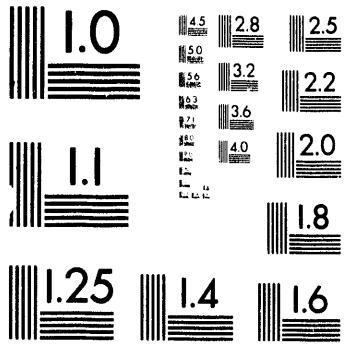

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

1 of 1

2.

Conf-9303248--1

PNL-SA-21769

HIGH PERFORMANCE COMPUTING IN CHEMISTRY AND MASSIVELY PARALLEL COMPUTERS: A SIMPLE TRANSITION?

R. A. Kendall

March 1993

Presented at the
Sanible Symposia 1993
March 15-19, 1993
St. Augustine, Florida

Work supported by
the U.S. Department of Energy
under Contract DE-AC06-76RLO 1830

Pacific Northwest Laboratory
Richland, Washington 99352

Submitted to Proceedings of the Sanibel Symposia 1993

High Performance Computing in Chemistry and Massively Parallel Computers: A Simple Transition?

Rick A. Kendall[‡]
Molecular Science Software
Molecular Science Research Center
Environmental Molecular Sciences Laboratory
Pacific Northwest Laboratory[£]
Richland, WA 99352, USA

Abstract:

A review of the various problems facing any software developer targeting massively parallel processing (MPP) systems is presented. Issues specific to computational chemistry application software will be also outlined. Computational chemistry software ported to and designed for the Intel Touchstone Delta Supercomputer will be discussed. Recommendations for future directions will also be made.

---

‡ The author may be reached via email at: ra_kendall@cagle.pnl.gov

£ Pacific Northwest Laboratory is operated by Battelle Memorial Institute for US. Department of Energy (DOE) under Contract DE-AC06-76RLO1830.

I. Introduction

The advent of massively parallel processing (MPP) supercomputers has been an exciting and challenging benefit to computational science. Many of the algorithms and theoretical models used by the computational chemistry field are very compute intensive, and the computational chemistry market is an obvious target of many MPP vendors. Computational chemists have long been at the forefront of utilizing and developing software on the leading edge of computational technology. The delivery of first and second generation usable MPP hardware has enticed many computational chemistry groups to begin focusing efforts on the development of chemistry software for parallel computing systems. To date, efforts on modern MPP systems are distributed among primarily academic and national laboratory environments with little effort from the vendor community. These efforts also span the entire spectrum of computational chemistry methodologies and algorithms (e.g., from molecular modeling and dynamics to full configuration interaction calculations) and have been conducted on the gamut of available MPP hardware. Furthermore, these efforts have made significant progress, but the use of high-performance computing systems, specifically massively parallel computers, is far from routine.

In this article I outline issues that challenge software development with respect to a very technologically volatile hardware industry as a whole with a focus on MPP systems of today and tomorrow. The perspective is that of a computational chemistry application developer, and what I see available now and in the near future. No one can predict long-term trends in the extremely volatile computer industry, so I will not try, except to state that software development in the future will be different than the way applications are developed today. In section II, discussions of the general issues regarding MPP technology and issues pertinent to computational chemistry applications are presented. In section III, results of chemistry applications on the Intel Touchstone Delta are presented. Conclusions and recommendations are made in section IV.

II. Software Development Issues for MPP Computational Chemistry Software.

A. Software Engineering, Hardware, and Software Life Cycle.

Any software engineering (SE) text warns that the lack of effort in the design of modular and reusable software will eventually cause a complex software system to collapse under it's own weight [1]. The concepts of modularity and reusability are without regard to the particular programming language used for a given software system. The staffing requirements for maintaining a required but poorly designed application are substantial. Computational chemistry software applications are not immune from this process; in fact many of the software developers in computational chemistry are graduate students rarely trained in any aspects of modern computer science or SE. The programming effort is usually the last part of the theoretical or model development and is sometimes less interesting to a student trying to finish a thesis effort.

Computing hardware has undergone a tremendous series of advances over the last two decades. The supercomputing industry was born in the 1960s, flourished in the 1970s and 1980s, and has grown dramatically in the last few years. In the 1970s and early 1980s, the hardware designs lasted on the order of 5 to 7 years. The doubling of computer power with advances in hardware technology, and the interface to that technology, operating systems, optimizing compiler design, etc., progressed at a rate that allowed software developers time to adapt and make algorithmic modifications that made optimal use of the computing resources available. In the last 5 years, the hardware technology growth curve has drastically changed. With the advent of reduced instruction set chip (RISC) technology, the period for doubling raw compute power is now somewhere between 12 and 18 months (see Figure 1) [2]. By the time a computer system is procured and delivered, it is most likely out of date (although not obsolete). This rapid development has even caused the computing industry to implement leap frog hardware development efforts to keep pace with the demands of the computing user base.

