		Software	
		Sequential	**Concurrent**
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

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

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Core i7

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

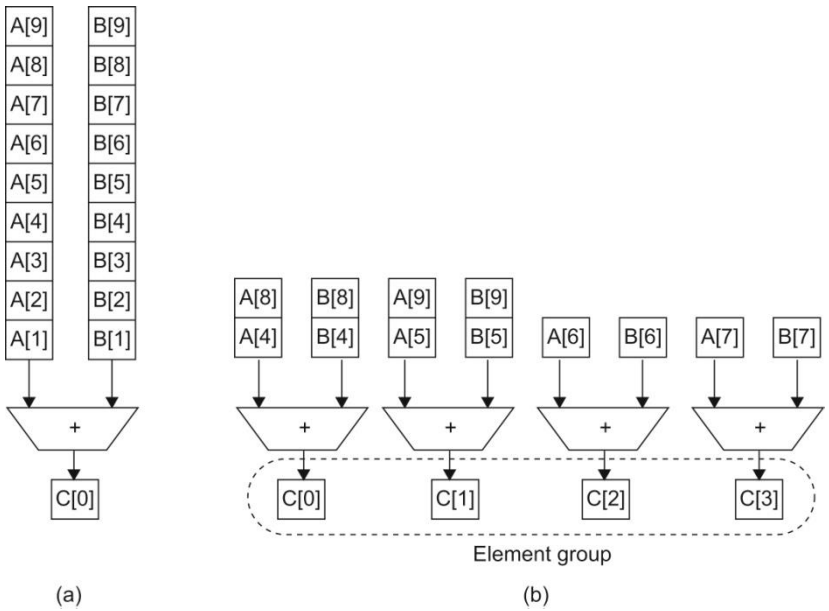


Figure 6.3 Using multiple functional units to improve the performance of a single vector add instruction, C = A + B. The vector processor (a) on the left has a single add pipeline and can complete one addition per cycle. The vector processor (b) on the right has four add pipelines or lanes and can complete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

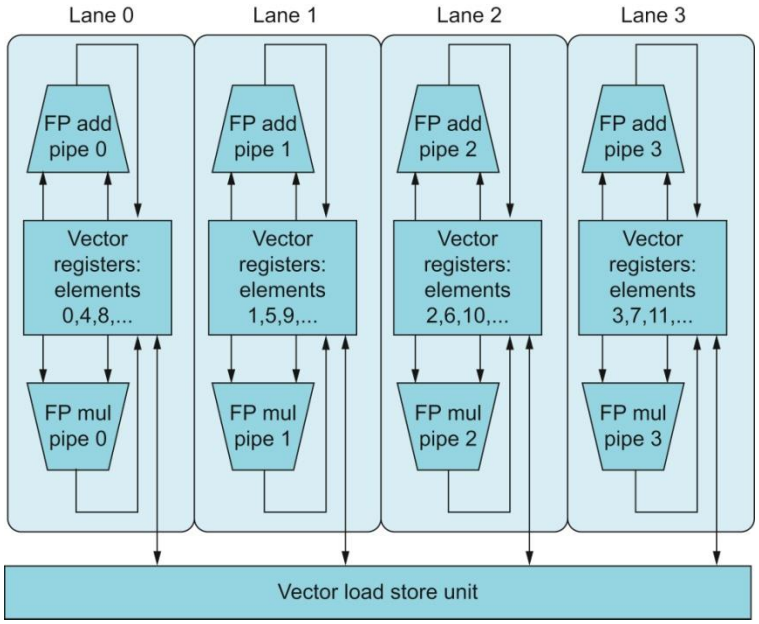


Figure 6.4 Structure of a vector unit containing four lanes. The vector-register storage is divided across the lanes, with each lane holding every fourth element of each vector register. The figure shows three vector functional units: an FP add, an FP multiply, and a load-store unit. Each of the vector 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.

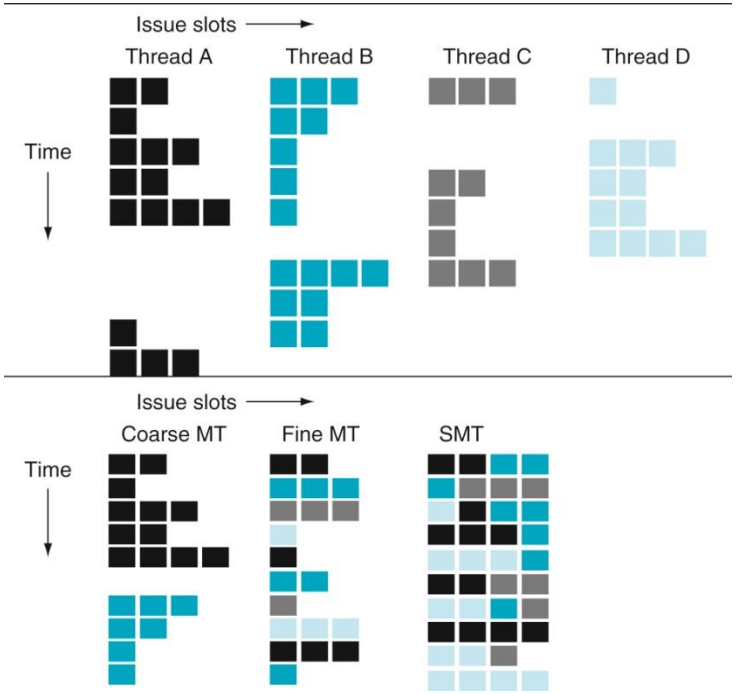


Figure 6.5 How four threads use the issue slots of a superscalar processor in different approaches. The four threads at the top show how each would execute running alone on a standard superscalar processor without multithreading support. The three examples at the bottom show how they would execute running 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.

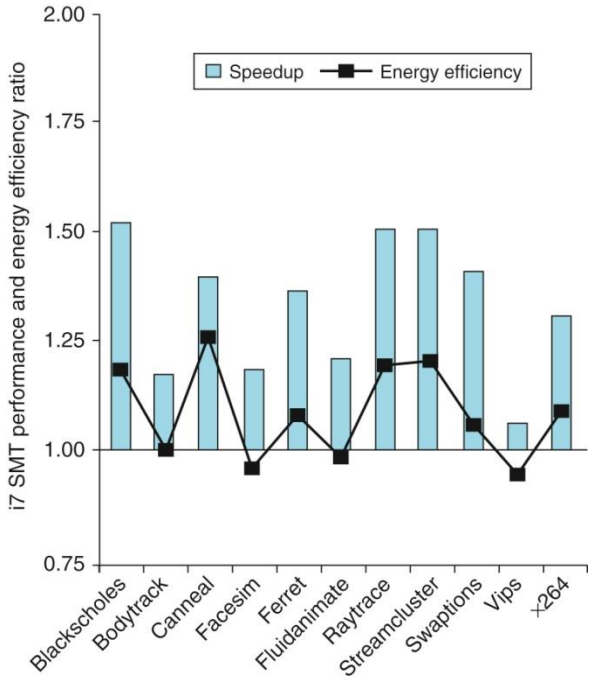


Figure 6.6 The speed-up from using multithreading on one core on an i7 processor averages 1.31 for the PARSEC benchmarks (see Section 6.10) and the energy efficiency improvement is 1.07. These data were collected and analyzed by Esmaeilzadeh et al. [2011].

