OCCASION

This publication has been made available to the public on the occasion of the 50th anniversary of the United Nations Industrial Development Organisation.

DISCLAIMER

FAIR USE POLICY

CONTACT

Please contact publications@unido.org for further information concerning UNIDO publications.

For more information about UNIDO, please visit us at www.unido.org

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION
Vienna International Centre, P.O. Box 300, 1400 Vienna, Austria
Tel: (+43-1) 26026-0 · www.unido.org · unido@unido.org

21436

Chapter I. Parallel Processing - An Introduction

Boleslaw K. Szymanski
Department of Computer Science & Scientific Computation
Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
e-mail: szymab@rpi.edu

1. Introduction

This document presents the current state-of-the art in parallel processing. In this chapter, we start with the overall introduction to the problems and challenges of parallel computing. More detailed and regional perspectives are described in the following chapters.

2. The role of computers

It is widely recognized that the computer technology has become a critical component of everyday life of a modern society. The computer has become ubiquitous in manufacturing, services, products, entertainment. Computers have been changing ways in which we conduct business, produce goods and do science. Export controls introduced on certain computer equipment and secrecy surrounding some computer projects underline importance of computer technology in military and security services. There have been three stages of exponential growth in the power of computers and related systems. First, in 1960's and 1970's computing centers were being constantly upgraded with ever-more powerful machines. This was followed by an exponential growth in the number of computers during so-called personal computer (PC) revolution in 1980's. Currently we are undergoing yet another stage of the same process: an exponential growth in interconnectivity and bandwidth of the network joining computers and systems together.

Computer literacy is becoming a norm, not the exception, and many knowledgeable workers, managers and other professionals now have the technical ability and skills to write software.

According to [1] there are nearly two million people in the US that work directly with software and about ten million managers, engineers, architects, accountants and other knowledge workers who know enough about programming to be able to build end-user applications with high-level tools (spreadsheets, databases, visual languages). Similar ratios are found in other industrialized countries around the world.

Country	Software Professional	End Users with Programming Skills	Percentage of workforce
United States	1,750,000	10,000,000	9%
Japan	850,000	3,500,000	6%
United Kingdom	385,000	1,750,000	6%
France	375,000	1,700,000	7%
Germany	350,000	1,650,000	4%
Brazil	475,000	1,500,000	3%
China	950,000	1,250,000	<1%
India	750,000	1,200,000	<1%
Russia	750,000	900,000	1%
South Korea	300,000	750,000	5%

Table 1. Professional software personnel and end-users who program, 1995 estimates (source [1]).

On the global basis, there are more than ten million professional software personnel and more than 30 million end-users who can program. Table 1 gives rough estimates for each group in ten countries around the world (data are with high margin of error). An interesting observation is that Brazil and South Korea have as high percentage of the workforce able to write software as the most developed countries of the world.

The end-user programming population seems to be growing at more than ten percent per year worldwide. The percentage growth rate for software professionals is now down to a single

digit in industrialized countries. In developing countries, the number of both end-user and professional programmers is still growing at double-digit percentage rates. In view of this developments, it is not an exaggeration to compare the impact of computers on society to the Industrial Revolution of the 18th Century.

The Industrial Revolution had freed physical workers from the enslavement of manual labor and had transformed craft and handwork into the mass-producing industries of today. Likewise, the Computer Revolution, which we are witnessing now has been freeing the office workers from routine mental tasks which were and often are still done by assistants, clerks and low-level managers.

3. Significance of Parallel Processing

Parallel processing is currently a small fraction of overall computer technology and the Computer Revolution. Yet, there are two compelling reasons for parallel processing to be of much higher importance than indicated by its current share of computer technology. The first reason is that parallel processing supports the largest computations which became an integral part of the science, medicine and manufacturing. Large-scale computer modeling enabled by parallel processing impacts decision making in banking, finance, military and government. Parallel computers empower decision makers, such as high-level managers, military leaders and chief scientists with the ability to gather, access, and synthesize information, as well as to simulate real-life processes to measure the impact of social, economical and industrial decisions. The quality of the simulations and synthesized information strongly depends on the applied computational power. Today, even the largest uniprocessor computers are too slow for the more challenging problems of this kind.

The second reason for importance of parallel processing is the transient nature of the current dominance of sequential computing. There are clear indications, discussed below, that the semiconductor industry ability to double the processor speed every 18 months, as it has been done for the last decade

cannot continue and that the processor design technology is maturing. An interesting comparison of this process to historical technological breakthroughs are presented in [2]. The authors contend that virtually every industry more than a few decades old underwent similar phase changes which are caused by economic principles of supply and demand. The past transitions of older industries such as aviation, automobiles and railroads could be used as indicators of what to expect of the semiconductor industry.

For example, aviation, like the semiconductor industry, went through a period of rapid growth. In less than four decades the industry moved from the Wright brother's monoplane to the Pan Am Clipper jet and Super-fortress bomber. The growth of aviation was initially fueled by military markets, then moved on to civil transportation, much like the growth pattern in the semiconductor industry. Progress in aviation was made by increasing speed of airplanes (thus reducing transit time) and by lowering the costs of transportation. Such dual progress is similar to the computer processor's increasing speed (which reduces the time to obtain answers) while lowering the processor price. After several decades of growth in passengers capacity and airspeed, these trends peaked with Boeing's 747 as the highest mark for capacity and the Concord as one for speed. Further progress was stopped by the economic constraints (at least in civil aviation, for military application, where cost is not a primary consideration, fighters produced by many manufactures exceed the speed of a Concord). The limiting factor for Boeing's 747 was the difficulty of filling the available space on all but the longest or most popular routes. The usefulness of Concord was limited by the cost of fuel and noise pollution. After these technological marvels, aviation entered a second phase in which a plethora of smaller/slower airplanes were designed and produced for more specific markets. The focus of research and development shifted from aircraft's speed and size to operational efficiency and passenger's comfort.

Another example given in [2] is railroads. The research there focused initially on increasing the power of locomotives to lower the cost of transportation. This trend peaked at EMD

DD-40, a monster locomotive that was too big and inflexible for any other purpose than hauling freight across the U.S. These limitations resulted in increased use of smaller engines that could operate separately for small loads but could also be joined together for transporting big loads. The authors conclude that today the semiconductor industry is in a similar position as railroad companies were just before the EMD DD-40 was designed. The high cost of developing factories for future generation processor chips forced the semiconductor companies to join forces to attempt to manufacture extremely dense chips economically.

The final example used in [2] is the automobile industry. Ford's initial success in car production resulted from lowering costs by concentrating production in ever-larger factories. These trend led to diminishing ability to vary the product. In early 1930's, General Motors recognized that big factories were good only for building large numbers of the same product and that the efficiency no longer increases with the factory size once some critical size is achieved. Therefore, General Motors split the company into divisions with clearly defined markets and factories dedicated to support them. The resulting wider variation in designs allowed General Motors to gain market share at Ford's expense.

A similar scenario is happening today in semiconductor industry. Intel offers more than 30 variations of its 486 microprocessor, while in the early 80's the company offered just three versions of its 8086 microprocessor. The authors conclude that in technology driven industries the initial phase is dominated by improvements both in performance and costs. The second, mature phase, is characterized by product refinement and diversity - similar to what is now starting to happen in the semiconductor industry. The slowing rate of progress in processors will provide a more stable environment for computer architecture and software. As a result, parallel processing will become much more widely spread than it is today.

4. Applications of Parallel Processing

In the United States, the quest for higher-speed machines has been fueled by computationally intensive problems with profound economical and social impacts referred to as Grand Challenges [3]. It is difficult to list all Grand Challenge problems because so many areas of science and engineering are potential sources of such problems. The short list typically includes:

- High-resolution weather forecasting crucial for agriculture, disaster prevention, etc.
- Pollution studies that include cross-pollutant interactions, important in environmental protection.
- Global modeling of atmosphere-ocean-biosphere interactions to measure the long-term impact of human activities on the stability of the global ecosystem.
- Human genome sequencing that will assist in recognizing, preventing and fighting genetic diseases.
- The design of new and more efficient drugs to cure cancer, AIDS and other diseases.
- High-temperature superconductor design that can revolutionize computer design, electrical devices, etc.
- The aerodynamic design of aerospace vehicles (airflow modeling) and improvements in automotive engine design (ignition and combustion modeling) that can lead to more efficient use of depletable fossil fuels in transportation.
- The design of quantum switching devices important for building more powerful computers.

U.S. research support agencies, such as the National Science Foundation, various agencies in the Departments of Defense and Energy as well as the National Aeronautical and Space Agency together fund projects targeting Grand Challenge problems. This is a five year effort referred to as High Performance Computation and Communication Program or HPCC in short. It started in 1993 and has a yearly budget of several hundred million dollars. The investigations conducted under HPCC Program involves multidisciplinary teams of researchers in natural sciences, applied mathematics, computational and

computer science from different institutions.

As an example, the author of this chapter is involved in four projects involving Grand Challenges problems investigated at Rensselaer Polytechnic Institute. Two of these projects are funded directly from HPCC Program and two others indirectly. One of the projects focuses on modeling human joints, in particular shoulders and knees. The research is conducted in cooperation with Orthopedics Division of Columbia University Medical Center. The goal is to be able to guide the surgeons operating on joints by simulating the behavior of a joint under different operation scenarios. Adaptive meshes and finite element methods are used to solve the partial differential equations that describe joints' behavior.

Another project focuses on problem solving environments for optimization and control of chemical and biological processes. This investigation is conducted in cooperation with groups at University of Minnesota and University of California in San Diego. The primary goal of this research is the development of a high-performance problem solving environment (PSE) for the optimization and control of chemical and biological processes, with initial emphasis on bioengineering applications. The optimization and control of such processes requires the repetitive solution of time-dependent partial differential equations (PDEs) in two or three spatial dimensions. The computational requirements of this problem, which must be solved interactively, can only be met by the use of massively parallel computers. Such a comprehensive and powerful PSE does not currently exist, and its development presents significant computational and computer science challenges.

The third project is a part of a tokamak design with an ultimate goal of building a sustainable plasma generation device fueled by hot fusion. The purpose of our investigation is to develop a scalable and portable Plasma in Cell (PIC) for simulation of plasma behavior in a self-generated electromagnetic field. This work is being done in cooperation with researchers at University of California in San Diego and Jet Propulsion Laboratory. Finally, the fourth project focuses on individual based modeling of epidemics. In cooperation with

biologists from State University of New York in Albany we are investigating the spread of Lyme disease and the ways of controlling this spread.

Our basic computational tool is a 36-node IBM SP2 parallel computer with peak performance of about 9 gigaflops (i.e., 9 billion floating point operations per second) which we have available on campus. This machines is used mainly for code development and test runs. The production runs are conducted on 400-node SP2 at the Maui (Hawaii) Supercomputing Center and 512-node SP2 at Cornell Supercomputing Center located at Cornell University. Both of these machines have a peak performance in the order of a hundred of gigaflops. The research on plasma simulation involves additional machines, Cray T3D and Intel Paragon at Jet Propulsion Laboratory as well as a network of Sun workstation at Rensselaer Polytechnic Institute. These example of research involvement of a single scientist perhaps best describes how diversified the HPCC Program is and how much cooperation it has fostered.

