Exam Assignments 1

Parallelism vs Concurrency

Parallelism is a subset of concurrency.
Concurrency describes multiple tasks being in progress at once. This means that one task can start while another is already running. This may be accomplished by having multiple tasks running at the same time, or by pausing one task to run the other.
Parallelism is the former of these two options, allowing more than one task to run at once.

Fork-Join Parallelism

Fork-join parallelism is a design pattern of parallel computing in which a process starts with one master thread, which can then **fork** itself to create multiple threads, each of which has the ability to fork as well. However, each fork needs to be resolved by **joining** the threads together again, creating sort of a parentheses structure. The process thus needs to end with the master thread again.

Particularly Interesting Thing in Chapter 1 of Computer Systems: A Programmer's Perspective

Knowing roughly about how pipelining works, I wondered how a superscalar processor would function.

It is achieved by having multiples of each of the execution units that are responsible for processing an instruction, thus allowing multiple instructions to be processed in parallel.

From this I came to wonder how two instructions that operate on the same data would be run in parallel, since this would lead to a race condition. The answer to this problem is a dynamic run-time check the CPU does. I assume that when a data dependency is recognized, the instructions are just executed sequentially.

Explaining the Figure "Performance gains after Moore's law ends"

The figure presents areas of computing technology where opportunities for performance gains can be found after the exponential growth of transistor count on processors, also called Moore's Law, ends.

"The bottom" of the computing stack refers to low-level computing technology like semiconductor technology, improvements of which echo up the entire computing stack. The bottom has seen continual improvement in the form of minituarization since the dawn of semiconductor technology. However, this trend is predicted to stop soon as minituarization is hitting physical limits. Thus, other areas of improvement need to be found to handle increasingly intensive computations.

"The top" of the computing stack refers to high-level computing technologies like software, algorithms and hardware architecture. Each of these technologies brings opportunities for performance improvements:

- Software performance can be improved by removing software bloat (In particular, replacing reductionist software solutions with ones that are tailored specifically to the task at hand.) and by writing software in a way that makes use of the underlying hardware (For example, by using parallel processing and vector instructions.)
- Development of new algorithms can improve performance particularly in new problem domains and by making use of new machine models, for example making use of parallel processing. Improvements in algorithms are also possible for older problem domains, but will see diminishing returns.
- Hardware architecture sees opportunities for performance improvements mainly in the streamlining of hardware. Until now, CPUs have become more and more complex, to handle sequential work more efficiently. As this kind of improvement is hitting limits, a simplification of CPUs is foreseen to hold significant opportunities, as this would allow more cores to fit on a single processing unit, bringing more parallelism. Furthermore, hardware is predicted to specialize more for specific domains. As can be seen with GPUs, specialized hardware can vastly outperform general purpose hardware in their respective fields, while also allowing for some generalization. GPUs for example were developed for graphics processing, but are also highly used in machine learning.