Figure 6.1 Hardware/software categorization and examples of application perspective on concurrency versus hardware perspective on parallelism.

Figure 6.2 Hardware categorization and examples based on number of instruction streams and data streams: SISD, SIMD, MISD, and MIMD.

Figure 6.3 Using multiple functional units to improve the performance of a single vector add instruction, C = A + B. The vector processor (a) on the left has a single add pipeline and can complete one addition per cycle. The vector processor (b) on the right has four add pipelines or lanes and can complete four additions per cycle. The elements within a single vector add instruction are interleaved across the four lanes.

Figure 6.4 Structure of a vector unit containing four lanes. The vector-register storage is divided across the lanes, with each lane holding every fourth element of each vector register. The figure shows three vector functional units: an FP add, an FP multiply, and a load-store unit. Each of the vector arithmetic units contains four execution pipelines, one per lane, which acts in concert to complete a single vector instruction. Note how each section of the vector-register file only needs to provide enough read and write ports (see Chapter 4) for functional units local to its lane.

Figure 6.5 How four threads use the issue slots of a superscalar processor in different approaches. The four threads at the top show how each would execute running alone on a standard superscalar processor without multithreading support. The three examples at the bottom show how they would execute running together in three multithreading options. The horizontal dimension represents the instruction issue capability in each clock cycle. The vertical dimension represents a sequence of clock cycles. An empty (white) box indicates that the corresponding issue slot is unused in that clock cycle. The shades of gray and color correspond to four different threads in the multithreading processors. The additional pipeline start-up effects for coarse multithreading, which are not illustrated in this figure, would lead to further loss in throughput for coarse multithreading.

Figure 6.6 The speed-up from using multithreading on one core on an i7 processor averages 1.31 for the PARSEC benchmarks (see  Section 6.10) and the energy efficiency improvement is 1.07. This data was collected and analyzed by Esmaeilzadeh et. al. [2011].

Figure 6.7 Classic organization of a shared memory multiprocessor.

Figure 6.8 The last four levels of a reduction that sums results from each processor, from bottom to top. For all processors whose number i is less than half, add the sum produced by processor number (i + half) to its sum.

Figure 6.9 Simplified block diagram of the datapath of a multithreaded SIMD Processor. It has 16 SIMD lanes. The SIMD Thread Scheduler has many independent SIMD threads that it chooses from to run on this processor.

Figure 6.10 GPU Memory structures. GPU Memory is shared by the vectorized loops. All threads of SIMD instructions within a thread block share Local Memory.

Figure 6.11 Similarities and differences between multicore with Multimedia SIMD extensions and recent GPUs.

Figure 6.12 Quick guide to GPU terms. We use the first column for hardware terms. Four groups cluster these 12 terms. From top to bottom: Program Abstractions, Machine Objects, Processing Hardware, and Memory Hardware.

Figure 6.13 TPUv1 Block Diagram. The main computation part is the Matrix Multiply Unit in the upper right corner. Its inputs are the Weight FIFO and the Unified Buffer, and its output is the Accumulators. The 24 MiB Unified Buffer is almost a third of the TPUv1 die, and the Matrix Multiply Unit with 65,536 multiple-accumulate ALUs is a quarter, so the datapath is nearly two-thirds of the TPUv1 die. For CPUs, multilevel caches are often two-thirds of the die.

Figure 6.14 Classic organization of a multiprocessor with multiple private address spaces, traditionally called a message-passing multiprocessor. Note that unlike the SMP in Figure 6.7, the interconnection network is not between the caches and memory but is instead between processor-memory nodes.

Figure 6.15 Network topologies that have appeared in commercial parallel processors. The colored circles represent switches and the black squares represent processor-memory nodes. Even though a switch has many links, generally only one goes to the processor. The Boolean n-cube topology is an n-dimensional interconnect with 2*n* nodes, requiring n links per switch (plus one for the processor) and thus n nearest-neighbor nodes. Frequently, these basic topologies have been supplemented with extra arcs to improve performance and reliability.

Figure 6.16 Popular multistage network topologies for eight nodes. The switches in these drawings are simpler than in earlier drawings because the links are unidirectional; data comes in at the left and exits out the right link. The switch box in c can pass A to C and B to D or B to C and A to D. The crossbar uses n2 switches, where n is the number of processors, while the Omega network uses 2n log2n of the large switch boxes, each of which is logically composed of four of the smaller switches. In this case, the crossbar uses 64 switches versus 12 switch boxes, or 48 switches, in the Omega network. The crossbar, however, can support any combination of messages between processors, while the Omega network cannot.

