Chapter 6

Great *and* Big Ideas in Computer Structures

Gordon Bell

ED - ?
and Big (OK?)
or: and Big (?)
MS: unclear

AU - ?
Please add your affiliation.

DEDICATION

I went to Carnegie Tech after spending my first 6 professional years designing computers at Digital Equipment Corporation to be part of the startup faculty of the computer science department. Allen Newell, Alan Perlis, and Rod Williams, who headed the Electrical Engineering Department, made this career changing choice easy for me.

Like everyone who knew Allen, I feel deeply honored to have known him and was influenced by him in many ways. Allen was the most thoughtful, kind, and gentle gentleman I know. His intellect, coupled with his enthusiasm and smile, virtually always led a group in the right direction. He was the role model for a scientist, teacher, husband, father, and person. I cite Allen as "my ideal" when engineers ask me for career advice because Allen maintained a constant enthusiasm for work, and doing science. He saw no higher state of being. We worked together to research and write *Computer Structures: Readings and Examples* (Bell & Newell, 1971), which was a classic for almost 20 years, or about two hardware generations. We developed the processor-memory-switch (PMS) and instruction-set processor (ISP) notations for describing computers. Although thousands of computers have been introduced since 1971, only a few basically new ideas for computers have emerged since *Computer Structures*. Our taxonomic framework has proved to be equally descriptive of computing alternatives, in-

ED: 1 LINE SHORT

cluding the massively parallel computers of the 1990s. Designing taxonomies to generate alternatives still remains my favorite approach for design. This fascination with taxonomies is based on their ability to take a number of artifacts, and relate them in a common framework that shows attributes and relationships. This is a form of generality—one mechanism serves many functions.

Design systems using the PMS and ISP notations were built in the early 1970s by Haney, Knudsen, Siewiorek, and Barbacci, and these influenced languages for simulating digital systems. *Computer Structures: Principles and Examples* (Siewiorek, Bell, & Newell, 1982) further refined and utilized the notations. Our interest in design helped motivate the framework for register transfer level design. We also used the design framework for *Designing Computers and Digital Systems Using PDP16 Register Transfer Modules* (Bell, Grason, & Newell, 1972). For a variety of reasons, the book is hardly known, but everyone who read it was enthusiastic about a book on design. This register transfer level approach allowed Barbacci to build the first computer design system. Commercial systems are just beginning to synthesize digital systems.

I wanted to write this chapter to interact with Allen about what's changed in *Computer Structures* (Bell & Newell, 1971; Siewiorek et al., 1982). *Computer Structures III* hasn't progressed very far because its focus is unclear, given the basic need for a comprehensive architecture book has been satisfied (Hennessy & Patterson, 1990). Being at Carnegie and working with Allen was the most happy and productive time for me. I certainly learned the most. I can only hope the patient time away from Allen's main line of research to work on low-level machines with me was as worthwhile for him as it was for me. I hope these comments will be worthwhile to you.

APPROACH

This chapter combines an approach to explaining the evolution of computer structures that I hope Allen would have enjoyed, but lacks the benefit of his interaction. The chapter has six parts, describing the two revolutionary ideas, the principle of locality, and three sets of design principles. The idea is to show that computer structures have evolved based on just these ideas. The two really big, revolutionary ideas of this century, and possibly of all time, are the stored program computer and the integrated circuit, the crude oil used to implement computers. The principle of locality is a behavioral phenomenon of programs on which many computer mechanism are based. Three general design principles explain many computer structures variants: replicatability provides increased performance,

ED: 1 LINE SHORT

TABLE 6.1
Two Great Ideas, Locality, and Three Design Principles

1. The stored program computer is one of the great inventions. The variants:
   a. Computers provide an unlimited number of functions and applications—they provide control, memory, switching and processing of information.
   b. Every level of a computer hierarchy is built on a lower level—thus, hardware may be traded for software and vice versa (first two to three levels).
   c. One computer can provide an environment for many computers for multiprogramming and timesharing.
   d. A set of distributed computers can behave as one computer, given enough software.
2. The integrated circuit invention and evolution is equally as important as the computer.
   a. One more bit is needed every 18 months to address physical memory.
   b. A hierarchy of memories exist and can exploit locality and fill economic needs.
3. Principle of locality: Temporal and spatial locality are a property of computation.
   a. Members of the memory hierarchy (e.g., the cache) can exist.
   b. Multiple processors can exist as scalable computers
   c. Distributed workstations (i.e., scalable mCs) can behave as a single system.
4. Replication design principles: Replication provides parallelism for performance, spatially distributed computing, and redundancy for higher reliability.
   a. Replicatability within a computer generates multiprocessors and the opportunity and need for parallel processing
   b. Replicatability of computers generates multicomputers and provides an opportunity and need for them to be used as a single resource
5. Economics design principles: A somewhat rational market based on economic utility, determines the form and function of computation, not elegance, religion, architecture, etc.
   a. The investment in software and computer families motivated the 360 family, the VAX hierarchical computing environment, Intel X86, and scalable computers.
   b. A computer system must be balanced across processing, the memory hierarchy, terminal access including visualization, and networking.
   c. *Amdahl's law* of diminishing returns for fast and slow, serial and parallel use.
   d. Computer size, generation, and problem scalability.
6. Generality design principle
   a. Generality allows resources to be bound late and flexibly (e.g., computation, memory, and bandwidth, as well as hardware and software trade-offs occur).

reliability, and allows physical distribution; economics constrain and provide the objective function for the world, including computer structures; and good designs are based on finding general-purpose mechanisms. These six ideas and the variants of them is given in Table 6.1.

ED-? These six ideas ... are given in

Table 6.2 gives what I believe to be the two great ideas, some big and good ideas together with the machines that embodied the principles. I refrain from enumerating the bad ideas, but these would be instructive. Before looking at the ideas, it's important to note the difference between revolution and evolution.

ED: 1 LINE SHORT

TABLE 6.2
The Great, Big and Good Ideas, and Computers That Embody Them

1945	von Neumann's Edvac (Draft) Report and IAS machines or "x"-iacs (the first computers were Manchester Mark I and Cambridge EDSAC)
1950	Commercial computer (Univac) that stimulated others (e.g., IBM).
1960s	Second generation—the "one level store" and subsequent virtual memory (Atlas)
	Minicomputers for control, switching, etc. (PDP-5 > PDP-8)
	Architecture and computer families (360)
	Multiprogramming and timesharing (B5000, IBM/360, PDP-6/10)
	Multiprocessors (1–4) for reliability and performance (Burroughs, DEC, IBM, Univac)
	Cache memory using solid-state memories (IBM/360 Model 85)
1970s	Third generation, hundreds of minicomputers using integrated circuits
	Packet switching (ARPAnet)
	First microprocessor [4004 > 8008 > 8080 · · · 80x86 (the PC)]
	Vector processor supercomputer (Cray 1)
	Fault-tolerant smC (Tandem)
	LAN-based distributed computing (at Xerox Parc)
1980s	Fourth generation—powerful CMOS microprocessors and distributed, LAN-based computing
	Hierarchy of computing facilities (VAX Homogeneous Environment)
	32-bit CMOS microprocessors for WSs using LAN distributed process (Apollo and Sun)
	Intel CMOS micros for PCs (X86s) > LAN-based computing
	Multis (small scalable mPs with 1–20 processors) (Arete, Encore, Sequent, etc.)
	RISC (HP, IBM, MIPS, and Sun)
	Scalable multiple computers (e.g., hypercubes, Transputers, and switches for smCs)
	Parallel thread execution by a multiprocessor (Alliant, Ardent, Convex, Cray)
1990s	Fifth generation . . . and the disappearance of the computer
	Size- and generation-scalable mPs (KSR)

