1.What are three traditional ways that hardware designers make computers run faster? -faster clocks-more work per clock cycle - adding more processors  
2.What are David Patterson's Three Walls of Computer Architecture? 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? gigaflops/watt.  
4.What techniques are computer designers today using to build more power-efficient devices? Please circle all that are true.  
-having more, but less complex processors; maximizing the speed of the processor clock; trade control for compute power.

5. Name four common patterns of parallel computation?   
For each pattern, discuss whether the many-core or multi-core architecture model is more appropriate.

These are the 4 structural patterns. But i'm not sure if that is what is being asked

1. Pipe and Filter - imagine graph with edges and vertices - vertices are filters for computation -edges are pipes for communication MULTI
2. Map-Reduce - The same function is independently applied to everything - MANY
3. 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 - Many
4. Iterative Refinement - MULTI -you will need for them to talk to each other

There are also 4 implementation strategy patterns

1. Fork/ Join - Create threads/ split off == fork - Join back together after done == join
2. Data/ kernel Pattern - an index space is defined onto which the data structures
3. Loop level parallelism - The problem is expressed in terms of a modest number of compute intensive loops. The loop iterations can be transformed so they can safely execute independently
4. Single Program - Multiple Data - A popular pattern where a process/thread ID is used to index data

6. What is concurrency control, and discuss what is the difference between pessimistic and optimistic concurrency control? -DONE

Concurrency control is how conflicts are addressed when simultaneous access or alteration of data occurs.

1. Pessimistic Approach - assumes the worst - Records are locked right after they are used
2. Optimistic approach - assumes multi access is unlikely - Does not lock during use - But checks if it has been updated by multiple threads and only keeps one

7. How does Fork/Join parallelism differ from Kernel data-parallelism? - DONE

Fork - Join - execute then branch off in parallel, to later join back together

Kernel data parallelism is where you have the grid,block structure and specify these things from a program level. You don’t just start with one thread and branch off subthreads, you allocate threads and thread blocks you are to use and use those threads and thread blocks to each do a specific task in a kernel -Kernel Data Parallelism is what we use - BEtter for many simple processors

8. Circle all the true statements. - DONE  
-A thread block contains many threads. - TRUE  
-An SM might run more than one thread block. Think this is true  
-A thread block may run on more than one SM. nah  
-All the threads in a thread block might cooperate to solve a subproblem. - True  
-All the threads that run on a given SM may cooperate to solve a subproblem. - True  
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. - DONE  
-specify that block x will run at the same time as block y CANT BE CERTAIN - THIS IS A NO  
-specify that block x will run after block y. NO, YOU CANT MAKE THIS GUARANTEE  
-specify that block x will run on same SM as y I dont think the programmer makes this decision  
-none of the above - this one is none of the above, it was in the videos.  
10. Circle all the statements that are true. - DONE - was also in videos  
-All threads from a block can access the same variable in that block's shared memory. - TRUE  
-Threads from two different blocks can access the same variable in global memory - TRUE  
-Threads from different blocks have their own copy of local variables in local memory. - TRUE  
-Threads from the same block have their own copy of local variables in local memory. - TRUE  
11. Does a \_\_syncthreads() call apply to threads within a block or threads within a grid? -Block ; it is a block level command  
12. What is a parallel map operation? Circle all problems that can be solved using map. - DONE  
-sort an array - no, scatter is for sorting  
-add one to each element of an array - yes, map would be good for this  
-summing all elements in array - no, reduce is for all elements going to one output  
-apply a predicate to each element in an array - yes, map would be good  
-move data in parallel based on array of scatter addresses -

This is asking about a Gather-Scatter operation I think

A Map is when an operation is applied to all elements. 1:1 ratio of inputs to outputs  
13. Circle which operators are both binary and associative and therefore can be used in a reduction or scan.

What is binary?? But wtf is binary

What is associative?? a\*b = b\*a OK  
-multiply -minimum -factorial -exclusive or -bitwise and -exponentiation -integer division

Binary : our op. Is between to inputs such that op(a,b)

Associative: op(op(a,b), c) = op (op(a,c), b)

These operators are: max, min, +, \*, &&, ||   
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.