Figure 6.17 Examples of parallel benchmarks.

Figure 6.18 Arithmetic intensity, specified as the number of float-point operations to run the program divided by the number of bytes accessed in main memory [Williams, Waterman, and Patterson, 2009]. Some kernels have an arithmetic intensity that scales with problem size, such as Dense Matrix, but there are many kernels with arithmetic intensities independent of problem size. For kernels in this former case, weak scaling can lead to different results, since it puts much less demand on the memory system.

Figure 6.19 Roofline Model [Williams, Waterman, and Patterson, 2009]. This example has a peak floating-point performance of 16 GFLOPS/sec and a peak memory bandwidth of 16 GB/sec from the Stream benchmark. (Since Stream is actually four measurements, this line is the average of the four.) The dotted vertical line in color on the left represents Kernel 1, which has an arithmetic intensity of 0.5 FLOPs/byte. It is limited by memory bandwidth to no more than 8 GFLOPS/sec on this Opteron X2. The dotted vertical line to the right represents Kernel 2, which has an arithmetic intensity of 4 FLOPs/byte. It is limited only computationally to 16 GFLOPS/s. This data is based on the AMD Opteron X2 (Revision F) using dual cores running at 2 GHz in a dual socket system.

Figure 6.20 Roofline models of two generations of Opterons. The Opteron X2 roofline, which is the same as in Figure 6.19, is in black, and the Opteron X4 roofline is in color. The bigger ridge point of Opteron X4 means that kernels that were computationally bound on the Opteron X2 could be memory-performance bound on the Opteron X4.

Figure 6.21 Roofline model with ceilings. The top graph shows the computational “ceilings” of 8 GFLOPs/sec if the floating-point operation mix is imbalanced and 2 GFLOPs/sec if the optimizations to increase ILP and SIMD are also missing. The bottom graph shows the memory bandwidth ceilings of 11 GB/sec without software prefetching and 4.8 GB/sec if memory affinity optimizations are also missing.

Figure 6.22 Roofline model with ceilings, overlapping areas shaded, and the two kernels from Figure 6.19. Kernels whose arithmetic intensity land in the blue trapezoid on the right should focus on computation optimizations, and kernels whose arithmetic intensity land in the gray triangle in the lower left should focus on memory bandwidth optimizations. Those that land in the blue-gray parallelogram in the middle need to worry about both. As Kernel 1 falls in the parallelogram in the middle, try optimizing ILP and SIMD, memory affinity, and software prefetching. Kernel 2 falls in the trapezoid on the right, so try optimizing ILP and SIMD and the balance of floating-point operations.

Figure 6.23 Block diagram of a TPUv3 TensorCore.

Figure 6.24 A TPUv3 supercomputer consisting of up to 1024 chips (left). It is about 6 ft tall and 40 ft long. A TPUv3 board (right) has four chips and uses liquid cooling.

Figure 6.25 Key processor features of TPUv1, TPUv3, and NVIDIA Volta GPU.

Figure 6.26 Rooflines of TPUv3 and Volta.

Figure 6.27 Adjusted comparison of GPU and TPUv3. Die sizes are adjusted by the square of the technology, as the semiconductor technology for TPUs is similar but larger and older than that of the GPU. Google picked 15 nm for TPUs based on the information in Figure 6.25. Thermal Design Power (TDP) is for 16-chip systems.

Figure 6.28 Performance per chip of TPUv3 relative to Volta for five MLPerf 0.6 benchmarks and four production applications.

Figure 6.29 Supercomputer scaling: TPUv3 and Volta.

Figure 6.30 Traditional versus TPUv3 supercomputer Top500 and Green500 rank (June 2019) for Linpack and AlphaZero.

Figure 6.31 OpenMP version of DGEMM from Figure 5.48. Line 27 is the only OpenMP code, making the outermost for loop operate in parallel. This line is the only difference from Figure 5.48.

Figure 6.32 Performance improvements relative to a single thread as the number of threads increase. The most honest way to present such graphs is to make performance relative to the best version of a single processor program, which we did. This plot is relative to the performance of the code in Figure 5.48 without including OpenMP pragmas.

Figure 6.33 DGEMM performance versus the number of threads for four matrix sizes. The performance improvement compared to the original C code in Figure 2.43 for the 960 × 960 matrix with 32 threads is an astounding 150 times faster!