5. Required Computational Power

It is estimated that to achieve interactive response time for Grand Challenge problems, in the order of minutes for smaller instances and hours for larger ones, will require a machine with performance of teraflops (which is a thousand billion floating point operations per second). Today, there are several parallel machines with a theoretical peak of teraflops and the price below US$ 100 million (e.g., AVALON computer based on DEC Alpha chip and a fast interconnection network). However, the sustained performance has been demonstrated at the level of tenths of teraflops, i.e., about several hundred gigaflops. Even in those cases, such speed was achieved only on certain very large, highly localized, finely tuned, often idealized applications. The real drawback is in the software and the programmer's ability to find enough useful parallelism in an application to utilize all of the processors of a parallel computer most of the time. Yet, parallel processing is the only feasible option for sustained growth in computer performance in view of the incoming stalemate in the semiconductor industry discussed earlier.

In addition to economical forces (exponentially increasing cost of hardware needed to fabricate chips with smaller dimensions) there are basic laws of physics that put limit on the speed of a uniprocessor. The speed of signal transmission in a computer cannot exceed the speed of light in the transmission media, which is about 300,000 km/sec for silicon. Consequently, it takes one billionth of a second for a signal to propagate in a silicon chip of an inch in diameter. However, one signal propagation can at most support one floating point operation. Hence, a sequential computer built with a chip of such size can provide at most one gigaflops of performance, merely one-thousandth of the needed teraflops.

6. Parallel Architectures

An interest in parallel computing systems is not new and can be traced back as far as the 1920's. However, as late as the early 1970's, major criticism of parallel processing was based on Grosch's law which states that the computing power of a single processor increases in proportion to the square of its cost. Recent careful analysis of Grosch's law showed that it is valid only within one technology. Economy of scale for mass-produced memory and RISC (Reduced Instruction Set Chip) processors makes them orders of magnitude less expensive than custom designed chips for mainframes and traditional vector supercomputers. The improving computer chip technology enables the placement of ever-faster processors with ever-increasing amounts of memory on a single wafer. Hence, introduction of RISC technology made Grosch's law obsolete. Massively parallel computers built from a large number of RISC processors provide a superior performance-to-price ratio compared to computers based on the powerful, custom-designed CISC (Complex Instruction Set Chip) processors.

The traditional vector supercomputers are built from a limited number of powerful, specially designed processors connected to large shared memory. In addition, they explore array operation parallelism through vector co-processors. However, the support for shared memory limits the number of processors that can be clustered together in such a way that all have the same

access time to the whole memory. Hence, purely shared memory machines are not scalable. In contrast, massively parallel computers consist of a large number of off-the-shelf processors with local memories. The processors are connected directly to each other by a network. The cost of such a parallel computer is roughly proportional to the needed number of processors. Therefore the size of the computer installation is more limited by costs than technical considerations. In addition, the process of technological progress for off-the-shelf processors is driven by the general computing market which is two orders of magnitude larger than the parallel computing market. As a result, off-the-shelf processors enjoy larger increases in speed and reductions in prices than the custom designed processors do. Hence, the Massively Parallel Processors (MPP's) have three advantages over traditional vector supercomputers:

1. An accelerated rate of advance of peak processing power. In the last decade, microprocessor performance has increased four times every three years, following the rate of integrated circuit logic density improvement. By contrast, the clock rates of vector machines have improved much more slowly, doubling every seven years [3]. These trends are expected to continue for at least the rest of 1990's.

2. An improvement in the performance-to-cost ratio. This ratio was between two to eight times higher for MPP's than for the vector supercomputers in 1993.

3. Scalability of the machine. The smallest configurations of MPP's are usually priced low to entice initial purchase (in 1995, the least expensive MPP's was priced below $50,000). The initial configuration of the MPP can be incrementally upgraded as the needs and available funds arise.

The clear conclusion is that only massively parallel computers can deliver the much needed teraflops level of performance.

7. Parallel programming

Parallel programming has experienced a long and difficult

maturation process. The reasons are many, but according to [2] the most critical one has been the difficulty in programming the constantly developing new architectures. Porting and tuning an application to a new architecture often takes as long as the time between introduction of the subsequent architecture, making a newly developed code obsolete at the moment of it is fully implemented. In such environments, programmers face a daunting challenge, especially with increasingly large and complex applications. Programmers must identify parallelism in an application, translate that parallelism into code and design the corresponding communication and synchronization for the program. All these steps must be done in the context of currently available architectures which may change tomorrow, making some of the designs suboptimal or inefficient.

One of the promising approaches to curb the cost of the parallel software redevelopment resulting from new architecture introduction is object-oriented programming. However, according to Grimshow [4], the object-oriented parallel programming community is divided over the issue how to support parallelism in an application. There are two basic camps. The first one, the libraries group, advocates building highly optimized, extensible class libraries that will encapsulate parallelism leaving the language uncharged. Users would be able to use such class libraries without knowing anything about parallelism, the target architecture or implementation details of the class library. The proponents of the library approach argue that C++ already provides a powerful mechanism for language extension via classes, inheritance, and templates. Additional extensions would only clutter the language. Furthermore, with no consensus on language features, compiler vendors are unlikely to support any language extensions, and users will not want to risk embracing the feature which will no make it to the future standard.

The second camp, the extension group, believes that the best way to achieve parallelism is via language extensions. The proponents of this approach argue that parallel composition is as important and fundamental a concept as sequential

composition. They point out to languages such as OCCAM and ADA in which explicit parallelism is a part of the language and therefore their compilers have been able to develop parallel code optimizations. With concurrency being a part of C++, the same process would happen for C++ compilers.

Grimshow [4] conjectures that the parallel processing is at a crossroad. In the past, parallel processing was relevant only to expensive supercomputers and software was often developed in-house because such a narrow market was not financially viable for the commercial software developers. Instead, the successful commercial softwarehouses concentrated on booming personal computers and workstations market. Today, however, the cost-effective desktop computers are closing the performance gap to supercomputing. The desktop software is moving increasingly towards object-base and object-oriented interoperability standards. At the same time, many traditional parallel processing users are downsizing and no longer have the resources to develop software in-house. Therefore, the parallel processing community has an incentive and opportunity to adapt and conform to emerging standards and link to desktop software market.

Interoperability standards are also important for integration of parallel components developed by different research groups. Increasingly important multidisciplinary simulations require coupling of different models to create a more realistic model of a phenomena. For example, tokamak simulation requires linking of a model of plasma particles motion in an electromagnetic field with a model of chemical reactions that these particles undergo. The individual components of such simulations are often stand-alone parallel codes. While the system file can be used as an interface and data exchange mechanism, more efficient integration methods are needed. Object technology simplifies description of an interface mechanism, a data exchange and conversion mechanism. As a result object technology can extend the life of the parallel components.

Grimshow [4] concludes that parallel processing components can conform to standard interface description by encapsulating

parallelism within objects and making the parallel component a particularly fast version of an existing sequential code.

From that perspective, the decision of the High Performance Fortran (HPF) designers to based the language on Fortran90 was very helpful. Fortran90 includes all the basic constructs required of object oriented programming and there is an increasing interest in object-oriented programming using Fortran90.

8. Summary of the following chapters

The discussed above trends have focused on the global perspective of parallel programming. The rest of this document provides more detailed and regional points of view on these issues.

8.1 USA Perspectives

First, in a chapter entitled "Trends in Software Engineering for Parallel Processing" the author assesses the current state-of-the-art in this area from the United States perspective. The U.S. amounts for about 50% of the parallel computing power installed in the world, has national research and development programs in the area of high performance computers and the highest number of personal computers and computer users. Therefore the United States provides an important perspective for other countries and an early indication of future developments.

The chapter starts with a discussion of trends in architectural design including the deep transformation which occurred in the U.S. High Performance Computing industry. The emphasis of this industry shifted from record-breaking performance for any price to price-performance optimization. The successful companies, such as Silicon Graphics or IBM, use the performance gains driven by the general market to improve the performance of their parallel machines. In contrast, companies that relied on processors designed specifically for their architectures, like Kendall Square Research or Thinking Machine Corporation, were forced out of the computer design

market.

Another trend discussed in the following chapter is an increasing importance of the memory for speed and economy of parallel processors. The increasing speed of the processors is matched by the increasing capacity but not speed of the memory. As a result, the gap between the speed of processors and memory widens every year. The gap is masked by the use of ever-increasing cache. As evidenced by the pricing and performance of Silicon Graphics Challenger, the ratio of cache to memory improves price-performance of parallel machines on memory intensive applications.

The chapter also discusses trends in speed, price-performance and distribution of parallel processors. The U.S. annual Gordon Bell Awards indicate steady exponential growth in speed and price-performance of parallel computers for the last decade. The parallel computing speed on useful applications reached several hundred gigaflops last year. The price-performance ratio is in the order of ten gigaflops per million dollar of hardware cost. The largest world-wide supercomputing sites are still dominated by the governmental centers. However, the medium sites are mainly industrial and the smallest ones are mainly academic. This distribution contrasts with the overwhelmingly governmental centers in all categories just five years ago. Such shifts in distribution indicate that the impact of parallel processing on industries is growing.

Another topic discussed in this chapter is the development of software models for parallel processing. First, the traditional models are discussed, such as Single Instruction Multiple Data (SIMD) and Single Program Multiple Data (SPMD), which is a restricted version of the more general Multiple Instruction Multiple Data (MIMD) model. Then a new Bulk Synchronous Parallelism Model (BSP) is described together with its library. The final pages of the chapter are devoted to trends in languages. In particular, the basic ideas behind High Performance Fortran (HPF) are described and compared to Message Passing Interface (MPI) based approach.

8.2 Asian Perspectives

The third chapter of this document, entitled "High Performance Computing in India and Far-East" and authored by Prof. Patnaik from India focuses on Asian perspective on parallel processing. The author describes India's national initiative in supercomputing and its goal to develop a modern distributed memory parallel computer. The result was PARAM-8000 parallel computer based initially on INMOS transputer (1990-92). The sustained performance of 16-node PARAM 8600 was in the range of 0.1 - 0.2 gigaflops. The next generation of PARAM computers, PARAM-9000, was based on SuperSparc series processor, thus following the modern trend of using off-the-shelf mainstream processors in the parallel machine architecture. After discussing architectural details of these machines, prof. Patnaik describes its software environment which includes support for HPF for data parallelism and PVM and MPI for MIMD parallelism. Then, the author focuses on applications run on PARAM which currently are of typical scientific computation mix. However, plans are made to incorporate the production quality industrial codes in the near future.

In the following pages, prof. Patnaik describes several others Indian parallel computers developed by different governmental research centers. Discussing the performance capabilities of these machines, prof. Patnaik demonstrates that by employing sufficient number of nodes, PARAM-9000 can achieve a peak performance of teraflops. The support for Indian High Performance Computing activities comes from different agencies of the government.

In his chapter, prof. Patnaik summarizes also briefly High Performance Computing activities in other countries from the Far-East Region. His summary discusses developments and programs in Singapore, China, Australia, Korea, New Zealand and Hong Kong. The chapter concludes with a brief assessment of the major trends in parallel computation.

Chapter II. Trends in Software Engineering for Parallel Processing

Boleslaw K. Szymanski
Department of Computer Science & Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
e-mail: szymab@rpi.edu

1. Introduction

The increasing importance of parallel processing caused by its rapid growth encouraged development of standards in parallel programming languages and tools. Yet, there is no evidence of convergence of the supported paradigms to a single model. In this chapter, we review two currently most popular models for parallel program design: data parallelism and message passing. We also discuss the relevant developments in object oriented programming techniques as well as in client-server distributed/parallel processing. The declining share of the parallel processing market held by traditional supercomputers and the waning popularity of SIMD machines, together with the increasing role of clusters of workstations, created the conditions for rapid spread of parallel machines in government and industry. The price of entry into parallel processing decreased significantly making abundant opportunities for new enterprises in software industry in this area. The chapter discusses these developments and describes also changes in corresponding areas of software engineering for high performance distributed computing. Finally, there is a review of the perspectives and impact of the changing parallel computing industry on information processing at international and national levels.