REVOLUTIONS IN TECHNOLOGY: THE COMPUTER AND THE INTEGRATED CIRCUIT

Figure 6.1 shows two models of progress (Gomory & Schmitt, 1988). One model is a "ladder" of scientific revolution based on important milestones in computer technology, whereas the other is a "wheel" of evolution based on continuous refinement of a basic design and process. The "rungs" of scientific revolution include the introduction of the stored-program computer (circa 1946), and the dates given are for the introduction of a particular technology into computers, not for the initial availability. For example, vacuum tubes were used in radios long before 1944. The most interesting aspect of the ladder is that it shows no computer-technology revolutions since the introduction of integrated circuits in 1967. Although optical technology (now used in communications) may eventually find its way into computers, products based on this technology are unlikely to appear during the 1990s, because there's a substantial delay between labo-

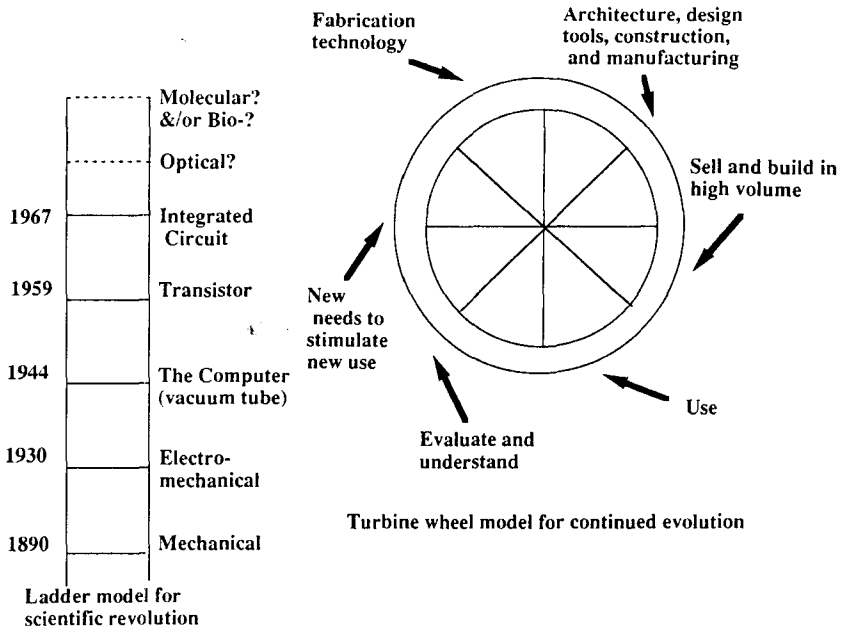

FIG. 6.1. Revolutionary (ladder) and evolutionary (wheel) models of technology progress.

ratory development and product introduction. I quantify the typical laboratory-to-product delay time like this: It takes roughly a decade (or one technology generation) from the time a significant laboratory invention occurs until it has significant use in products. Carver Mead believes major revolutions in semiconductors have occurred every 12 years: for example, the transistor (1947), integrated circuit (1959), microprocessor (1971), and silicon compiler (1984). In addition to the delay from first use to commercial exploitation, another decade may be required for wide scale assimilation. Although companies always use *revolution*[1] to describe new products or developments, evolution is a more realistic word, because progress is generally based on well-established technology and a set of design principles. In particular, circuit and memory technologies (i.e., the technologies involving the physical components that actually process and store information) are the key determinants of a computer's performance and cost, and during the past 20 years, progress in these technologies has been considerably more evolutionary than revolutionary. Unfortunately for the United States, which excels in the few revolutionary inventions, Japan excels in industrial evolution.

[1]A revolution should be a significant "leap" that produces an even more significant benefit.

The cycle of evolution in computer technology is driven by the interaction of many processes. New basic materials and circuits, along with advances in fabrication technology, make possible new architectures and new ways of producing the next computer. The process of selling, building in higher volumes, using, evaluating, and understanding computers raises aspirations for the next cycle of evolution. Some of these factors involve computer manufacturers, some involve users, and a few involve computer science. With the advent of increased capabilities comes the discovery of new uses and needs, which unleashes more funds to fuel the next cycle. As we enter an era of declining computer prices and profits, this cycle could undergo change because industry lacks the slack resources to search for new ideas!

THE STORED-PROGRAM COMPUTER FROM A TO Z

Virtually everyone who proposed or built an early calculator aided in the invention of the stored-program computer. Although this is not another history of the invention, it is almost safe to say that Aiken, Atanasoff, Burks, Eckert, Mauchley, von Neumann, Stibitz, Turing, and Zuse were major contributors, along with Kilburn, Williams, and Wilkes, who made the first operational computers in the United Kingdom. Turing's paper on computability planted the idea that computation was a "machine-like" process. A physical machine that could be built to compute followed. Several calculators were built in World War II to aid defense. von Neumann's name is synonymous with the stored-program concept because he drafted the first paper describing the EDVAC architecture for a working group that included Eckert and Mauchley. von Neumann also authored the paper with Burks and Goldstine (Bell & Newell, 1971) that described an architecture for a computer that was replicated at research organizations throughout the world. The economics design principle requiring compatible computers so that all sites could share and build on each other was invented just after the computer. In fact, British manufacturers made copies of the machines designed at Cambridge and Manchester Universities and the National Physical Laboratory for the software benefits.

Computers Are Universal Across a Broad Range of Functions, Applications, and Sizes

Because computers are universal finite-state machines, they can be used to simulate just about any other information-processing system functions (processing, memory, switching, control, transduction) that we understand well enough to program. They substitute for and supplement other information processors, including people. They can be used to simulate virtually anything that can be described as a procedure or collection of processes or

rules (thanks to Allen). Of course, the many varieties in size and function are a result of integrated circuit evolution permitting new applications, not the computer (Fig. 6.2).

Computers Are Universal, Allowing Hierarchies of Computers to Be Built

The most important aspect of a computer is that other computers can be built on another computer in a hierarchical fashion to create more complex, higher level computers starting from basic hardware interpreter. A system of arbitrary complexity can be built in a fully layered fashion with each layer building on a lower level layer. The usual levels are micromachine—especially if microprogramming is used; hardware or instruction-set processor (ISP), but ISA (architecture) is used now; operating system; system programming language; higher level language; and an application programming language such as those used in word processors, databases, or spreadsheets.