The hardware used in the past and today has a standard life cycle [1]. When the hardware is first delivered or developed, it usually has a high failure rate or is not as useful as might be expected. This is sometimes due to the overall software interface to that hardware. The bugs in the system get worked out, and the system becomes useful. Then the user community, computational chemists included, saturate the resource to do scientific development and applications. Over time, the system becomes obsolete due either to inevitable hardware failures or, as is more likely, to the

fact that the resource can no longer meet the computational requirements of the user community in a cost effective manner (see Figure 2).

This simplistic "usefulness" model also applies to software as well [1]. The initial software application usually has a very limited set of functionality. As more robust algorithms and additional functionality are implemented, the software becomes more "useful" to the user community. Without further algorithmic developments the "usefulness" of a software system will asymptotically approach some steady-state level and not deviate from that (see Figure 3). What more realistically happens is that as new functionality is added more bugs and design flaws are uncovered and the failure rate of the application grows, with the software becoming less "useful." In time, maintenance and development efforts usually reduce the failure rate and make the code "useful" again. It has been said that the process of porting an application to an MPP environment is a process of "rebugging software," and the experience of the efforts in the computational chemistry community have shown this to be true.

B. Programming Models.

There are a wide variety of specific programming models and tools that can be used to develop a working programming system on a parallel computer [3-6]. The obvious and key point is that there must be a parallel algorithm for the requisite computational task(s). The various programming models fall into the traditional classes of data parallel (DP), shared memory (SM), and distributed data/task models. The latter is the common multiple-instruction, multiple-data (MIMD) programming model with message passing between processors. In practice most developers use a single program MIMD model and assign data and tasks based upon the identity of a given processing node. The basic problem with all programming models is that on any given hardware there exists a locality of data problem. The program becomes a bookkeeping algorithm that finds the data for the computational task at hand.

In traditional SM environments, there is a flat memory, so access to any segment of memory is uniform. In DP and MIMD you have an obvious data locality problem when sharing data among processes and tasks. Simple replicated data algorithms circumvent this by holding a copy of the requisite vectors/matrices on each node. With the current vendor offerings of 16 to 64 Mbytes per physical processing node, this is a clear limitation. Moreover, the replicated data algorithms do not scale to hundreds or

thousands of processors. These obvious problems have caused many vendors to look at globally addressed memory that is physically distributed. This gives the programmer the look and feel of a SM programming environment, a major benefit, but there are potentially drastic performance penalties for accessing non-local memory. This brings the data locality problem back to the programmer and the software tools available on the system. Now that several general aspects of various programming models have been outlined, a discussion of each in more detail follows.

The Shared Memory programming model is probably the most widely used and best understood model simply because of the amount of time that the computational community has used this technique, but few quantum chemistry codes make use of this programming model even today. There are several specific approaches that have been implemented on UNIX Workstations and on various low and high end supercomputers (Silicon Graphics, Stardent, Alliant, Convex, Cray, etc.). The general scheme used to parallelize applications is to identify shared and private segments of memory and have the owner compute a specific portion of the shared data structures and all of the private data/tasks available to the specific process. There has been much compiler work done on this programming model and it is reasonably well understood by the computer science field. Programmers can tune and optimize their code using compiler directives that help the compiler understand the application layout. This is in my estimation the most efficient parallel programming model for computational chemistry applications in wide use today. Unfortunately, the underlying hardware is not scalable; it is very expensive to realistically extend to large numbers of processors (e.g., greater than 100) and very large memory sizes (e.g., greater than 10 Gwords). This has forced the development of the above mentioned global accessible memory that is physically distributed (e.g., Kendall Square Research, Cray Research Inc. T3D, etc.). The overall programming model stays the same, but there is an added task of making sure the data locality is preserved to avoid thrashing of pages from non-local processors.

