Chapter 04

Authors: John Hennessy & David Patterson

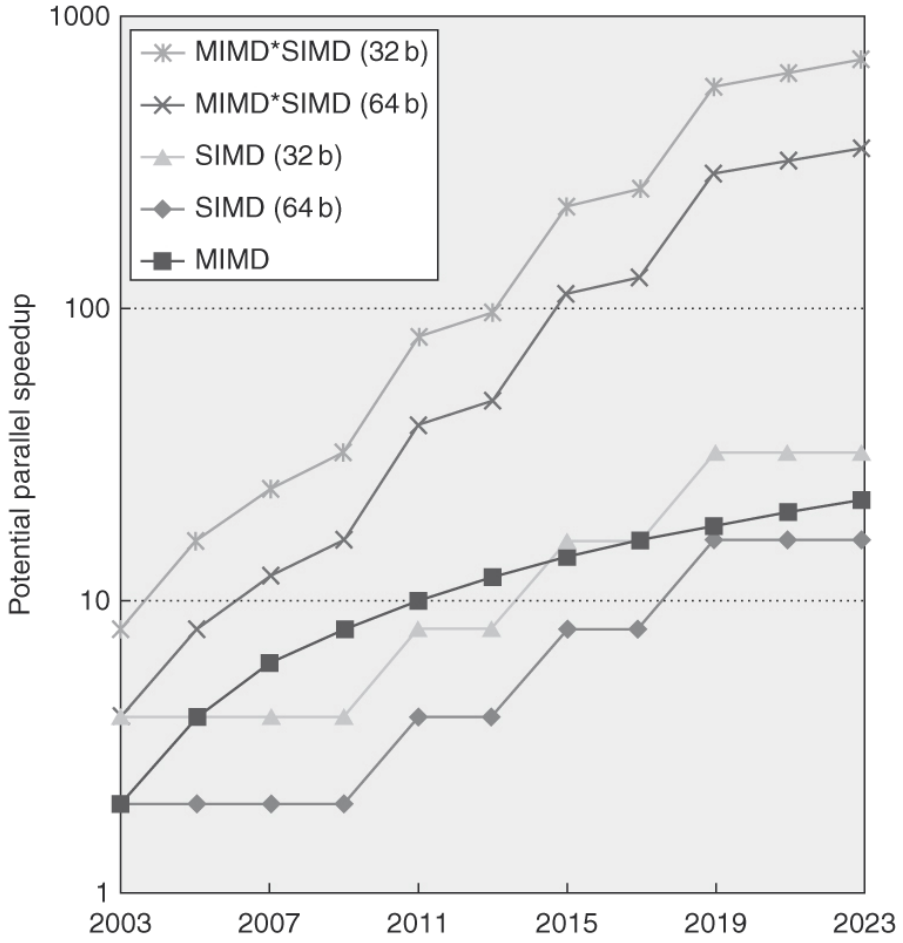

Figure 4.1 Potential speedup via parallelism from MIMD, SIMD, and both MIMD and SIMD over time for x86 computers. This figure assumes that two cores per chip for MIMD will be added every two years and the number of operations for SIMD will double every four years.

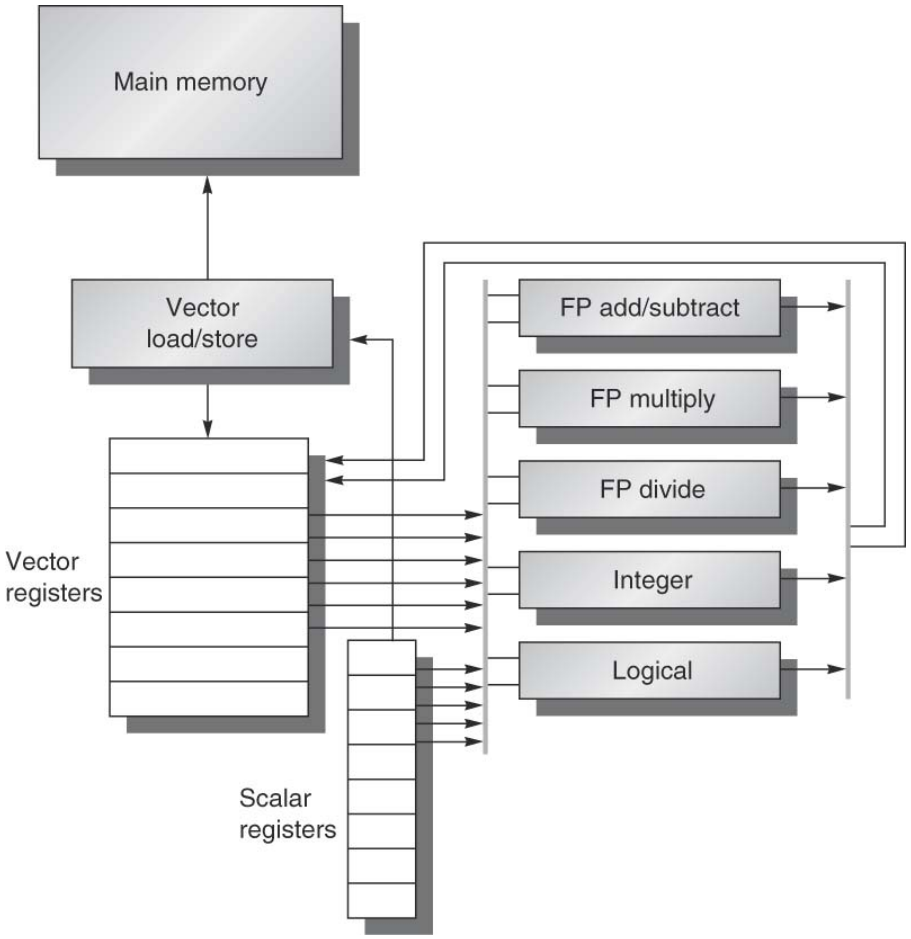

Figure 4.2 The basic structure of a vector architecture, VMIPS. This processor has a scalar architecture just like MIPS. There are also eight 64-element vector registers, and all the functional units are vector functional units. This chapter defines special vector instructions for both arithmetic and memory accesses. The figure shows vector units for logical and integer operations so that VMIPS looks like a standard vector processor that usually includes these units; however, we will not be discussing these units. The vector and scalar registers have a significant number of read and write ports to allow multiple simultaneous vector operations. A set of crossbar switches (thick gray lines) connects these ports to the inputs and outputs of the vector functional units.