Using this principle, a designer can make a decision about the appropriate level (e.g., hardware vs. software) to implement a given function. Paraphrasing, Perlis' "One man's constant is another man's variable"—"one person's system is another person's component." From a computer structures perspective, the hardware–software trade-off ability is the most important result of having all the levels. Frequently executed operations are placed in hardware in order to get a factor of 2–50 (depending on the operation), and we leave in software those operations that might change or that we don't understand (either because it's a new use, or an architect is uninformed). Some changes have been to place floating-point arithmetic in hardware, including vector operations, and when fast reduced instruction set computer (RISC) processors became available in the mid 1980s to move hardware functions, including memory management and infrequently executed operations, to software. One consequence is that the instruction-set architecture is covered up by the programming language . . . and usually, unless compatibility is a constraint, "speed beats elegance," although I would like to believe elegant designs are simpler, and go faster.

Computers Are Universal and One Computer Can Simulate Many Computers

A most important property of computers is first to allow computers to be multiprogrammed and hold, that is, host, a variety of independent machines so that one physical computer can do a variety of independent or related tasks. In the 1950s, the interrupt was invented to allow a machine to compute and process input/output (I/O) concurrently. In the mid 1960s multiprogramming was invented to allow multiple batch job streams with concurrent

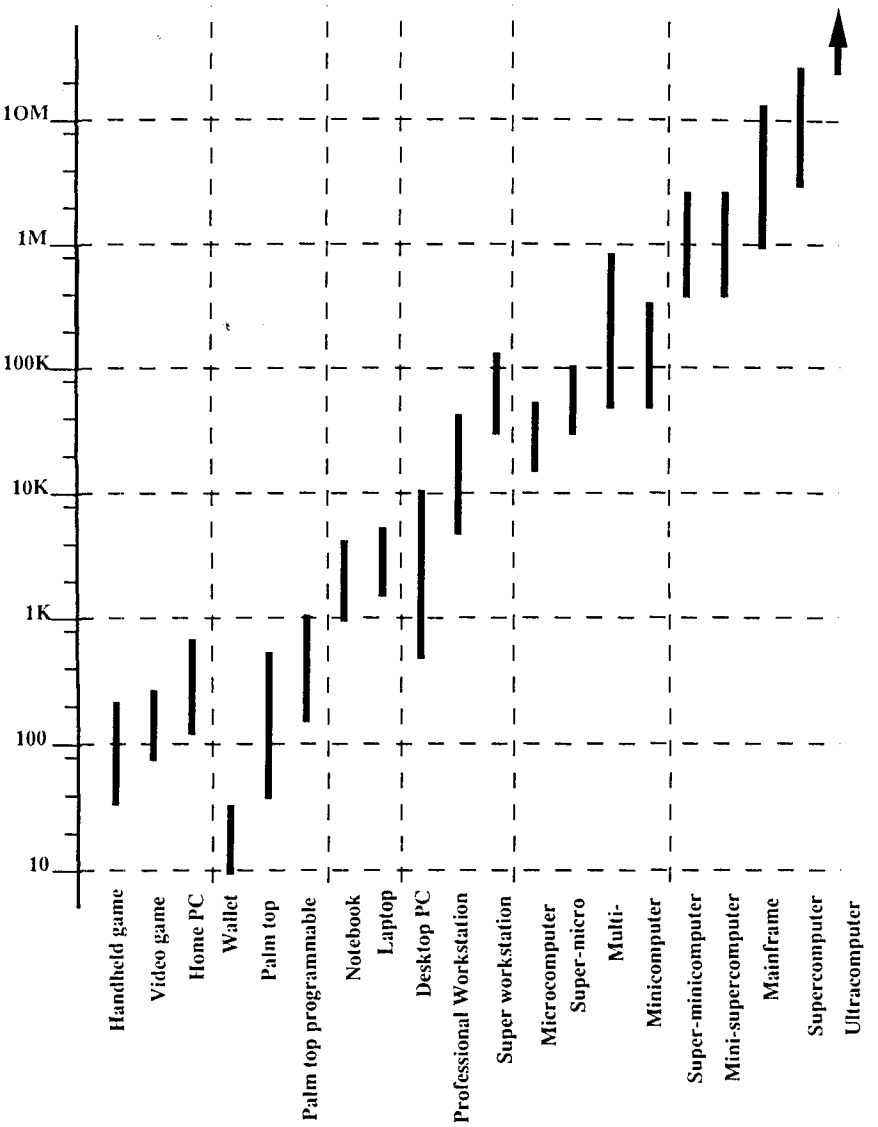

FIG. 6.2. Computer classes/prices in 1992.

I/O. Timesharing was invented to give a virtual computer to everyone who used it.

Concurrent with providing an environment that could hold and share multiple, independent programs, the principle of locality permitted programs to be located across parts of the memory hierarchy as described below. In 1960, Atlas' One-Level provided a large virtual memory using a

relatively small physical memory as the first example of caching based on working sets (Denning, 1980). The paging mechanism was elegant because it also allowed programs and data to be shared among independent processes. Similarly, large addresses allows addresses to be used as name spaces.

The programming model of a single memory for multiple, large programs, including the operating system (O/S), provides the ultimate generality/flexibility because it provides a single pool of sometimes fungible resources. This motivation is critical to allowing many processes to share data and programs with one another. By the late 1980s multiple processors could operate concurrently on parallel threads of a single program, providing the first indications that fine-grain parallel processing of a single program was possible.

In 1992, massively parallel, scalable multiprocessors and multicomputers appeared with identical functionality, differing only in the underlying operations that are carried out in hardware (the multiprocessor) versus software (the multicomputer). In both cases, the end goal is a scalable multiprocessor that supports a single, multiprogrammed virtual memory environment.

ED-7
Computers Are
^

Computer Are Universal and Many Distributed Computers Can Behave as One

It has always been relatively easy to interconnect computers physically, but the challenge has been to make them operate as one. This ability to aggregate computers to accomplish a common set of tasks that provide the appearance of a single, shared virtual memory computer emerged from three directions:

1. Tandem pioneered fault-tolerant computers composed of multiple computers interconnected via high-speed links where the same program is executed in two machines, and messages are sent to update the state of a program and synchronize operations. As a by-product, such a computer can be scaled over a wide range of size and be spatially distributed.

2. Distributed computing came from Xerox PARC's 1970s work and Datapoint's ARCnet. Products emerged in the early 1980s from Apollo and Sun Microsystems (Bell & McNamara, 1991) in the form of workstations. Initially computers were distributed so that each person had their own, but the trick was getting them back together in order to appear like a single timesharing system with shared files and printers, and so forth. Distributed personal computers (PCs) trailed workstations by approximately 6 years, or a half generation.

3. Scalable multicomputers using hundreds of fast, CMOS microprocessors to provide a high performance computer system were pioneered at Cal Tech and Intel, and in Europe using the Transputer. In 1992, dozens

of companies lash together microprocessors in order to achieve a wide range of scalable power from 100 Mflops to over 100 Gflops. Several thousand multicomputers were built in the 1980s for problem-specific applications. Workstations become multicomputers as the performance of individual nodes increased and could be interconnected at high data rates in the mid-1990s. Furthermore, software systems of various kinds, including compilers that automatically parallelize a program, are essential.

