**Udacity - Lesson 1 - DONE**

Modern GPU has

1. Thousands of ALUs
2. Hundreds of Processors
3. 10s of thousands of concurrent threads

Over time clock speed has stayed constant, but we continue to get more transistors as time goes on.

Power becomes a huge bottleneck and contraint as wqe continue to add more and improve on these.

CPU - Optyemized for latency

GPU - Optemized for throughput

Core GPU Design Tenents

1. Lots of simple compute units (trade control for compute)
2. Explicitly parallel programming model
3. Optimize for throughput, not latency

CPU = “Host”

GPU = “Device”

These two work together as ‘coprocessors’, with seperate memory in cuda.

cudaMemCpy = move data from CPU to GPU or the other way around

cudaMalloc = Allocate GPU memory

It is better to have a high amount of computation over communication. Leave the logistics to a minimum and do the most work you possibly can.

\*note - A kernel is the central part of an operating system. It manages the operations of the computer and the hardware

The GPU computation Big Idea:

Kernal looks like serial programs. Write your program as if it will run on one thread then tell it how many threads to run on. The GPU will run it on many threads.

The GPU is good at

1. launching a large number of threads efficiently
2. Running those threads in parallel.

Squaring numbers 0 - 63.

CPU - one thread looping through and doing a computation 64 times. 64 multiplications total.

For (int i = 0 ; i < 64; i++){ Array[i] = array[i] \* array[i}; }

// transfer the array to the GPU

cudaMemcpy(d\_in, h\_in, ARRAY\_BYTES,cudaMemcpyHostToDevice)

// copy back the result to the CPU

cudaMemcpy(h\_out, d\_out, ARRAY\_BYTES,cudaMemcpyDeviceToHost)

// free GPU memory allocation

cudaFree(d\_in)

cudaFree(d\_out)

// kernal call

square<<<1,64>>>(d\_in,d\_out)

Launching 1 block with 64 threads per block

There is a max number of threads a block can have (either 512 02m 1024)

Cuda supports multiple dimensions of blocks and threads. Both 3 dimensional.

dim3(x,y,z)

// Same as above using the 3 dimensional notation

square<<<dim3(1,1,1), sim3(64,1,1) >>>(d\_in,d\_out)

**Lesson 2 - Communication Patterns - DONE**

Map - 1:1 in to out

Gather - each thread gathers info from other places to get its value

Scatter - thread computes where to write its result to, still 1:1, (sorting)

Data reuse going on (over lapping pixels).

Transpose - row major order or column major order

Communication Patters

1. Map 1:1 operation on every element ex: \*2 or ^2
2. Scatter - where it needs to put the result is computed
3. Transpose 1:1
4. Stencil - several to 1 - ex: taking averag of many pixels as your stencil
5. Gather - reading from multiple locations on the input array to get its result
6. Reduce - all-to-one
7. scan / sort - all to all

Thread Index = threadIdx.y or threadIdx.j

Thread - a path/ thread of execution through the code - different threads can take different paths

Thread block - group of threads that work together

GPU is responsible for allocating blocks (blocks of threads) to run on SMs (the hardware)

Programer is responsible for defining the threads and shit

The programmer cannot specify…

* That blocks will run at the same time
* That block x will run after block y
* That block z will run on SM Q

Cuda makes very few guarantees about where thread blocks will run.

BAD THINGS

* Blocks cannot communicate = DEADLOCK
* Threads and blocks MUST complete. Or it is bad

Blocks can excecute on a different order every time. 16 blocks = 16! Possibilities = 21 trillion

Cuda does guarantee that

* Threads in a block run at the same time
* All blocks in a kernal finish before blocks from the next kernal run

Memory Statements - All True

* All threads from a block can access the same variable in **shared memory**
* Threads from 2 different blocks can access the same variable in **global memory**
* Threads from different blocks have their own copy of local variables in **local memory**
* Threads from the same block have their own copy of variables in local memory

Threads have to have SYNCHRONIZATION beacuse they can access the same shared and global memory

* Barrier - thread stops and waits for the other threads to catch up. \_\_syncthreads()

// Writing Efficient Programs

TFLOPS - floating point ooperations per second - modern can do 3trillion. Wow

Minimize time spent on memory

Local > Shared > Global > CPU memory (In general as far as time getting stuff from memory)

GPU is most efficient when read or write is on contiguous memory.