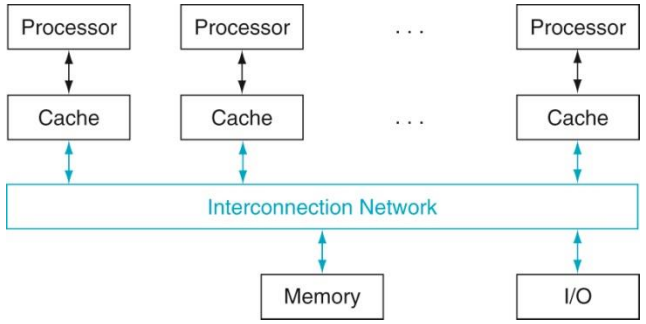


Figure 6.7 Classic organization of a shared memory multiprocessor.

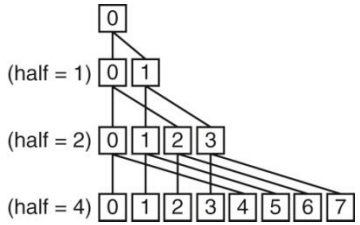


Figure 6.8 The last four levels of a reduction that sums results from each processor, from bottom to top. For all processors whose number i is less than half, add the sum produced by processor number (i + half) to its sum.

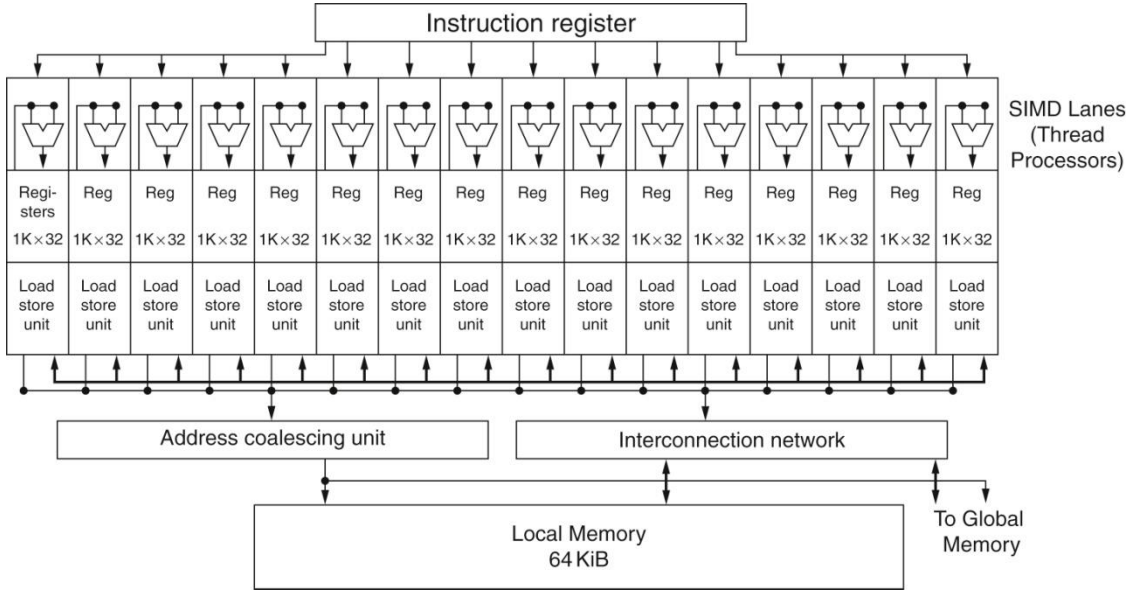


Figure 6.9 Simplified block diagram of the datapath of a multithreaded SIMD Processor.
It has 16 SIMD lanes. The SIMD Thread Scheduler has many independent SIMD threads that it chooses from to run on this processor.

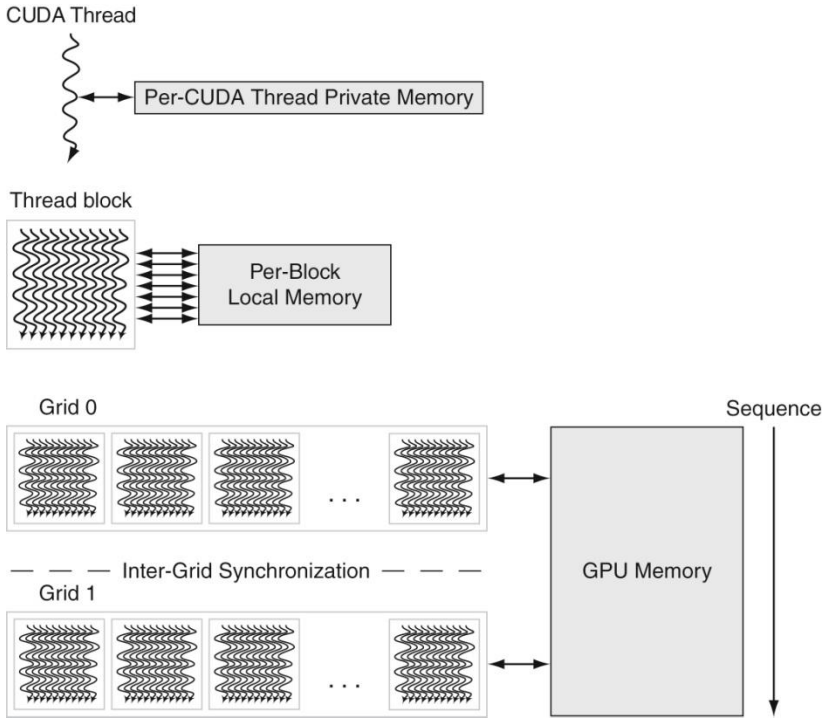


Figure 6.10 GPU Memory structures. GPU Memory is shared by the vectorized loops. All threads of SIMD instructions within a thread block share Local Memory.