SEMICONDUCTOR EVOLUTION: COMPUTERS (AND MEMORIES), IN ALL SHAPES AND SIZES

While many developments have permitted the computer to evolve rapidly, the most important evolution was density increases in semiconductors and magnetics. Although improvements in these technologies have been evolutionary (i.e., conforming to the "wheel" model in Fig. 6.1), their impact on computer architecture and applications has paved the way for revolutionary changes (i.e., conforming to the "ladder" model) in those areas. At the present rates of progress in semiconductors and magnetics, the cost of hardware for computers of the type and size commonly used in 1990 will be near zero by the end of the century. Semiconductor people often make the analogy that "If cars evolved at the rate of semiconductors, today we would all be driving Rolls Royces that go a million miles an hour and cost $0.25." The difference lies entirely in the technology: Maxwell's equations governing electromagnetic radiation, which moves at the speed of light, versus Newton's laws governing the motion of objects with mass, which move at the speed of sound.

The integrated circuit was invented in 1958, the year when discrete transistors first started being used in computers. Every year from 1958 until about 1972, the number of transistors per die doubled. Starting in 1972, the number began doubling only every year and a half, or increasing at roughly 60% per year, resulting in a factor of 100 improvement each decade. Intel's founder and chairman, Gordon Moore, posited two laws based on this phenomenon:

Moore's law (1964): The density of chips doubles every year.

Moore's law (1975): The density of chips doubles every 1.5 years.

The use of memory circuits that require only one transistor per bit stored (plus some capacitance) have made bits per chip rather than transistors per chip a more interesting measure, but density has continued to quadruple every three years. This 3-year pattern is illustrated by these statistics on the number of bits per chip and the year in which each chip was introduced: 1K (1972), 4K (1975), 16K (1978), 64K (1981), 256K (1984), 1M (1987), 4M (1990), 16M (1993). This equation describes the evolution:

$$\text{Number of bits/chip} = 1\text{K} \times 2(t - 1972)/1.5$$

This trend seems likely to continue until the year 2000, when extrapolation suggests that a single memory chip will store 256 million bits. In 1995 Semetech extended the forecast to 2010. The 256-million-bit figure may be slightly optimistic, however, because Meindl (1987) predicted that growth will slow down from 60% to between 20 and 35% beginning in 1992–1998. However, Meindl saw 20–35% growth persisting for another 20 years, in which case, a single die will store between 1 trillion and 100 trillion bits. The increase in density has allowed computers to operate faster while costing less, because of the following two rules:

1. The smaller everything gets, approaching the size of an electron, the faster the system behaves.
2. Miniaturized circuits produced in a batch process tend to cost very little to produce after the factory is in place.

The cost impact of the increased densities shown in Fig. 6.3 is reflected in changes in the relative cost of computing in various computer classes. For scientific computing the cost has declined over four orders of magnitude

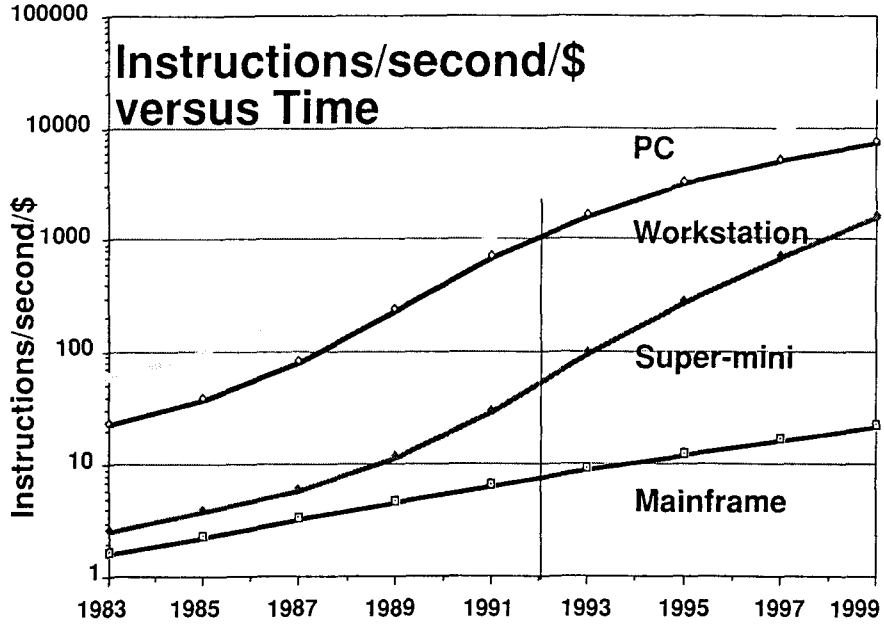

FIG. 6.3. Estimate of instructions processed per dollar versus time for various computer classes (Courtesy of the Gartner Group).

during the 1950–1990 period, representing a price drop of 20% per year. In 1992, a supercomputer supplied 500–1,000 floating-point operations per second (flops) per dollar spent on the computer, whereas a workstation or scalable multicomputer provided about $5,000 flops/$. The cost per operation is likely to decline substantially more quickly over the next decade as semiconductor densities increase and commodity chips create strong competition.

Not all of the cost benefits of increased memory chip density have translated into a reduction in system cost. Some of the cost benefits have translated into larger memories, because the advances permit a given computer to have more memory for a constant price. In the 45-year period beginning in 1945, primary memories for a large computer have grown by about six orders of magnitude (2 Kbytes to 2 Gbytes), representing an increase in size at the rate of 35% per year. Figure 6.4 shows how various constant size–constant cost memories grow with time. Notice that in 2000, the memory for a current workstation (WS) or PC only occupies a fraction of a chip, and the memory portion would sell for about $10.

Processor evolution has been in response of exploiting exponential improvements in circuit/packaging technology: mainframes, minicomputers, supercomputers, and microprocessors that enabled PCs, WSs, and

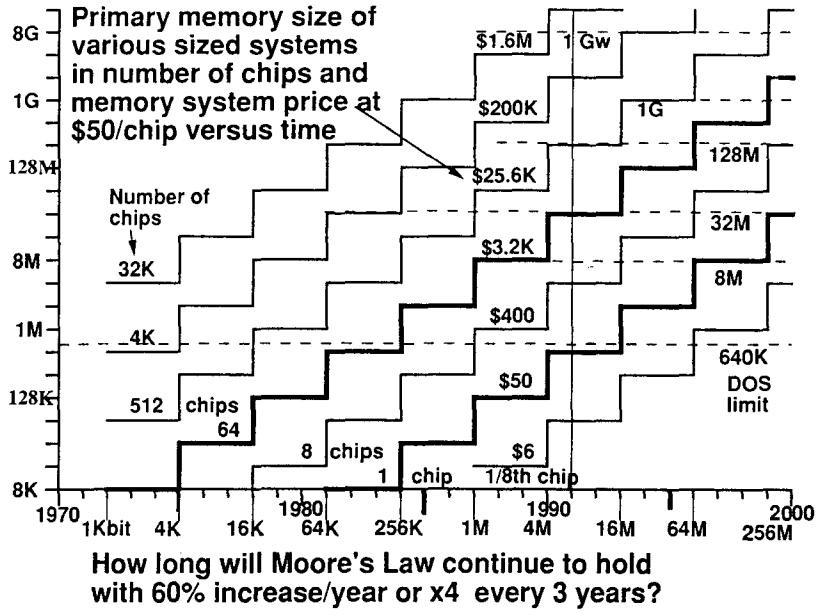

FIG. 6.4. Evolution in memory size for various constant cost–constant numbers of memory chips with time.

scalable multiprocessors. RISC microprocessors versus microprogrammed computers evolved in response to shifts in memory types, sizes, and costs.