Coalesce → puts them all right next to each other. Because strided memory, where they are separate and spread out is inefficient.

Ex:

* G[i] = something - coalesced. Good and fast.
* G[i\*2] = strided, every other. Not good

Atomic Memory Operations - special memory things for a particular problem

atomicCAS() - compare and swap

atomicAdd(pointer to mem, value to add to that thing)

Atomic Mem Limitations

* Only certain data types supported (Mostly just integer stuf)
* Only certain operations supported (No %)
  + You can work around these using atomicCAS() sometimes
* No order constrains though. (This is good)
* Serializes memory access. Which is slow. Not good.

Also to be faster - avoid thread divergence.

**Lesson 3 - Fundamental GPU ALgorithms - NOT DONE**

Step Complexity - Tree Height

Work Complexity - Number of nodes

Reduce(set of elements, reduction operator)

Ex: Reduce(1 2 3, +) = 6

Reduce is dependent on the previous one being done.

For (int i = 0 ; i < 5; i++){

Sum = sum + i; // dependent on the previous sum

}

Step(height) Complexity of parallel reduction = log2(n)

1024 operations takes 10 steps in parallel, and 1023 in serial.

Inclusive Scan - Sum(all elements before it and itself)

Exclusive Scan - Sum(all elements before it)

Lesson 4 - Fundamental GPU Algorithms

Compact - Like Filter

Sparse or Dense Filter

1 2 3 4 5

Filter = keeps odds

1 null 3 null 5 = sparse, keeps same position

1 3 5 = dense, no gaps

WE WANT DENSE

Use compact when large number of elements and compute time is expensive. So we can save time.

Steps to Compact

* Predicate
* Scan in array, true or false
* Exclusive sum scan
* Scatter input into output( if true)

Segmented Scan - many small scans

* Resets at each |, then do it again

Sparse Matrix Multiplication Something = SpMv

CSR - compressed sparse row

30. Show the contents of the CSR (Compressed Sparse Row) format for the following 5x5 matrix:  
02300  
10050  
00400   
00020

We have

Value: [2, 3, 1, 5, 4, 2]

Column: [1, 2, 0, 3, 2, 3] -- good up to this point

rowPtr: [0, 2, 4, 5] -- how TF - NEVERMIND GOT THIS ONE

Segemented Rep’n : [2 3 | 1 5 |4 | 2] --

SORTS

Brick sort

[5 1][4 2] 3

1 [5 2] [4 3]

[1 2] [5 3] 4

…

etc

CS6068 Fall 2016 Midterm Practice Exam

1.What are three traditional ways that hardware designers make computers run faster? Please circle all that are true.  
-faster clocks  
-longer clock periods  
-more work per clock cycle  
-a larger hard disk  
-adding more processors   
-reducing the amount of memory

2.What are David Patterson's Three Walls of Computer Architecture? - DONE

Power Wall + Memory Wall + ILP Wall = Brick Wall

Power - Power is expensive. Now we have more than enough transistors to put on a chip but don’t have a good way to turn them on.

Memory - memory is slow

ILP - Instruction Level Parallelism means executing multiple instructions or pieces of instructions at the same time to make the computer run faster. Computers have hit the parallelism wall.

3. For a class of GPUs, what is appropriate measure to compare power consumption of different devices?  
  
4.What techniques are computer designers today using to build more power-efficient devices? Please circle all that are true.  
-having fewer, but more complex processors,  
-having more, but less complex processors,  
-maximizing the speed of the processor clock   
-increasing the complexity of the control hardware  
  
5. Name four common patterns of parallel computation?   
For each pattern, discuss whether the many-core or multi-core architecture model is more appropriate.