Feature	Multicore with SIMD	GPU
SIMD processors	8 to 32	15 to 128
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Largest cache size	48 MiB	6 MiB
Size of memory address	64-bit	64-bit
Size of main memory	64 GiB to 1024 GiB	4 GiB to 16 GiB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Cache coherent	Yes	No

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

Type	More descriptive name	Closest old term outside of GPUs	Official CUDA/ NVIDIA GPU term	Book definition
Program Abstractions	Vectorizable Loop	Vectorizable Loop	Grid	A vectorizable loop, executed on the GPU, made up of one or more Thread Blocks (bodies of vectorized loop) that can execute in parallel.
	Body of Vectorized Loop	Body of a (Strip-Mined) Vectorized Loop	Thread Block	A vectorized loop executed on a multithreaded SIMD Processor, made up of one or more threads of SIMD instructions. They can communicate via Local Memory.
	Sequence of SIMD Lane Operations	One iteration of a Scalar Loop	CUDA Thread	A vertical cut of a thread of SIMD instructions corresponding to one element executed by one SIMD Lane. Result is stored depending on mask and predicate register.
Machine Object	A Thread of SIMD Instructions	Thread of Vector Instructions	Warp	A traditional thread, but it contains just SIMD instructions that are executed on a multithreaded SIMD Processor. Results stored depending on a per-element mask.
	SIMD Instruction	Vector Instruction	PTX Instruction	A single SIMD instruction executed across SIMD Lanes.
Processing Hardware	Multithreaded SIMD Processor	(Multithreaded) Vector Processor	Streaming Multiprocessor	A multithreaded SIMD Processor executes threads of SIMD instructions, independent of other SIMD Processors.
	Thread Block Scheduler	Scalar Processor	Giga Thread Engine	Assigns multiple Thread Blocks (bodies of vectorized loop) to multithreaded SIMD Processors.
	SIMD Thread Scheduler	Thread scheduler in a Multithreaded CPU	Warp Scheduler	Hardware unit that schedules and issues threads of SIMD instructions when they are ready to execute; includes a scoreboard to track SIMD Thread execution.
	SIMD Lane	Vector lane	Thread Processor	A SIMD Lane executes the operations in a thread of SIMD instructions on a single element. Results stored depending on mask.
Memory Hardware	GPU Memory	Main Memory	Global Memory	DRAM memory accessible by all multithreaded SIMD Processors in a GPU.
	Local Memory	Local Memory	Shared Memory	Fast local SRAM for one multithreaded SIMD Processor, unavailable to other SIMD Processors.
	SIMD Lane Registers	Vector Lane Registers	Thread Processor Registers	Registers in a single SIMD Lane allocated across a full thread block (body of vectorized loop).

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.

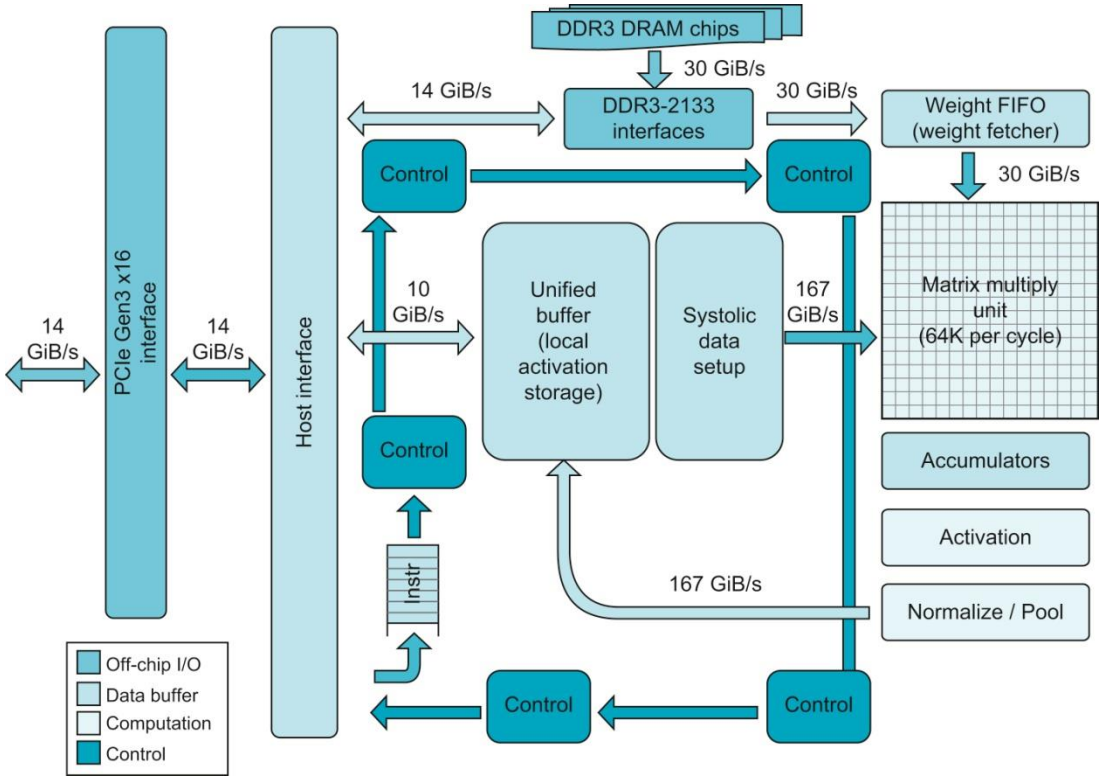


Figure 6.13 TPUv1 Block Diagram. The main computation part is the Matrix Multiply Unit (MXU) in the upper-right corner. Its inputs are the Weight FIFO and the Unified Buffer and its output is the Accumulators. The 24 MiB Unified Buffer is almost a third of the TPUv1 die, and the MXU with 65,536 multiple-accumulate 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. (Adapted from Hennessy JL, Patterson DA. *Computer Architecture: A Quantitative Approach, 6th edition,* Cambridge, MA: Elsevier Inc., 2019.)

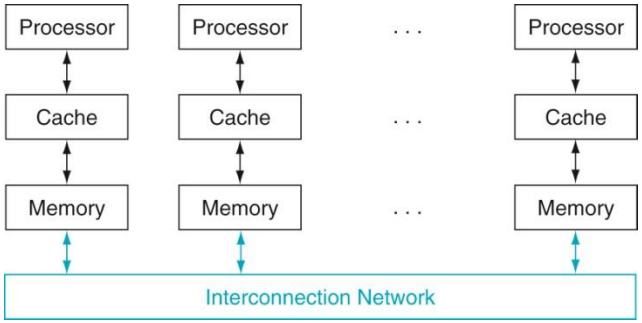
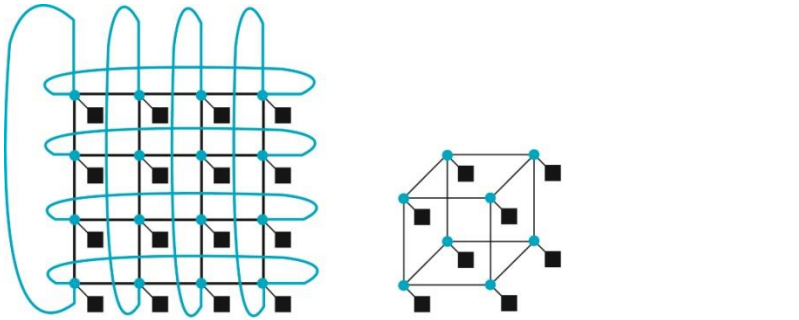


Figure 6.14 Classic organization of a multiprocessor with multiple private address spaces, traditionally called a message-passing multiprocessor. Note that unlike the SMP in Figure 6.7, the interconnection network is not between the caches and memory but is instead between processor-memory nodes.



a. 2D grid or torus of 16 nodes

b. *n*-cube tree of 8 nodes ($8 = 2^3$ so $n = 3$)

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 2n 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.