In summary, the semiconductor density evolution has been extremely dramatic. It has spawned entire new classes of computers, new computer systems, new companies, new application opportunities, and new industries. All of the historical computer classes based on price continue to exist once established because of the need to maintain software and data in these "code museums." However, as high-performance personal computers emerge that have the power of yesteryear's mainframe or minicomputer, the number of more expensive computers will decline in a "downsizing." Most likely, this phenomena is a major cause of a continuing, worldwide recession.

TECHNOLOGY PROVIDES A HIERARCHY OF MEMORIES WITH VARYING COST AND ACCESS TIMES

Semiconductor memories are only one part of computer memory systems, which can be thought of as a hierarchy. Without the principle of locality, a hierarchy would be unnecessary. Information pertaining to a present computation is stored in fast registers, including vector registers and temporary vector registers as in Cray's architectures, that are part of the central processing unit, whereas recently referenced information is held in cache memories. Information referenced less often is stored in primary (semiconductor array) memories or in caches within the memory chips. Infrequently referenced information is stored using electromechanical technologies that record information on magnetic disks, magnetic tape, and electro-optical media. Similarly, magnetic devices use a series of cache memories for access time reduction. Although each lower level in this technological hierarchy is characterized by slower access times, the cost per bit stored is correspondingly lower.

Just as increasing transistor density has improved the storage capacity of semiconductor memory chips, increasing areal density[2] has directly affected the total information-storage capacity of disk systems. IBM's 1957 disk file, the 350 RAMAC, recorded about 100 bits along the circumference of each track, and each track was separated by 0.1 inch, giving an areal density of 1,000 bits per square inch. In early 1990, IBM announced that one of its laboratories had stored 1 billion bits in 1 square inch. This technology progression of six orders of magnitude in 33 years amounts to a density increase at a rate of over 50% per year.

Increases in areal density have led to magnetic storage systems that are not only cheaper to purchase but also cheaper to own, primarily because the

[2]The amount of information that can be stored per unit area.

density increases have markedly reduced floor-space requirements (which are a substantial item of expense in many environments). The 5¼- and 3½-inch drives can be mounted within a workstation, and without such high-density disks, the modern workstation environment would be impossible. In 1992, electro-optical disk technologies provide a gigabyte of disk memory at the cost of a compact audio disk, making it economically feasible for PC or workstation users to have roughly 400,000 pages of pure text or 10,000 pages of pure image data instantly available. Similarly, advances in video compression using hundreds of millions of operations per second permit VHS-quality video to be stored on a CD. In short, along with semiconductors, disks have been a necessary enabling technology for every computer class, including mainframes, minicomputers, workstations, PCs, laptops, and pocket computers.

EVOLUTION OF COMPUTERS BASED ON LOCALITY AND THREE DESIGN PRINCIPLES

The remaining ideas come from the principle of locality and three general principles of all design that apply to all systems.

The Principle of Locality

Temporal and spatial locality have been observed for much (most) computation. This means that when a datum is accessed it will be accessed again, and data that is close is likely to be accessed. Fortunately, a hierarchy of memory elements exists (described earlier) that has allowed computer architects to exploit locality through caching data in large blocks throughout the hierarchy for temporal and spatial locality.

In 1992, the key to making large, scalable, parallel computers is a belief in spatial and temporal locality, because very large systems have inherently longer processor to memory latencies than small, central systems. All supercomputers have a hierarchy of registers that are designed to exploit locality. Locality is the phenomenon that allowed the first one-level store computer, Atlas, to work. This led to the understanding of paging, virtual memory, and working sets that are predicated on locality. Caches exploit spatial and temporal locality automatically. Large register arrays, including vector registers, are mechanisms to exploit locality.

ED-?
processor ≠ to ≠ memory latencies

Replication of a Design: Performance, Reliability, and Distribution

Replication is an important design principle that allows large, man-made structures to be built efficiently from common components rather than having ad hoc designs. Replication of components provides three functions:

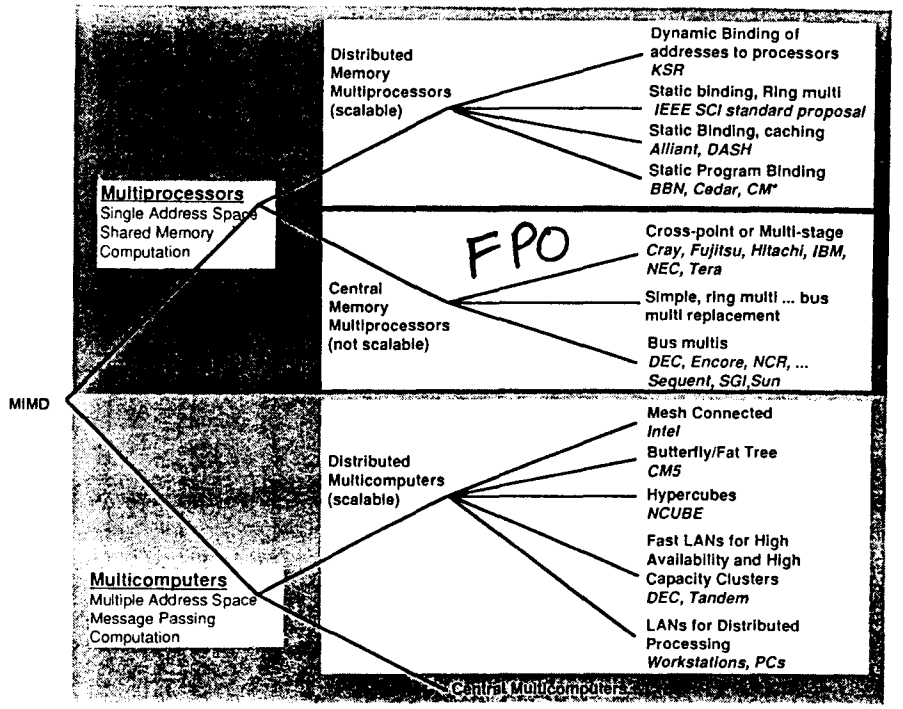

FIG. 6.5. Taxonomy of multiprocessor and multicomputer computer structures.

increased performance through parallelism, reliability through redundancy, and spatial distribution. Figure 6.5 shows a taxonomy of the alternative structures that come through replicating computers, processors, processing elements, and memories to increase parallelism spatially and temporally (pipelining). We can also look at these alternatives as transformations on the von Neumann machine (Fig. 6.6) in the search to increase performance.

