International Journal of Parallel Programming, Vol. 22, No. 1, 1994

Scalable, Parallel Computers: Alternatives, Issues, and Challenges

Gordon Bell

Received February 1993; revised December 1993

The 1990s will be the era of scalable computers. By giving up uniform memory access, computers can be built that scale over a range of several thousand. These provide high *peak announced performance* (PAP), by using powerful, distributed CMOS microprocessor-primary memory pairs interconnected by a high performance switch (network). The parameters that determine these structures and their utility include: whether hardware (a multiprocessor) or software (a multicomputer) is used to maintain a distributed, or shared virtual memory (DSM) environment; the power of computing nodes (these improve at 60% per year); the size and scalability of the switch; distributability (the ability to connect to geographically dispersed computers including workstations); and all forms of software to exploit their inherent parallelism. To a great extent, viability is determined by a computer's generality—the ability to efficiently handle a range of work that requires varying processing (from serial to fully parallel), memory, and I/O resources. A taxonomy and evolutionary time line outlines the next decade of computer evolution, included distributed workstations, based on scalability and parallelism. Workstations can be the best scalables.

KEY WORDS: Scalable multiprocessors and multicomputers; massive parallelism; distributed or shared virtual memory; high performance computers; computer architecture.

1. INTRODUCTION

In this decade, computer engineers, computer and computational scientists, and users will focus on understanding and exploiting the parallelism inherent in computers formed by interconnecting many low-priced, extremely fast, "killer" CMOS microprocessors. A computer using at least 1000 processing elements, processors, or computers in parallel is "massively

0885-7458/94/0200-0003$07.00/0

parallel." In principle, *ultracomputers* (Bell[1]) with 1000s of processors and costing $30–$250 million could be built. However, based on results, an aggressive target for 1995 or 1998 (i.e., one or two generations) is *applications* (apps) that routinely achieve 10 to 100-fold parallelism.

Two, massively parallel computer structures have been introduced in a race to provide a "peak" teraflop of computing power (Bell[1]) by 1995: the *scalable, shared memory multiprocessor* (smP) and the *scalable multicomputer* (smC). In order to make a large, scalable, general purpose computer, computer modules, i.e., processor-memory pairs, are interconnected by a high performance, low latency switch, i.e., network; and performance, using an appropriate measure, grows in proportion to the amount of resources.

Multiprocessors (Fig. 1a) communicate by accessing a single, shared common memory. Multicomputers (Fig. 1b) are independent computers that communicate by passing messages to one another through the switch. A software library layer for an smC creates a single address space creating a *distributed shared virtual memory* (DSM), hence mPs and mCs converge. An smP has one address space and message passing is facilitated by simply passing pointers. Nitzberg and Lo[2] provide a survey of mP and mC DSMs', including the issue of maintaining a single, coherent memory.

Two, basic programming paradigms are used: *data parallel* using a dialect of FORTRAN, such as FORTRAN 90, High Performance FORTRAN (HPF), or just FORTRAN 77—multiple copies of a Single Program that operate on Multiple Data items in parallel (SPMD); *multi-*

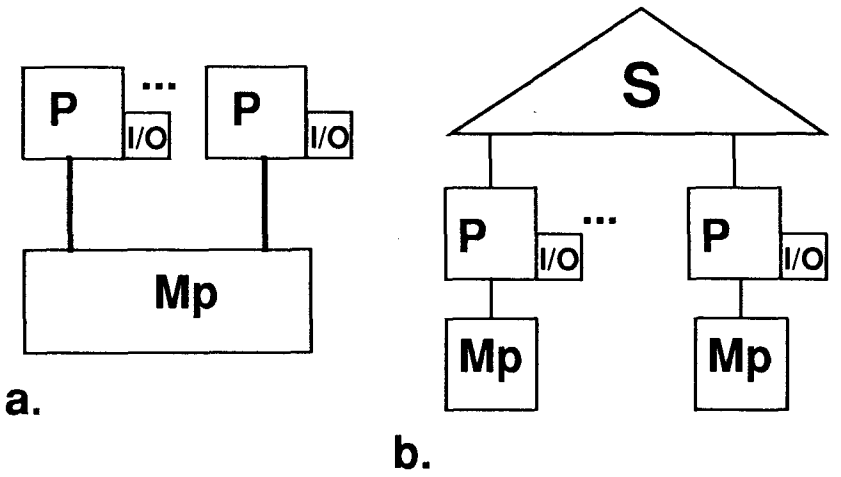

Fig. 1. Programming views of a shared memory multiprocessor (a) and a multicomputer (b) a distributed multicomputer.

process using a program that is divided into sub-problems and distributed among the nodes that communicate by explicit message passing. Multi-process apps can be divided by function (i.e. different processes handle different types of tasks) or by data (i.e., different processes handle different data). Ordinary operating system mechanisms such as pipes, sockets, and threads facilitate parallelism by providing communication among and within processes. Programming environments that operate on all computer structures, including networks, have been developed for multi-processing, such as the Parallel Virtual Machine (PVM), Linda, and Parasoft.

Computer size scalability is defined pragmatically, as a computer designed from a small number of basic components, with no single bottleneck component, such that the computer can be incrementaly expanded over its designed scaling range, delivering linear incremental performance for a well-defined set of *scalable* apps. The components include: computers (i.e., a processor-memory pair), secondary memory, communication links and terminals, switches, cabinets, and especially the computer's programming environment (operating system, compilers, performance monitoring, etc.). Researchers have posited several definitions of scalable computers (Hill[3]; Nussbaum and Agarwal[4]; and Scott[5]).

Evolvability, i.e., generation or technology scalability is the ability to implement a subsequent computer of the same family using faster components. Evolvability is an essential property of a scalable computer because of the long time and large investment required to develop parallel programs. Evolvability requires that all rate and size metrics such as processing, memory and I/O bandwidth, memory size, and especially interconnection bandwidth must increase proportionally from generation to generation.

Program or problem scalability, first observed by Gustafson *et al.*,[6] is a property of a program/machine combination that determines the ability of a problem to operate at various scales (sizes) on a given scale computer using goodness measures of constant efficiency (Kumar and Gupta[7]), or constant speed (Sun and Rover[8]) or simply increased speedup (Karp[9]). By scaling a problem to a sufficiently large size to reduce the computation to communication ratio, overhead can be reduced to increase processing rates.

The IBM 360 (c1964) and VAX (c1978) series of compatible general purpose computers were successful during eras where evolving and varied technologies could be used to provide some scalability with a range of models at a given time and over time. VAX evolved to a factor of 100 performance range in 10 years, not including LAN workstation multi-computers. Workstations provide size scalability (large installations have 10,000 workstations) and evolvability to some degree, although LAN

communication rates have remained constant at 10 Mbits/sec, while processor power increased a factor of 100 between 1982 and 1992.

In 1993, Cray Research offers a range of products from $300,000 to over $30 million spanning a performance range of 100, using both CMOS and ECL implementations of the Cray supercomputer architecture. Cray supers evolved from a 120 Mflops processor and 1 Megaword memory (1976) to sixteen 1000 Mflops processors and 4 Gigaword memory (1992). A cluster of 4-C90s increases the range to 400. This amounts to a factor of 17% and 36% per year performance increase for processor and system (including multiprocessor), respectively. In contrast, 1994 scalable computers offer a factor of 1000 practical range of performance using just one component type. However, only limited scalable products (8–1024) are offered.

In 1994 *an ideal, scalable computer* should be useful as a single processor, and extend to 1000s of processors, with correspondingly scalable I/O. It should be able to handle a wide range of scalable parallel apps, including a general workload. The interconnect network must be generation scalable over at least a decade to support binary compatibility of apps among generations! Furthermore, since the processor-memory pairs are independent, the ideal scalable computer should be distributable beyond a single room to include a campus. By solving many problems in security and fault-tolerance, a distributed computer that would occupy a building or even a large campus can be designed.

While scalable computers provide a factor of 5–8 more peak *announced* performance, or PAP (Worlton[10]) and performance/price as compared with traditional supercomputers, their position as a "main line" computer structure is by no means assured. For example, while parallel programs with little coupling among the computational threads approach PAP for all computers, the Cray C90 supercomputer provided both the greatest performance and best performance/price for a mix of computational fluid dynamics apps characterized by the NAS benchmarks, causing the CFD computational scientists to warn (Bailey *et al.*[11]):

> "Some scientists have suggested that the answer to obtaining high performance rates on highly parallel computers is to substitute alternative algorithms that have lower inter processor communication requirements. However, it has been the experience of the scientists in our research group that a certain amount of long-distance communication is unavoidable for these types of applications. Alternative algorithms that have higher computation rates usually require more iterations to converge to a solution and thus require more overall run time. Clearly it is pointless to employ numerically inefficient algorithms merely to exhibit artificially high performance rates on a particular parallel architecture (Bailey[12])."

Whether ECL supercomputers should "cost" so much more than CMOS microprocessors is unclear, but based on 20% manufacturing learning curves,[2] a product with 512 times the unit volume (say 50 K versus 100 units/year) costs one-eighth as much. Furthermore, the design cost for a vector processor is high for a very low volume computer. This helps account for the difference in price per PAP of the two computers. On the other hand, ARPA, as part of the *High Performance Computer and Communications* (HPCC) program, has provided "massive funding" equal to the market's annual revenues over the last decade to "State Computer Companies[3]" for development and mandated purchases without benchmarking or acceptance testing. Funding this small, overcrowded computer market has distorted cost structures, creating both a weakened supercomputer industry and poor, unprofitable "State" ventures. Still, hardware design is small in comparison to system software costs. Custom and specific market apps are the true "Achilles Heel" of massive parallelism. Apps costs dwarf design, purchase, and operational costs.

Viability of computers, i.e., commercial success, has historically favored *generality*, including software compatibility among a variety of different sized hardware platforms over a long time period, ability to handle a variety of job sizes, application types, degree of parallelism, and mix of computational resources (processing, primary and secondary memory, network and human interface communications, etc.) Not every scalable computer formed by interconnecting processor-memory pairs is equally general, or able to work on a variety of application types: commercial, doing batch processing, and database for decision support and transaction processing; real-time, such as communications; and technical, operating on floating-point data that usually require large files, interactivity, and visualization. The evolution to scalable computers will be defined and limited by one factor, ease of programming; this in turn is influenced by the degree of granularity each structure can achieve for parallel apps. As a minimum condition for viability, distributed computers must run *existing* supercomputer apps competitively. The lesson of poor scalar capability on the CDC Star, that begot the CDC 205, that begot the ETA 11 is an important one for scalable computers: in order to be a viable challenger, the challenger must completely cover the incumbent.

Worlton[13] accurately describes massive parallelism as a clear example of the "bandwagon effect" where we make the biggest mistakes in managing technology. A bandwagon is "a propaganda device by which the

[2] For each doubling of the number of units produced, the cost of the units is reduced by 20%.
[3] Cray Research T3D, Intel iPSC2, and Paragon. Teracomputer. Thinking Machines CM1, 2, 200, and CM5.

purported acceptance of an idea, product or the like by a large number of people is claimed in order to win further public acceptance." The massively parallel bandwagon is drawn by: vendors, computer science researchers, and bureaucrats who gain power by increased budgets to dole out. Innovators and early adopters are the riders. The bandwagon's four flat tires are caused by the lack of: systems software, skilled programmers, guideposts (heuristics about design and use), and parallelizable applications.

