1 warp) in this execution.
- Further assume each shared-memory atomic operation requires 1 ns, whereas each global-memory atomic operation requires 100 ns.
- All atomic operations are blocking, which means the next instruction cannot execution until the atomic operation is completed.
- All other operations require 0 ns.
- You may assume the full warp starts execution at time 0, and executes every cycle.

With these assumptions, what is the execution time in nanoseconds for this thread block?

25*1ns + 7* 100ns + 32*100ns
or
25*1ns + 7*100ns + 256*100ns

Your Answer Here:

Name: _____

Problem 3 (15 points): Multiple Prefix Sums

For this problem, we will implement a CUDA kernel for generating the prefix sums of multiple lists of data. Data from a biology experiment recorded the amount of growth of a plant specimen in millimeters in a given day for some number of plants. From this data, we want to generate the plant size in millimeters on each day for each plant. That is we need to perform a prefix sum for each plant. For this, we will devise a CUDA kernel based on the Brent-Kung approach for prefix sum. The table below provides an example of the input data for 3 plants across 4 days.

	Plant 0	Plant 1	Plant 2
Day 1	30	10	1
Day 2	20	5	5
Day 3	15	20	4
Day 4	17	4	6

The input data **`input`** is a matrix, with **num_plants** as width and **num_days** as height. The matrix will be stored as one-dimensional array in the row-major layout. We've used a grid with 2D thread blocks to solve this task. All threads in the same x dimension will process the data of one plant. Threads in the same x dimension in a block will process **2*BLOCK_SIZE** data.

Part 3a (3 points): Why does Brent-kung Prefix Sum algorithm not require double-buffering? Please answer the question in one line.

There is no read after write dependency among threads.

Your Answer Here:

Name: _____

Part 3b (12 points) Fill in the blanks to complete the prefix sum kernel described above.

```
 0   #define BLOCK_SIZE 32
 1   __global__
 2   void PlantScan(float *input, float *output,
 3                  int num_plants, int num_days)
 4   {
 5      __shared__ float partialSum[BLOCK_SIZE][BLOCK_SIZE*2];
 6      int tx = threadIdx.x;
 7      int ty = threadIdx.y;
 8      if (blockIdx.x * BLOCK_SIZE + tx < num_plants) {
 9         if (2 * blockIdx.y * BLOCK_SIZE + ty < num_days) {
10            partialSum[tx][ty] =
11                      input[(2 * blockIdx.y * BLOCK_SIZE + ty) *
12                      num_plants +  blockIdx.x * BLOCK_SIZE + tx];
13         }
14         else partialSum[tx][ty] = 0;
15         if ((2 * blockIdx.y + 1) * BLOCK_SIZE + ty < num_days) {
16            partialSum[tx][ty+BLOCK_SIZE] =
17                     input[((2 * blockIdx.y + 1) * BLOCK_SIZE + ty) *
18                     num_plants + blockIdx.x * BLOCK_SIZE + tx];
19         }
20         else partialSum[tx][ty] = 0;
21      }
22      else partialSum[tx][ty] = 0;
23
```

Name: _____

```
24      int stride = 1;
25      while(stride < 2 * BLOCK_SIZE) {
26         __syncthreads();

27         int index = ______(ty+1)*stride*2-1_____________;

28         if (index < 2 * BLOCK_SIZE && ______index-stride_____ >= 0)
29            partialSum[tx][index] +=

30                         partialSum[tx][______ index-stride_______];
31            stride = stride * 2;
32         }
33
34      stride = BLOCK_SIZE/2;
35      while(stride > 0){
36         __syncthreads();
37         int index = (tx + 1) * stride * 2 - 1;

38         if (_____index+stride_______ < 2 * BLOCK_SIZE) {

39            partialSum[tx][___ index+stride__]+=partialSum[tx][index];
40         }
41         stride = stride / 2;
42      }

43      _______syncthreads()_____________________;
44     ...
45   }
46
47   // Host Code
48   int main ()
49   {
50      ...
51      // Invoke Kernel Here
52      dim3 dimGrid(num_plants/BLOCK_WIDTH, num_days/BLOCK_WIDTH, 1);
53      dim3 dimBlock(BLOCK_WIDTH, BLOCK_WIDTH, 1);
54      PlantScan<<<dimGrid, dimBlock>>>
55               (input, output, num_plants, num_days);
56      ...
57   }
```