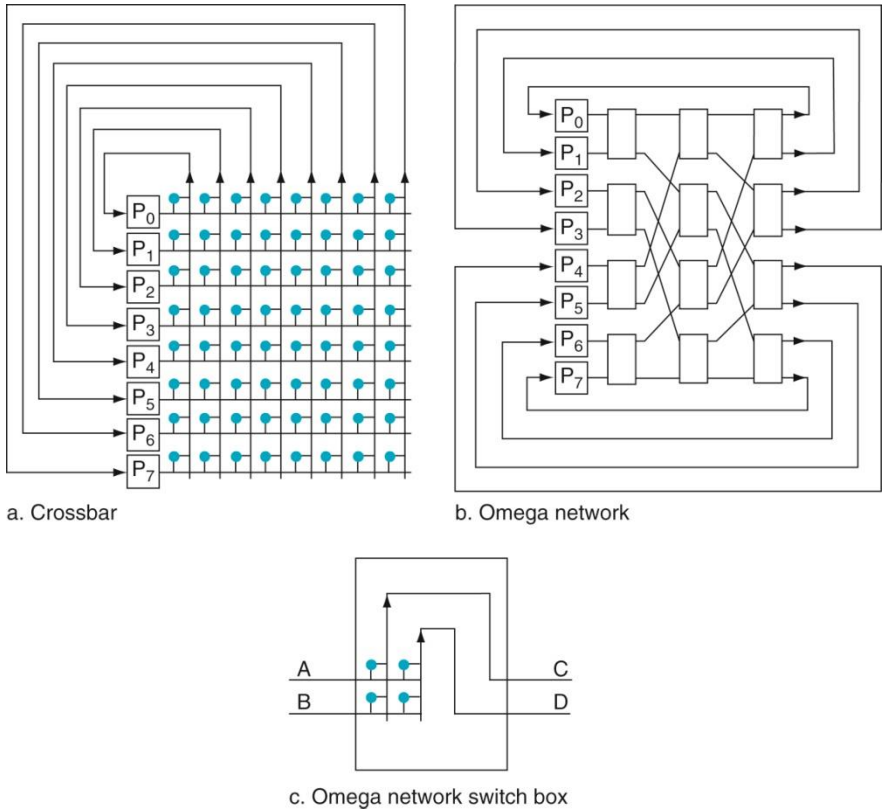


Figure 6.16 Popular multistage network topologies for eight nodes. The switches in these drawings are simpler than in earlier drawings because the links are unidirectional; data come in at the left and exit out the right link. The switch box in c can pass A to C and B to D or B to C and A to D. The crossbar 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.

Benchmark	Scaling?	Reprogram?	Description
Linpack	Weak	Yes	Dense matrix linear algebra [Dongarra, 1979]
SPECrate	Weak	No	Independent job parallelism [Henning, 2007]
Stanford Parallel Applications for Shared Memory SPLASH 2 [Woo et al., 1995]	Strong (although offers two problem sizes)	No	Complex 1D FFT Blocked LU Decomposition Blocked Sparse Cholesky Factorization Integer Radix Sort Barnes-Hut Adaptive Fast Multipole Ocean Simulation Hierarchical Radiosity Ray Tracer Volume Renderer Water Simulation with Spatial Data Structure Water Simulation without Spatial Data Structure
NAS Parallel Benchmarks [Bailey et al., 1991]	Weak	Yes (C or Fortran only)	EP: embarrassingly parallel MG: simplified multigrid CG: unstructured grid for a conjugate gradient method FT: 3-D partial differential equation solution using FFTs IS: large integer sort
PARSEC Benchmark Suite [Bienia et al., 2008]	Weak	No	Blackscholes—Option pricing with Black-Scholes PDE Bodytrack—Body tracking of a person Canneal—Simulated cache-aware annealing to optimize routing Dedup—Next-generation compression with data deduplication Facesim—Simulates the motions of a human face Ferret—Content similarity search server Fluidanimate—Fluid dynamics for animation with SPH method Freqmine—Frequent itemset mining Streamcluster—Online clustering of an input stream Swaptions—Pricing of a portfolio of swaptions Vips—Image processing x264—H.264 video encoding
Berkeley Design Patterns [Asanovic et al., 2006]	Strong or Weak	Yes	Finite-State Machine Combinational Logic Graph Traversal Structured Grid Dense Matrix Sparse Matrix Spectral Methods (FFT) Dynamic Programming N-Body MapReduce Backtrack/Branch and Bound Graphical Model Inference Unstructured Grid

Figure 6.17 Examples of parallel benchmarks.

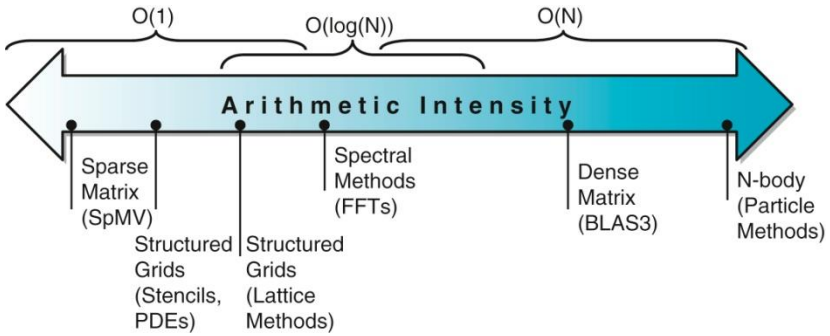


Figure 6.18 Arithmetic intensity, specified as the number of floating-point operations to run the program divided by the number of bytes accessed in main memory [Williams, Waterman, and Patterson, 2009]. Some kernels have an arithmetic intensity that scales with problem size, such as Dense Matrix, but 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.

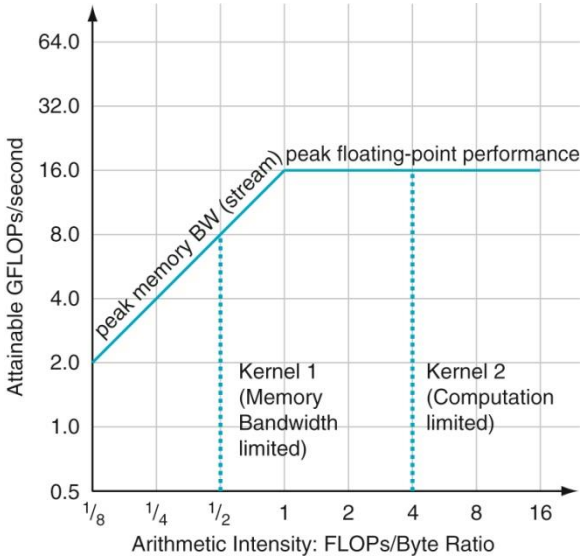


Figure 6.19 Roofline Model [Williams, Waterman, and Patterson, 2009]. This example has a peak floating-point performance of 16 GFLOPS/sec and a peak memory bandwidth of 16 GB/sec from the Stream benchmark. (Since Stream is actually four measurements, this line is the average of the four.) The dotted 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. These data are based on the AMD Opteron X2 (Revision F) using dual cores running at 2 GHz in a dual socket system.