Independent of supercomputers and massive parallelism, most technical computing is carried out on high volume PCs and workstations with 20 and 0.6 million units delivered in 1992. By 1995, if each workstation delivers 100 Mflops,[4] then 1 million workstations will provide 100 teraflops of power, and a large installation of 10,000 workstations would provide a PAP of one teraflop. For example, Nakanishi, Rego, and Vaidy-Sunderam were the cost-effective winner of the 1992 Gordon Bell Prize (Karp *et al.*[14]) using 192 workstations to solve to solve a single problem in parallel. This *potential* power, at no extra cost, argues that R&Ds' greatest payoff is using workstations in parallel, enabled by high bandwidth, low latency/low overhead switches. The HPCC program should focus on this goal, based on standards, to make the greatest impact.

The focus of scalable computers (measured by HPCC funding) has been on parallelism in order to deliver the greatest number of floating-point operations. HPCCs' goal is to stimulate the design of a large computer that can provide a peak announced performance (PAP) of one teraflop (10^{12} floating point operations per second), with the eventual goal of applying these computers to several targeted "Grand Challenge" apps. True supercomputers are not in the teraflops race because they are less likely to be able to provide a PAP terflop soon enough for HPCC funders, and have been disqualified from the race. In 1993, traditional supercomputers are likely to provide most of the supercomputing capacity until 1995 and probably one generation beyond (c1998) since the Real Application Performance (RAP) for equally priced supers and scalables is equal, since the PAP-to-RAP ratio has been a factor of 5–8 worse for scalables.

The paper will first give a taxonomy of scalable computers and a comparison of their strength and weakness. The next section presents a description of various functions that future scalables must handle together with how the mC will converge to include the mP. Key benchmarks and machine parameters are provided in order to order to evaluate the alternatives. The final section examines the design issues in future scalables.

[4] Leading edge workstations will provide 400–800 Mflops.

2. COMPUTER SPACE TAXONOMY AND SCALABLE COMPUTERS

A computer space taxonomy given in Fig. 2 will be used to provide a perspective on the evolution and challenges of building parallel computers. Uniprocessors will be described as components for scalable computers. The MasPar SIMD is given, and while it does not provide the peak power or a great scaling range of a supercomputer, it is important as measured by performance, performance/cost, mean time before answers (mtba) and is an alternative "server" for massively parallel processing.

MIMD computers are the focus of the paper because they are scalable and offer the greatest opportunity for exploiting parallelism of all kinds from multiple jobs to a single job. Four dimensions have been used to structure the MIMD taxonomy:

1. multiprocessor (mP) versus multicomputer (mC) forms the first branch;

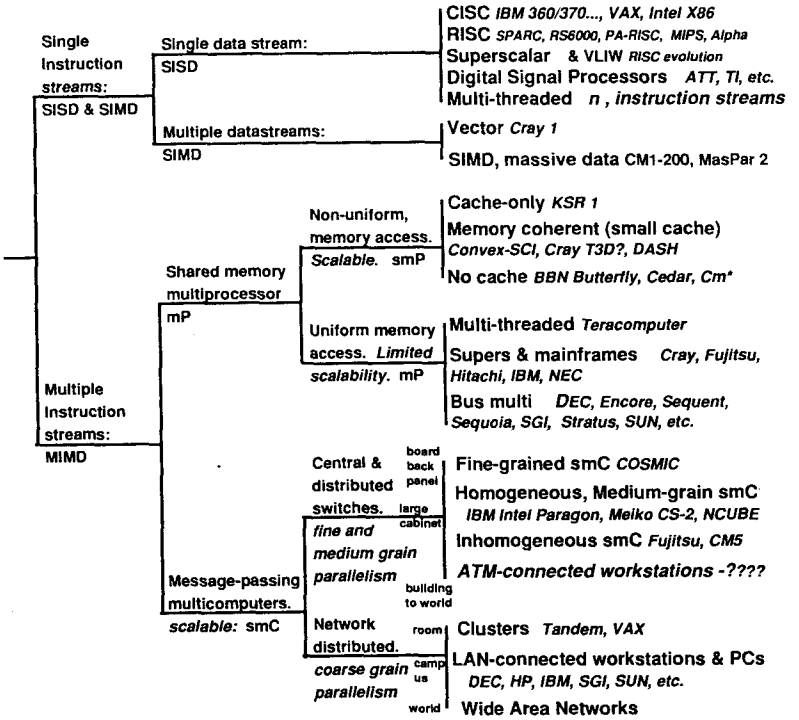

Fig. 2. Computer taxonomy showing parallel and scalable structures.

2. scalability determines the second branch, note that every mC should be scalable, subject to switch and distributability limitations; Uniformity of memory access could be an alternative characterization of the dimension.
3. distributability inter-node latency, covering the range: backpanel, room (1 μsec), building, campus (10–100 μsec), and wide area (milliseconds); This dimension affects apps granularity.
4. homogeneity (symmetry) of nodes and the need of, or coupling to, host processors.

A final attribute, ease or likelihood of *evolvability* is critical, albeit difficult to assess. Every computer has some "bottleneck" such as I/O, communications or its interconnection network that is difficult to increase from generation to generation.

Distributability determines latency and whether a fully computer can be applied to a single workload or problem. While LAN-based workstations within a campus can potentially provide enormous power, it is unlikely that a collection of computers acting on a single workload will be distributed outside of a building because of long latencies (milliseconds), low bandwidth (1 Mbyte/sec), and high message overhead (1 millisecond). The human organizational aspects usually confine or associate a cluster of computers with a group. Thus, the latency of a single system is required to be only a few microseconds in order to provide medium grain parallelism (<1000 instructions per computational thread). For efficient operation using 100 Mflops/sec microprocessors and task-to-task communication overhead of 10 μsec., relatively large grain problems (i.e., program threads that carry out at least 1000 operations) are required in order to obtain half-peak performance of a single node. Table I shows the distribution and

Table I. 1995 Distributed Scalable Computers Distance and Latency Characteristics

	Board	Cabinet	Room	Building	Campus	Continental
distance (m)	0.5	3	30	300	3 Km	10,000 Km
delay (μs)	0.01	0.1	<1	6–10	60–100	200 ms
LAN bandwidths	·····	·····	·····	·····	·10–100 Mbit	
link bandwidths	····	100 Mbytes-1 GByte		·····	····· Gbit	1.5–622 Mbit
ATM bandwidth				·····	·155–622 Mbit	
mP & smP	····	multis Cray, KSP		?	??	???
smC	fine grain	···· Meiko, Paragon, TMC			··· workstation-based	

latency characteristics that next generation scalable computers could achieve. Note that high bandwidth network switches can provide the switching characteristics of 1992 generation multicomputers, while 1995 generation microprocessors will improve by a factor of 2–4.

2.1. Single Instruction Streams

The SISD alternatives are shown at the top of the taxonomy, and include the range of CISC to very long instruction word processors. While conventional processors are important, the average amount of parallelism is one instruction per clock tick in the early 1990s, but could approach four by 2000 if longer instruction word architectures and parallel interpretation of a single instruction stream become more prevalent, and higher bandwidth memories are available. The extra-long instruction word processors can be viewed as a RISC or microprogrammed vector processor; various portions of the instruction word control what is fundamentally the address and data registers of a vector processor. Clearly, research challenges exist in architectures and compilers to increase parallelism for a single processor. As architectures become more complex, processor implementation times increase, making coarser grain scalable computers based on simple microprocessors a more attractive alternative.

The multi-threaded processor provides the best solution to hide longer latency inherent in the memory hierarchy and for scalable distributed memory computers. The multi-threaded processor appears as a uniprocessor, since one processor interprets several, independent instruction/data streams as a multiprocessor. This structure also appears as a multiprocessor. Tera, is building a large mP consisting of many, multi-threaded mPs that interconnect to many memories through a large, central crosspoint switch. In this fashion, the apparent access time for a processor can be reduced to a single clock. Caltech's Mosiac-C and other research projects are based on multi-threaded processors.

2.1.1. "Killer" CMOS Microprocessors are the Basis of Scalable Computers

Progress toward the affordable teraflop using "killer" CMOS micros is determined by advances in microprocessor speed. The projection (Bell[15]) that microprocessors would improve at a 60% per year rate following Moore's Law, providing a quadrupling of performance each three years, still appears to be possible (Bell[1]). Since clock speed improves at 25% per year (a doubling in 3 years), the additional speed comes from architecture (e.g., superscalar, wider words, cache memories, and vector processing).

The leading edge microprocessors described at the 1992 International Solid State Circuits Conference included a microprocessor based on Digital's Alpha architecture with a 150 or 200 Mhz clock rate and the Fujitsu 108 Mflop vector processor chip used in Meiko's CS 2. The most important improvement to enhance parallelism is the 64-bit address in order that a computer can have a global address space. By 1995, using 64-bit addresses, the ease of addressing and lack of global address limitations of multicomputers can be overcome.

In 1995, $20,000 microprocessors with peak speeds of 400–800 Mflops would provide 20,000–40,000 flops/$ or 4–8 times greater peak performance/$ than in 1992. Table II shows the evolution of various performance metrics. At 60% yearly increases, a teraflop will be reached in 1998.

One of the important arguments for a MIMD computer is that it is based on a standard, high volume CMOS microprocessor. Only Convex, to use HPs PA RISC operating system, and Meiko using SPARC, Fujitsu Vector processor, and Solaris, use standard microprocessors and software such that apps run without recompilation. The ability to create the leading edge microprocessor is not held with a single manufacturer. MIPS introduced the first and fastest commercial RISC micro in 1985. Over time the speed lead shifted to IBM, HP, and DEC.

2.1.2. SIMD: Vector and Data Parallel

The SIMD vector processor is the core of the supercomputer and used for floating-point intense apps including graphics transformations. Vector processors have been largely ignored for study or use by the computer science community. A vector processor derives its PAP from several sources:

Table II. "Killer" CMOS Microprocessor and Supercomputer Clock and Performance Evolution[a]

Micro	Year	Clock (Mhz)	SPEC (SPECmarks)	Linpack (Mflops)	Lapeak (Mflops)	LFK(.hm) (Mflops)	Pk
VAX780	1978.2	5*	1	0.15		0.16	1
Fujistu-VP	1992.4	50		>50	95	12.5	108
HP PA	1992.2	100	138	56	67	—	200
1995 est.	1995	200–400	300				400–800
2000 est.	2000	300–500					
2000 est.	2000	1000 (supercomputer clock, using custon bipolar or GaAs chips)					

[a] CISC architecture. A comparable RISC architecture would operate at approx. 2 Mhz. 1995 designers at DEC, HP, SGI, and SUN; 2000 consensus at IEEE Computer Elements Workshop, Phoenix, January 1993.

1. single instructions specify operations on 16 vector registers each with 32–128 elements; In addition to load-store, and operations on vector registers; a vector architecture operates on data that is gathered from memory via pointers held in a vector register, allowing it to operate on sparse data and sets of data in parallel.
2. the registers are fundamentally a program controlled cache of typically 16, 64 elements (8 Kbytes) that permits a program to exploit locality;
3. instructions are both overlapped and chained together for parallel and pipelined execution;
5. vector processors have high bandwidths to load/store registers from/to memory.

Supercomputers will be described next as limited scalability MIMDs.

The "array," or massively data parallel SIMD provides outstanding performance for highly data parallel apps including SPMD by using 1 K to 16 K processing elements. Thinking Machines abandoned the SIMD architecture because of its inability to handle sequential tasks, and limited multiprogramming capability. A powerful SIMD is simply a server for data parallel apps. Few apps are only data parallel.