Name: _____

Problem 4 (20 points): Sparse Matrix Multiplication

This question tests your knowledge of sparse matrix representation and operation.
You are given a sparse matrix representation in CSR (Compressed Sparse Row) format:

Nonzero values: **`data`** `[2, 5, 3, 4, 1, 2, 2, 3]`
Column indices: **`col_index`** `[0, 3, 1, 2, 0, 1, 2, 3]`
Row pointers: **`row_ptr`** `[0, 2, 4, 4, 8]`

Part 4a (2 points): Write down the dense matrix (4 x 4) of the given CSR format

For the following parts, use * to represent zero element

Part 4b (2 points): Provide the COO representation of the same matrix:

COO:
Non zero values: **`data`** `[ ]`
Column indices: **`col_index`** `[ ]`
Row indices: **`row_index`** `[ ]`

Part 4c (2 points) Provide the JDS representation of the same matrix:

Non zero values: **`data`** `[ ]`
Column indices: **`col_index`** `[ ]`
Row pointer: **`row_ptr`** `[ ]`
Row indices: **`row_index`** `[ ]`

Part 4d (2 points) Provide the JDS_T representation of the same matrix:

Non zero values: **`data`** `[ ]`
Column indices: **`col_index`** `[ ]`
Column pointer: **`col_ptr`** `[ ]`
Row indices: **`row_index`** `[ ]`

Part 4e (2 points) For sparse matrix-vector multiply, what is the major drawback of the COO representation, and why don't the other representations suffer from the same limitation?

Name: _____

Part 4f (10 points) CSC (Compressed Sparse Column) format is similar to CSR, in which the non-zero values in column are stored continuously in the memory. CSC format of matrix is equivalent to the transpose of CSR format.

CSC Format Example:

$$A = \begin{bmatrix} 0 & 4 & 1 & 5 \\ 1 & 0 & 4 & 0 \\ 0 & 2 & 0 & 2 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

Nonzero value:	`data`	`[ 1, 3, 4, 2, 1, 4, 5, 2 ]`
Row indices:	`row_index`	`[ 1, 3, 0, 2, 0, 1, 0, 2 ]`
Column pointer:	`col_ptr`	`[ 0, 2, 4, 6, 8 ]`

Given the information above, please fill in blank for the SpMV_CSC kernel.
Note: data – input matrix; x – input vector; y – output vector

```
__global__ void SpMV_CSC (int num_cols, float *data, int
*col_ptr, int *row_index, float *x, float *y)
1. {
2.     int col = blockIdx.y*blockDim.y+threadIdx.y;
3.     if (col < num_cols) {

4.         int col_start = col_ptr[_______];

5.         int col_end = col_ptr[________];
6.         for (int elem = col_start; elem < col_end; elem++) {

7.             float dot = data[________]*x[_________];

8.             atomicAdd( &(y[_______________]), dot );
9.         }
10.    }
11. }
```

Answer:

A. $\begin{bmatrix} 2 & 0 & 0 & 5 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 2 & 2 & 3 \end{bmatrix}$

Name: _____

B. COO:

values: `data [ 2, 5, 3, 4, 1, 2, 2, 3]`
Column indices: `col_index [ 0, 3, 1, 2, 0, 1, 2, 3]`
Row indices: `row_index [ 0, 0, 1, 1, 3, 3 ,3, 3]`

C. JDS:

values: `data [ 1, 2, 2, 3, 2, 5, 3, 4 ]`
Column indices: `col_index [ 0, 1, 2, 3, 0, 3, 1, 2 ]`
Row pointer: `row_ptr [ 0, 4, 6, 8, 8 ]`
Row indices: `row_index [ 3, 0, 1, 2 ]`

D. JDS_T:

values: `data [ 1, 2, 3, 2, 5, 4, 2, 3 ]`
Column indices: `col_index [ 0, 0, 1, 1, 3, 2, 2, 3 ]`
Column pointer: `col_ptr [ 0, 3, 6, 7, 8 ]`
Row indices: `row_index [ 3, 0, 1, 2 ]`

E. Drawbacks of COO: Need atomic operation, therefore less efficient. Each thread process a portion of the data elements and use an atomic operation to accumulate result into output.

This is because the threads in SpMV kernel with COO format are no longer mapped to a particular row. COO comes with the cost of additional storage for the row_index array.