2. Trends in Architectural Design

Recent events, such as the filing for Chapter 11 bankruptcy by the Thinking Machine Corporation in August 1994, the discontinuation of manufacturing and sale of KSR

supercomputers by Kendall Square Research in September 1994, and the disappearance of Cray Computer this year raised the question how secure is the future of parallel computing industry.

It is important to realize that currently the parallel computing constitutes a small fraction of the overall information technology industry. In 1994, the overall U.S. information technology products were worth about $500 billion [5]. This value is a middle point of two estimates. The first estimate was provided by the U.S. Department of Commerce and was based on data from the U.S. Bureau of the Census. It values shipments for the information technology industry at $421 billion for 1993. This number includes computers, storage related devices, terminals, and peripherals, packaged software, computer software manufacturing, data processing, information services, facilities managements and other services and telecommunication equipment and services. However, this number does not include revenue from equipment rentals, fees for after-sale service and mark-ups in the product distribution channel, as well as office equipment.

The Computer and Business Equipment Manufacturers Association (CBEMA) values the worldwide 1993 revenue of the U.S. information technology industry at $602 billion. This number includes sales of office equipment, and CBEMA reports larger revenues for information technology hardware and telecommunications equipment than the reports provided by the Department of Commerce.

In the United States, the total revenues of parallel computer hardware manufacturers were estimated at about $1 billion in 1993. Out of it, about $400 million was received by manufacturers of massively parallel machines. Even with services and software revenues, the parallel computer industry was about 0.5% of the U.S. information technology industry. Such a small percentage of the overall market indicates a narrow user base that can be easily saturated with new products. In addition, parallel computing has been highly dependent on government policies; institutions and government-supported universities traditionally constituted more than 50%

of all users.

The end of the Cold War and the associated shift of governmental spending in the United States drastically changed the market for parallel machines and supercomputers. As a result, companies relying solely on the manufacturing of parallel machines have suffered the most. At the same time, companies for which parallel computing manufacturing is only a part of the product line (e.g., IBM Corp., Silicon Graphics Inc., Intel Supercomputing) have persevered, and others (most notably Hitachi and NEC in Japan) have entered or expanded their presence in the parallel systems market.

Predictions about a lasting impact of the current changes on the parallel computing industry vary widely. Some see in the recent bankruptcy protection requests the beginning of the end of parallel computing based on massive parallelism. Others argue that it is just an end of a beginning. In the first camp is Gordon Bell, the founder of several computer companies and the sponsor of the yearly Gordon Bell Awards for the fastest parallel computer [1]. Mr. Bell believes that the latest threat to the very existence of the industry comes from standard workstations and fast, low-latency networks based on ATM. These networks, according to Bell, like massively parallel machines, offer size scalability (smooth transition from fewer to more processors). However, unlike parallel machines, they also support generational scalability (from previous to future hardware generations) and space scalability (from multiple nodes in a box, to computers in multiple rooms to geographical regions). The most important capability offered by these networks is application compatibility with workstations and multiprocessor servers. This is a capability which massively parallel computers sorely lack. According to Bell, the weaknesses of massively parallel machines stem from the following two factors:

1. parallel architectures are best suited to highly-tuned, course-grained, and/or data parallel problems

2. every new generation of parallel architectures differs from the previous one, forcing the users to redevelop

their applications.

Bell sees the future of parallel processing in networked workstations and shared-memory multiprocessors.

In a rebuttal to Bell's criticism, James Cownie from Meiko [4] cited the reasons why networked workstations are not an answer in many environments. The most important reason is the need for data security and availability. For a large commercial organization, security of data and its accessibility to those who need it are crucial. The solution is a single, central repository. However, the central repository could use the same components as workstations to amortize the cost of their development. The natural solution is to use multiple processors compatible with the workstations. High performance requires a small physical size because the speed of light limits the performance of highly distributed machines (to keep the latency of communication below one microsecond, the distance between computers must be kept below 300 m). Switching technology cannot be based on ATM's in such a repository, because ATM switches are an order of magnitude slower and more costly than proprietary switching technology. According to Cownie, the only alternative is a massively parallel machine.

A similar point is raised by Philip Carnelley and William Cappelli of Ovum Ltd [2]. They underline that effective manipulation of large amounts of data is crucial for companies in maintaining a competitive advantage in the market. The complex applications in manufacturing, commerce, travel, and entertainment require a sophisticated database support that demands enormous computing power. The costs of hardware and application development restricted parallel processing to niche applications such as scientific computing, weather forecasting, etc. Yet, only parallel computing can meet the current challenge of information processing and, in response to those needs, parallel processing has entered the commercial mainstream. Parallel computers built from standard components (e.g., shared-memory, like Sequent, or distributed memory, like IBM SP2) can run powerful parallel relational databases. Such systems can process data extremely quickly, are

reasonably priced, and are impressively scalable. Today, most of the commercial uses focus on data repository. Carnelley and Cappelli predict the future applications of parallel systems will transform operational systems, decision support systems, and multimedia applications, and, in the process, will provide an enormous impetus for the parallel computing industry.

Ken Kennedy from Rice University [8] underlines that part of the difficulty in making parallel computing wide-spread and popular was the lack of standards in parallel programming interfaces. As discussed later in this article, such standards have been developed and are gaining wide-spread acceptance.

The optimistic views on the future of parallel processing are supported by an exponential growth in the usage of parallel supercomputers at the NSF Supercomputing Centers in the U.S. (see Table 1).

Fiscal Year	Active Users	Usage in Normalized CPU Hours
1986	1,358	29,485
1987	3,326	95,752
1988	5,069	121,615
1989	5,975	165,950
1990	7,364	250,628
1991	7,887	361,037
1992	8,758	398,932
1993	7,730	910,088
1994	7,431	2,249,562

Table 1. Supercomputing Usage at NSF Centers in US (Source: National Research Council [5], 1995).

Some analysts see the exponential growth of revenues for

massively parallel computers in the near future. Terry Bennet, director of technical systems research for Infor-Corp in Beaverton, Oregon, was quoted in [12] as saying that the industry is currently in a ''lag'' where traditional vector supercomputers are fading out while other approaches are maturing. Bennet predicts that by 1996 there should be a reasonable upswing in the high-performance computing business and the market will continue to grow over $4.5 billion in 1998. The strong sales of relative newcomers to the market, IBM Corp. with its SP series and Silicon Graphics Inc. with the Challenger computer, agree with Bennet's prediction.

	1992	1995	1998	2001	2004	2007
Feature size (micron)	.5	.35	.25	.18	.12	.1
Gates per chip	300K	800K	2M	5M	10M	20M
Bits per chip in DRAM	16M	64M	256M	1G	4G	16G
Microprocessor chip size in square mm	250	400	600	800	1000	1250
Memory (DRAM) chip size in square mm	132	200	320	500	700	1000
Wafer diameter in mm	200	200	200-400	200-400	200-400	200-400

Table 2. Semiconductor Technology Trends (Source: Semiconductor Industry Association, March 1993).

Steve Wallach of Convex [13] argues that parallel processing has been becoming ubiquitous on all levels of computing technology. In microprocessor design, super-scalar techniques

- executing multiple instructions at the same time - are now a standard. Multiprocessor file servers are in the process of becoming a standard. The continuing increase in the semiconductor density (see Table 2) will naturally lead to multiple processors on one semiconductor die. If a standard 64-bit RISC microprocessor has 1-2 million transistors (without cache), what else (other than creating a multiprocessor chip) can be done with transistors when 100 million and 1 billion transistors become available? Wide-spread use will drive the costs of such a chip down and therefore will make massively parallel computing cost effective.

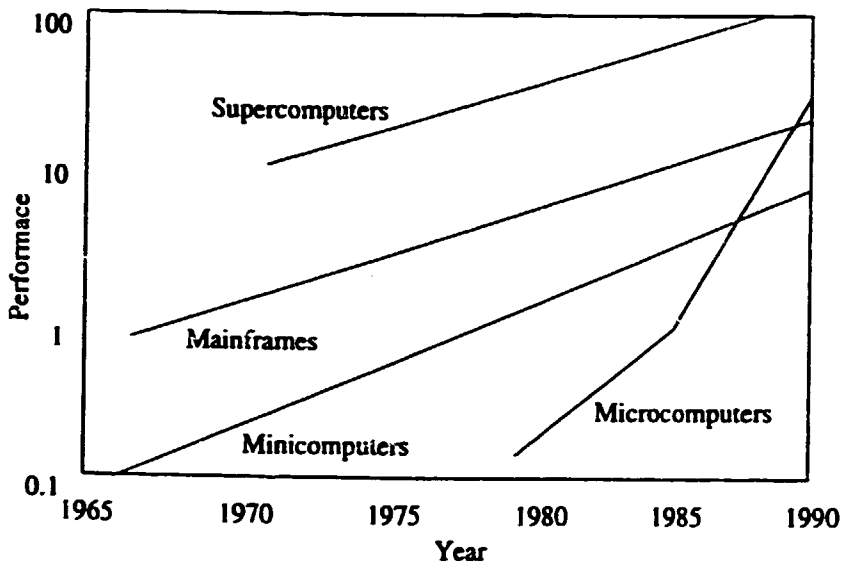

Figure 1. Trends in microprocessors and Mainframe CPU Performance Growth.

The case against supercomputing and massively parallel computers often is based on the difference in speed with which the performance of microprocessors and other CPU's grew (see Figure 1, which was based on [7]). However, the CPU performance gains are of one of two kinds:

1. Architectural advances: bit-parallel memory and arithmetic, cache, interleaved memory, instruction lookahead,

instruction pipelining, multiple functional units, pipelined functional units and data pipelining.

2. Pure hardware advances, basically improvement in instruction cycle time which is costly and limited by the physics of propagating signals through a medium and dissipating heat generated by transistor operation.

Microprocessors only relatively recently started to use architectural advances, whereas supercomputer CPU's used some of them before 1970's. Consequently, performance improvement resulting from some of the architectural advances is not seen in the plot for supercomputer CPU improvements. However, the pure hardware advances are based on advances in technology, which are increasingly costly. For example, the capital cost of a semiconductor fabrication line is growing rapidly with improvements in wafer and features sizes (see Figure 2).

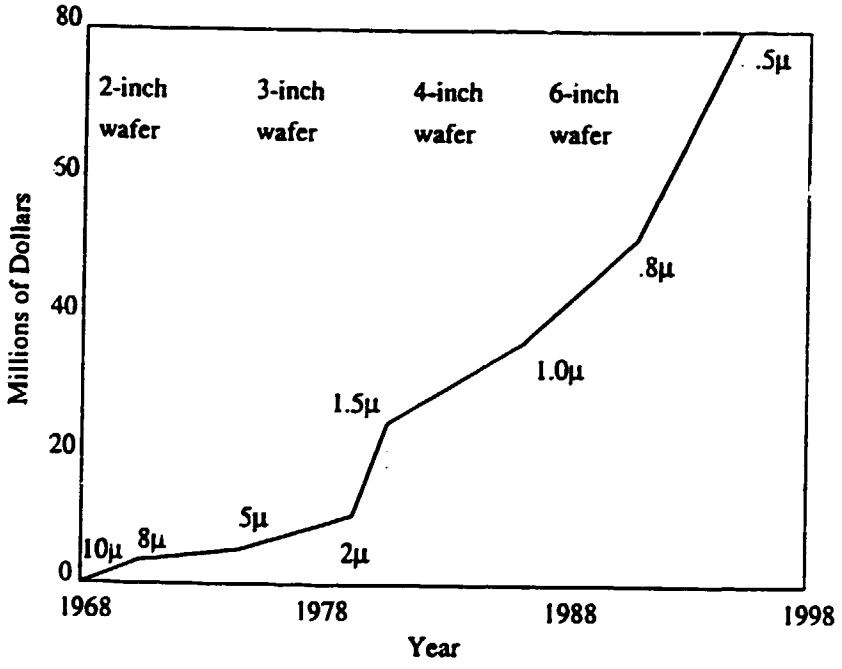

Figure 2. Semiconductor fabrication line capital cost per thousand wafers per week. Feature size is measured in microns. Source: [7].