2.1.2.1.MasPar' MP-2. MP-2 is a cost-effective computer with a price range of a $100,000 to $1.5 million using the massive SIMD formula and 1 K to 16 K processing elements. In order to get parallelism, processing elements controlled by a single instruction are placed with distributed memory. Data is moved among the processing elements through a high speed switching network. MP-2 has a high bandwidth memory that can access 2.5 words of memory for each floating-point operation. The MP-2 SIMD has several advantages over its MIMD counterparts: since only one instruction is executed at a time, it is inherently fine grain and synchronized, permitting SPMD programming; the fast, low-latency network interconnecting the processing nodes means that inter-node communication delays are small, such that memory can almost be treated as centralized; it has a fast I/O system for disks and real time data, such as video or radar data.

2.2. Multiple Instruction Streams: Multicomputers

2.2.1. Workstations and other Distributed, Scalable Multicomputers

Since 1975 Tandem has sold clusters of a few (16) computers for redundancy and increased capacity; DEC introduced VAX clusters in 1982

with the additional function of providing access to shared disk servers and allowing each of the computers to operate in a diskless fast LAN-connected environment. LAN connected workstations are the predominant form of scalable computers since they scale over a range of a few dozen for a single work group to several thousand for a building and campus. Workstations are the purest and simplest computer structure able to exploit microprocessors since they contain little more than a processor, memory, frame buffer for the CRT, network connection, and disk. Furthermore, their CRTs solve a significant part of the I/O problem. A given workstation or server node (usually just a workstation without a CRT, but with large memory and a large collection of disks) can also become the basic component of the IBM multicomputers, for example.

Lawrence Livermore National Laboratory (LLNL) made the observation that it spends about three times as much on workstations that are only 15% utilized, as it does on supercomputers. By 1995, microprocessor based workstations with a PAP of 500 Mflops, (25,000 flops per dollar) are 10 times the projected cost-effectiveness of a super. This would mean that inherent in its spending, a lab would have 25 times more unused peak power in its workstations than it has in its central supercomputer or specialized massively parallel computer. Note, the workstations deliver five times the power of the super during the 15% of the time they are being utilized.

In 1993, the inhibitor to using workstations as a scalable multicomputer (smC) is the low bandwidth communication links that limit them to very coarse grain apps. A process-to-process message overhead and latency of 250–500 μsec for a 100 Mflops workstation would imply grains of computation of at least 25 K operations in order to sustain half peak app performance. While being geographically distributed, workstations could in principle, be brought together in a single computer room in order to reduce switch latency to a μsec just like specialized multicomputers. In a central facility, encoded video could be sent to terminals at each user's desk. Alternatively, distributed workstations using ATM switches operating at 155–622 Mbits/sec (20–80 MB/sec) with well designed communications interfaces would have latency, bandwidth, and communication overhead comparable to multicomputers, e.g., CM5. Centralized workstations would be advantageous in terms of power, cost-effectiveness, and administration compared with LAN connected workstations and coarse grain multicomputers. However, unlike traditional timeshared facilities, processors must be dedicated to individuals to provide guaranteed service. With the arrival of HDTV, low cost video can be distributed directly to the desktop, and as a by-product, users would have video conferencing.

In 1993 and 1994, the Cornell Theory Center will use IBM's scalable

parallel systems that will provide 8 and 125 Gigaflops using 64, 125 Mflops and 512, 250 Mflops RS 6000 workstations. Up to 1 K workstations can be interconnected using IBM's high speed, low-latency switch to provide a scalable, multicomputer.

2.2.2. Medium Grain, Scalable Multicomputers

Two types of smCs will be described:

1. homogeneous or symmetric smCs where every node is under control of the same operating system and can be used for any function, e.g. IBM switch-connected workstations, Intel Paragon, Meiko CS-2, and NCUBE; and
2. host-based inhomogeneous smCs where an array of computers operate under control of a host i.e. Thinking Machines' CM5, and Fujitsu's VPP500.

The Intel Paragon and Thinking Machines CM5 are described briefly. The Meiko CS-2 is given in more detail.

2.2.2.1. Intel Paragon. Intel Paragon is a homogeneous multicomputer with up to 1 K nodes interconnected by a 2D mesh. A given node currently consists of i860 microprocessors[5]: one carries out computation operating at a 75 Mflops peak rate and a second is to handle communication. Memory is only 16 Mbytes. The message passing processor and the 2D mesh topology provides communication at 200 Mbytes/sec. The mesh provides primitives for synchronization and broadcasting.

Paragon is formed as a collection of nodes that are controlled by the OSF1 (Mach) operating system with micro kernels that support message passing among the nodes. Each node can, *in principle*, be dynamically configured to be a service processor for general purpose timesharing, or part of a parallel processing partition, or an I/O computer because it attaches to a particular device. Software could, *in principle*, allow a single node to run a single, large program that would "borrow" physical memory from other nodes using memory to memory paging, however, this capability is not supported and large programs page to disk. These characteristics of a homogeneous multicomputer illustrate its limitation, versus a multiprocessor, for flexible resource utilization.

[5] The Intel i860 was introduced as a desktop supercomputer, and initially sold to do graphics processing and highly tuned applications that could carried out with small cache and tolerate long context switching times. It has been withdrawn as a mainline computer by Intel. Hence, evolvability is questionable.

Paragons support one basic forms of parallelism using a shared virtual memory software layer and MIMD. Memory consistency is maintained on a page basis. With MIMD, a program is optimized to provide the highest performance within a node using i860 vector-like processing. Messages are explicitly passed among the nodes. Each node can also operate independently within its own virtual memory address space. Although introduced in 1991, little benchmark, apps, or message passing rates, latency, and overhead data are available since the computer only began to operate effectively in late 1993.

2.2.2.2. Thinking Machines CM5. CM5 is an asymmetrical or inhomogeneous multicomputer consisting of: 1–32 Sun server control computers that "host" user programs and control an array of 32 to 1024 computational computers, each of which have four 32 Mflop floating point arithmetic units and 32 or 128 Mbytes of memory; Sparc-based I/O server nodes; and a tree-structured switch to interconnect nodes. The system is divided into a statically assigned, independent partitions with at least 32 computational nodes managed by a control computer. Each computation node runs a small kernel. The Sun server host and I/O computers run variants of Sun O/S and handle networking, file system, and graphical user interfaces and act as the "Amdahl bottleneck" for workloads with scalar, and nonparallel tasks. Conceptually, the CM5 is an evolution of a SIMD (CM1, CM2, CM200) with a single SPMD program residing in each computation node and a main control program in the control computer. By being inhomogeneous, independent jobs can not run in the computational computers; thus, a CM5 perpetuates the fundamental flaw of SIMD by being unable to process scalar, moderately parallel, and workloads effectively. The CM5 is for large, data parallel batch apps.

The CM5 switch has three parts: diagnosis and re-configuration; data message passing; and control. Control network messages include: broadcasting (e.g., sending a scalar or vector) to all selected nodes, results recombining (carrying out arithmetic and logical operations on data from each node), and global signaling, and synchronization for controlling parallel programs. Lin *et al.*[16] have measured the data switch operating at between 2–10 Mbytes per second, with latencies of 7.8 or 167 μsec plus data-transmission time, and overhead times of 3.4 or 302 μsec, depending on the message passing library. With high message passing overhead and long latencies a computer is limited to coarse grain apps since each μsec of overhead loses the opportunity to carry out 100+ floating-point operations. While subsequent computational nodes can evolve to higher performance such as those provided by Meiko and greater memory size, a next generation CM5 requires a proportional increase in the communica-

tion network. It's unclear whether CM5s' wire limited switch can evolve as rapidly as its microprocessor-based nodes to provide generation scalability.

2.2.2.3. Meiko CS-2. CS-2 is an exemplary medium grain multicomputer. Meiko, founded in 1985 by the INMOS Transputer development team, has the most experience with multicomputers, having delivered the largest number of computers as a profitable start up. The CS-2 is important for several reasons, and is likely to outperform other medium grain multicomputers. Each processing element is a SPARC computer running SUNs' standard Solaris operating system. Existing apps without recompilation. Program compatibility is important because it determines whether a computer has initial utility, synergy with other markets, and a basis for evolution to parallel processing. Meiko has designed a general purpose computer for the commercial and technical markets, depending on which processing elements are bought, subject to multicomputer limitations—each node is independent and cannot easily share memory resources. The network design addresses these important problems: task-to-task bandwidth, latency, and processor overhead; direct addressing of remote node memories to load-store data; $n+1$ redundancy and fault tolerance.

The CS-2 is a symmetrical (homogeneous) multicomputer with up to 1024 processing elements (a large, printed circuit board) in 4 expandable configurations of 16, 64, 256 or 1 K elements. An element can be one of three types: a SPARC processor and two, 100 Mflops (200 Mflops single precision) true vector processors, a SPARC processor and I/O channels, or four SPARC processors. A SPARC processor operating at 50 MHz provides a peak of 150 Mips, 50 Mflops, or 80 SPECmarks. Four elements are interconnected to form a module (a small cabinet) and modules are interconnected using a backplane network switch. Subsequent processing elements, CS-2, include a SPARC processor with 4 vector processors, giving a peak of 500 Mflops.

The CS-2 network switch operates at 70 Mhz providing 100 Mbytes/sec full-duplex using a multyi-stage network consisting of 8×8 cross-point switches. The worse case path in a 256 node system requires 7 hops introducing 1.4 μsec of node-to-noide latency. The time required for a user process to transfer 1 byte of data to or form a user process on another node is 10 μsec with 1 μsec of processor overhead. The data transfer is carried out entirely by hardware with no processor intervention once the transfer is initiated. The latency to send and received a message is $2 \times 4.3 + 1.4$ μsec + the message transmission time at 0.02 μsec/byte with only 2×0.5 μsec of lost time.

The CS-2 programming environment is determined by Solaris. The parallel processing is carried out using both data parallel (SPMD model

using HPF) and multi-process programming. Higher level primitives include a global memory model (virtual shared memory—DSM), and various standard low level and high level application libraries.

The CS-2 commercial environment is an Oracle Parallel Server using a parallel file server and parallel lock manager. The load balancer allocates incoming requests for service (instances) to the database, transaction processing, and various apps, e.g., report generation. The CS-2 exhibited linear speed-up on Oracle apps.

2.2.2.4. Fujitsu's VPP500 supercomputer. The VPP500 is a medium-coarse grain, inhomogeneous multicomputer with 7 to 222 1.5 Gflops vector supercomputer nodes, each with 256 Mbyte memories, that connect to a Fujitsu VP2000 supercomputer. Because nodes are so powerful, a factor of 10–20 fewer nodes achieve the same level of performance as a computer using CMOS micros, thus lessening the effect of Amdahl's Law. Only 64 nodes achieve 100 Gflops. On the other hand, fast nodes require a lower latency, lower overhead switch as compared to microprocessor based multicomputers. The 400 Mbytes/sec, low latency cross-point switch and interface manage process-to-process data transmission without processor intervention. VPPs principal advantage is that it can achieve incredibly high throughput by using a single node, thus it can be used effectively as a workload computer that requires little or no parallelization beyond vectorization. A 140 node VPP operated at over 100 Gflops in 1993.

2.2.3. Scalable, Fine Grained Multicomputers

The Seitz, Caltech group that invented the first Hypercube multicomputers and 2D routing chips used by the Paragon and DASH computers, has been following a fine grain approach. The 500 K, transistors Mosaic-C chip contains a 64 KB DRAM, a 16-bit 11 Mips multi-threaded processor, and a 60 MByte/sec 2D router. Mosaic-C operates with two threads, one for computation, and the second to control the router and memory interface. An 8 by 8 array of chips is mounted on an $8'' \times 8''$ board. The focus of the research is to assemble large, dense 2D and 3D arrays that provide substantially more power and bandwidth than coarse grain multicomputers. The fine grain aspect requires a reduction of communication overhead by completely overlapped with execution. The latency for a 20 Byte packet to only 44 instructions for a 16 K node configuration, using the chips no overhead, high bandwidth switch. The next generation Mosaic has an additional thread for controlling peripherals, such as memory or another processor, a 32-bit processor operating 40 Mips, a 128 KByte memo, and 200 MByte/sec channels.