The data parallel programming model is one that has been used primarily on the SIMD architectures by design but is not limited to these machines. This is also the underlying principle behind the High Performance FORTRAN (HPF) language specification [7], which augments the recently standardized FORTRAN90. HPF offers the ability to distribute vectors and matrices across the processes on a machine via compiler directives and declaration statements. The DP rule of thumb is that the owner

computes the portion of the matrix/vector that it controls. The DP programming model works extremely well if there is no load balancing or data dependencies across processes (Finite Element or grid calculations, Fourier transforms, etc.). In a pure SIMD program, load imbalance causes many processors to be idle at either the beginning or end of a series of parallel tasks. HPF offers extrinsic procedures or routines to handle access to message passing facilities that can handle data dependency aspects. Most computational chemistry algorithms have either very irregular data structures or access a regular data structure in an irregular fashion and are thus not suited for pure DP programming models. For example, in the formation of the closed shell Fock matrix, a two electron integral contributes to six independent Fock matrix elements all of which may not be accessible in a given distribution of the Fock matrix. The HPF draft standard acknowledges the difficulties of irregular data structures or access to data structures, and there are plans for a follow-on language specification that will address irregularity issues. There is great potential for such standard languages or extensions to existing languages once this issue is addressed.

The message passing programming model is probably the most widely used programming model on distributed memory architectures. The basic principles are that every process has a local memory addressable only by that specific processor, and it can only access non-local information by passing a message to another processor that has other needed data. Tasks and the required data structures can be farmed out to various processors via these messages and the parallel calculation performed. Messages can usually be sent synchronously or asynchronously, but this is system dependent. Synchronous messages require the cooperation of both processors. Asynchronous message passing can be useful but requires buffer space for messages to make them effective. The ability to pass asynchronous messages offers a more robust programming environment because it does not require a sequential bottleneck of the sending and receiving processors. A message passing program makes the speed and latency of the underlying hardware an important aspect of the algorithm and software design. If the application requires only short messages, then the latency or overhead for sending each message is important (e.g., molecular dynamics simulations). If the application requires a few messages that are relatively large (~ 1 Mbyte), then the speed or bandwidth of the underlying communication network is important (e.g., a replicated data Hartree Fock code). Many computational chemistry applications contain both aspects. The general course grained aspect of integral generation and Hartree Fock procedures are ideal for this programming model

and the currently available vendor offerings. More complicated and useful chemistry applications have finer granularity and are thus limited by the underlying communication network (e.g., Multiconfiguration Self Consistent Field energy and energy derivative methods).

The final programming model that I would like to discuss is that of distributed data. This programming model borrows strongly from that of the Linda language [8]. The concept is straightforward and is really an extension to any of the above programming models. The concept is similar to that of elaborate memory paging algorithms used in many time sharing computers today. In Linda, the user has the concept of a tuple as an abstract data object that can be stored to and retrieved from "tuple space" as well as user defined functions that can be used to transform the data via an "eval" call. The advantage of having this "secondary" memory storage is that the location of the data and the mechanisms for moving or "paging" the data are removed from the programmer. This gives the look and feel of a segment of "shared" memory that is accessible by all processes. There are potentially performance problems with the way the distributed data is accessed, stored, and transformed, but the distributed data programming model lessens the impact on computational programmers using distributed computing models. The details of the storing and retrieving data from the distributed data space can be implemented in shared memory or message passing allowing applications to be more portable. Harrison has developed and successfully demonstrated a distributed data model for the Intel Touchstone Delta supercomputer that uses the interrupt driven mechanisms available on that machine [9]. Similar work has been done by Rendell et. al. specifically for the closed shell coupled cluster algorithm [10]. In general, computational chemistry applications will not need the full functionality of a Linda type implementation, but only well defined data types (e.g., for FORTRAN integer, double precision, character, etc.)[9]. This functionality is projected to be the first useful programming model for the next generation of scalable computational chemistry applications.

C. Portability and Resource Utilization.

Portability is an issue that has plagued FORTRAN computational chemistry applications for many years. Unlike many modern languages, there is no language specific mechanism for isolating machine dependent code in FORTRAN. There are several different approaches that can be used to get around this problem. Many free

and some commercial systems for FORTRAN code maintenance exist and are widely used by the computational chemistry community. Writing only FORTRAN77 or FORTRAN90 is not a viable option because many chemistry applications interface with the system environment to get timing information, use special disk I/O routines, check system runtime characteristics, etc. The various complexities of the above mentioned programming models will also add to the overall complexity of a "portable" application. A review of the computational chemistry literature over the last few years, shows that chemists will use as many of the theoretical models as is feasible for the solution of a given chemistry problem. This aspect alone will compel the integration of computational chemistry applications into a suite of functionality with a common "user interface." The development of this interface is a research topic in its own right and beyond the scope of this article. It is imperative that the core functionality application suite use modularity and more commercial-style software practices (e.g., long-term use or reuse of software) to maintain the integrity of applications across the various platforms, from the workstation to the high performance computing supercomputer.