Although designers have been building a large range of computers using different models, the intriguing goal has always been to have a single set of components that can be used to implement a wide range. In *Computer Structures* (Bell & Newell, 1971), Allen and I posited an alternative set of machines for the IBM System/360 that was predicated on just a few multiprocessor models. Replicatability is a form of generality, because more general components are required when a single type is needed to carry out a range of functions. Replicatability is a way of extending a design to cover a wide range of utility without specialized designs. The ultimate form of replicatability and generality is the scalable multicomputer for performance that began to emerge in the late 1980s. Scalable multicomputers for reliability and physical distribution existed for some time to a small degree

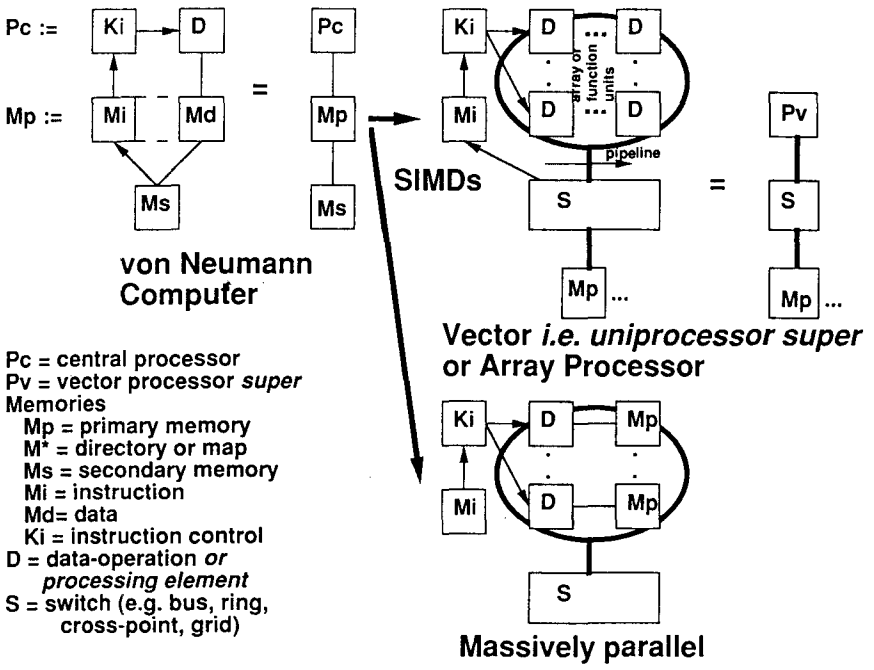

FIG. 6.6. Transformations on the von Neumann architecture to increase performance through parallelism.

at Tandem (for reliability), and as LAN connected workstation for spatial distribution.

Replicatability Within a Computer: Multiprocessors and the Opportunity for Parallelism

Multiprocessors permit a single computer to be extended to provide a completely balanced system of all resources including processing. Beginning in the mid-1960s multiprocessors evolved with several design objectives: Incremental performance increase allows the best system balance, fault tolerance, and increased performance through parallelism (Bell, 1991).

If a completely scalable computer can be built, then an arbitrary or scalable amount of processing can be provided within a given computer. The only way a scalable computer can exist is by placing the memory and processor together in exactly the same way as a collection of computers are interconnected, that is, by not having a single switch that limits performance, between the processor and memory.

The big idea of the 1990s is the scalable multiprocessor (see Fig. 6.7) that allows a single shared memory to have arbitrary processing, memory, and

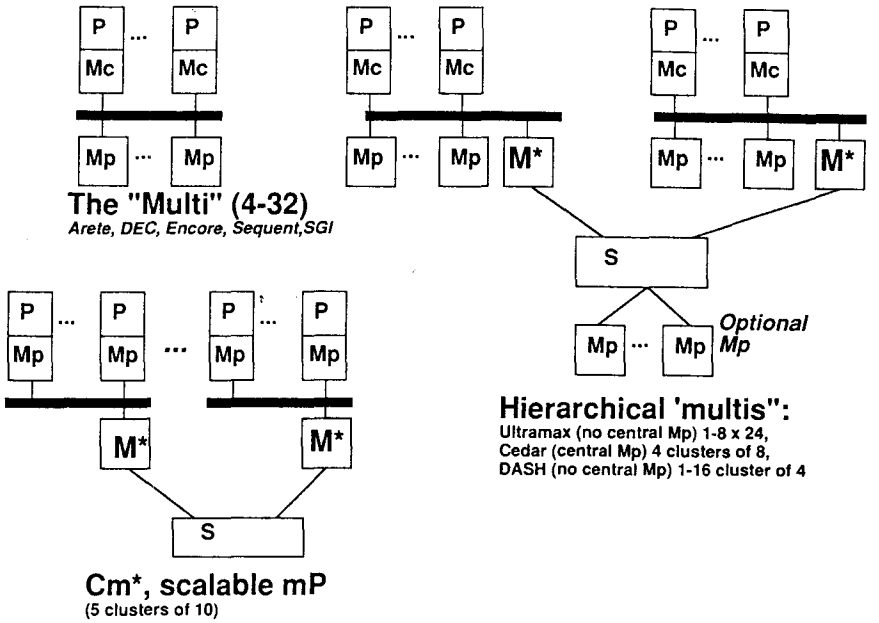

FIG. 6.7. Scalable multiprocessor structures that have limited and nearly unlimited scalability.

I/O. Carnegie Mellon University's Cm* is the first attempt to build a fully scalable multiprocessor (mP), shared-memory computer. In 1992, the first production, scalable mP was introduced by KSR, and is described later.

With scalable multiprocessors and the ability for an arbitrary number of processors to share work held in a common memory, a much larger opportunity is created: exploiting parallelism.

Replicatability of Multiple Computers, and the Problem of Making Them Operate as One

The replication and physical interconnection of multiple computers is the easy part; putting the computers together again so that they are one has always proven difficult. The degree of success depends on the objectives: reliability, physical separation with some small degree of resource sharing, or performance where all computers must behave as one.

Figure 6.8 shows that scalable multiprocessors and multicomputers interconnected by a large switch appear to be identical from a distance. Unfortunately, the difference is the amount of hardware in a multicomputer that permits it to behave as one shared-memory multiprocessor computer. Thus, the question comes back to the big idea of using software to