While this approach is radical departure from off-the-shelf microprocessors, it tailors silicon to the task. By having several threads, it will be possible to provide various computational structures including multiprocessors that require substantial computation for memory management. Mosaic will be successful, and most likely change the direction of multicomputers. One of the first applications for the Mosaic is an ATM switch that uses nodes to interface the communication lines and to route the packets among the nodes.

A similar effort at MIT, by Daly, formerly of Caltech, on the J-Machine with 64 K nodes provides further impetus for the fine grain, large numbers of nodes approach. nCUBE also follows this line.

2.3. Multiple Instruction Streams: Multiprocessors

The author believes that multiprocessors are the main line of computer systems development (Bell[7]) based on history and because they offer the most flexible use of resources by providing all the resources in a single, fungible, pool. Furthermore, 1992 multicomputers operate as limited multiprocessors with limited capability, distributed shared memory DSM software. The following section will elaborate this reasoning about this evolution.

2.3.1. Traditional Supercomputers and Limited-Scalability Multiprocessors

Two alternative switches, the bus and cross-point switch, are used to build limited-scalability multiprocessors. A single bus is the simplest way to build a multiple microprocessor or "*multi*" (Bell[15]). With the evolution of microprocessors to support "multis" any computer from the simplest PC becomes a multiprocessor. The bus that's formed by printed wiring limits size scalability to roughly 30 and generation scalability. Multis "work" because each processor has a cache that serves three functions: it provides fast memory for the local processor, assuming the data or instructions are in the cache; by being in the cache, the switch (bus) is not required, and other processors may use it; and finally, as data is moved between the processor and memory, each processor can observe or "snoop" on the bus transactions in order to invalidate any data that the cache may hold, thereby providing a coherent shared memory. Cache size, data-ownership, and protocols to invalidate and control data-ownership in limited scalability multiprocessors are relatively well-understood (Hennessy and Patterson[18]). In 1993, nearly all microprocessors are capable of being components for a "multi." All manufacturers offer "multis"!

In the future, the bus could be replaced by a ring, providing the essential features of a bus, but scales automatically with generation, i.e. clock speed since each processor only drives a neighbor. Although the bandwidth for a "*ring multi*" is fixed by the number of bits that flow through a given node, the number of nodes can be increased arbitrarily, but at the expense of increased latency. Increased latency is an unfortunate by-product of scalability that has to be dealt with in operating systems, compilers, and apps—hence, the ring could be a "natural" structure.

"True" supercomputers from Cray Research, Fujitsu, Hitachi, IBM (for accelerating mainframes) and NEC use the Cray design formula: ECL circuits and dense packaging technology to reduce size, use the fastest clock; one or more pipelined vector units with each processor provide peak floating-point processing for FORTRAN programs; and multiple vector processors communicate via a switch to a common, shared memory to handle large workloads and parallel processing. Because of the dense physical packaging of high power chips and relatively low density of the 100,000 gate ECL chips, the cost per operation for a supercomputer is roughly 500–1000 peak flops/$ or 4–10 times greater than simply packaged, 2 million transistors CMOS microprocessors used in workstations (5000 peak flop/$). Future supers may evolve to GaAs or bipolar circuits with better performance and power dissipation characteristics.

Mainframes and supercomputers use high-bandwidth, cross-points and multi-stage networks (the "dance hall" structure) to interconnect processors and memories (Fig. 3). The high cost switch, back panel, and

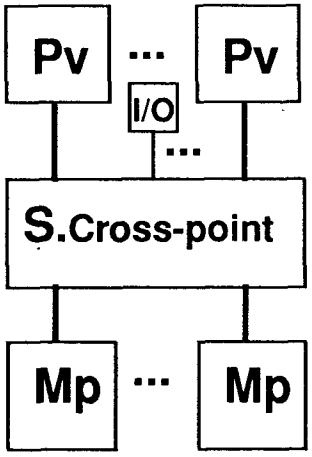

Fig. 3. Multiple vector processor supercomputers (e.g. *Cray, IBM, Fujitsu, Hitachi, NEC*).

power supplies for the processor and memory increase the fixed cost, thereby limiting the scaling range to 4–6. Cray offers three models with varying ranges of expandability in its C90 line to cover 1–16 processors. Increasing the switch bandwidth increases the latency between processor and memory which in turn reduces the scalar performance, making a computer less general purpose. For example, the memory access time has remained relatively constant at about 100 nanoseconds over three generations (XMP, YMP, and C90), even though the cycle time has been halved to 4.1 ns. In order to sustain the pipelined execution of independent, worst case statements such as $A_i = B_j + C_k$ that compute one floating-point operations per second (flops), three memory accesses (aps) are second required. The switch, together with the market size for a computer costing over $30M, limits how large a super can be.

Mainframes are essentially supercomputers without vector units, although IBM's mainframes with 1–6 processors do have vector processor attachment options. Since mainframes have lower latency and higher scalar performance than supers, they are "more" general. Mainframes require less bandwidth compared to supers, and maintain memory coherence using processors with caches and directories.

2.3.2. Non-Uniform Memory Access (NUMA), Memory Coherent, Scalable Multiprocessors

The taxonomy and evolutionary time line show (Figs. 2 and 4) the scalable mP evolution starting with the CMU Cm* (c1975). This first scalable computer was constructed as a hierarchy of buses that interconnected computer modules formed from DEC LSI-11 microprocessor and memory nodes. Since the computer nodes had no caches, memory coherence was maintained automatically because only copy of a memory existed. The BBN Butterfly (c1985) computer nodes were interconnected using a high speed switch enabling a processor to access data in any node via the switch. Both computers reverted to static allocation of programs that communicating by passing messages. The CEDAR project at the University of Illinois on CEDAR, based on a distributed hierarchy of Alliant vector multiprocessors, showed that an efficient compiler could automatically control data migration. These multiprocessors had a single global address space and nonuniform memory access architecture (NUMA), where remote nodes required several times longer access times than local nodes. The common finding was the difficulty of statistically allocating programs and memory to nodes, and communicating results among computational threads.

The "multis" structure appeared to be a solution to the static binding

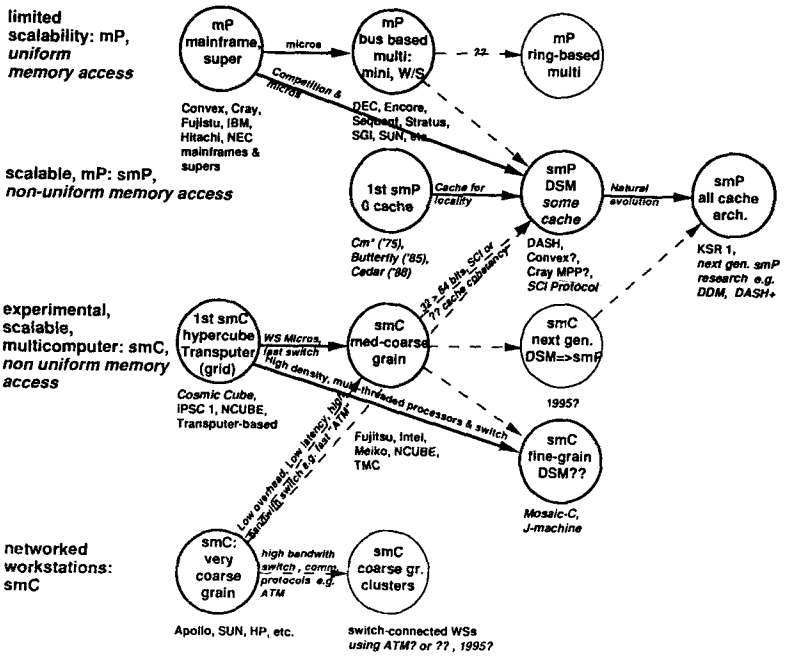

Fig. 4. Evolution and projected evolution of scalable multiprocessors, multicomputers and workstations.

problem. The small processor cache could provide locality to hide latency caused by accessing remote memories inherent in a non-uniform memory access computer. Stanford's DASH is the first scalable computer to use the cache in this fashion.

2.3.2.1. The Stanford DASH. DASH (Hennessy[19]; Lenoski *et al.*[20]) has up to 64 processors arranged in a grid of 4×4 nodes interconnected by Caltech's routing chips. Each node consists of a four processor Silicon Graphics multiprocessor. The access time for various accesses is roughly 1 (local cache), 10 (second level cache), 20 (local memory), 60 (remote memory), and 80 (remote memory with data located in another module) clock cycles depending on the location of the data.

DASH demonstrated linear speedups for a wide range of algorithms, and is used for compiler research, including positing the SPLASH parallel benchmark suite (Singh *et al.*[21]). Some apps ran at over 100 Mflops for 16 processors, roughly equal to a four vector processor system with 1/2 the PAP.

2.3.2.2. The IEEE Scalable Coherent Interface (SCI). SCI is a protocol and IEEE standard to interconnect computers to operate as a single, shared memory multiprocessor. The SCI communication protocol can be carried on a ring, like KSR, or through a central switch. Pointers in each memory track data as copies migrate among nodes, causing much traffic to maintain coherence when a large amount of sharing occurs. Although the standard is well documented and interface chips are available, the SCI scheme has yet to be implemented in a system.

2.3.2.3. Convex MPP. The Convex MPP will be introduced in 1994 using a fast switch to interconnected hundreds of off-the-shelf HP PA RISC processors. Convex will use some form of the SCI work to interconnect nodes and maintain memory coherence. The design has four goals: provide a fast switch such that the nodes appear to have a single memory with the lowest possible inter-node memory latency; to be able to run dusty deck FORTRAN 77 programs without modification through automatic parallelization—to not force users to convert programs to HPF; offer a scalable system that is no more than 15% more expensive than comparably priced workstations; and to use unmodified, single-threaded application programs that run on the HP RISC. In 1993, an HP RISC processor provides 150 SPECmarks of performance. By 1995, the leading edge microprocessors such as HPs' RISC are expected to follow Moore's Law[6] and deliver 300–500 SPECmarks[7] and peak Megaflops.

2.3.3. The Cache-Only Architecture: Evolution from No Cache and Some Cache

It should be clear from the evolution of scalable computers, that the greatest problem in a distributed memory computer, independent of whether it is an mP or mC, is the need to constantly move data and programs in the distributed memory to correspond to the static and dynamic loading of a workload or a particular job. Also, to be able to use any collection of resources (i.e., processors, memory, or I/O) for a single job or workload. Ideally, the entire distributed memory is viewed as a single virtual memory.

[6] Moore's Law states that semiconductor density doubles every 1.5 years, or 60% yearly improvment. For RISC microprocessors, it has been observed that performance has also increased at 60% per year.

[7] A benchmark consisting of two parts that measure integer and floating-point performance. A Specmark is measured in terms of the performance of the first VAX 11/780 computer that was introduced in 1978!

For example, High Performance FORTRAN introduces directives that require a user to control locality by causing just-in-time, pipelined movement of data from node-to-node for subsequent use. These primitives force apps programmers to become machine language coders who are required to understand machine idiosyncrasies and express pipelining in FORTRAN syntax. While explicit mechanisms to pipeline data to a remote node for processing are better dealt with by compiler technology, they provide an excellent motivation for a programming model based on a single, shared memory. In order to efficiently use multicomputers, it may be necessary for programmers to manage the allocation of work using a message passing programming model. Intel, Meiko, and nCUBE mainly support this programming style.