Resource utilization is an issue that will have to be addressed by both the user and vendor communities. Users are accustomed to sharing workstations and traditional supercomputers based on a round robin or time-slice multiuser scheduling system. On current MPP offerings this is simply not feasible. The scheduling of resource utilization is a research topic in the computer science field. Users will have to become accustomed to more batch utilization and space sharing of the resources. The disk I/O capacity is usually the limiting factor. For example, on the Intel Touchstone Delta supercomputer there are 512 compute nodes with 16 Mbytes of memory each, for a total of 8 Gbytes of memory. The aggregate I/O rate on that machine under the normal operating system is less than 12 Mbytes/sec. This means to roll out a job using all nodes would require at least 11 minutes. The next generation MPPs will not be much better because the I/O subsystem is the least improved component. Again this is an active computer science research topic. The computational chemistry community cannot and will not wait for these problems to be solved. This means that the application developers must be more aggressive in checkpointing their own algorithms with only the requisite restart data being written to disk.

III. Review of Chemistry Applications.

HPC in Chemistry and MPP Computers

The use of parallel applications in chemistry is not a new idea. Reports of using available parallelism on minicomputers date back to the early and mid-1980s [11-15]. There are a few current research efforts around the world specifically targeted at the development of software on current and future generation MPPs. These include mostly academic and national laboratory efforts, although a few vendors have openly stated that they have started a port of the Gaussian software in collaboration with Gaussian, Inc. The academic research efforts include GAMESS-USA from Mark Gordon's group at Iowa State University, DISCO from Jan Almlöf and coworkers at University of Minnesota, Columbus from Hans Lischka at the University of Vienna (in collaboration with the Ohio State University and Argonne National Laboratory), various applications from Bill Goddard at Caltech and a new initiative at the San Diego Supercomputer Center initiated by Peter Taylor. The national laboratory efforts include various chemistry applications at Argonne National Laboratory, Pacific Northwest Laboratory, Sandia National Laboratories, the National Institutes of Health, and the SERC Daresbury Laboratory in England. This list does not encompass all researchers working on or planning MPP application development but does include the research groups that have significant resources for their efforts. I have also limited the list to efforts I have some direct knowledge of the software being developed.

Molecular dynamics (MD) applications have been using MPP systems from the beginning of the development of these parallel computing systems. Various algorithms have been developed: systolic loop, linked cells, and replicated data systems[16,17]. Due to the relatively small memory requirements of most MD applications, the replicated data algorithms have been most widely used (c.f., Ref. 16.). The advent of larger memory capacities and faster communication subsystems with decreased latency characteristics on next generation MPPs will keep the replicated data algorithms in wide use over the next few years. The same technological advances will also allow the refinement and improved performance of the other parallel algorithms that have been implemented. The replicated data algorithms do not scale to thousands of nodes and thus MD applications will need further development once the scientific demands of the applications increase. The MD applications are approaching routine utilization of current MPP systems and are generating results that require the increased computational resources available at various MPP sites.

Traditional *ab initio* software applications have had much less routine development on MPP machines. Simple replicated data algorithms are useful but the memory size on most MPP machines ranges from 16 to 32 Mbytes per node, which is a severe limitation. For example, in a replicated data Hartree Fock code, the entire density matrix and Fock matrix would be stored on each node. This would allow each integral to be partially summed with the appropriate density matrix element into the requisite six Fock matrix elements. Once all integrals are computed, the "partial" Fock matrices would then be globally combined into the full Fock matrix and diagonalized. After a new density matrix was formed and broadcast to each process the iterative processor would continue. This scheme replicates two N^2 matrices and is thus parallel over the $N^4/8$ integral generation work. However, on a machine that has 16 Mbytes of memory there could be no more than two 1000x1000 square matrices in core memory. In reality the operating system takes up some memory, the software uses memory and thus there is room for much fewer matrix elements. The thrust of most efforts is to go beyond this limitation by distributing the data structures of a given calculation to allow the problem to scale to the full memory of the machine and not the limitations imposed by the per node memory. From this simple example, it is hopefully clear that the transition to MPP software development also requires more up front analysis of the algorithmic designs.