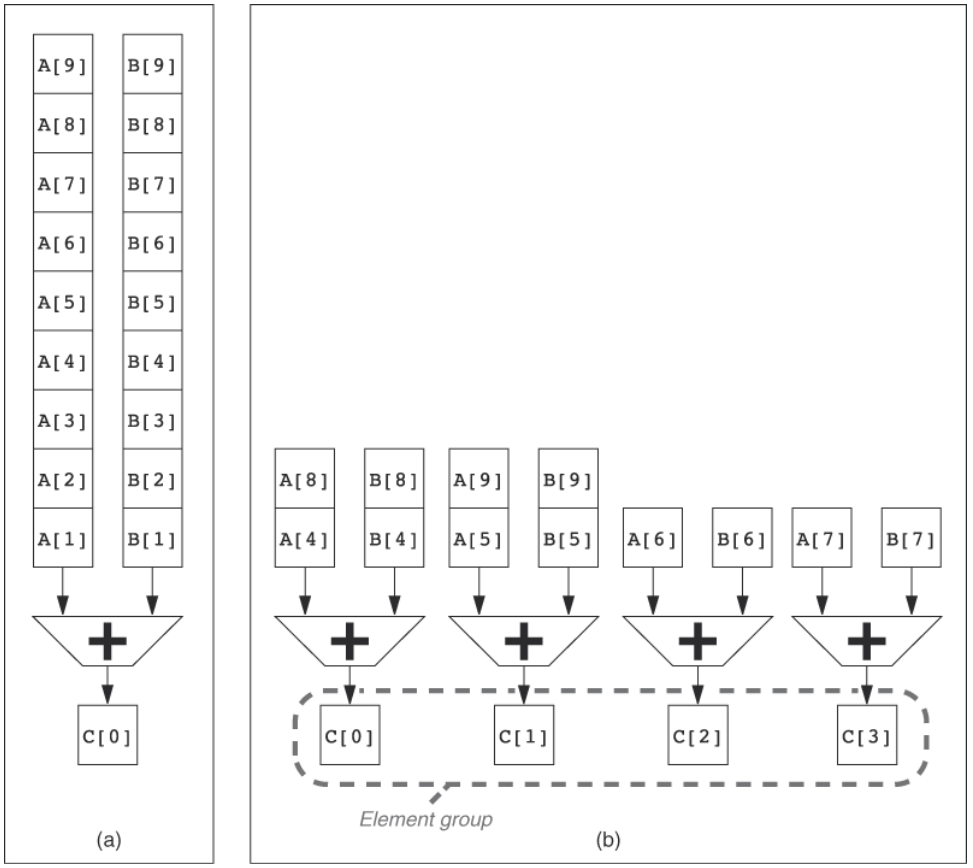

Figure 4.4 Using multiple functional units to improve the performance of a single vector add instruction, C = A + B. The vector processor (a) on the left has a single add pipeline and can complete one addition per cycle. The vector processor (b) on the right has four add pipelines and can complete four additions per cycle. The elements within a single vector add instruction are interleaved across the four pipelines. The set of elements that move through the pipelines together is termed an *element group*. (Reproduced with permission from Asanovic [1998].)

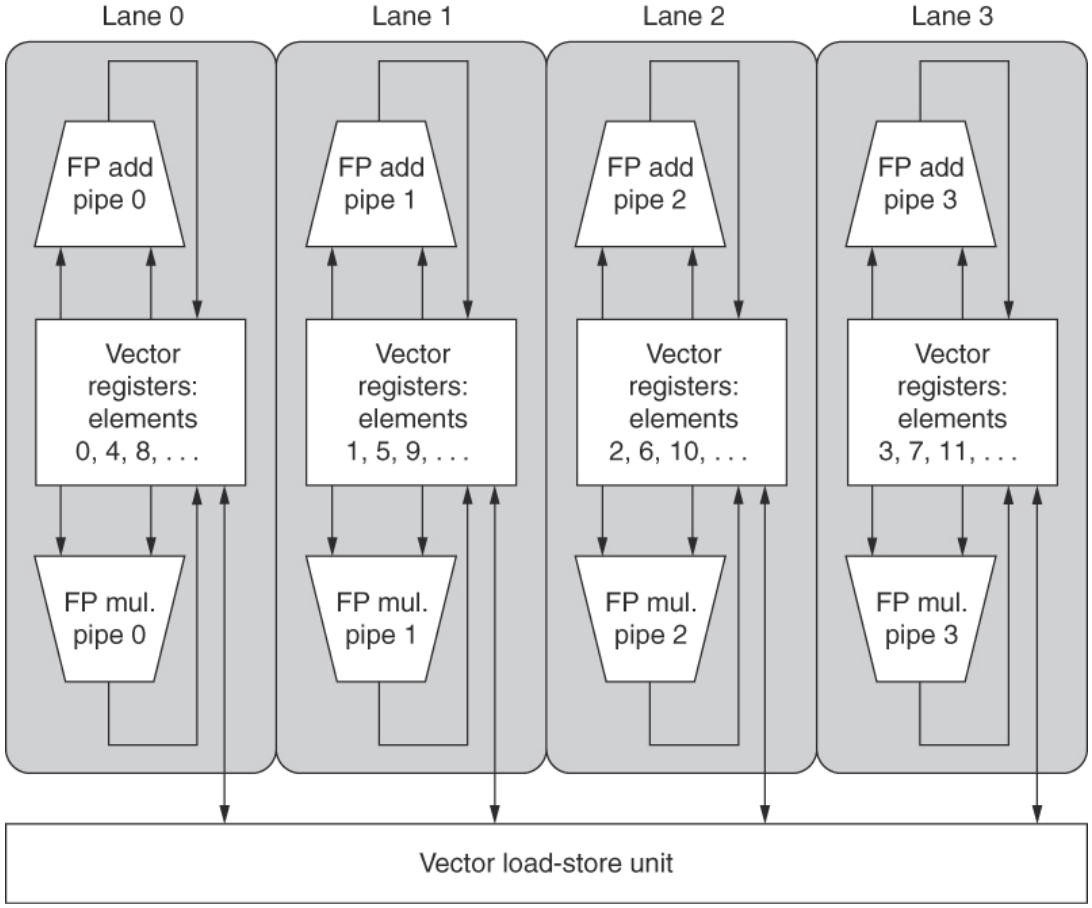

Figure 4.5 Structure of a vector unit containing four lanes. The vector register storage is divided across the lanes, with each lane holding every fourth element of each vector register. The figure shows three vector functional units: an FP add, an FP multiply, and a load-store unit. Each of the vector arithmetic units contains four execution pipelines, one per lane, which act in concert to complete a single vector instruction. Note how each section of the vector register file only needs to provide enough ports for pipelines local to its lane. This figure does not show the path to provide the scalar operand for vector-scalar instructions, but the scalar processor (or control processor) broadcasts a scalar value to all lanes.