The situation of programmer-managed memory is similar to the situation in 1960. The belief was that programmers, and eventually compilers or their run time systems should be able to move data around in an "overlay" fashion between a secondary and primary memory in order to reuse memory and have data in the right place at the right time. This condition stimulated the invention of Atlas's one level store and virtual memory that allowed a smaller physical memory to hold just the active portion of a user's much larger virtual memory. The Atlas paging mechanism also provided for sharing of programs and data, and efficient multiprogramming. Even with large physical memories, virtual memories are still much larger than physical memories as they hold files and the operating system. A paged memory is important for allocating memory among jobs. However, the most important part of virtual memory is locality as embodied in the concept of the working sets (Denning[22]) and hardware managed caches. These aspects of virtual memory and caches are what the all cache architecture uses to "cache" the active portion of a program and automatically exploit temporal and spatial locality.

Figure 5 shows the structure of a distributed shared memory computer and the sharing and replication of data among various nodes. The cache only architecture, also known as COMA, is a generalization of virtual memory, and essential for a distributed memory architecture. Like virtual memory, it first eliminates the notion of a physical memory location that a program uses to address data items; the distributed physical memory is a place(s) to hold (map) virtual memory locations. When only a single physical memory is used to map a virtual memory (usually held on disk), a coherent view of memory is easily maintained. However, when multiple copies of the virtual memory are replicated, several problems arise: maintaining a single view for all processors (like multiprocessor caches), and moving data among the various nodes that require access.

Cache only, a natural extension of virtual memory and multiprocessor

caching, first permits a single data item to exist in more than one location at a time. Once a memory page is brought from secondary memory to one of the nodes, hardware and software automatically move, replicate, and control data flow with other nodes on an element basis e.g. the KSR 1 moves subpages of 128 bytes. This element by element movement is not

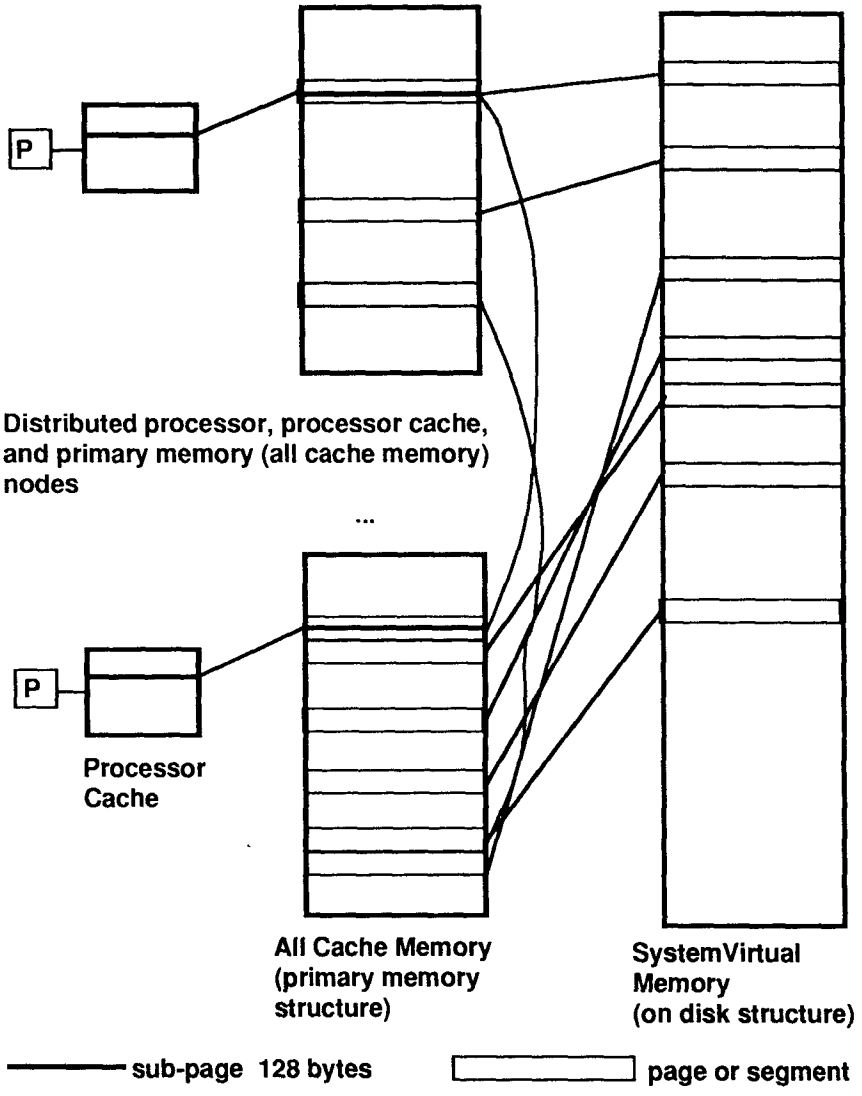

Fig. 5. Structure of a distributed shared memory "all cache" architecture.

possible without fundamental changes to the hardware and software paging mechanisms. A conventional paged virtual memory mechanism moves pages between one physical primary and one secondary memory, whereas the cache only architecture moves smaller elements among the distributed nodes in response to each processor's needs. Thus, unlike an mP with some cache, e.g., DASH, the memory of a cache only computer node will automatically and gradually be replaced on an element by element basis as needed in order to be associated with the processor.

Gupta *et al.*[23] show the cache only architecture outperforms the DASH architecture, and describe a variant of the KSR 1 hierarchical multiprocessor based on interconnecting processor-memory pairs through a central switch. Hagersen[24] gives an excellent description, motivation, and simulated performance characteristics of the cache only architecture, together with a description of the experimental Data Diffusion Machine being built at the Swedish Institute of Computer Science.

2.3.3.1. Kendall Square Research (KSR). The KSR 2 is a size and generation scalable, shared memory multiprocessor computer (Frank *et al.*[25]; KSR[26]). It is formed as a hierarchy of interconnected "ring multis." Scalability is achieved by connecting 32 processors to form a "ring multi" operating at one gigabyte/second (128 Million accesses per second). This current generation ring has roughly the capacity of a typical cross-point switch found in a supercomputer room that interconnects 8–16, 100 Mbytes/sec HIPPI channels. The KSR 2 uses a 2 level hierarchy to interconnect 34 rings (1088 processors). The ring design supports an arbitrary number of levels, permitting computers with 1000s of processors to be built.

Each node is comprised of a primary cache, acting as what would normally be a 32 Mbyte 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 are designed for both scalar and vector operation. Sub-pages of 128 bytes, corresponding to the size of all cache elements can be prefetched at one time. A processor also has a 0.5 Mbyte sub-cache supplying 40 Million accesses per second to the processor. 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 elements to and from other nodes, maintaining memory coherence throughout the system using distributed directories, and ring control.

The KSR 2 is significant because it provides: size (including 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 programming model; and dynamic management of memory through hardware migration and replication of data throughout the distributed, processor-memory nodes using its ALLCACHE™ mechanism.

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 multi-threaded 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 sent to other nodes to reduce access time. A processor can pre-fetch data into a local cache and post-store data for other cells. The hardware is designed to exploit spatial and temporal locality. For example, in the SPMD programming model, 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 pre-fetch data to the processor's registers. When a processor writes to an address, all cells are updated and memory coherence is maintained. Automatic data movement occurs in sub-page elements of 128 bytes (16 words) of its 16 K byte pages.

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

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 K node system provides almost thirty times more processing power, primary memory, I/O bandwidth, and mass storage capacity than a multiprocessor mainframe. The 32 and 320 node systems are projected to deliver over 1000 and 10,000 transactions per second, respectively, giving it five to ten times the throughput of the most powerful multiprocessor mainframe.

2.4. The Alternatives and Trade-off

Table III lists the alternative structures together with their advantages and limitations.

Table III. Computer Alternatives, Advantages and Limitations

Which	Grain	Advantages	Problems
SIMD *MasPar*	f	// dusty decks, vector apps	scalability, scalar, non-//
Supers *Cray, Fujitsu,...NEC*	f	>scalar, workload, vector apps, existing apps, understanding, user base	PAP, perf/$, PR
smP.all cache *KSR 1*	m	PAP, perf/$, >workload	fine grain apps
smP *Cray T3D?, Convex*	f-m	PAP, perf/$, workload	entry cost
smP.multi-thread *Mosaic C*	f	>PAP, >perf/$	poor scalar fp, workload
smC.mt (Tera)	f	PAP, perf/$, fast mPsimulator	no experience doesn't exist
smC.homogeneous*	m-c	PAP, per/$	fine gr. apps, workload
smC.inhomogeneous*	m-c	PAP, perf/$	fine gr. apps, nil workload

A SIMD is perhaps the easiest computer to use since programs have a single thread of control. A SIMD can take advantage of existing highly parallel, e.g., two-dimensional arrays for images, and is a highly tuned server for massive data parallelism. While such a computer could scale to 32–64, its single program, slow scalar and inability to handle a general workload, limits it to a highly parallel apps server.

Supercomputers provide the highest performance for apps that are not fully (i.e., >99% parallel) including scalar problems. In 1993, users have over 15 years of experience with five generations of vector supercomputers, algorithms, and libraries of highly turned apps. The supercomputer lacks having the highest PAP by a factor of 8. The Public Relations budget for supercomputers is small in comparison to the hype associated with HPCC and massive parallelism with 1000s of processors, 100s of Gflops, and the race to be first with a PAP of one teraflop. In 1993, based on a look at delivered performance, measured in flops or solutions per month, there is no alternative to a super for a wide range of workloads and apps. A C90 processor delivers a harmonic mean of 64 Mflops on Livermore's FORTRAN Kernels (LFKhm) for a total workload capability of over one Gflops.

The KSR 2 smP has a PAP of 1088×80, or 87 Gflops. This is 5.4 times the PAP of a C90, and is a factor of 6 times more cost-effective based on PAP. A KSR 2 processor delivers 13 Mflops, or one-fifth a C90 processor for LFKhm. However, the aggregate workload capacity is 14.1

Gflops, or 14 times the C90 provided all 1088 processors can operate individually on a workload. The KSR 2 appears to provide superior performance and workload characteristics for the NAS benchmarks.

The Convex and Cray MPP multiprocessors are likely to have very high PAP and exceptional performance/$, although no results are available in early 1933. How well each performs for large jobs and dusty decks will depend on how closely each is to an evolved multiprocessor that can provide a single, large shared memory with fast inter-node communication, at low latency and low overhead. Clearly these computers will evolve to the all cache structure.

All of the multicomputers have very high PAP and PAP/$ because they consist of a collection of an appropriate number of floating-point unit. The greatest problem for multicomputers is their inability to process a workload with varying processing (from scalar to partially parallel) and memory greater than a single node. Inhomogeneous computers, like SIMDs have negligible workload capacity, making them unsuitable, general purpose computers. They are batch machines for large jobs.

Multicomputers have yet to provide uniquely high performance on a diverse range of existing or real apps, benchmarks, or kernels beyond matrix multiplication and linear equation solving (Linpack), despite their high PAP and 10 year existence. For example, since 1991 Intel's Delta, a prototype for its 1993 Paragon, has operated at Caltech to serve about 150 users. Delta's availability was 90%, operating half time in debug mode and half time in production mode serving a dozen and 1–4 users, respectively. Production programs run at 1–5 Gigaflops, out of 30 Gigaflops PAP, using a limited I/O and file system. The user load on Delta could be satisfied by 2–4 processors of a 16 processor Cray C90, and users would have been spared the rewriting of programs. What each multicomputer appears to provide in PAP, it loses in communication, imbalance, inadequate secondary memory, and programming environment. The greatest loss is that users must search for new algorithms and rewrite programs for an experimental computer.