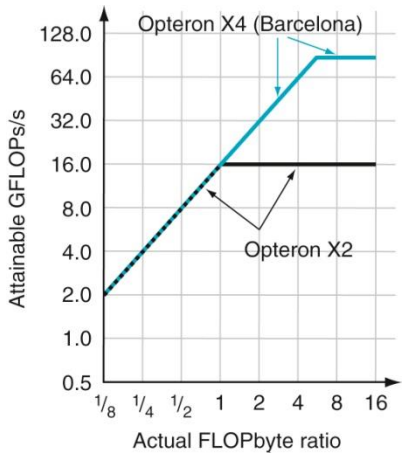


Figure 6.20 Roofline models of two generations of Opterons. The Opteron X2 roofline, which is the same as in Figure 6.19, is in black, and the Opteron X4 roofline is in color. The bigger ridge point of Opteron X4 means that kernels that were computationally bound on the Opteron X2 could be memoryperformance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

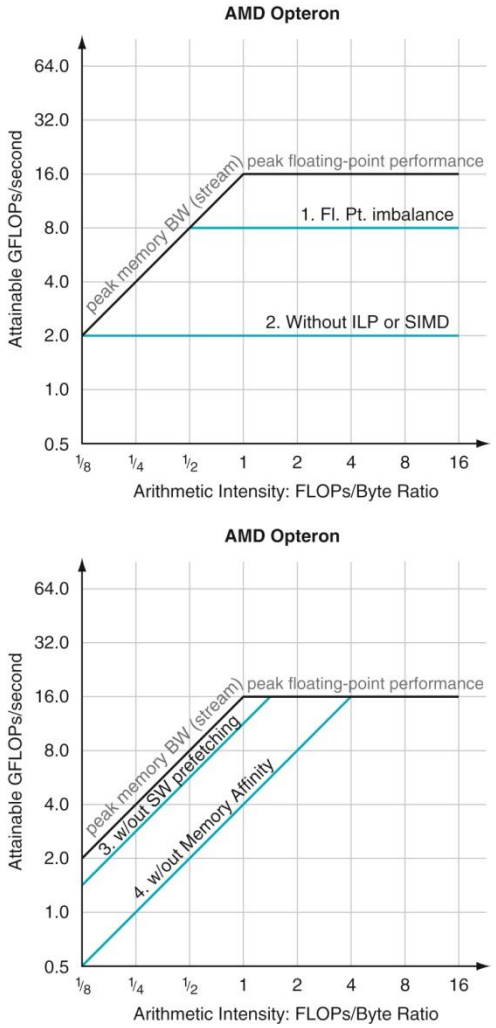


Figure 6.21 Roofline model with ceilings. The top graph shows the computational "ceilings" of 8 GFLOPs/sec if the floating-point operation mix is imbalanced and 2 GFLOPs/sec if the optimizations to increase ILP and SIMD are also missing. The bottom graph shows the memory bandwidth ceilings of 11 GB/sec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

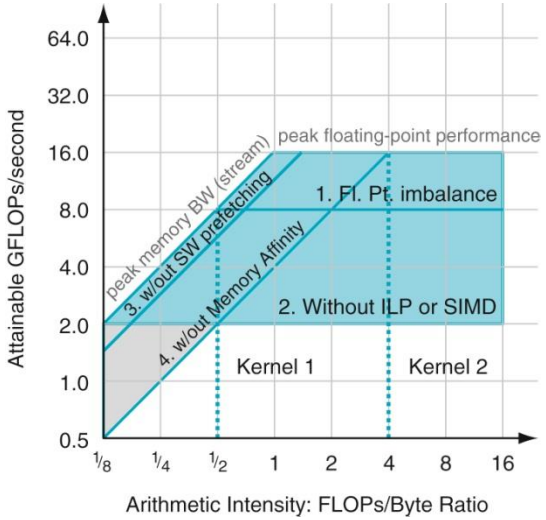


Figure 6.22 Roofline model with ceilings, overlapping areas shaded, and the two kernels from Figure 6.19. Kernels whose arithmetic intensity land in the blue trapezoid on the right should focus on computation optimizations, and kernels whose arithmetic intensity land in the gray triangle in the lower 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.

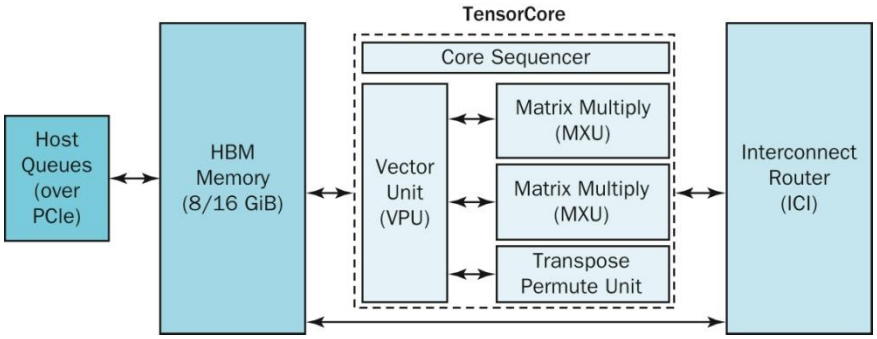


FIGURE 6.23 Block diagram of a TPUv3 TensorCore.

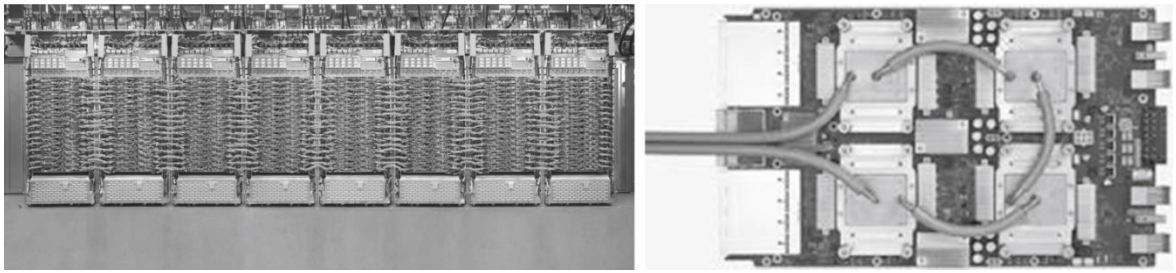


FIGURE 6.24 A TPUv3 supercomputer consisting of up to 1024 chips (left). It is about 6 feet tall and 40 feet long. A TPUv3 board (right) has four chips and uses liquid cooling.