In the above Hartree Fock application domain the type of distribution has different computation, communication, and memory tradeoffs. Colvin and coworkers [18] have implemented an application that distributes both the Fock and density matrices that requires $N^4/2$ instead of the typical $N^4/8$ integral generation work. This distribution scheme also can suffer from load imbalance in the parallel integral computation. In this distribution scheme a lack of memory has been traded for more integral computation. The most promising technique to date, is that of Foster[19] who has developed a distribution scheme that stores O(N) Fock and density matrix elements on each processor but it is based on the following assumptions. The integrals all cost the same number of flops and are computed one at a time (i.e., not in groups over shells). Unfortunately, these assumptions are not in line with modern efficient and parallel integral algorithms. In this distribution scheme communication is increased to allow for a fixed computational cost of the integral evaluation. However, Foster's efforts do show promise once the shell grouping of integrals is addressed.

Even with the above mentioned problems, there are several useful *ab initio* applications in use on MPP systems. GAMES-USA, GAMES-UK and Columbus are now in production use by their respective groups on Intel MIMD machines with specific functionality (i.e., not all) parallelized [20-23]. Harrison and Stahlberg have implemented an object oriented style full CI code on the Intel Touchstone Delta supercomputer that sustains 4 GFLOPS (20% of peak performance) [24]. The distributed data models used by Harrison and Stahlberg were implemented by Rendell and coworkers in his coupled cluster singles and doubles code [10]. Feyereisen and coworkers [25] have implemented a master/slave replicated data version of Almlöf and coworkers DISCO SCF/MP2 program on the Intel Touchstone Delta supercomputer using a message passing library, TCGMSG, written by Harrison[26]. DISCO has also been ported to workstation clusters using TCGMSG, LINDA, PVM, and EXPRESS[27]. For all these codes to become production quality the above mentioned scaling issues (with respect to problem size and machine size) need to be addressed. These developments show great promise and are likely to be the foundation of *ab initio* applications developed in the future. I would recommend the reading some of the specific references for more details of the implementation and performance parameters of each of these codes on current MPP systems.

IV. Conclusions.

What is needed to bring MPP into routine computational chemistry production? This can summed up in one word, *software*. This means both the software to facilitate application developments and the application software itself. It is unlikely that the large industrial user community will convert to MPP utilization until the commercial software base or high quality, high performance software from academic and national laboratory efforts is available for use on these high performance supercomputers or MPPs. The commercial software developers probably will not make the effort until a more significant market exists. This will require the computing environments on the MPP machines to be much more robust, and a pool of experienced development personnel must become available to commercial software companies from the academic areas.

The development of new algorithms and associated software must keep abreast of changes in the computing environments available. In this article I have attempted to point out a subset of the issues that need to be considered and addressed by both the computational science and computer science communities. MPP development has coerced a coupling of these disciplines and this coupling provides a new opportunity to guide the development to the solutions of some of these issues. No one discipline can solve all the problems that exist or that will be uncovered over the next decade. I have also tried to point out some of the problems associated with computational chemistry applications with respect to MPP development and production use. The important factor here is doing the chemistry required to solve problems posed to us either in basic or applied research. MPPs are an obvious tool to use due to the computational requirements of the theoretical models used today and that will be used in the future. I have also outlined a subset of the research efforts currently in place. Beyond the required computational science training and education, it is important that software development groups get access to the latest technology to develop and refine research ideas. Moreover, collaborative efforts with computer science and other computational science efforts are essential for the development of MPP software. In regards to the question I have posed in the title, is the transition to MPP high performance computing a simple one? I hope so.

V. Acknowledgments.

This work was performed under the auspices of the Office of Environmental Restoration and Waste Management, U. S. Department of Energy under the Environmental and Molecular Sciences Laboratory Project D-384 and under contract DE-AC06-76RLO 1830 with Battelle Memorial Institute, which operates the Pacific Northwest Laboratory. I would also like to thank R. J. Harrison, M. F. Guest, A. P. Rendell, R. L. Stevens, and R. J. Littlefield for valuable insight and review of the manuscript. I gratefully acknowledge the support of the Sanibel organizing committee for their partial support in attending the 1993 Sanibel Symposium. Some of the research efforts described in this article use in part the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Pacific Northwest Laboratory

References:

