Algorithm Engineering
Exam Assignments

Sheela Orgler

Friedrich Schiller Universität, Jena

1 Assignment

1.1 Describe how parallelism differs from concurrency

Parallelism is a subset of concurrency. If a system is parallel it is also concurrent, but not the other way around. A system is considered parallel if two or more tasks are executed simultaneously.
Concurrency only means support for two or more actions at the same time. Parallelism refers to execution of more than one action at the same time.

1.2 What is fork-join parallelism?

Fork-join parallelism refers to a concept where we have a master thread that divides into a team of threads. With this concept we only have parallel regions where threads are executed simultaneously. In between these parallel regions the execution is sequential following the master thread.
We can imagine execution as a line that is followed from starting to end point. With fork-join parallelism this line is our master thread. Following our master thread at some point the thread can divide into two or more threads. These threads are joined after some time to our master thread. In these sections where we have two or more threads execution will be parallel and after these sections sequential.

1.3 Chapter 1 – Computer Systems: A Programmer's Perspective

Discuss one thing - Caches As discussed in Chapter 1 of the book caches are helpful to deal with the processor-memory gap.

The processor can read data from the register file, which is within the CPU almost 100 times faster than from memory. The gap between processor and memory is continuously growing.

Cache memory is used to bridge the gap. It is used to temporarily store information that will likely be used in the future. There exist several levels of caches - L1, L2, L3 - which go from smallest to largest.

A cache reduces the acess time to data in memory. Frequently used data and instructions are kept in the cache. Access to data kept there is faster.

Modern desktops, servers and industrial CPUs have at least three independent caches: instruction cache, data cache and the TLB. The instruction cache is

used to speed up executable instruction fetches. The data cache speeds up data fetch and store. It is organized into more cache levels:

- L1 cache - Primary cache is fast but small and usually embedded in the processor chip (CPU)
- L2 cache - Secondary cache is larger and can be embedded in the CPU or on a seperate chip or coprocessor containing a high-speed alternative system bus connectin cache and CPU.
- L3 cache - specialized memory to improve L1 and L2. L3 is usually double the speed of DRAM

The small caches L1 are backed up by larger and slower caches L2, L3 to address the tradeoff between cache latency and hit rate. The faster cache is generally checked firt. If that cache misses, the next chache is checked and so on, before accessing main memory.

For example the ARM-base Apple M1 CPU has 8 cores: 4 high-performance and four high-efficiency cores. The four high-performance cores have a 192 KiB L1 cache for each of the cores and the four high-efficiency cores only have 128 KiB.

The Translation lookaside buffer is used to speed up the translation from virtual to physical addresses. It is part of the memory management unit and not directly related to CPU caches.

1.4 Paper – There's plenty of room at the Top: What will drive computer performance after Moore's law?

Explain figure "Performance gains after Moore's law ends" The figure illustrates the "Top" and "Bottom" referring to computer performance gains. The "Bottom" refers to the miniaturization of computer components seen in the last decades. Due to physical limits the opportunities for gains at the bottom will slowly come to an end. Nevertheless, there are still opportunities for growth at the "Top". The "Top" shows the three aspects where growth can be expected: software, algorithms, hardware architecture. These aspects are divided into technology, opportunity and examples all looking into the growth opportunities in the specific area.
Growth opportunities at the "Top":

- making software more efficient by performance engineering
- minimizing the time it takes to run and not the development time
- an increasing number of processor cores running parallel
- reengineer modularity to obtain performance gains

2 Assignment

2.1 What causes false sharing?

2.2 How do mutual exclusion constructs prevent race conditions?

2.3 Explain the differences betweent static and dynamic schedules in OpenMP.

2.4 What can we do if we've found a solution while running a parallel for loop in OpenMP, but still have many iterations left?

2.5 Coding warmup slide 22: Explain in your own words how std::atomic::compare_exchange_weak work

References

1. Armv8-A Reference Manual pdf, pages 30-70, pages 1708-1808 https://developer.arm.com/documentation/ddi0487/ga.
2. Armv8-A Address Translation pdf, https://developer.arm.com/documentation/100940/0101/.