Feature	TPUv1	TPUv3	Volta
Peak TeraFLOPS / Chip	92 (8b int)	123 (16b), 14 (32b)	125 (16b), 16 (32b)
Network links x Gbits/s / Chip	–	4 x 656	6 x 200
Max chips / supercomputer	–	1024	Varies
Clock Rate (MHz)	700	940	1530
TDP (Watts) / Chip	75	450	450
Die Size (mm2)	<331	<648	815
Chip Technology	28 nm	>12 nm	12 nm
Memory size (on-/off-chip)	28 MiB / 8 GiB	37 MiB /32 GiB	36 MiB / 32 GiB
Memory GB/s/Chip	34	900	900
MXUs / Core, MXU Size	1 256 x 256	2 128 x 128	8 4 x 4
Cores / Chip	1	2	80
Chips / CPU Host	4	8	8 or 16

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

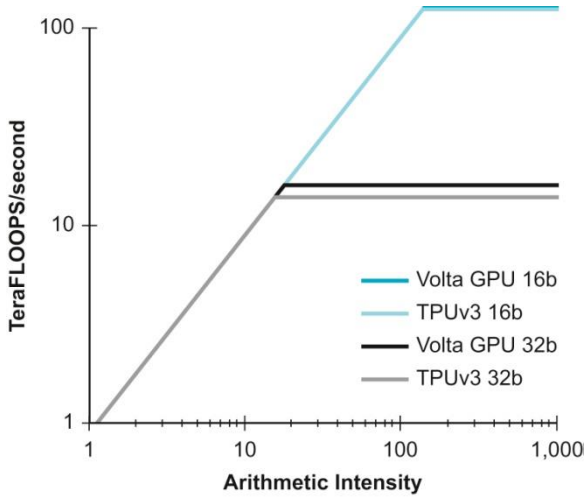


FIGURE 6.26 Rooflines of TPUv3 and Volta.

	Die size	Adjusted die size	TDP (kw)	Cloud price
Volta	815	815	12.0	$3.24
TPUv3	<685	<438	9.3	$2.00

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.

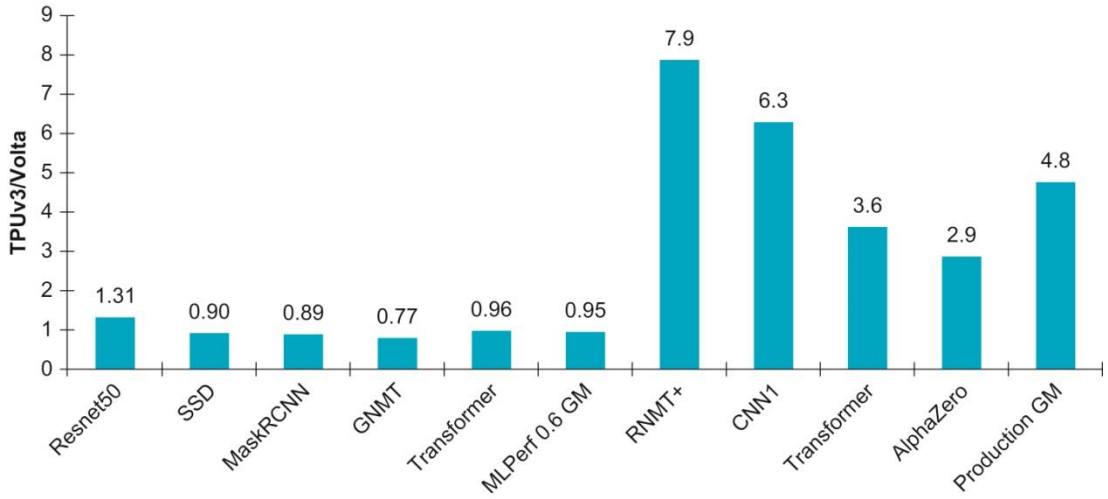


FIGURE 6.28 Performance per chip of TPUv3 relative to Volta for five MLPerf 0.6 benchmarks and four production applications.

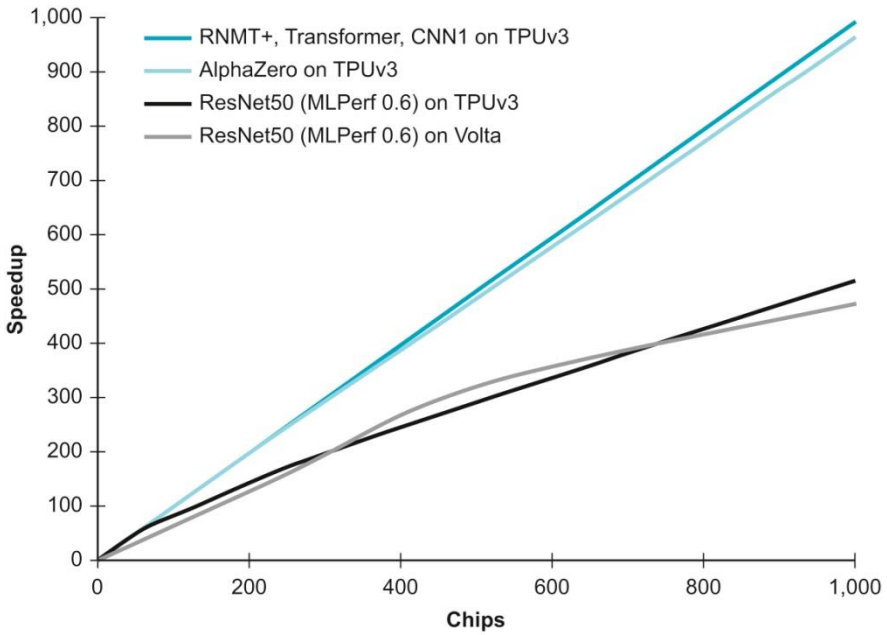


FIGURE 6.29 Supercomputer scaling: TPUv3 and Volta.

Name	Cores	Benchmark	PetaFlop/s	% of Peak	Megawatts	GFlops/Watt	Top500	Green500
Tianhe	4865k	Linpack	61.4	61%	18.48	3.3	4	57
SaturnV	22k	Linpack	1.1	59%	0.97	15.1	469	1
TPUv3	2k	AlphaZero	86.9	70%	0.59	146.3	4	1

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

```
1   #include <x86intrin.h>
2   #define UNROLL (4)
3   #define BLOCKSIZE 32
4   void do_block (int n, int si, int sj, int sk,
5                  double *A, double *B, double *C)
6   {
7     for ( int i = si; i < si+BLOCKSIZE; i+=UNROLL*8 )
8       for ( int j = sj; j < sj+BLOCKSIZE; j++ ) {
9         __m512d c[UNROLL];
10        for (int r=0;r<UNROLL;r++)
11          c[r] =  _mm512_load_pd(C+i+r*8+j*n); //[ UNROLL];
12
13        for( int k = sk; k < sk+BLOCKSIZE; k++ )
14        {
15          __m512d bb = _mm512_broadcastsd_pd(_mm_load_sd(B+j*n+k));
16          for (int r=0;r<UNROLL;r++)
17            c[r] = _mm512_fmadd_pd(_mm512_load_pd(A+n*k+r*8+i), bb, c[r]);
18         }
19
20       for (int r=0;r<UNROLL;r++)
21         _mm512_store_pd(C+i+r*8+j*n, c[r]);
22       }
23    }
24
25  void dgemm (int n, double* A, double* B, double* C)
26  {
27  #pragma omp parallel for
28    for ( int sj = 0; sj < n; sj += BLOCKSIZE )
29      for ( int si = 0; si < n; si += BLOCKSIZE )
30        for ( int sk = 0; sk < n; sk += BLOCKSIZE )
31          do_block(n, si, sj, sk, A, B, C);
32  }
```

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.