F.

```
1. __global__ void SpMV_CSC (int num_cols, float *data, int
   *col_ptr, int *row_index, float *x, float *y)
2. {
3.    int col = blockIdx.y*blockDim.y+threadIdx.y;
4.    if (col < num_cols){
5.       int col_start = col_ptr[_col__];
6.       int col_end = col_ptr[__col+1___];
7.       for (int elem = col_start; elem < col_end; elem++){
8.          float dot = data[__elem__] * x[__col__];
9.          atomicAdd(&(y[_row_index[elem]_]), dot);
10.           }
11.    }
12.
```

Name: _____

Problem 5 (20 points): Convolutional Neural Network (CNN)

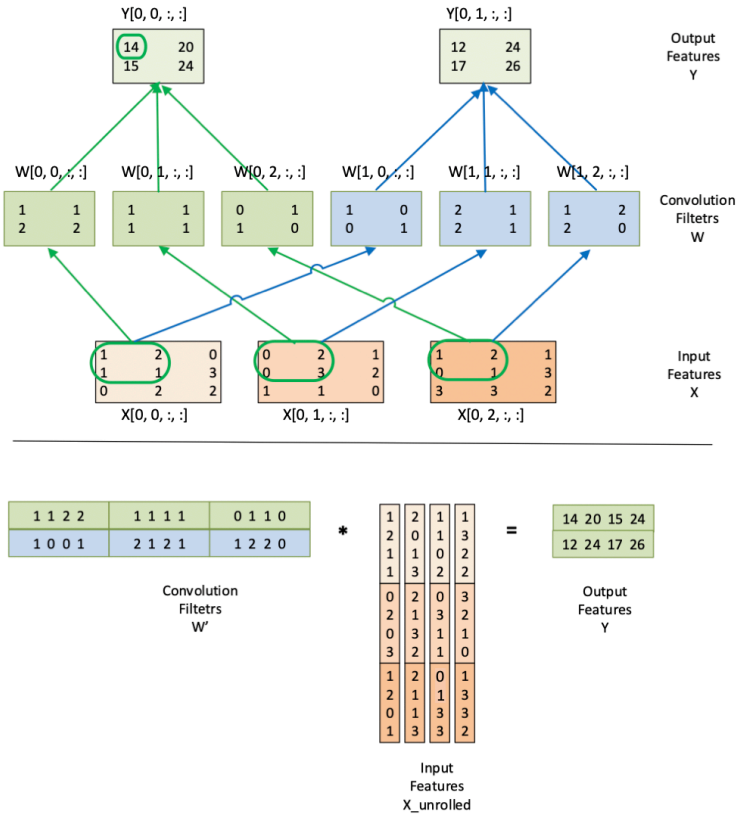
A basic convolution layer in a CNN consists of filter W, input X, and output Y. In this question, we want to accelerate the forward propagation of convolution layers in the training process.

W is the convolution filter weight tensor, organized as a tensor W[M, C, K, K], where M is the number of output feature maps, C is the number of input feature maps, K is the height and width of each filter. Tensors are stored as multi-dimensional arrays in the memory.

X is the input feature map, organized as a tensor X[B, C, H, W], where B is the number of images, H is the height of each input feature map, and W is the width of each input feature map.

Y is the output feature map, organized as a tensor Y[B, M, H_out, W_out], where H_out = H-K+1 is the height of each output feature map and W_out = W-K+1 is the width of each output feature map.

One way to implement the forward propagation in CUDA is to reduce the convolution into the general matrix multiplication (GEMM). The diagram below shows the relationship between basic convolution and how that can be done by using GEMM. Note that the diagram only shows a single mini-batch, e.g. batch 0.



Name: _____

Part 5a (2 points): From the lecture, we learned that before applying the GEMM, we first need to unroll the input feature map (X) into the correct shape. Your friend told you that you need to unroll the weight matrix (W) as well. Is this necessary? Why or why not?

Ans: No need for conversion. W is stored in the correct form for GEMM in the device memory.

Part 5b (2 points): How many times on average will be each X element be replicated after the unrolling? Provide your answer as an expression using tensor dimensions.

Ans: The size of the unrolled matrix will be C*K*K x H_out*W_out. The size of the input feature maps is C*(H_out+K-1)*(W_out+K-1). The ratio of the two gives the answer: K*K *(H_out*W_out)/((H_out+K-1)*(W_out+K-1).