Pipe and Filter

* imagine graph with edges and vertices
* vertices are filters for computation
* edges are pipes for communication

Map-Reduce

* The same function is independently applied to everything

Agents and Repositories

* Solves the problem where data is modified at irregular times by a flexible set of operations.
* By having a centrally managed data repository
* And a manager that schedules the agents to make the changes
* Gives the whole thing more consistency

Iterative Refinement

6. What is concurrency control, and discuss what is the difference between pessimistic an optimistic concurrency control?  
  
7. How does Fork/Join parallelism differ from Kernel data-parallelism?  
  
8. Circle all the true statements.  
-A thread block contains many threads.  
-An SM might run more than one thread block.  
-A thread block may run on more than one SM.  
-All the threads in a thread block might cooperate to solve a subproblem.   
-All the threads that run on a given SM may cooperate to solve a subproblem.  
  
9. If we have a single kernel that is launched on many thread blocks, including block x and block y, the programmer can do which of the following: Circle all the true statements.  
-specify that block x will run at the same time as block y  
-specify that block x will run after block y.  
-specify that block x will run on same SM as y   
-none of the above  
  
10. Circle all the statements that are true.  
-All threads from a block can access the same variable in that block's shared memory.   
-Threads from two different blocks can access the same variable in global memory   
-Threads from different blocks have their own copy of local variables in local memory.   
-Threads from the same block have their own copy of local variables in local memory.  
  
11. Does a \_\_syncthreads() call apply to threads within a block or threads within a grid?  
  
12. What is a parallel map operation? Circle all problems that can be solved using map.  
-sort an array   
-add one to each element of an array   
-summing all elements in array  
-apply a predicate to each element in an array   
-move data in parallel based on array of scatter addresses  
  
13. Circle which operators are both binary and associative and therefore can be used in a reduction or scan.   
-multiply -minimum -factorial -exclusive or -bitwise and -exponentiation -integer division  
  
14. Using 1D global indexing, how would you specify the parallel execution mapping the ith thread to the task of reading and then squaring the ith item from an large array X in global memory.  
  
15.Circle all statements that are true. When running reduction code running on an input of size n?  
-it takes at least n operations  
-its work complexity is order of n  
-its work complexity is order n\*n  
-its step-complexity is order of 1, independent of the size of the input.  
  
16. Circle the correct answer. The number of steps in a reduction of n elements as a function of n is:   
- square root of n  
- log base 2 of n  
- n  
- n times log base 2 of n  
  
17. Circle all that are true  
- map operations have arguments that are functions with a single argument   
- map operations can be applied to arrays of any number of dimensions  
- map operations are generally very efficient on GPUs  
- a compact operation requires a map operation to be performed.  
  
18. What is the impact of granularity on performance when considering the latency of global memory communication?   
Calculate the granularity of dot product of two vectors that reside in global memory?  
  
  
19. What is the output of a max scan operation on the list of unsigned ints [5 4 7 3 1 8 2 6]?   
Provide a solution to both inclusive and exclusive scans.  
  
20. Compute the max (inclusive) scan of this input sequence 2 1 4 3 showing all work when using  
a) the Hillis-Steele algorithm  
b) the Blelloch algorithm  
  
21. Explain which scan algorithm (Hillis-Steele or Blelloch algorithm) is best suited and why?  
You are scanning a 512 element vector and a GPU that has 512 processors.  
  
22. Explain which scan (Hillis-Steele or Blelloch algorithm) is best suited and why?   
You are scanning a 1 million element input vector in 512 processors machine.  
  
23. Suppose you are computing a histogram with 10 bins. Discuss an efficient GPU solution.   
Suppose you are computing a histogram with 10000 bins. Discuss an efficient GPU solution.   
Indicate for each whether or not you are using atomics to manage access to bins of the histograms.  
  
24. True or false - In a scatter operation a \_\_syncthreads() command is needed to overcome write conflicts.  
  
25. Is the compact parallel operation more useful in scenarios where we delete a (small number) or a (large number) of elements from an array?  
  
