CS 441 Midterm Exam

0.) NAME: Parker Whaley
2015-03-03. Your answers should be yours alone, after consulting with any written or online reference material. References that substantially influence your answer should be cited. Do NOT consult with any living sources, including professors or fellow students, and not even via the Internet.

1.) We've looked at a number of different parallelism technologies this semester. Give a brief (one sentence) definition of each, and give a brief (one sentence) summary of their primary advantages.

Technology ...	*... brief definition ...*	*... advantages.*
Circuit-level parallelism	All parts of an electrical circuit can operate at the same time.	A billion-transistor chip has up to billion-way parallelism.
Pipelining	Each clock cycle we only execute a small part of each instruction, one instruction could take ~20 clock cycles to execute.	It might take ~20 clock cycles to execute a instruction but ~20 instructions can be in different stages of the pipeline at the same time, also the clock can be (ideally, each of our segments takes the same amount of time) run ~20 times faster.
Superscalar	Multiple instructions can be executed simultaneously by the hardwhere extracting parallelism.	No change in machine code necessary to us it looks like the hardwhere is just running faster do to it extracting the parallelism in our machine code.
SIMD (e.g., x86 SSE)	One instruction can be executed on a lot of data simultaneously.	"Increment these four integers" = one instruction = fast. Farley straightforward why this helps.
Multicore	Multiple threads can be executed on different cores on the chip.	Since we have multiple cores we don't need to do as many context switches, in other words each thread gets more time on a core to execute.
Hyper-Threading (SMT)	With hyper-threading we make the pipeline look like multiple cores.	We now extract some of the parallelism instead of it being extracted by the hardwhere by making our code run on multiple virtual cores.

1.a.) Which of these technologies has the **least** impact on programmers using it? Which has the **most** impact? Why?

Superscaler requires the least thought simply put in the exact same code in to the CPU and it executes faster. Since the CPU has curcits that extract parallelism we don't really need to think, we can help a little by making our code more friendly to superscaler basically try not to do sequential reads and writes on a single data member.

I would say circuit-level parallelism is the hardest to use as it requires us to design FPGA code and actually design a curcit.

1.b.) Which of these technologies has the most “headroom”, or future potential performance improvement, in each of these application areas?

Application Area	*Technology with the most headroom, and why*
Desktop CPU	Multicore, Hyperthreading -other than that desktops already take advantage of most improvements
Desktop GPU	
Embedded Processor (e.g., Arduino)	Circuit-level parallelism,Pipelining,SIMD

I have a issue with this question, since we are dealing with physical chips we have a limited amount of die space. Thus if we add more abilities to the GPU we inevitably end up needing to reduce the number of cores or increase the die size, ether one is a issue. Basically almost everything ends up being a trade off, theres a reason why we have both CPUs and GPUs they are useful for different things.

2.) Currently, we use electrons to communicate between nearby electrical components, but we use fiber optics for long range communication (for example, Terabit Ethernet requires fiber optics, not copper). Intel has invested significant effort in silicon photonics for future optical communication, even on chip. Please list the properties of photons for communication, as compared to conventional electrons.

Property ...	*... for electrons?*	*... for photons?*
Signal velocity	Up to half the speed of light	At the speed of light divided by optical density.
Interference	Major issue for long wires (act as antenna)	Photons carry no charge and thus do not radiate energy, theres no interference from nearby cables like we experience with electrons.
Amplification	Easy, with transistor	Easy on long fiber optic cables (like trans Atlantic) we use erbium to amplify the signal.
Leakage	Due to quantum tunneling, a significant problem	Photons can still experience quantum tunneling, but this is a smaller effect in fiber optics.

2.a.) A signal's transit time [seconds], is its path length [meters], divided by the signal velocity [meters/second]. What is the maximum clock rate could you reach with a core of diameter 1mm using optical communication?

Assuming the question to be how long it would take a signal to propagate across a circular 1mm chip. 1mm/c=3.3ps. Although we could get faster if we buffer transmission across the chip, pipeline cross chip communication.

3.) Both the CPU (e.g., C++11) and the GPU (e.g., CUDA) use “threads”, but on each platform this same term means something slightly different. For each property, write YES or NO, and briefly why this property does or does not apply.

Property ...	*... for CPU/C++11 threads?*	*... for GPU/CUDA threads?*
An atomic operation can be used to avoid a memory race condition.	YES, std::atomic<int>.fetch_add	YES, atomicSum
A lock (mutex) can be used to avoid a memory race condition.	YES, std::mutex has lock and unlock	NO, there is not a inherent ability to do this since we don't want to tie up all of the cores. However we could do this with a atomic swap if we really wanted to. Basically put a one in a variable each thread tries to atomic swap in a 0 the one that gets the one can execute (this is a bad idea but it can be done).
Memory accesses limit overall system performance more than arithmetic.	YES, if we need to reach into memory it can take a long time (relative to a arithmetic operation) if it goes out to ram.	Yes, if we have to access the shared memory between all of the blocks this can take a while, however if it is in this warp's local memory we can access it fast.
For performance, you should avoid situations where threads write to nearby memory.	YES, since each core is significantly separated accessing nearby data safely requires that the cores talk to each other, this takes a lot of time.	NO, in a block it is cheep to keep data accessible to all of the threads.
Adjacent threads branch together as one “warp”.	NO, each thread is almost completely separate, they can all do different things at no cost.	YES, threads are grouped into warps of 32 threads.
The problem must be (re)structured to allow parallel evaluation.	NOT NECESSARILY, it can significantly help to let our code run a little in parallel but most code is not.	YES, without huge parallelism the GPU is worthless compared to the CPU.
You can get good performance with 8 threads.	YES, most home computers only have 8 cores	NO, if you only have 8 fold parallelism might as well run this code on the CPU it would do far better.
You can get good performance with 800,000 threads.	NO, you will be doing worse than you were with 8 threads due to the overhead of creating threads. Plus the OS will need to spend a lot of time doing context switches between all of these threads, a waste.	YES, this is exactly what the GPU was designed for.