Part 5c (16 points): After finishing the unrolling kernel and matrix multiplication kernel, you realized the performance could be better if we use only the tiled matrix multiplication kernel without actual unrolling. That is, instead of having a separate unrolling kernel, we perform unrolling when loading the tile into shared memory by correctly calculating the data indices. You need to fill in the missing parts so that the convolution layer is complete. (Note that we use multidimensional indexing notation for simplicity.)

```
// The code will be launched by using the following configuration.
dim3 gridDim(ceil(H_out*W_out/(1.0*TILE_WIDTH)),
             ceil(M/(1.0*TILE_WIDTH)),B);
dim3 blockDim(TILE_WIDTH,TILE_WIDTH,1);

// Kernel code.
01: __global__ void ConvLayerForward(int C, int K, int W_out, int H_out,
float* X, float* W, float* Y) {
02:    __shared__ float tileMatA[TILE_WIDTH][TILE_WIDTH];
03:    __shared__ float tileMatB[TILE_WIDTH][TILE_WIDTH];
04:
05:    int b = blockIdx.z;
06:
07:    int tx = threadIdx.x, ty = threadIdx.y;
08:    int row = blockIdx.y * TILE_WIDTH + ty;
09:    int column = blockIdx.x * TILE_WIDTH + tx;
10:    int numMatAColumns = C*K*K;  // This is the same as numMatBRows.
11:
12:    float acc = 0.0;
13:
14:    int num_iterations = ceil(numMatAColumns/(1.0*TILE_WIDTH));
15:
16:    for (int i = 0; i < num_iterations; i++) {
```

Name: _____

```
17:     int temp_col = i*TILE_WIDTH + tx, temp_row = i*TILE_WIDTH + ty;
18:     tileMatA[ty][tx] = 0;
19:     tileMatB[ty][tx] = 0;
20:
21:     // Original indices in the filter tensor.
22:     int W_m = row;
23:     int W_c = ___ temp_col/(K*K)_____;
24:     int W_h = ___(temp_col%(K*K))/K_____, W_w = _(temp_col%(K*K))%K____;
25:
26:     if (temp_col < numMatAColumns && row < M)
27:       tileMatA[ty][tx] = W[W_m, W_c, W_h, W_w];
28:     else
29:       tileMatA[ty][tx] = 0;
30:
31:     // Original indices in the input tensor.
32:     int X_b = b;
33:     int X_c = ___ temp_row/(K*K)__;
34:     int X_p = __ temp_row%(K*K)/K __, X_q = ___(temp_row%(K*K))%K___;
35:     int X_h = ___ column/W_out __, X_w = __ column%W_out___;
36:
37:     if (temp_row < numMatAColumns && column < H_out*W_out)
38:       tileMatB[ty][tx] = X[X_b, X_c, X_h + X_p, X_w + X_q];
39:     else
40:       tileMatB[ty][tx] = 0;
41:
42:     __syncthreads();
43:
44:     for (int q = 0; q < TILE_WIDTH; q++)
45:       acc += tileMatA[ty][q] * tileMatB[q][tx];
46:     __syncthreads();
47:   }
48:
49:   // Original indices in the output tensor.
50:   int Y_b = b;
51:   int Y_m = row;
52:   int Y_h = column / W_out, Y_w = column % W_out;
53:
54:   if (row < M && column < W_out*H_out)
55:     Y[Y_b, Y_m, Y_h, Y_w] = acc;
56: }
```

Name: _____

Problem 6 (5 points): Profiling

One fine Monday morning at Pied Piper, you get an email from Richard Hendricks! The email is as follows:

"Greetings from Richard Hendricks!

I learned that you have experience optimizing CUDA code through your ECE408 coursework! I would like to take your inputs to identify performance bottlenecks in our next generation machine learning model. Why don't you educate me as I am new to it

Regards
R.H."

After reading this email, you are delighted to know that you can educate Richard and you meet him in the afternoon. Richard has already done his homework and has some profiling data and it's your time to explain the bottlenecks in his code.