This capital cost can be much more easily amortized if the produced chips are used not only in supercomputers or parallel computers but also in all other lines of computers. Another reason for using stock hardware in building parallel machines is the constantly improving performance of microprocessors. As shown in Figure 3, designing a specialized processor for parallel processing which has a 10 fold performance advantage over the current uniprocessor design gives the designers just four years of speed superiority. After that time, the improvements in general microprocessor design will nullify any initial performance advantage. Perhaps this is the reason why

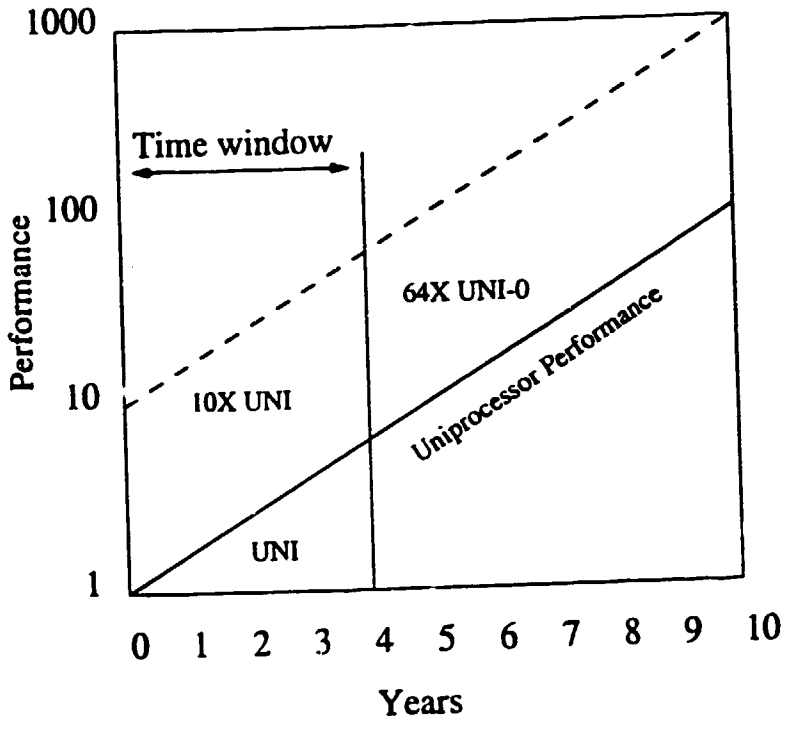

Figure 3. Performance window of opportunity for custom design chips.

all three companies mentioned at the beginning of this section as getting out of the parallel processing manufacturing were using custom design chips in their products. On the other hand, the recently most successful parallel hardware manufacturers (IBM Corp. and Silicon Graphics, Inc.) use the standard CPU chips designed for in their main line of workstations.

3. Importance of Memory

Technology is behind yet another phenomenon important for design and programming of parallel machines. There is a clear trend of DRAM speed improvements lagging behind the processor speed improvements (see Figure 4). In the last twelve years, the CPU speed increased several hundred times, whereas the speed of DRAM chips merely doubled. Both chips are produced by the same technology, however; the advancement in technology for DRAM chips is used to increase RAM density, not speed.

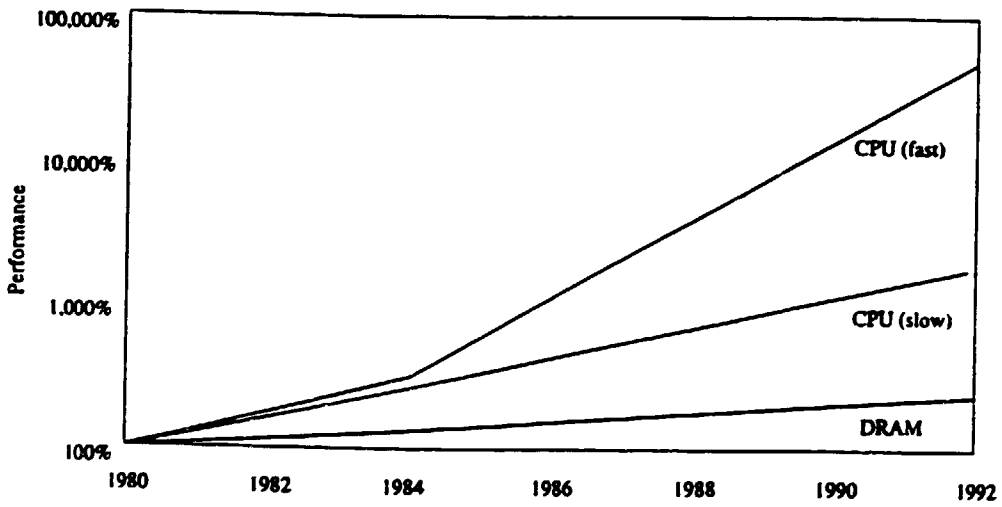

Figure 4. Trends in DRAM and processor cycle time: Source [7].

To mask the difference in speed between the processor and memory, modern processors use caching systems, often two level caches. A cache trades capacity for speed. During program execution, the most recently referenced fragment of the memory

is kept in the cache and data are retrieved from it. Each time data needed by the processor are already in cache, the access is done at (roughly) processor speed. Such an access is called a cache hit. When the data are not available in cache, cache miss happens, and a bucket of data (equal in size to the cache line) that contains the needed data is moved from a slow memory to cache. The access to data is slow in such a case. The cache miss ratio (or in other words the percentage of cache misses over all data accesses) dictates the resultant speed of processing. The bigger the difference in speed between the memory and the processor, the lower the cache-miss ratio must be for the processor to work at near capacity (see Figure 5).

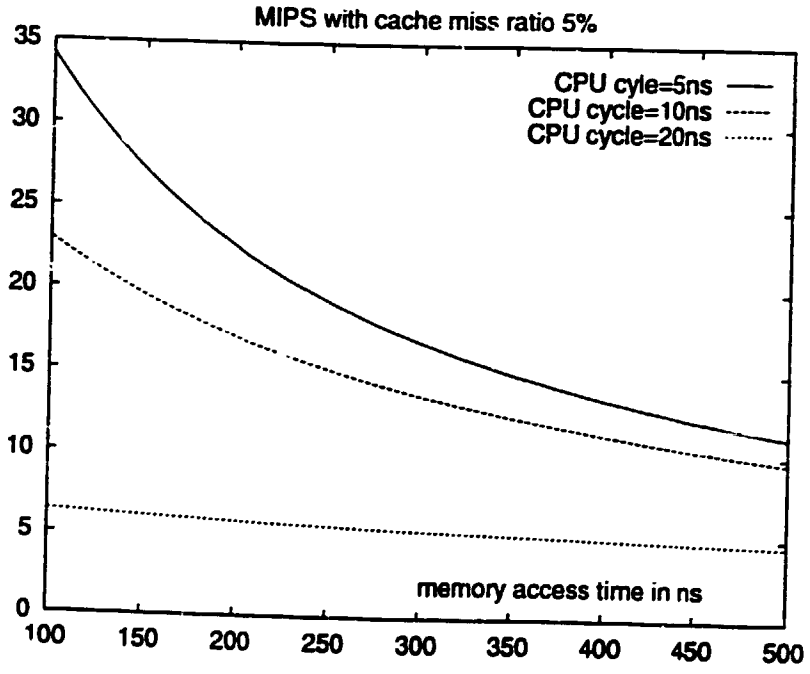

Figure 5. Effects of Memory-Access Time on Speed of Processing.

In general, the cache miss ratio can be improved only by increasing the size of cache. Consequently, an increase in difference between performance of processor and memory is compensated by an increase in the cache size.

A multiprocessor brings not only multiple power of the CPU's together but also improved memory capacity and performance thanks to multiple cache capacity. As a result, a multiprocessor consisting of n processors may perform faster than n times the speed of a single processor for some applications. In such cases, the multiprocessor achieves super-linear speedup. It should be noted that the improved cache miss ratios of component processors of a multiprocessor must provide performance improvement that exceeds the overhead of parallel execution (such as load imbalance, not all processors having the same amount of work, and communication overhead, delayed access to non-local data), so cases of super-linear speedup are rare. However, in all applications, the impact of the extended memory of the parallel computer versus its single processor counterpart can be significant.

David Wood and Mark Hill discuss in [15] the concept of a costup and show that large memories can make parallel computing cost-effective even with modest speedups. Let s(p) denotes the speedup of a program when executed on p processors, i.e.,

$$s(p) = \frac{1/time(p)}{1/time(1)} = \frac{time(1)}{time(p)} .$$

The speedup is linear when s(p) = p, super-linear when s(p) > p, and sublinear (the most often case) when s(p) < p. Let c(p) denotes cost of p-processor machine. The cost-performance of such a machine, costperf(p) is then c(p)*time(p). If a parallel machine is to achieve better cost-performance than a uniprocessor, then costperf(p) < costperf(1), which leads to the following conclusion (see [15]):

p-processor parallel computing is more cost-effective then uniprocessor computing whenever s(p) > c(p).

The main point is that often c(p) < p because processors in the multiprocessor may have less memory each then the uniprocessor. The authors provide an example of SGI systems. As of July 1994, a uniprocessor Challenge DM was priced according to the formula:

$$cost(1,m) = \$38,400 + \$100*m$$

where m is memory size measured in Mbytes. The comparable p-processor, SGI Challenge XL, was priced as follows:

$$cost(p,a,m) = \$81,600 + \$20,000*p + \$100*a*m$$

where a > 1 is the factor of the memory overlap on different processors. By substitution, David Wood and Mark Hill obtained the following formula for SGI machines:

$$c(p,a,m,) = (2.125+0.521*p+0.0026*a*m)/(1+0.0026*m)$$

Figure 6 illustrates costups with SGI prices under the assumption that a = 2, i.e. that the parallel implementation requires twice the memory of the uniprocessor program. Different lines correspond to different number of processors p. The data support the assertion that parallel computing can be cost effective at speedups much less than p for large but practical memory sizes. Wood and Hill conclude that more than one processor might be needed to effectively utilize sufficiently large memories.

In the closing of this section, it should be noted that several different architectural approaches to parallel processing are slowly converging to a similar solution. The workstations interconnected through a fast network, when dedicated to a single application behave like a multiprocessor. The modern shared memory multiprocessor relies on an interconnection network between the global memory and local processor caches, and therefore behaves similarly to the distributed memory multiprocessor. Finally distributed memory machines through extensive use of caches approach in their behavior shared memory machines. The overall trend is to use powerful computing nodes interconnected through a high speed

network of large capacity. The trend is to rely on standard, off-the-shelf components.