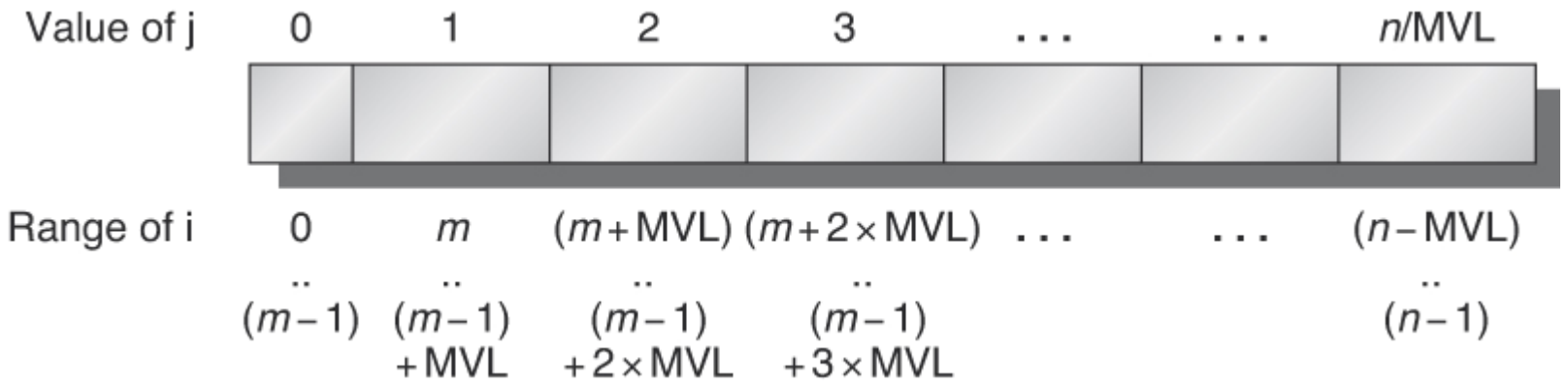

Figure 4.6 A vector of arbitrary length processed with strip mining. All blocks but the first are of length MVL, utilizing the full power of the vector processor. In this figure, we use the variable *m* for the expression (n % MVL). (The C operator % is modulo.)

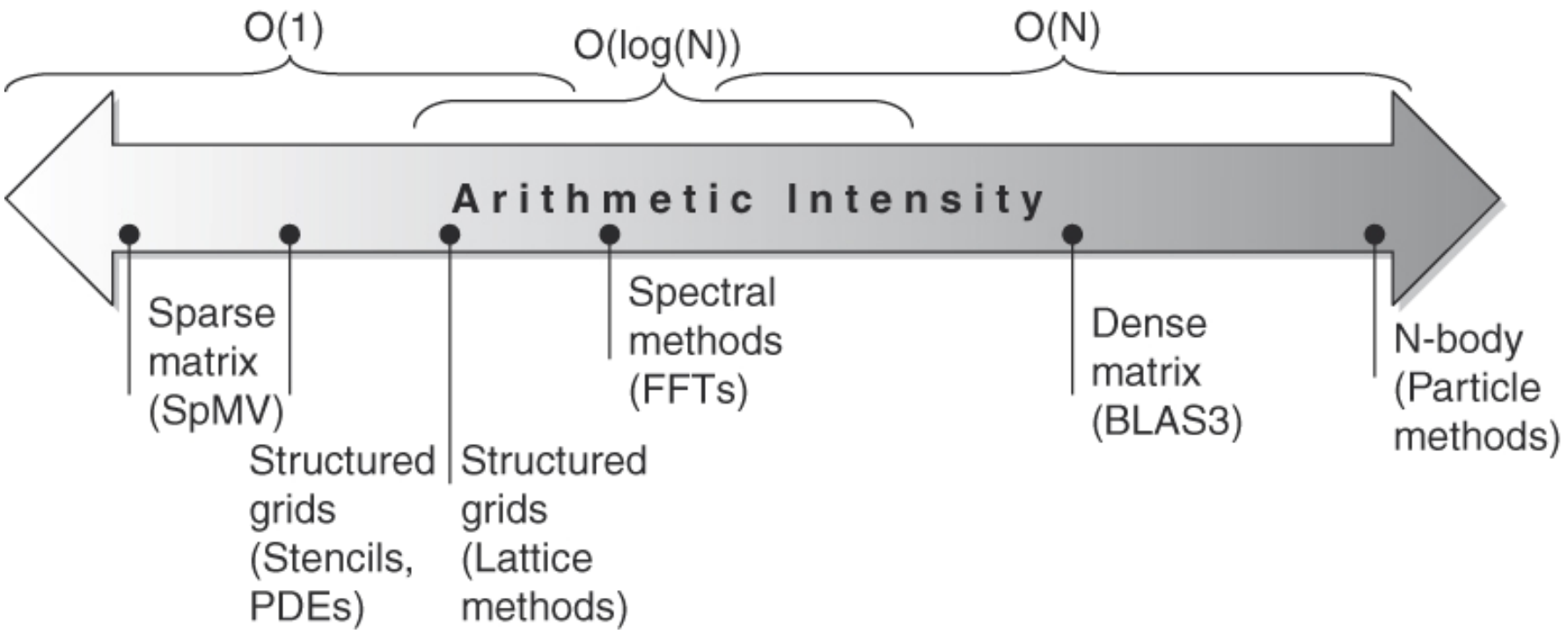

Figure 4.10 Arithmetic intensity, specified as the number of floating-point operations to run the program divided by the number of bytes accessed in main memory [Williams et al. 2009]. Some kernels have an arithmetic intensity that scales with problem size, such as dense matrix, but there are many kernels with arithmetic intensities independent of problem size.