Part 6a (3 points) I have a code base with 1 kernel and 3 different optimizations. I profiled my kernel and saw the following in the profiler's visual analysis tool. See the charts on the following page. For each of the optimization, identify as specifically as possible the utilization limiting factor (i.e., memory bound, compute latency bound, resource bound).

a. Optimization 1: _____ Memory bound_____

b. Optimization 2: _____Compute latency bound_____

c. Optimization 3: _____Resource or latency bound_____

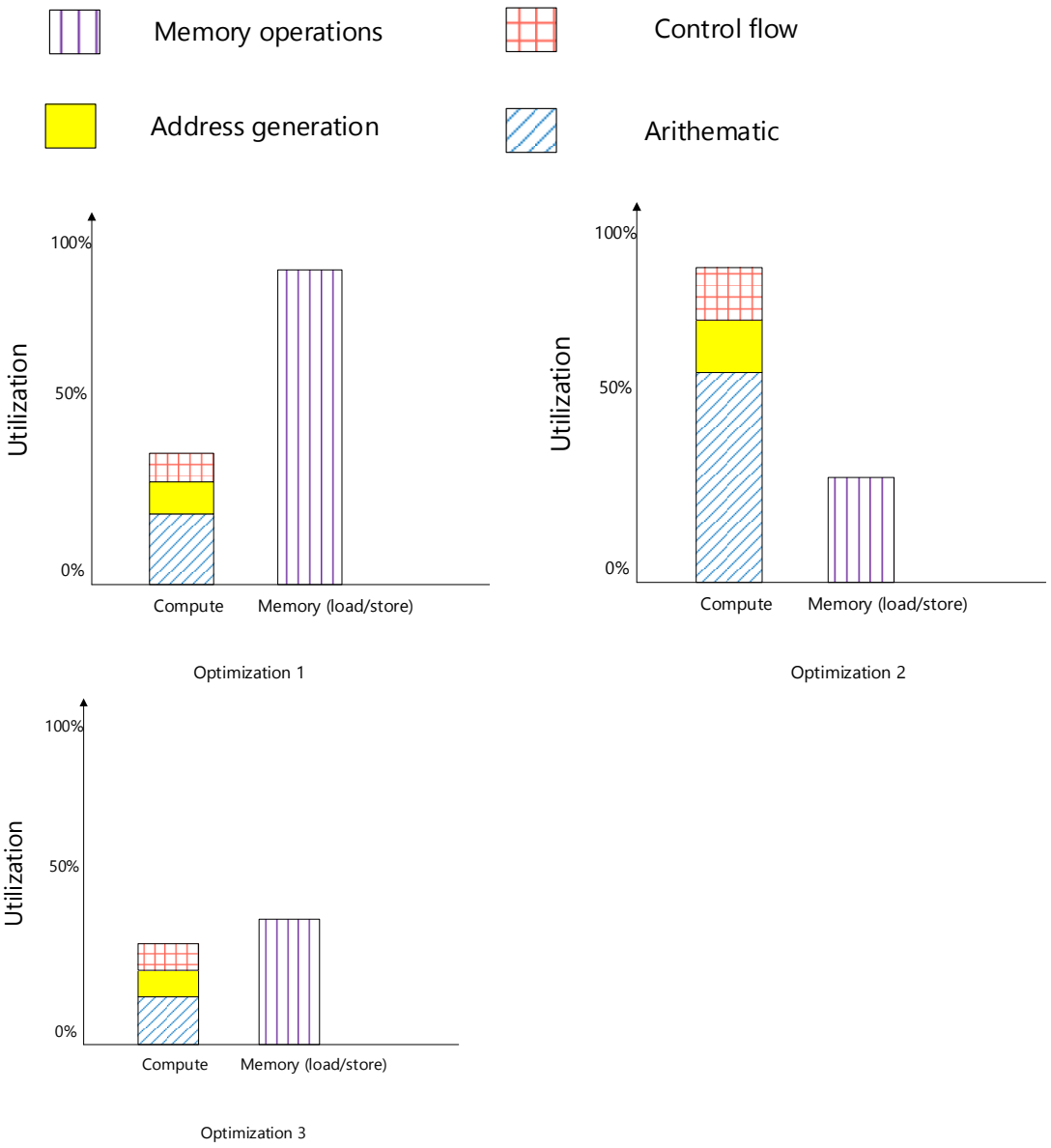
Part 6b (2 points) Richard asks you to guess the optimization that allowed him to improve his performance from optimization 1 to optimization 2. There are several major optimizations that could have achieved such results. Provide two below.

a. _____

b. _____

Possible answers:
i. Shared memory
ii. reduced the memory divergence or remove uncoalesced accesses
iii. register tiling or reducing the redundant or repeated usage of memory operations

Name: _____


Memory operations
Control flow
Address generation
Arithematic
100%
50%
0%
Utilization
Compute
Memory (load/store)
Optimization 1
Optimization 2
Optimization 3

Name: _____

This blank page is provided as extra space for calculations