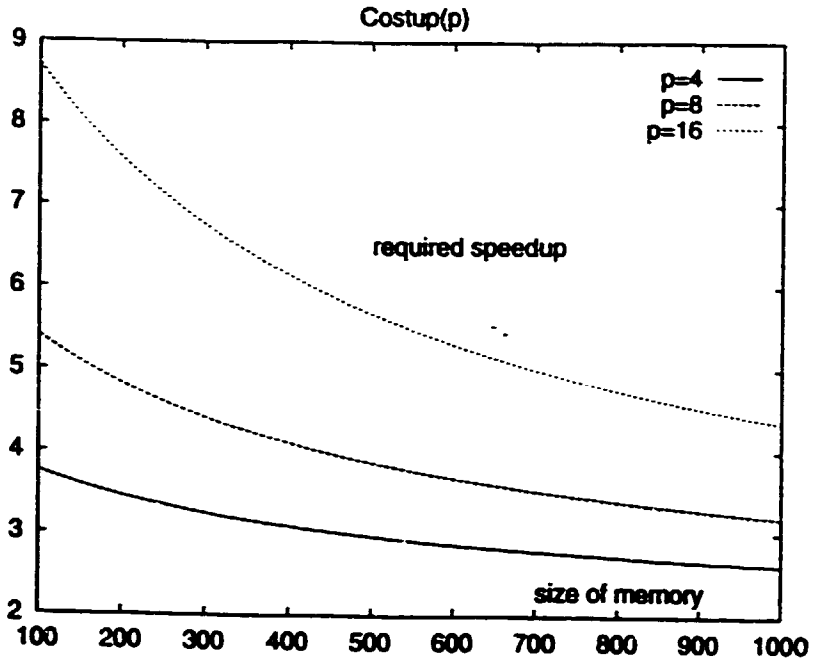

Figure 6. SGI costups with double memory overhead for a = 2.

4. Performance of Parallel Computers

As already discussed, the speedup s(p) of a parallel machine with p processors can be found by comparing time of execution of a program by a uniprocessor (time(1)) and by a multiprocessor (time(p))

$$s(p) = \frac{time(1)}{time(p)}$$

The well known adage that the chain breaks at the weakest link has a computer science counterpart in Amdahl's Law which states that the least parallelizable part of the code limits the speedup. More precisely, if f is the fraction of the code which is inherently sequential (so called Amdahl fraction)

then, independently of the number of processors used s(p) <= 1/f, simply because f*time(1) <= time(p).

Amdahl's Law seems very pessimistic: after all, every program has sequential parts and even if these parts are small and limited to few percent of the code, still the speedup is limited to less than hundred times (see Figure 7).

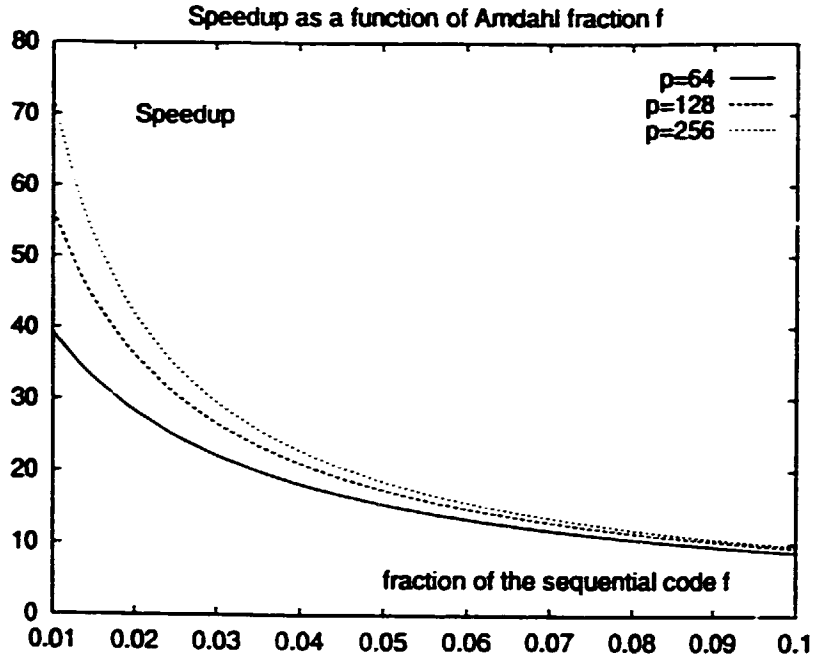

Figure 7. Impact of the Amdahl's Law on the Maximum Speedup.

Fortunately, often the execution time of sequential parts of the algorithm do not change, or change slowly with the growth of the problem size, whereas execution time of parallelizable parts changes rapidly when the problem size is increased.

Hence, the Amdahl fraction is dependent on the problem size. For a wide class of problems f can be made arbitrary small by selecting sufficiently large problem size. Consequently, for such problems, the speedup can be made arbitrary large.

Often the problems computed on parallel machines are too large too fit on a uniprocessor, so measuring an Amdahl fraction for them is impossible or difficult. John Gustafson [6] proposed different measure, g, that represents a fraction of time during which the parallel machine executed the sequential part of the code. Therefore time(p) = g+(1-g) = 1 but time(1) = g+(1-g)*p = p-(p-1)*g, so the speedup is:

$$s(p) = p[1-(1-1/p)*g]$$

The nice feature of this formula is that it clearly shows how to improve the speedup. If we start adding processors (i.e., increasing p) but keep the work of all processors the same, then most likely g will stay the same and the speedup will grow. Likewise, with the constant number of processors, we decrease g by increasing the problem size. The final conclusion is similar to what the Amdahl's Law implies: by selecting large enough problem to keep all processors occupied for a long time, the impact of the sequential parts of the program could be made negligible.

Though these principles may seem simple in theory, applying them to real problems is difficult. To encourage innovation, the annual Gordon Bell Awards are given for achievements in supercomputing. The three categories are performance, price/performance and compiler parallelization. The last six Gordon Bell Awards are summarized in Figure 8. They provide a wealth of information about the current trends in parallel computing.

In figure 8, the winners of price/performance category are marked with black rectangles and winners of performance category by black circles. After initial successes of SIMD machines (please note three CM2 machine winners in 1989-90)

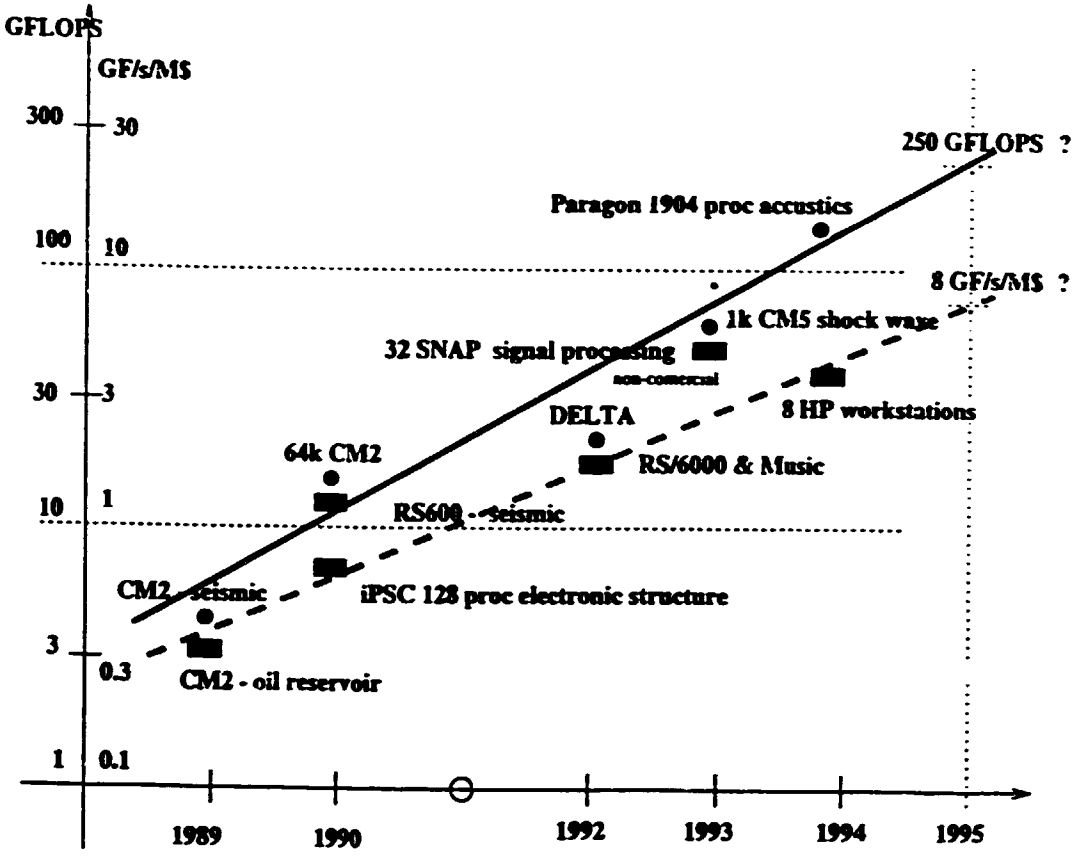

Figure 8. Trends in Gordon Bell Awards Winning Entries.

came the reign of Intel machines (Intel hypercube iPSC in 1990, DELTA in 1992 and Paragon in 1994). The price/performance category is clearly dominated by workstations. A quick glance through applications indicates that scientific computing is still the dominant and favored domain. The trend is very clear in both categories and it indicates rapid exponential growth in capabilities of the

machines.

Over the years, the actual speed record was rapidly growing and the holders of the record were changing quickly. Until recent years, the record was usually held by vectorizing supercomputers with a single or a few processors. Only recently have massively parallel machines started to provide better performance. As of this writing, Intel's Paragon XP/S MP Supercomputer is the record holder with a sustained speed of 281 gigaflops, achieved on an industry standard MP Linpack benchmark in December 1994. The previous record of 170 gigaflops was made by Fujitsu in August 1994. The Intel machine even achieved 328 gigaflops while executing a double-precision complex LU factorization code. The Paragon system used for the record-breaking runs was created by joining two machines at Sandia National Laboratory in U.S. It included 2256 compute nodes, each with three Intel i860 XP processors, a total of 6,768 microprocessors.

It should be noted that since 1993, awards similar to the U.S.'s Gordon Bell Prizes were introduced in Europe. The so-called SuParCup's are awarded yearly at Mannheim Supercomputer Conferences.

5. Distribution of Parallel Programming Resources

Category	USA and Canada	Japan	Europe	Other countries
Number of computers	248	82	143	27
Installed power	54%	27%	17%	2%
Leading countries	USA	Japan	Germany, UK	Korea, Australia

Table 3. The Distribution of Powerful Computers over the World

Distribution of parallel machines and supercomputers is still heavily concentrated in the most developed countries. The data quoted below are based on the TOP500 list of the most powerful computers in the world, compiled by Dongarra, Meurer and Strohmeir for November 1994. The list publishes Rmax, the maximum performance of a machine on one particular benchmark, so it is not indicative of the speed which can be achieved on an entire application. The total performance installed worldwide in these 500 machines is over 2,600 gigaflops, or 2.6 teraflops. The geographical distribution of the computers is given in Table 3.

A similar list of nearly 200 sites with the most powerful computers is maintained by Gunter. The summary of this list for May 1995 is given in Table 4.