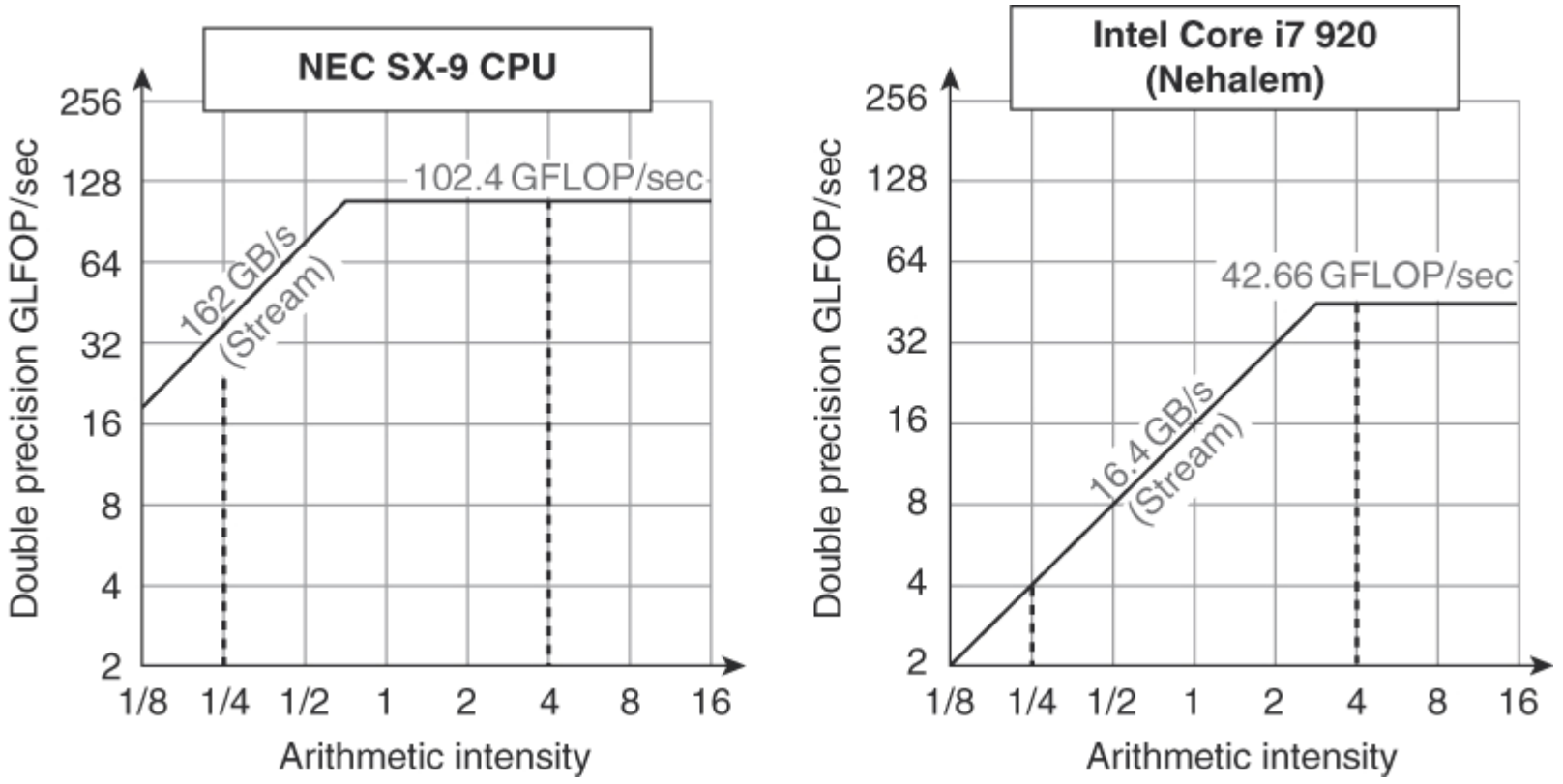

Figure 4.11 Roofline model for one NEC SX-9 vector processor on the left and the Intel Core i7 920 multicore computer with SIMD Extensions on the right [Williams et al. 2009]. This Roofline is for unit-stride memory accesses and double-precision floating-point performance. NEC SX-9 is a vector supercomputer announced in 2008 that costs millions of dollars. It has a peak DP FP performance of 102.4 GFLOP/sec and a peak memory bandwidth of 162 GBytes/sec from the Stream benchmark. The Core i7 920 has a peak DP FP performance of 42.66 GFLOP/sec and a peak memory bandwidth of 16.4 GBytes/sec. The dashed vertical lines at an arithmetic intensity of 4 FLOP/byte show that both processors operate at peak performance. In this case, the SX-9 at 102.4 FLOP/sec is 2.4x faster than the Core i7 at 42.66 GFLOP/sec. At an arithmetic intensity of 0.25 FLOP/byte, the SX-9 is 10x faster at 40.5 GFLOP/sec versus 4.1 GFLOP/sec for the Core i7.