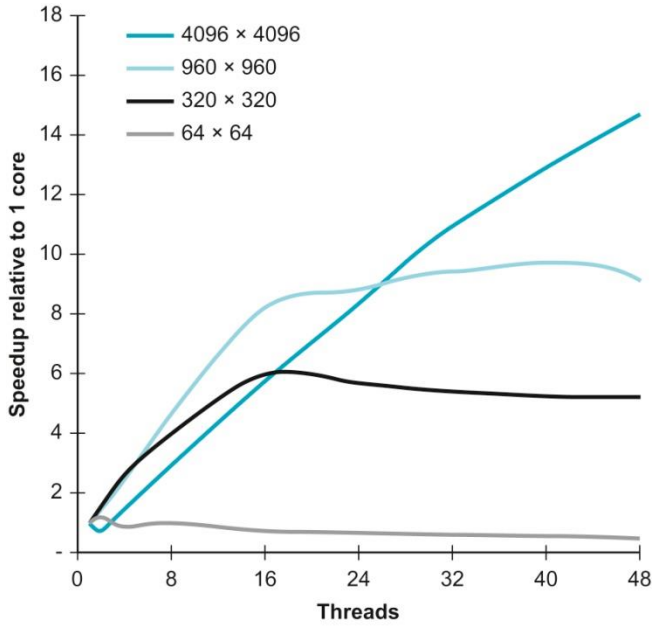


Figure 6.32 Performance improvements relative to a single thread as the number of threads increase. The most honest way to present such graphs is to make performance relative to the best version of a single processor program, which we did. This plot is relative to the performance of the code in Figure 5.47 *without including OpenMP pragmas.*

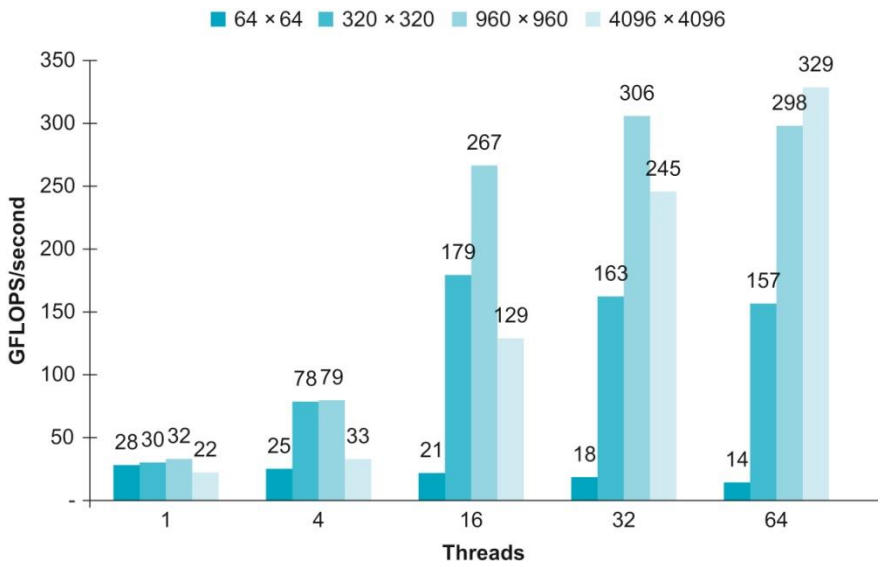
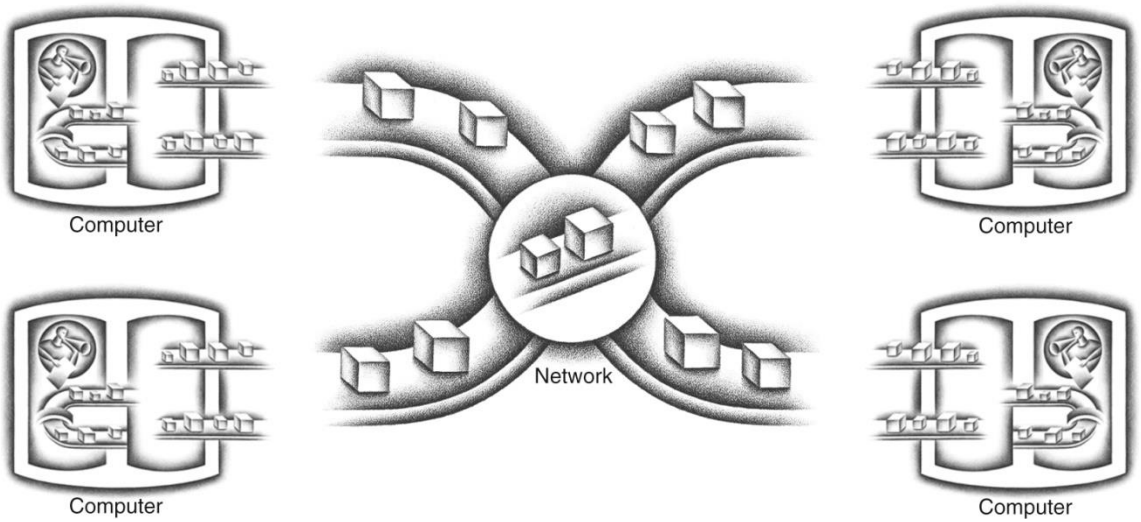
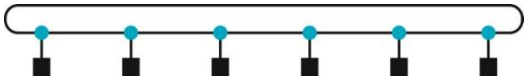


Figure 6.33 DGEMM performance versus the number of threads for four matrix sizes.
The performance improvement compared to the original C code in Figure 3.20 for the 960 × 960 matrix with 32 threads is an astounding 152 times faster!.





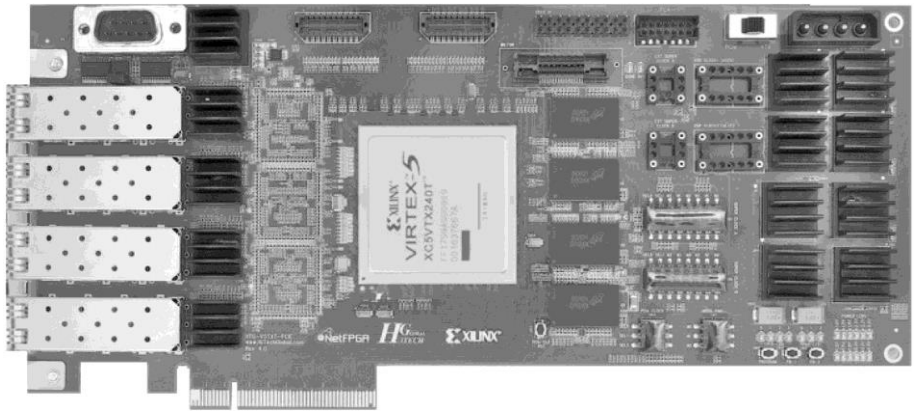


FIGURE e6.10.1 The NetFPGA 10-Gigabit Ethernet card (see http://netfpga.org/), which connects up to four 10-Gigabit/sec Ethernet links. It is an FPGA-based open platform for network research and classroom experimentation. The DMA engine and the four “MAC chips” in Figure e6.9.2 are just portions of the Xilinx Virtex FPGA in the middle of the board. The four PHY chips in Figure e6.9.2 are the four black squares just to the right of the four white rectangles on the left edge of the board, which is where the Ethernet cables are plugged in.