Category	USA	Japan	Europe	Others	Government	Academia	Industry
Sites 1-25	16	7	2	0	14	5	6
Sites 26-100	31	15	26	3	30	15	30
Sites 100-190	40	10	33	8	14	43	34

Table 4. The Distribution of Powerful Computing Sites over the World

The sites outside of US, Europe and Japan were located in Canada, Korea and Taiwan (for sites ranked 26-100) and two sites in Australia, two in Hong Kong, as well as single sites in Mexico, Canada, China and Saudi Arabia in tier 100-200. It is interesting to observe that the most powerful sites are mainly governmental laboratories, medium sites are mainly commercial and the smallest sites are mainly academic.

The growing importance of parallel computing to many countries in the world was demonstrated in the special session of

Supercomputing'93 entitled Supercomputing Around the World. Among others, researchers from Singapore, Indonesia and Malaysia were talking about their countries' support for using parallel computers for aerospace, oil, and environmental applications.

6. Software Models

The increasing importance of parallel processing prompted growth in the body of standardization in parallel programming languages and tools. Yet, there is no evidence of convergence of the supported programming paradigms to a single model. Currently there are two most popular models for parallel program design: data parallelism and message passing.

Data parallelism is popular because of its simplicity. In this model, a single program (and therefore a single thread of execution) is replicated on many processors and each copy operates on separate part of data. Depending on the tightness with which the execution of programs is synchronized, there are two modes of using data parallelism. When each instruction of the program is synchronically executed on all processors, then Single Instruction Multiple Data (SIMD) mode is used. Such tight synchronization requires hardware support.

SIMD machines were quite popular at the turn of the last decade (see Gordon Bell Awards in the previous section). From the software engineering point of view, SIMD machines are easy to program because there is a single flow of control on all processors. The main focus of parallelization is to find large data structures that can be distributed to all processors to keep them all occupied. Another concern is to minimize the data movements necessary to provide data to processors that are to execute them. Due to the small granule of parallelism (single instruction) SIMD machines consist of a very large number of simple processors (tens or hundred thousand of processors in a single machine is not unusual). Each of these processors must either execute the same statement as all the others or idle, so SIMD machines achieve poor efficiency on programs that do not contain sufficiently large data structures. They also do not perform well on programs which

require irregular data references (list structures, dynamic memory, etc.). The consensus is that SIMD architecture has a very specialized niche of applications (e.g., visual information and scene processing), but it is not the best choice for general parallel processing.

Data parallelism can be also used in a loosely synchronized mode, when the program execution consists of two stages:

1. computational stage, when copies of the same program are executed in parallel on each processor locally. The execution can differ in the conditional branches taken, number of loop iteration executed, etc.,

2. data exchange stage, when all processors concurrently engage in exchanging non-local data.

It should be noted that the data exchange stage is very simple in case of shared memory machines (when it can be enforced by use of locks or barriers). The frequency of synchronization in SPMD model can be adjusted to correspond to the latency of the interconnection network. The SPMD model is very adequate for scientific computing which often requires applying basically the same algorithm at many points of computational domain. SPMD parallel programs are conceptually simple because of a single program executing on all processors, but more complex then SIMD programs.

For more complex applications, running a single program across the parallel machine may be unnecessarily restrictive. In particular, dynamically changing programs with unpredictable execution times result in poorly balanced parallel computations when implemented in SPMD mode. This is because in SPMD mode, processors synchronize at the data exchange stage, and none of the processors can proceed to the next computational stage until all others reach the data exchange stage.

The SPMD model was abstracted into a Bulk-Synchronous Parallelism model proposed by Leslie Valiant of Harvard University [14]. The model attempts to provide the abstraction

for parallel algorithm description that lends itself to performance analysis. The model also became a basis for a library that facilitates the creation of portable parallel software.

The BSP model consists of three components

1. Processors perform processing or memory functions.

2. A router provides point to point communication between pairs of components.

3. A synchronization mechanism synchronizes all or a subset of the components at regular intervals of L time units (L is called also the synchronization periodicity).

In the BSP model, computation consists of a sequence of supersteps. In each superstep, a component performs some local computation and transmits messages to other components. After a period of L time units, a global check is performed to determine if all components completed the superstep. If not, the superstep is extended by another L time units, after which the check is made again. In the BSP model, the data transmitted are not guaranteed to be available at the destination until after the end of the superstep at which they were sent.

Using this model, the cost of an algorithm can be expressed in terms of L and g, two parameters that are defined by the network latency and bandwidth, respectively. Using the BSP cost of an algorithm, it is possible to predict the performance of the algorithm on new hardware, given the values of the parameters L and g for this hardware. The BSP model facilitates an algorithm optimization through data distribution selection based on the characteristics of the problem rather than the architectural features of the target machine.

A BSP computer is characterized by the following set of parameters: number of processors p, processor speed s, synchronization periodicity L, and a parameter to indicate the

global computation to communication balance g. The synchronization periodicity L is the smallest number of time steps between successive synchronization operations. Parameter g is the ratio of the total number of local operations performed by all processors in one time unit to the total number of words delivered by the communication network in one time unit. Processor speed s is measured in flops (floating point operations per second). Synchronization parameter L is measured in flops. Parameter g is measured in flops per word.

BSP parameters allow for algorithm performance analysis. For example, consider a superstep that needs to communicate h words of data. Since it takes g*h time units for the communication network to deliver the data to its destination, and L units to synchronize all the processors performing the superstep, at least L+g*h units of computation are needed to keep the processor busy; a level of computation less than this threshold results in idling of some processor, and therefore is a source of inefficiency.

In terms of the BSP parameters, distributed memory parallel machines are often characterized by large values of s (relatively fast processors) and low values of L and g (a communication network with low latency and large bandwidth). A general purpose network of workstations, on the other hand, is characterized by values of s that are somewhat lower than for the parallel machines and values of L and g that are much larger than the corresponding values for the parallel machines (high latency and low bandwidth due to the loosely coupled nature of these networks). Thanks to this distinction, optimal BSP algorithms for network of workstation use different data distribution then those designed for a parallel computer.

BSP algorithms can be directly implemented in a high-level traditional language (e.g., C or Fortran) with addition of the necessary calls to BSP primitives. The Oxford BSP Library [9], developed by Richard Miller can be used for this purpose. The library is based on a slightly simplified version of the model presented in [14]. These simplifications require that the processors are allocated statically before the program is run and the programs are written in a SPMD mode. The most

significant feature of the library is the support for remote assignment as a means for non-local data access. The library consists of just six functions and is simple to use. Despite its simplicity, we have found it to be quite useful and robust. Tne library for C-BSP programming includes the functions for

1. Starting and ending a BSP session

2. Starting and ending a superstep

3. Fetching and storing a values from a remote processor.

Computationally intensive applications with frequent communication and synchronization require careful design for efficient execution on networks of workstations. Such design is supported by Bulk-Synchronous Processing (BSP) model. In [11] authors demonstrate an implementation of a plasma simulation on a network of workstations and the use of BSP analysis techniques for tuning the program for this kind of a machine. They also compare the performance of the BSP implementation with a version based on MPI and conclude that the BSP model, serving as the basis for an efficient implementation, compares favorably with MPI.

The memory distributed machines use message passing for exchanging data between different processors. The SPMD model may shield the user from specifying the detailed data movements thanks to data distribution directives from which a compiler generates the message passing statements. However, the user which decides to write the message passing statements himself has full control over the program execution. In particular, the user may define when and how many processors synchronize in their execution. This gives the user a lot of flexibility at the cost of requiring the user to make very intricate and detailed description of the program. The programs tend to be longer and more complex than their SPMD counterparts, and therefore more error-prone. Once debugged and tuned up, they are also more efficient. The flexibility of the message passing model makes it applicable for a wide variety of problems. As discussed below, the newly developed

standard library of functions for message passing, MPI, has a potential of becoming a universal tool for parallel software development.

7. Trends in Languages

There is a plethora of research parallel programming languages with different flavors to choose from, starting from functional, dataflow to object oriented, logical etc However, the majority of parallel programs are still itten in Fortran. Since 1950's this language was a favorite choice for writers of scientific programs and particularly for generations of graduate students in applied sciences. Over the years, Fortran underwent a remarkable transformation, from one of the first languages at all to the first language with a well defined standard (Fortran66) to structured programming of Fortran77, to data parallel and object-oriented Fortran90 and finally to the newest standard of High Performance Fortran (HPF). Each generation brought with it new features and set a new standard for the manufacturers of hardware and compilers.

Compared to Fortran77, Fortran90, which was introduced at the beginning of this decade, brought to the world of Fortran users several modern language design features, such as:

1. Derived types, kinds, pointers and dynamic memory allocation that enable the user to define own data types and dynamically allocate data structures.

2. Modules, characterized by public and private data types. Modules can be imported from other programs by the **USE** clause and renaming.

3. Array operations and new control structures allowing for a very concise and elegant definition of data parallel programs.

4. Recursive procedures.

5. Interface blocks for abstract definition of the input/output as well as terminal-oriented source forms.

The first two features enable the users to write object-oriented programs. In brief, object-oriented programming involves developing user's own abstraction of the application domain. This abstraction is defined by the user in the form of data abstraction, object types and type inheritance. An object is defined by its (hidden) state and a set of operations that are applicable to it. Abstract data type is just a set of objects whereas a class is an abstraction of objects. Each object has private data and attributes (that define its implementation) and public data and attributes visible to users of the object. Such a distinction between object's data and attributes is often referred to as data encapsulation. Finally, polymorphism and function overloading are another characteristics of an object oriented language. Basically, they allow an operator or a function to carry different processing for different types of their arguments. The simplest example of such overloading is its use to define an optimized function of raising to a power. It could be done by using a power series approximation for non-integer exponents and also by an iterative multiplication for integer exponents. Careful analyses of Fortran90 features indicates that all the above features can be expressed in Fortran90 [10].

Another important feature of Fortran90 is the ability to operate on the whole arrays. Array expressions allow the user to define arrays of various shapes and apply operators to such arrays in a piecewise manner. Array shapes can use a set of conditions to decide to which particular elements of the argument an operation should be applied (**WHERE** clause). The array expressions allow for a very succinct definition of data parallel operations.

A new generation of parallel Fortran, HPF was introduced in 1993 by an HPF Forum, a group of parallel hardware manufacturers and academic, industrial and governmental users of high performance machines. During 1994 there were six announced commercial HPF products and 11 announced commercial HPF efforts, with many of these compilers becoming available now (mid of 1995). HPF introduced new data partitioning directives, like **ALIGN/REALIGN** data structures relative to

each other, **DISTRIBUTE/REDISTRIBUTE** data structures (or their templates) to processors according to one of the predefined patterns (**BLOCK, CYCLIC**, or **BLOCK-CYCLIC**). Directives for definition of processor arrangements (**PROCESSORS**), or loop parallelization (**INDEPENDENT/FORALL**) are available. These directives enable the user to define data movements indirectly without the need of detailed description of the message passing statements that are must be executed to achieve directive-defined effect.

Critics of HPF think that the HPF standard is not general enough. In particular, HPF does not allow for dynamically defined alignments and distribution that are permitted in Fortran HPF+ [3]. However, standardization of the language features is extremely important for users and compiler and tool writers because it protects their software investments against changes in the architecture. In that respect, introduction of Fortran90 and then HPF is an important step forward towards more stable parallel software.

HPF can be seen as the flagship of the data parallelism camp. On the other hand, the supporters of message passing based parallel programming achieved standardization of their approach in the Message Passing Interface (MPI). MPI is a large library of the message passing utilities that includes 125 functions. The basic MPI subset, sufficient for writing simple applications, consists of just the following six functions:

1. **MPI_INIT** - to initialize MPI on in a process,

2. **MPI_COMM_SIZE** - to find the number of processes participating in the MPI session,

3. **MPI_COMM_RANK** - to find a unique rank of the calling process among the MPI session participants,

4. **MPI_Send** - to send a message to the other processes,

5. **MPI_Receive** - to receive a message,

6. **MPI_Finalize** - to terminate the MPI session.