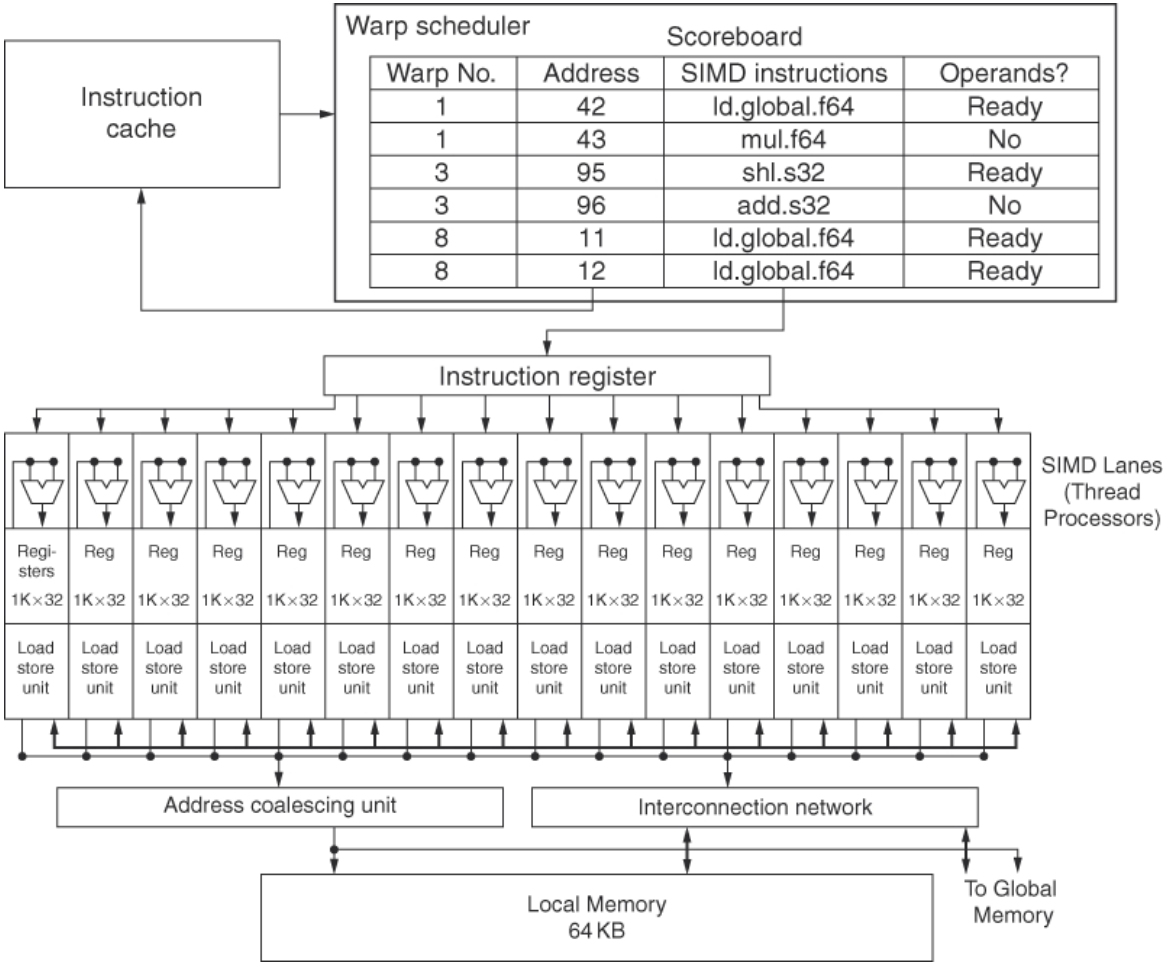

Figure 4.14 Simplified block diagram of a Multithreaded SIMD Processor. It has 16 SIMD lanes. The SIMD Thread Scheduler has, say, 48 independentthreads of SIMD instructions that it schedules with a table of 48 PCs.

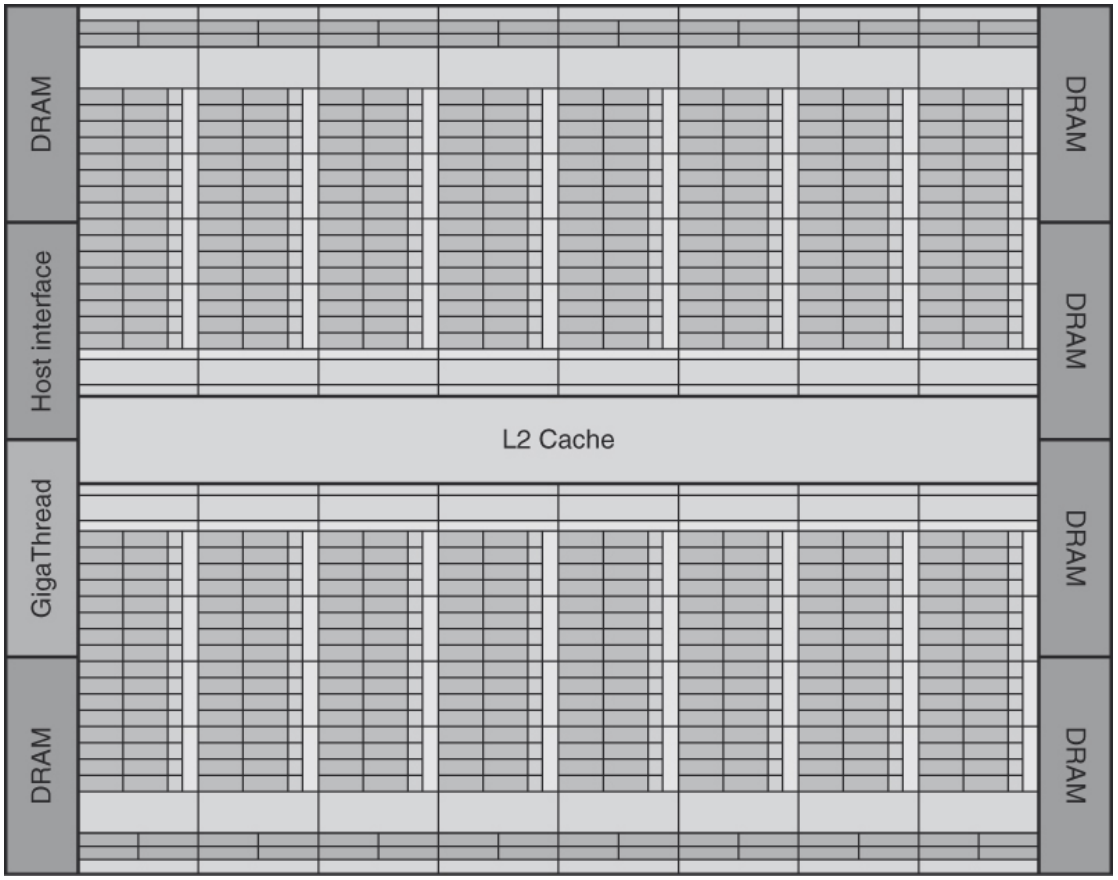

Figure 4.15 Floor plan of the Fermi GTX 480 GPU. This diagram shows 16 multithreaded SIMD Processors. The Thread Block Scheduler is highlighted on the left. The GTX 480 has 6 GDDR5 ports, each 64 bits wide, supporting up to 6 GB of capacity. The Host Interface is PCI Express 2.0 x 16. Giga Thread is the name of the scheduler that distributes thread blocks to Multiprocessors, each of which has its own SIMD Thread Scheduler.

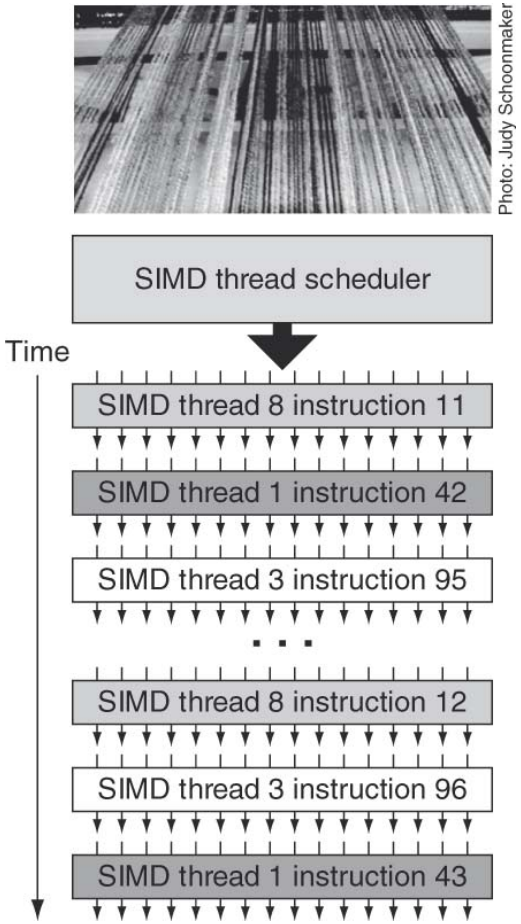

Figure 4.16 Scheduling of threads of SIMD instructions. The scheduler selects a ready thread of SIMD instructions and issues an instruction synchronously to all the SIMD Lanes executing the SIMD thread. Because threads of SIMD instructions are independent, the scheduler may select a different SIMD thread each time.