For a 1D array, use our threadIdx.x as our index then accessing and squaring that then taking that output and writing it like: globalOutputArray[threadIdx.x] = mySquareResult

If we have more than the number of threads per block elements, things get more complicated though, then you need to use block Indexing

You would want to populate shared memory with the initial array so we can cut down the global Memory accesses

sharedThing[threadIdx.x] = globalIInputArr[threadIdx.x];

globalOutputArray[threadIdx.x] = sharedThing[threadIdx.x] \* sharedThing[threadIdx.x]

15.Circle all statements that are true. When running reduction code running on an input of size n?  
-it takes at least n operations - false, 4 inputs = 3 operations. And so on  
-its work complexity is order of n - This is true, O(n-1) = O(n)   
-its work complexity is order n\*n - no, I still think it is N -1  
-its step-complexity is order of 1, independent of the size of the input. - false it is dependent on inputs - it is O(Log2(N)) - this is for sure right though  
16. Circle the correct answer. The number of steps in a reduction of n elements as a function of n is: log base 2 of n  
17. True statements - 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.

Requires a Scan, Exclusive Sum Scan, and Scatter. Never mind, this is like the filter thing.  
18. What is the impact of granularity on performance when considering the latency of global memory communication? - You want high computation/memory access, higher the granularity, the lower the latency  
Calculate the granularity of dot product of two vectors that reside in global memory?

Granularity = work done / memory access - For a dot product of two vectors of size n, you would do n multiplications and n additions for 2n memory reads, which would equal a granularity of 1. With higher granularity you can get better results because memory reads take forever.

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.  
I think the trivial scan we start with 1 element all the way up to our nth element in parallel, we would run n (size of the list) different scans if we don’t do any \_\_syncthreads calls

// inclusive

5 4 7 3 1 8 2 6 start

5 4 7 3 1 8 2 6

5 5 7 3 1 8 2 6

5 5 7 3 1 8 2 6

5 5 7 7 1 8 2 6

5 5 7 7 7 8 2 6

…

5 5 7 7 7 8 8 8 = 8 is max?

// exclusive - not including the Jth element

5 4 7 3 1 8 2 start

5 4 7 3 1 8 2

5 5 7 3 1 8 2

5 5 7 3 1 8 2

5 5 7 7 1 8 2

5 5 7 7 7 8 2

…

5 5 7 7 7 8 8 = 8 is max? AGAIN

20. Compute the max (inclusive) scan of this input sequence 2 1 4 3 showing all work when using

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

Hillis steele

2, 1, 4, 3 original

2, 2+ 1 = 3, 1+ 4 = 5, 4 + 3 = 7 , first step add one away, first one drops down

2, 3, 5, 7 end first step

2, 3, 2+ 5 = 7, 3 + 7 = 10 , add 2 away, first 2 drop down again

2, 3, 7, 10 end 2nd step

= 10

Blelloch

2 1 4 3

3 7

10

3 0 = identity operator

2 0 copy 4 3

0 2 3 7

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. - DONE

Hillis Steele - More Step Efficient - Use for more processors than work. Work = nlog(n) Steps = log(n) - 512 element vector and a GPU that has 512 processors

Blelloch - More work efficient - More work than processors Work = 2n Steps = 2log(n) - scanning a 1 million element input vector in 512 processors  
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.

If we have a histogram with 10 bins, it is wise to probably make a lot of local histograms then reduce the bins in parallel, we will likely not be using atomics here because of all the conflict issues. If we have tons of bins, we will not have a lot of conflicts while using atomics so we can probably use the more simple implementation of just incrementing bins using atomics

24. True or false - In a scatter operation a \_\_syncthreads() command is needed to overcome write conflicts.

In a scatter operation, we will usually be scattering operations to many locations meaning that many elements will be written to multiple times. If we are not using atomics, we are going to have issues and will probably have race conditions that evolve, I don’t think \_\_syncthreads() is useful for this situation and atomics is a better option. For example, when you do something like averaging pixel values from surrounding pixels, you are really only using an increment operation therefore since it is associative syncing all of our threads in a thread block would not really do anything.

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?

It’s obviously if we filter out a lot of elements and ideally these operations are pretty expensive  
26. Is the compact parallel operation more useful in scenarios where we need to run (cheap) or (expensive) function on filtered elements.

Expensive, therefore we are doing a good amount of work and saving us a lot of time compared to a serial algorithm

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

In the Predicate stage, we need to scan N elements to see if the result is true or false. Therefore, A and B will be the same

In the Scan stage, we need to scan all of the booleans from the predicate stage to set up the scatter stage, this will also take N for both

For the scatter operation, we are taking for each true element from the predicate, scatter that input element to some address found from the exclusive sum scan. We will be looking at less addresses for A because most of them will be filtered out by the predicate so

A will run faster

28. What is the difference between latency and bandwidth? Which is more easily enhanced by advanced architectures? - DONE  
Say you are give a task of transporting 20 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?

* Bandwidth = capacity
* Latency = speed
* which is easily enhanced by advanced architechtures? Probably bandwith - allows more to happen at once. Where latency would just allow each individual thing to happen faster
* Latency for sports car = 200m/100mph = 2
* Latency for bus = 200m/50mph = 4
* Bandwidth for sports car = 2people/ 2 hours = 1
* Bandwidth for bus = 10people/ 4 hours = 2.5
* So bus would be better because its getting 2.5 per same time the sports car is getting 1.