The innovations of MPI are centered around an abstract view of the communication. This abstract view supports portability of programs using MPI to different machines. Messages in MPI are described as triples consisting of an address, data count and data type. Data types in such triples can be user defined. MPI allows the processes to group themselves and arrange themselves into a hierarchy where each process has its own rank. A process can have different ranks inside different groups and it can participate in different communication sessions concurrently. MPI provides also a default initial group whose members are all processes that executed the **MPI_INIT** function. Families of messages can be defined in terms of communication context and group.

MPI provides also more complex features, such as collective communication that includes data movements and global reduction operations. MPI allows the user to define virtual topologies and use different communication modes. It provides also functions for debugging and profiling and support for heterogeneous networks. The MPI standard does not define, purposefully, how the MPI startup is implemented, the amount of system buffering, or the limitations on recognized errors to avoid unnecessary restrictions on implementations.

Judging from the widespread popularity of the Parallel Virtual Machine (PVM), MPI can become an important step towards providing efficient and unifying tool for expressing message passing in parallel and distributed applications. Although introduced recently (in 1993), MPI has been quickly embraced by manufacturers and is supported on many parallel machines (among them Cray T3D, Intel Paragon and IBM SP2).

8. Conclusions

Parallel processing is at a critical point of its evolution. After a long period of intense support by government and academia, it slowly moves to derive the bulk of its support from the commercial world. Such a move brings with it a change of emphasis from record breaking performance to price

performance and sustained speed of program execution. The winning architectures are not only fast but also economically sound. As a result, there is a clear trend towards widening the base of parallel processing both in hardware and software. On the hardware side, that means using off-the-shelf commercially available components (processor, interconnection switches) which benefit from rapid pace of technological advancement fueled by the large customer base. The other effect is the convergence of different architectures thanks to spreading the successful solutions among all of them. Workstations interconnected by a fast network approach the performance of parallel machines. Shared memory machines with multilevel caches and sophisticated prefetching strategies execute programs with efficiency similar to distributed memory machines.

On the software side, the widening the base of the users currently relies on standardization of parallel programming tools. By protecting the programmer's investment in software, standardization promotes development of libraries, tools and application kits that in turn will attract more end-users to parallel processing. It appears that parallel programming is ending a long period of craft design and is entering a stage of industrial development of parallel software. This is an industry in the making that will provide new opportunities for software developers and investors.

Acknowledgment

Some of the research discussed in this chapter has been supported from the following U.S. agencies: Office of Naval Research under N00014-93-1-0076, National Aeronautical and Space Agency under NGT-70334, and National Science Foundation under grants ASC-9318184 and BIR-9320264. The content of this chapter does not necessarily reflect the position or policy of the U.S. Government - no official endorsement should be inferred or implied.

References

[1] Gordon Bell, ''Why there won't be apps: The problem with MPPs,'' IEEE Parallel & Distributed Technology: Systems & Applications, vol. 2, no. 3, pp. 5-6, 1994.

[2] Philip Carnelley and William Cappelli, Beyond the Data Warehouse, New Markets for Parallel Computing, Ovum Reports, 1995.

[3] Barbara Chapman, Hans Zima and Piyush Mehrotra, ''Extending HPF for Advanced Data-Parallel Applications,'' IEEE Parallel & Distributed Technology: Systems & Applications, vol. 2, no. 3, pp. 59-70 , 1994.

[4] James Cownie, ''Why MPPs?'' IEEE Parallel & Distributed Technology: Systems & Applications, vol. 2, no. 3, pp. 7-8, 1994.

[5] Evolving High Performance Computing and Communications Initiative to Support the Nations's Information Infrastructure, National Research Council, National Academy Press, Washington, D.C. 1995.

[6] John L. Gustafson, ''Reevaluating Amdahl's Law,'' Communications of the ACM, vol. 31, no. 5, pp. 532-3, 1988.

[7] John Hennessy and David A. Patterson, Computer Architectures: A Quantitative Approach, Morgan Kaufman Publishers, Inc., 1990.

[8] Ken Kennedy, ''Is Parallel Computing Dead ?'' Parallel Computing Research, vol. 2, no. 4, pp. 2-19, 1994.

[9] Richard Miller and J.L. Reed, The Oxford BSP Library: Users' Guide, Version 1.0, Oxford Parallel Technical Report, Oxford University Computing Laboratory, 1994.

[10] Charles Norton, Boleslaw Szymanski and Viktor Decyk, ''Object Oriented Parallel Computation for Plasma PIC

Simulation,'' Communications of the ACM, vol. 38, no.10, October, 1995.

[11] M. Nibhanupudi, C. Norton, and B.K. Szymanski, ''Plasma Simulation on Networks of Workstations using the Bulk-Synchronous Parallel Model.'' Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '95). Athens, GA. November, 1995, CSREA, 1995 pp. 13-22.

[12] Monica Snell, ''Supercomputing: Same name, different game,'' IEEE Computer, vol. 27, no. 11, pp. 6-7, 1994.

[13] Steve Wallach, ''Teraflops into laptops,'' IEEE Parallel & Distributed Technology: Systems & Applications, vol. 2, no. 3, pp. 8-9, 1994.

[14] Leslie Valiant, ''A bridging model for parallel computation,''Communication of the ACM, vol. 33, no. 8, pp. 103-11, 1990.

[15] David Wood and Mark Hill, ''Cost-Effective Parallel Computing,'' IEEE Computer, vol. 28, no. 2, pp. 69-72, 1995.

Glossary

ATM: Asynchronous Transfer Mode, a new standard of transmitting data over a network that unifies the need of computer processing and telecommunications (voice and video transmission).

BSP: Bulk Synchronous Parallelism model developed to unify algorithm description for parallel machines (Section 6).

Cache: Fast but expensive memory used to speed up access to data in main memory of the computer (see Section 3).

Cluster of Workstations: or COW, a parallel machine created by

joining independent workstations by a network (usually Local Area Network, LAN).

DRAM: Dynamic Random Access Memory, is currently the technology producing the densest computer memory chips (16-64MB or millions bytes in a single chip).

File server: A special computer in a network of workstations responsible for providing the file storage and services for the entire network.

Massively parallel machine: A computer with many processors, not necessarily the fastest computer on the market (see supercomputer).

Multimedia: Use of numerical data, voice and pictures/movies, in data processing.

Multiprocessor: A computer with many processors, a synonym of parallel computer. Supercomputers often are multiprocessors.

SIMD machine: Single Instruction Multiple Data multiprocessor. It consists of a large number of simple processors, each executing the same instruction.

SPMD: Single Program Multiple Data mode of parallel processing. Each of many processors executes the same program on different data. Unlike SIMD computer, conditional statements may cause that at any given instant each of the processors in SPMD mode may execute different instructions.

Supercomputer: Ultra-fast computer for numerical computation, usually based on vector units (specialized processors for matrix and vector operations) and some, not necessarily massively, parallelism.

Uniprocessor: a computer with a single processor, a synonym of sequential machine. The first supercomputers were uniprocessors with vector units.

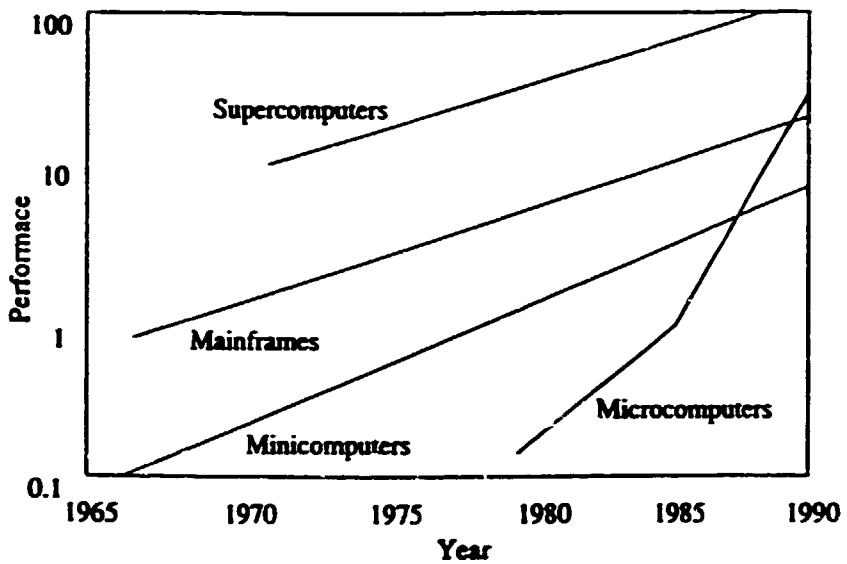

Figure 1: Trends in microprocessors and Mainframe CPU Performance Growth

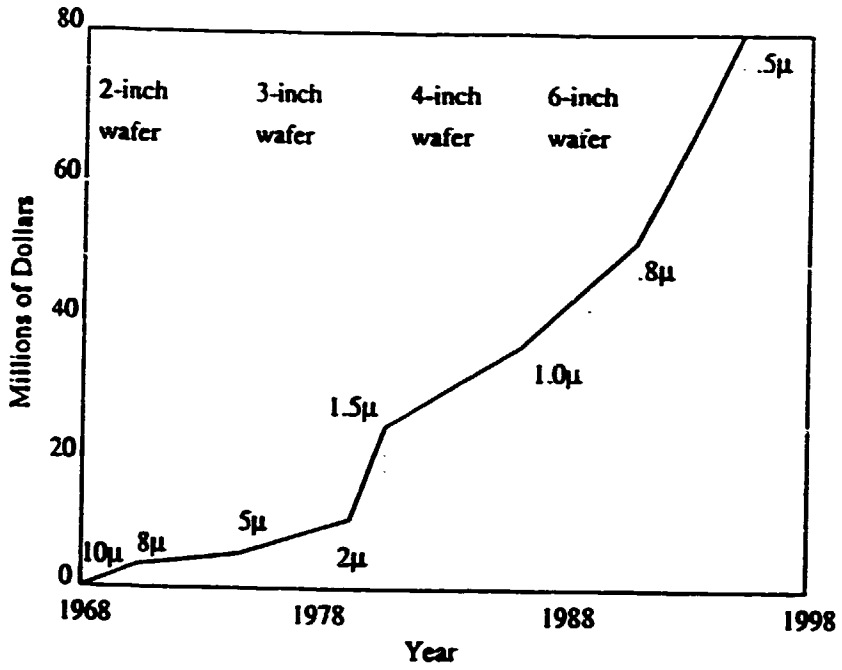

Figure 2: Semiconductor fabrication line capital cost per thousand wafers per week. Feature size is measured in microns. Source: [7]