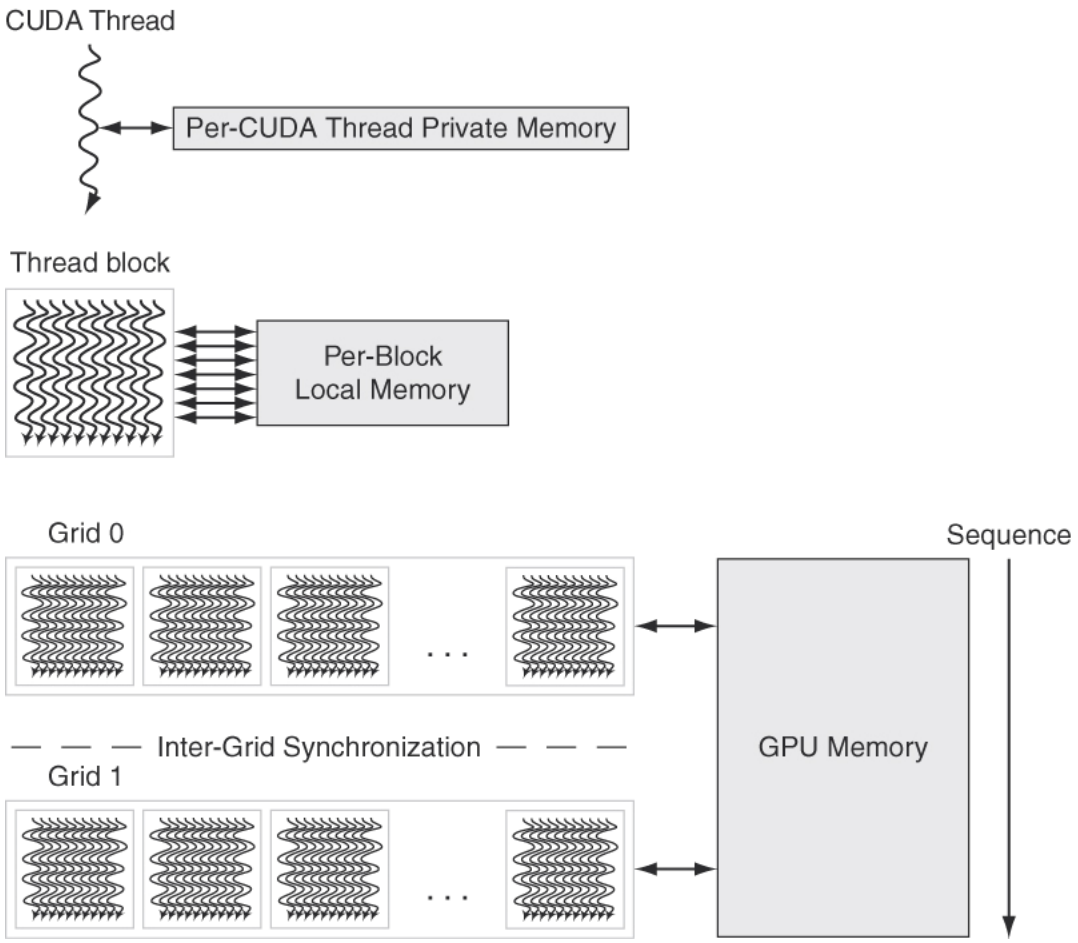

Figure 4.18 GPU Memory structures. GPU Memory is shared by all Grids (vectorized loops), Local Memory is shared by all threads of SIMD instructions within a thread block (body of a vectorized loop), and Private Memory is private to a single CUDA Thread.

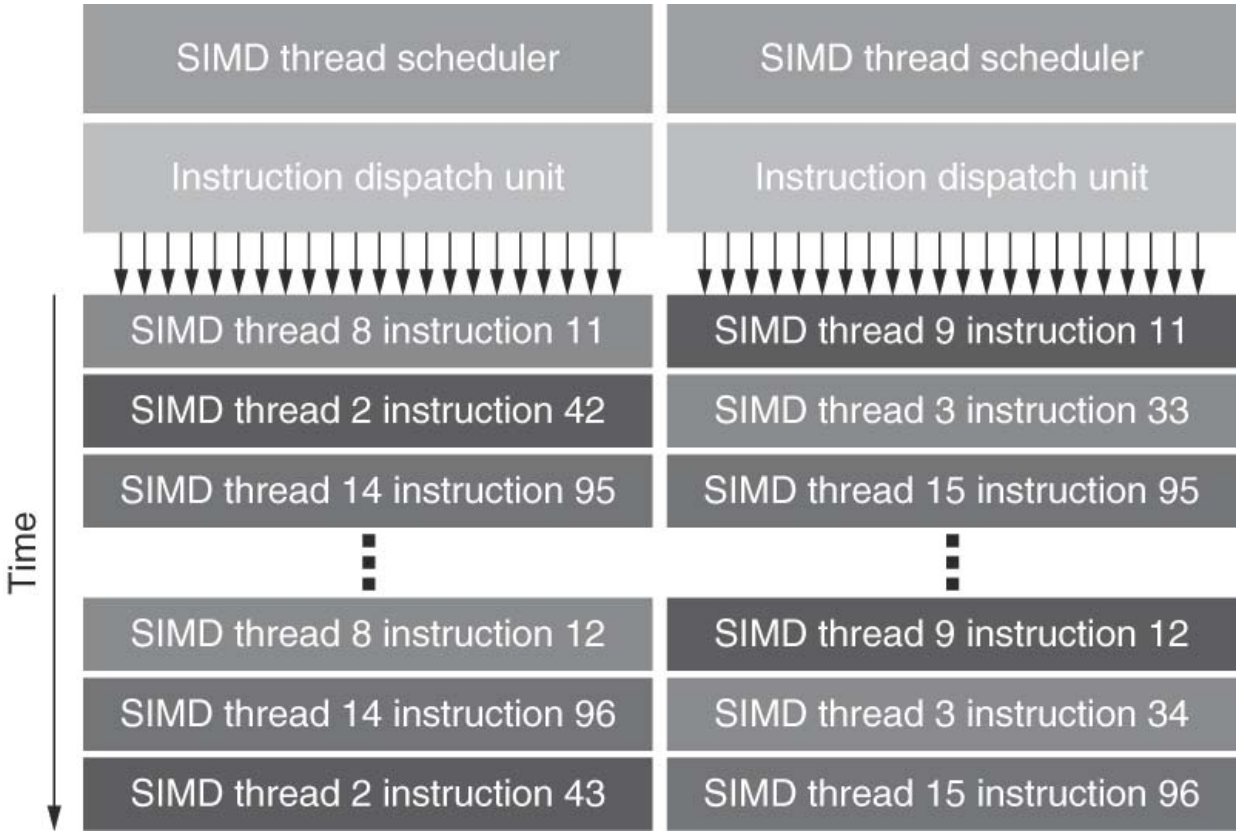

Figure 4.19 Block Diagram of Fermi's Dual SIMD Thread Scheduler. Compare this design to the single SIMD Thread Design in Figure 4.16.