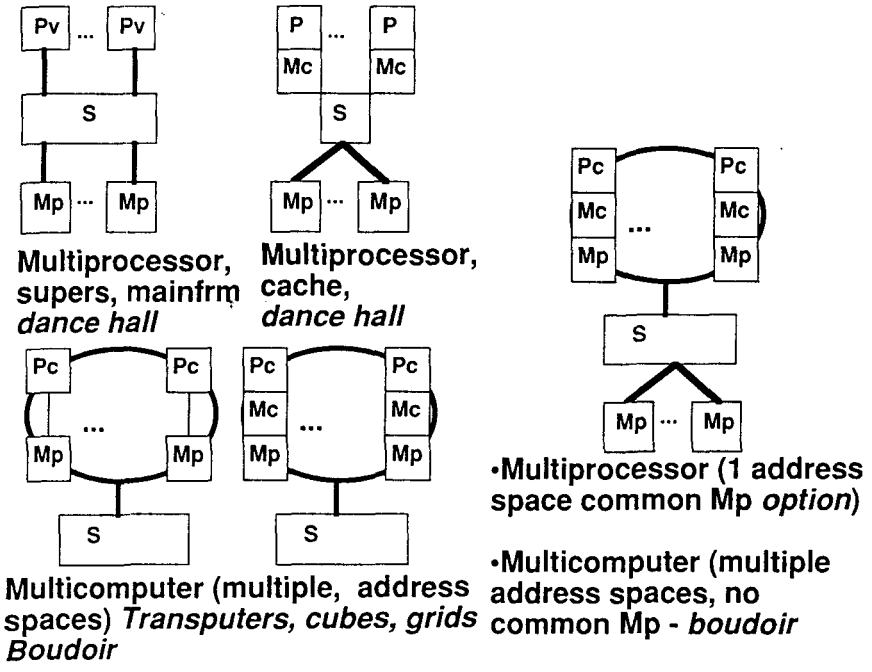

FIG. 6.8. Distributed memory computers that result in either distributed shared memory (DSM) multiprocessors or multicomputer clusters.

carry out functions that hardware normally provides, including address formation and translation, data migration across the computing nodes, and the use of messages to access memory in other nodes instead of directly accessing it by name.

Economics Constrains Design

As in all systems, economics is the key part of the objective function that determine design. The market (economics) determines the form and function of computation, not elegance,[3] religion, architecture fads, and so on. Whether a computer is the "best" by economically objective measures such as performance/price or performance/time is usually irrelevant because of an organization's ability to market a product. For example, Dijkstra had nothing good to say about the System/360 announcement, yet the 360 will most likely survive by evolution longer than any other computer—except Intel's evolution of the X86 architecture from the 8080 for the PC: 8086/8088 > 80286 > 80386 > 80486 > Pentium > P6 > 64 bit architecture.

[3] *Elegance* is defined as having one feature that provides at least two functions.

Software Compatibility and Computer Families

Economics argue for the smallest number of machines (e.g., ISAs, O/Ss, programming languages) at each of level of the hierarchy in order to maximize user learning, build on other work, and have the greatest market. Of course, industrial forces act to increase the number of machines so that each company can differentiate itself and create monopolies. Aside from having a single computer that evolves with the addition of more address bits to access the larger memories that come with time, three innovations came about based on computer families. In 1962, SDS (now defunct) introduced three compatible minicomputers. The IBM System/360, introduced in April 1964, was the first successful family that covered a wide range of sizes (Bell & Newell, 1971). In all cases, the programming style and use evolved from a centralized, batch-processing approach. In 1975 we started the VAX project at Digital to provide a range of computers that covered a wide range of computing styles across a hierarchy of use from personal (the workstation), to distributed departmental computing, to centralized mainframe style computing in clusters that can behave as one computer.

BALANCING RESOURCES

Throughout the development of computers, a key requirement for a successful computer has been the ability to appropriately configure a computer to serve a variety of functions. This means that a given computer must be able to have an almost arbitrary amount of processing power, a memory hierarchy with a variety of sizes at each level, terminal capability including visualization, and networking to other sites. The Amdahl–Case rules of thumb were posited in the 1960s for balancing instruction speed, memory size, and I/O data rate: 1 instruction per second requires 1 byte of primary memory and 1 bit per second of I/O bandwidth. In the 1990s it's unclear whether ratios like these have meaning when a single computer is used both as a personal workstation and as a multiuser server with virtual memory. In the case of scientific computing, balancing floating-point operations per second (flops) with memory can vary from less than 1 flops/byte to 1 flops/8 bytes. Los Alamos National Laboratory estimates that for a given computer, from 5,000 to 1/15 byte of secondary memory is required per byte of primary memory for several "grand challenge" problems.

AMDAHL'S LAW

In 1967, Gene Amdahl pointed out the problem of using a large number of slow processors to do the same amount of work that a faster processor could accomplish. He looked at the problem of computation as being

composed of a slow, sequential part and a part that could be accomplished in virtually zero time. The total time to do work is governed by the slow part. This applies to I/O, sequential processing, operating system overhead time—the slow part versus the fast parallel part in an mP, mC, or a fast vector operation. Today, the parallel processing dilemma is the need for a very large fraction, F, of a given program to be parallel, when using a large number of processors, N, to obtain high efficiency, $E(F,N)$:

$$E(F,N) = 1/[F + N \times (1 - F)]$$

Thus, scaling up slow processors is a losing proposition for a given fraction of parallelism. For 1000 processors, F must be 0.999 parallel for 50% efficiency. The goal in massive parallelism is not to have the greatest number of processors, but rather providing the greatest number of usable operations in the shortest time.

ED - ? For 1,000 processors,

COMPUTER SIZE, COMPUTER GENERATION, AND PROBLEM SCALABILITY

The perception that a computer can grow forever has always been a design goal. For example, the IBM System/360 (circa 1964) provided a 100 : 1 range, and VAX existed at a range of 1000 : 1 over its lifetime. Ideally, one would start with a single computer and buy more components as needed to provide size scalability. Similarly, when new processor technology increased performance, one would add new generation computers in a generations-scalable fashion. Ordinary workstations provide some size and generation scalability, but are LAN-limited. Problem scalability is the ability of a problem, algorithm, or program to exist at a range of sizes so that it can be used efficiently and correctly on a given, range of scalable computers. Problem scalability, discovered in 1987, provides the economic basis for scalable computers.

ED - ? 1,000 : 1 over

GENERALITY IN DESIGN

Good design comes about in a search for generality by finding a few, simple mechanisms to accomplish a task rather than having a number of ad hoc functions that are unlikely to be related to each other. I define these designs as "elegant" because one mechanism provides multiple functions. I have enjoyed the search for generality in working on designs of general register computers, the Unibus, Ethernet, and many multiprocessors. My focus on multiprocessors as opposed to multicomputers is solely

because they are elegant because they are "general" (Bell, 1991). That is, unlike multicomputers, one resource (processor, memory, or I/O channel) can be used by any process.

General registers that came about in the mid 1960s are a good example of providing one set of registers for accumulation, multiplier-quotient, indexing, stack control, and so on, rather than a collection of special registers. For performance reasons, fixed-point and floating-point registers are separated so they can be operated on independently and in parallel. The Unibus, Ethernet, and Token Rings all provide interconnection of components on a peer (general) rather than hierarchical basis. The mapping of I/O registers (state) into the memory address space used in the PDP-11, and now in all computers, is another example of generality. It is possible to go too far with generality if it costs time and space; for example, providing bit addressing and arbitrary length instruction fields has been used in unsuccessful computers.

Multiprocessors illustrate another use of generality for processing I/O including graphics and transformations and for carrying out specialized arithmetic operations. Invariably, providing more resources of a single kind (processing, memory, communication bandwidth), in a single place invariably provides a more flexible and cost-effective structure. As a taxonomist and architect, I'm interested in computers that evolve all of the design principles by extending every capability that a machine has. The following example, the last computer noted in Table 6.2, is as near the ideal of all computers that I've sought out during my three-plus decades of building computers.