Networked workstations offer the greatest potential because they already exist in an environment, and do not require additional expenditures. In 1993, networked workstations provide inherent, very coarse grain parallelism. However, today's workstations are inadequate for main line parallel processing because of the high overheads and latencies of their interconnections. These can best be characterized as very coarse grain because the fastest networks are 100 Mbits/second, with message latencies and overheads of 500–1000 μsec. The fact that Meiko uses SUN workstations and the Solaris operating system is an existence proof that

workstations can evolve to become medium grain smCs. The HPCC program should begin to carry out the R&D to utilize our workstations, great untapped resources.

3. EVOLUTION OF smPs, NETWORKS, AND smCs TO smPs

Multiprocessors, networked workstations, and experimental scalable multicomputers will follow three, distinct evolutionary paths (Fig. 4). First, the main line of super, mainframe, and minicomputers will continue to be the multiprocessor. A major transition *can only* occur if limited scalability mPs can be replaced by fully scalable mPs for a significant number of apps, including commercial transaction processing and databases. This requires a commitment by suppliers and users to understand and handle the latency that scalability implies. Second, the smP will evolve to the cache-only scalable architecture. Networked, distributed workstations will evolve to support finer grain apps ranging from shared objects and servers, to parallel processing as fast, low latency networks replace slow, high overhead LANs. The payoff for this transition is immense because it allows existing workstations to be used in parallel. Finally, the plethora of experimental multicomputers will bifurcate to become either smPs (using more hardware and software) or fully distributed, networked workstations. The Caltech fine grain multicomputers provide opportunities for switches, computer components, and parallel computing even though they are unlikely to influence existing multicomputers in the near term.

3.1. Multiprocessor Evolution: Supercomputers, Mainframes, and Minicomputers

In 1993, multiprocessor computers in these computer classes comprise 1.8%, 22%, and 23% respectively of the computer market (by $s). Multiprocessors will evolve as follows:

1. limited scalability multiprocessors with more processors and greater scaling range including fully scalable computers
2. bus-based multis may evolve to ring-based multis
3. next smP computers evolve to the all cache architecture, like the KSR for efficiency
4. mainframes and supercomputers slowly transition to be an smP. The supercomputer will be replaced slowly because it has no inherent communication latency or overhead and has infinite

bandwidth among processors. Convex's or Cray's MPP could be the archetype for future supers. The scalable mainframe will be replaced slowly, constrained because by lethargic designers and "code museum" system software.

3.2. Workstation Evolution

In 1993, workstations and PCs, comprise 9% and 45% of the computer expenditures. These computers present the greatest untapped resource for using a single set of resources and parallel processing. Networked workstations and PCs interconnected by LANs such as Ethernet of FDDI (100 bits/second) and providing very coarse grain parallelism with 500–1000 μsec latency and overhead are likely to follow these evolutionary paths using:

1. faster LANs or ATM switches to provide bandwidths of 155–622 Mbits/sec. Interconnecting networks using faster LANs and switches with lower latency, lower overhead (100 μsec), could support coarse grain parallelism thereby increasing workstation utility for a range of apps, including executing distributed, shared objects.
2. workstation clusters. IBM is delivering their SPI multicomputer with up to 1 K workstations, *sans CRT*, in 1994 using a switch with <10 μsec overhead/latency. Meiko's CS-2 is an archetype. A well-designed "ATM" switch, e.g., Digital Equipment Corp., and interface provides this capability.
3. direct evolution to an smP. SGI has announced a 32-processor mP with limited scalability. Stanford's DASH that uses SGI workstations provides a feasible prototype for an smP.

3.3. Experimental Scalable Multicomputers for Parallel Processing

The future of multicomputer is unclear because they are a complete artifact of federal HPCC funding, and remain experimental with no infrastructure or motivation for attracting standard software apps. In 1992, they constituted 0.250% of the computer marketplace despite the fact their funding support level has exceeded their market value for a decade. These research funds for APRAs' State Computers constitute about 25% of federal computer engineering and computer science research.

Chuck Seitz,[27] the inventor of the multicomputers has expressed reservations about the viability of commercial multicomputers:

> "I believe that the commercial, medium grained multicomputers aimed at ultra-supercomputer performance have adopted a relatively unprofitable scaling track, and are doomed to extinction. With their relatively modest number of nodes, they may as Gordon Bell believes be displaced over the next several years by shared memory multiprocessors. For loosely coupled computations at which they excel, ultra-super multicomputers will, in any case, be more economically implemented as networks of high-performance workstations connected by high-bandwidth, local area networks such as ATOMIC.[8]"

I concur. Experimental, multicomputers must evolve to multiprocessors using hardware or hardware/software and solve the basic problems (see Table III) inherent in special purpose computers. These computers offer little compared to *existing* very coarse grain workstations that users already have. As described earlier, workstations interconnected by faster LAN-based switches such as ATM or the Scalable Coherent Interface will eliminate smC as such. The evolution of scalable multiprocessors described above provides the template as to the motivation for main line parallel computing evolution that will co-exist with distributed workstation evolution.

Figure 4 shows several evolutionary paths for experimental smCs:

1. straight-forward evolution to either a scalable mC using next generation microprocessors. This is an evolutionary path based on improved design of 64-bit micros and lower latency, lower overhead, higher bandwidth switches. With better nodes and much software, the program environment may gradually approach an smP.
2. direct transition to smP with a DSM is the most likely scenario using 64-bit micros. The DASH distributed directory scheme provides a prototype for this transition.
3. revolution, following the work of Seitz and Daly to a custom, truly fine grain multicomputer that would permit much higher density with at least an order of magnitude more processors. Unfortunately, none of the State Computer companies are likely to be able to afford to move very far off their evoluationary paths because of microprocessor, software, and organizational (NIH) constraints. To follow this potentially high payoff path would require a substantial increase in parallelism, new software, and perhaps more user retraining.

[8] ATOMIC, for ATM over Mosaic, is a switch built by Danny Cohen at the University of Southern California Information Sciences Institute (USC/ISI) from Seitz's Mosaic fine-grain, multicomputer modules.

4. APPLICATIONS PARALLELISM, BENCHMARKS, AND COMPUTER CHARACTERISTICS

Before trying to understand computer performance, it is critical to look at the various forms of parallelism that computers can and must exploit for generality and viability. The most basic form is at the workload level using multiprogramming where a common pool of computational resources (processing, primary and secondary memory, networking) is available to trade-off among a large job mix with varying degrees of parallelization (including completely scalar operation). A computer's resources should be fungible on an instantaneous basis in order to use any resource for work. A workload should be processable by multi-processing a common job queue with a common, parallelized file system. The greatest potential parallelism is available at very coarse grain through: batch processing and on-line report generation (analysis) against a parallelized database time-shared, multi-programmed servers for LAN-based, *X*-windows clients and arbitrarily large transaction-processing systems using a common parallelized database and transaction apps servers for 1000s of on line users.

For peak performance of a single job, two forms of parallelism may be required: transparent (or implicit); and explicit, multi-process parallelism where the user is required to formulate a job in terms of both functional and data parallelism. Ideally, a parallelizing compiler such as FORTRAN, processes dusty decks from supercomputers in an implicit fashion. Like supercomputer apps, such programs must be properly structured to exploit the parallelism. To support these environments requires multiple threading or light weight process that share a single address space. For multicomputers, this implies a DSM. Multiple job streams and multi-processing of jobs that operating system mechanisms allow, using the pipes and sockets mechanisms, provides great opportunities for parallelism. Shared memory facilitates these mechanisms in conventional single memory computers.

Figure 6 shows the structure of a basic unit of multi-threaded computation independent of whether it's run on a SIMD, multiprocessor or multicomputer, or has distributed or shared components. The computation starts with a sequential thread (1) that includes job scheduling and other serial computation. A basic loop starts with supervisory scheduling (2) following by threads of the computation (3) and inter-computer message (4) phases of a thread. The synchronization part (5) occurs prior to returning to scheduling the next unit of parallel work (2).

Multicomputers have networks to assist scheduling and synchronization. Multiprocessors use various semaphores implicit in shared memory to facilitate scheduling and synchronization. Multicomputers may provide

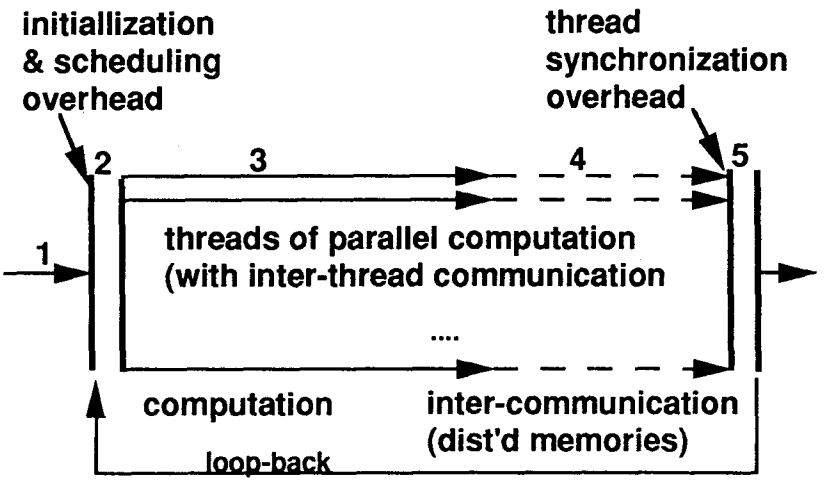

Fig. 6. Structure of multi-threaded, parallel computation.

hardware, including multi-threaded processors, to transmit messages to reduce processor overhead and allow for covering latency. Multiprocessors take negligible time to send messages because these are just load store instructions, but they must have pre-fetch and post-store operations to cover the delays (latency) inherent in waiting for responses through an intercommunication network. Furthermore, since message-passing libraries are likely to repeatedly copy messages, using a multiprocessor that passes pointers to messages dramatically reduces message passing overhead. Each of these machine/app parameters will be described here as they limit grain size.

4.1. Benchmarks and Real Application Performance (RAP)

The benchmarking process has been a key to understanding computer performance until the teraflop race started and PAP replaced reality as a selection criterion. Computer performance for an installation can be estimated by looking at various benchmarks of similar programs, and collections of benchmarks that represent a workload (Berry *et al.*[28]). Benchmarks can be synthetic (e.g., Dhrystones and Whetstones), kernels that represent real code such as Livermore FORTRAN Kernels (i.e., Loops), NASAs' National Aerodynamic Simulation (NAS), numerical libraries (e.g., Linpack for matrix solvers, FFT), or fuller apps such as SPECmarks for workstations, Illinois' Perfect Club, Stanford's SPLASH, and Los Alamos Benchmarks.

Livermore Loop metrics provide a range of performance indicators: the arithmetic mean—the best case for apps (0.97 vector ops), the arithmetic mean—optimized apps (0.89 vector ops), geometric mean—tuned workload (0.74 vector ops), harmonic mean—untuned workload (0.45 vector ops), and harmonic mean compiled scalar—all scalar operation (no vector ops). The NAS benchmarks (Bailey *et al.*[11]) indicate the difficulty of running benchmarks, since NAS benchmark results are only available in late 1993, despite the fact State multicomputers have been announced since October 1991. Three Linpack measurements are important: Linpack 100×100, Linpack $1{,}000 \times 1000$ for typical supercomputing applications, and Linpeak (for an unconstrained sized matrix). Matrix multiply and Linpeak benchmarks provide best case results for large, scalable multicomputers because they can perform efficiently, given a large matrix and enough running time.