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? - DONE

Occupancy - Detailed on Lesson 1 of Udacity

* Limited number of threads/ blocks per thread/ shared memory, etc on a gpu
* You can be limited by max threads/ SM, max registers/ SM, max shared memory SM, or max thread blocks/ SM.
  + Max threads running = 1024
  + Max threads available = 1536
    - Occupancy = 66%
    - Occupancy = number of threads that are actually running vs the number that could be running.
* Impact on performance - You could have plenty of shared mem, plenty of blocks etc, but if max threads per block is 2, the rest of the resources go unused.
* Thread divergence -
  + Warp - set of threads that execute an instruction at same time frame - note a warp has 32 threads
  + Thread divergence = when they are suppose to be going together but they don’t, they diverge. When some threads are going, others are waiting idle, because all threads in a warp have to run together. So this wastes time. WARPS ARE DUMB.
  + This make thread divergence take longer. Bad for performance.

| = thread , here they are all running normally

||||||||| = warp 1

||||||||| = warp 2

||||||||| = warp 3

= 3 steps

Here, threads diverge so it takes an extra step

||||||||| = warp 1

|| || = warp 2 diverged!

|| ||| |= warp 2 diverged and is finishing! Still running all its threads. But not same time

||||||||| = warp 3

= 4 steps

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] --

Gather Vector Values using Column then multiply them out (pairwise) and reduce each segment

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? TPE  
b. Which approach will require more communication between threads? TPE  
c. Which approach will do more work per thread? TPR  
d. Which approach is more load balanced? TPE  
  
32. What does it mean for a sorting algorithm to be oblivious? - work the same no matter the input  
33. State the 0/1 sorting lemma for oblivious sorting algorithms.

Start with the least sig bit, split input into 2 parts based on if the bit is a 0 or a 1. Continue to the next most significant bit until we are at the mostSB then finish - If a sorting network sorts every sequence of 0's and 1's, then it sorts every arbitrary sequence of values.  
  
34. Provide the logic of BitonicSort – pseudocode is sufficient here.

Take some list/array

1. Take pairs of 2 in the array, do bitonic sort on each of the 2 element subsets. Take groups in the array and do bitonic sort on that and keep going 2^k until we have one large bitonic sequence
2. Take your increasing sequence and decreasing sequence, compare the first element of decreasing and first element of increasing and move all the larger elements down (swap), take the two subsequence you make and repeat until again you are all the way down to comparing 2 element subsets

35. What is the step and the work complexity of BitonicSort?

W(n) = nlog^2(n)

S(n) = log^2(n)

2 7 3 5 4 6 1 8 = start

2 7 5 3 46 8 1 = get two halves in bitonic order

2 3 5 7 8 6 4 1 = get whole thing in bitonic order

Then sort

2 3 4 1 8 6 5 7 = compare 4 away

2 1 4 3 5 6 8 7= compare two away

1 2 3 4 5 6 7 8 = compare one away  
37. What is the work and step complexity of countingSort?  
S(n) = k

W(n) = O(kn)

where k = number of bits, n is the size of the  
38. What is the expected work when hashing using chaining when the hash is to a chain of length k.

O(kn) i’m pretty sure for n elements, we are just using constantthink it’s a reduce time to put it into the table and k to insert it into the chain  
39. True or false: Bloom filters are a data structure that allows fast set membership operations, but with low probability of false negatives.  
False negatives are generally not possible. This is True

40. List the following in the order of their work complexity from least to most.   
a. parallel compact (essentially filtering) - work complexity = N??

Get rid of a large number of elements (filter them out)

Because there compute is expensive.

b. parallel scan - work complexity = N, step complexity = Log2(N)

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]; } }

Int tid = threadIdx.x;

If (n > tid)

Y[tid] = a \* x[tid] + y[tid];  
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]++;  
}

Yeah it does, you need to do an atomic add, the increment operator is not atomic so we will have read a stale value because we are not using atomics then save the incremented stale value in memory multiple times

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[];

sdata[blockIdx.x] = input[blockIdx.x];

For (unsigned int s = blockDim.x / 2; s > 0; s >>=1)

{

If (threadIdx.x < s)

sdata[blockIdx.x] = sdata[blockIdx.x] + sdata[blockIdx.x + s];

\_\_syncthreads();

}  
 // 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]; }  
}

////

square(in, out){

Index = blockIdx.x \* blockDim.x + threadIdx.x;

Out[index] = in[index]\*in[index];

}