26. Is the compact parallel operation more useful in scenarios where we need to run (cheap) or (expensive) function on filtered elements.  
  
27. Suppose we are running compact operations on a list of numbers with range from 1 to 1 million. Compact operation A, filters elements that are divisible by 17, and thus is only going to keep a very few of the input items. Compact operation B filters elements not divisible by 31, and thus is going to keep most of the input items. For each of the three phases of compact: predicate map, scan, and scatter phases of the compact operation, will A run faster, B run faster or will they take the same amount of time?  
a. Predicate map b. Scan c. Scatter  
  
28. What is the difference between latency and bandwidth? Which is more easily enhanced by advanced architectures?   
Say you are give a task of transporting many people 200 miles. You have two modes of transport:  
a single sports-car traveling 100mph with 2 passengers -- and -- a single van traveling 50mph with 10 passengers.   
For each case, what is the Bandwidth in people delivered per hour?  
For each case, what is the Latency of a person delivery (in hours)?  
For each case, what is the Occupancy of transport system?  
  
29. How is Occupancy defined in Cuda, and why does it have an impact on performance?  
What is thread divergence in Cuda, and why does it have an impact on performance?  
  
30. Show the contents of the CSR (Compressed Sparse Row) format for the following 5x5 matrix:  
02300  
10050  
00400   
00020  
  
31. Consider the sparse matrix dense vector product problem, and the two different parallel methods tpr(thread per row) or tpe (thread per element).  
a. Which approach will launch more threads?  
b. Which approach will require more communication between threads?  
c. Which approach will do more work per thread?   
d. Which approach is more load balanced?  
  
  
32. What does it mean for a sorting algorithm to be oblivious?  
  
33. State the 0/1 sorting lemma for oblivious sorting algorithms.  
  
34. Provide the logic of BitonicSort – pseudocode is sufficient here.   
  
35. What is the step and the work complexity of BitonicSort?  
  
36. In the BitonicSort figure presented in lecture identify each of the 'bitonic-comparison-and-swap' modules.   
  
37. What is the work and step complexity of countingSort?  
  
38. What is the expected work when hashing using chaining when the hash is to a chain of length k.  
  
39. True or false: Bloom filters are a data structure that allows fast set membership operations, but with low probability of false negatives.  
  
40. List the following in the order of their work complexity from least to most.   
a. parallel compact b. parallel scan c. sieveEratosthenes d. dense n-body  
e. bitonicSort f. sequential mergeSort  
  
41. List the following in the order of their step complexity from least to most.   
a. parallel compact b. parallel scan c. sieveEratosthenes d. dense n-body  
e. bitonicSort f. sequential mergeSort  
  
42. Write a CUDA kernel function that will effectively parallelize the following sequential function.  
void serial (int n, float a, float b, float \* x, float \* y) {  
for (int i = 0; i < n; ++i) {  
 y[i]= a \* x[i] + b \* y[i]; } }  
  
43. What does the following kernel code do? Does it contain a race condition problem? If so, give a means to overcome it?  
\_\_global\_\_ void naiveHisto(int \*d\_bins, const int \*d\_in, const int BIN\_COUNT)  
{  
 int myId = threadIdx.x + blockDim.x \* blockIdx.x;  
 int myItem = d\_in[myId];  
 int myBin = myItem % BIN\_COUNT;  
 d\_bins[myBin]++;  
}  
  
44. Complete the CUDA kernel function that computes, per-block, the sum of a block-sized portion of the input using a block-wide reduction.  
You should assume 1-dimensional thread and block indexing.   
\_\_global\_\_ void block\_sum(const float \*input,  
 float \*per\_block\_results,  
 const size\_t n)  
{  
 \_\_shared\_\_ float sdata[];  
 // TODO: load input into \_\_shared\_\_ memory   
 // TODO: use contiguous range pattern for reduction  
   
  
 // thread 0 writes the final result  
 if(threadIdx.x == 0)  
 { per\_block\_results[blockIdx.x] = sdata[0]; }  
}