1. R. S. Pressman, "Software Engineering a Practitioner's Approach", Third Edition, McGraw Hill, Inc. (1992).
2. D. F. Feller private communication.
3. K. M. Chandy and J. Misra, "Parallel Program Design", Addison-Wesley Publishing Company, Inc. (1988).
4. D. W. Heermann and A. N. Burkitt, "Parallel Algorithms in Computational Science", Springer-Verlag, (1991).
5. K. Hwang and F. A. Briggs, "Computer Architecture and Parallel Processing", McGraw-Hill, Inc. (1984).
6. G. R. Andrews, "Concurrent Programming Principles and Practice", The Benjamin/Cummings Publishing Company, Inc. (1991).
7. The HPF draft standard is available from anonymous file transfer from titan.rice.cs.edu. More information can be obtained by sending mail to hpff-info@cs.rice.edu.
8. N. Carriero and D. Gelernter, Communications of the ACM, **32**, 444 (1989).
9. R. J. Harrison, Theo. Chim. Acta, **84**, 363, (1993).
10. A. P. Rendell, M. F. Guest, and R. A. Kendall, "A Distributed Data Parallel Coupled Cluster Algorithm: Application to the 2-Hydroxypyridine/2-Pyridone Tautomerism" submitted to J. Comp. Chem.
11. R. Seeger, J. Comp. Chem., **2**, 168 (1981).
12. R. A. Bair and T. H. Dunning, Jr. J. Comp. Chem. **5**, 44, (1984).
13. E. Clementi, G. Corongiu, J. H. Detrich, H. Khanmohammadbaigi, S. Chin, L. Domingo, A. Laaksonen, and H. L. Nguyen, "Proc. Int. Symp. Struct. Dyn. Membr., Nucleic Acids Proteins", Adenine Press, 49-86, (1985).
14. P. O. Lowdin, Lect. Notes Chem. Supercomput. Simul. Chemis. **44**, 1-48 and 244-245, (1986).
15. R. A. Whiteside, J. S. Binkley, M. E. Colvin, and H. F. Schaefer, III, J. Chem. Phys. **86**, 2185, (1987).
16. W. Smith, Comp. Phys. Commun. 62, 229, (1992) and references therein.
17. Proceedings of "A Workshop on High Performance Computing and Grand Challenges in Structural Biology" held at Florida State University, January 24-27, (1992).
18. M. E. Colvin, C. L. Janssen, R. A. Whiteside, C. H. Tong, Theo. Chim. Acta., **84**, 301, (1993).
19. I. Foster private communication.
20. T. Windus and M. Gordon private communication.
21. M. F. Guest, private communication and "GAMESS-UK Users Guide and Reference Manual", Revision A.1, SERC Daresbury Laboratory, (1990).
22. R. J. Harrison and R. A. Kendall, Theo. Chim. Acta., **79**, 337 (1991)
23. T. Kovar and H. Lischka private communication
24. R. H. Harrison and E. A. Stahlberg, "Massively Parallel Full Configuration Interaction. Benchmark Electronic Structure Calculations on the Intel Touchstone Delta.", submitted to Journal of Parallel and Distributed Computing.
25. M. W. Feyereisen and R. A. Kendall, Theo. Chem. Acta, **84**, 289, (1993) and references therein.

26. R. J. Harrison, Int. J. Quant. Chem. **40**, 847 (1991).
27. M. W. Feyereisen, R. A. Kendall, J. Nichols, D. Dame, J. T. Golab, "An Implementation of the Direct SCF and RPA Methods on Loosely Coupled Networks of Workstations" accepted J. Comp. Chem. (1993).

Figure 1. The rate of change of raw compute power available from workstations[2].