Montry conjectures that the degree of parallelism in a program varies with problem size and inversely with program size. This characterizes the dilemma for running large benchmarks and existing apps. For example, assume with modest effort on 1% of a large program, taking 95% of the running time, can be completely parallelized to run infinitely fast to give a factor of 20 speed-up. It is likely that the remaining 5% of the time and 99% of the code size will slow down a factor of 10 because of slower scalar operations. Overall speedup is two.

Massive multicomputers rarely run an existing supercomputer app (i.e., dusty decks) without a new algorithm and new program including the necessity to use a multi-process model. Thus RAP is almost impossible to characterize. This flaw, together with the lack of understanding about app scalability for scalable machine characteristics, guarantees a negligible app market using existing, third party software vendors.

No matter what measure is used to understand a computer, the only way to understand how a computer will perform is to benchmark the computer with the applications that will be used. This also measures the mean time before answers (mtba), the most important measure of productivity for all computers. The fastest benchmark to run on any computer is whether a manufacture can and is willing to benchmark a program.

4.2. Metrics and Balance for Scalable Computers

Although it is dangerous to examine the basic characteristics of a computer such as PAP, useful conclusions can be drawn by considering machine characteristics. Some characteristics such as secondary memory size and speed are needed simply to size whether a computer can store or transfer data in time. Table IV shows resources estimates for several Grand

Table IV. Resources, Time, and Secondary Memory for Solving Several Grand Challenge Problems on a Computer with a PAP of 128 Gflops PAP and RAP of 20 Gflops

Project	Tera-ops	time*	Mp.By	Ms.By	Ms/.Mp	ops/By	$.8mm	$.Ms
Global Ocean	0.1M	170d	4G	20T	5000	500K	$100.K	$40.M
Porous Media	1M	1700d	1T	4T	4	250K		$8.M
Ductile Materials	1M	—	20G	3T	150	330K		
	1M	—	-T	20G	20	50M		
Plasma physics	1M	—	1T	100T	100	100K	$500.K	$200.M
Brain Topology	100	1/6d	15G	1G	1/15	6K		
QCD	1M	1700d	8G	8G	1	125M		

Challenge problems that are being addressed by Department of Energy's Los Alamos and Oak Ridge Laboratories. The following section describes key parameters.

4.2.1. Workload, Generality, and Number of Users: Real Application Performance (RAP)

The simple test for generality is to look at the these benchmarks, including file I/O and transaction processing rates. Each of the benchmarks gives an indication of performance. While highly unbalanced machines can have very high PAP, the best case RAP is substantially less. For example, Bailey *et al.*[11] site supercomputers providing 50% of PAP; whereas multicomputers deliver 2% to 5%. As a minimum users must look at the Linpack, Livermore Loops, Los Alamos, and NAS benchmarks together with app benchmarks for their domain.

If the computer is to be used in a general purpose environment, it must have the ability to handle a wide range of work and accept many users. The inhomogeneous computers are fundamentally special purpose, limited by their host. Along with handling a very large workload, scalable terminal access is critical, using workstations and Xterminals.

The final single test for a general computer is: Can a variety of users place work, requiring varying degrees of parallelism (including none), amounts of memory, and I/O and have this work run efficiently?

4.2.2. PAP: Peak Advertising Performance (PAP) Measures

When ARPAs' program was initiated the metric was operations per second, but as it became clear that high visibility apps were scientific, float-

ing-point ops became the metric. Occasionally instructions per second are important because MIMDs execute large numbers of instructions. A high Mips measure rewards RISC and penalizes vector processors designed for floating-point.

Hockney and Jesshope[29] posit performance parameters for vector and distributed memory computers, including several that depend on program characteristics. These allow estimates of performance under varying conditions, and include peak processing rates for various vector lengths, and scheduling, synchronization and communication overhead times.

4.2.3. Primary Memory per PAP

In the 1960s, IBM engineers posited the heuristic that a 1 Maps computer required 1 Mbyte of memory for batch processing environments before multiprogramming and virtual memory. In 1993 balance heuristics are nonexistent. It's hard to believe the requirements shown in Table IV are typical. The C90, KSR 2, and CM5 computers provide 32 Gigabyte memories for PAP of 16, 80, and 128 Gflops, (2, 0.8, and 0.25 bytes per flop) respectively.

Supercomputers have large, lower cost buffer memories built from dynamic random-access memories (DRAM) that are used for swapping, whereas primary memory is built from small, expensive, static random-access memory (SRAM). Scalable computers use DRAM memories.

4.2.4. Memory Bandwidth

No matter how much PAP a computer can claim, the bandwidth between the primary memory and arithmetic units limits many programs. Computational intensity is the number of memory access (map) per floating point operation (flop) required for a statement(s) or a program. For example, the computational intensity of the expression $Ai = Bj + C_k$ is 3, since 3 memory accesses are required for every flop. The inner loop of Linpack is a DAXPY $(Ax_i + y_j)$ instruction that requires 3 maps per 2 flops (CI = 1.5). For example, a C90 provides 1.5 maps per 1 flop allowing a C90 to run at full speed on the worst case Linpack benchmark, whereas a CM5 with 0.5 Maps/Mflop will operate at 1/3 of PAP.

4.2.5. Secondary Memory Size, I/O Bandwidth including Network and Terminals

In addition to secondary memory size, I/O bandwidth is likely to limit some applications. On the basis of Grand Challenge problem estimates

(Table IV) a wide range of memory sizes are required. Concurrent with scalability measured in terminals (or simultaneous users), terminal bandwidth must be scalable in size and with generation. By the end of the decade, the terminal of choice will undoubtedly be the HDTV, requiring architecture changes to support high bandwidth digital video.

4.2.6. Granularity Parameters: Scheduling-Synchronization, and Message Latency/Overhead

Figure 6 illustrated the basic times that determine the performance for parallel apps of a given granularity and number of processors. The times come from the computer and application. The time to schedule a new computational thread (2) and synchronize the completion of threads (5) before scheduling the next thread can be combined into one scheduling-synchronization overhead time. Unless specialized scheduling hardware is present, this time grows as $\log(p)$. The scheduling-synchronization overhead (expressed in time or in operations) determines the half-peak power scheduling-synchronization granularity, assuming that no inter-processor communication is required.

Three inter-communication parameters that affect performance are inherent in distributed computers with inter-node communication: bandwidth (the rate a message can be transferred from application to application), latency (time between when an app sends a message and when it is received by an app in another node), and overhead (processor lost time for sending and receiving the message). Latency is a function of the network delay and message-length/bandwidth. Assuming a computer has adequate bandwidth and that latency can be hidden by doing other work, processor overhead is the most important parameter and a function of the app. For a given app, the communication network latency (that cannot be covered within the grain) plus overhead (expressed in time or in operations) determines the half peak power communication granularity, assuming no scheduling and synchronization overhead.

Table V. Degrees of Application Granularity for a Microprocessor of 100 Mflops/50 Mips PAP

Granularity	Structure	mesg. latency/ovhd(μs.)	inst'ns/mesg*	flops/mesg*
WAN	network	100,000/2000	5M/50,00	10DM/100,000
very coarse	LAN	1000	50,000	100,000
coarse	smCs	100	5,000	10.000
medium	tuned smC	10	500	1,000
fine grain	smP, fine grain.smC	0.2–2	10–100	20–200

Table V posits degrees of granularity, assuming communication is the granularity limiter. The basic assumption is that a processor is tied up for a message and loses the opportunity to do useful work. For example, for a thread efficiency of 50%, the amount of computation in a thread is equal to the message latency and processor overhead; for efficiencies of 90%, the computation per thread must be increased by a factor of 9.

5. FUTURE SCALABLE, DISTRIBUTED GENERAL PURPOSE COMPUTERS

The physical structure of scalable, distributed multicomputers and multiprocessors have nearly converged. Furthermore, with increased hardware support for a single address space and software, including parallel libraries, they will continue to converge to the multiprocessor. As in the CISC versus RISC debate, the key design decisions will be around placing functions in hardware versus software. Functions can be provided either in software in a flexible and open-ended fashion, requiring additional time, or rapidly in hardware in an inflexible fashion. It is still easy to detect whether a computer is an mP, simply by asking whether it is general: i.e., can jobs within an arbitrary workload efficiently use any combination of a computer resources?

5.1. The Computing Nodes

This section will look at the fundamental capabilities of scalable, general purpose computers.

5.1.1. Naming Items Within a Memory

The number one computer design flaw is a lack of address bits to directly address the entire physical or virtual memory of a computer in order to access every item with a unique name. Hardware must provide this fundamental ability to access data without having to create addresses that are a concatenation of node number and address within a node. By not having system-wide names for memory locations, more instructions are required for system wide addressing. While modern microprocessors use 64 bit addresses, experimental multicomputers use 32-bit microprocessors with 32-bit registers, data-paths, and addresses, requiring further changes in the next generation. The biggest gain in next generation multicomputers will come about by having a single address space that does not require software operations and bookkeeping.

5.1.2. *Accessing* Distributed *Data: Load-Store, Message-Passing, and Latency*

Traditional programming is based on a sequential control stream and load-store data access; that is, programs "pull" data to a computation, and then store the result. Message passing requires a "push" or store/load model; that is, a computation occurs and data is broadcast to a node(s) that may ultimately need the data. This two-phase model (computation and communication) has been observed to degrade because of congestion in the communication phase (Zhou *et al.*[30]); communication needs to occur continuously within a thread.

All distributed computers use a message[9] to access remote nodes whether it be for messages or individually addressed data items. For a remote store (or message send), a message consisting of a node and local address, and data item are composed, then sent to the remote node using the switching network. The remote node receives the message, looks at the address and stores the data. Three delays are present: composing the address and data (the same delay as a local memory store operation a multiprocessor); the data transmission delay consisting of network latency plus message transmission (message length/bandwidth), followed by the local store (identical to a store in an mP). Providing direct access to other nodes using addresses will provide the greatest improvement to multicomputers. Caltech's Mosaic C addressed this fundamental problem by a multi-threaded processor for computation, local memory and co-processor access, and communication to other nodes.

Message passing is best carried out by not moving data at all. Instead, data is passed by passing a pointer such that the message consumer is free to take all or any of the data. Passing messages in this fashion reduces or eliminates the overhead that is typical of a message passing library that moves data several times (pack, send, receive, unpack).

The multicomputers overhead occurs because all data transfers require explicit programs, in multiprocessors only the network latency can cause delay. The latency can be dealt with in three ways: reducing access time by increasing spatial and temporal locality by caching; the processor can be multi-threaded or use supplementary hardware so that a process is run while sending messages to a remote node; and having pre-fetch and post-store instructions that do not wait on completion. For example the KSR 1 can issue up to 3 pre-fetch instructions without waiting for completion and the processor is only stalled when the datum fetched is unavailable and needed for an operation.

[9] Multicomputers pass explicit messages, and multiprocessors send hardware controlled messages that are implicit in the operation, e.g. load or store, being requested.

By far, the most important latency reducing technique is to reduce latency by eliminating it through a very large cache, provided that other overhead is not incurred by such an increase. The all cache architecture appears to significantly increase the likelihood of having the required data because every data item is moved to the memory of the processing node that requires it.

5.1.3. The Distributed or Shared Virtual Memory (DSM)

The first task of a DSM is to create a single address space for the entire set of distributed computers. This can be as simple as using a computer's large address as the global address. Apollo's Domain (c1982) provided a large, 64-bit system-wide, address even though 32-bit micros were used. Distributed file systems such as NFS operate by creating a single address space for networked workstations. Li and Shafer[31] have implemented a shared virtual memory, or DSM, consisting of a software layer to convert a set of independent address spaces on a distributed collection of computers (i.e., a multicompter) into a single, shared memory space with the ability to replicate pages in various memories. DSM (Nitzberg and Lo[2]) provides for page replication e.g., pages marked as read only that can reside in several hosts, and multiple writers—while a page is being written it can only reside in a single host.