THE KSR-1 SCALABLE MULTIPROCESSOR[4]

The Kendall Square Research KSR-1 is significant because it provides size (including memory and I/O) and generation-scalable smP where every node is identical; an efficient environment for both arbitrary workloads (from transaction processing to timesharing and batch) and sequential to parallel processing through a large, hardware supported address space with an unlimited number of processors; a strictly sequential consistent memory and programming model; and dynamic management of memory through hardware migration and replication of data throughout the distributed, processor-memory nodes using its Allcache mechanism.

The KSR-1 is a size- and generation-scalable, shared-memory multiprocessor computer. It is formed as a hierarchy of interconnected "ring multis."

[4]The company ceased operation in 1995 through no fault of its architecture. Distributed Shared Memory computers have been introduced by Convex and Cray Research in 1994. Several companies and various university research is aimed at building similar computers.

ED-?
Several companies
sentence ok?

Scalability is achieved by connecting 32 processors to form a ring multi operating at 1 Gbyte/sec (128 million accesses per second). Interconnection bandwidth within a ring scales linearly, because every ring slot may contain a transaction. Thus, a ring has roughly the capacity of a typical cross-point switch found in a supercomputer room that interconnects eight to sixteen 100-Mbyte/sec HIPPI channels. The KSR-1 uses a two-level hierarchy to interconnect 34 rings (1,088 processors), and is therefore massive. The ring design supports an arbitrary number of levels, permitting ultracomputers costing $50–250 million to be built.

Each node is comprised of a primary cache, acting as 32 Mbytes primary memory, and a 64-bit superscalar processor with roughly the same performance as an IBM RS6000 operating at the same clock rate. The superscalar processors, containing 64 floating-point and 32 fixed-point registers of 64 bits, is designed for both scalar and vector operations. For example, 16 elements can be prefetched at one time. A processor also has a 0.5-Mbyte subcache supplying 20 million accesses per second to the processor (computational efficiency of 0.5). A processor operates at 20 MHz and is fabricated in 1.2-μm CMOS. The processor, *sans* caches, contains 3.9 million transistors in 6 types of 12 custom chips. Three-quarters of each processor consists of the search engine, responsible for migrating data to and from other nodes, for maintaining memory coherence throughout the system using distributed directories, and for ring control.

With sequential consistency, every processor returns the latest value of a written value, and results of an execution on multiple processors appear as some interleaving of operations of individual nodes when executed on a multithreaded machine. With Allcache, an address becomes a name, and this name automatically migrates throughout the system and is associated with a processor in a cache-like fashion as needed. Copies of a given cell are made by the hardware and are sent to other nodes to reduce access time. A processor can prefetch data into a local cache and poststore data for other cells. The hardware is designed to exploit the principle of locality. For example, when executing a single program on parallel data, copies of the program move dynamically and are cached in each of the operating node's primary and processor caches. Data such as elements of a matrix move to the nodes as required simply by accessing the data, and the processor has instructions that prefetch data to the processor's registers. When a processor writes to an address, all cells are updated and memory coherence is maintained. Data movement occurs in subpages of 128 bytes (16 words) of its 16-Kbyte pages.

Every known form of parallelism is supported via KSR's Mach-based operating system. Multiple users may run multiple sessions, comprising multiple applications, comprising multiple processes (each with independent address spaces), each of which may comprise multiple threads of control running simultaneously sharing a common address space. Messages

passing is supported by pointer passing in the shared memory to avoid data copying and to enhance performance.

KSR also provides a commercial programming environment for transaction processing that accesses relational databases in parallel with unlimited scalability, as an alternative to multicomputers formed from multiprocessor mainframes. A 1-Knode system provides almost two orders of magnitude more processing power, primary memory, I/O bandwidth, and mass storage capacity than a multiprocessor mainframe and is completely scalable. A 1,088-node system can be configured with 15.3 terabytes of disk memory, providing 500 times the capacity of its main memory. The 32- and 320-node systems are projected to deliver over 1,000 and 10,000 transactions per second, respectively, giving it over a hundred times the throughput of a multiprocessor mainframe.

SUMMARY

Although every new computer that introduces a new idea or innovation is always described as revolutionary, such a label is almost surely marketing hype. The only two great revolutionary ideas are the computer itself, and the integrated circuit that evolves very rapidly and allows computers of all types, description, and functionality to be developed.

Three general engineering design principles have driven computer structures: general economics-based design, replication, and the search for generality.

The principle of locality is the important observation about the behavior of programs that has allowed computer inventions or big ideas to be discovered, including the memory hierarchy, the one-level store that led to the idea of a virtual memory, the cache memory, and finally scalable computers in the 1990s.

Once the computer was discovered, the big ideas that were variants on the theme of the computer included using computers for a wide range of applications, building a hierarchy of machines, and building a computer within a computer that could be shared by many programs, that is, multiprogramming and timesharing.

The building of physically distributed multicomputers created the desire to have such a collection behave as a single computer and, in 1992, to operate in parallel. In a similar fashion, building a multiprocessor computer from a common set of components created the desire to have such a collection of processors operate together on a single problem.

Thus, evolving computer structures to a collection of networked computers, each of which can have an arbitrary number of processing elements, has created two incredibly difficult problems to be solved in the 1990s:

getting these computer networks back up to the capability that existed when only a single processor controlled all the resources and behaved as an integrated system, and the opportunity for parallelism within a single virtual memory whereby all the resources can operate on a single program.

REFERENCES

Bell, C. G. (1985). Multis: A new class of multiprocessor computers. *Science, 228,* 462–467.

Bell, G. (1989). The future of high performance computers in science and engineering. *Communications of the ACM, 32*(9), 1091–1101.

Bell, G. (1991). Three decades of multiprocessors. In R. Rashid (Ed.), *CMU computer science: A 25th anniversary commemorative.* Reading, MA: ACM Press, Addison-Wesley.

Bell, G. (1992). Ultracomputers: A teraflop before its time. *Communications of the ACM, 35*(8), 27–47.

Bell, C. G., Grason, J., & Newell, A. (1972). *Designing computers and digital systems using PDP16 register transfer modules.* Maynard, MA: Digital Press.

Bell, G., & McNamara, J. (1991). *High tech ventures: The guide to entrepreneurial success.* Reading, MA: Addison-Wesley.

Bell, C. G., & Newell, A. (1971). *Computer structures: Readings and examples.* New York: McGraw-Hill.

Denning, P. J. (1980). Working sets past and present. *IEEE Transactions on Software Engineering, SE-6*(1), 64–84.

Gomory, R. E., & Schmitt, R. W. (1988). Science products. *Science, 240,* 1131–1132, 1203–1204.

Hennessy, J. L., & Patterson, D. A. (1990). *Computer architecture: A quantitative approach.* Palo Alto, CA: Morgan Kaufman.

Meindl, J. D. (1987). Chips for advanced computing. *Scientific American, 225*(10), 78–88.

Siewiorek, D., Bell, C. G., & Newell, A. (1982). *Computer structures: Principles and examples.* New York: McGraw-Hill.

ED-?
'Bell, G.' ref's should follow all 'Bell, C. G.' ref's