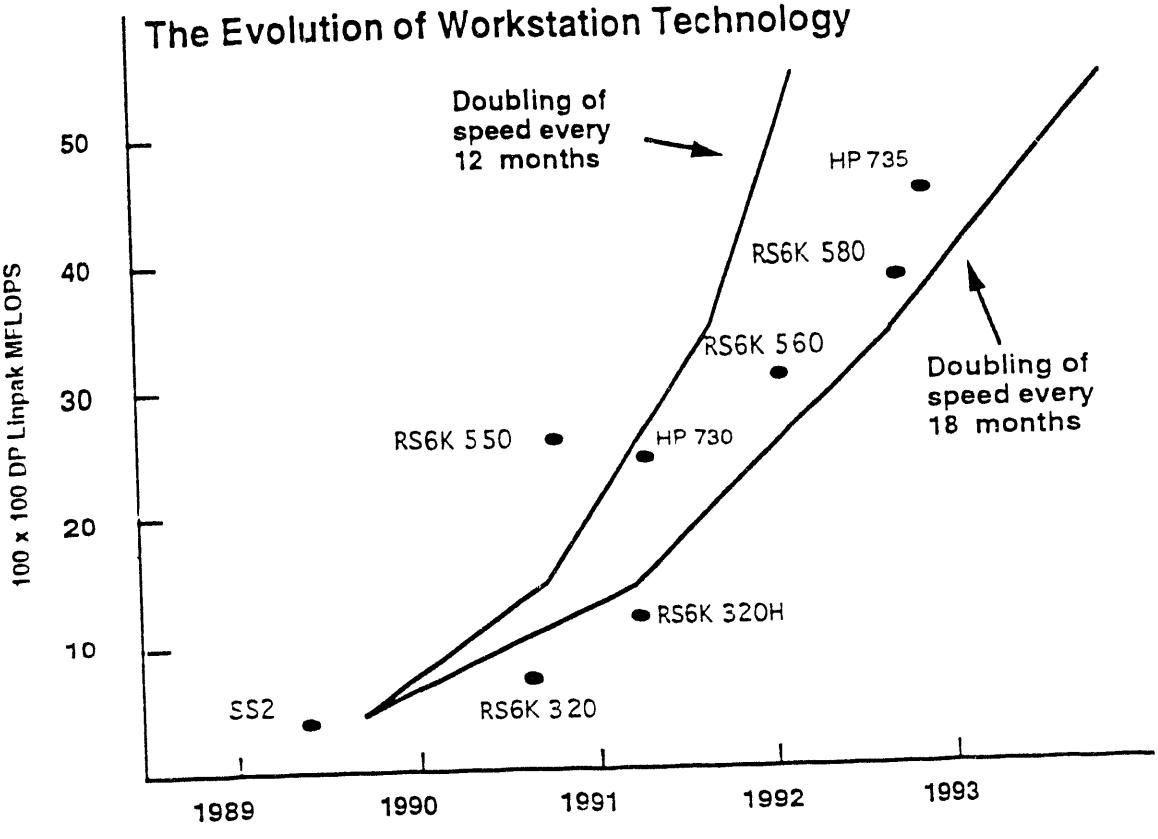

The Evolution of Workstation Technology
Doubling of speed every 12 months
Doubling of speed every 18 months
100 x 100 DP Linpak MFLOPS
50
40
30
20
10
1989
1990
1991
1992
1993
HP 735
RS6K 580
RS6K 560
RS6K 550
HP 730
RS6K 320H
SS2
RS6K 320

Figure 2. Hardware Life Cycle ("Usefulness over time").