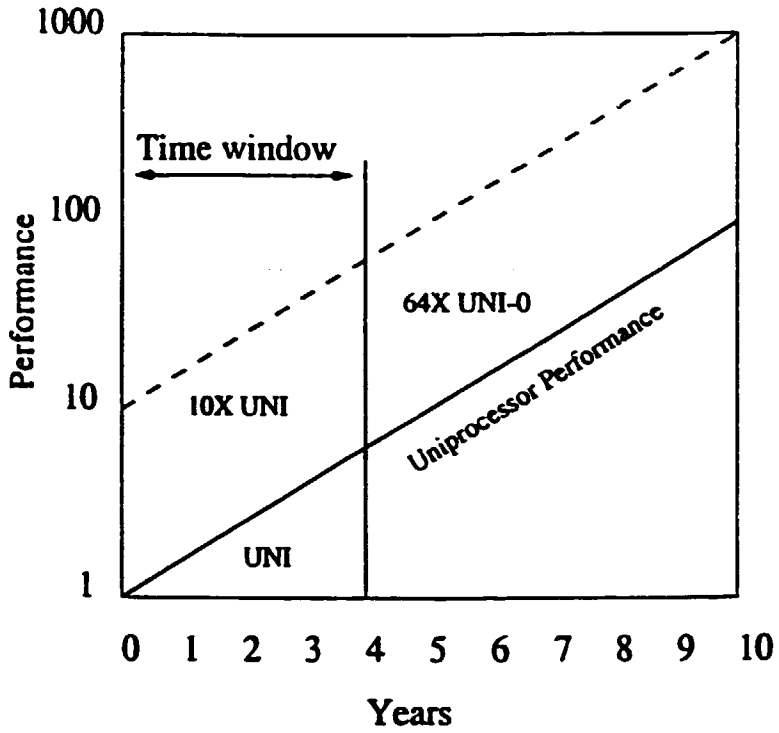

Figure 3: Performance window of opportunity for custom design chips

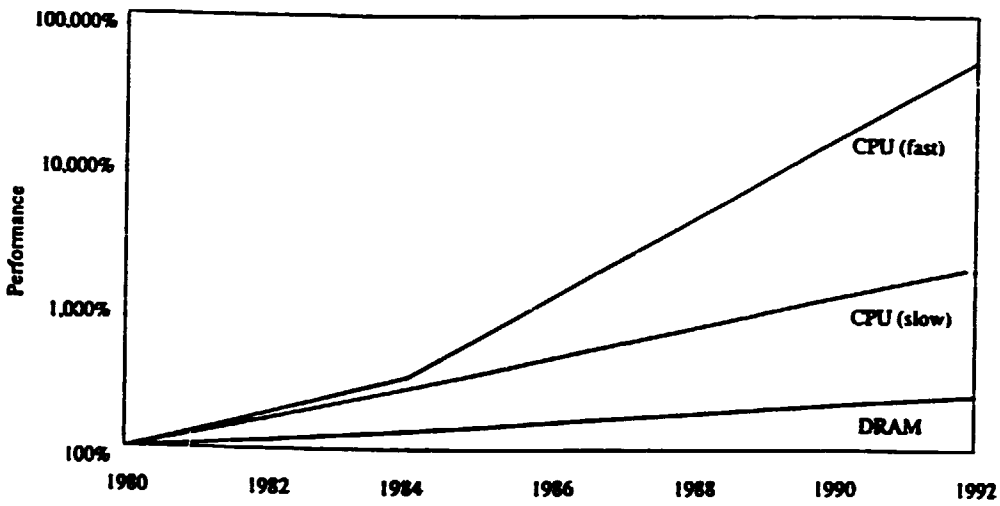

Figure 4: Trends in DRAM and processor cycle time: Source [7]

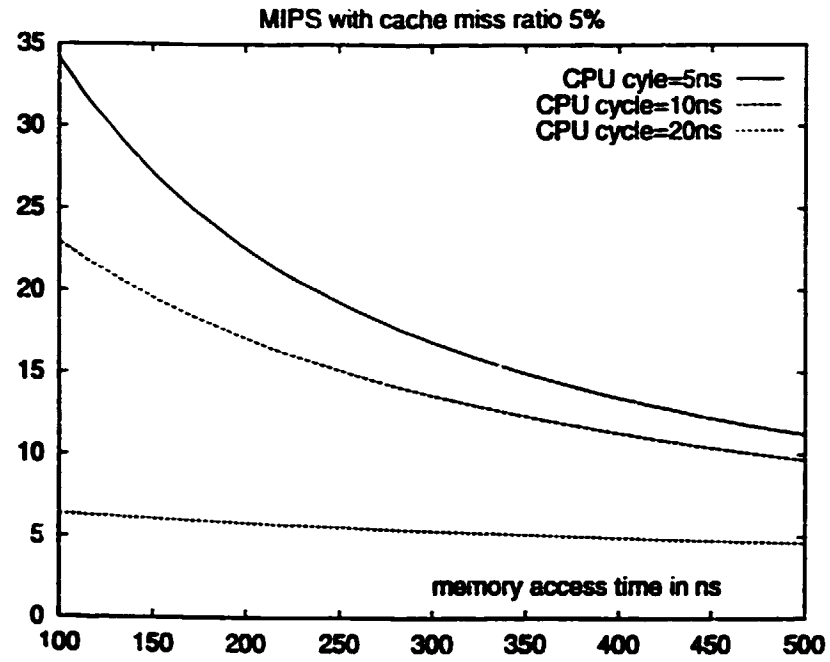

Figure 5: Effects of Memory-Access Time on Speed of Processing

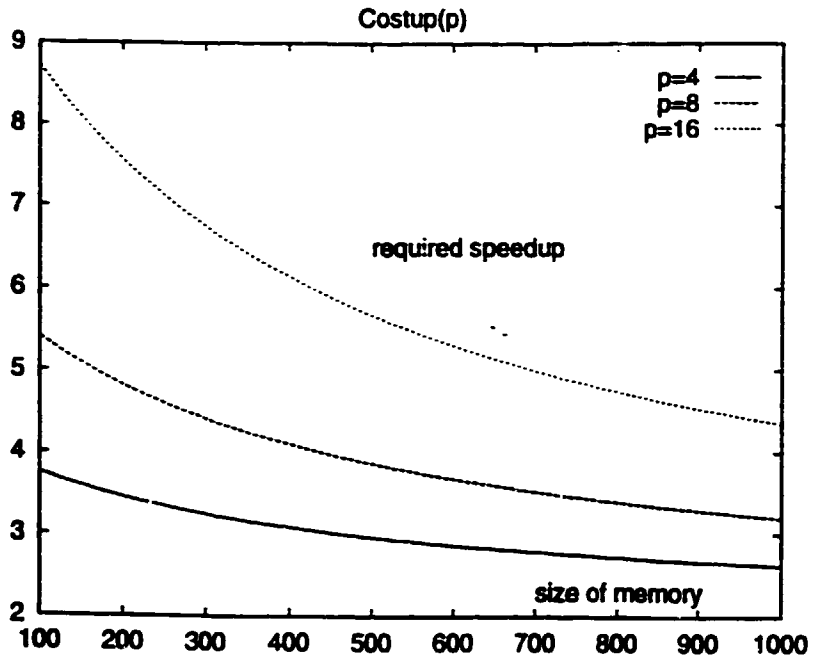

Figure 6: SGI costups with double memory overhead for $a = 2$

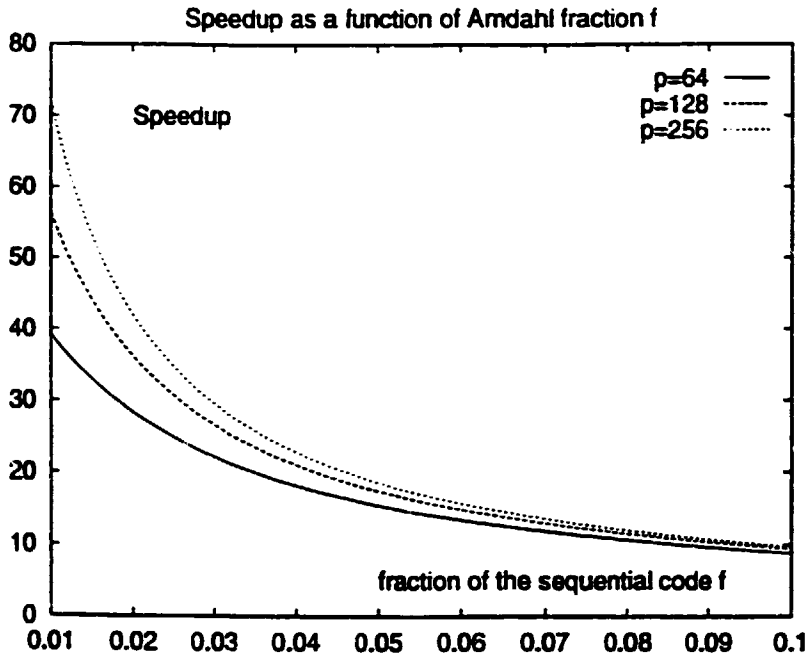

Figure 7: Impact of the Amdahl's Law on the Maximum Speedup

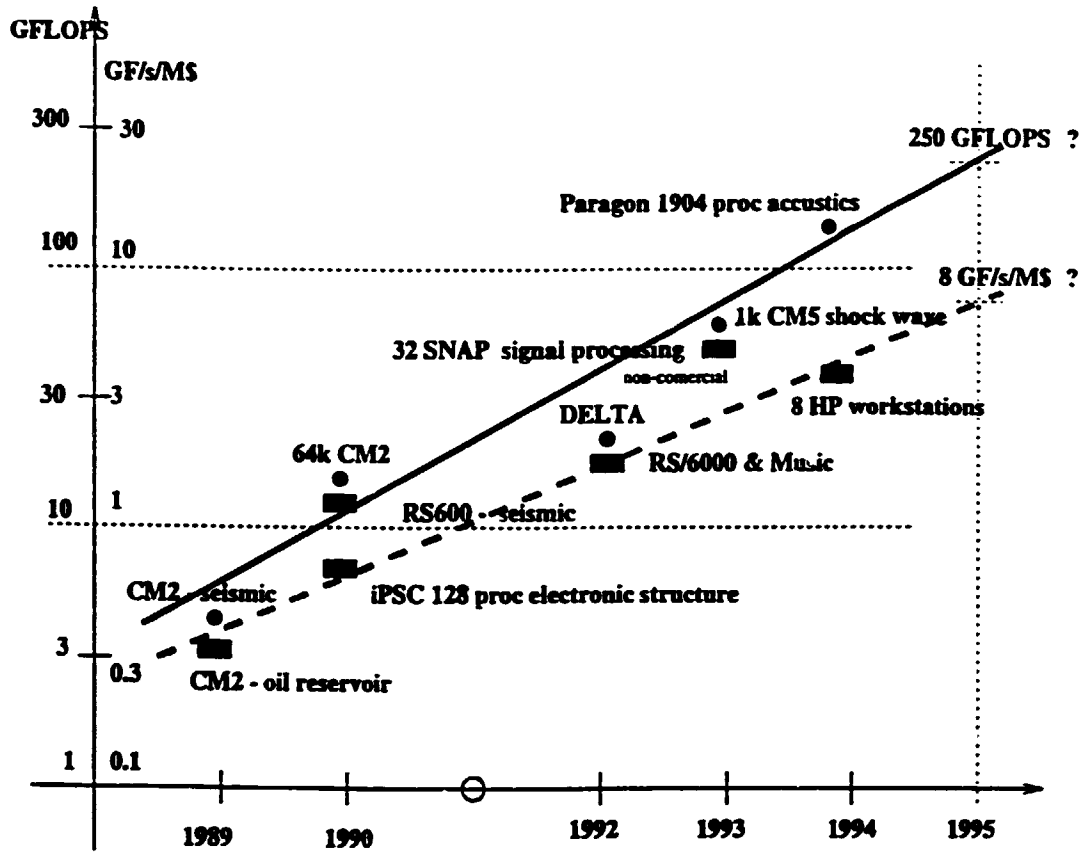

Figure 8: Trends in Gordon Bell Awards Winning Entries