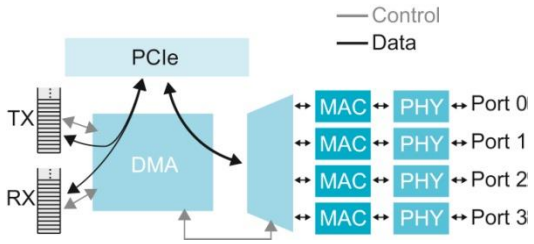


FIGURE e6.10.2 Block diagram of the NetFPGA Ethernet card in Figure e6.10.1 showing the control paths and the data paths. The control path allows the DMA engine to read the status of the queues, such as empty vs. on-empty, and the content of the next available queue entry. The DMA engine also controls port multiplexing. The data path simply passes through the DMA block to the TX/RX queues or to main memory. The “MAC chips” are described below. The PHY chips, which refer to the physical layer, connect the “MAC chips” to physical networking medium, such as copper wire or optical fiber.

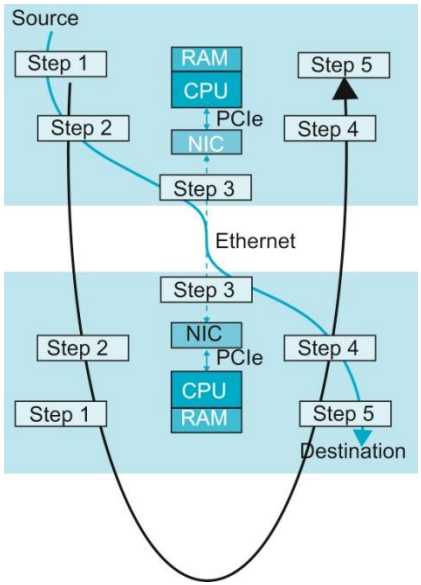


FIGURE e6.10.3 Relationship of the five steps of the driver when transmitting an Ethernet packet from one node and receiving that packet on another node.

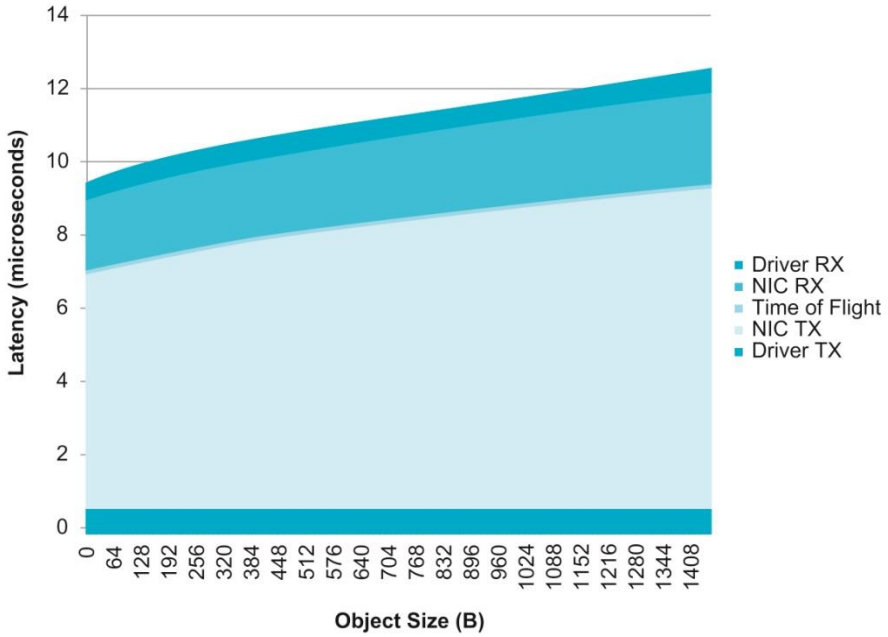


FIGURE e6.10.4 Time to send an object broken into transmit driver and NIC hardware time vs. receive driver and NIC hardware time. NIC transmit time is much larger than the NIC receive time because transmit requires more PCIe round-trips. The NIC does PCIe reads to read the descriptor and data, but on receive the NIC does PCIe writes of data, length of data, and interrupt. PCIe reads incur a round trip latency because NIC waits for the reply, but PCIe writes require no response because PCIe is reliable, so PCIe writes can be sent back-to-back.

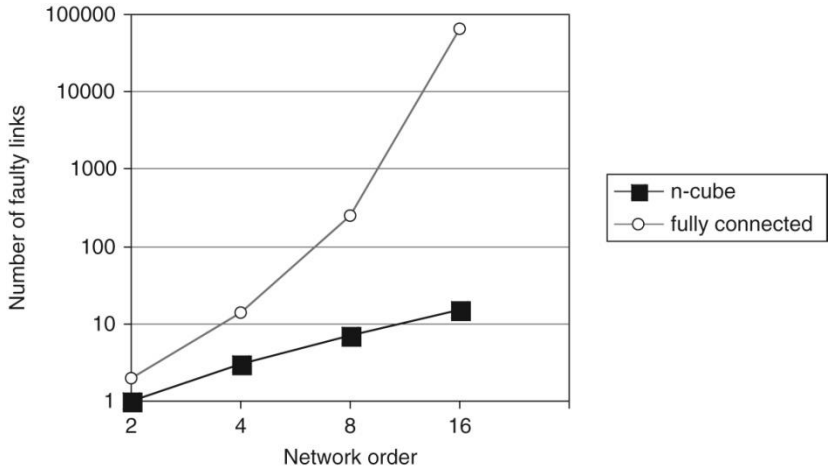
FIGURE e6.16.1 The Illiac IV control unit followed by its 64 processing elements. It was perhaps the most infamous of supercomputers. The project started in 1965 and ran its first real application in 1976. The 64 processors used a 13-MHz clock, and their combined main memory size was 1 MB: 64 × 16 KB. The Illiac IV was the first machine to teach us that software for parallel machines dominates hardware issues. *Photo courtesy of NASA Ames Research Center.*

Core 1	Core 2
A3	B1, B4
A1, A2	B1, B4
A1, A4	B2
A1	B3

.

FU1	FU2
A1	A2
A1	
A1	
B1	B2
B1	
A3	
A4	
B2	
B4	

FU1	FU2
A1	B1
A1	B1
A1	B2
A2	B3
A3	B4
A4	



Latency	Max TP rate	Avg. # requests per core
1 ms	5000/sec	1.25
2 ms	5000/sec	2.5
1 ms	10,000/sec	2.5
2 ms	10,000/sec	5