Hardware

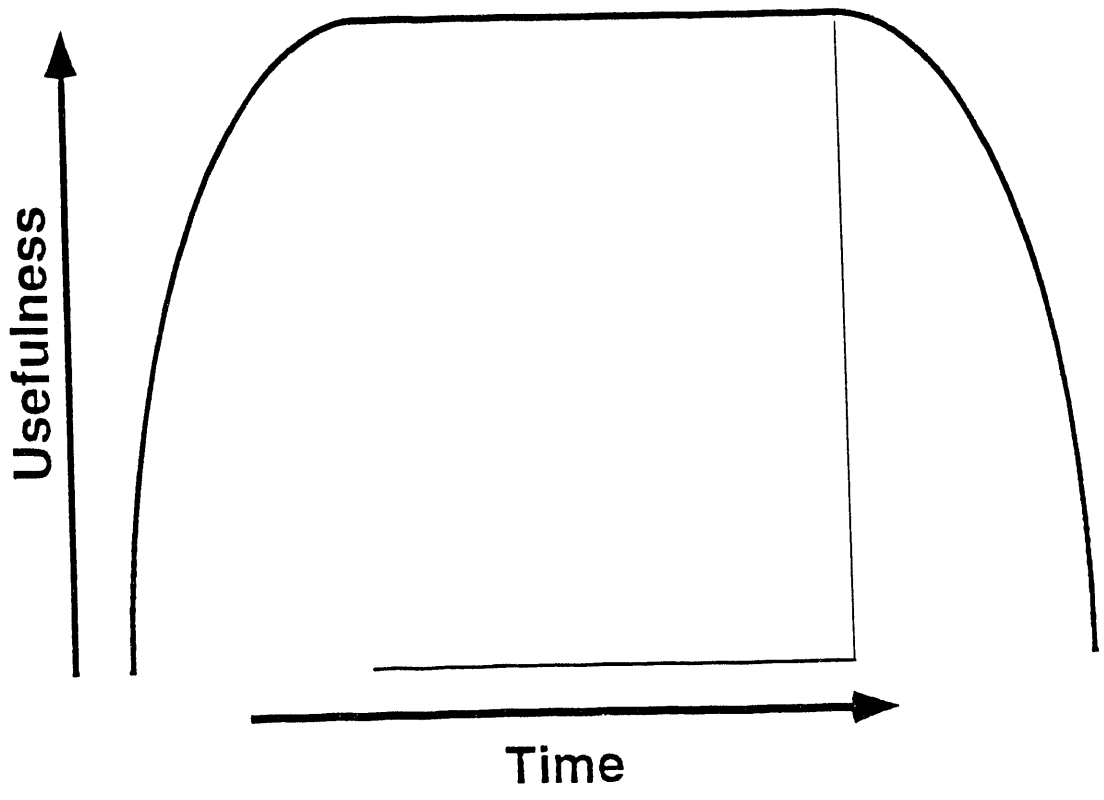
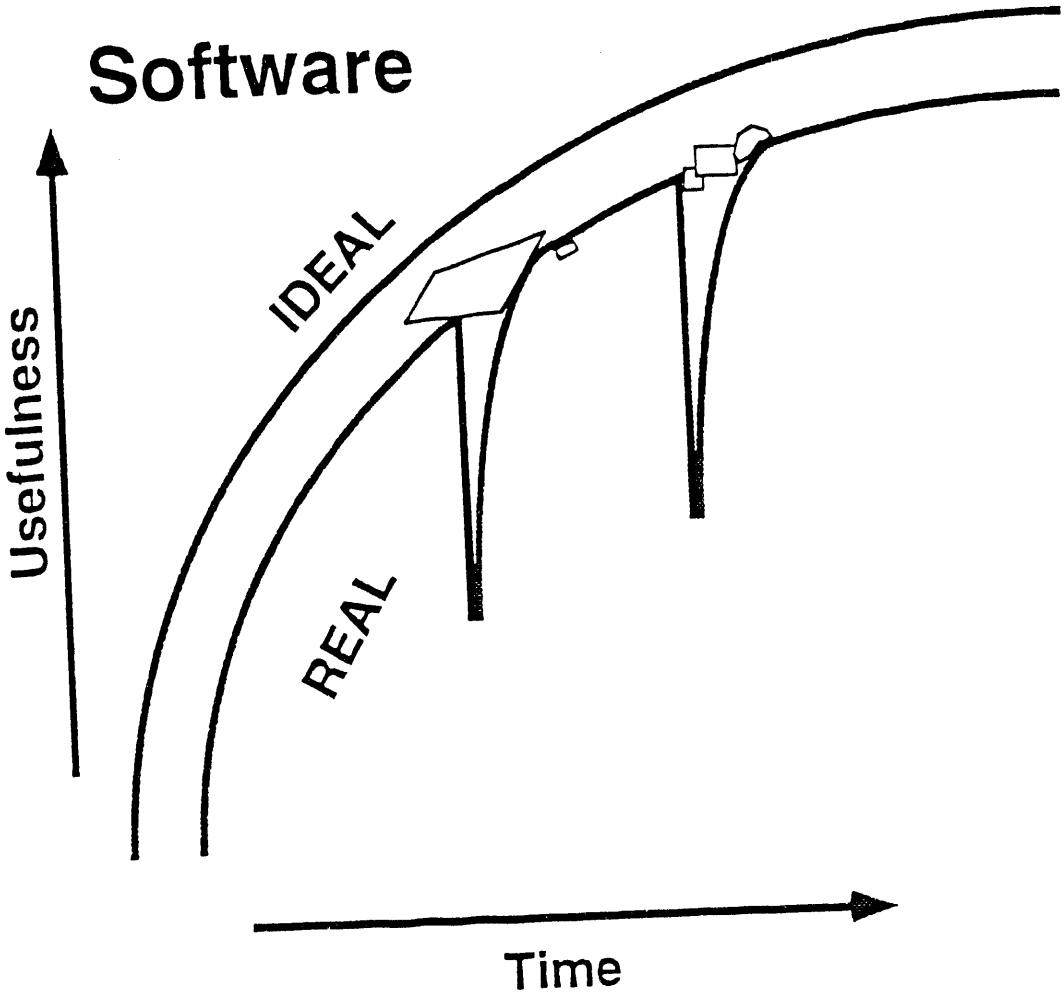


Figure 3. Software Life Cycle ("Usefulness over time").


Software
IDEAL
REAL
Usefulness
Time

DATE
FILMED
2/3/94
END