Fundamentally, DSM is a limited, software version of KSRs ALLCACHE™ architecture, with two critical differences; software carries out the movement and read/write control of the pages; and instead of controlling data at a page level, the KSR ALLCACHE™ architecture moves small data elements such as few cache lines (16, 64-bit words).

Researchers found that software provided DSM on LAN workstations and multicomputers is competitive or superior to message passing for most apps, since data is moved among nodes in large blocks for some algorithms and in small, on-demand basis for other cases. Zhou *et al.*[30] have created software versions of this "all cache" architecture for a heterogeneous multicomputer based on Li's work.

5.1.4. Memory Coherence

The degree of a system's ability to maintain a single, unified view of a memory for a number of independent observers (processors) that are sequentially reading and writing a distributed memory that holds multiple copies of the same address, is the domain of memory coherence. Most all programs are based on "strict coherence" where a load instruction returns the most recent value of memory that a processor stores. As additional pro-

cessors are added to a system with a single, centralized memory, "sequential consistency" is what programmers might expect; the state of the system is determined by some interleaved execution of programs on each of the processing nodes. As memory and processor nodes are distributed, maintaining a single, coherent memory becomes difficult (i.e., time- and network-consuming) any time multiple nodes share (read or write) multiple copies of data in any of the virtual memory. In order to reduce the need for "sequential consistency," multiprocessor designers have introduced weaker forms of consistency, beginning with "multis." "Release consistency" using processor instructions to control when data can be viewed by other processors has been posited as sufficient for parallel programming. Zucker and Baer[32] described various degrees of memory coherence ranging from sequential to release consistency and show the performance impacts on Stanford's Parallel benchmark suite. The degree of coherence that programs require depends on the granularity and inter-thread communication requirements. Providing weaker forms of consistency may trade-off ease of porting large apps and reliability for a slight increase in performance.

5.1.5. Microprocessor Architecture and Technology: CMOS, Multi-Threading, Wide-Word

On the basis of CMOS microprocessor learning curves, the main line of scalable computers are most likely to be CMOS with only Fujitsu deviating from this main line. Main line microprocessors will evolve to carry out more operations per clock tick. In 1993, the best microprocessors carry out two arithmetic operations per clock, and this may increase to >4 by the late 1990s. The alternative is for microprocessors to embrace the vector architecture that enables Meiko's CS-2. The advantage of a wider word versus vector approach is more flexibility at the expense of finer grain apps that exploit vectorization.

5.1.6. Evolvability

The architecture of future scalable computer nodes is based on the main line of microprocessor development used in workstations, e.g., DEC, HP, IBM, and SUN, and it is unclear that this is a sound strategy for evolvability. Convex (HP PA-RISC), Cray (DECs Alpha, but is not guaranteeing next generation compatibility), Intel Paragon (the last of the i860 product line), and Meiko (adds a Fujitsu vector processor chip) use unmodified, compatible off-the-shelf microprocessors. Convex and Meiko use manufacturer software.

On the basis of Caltech work and looking at the requirements for effective, scalable computers, it is unclear that microprocessors designed for workstations make the best scalable computers. For example, multicomputers need multi-threaded microprocessors to over the latency inherent in distributed memory computers, and to carry out overhead functions that convert a multicomputer into a multiprocessor such as the Mosaic C (computing, communication, and memory management and access). Workstations do not clearly benefit from being multi-threaded. Caltech's Mosaic C is a good indication that specialized microprocessors designed for multicomputers significantly out-perform off-the-self microprocessors used in workstations. KSR provides functions in hardware for managing the memory environment, for example.

Another requirement for evolvability is that the network must be improved to have reduced overhead and latency in proportion to the processor speedup. This could be a severe limitation in future scalable computers. Without generation scalability, apps will not be transportable from generation to generation, further impeding the adoption of scalable computers.

5.1.7. The Network

The interconnection network's most important dimension is distributability over a range of distances. This will come through understanding granularity to deal with longer latencies as described in Table I. In this way, geographically distributed, scalable computers are by-products of existing computers.

5.1.8. Controlling and Assisting Parallelism: Scheduling-Synchronization Functions

Multiprocessors use a central memory to control the scheduling and synchronization of work. Multicomputers either simulate mPs or statically assign work or use a special network such as the CM5 provides to synchronize processor completion and carry out reduction operations that require results from each computer. As mCs evolve to directly access one another's memory, these functions are likely to become more like the mP. Similarly, in order to reduce synchronization time, an mP could benefit from special hardware, such as global interrupts, or barriers that would suspend work until all processors have finished a thread, accurate timers, etc. Cray's T3D provides hardware for these functions.

6. Summary

Scalable, distributed, shared memory multiprocessors based on rapidly evolving CMOS microprocessors are likely to emerge as the main line of single system structures. For example, limited scalability supercomputers will be supplemented by scalable mPs in 1994, thus scalable computers have clearly not replaced the need for supers.

In 1993, switches and other overhead limit scalable multicomputers (i.e., they have low efficiency for real app performance). Even though PAP/$ is up to eight times higher for multicomputers than supercomputers, RAP/$ is about constant for the two for a real workload. Multicomputers have not demonstrated an ability to handle a general purpose workload. By 1998, i.e., two, three-year generations, all multicomputers developed as part of the HPCC program that use 32-bit microprocessors and distributed scalable memory (DSM) software for addressing and virtual memory management will converge to have the capabilities of a scalable mP with a single 64-bit, addressable, coherent memory. Switches pose the greatest risk to generation scalability where every component of a system must improve at the same rate. Switches must improve in latency, bandwidth, and overhead at 60% per year to track microprocessor evolution and insure generation to generation portability of apps.

LAN-based workstations, i.e. multicomputers, will evolve to be interconnected by fast, switches such as ATM, operating at 100 Mbytes per second and have the capability of 1992 multicomputers. Thus, parallel processing can exist as a by-product of a normal, highly distributed workstation environment without the need for specialized multicomputers. The HPCC must focus R&D on this approach to leverage these tremendous, existing resources.

The future of parallelism using scalable computers will continue to be slow and steady, limited by the fundamental understanding of computer, application characteristics, and standard programming environments.

REFERENCES

1. G. Bell, Ultracomputers: A Teraflop Before Its Time, *Comm. of the ACM* **35**(8):27–45 (August 1992).
2. B. Nitzberg and V. Lo, Distributed Shared Memory: A Survey of Issues and Algorithms, *Computer*, pp. 52–60 (August 1991).
3. M. D. Hill, What is Scalability? *Computer Architecture News* **18**(4):18–21 (December 1990).
4. D. Nussbaum and A. Agarwal, Scalability of Parallel Machines, *Comm. of the ACM* **34**(3):57–61 (March 1991).
5. S. L. Scott, A Cache Coherence Mechanism for Scalable, Shared-Memory Multi-

processors, *Proc. Int'l. Symp. of Shared Memory Multiprocessing*, Information Processing Society of Japan, Tokyo, April, pp. 49–59 (1991).
6. J. L. Gustafson, G. R. Montry, and R. E. Benner, Development of Parallel Methods for a 1024 Processor Hypercube, *SIAM J. Sci. Stat. Comput.* **9**(4):609–638 (July 1988).
7. V. Kumar and A. Gupta, Analyzing Scalabilty of Parallel Algorithms and Architectures, TR 91-18, Department of Computer Science, University of Minnesota (January 1992).
8. X. Sun and D. T. Rover, Scalability of Parallel Algorithm-Machine Combinations, Technical Report of the Ames Laboratory, Iowa State, IS 5057, UC 32 (April 1991).
9. A. H. Karp, Programming for Parallelism, *Computer*, pp. 43–57 (May 1987).
10. J. Worlton, MPP: All Things Considered, is it More Cost-Effective?, Worlton and Associates Technical Report No. 42, Salt Lake City, Utah (May 1992).
11. D. H. Bailey, E. Barscz, L. Dagun, and H. D. Simon, NAS Parallel Benchmark Result, RNR Technical Report RNR-92-002, NASA Ame Research Center (December 1992).
12. D. H. Bailey, Twelve Ways to Fool the Masses When Giving Performance Result on Parallel Computers, *Supercomputing Review*, pp. 54–55 (August 1991).
13. J. Worlton, Be Sure The MPP Bandwagon is going Somewhere Before You Jump on Board, *High Performance Computing Review*, p. 41 (Winter 1992).
14. A. H. Karp, K. Miura, and H. Simon, 1992 Gordon Bell Prize Winners, *Computer* **26**(1):77–82 (January 1993).
15. G. Bell, The Future of High Performance Computers in Science and Engineering, *Comm. of the ACM* **32**(9):1091–1101 (September 1989).
16. M. Lin, R. Tsang, D. H. C. Du, A. E. Kleitz, and S. Saraoff, Performance Evaluation of the CM5 Interconnect Network, *IEEE CompCon* (Spring 1993).
17. G. Bell, Three Decades of Multiprocessors, Richard Rashid (ed.), *CMU Computer Science: 25th Anniversary Commemorative*, ACM Press, Addison-Wesley Publishing, Reading, Massachusetts, pp. 3–27 (1991).
18. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Mateo, California (1990).
19. J. L. Hennessy, *Scalable Multiprocessors and the DASH Approach*, University Video Communication, Stanford, California (1992).
20. D. Lenoski, K. Gharacharloo, J. Laudon, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam, Design of Scalable Shared-Memory Multiprocessors: The DASH Approach, *ACM COMPCON* (February 1990).
21. J. P. Singh, W. D. Weber, and A. Gupta, SPLASH: Stanford Parallel Applications for Shared-Memory, *Computer Architecture News* **20**(1);5–44 (March 1992).
22. P. J. Denning, Working Sets Past and Present, *IEEE Transactions on Software Engineering* **SE-6**(1):64–84 (January 1980).
23. A. Gupta, T. Joe, and Per Stenstrom, *Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures*, Computer Systems Laboratory, Stanford California (1993).
24. E. Hagersten, Toward Scalable Cache Only Memory Architectures, Ph.D. Dissertation, The Royal Institute of Technology, Stockholm Sweden (October 1992).
25. S. Frank, H. Burkhardt, L. Lee, N. Goodman, B. I. Marguilies, and D. D. Weber, Multiprocessor Digital Data Processing System, U.S. Patent No. 5,055,999 (December 27, 1987).
26. KSR-1 Technical Summary, Kendall Square Research, Waltham, Massachusetts (1992).
27. C. Sietz, Mosaic C: AQn Experimental Fine-Grain Multicomputer, *25th Anniversary of the Founding of INRIA*, Springer-Verlag (to be published).
28. M. Berry, G. Cybenko, and J. Larson, Scientific Benchmark Characterizations, *Parallel Computing* **17**:1173–1194 (1991).

29. R. W. Hockney and C. R. Jesshope, *Parallel Computers 2*, Adam Hilger, Bristol (1988).
30. S. Zhou, J. Strumm, L. Li, and D. Wortman, Heterogeneous Distributed Shared Memory, IEEE Transactions on Parallel and Distributed Systems 3(5):540–554 (September 1992).
31. K. Li and R. Schafer, A Hypercube Shared Virtual Memory System, *Int'l. Conf. on Parallel System* (1989).
32. R. N. Zucker and J. L. Baer, A Performance Study of Memory Consistency Models, *Proc. of the 19th Annual Int'l. Symp. on Computer Architecture*, pp. 2–12 (1992).