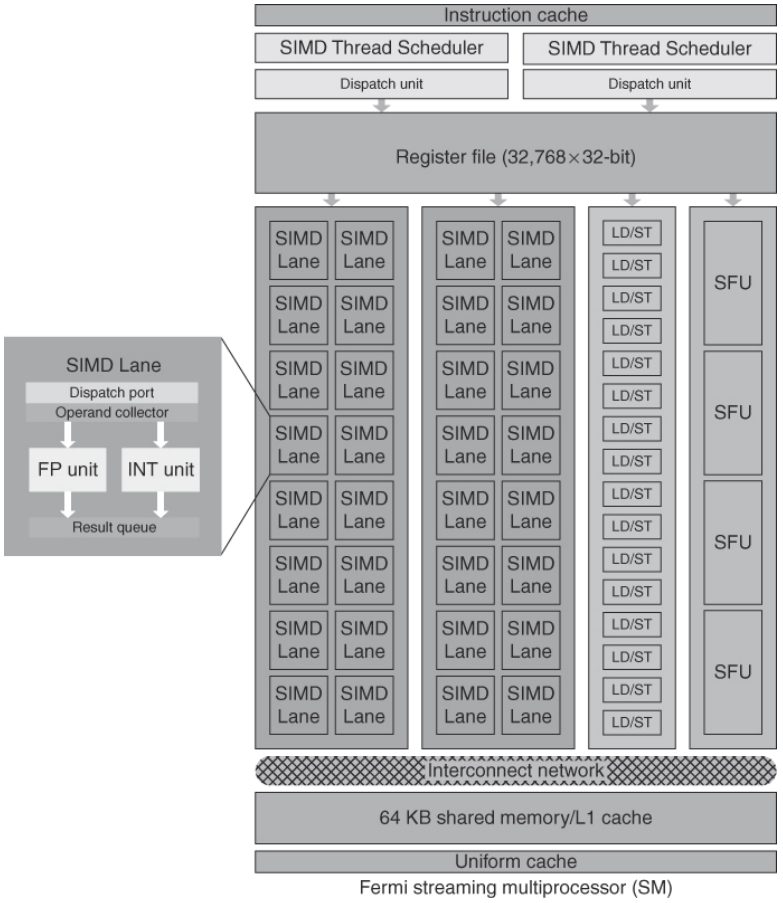

Figure 4.20 Block diagram of the multithreaded SIMD Processor of a Fermi GPU. Each SIMD Lane has a pipelined floating-point unit, a pipelined integer unit, some logic for dispatching instructions and operands to these units, and a queue for holding results. The four Special Function units (SFUs) calculate functions such as square roots, reciprocals, sines, and cosines.

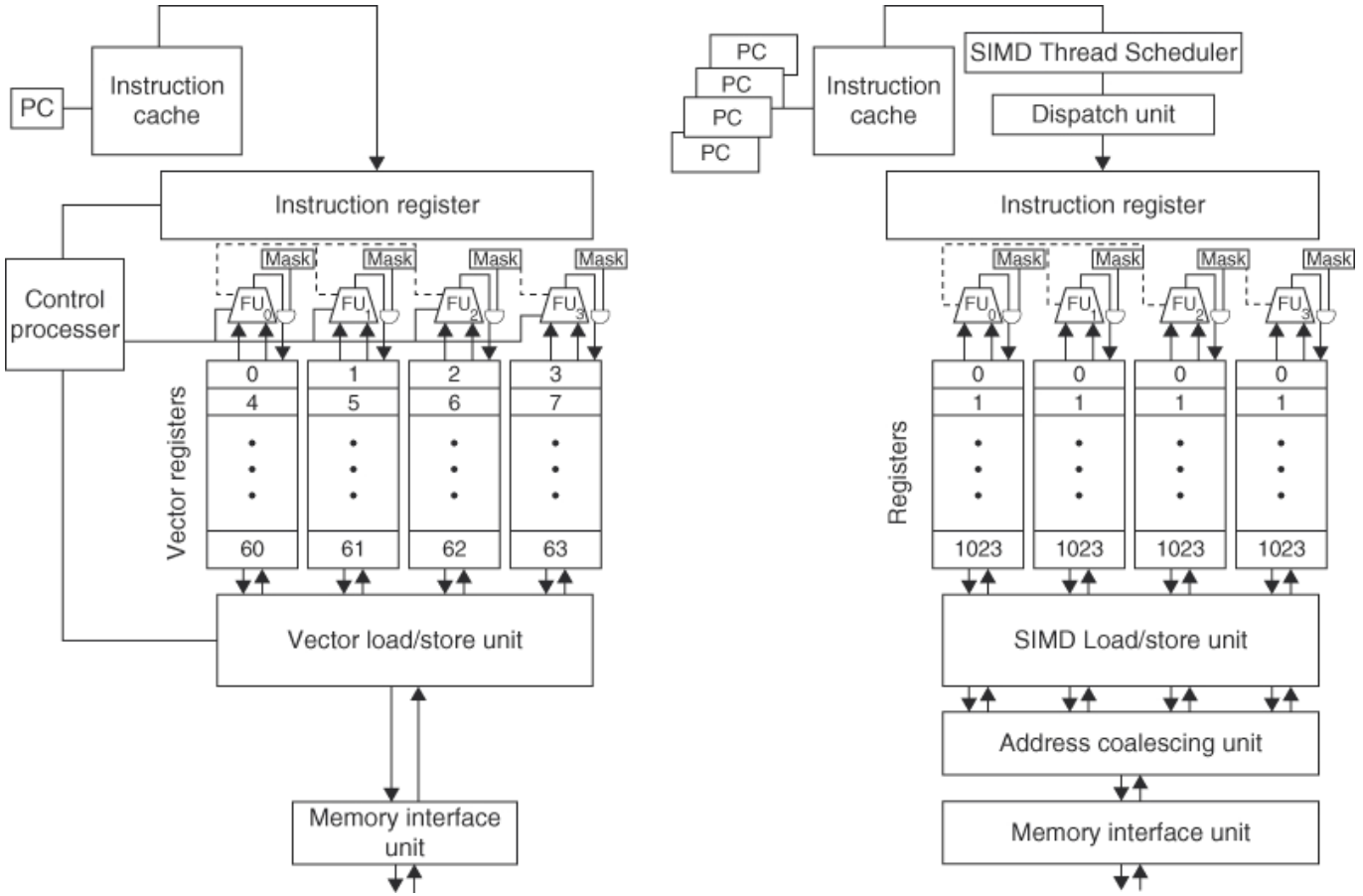

Figure 4.22 A vector processor with four lanes on the left and a multithreaded SIMD Processor of a GPU with four SIMD Lanes on the right. (GPUs typically have 8 to 16 SIMD Lanes.) The control processor supplies scalar operands for scalar-vector operations, increments addressing for unit and non-unit stride accesses to memory, and performs other accounting-type operations. Peak memory performance only occurs in a GPU when the Address Coalescing unit can discover localized addressing. Similarly, peak computational performance occurs when all internal mask bits are set identically. Note that the SIMD Processor has one PC per SIMD thread to help with multithreading.

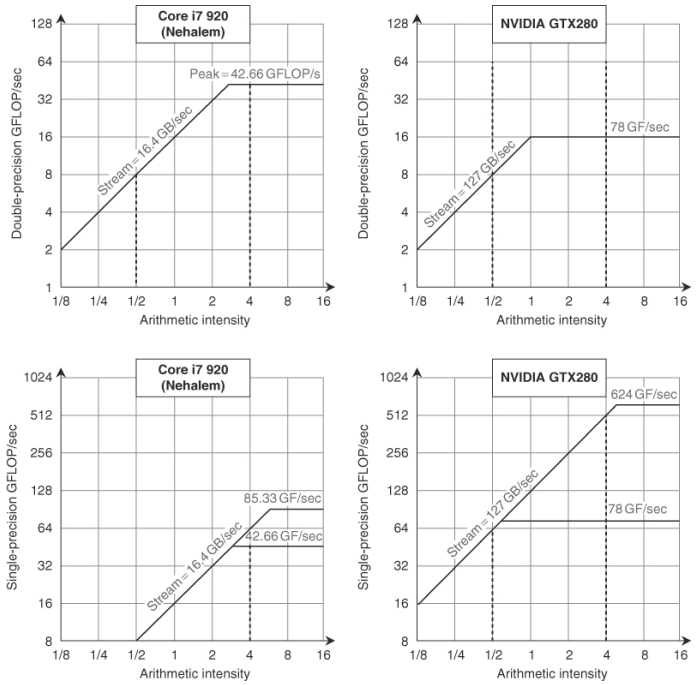

Figure 4.28 Roofline model [Williams et al. 2009]. These rooflines show double-precision floating-point performance in the top row and single-precision performance in the bottom row. (The DP FP performance ceiling is also in the bottom row to give perspective.) The Core i7 920 on the left has a peak DP FP performance of 42.66 GFLOP/sec, a SP FP peak of 85.33 GFLOP/sec, and a peak memory bandwidth of 16.4 GBytes/sec. The NVIDIA GTX 280 has a DP FP peak of 78 GFLOP/sec, SP FP peak of 624 GFLOP/sec, and 127 GBytes/sec of memory bandwidth. The dashed vertical line on the left represents an arithmetic intensity of 0.5 FLOP/byte. It is limited by memory bandwidth to no more than 8 DP GFLOP/sec or 8 SP GFLOP/sec on the Core i7. The dashed vertical line to the right has an arithmetic intensity of 4 FLOP/byte. It is limited only computationally to 42.66 DP GFLOP/sec and 64 SP GFLOP/sec on the Core i7 and 78 DP GFLOP/sec and 512 DP GFLOP/sec on the GTX 280. To hit the highest computation rate on the Core i7 you need to use all 4 cores and SSE instructions with an equal number of multiplies and adds. For the GTX 280, you need to use fused multiply-add instructions on all multithreaded SIMD processors. Guz et al. [2009] have an interesting analytic model for these two architectures.

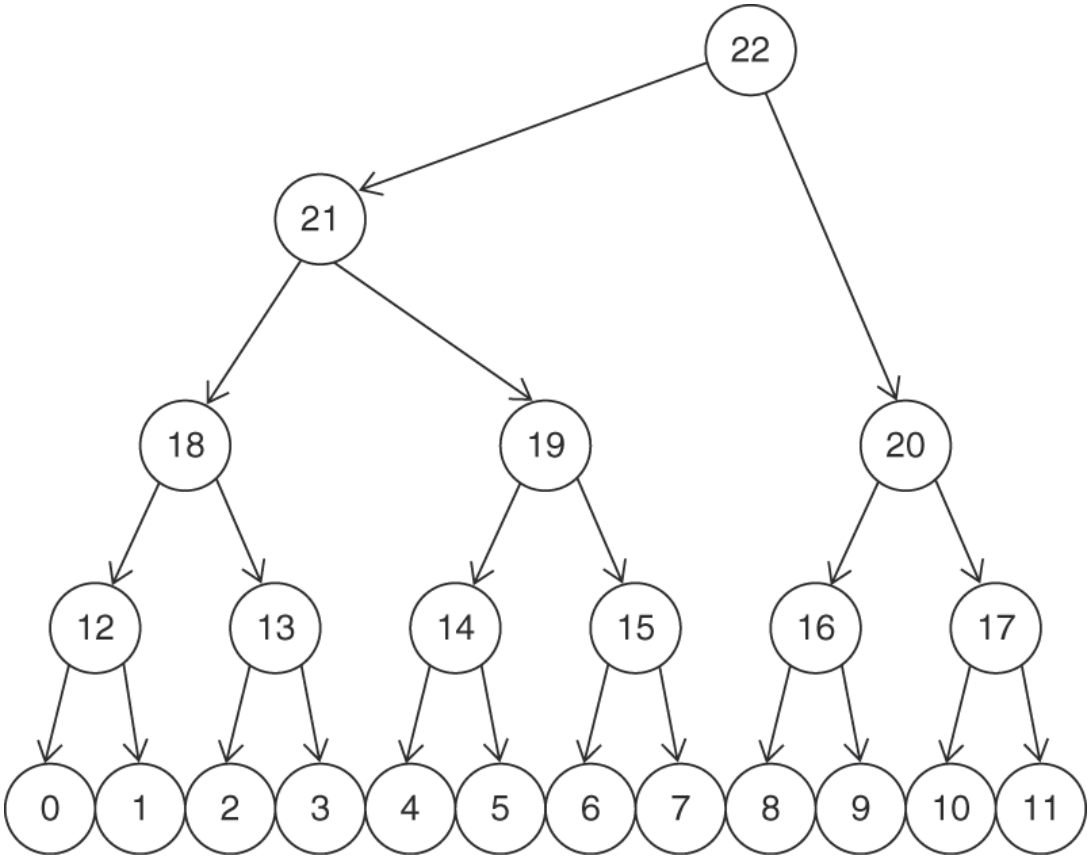

Figure 4.33 Sample tree.