Universal High Performance Computing — We Have Just Begun

Jerome A. Feldman

April, 1994

Universality has been the key to the overwhelming success of computing in the commercial, intellectual and military sectors of American life. The same hardware and often the same software can be applied to a wide range of problems. This has led to enormous economies of scale in manufacturing, training, and deployment. At present, high performance computing is entirely different. Although there have been some impressive successes and there are continuing advances in parallel scientific computing, only a small fraction of the nation's intensive computing problems are currently amenable to high performance parallel computing. Each of these applications has required a very large and expensive development effort. Because of the high development costs, HPC has primarily caught on in very cost-insensitive areas. This was not problematic when the corresponding hardware was prohibitively expensive. With the availability of cost-efficient parallel hardware, however, it is now widely understood that a breakthrough toward universal parallel computing is needed for HPC to meet its promise in National Challenge applications and to become a major commercial factor.

A prerequisite for Universal High Performance Computing (UHPC) is convergence in the hardware area. In serial computing, a common basic model called the Von Neumann model has allowed hardware and software developments to proceed with the assurance that basic compatibility would be maintained. Until recently, parallel computing has been plagued by a plethora of very different computation models that made it impractical to develop a broad software base. Fortunately, there now seems to be widespread agreement that HPC hardware should follow a model based on general purpose processors in a distributed memory system. This model covers everything from networks of workstations (NOW) to commercial parallel systems being produced by all the major manufacturers. The reason for this convergence is economic and is unlikely to change: the development cost for a new high-performance microprocessor is extremely high, whereas the market for HPC is limited compared to the market for personal computers and workstations. Building special processors for HPC is also very expensive and does not provide the same performance advantages over off-the-shelf components that it did only a few years ago. It is this convergence on a common hardware substrate that now offers the prospect for a software base for Universal High Performance Computing (UHPC).

Current software for HPC is developed for only a few areas. Since programming languages are the organizing core of applications software, this article will focus on programming languages and their role in various HPC domains. The most advanced of these is scientific computation where various FORTRAN systems have yielded very impressive performance on Grand Challenge and related numerical calculations. Even in this domain, there is now widespread understanding that new languages and tools are needed to address problems with the irregular and changing structures that arise in many important applications. The current commercial approach to this problem is to revert to lower level programming, usually in a variant of C or C++ with primitive mechanisms for communication and coordination. While any program can be written at a low level, this approach has many serious limitations. Low level programs are much harder to write, test, maintain and verify and are very much harder to reuse or to move to another platform.

Although many ideas should be explored, we believe that the best chance for achieving a breakthrough in UHPC is by exploiting object-oriented languages. Object-oriented programming is based on the encapsulation of functionality into reusable components. The goal of an object-oriented language is to provide a powerful framework for writing and using these components. Much effort will need to be devoted to the construction of large component libraries that encapsulate efficient parallel algorithms for architectures with different characteristics. Libraries can be designed so that underlying components optimized for different parallel architectures can be transparently used when an application is moved from one HPC machine to another. The application programmer primarily builds applications from such pre-existing library components, allowing the resulting applications to run efficiently on a variety of architectures. Most current work on HPC is devoted to the development of automatically parallelizing compilers for languages like FORTRAN. Such compilers are very machine specific, complex to build, and limited to specific computational structures like arrays because they must anticipate optimizations for the full range of programs they must operate on. In contrast, the object-oriented component approach allows for incremental development, complete flexibility of domain, and easy tuning for different architectures. Critically, the role of reusable libraries will be much more important in parralel than in serial computing because it is so much harder to construct each module.

Existing research provides a sound basis for new the agenda. Briefly, there have been four distinct lines of work on parallel programming and the project can draw on all four. Scientific parallel computing has mainly used various FORTRAN dialects and emphasized compile time analysis of simple structures, usually fixed arrays. Many valuable lessons have been learned, but no one knows how to extend this compiler-oriented approach to the full range of data structures and algorithms needed for UHPC. Most tasks have run-time data dependencies that make pure compile time analysis inherently infeasible. But we can exploit many of the analysis techniques in our compilers for more general languages.

Another approach that is used in limited situations is to exploit functional programming or data parallel techniques. This is elegant and efficient where it works, but again has inherent limitations for UHPC. We can use ideas from this approach in the design of the high-level constructs. In fact, we should explicitly encourage users to think in terms of data-parallel operations on arbitrary data structures.

A third approach is to overcome the limitations of restricted paradigms by employing extensions to universal serial languages, usually C or C++. This is an attractive short run solution and will continue to be popular for some years. Most of the work done here is directly relevant to more ambitious efforts and will continue to contribute. The problem is that C and thus C++ is a low level, weakly typed language developed long before parallel computing became important. We now know much more about language design and the properties that support UHPC.

Finally, there is an existing body of work in Object-Oriented programming languages (OOPS). For historical reasons the vast majority of work on parallel OOPS has focussed on loosely-coupled distributed systems rather than HPC platforms. This plus a minimalist concern for making the parallel version as much as possible like the serial one has limited the usability of such systems for UHPC purposes. As far as we know, pSather [24], is the first case in which the serial and parallel versions co-evolved, but there will be many others. This can lead to systems which are much better suited to the needs of UHPC. There are also a number of combinations of these techniques. The best known of these is Linda, which adds a simple tuple-space mechanism for communication to FORTRAN or C.

Most concurrent object-oriented languages try to unify the notion of objects and processes in order to provide protection of internal object consistency in the presence of concurrent requests by essentially following the "actors" model [2]. Important examples based on the active object approach with real implementations on parallel machines include ABCL/1 [43], POOL[6]. Recent overviews over activities in the field may be found in [1] and [3].

This approach is attractive for distributed and concurrent systems: Such systems are often designed to support multiple parallel tasks which *compete* for shared resources. *Fairness* and *mutual protection* are central issues in these systems and consume a large part of the available computing power. The main goal for parallel systems, however, is to make many resources (processors, I/O-devices, etc.) *cooperate* on a single large task. Although the interaction and synchronization of multiple flows of control are common issues in both fields, the trade-offs in parallel systems are fundamentally different from those in distributed and concurrent systems.

The "actor" approach does not fit well with our goals. "Actor" language semantics are fundamentally different from the semantics of sequential object-oriented languages. Sequential objects are passive entities activated by method calls. It is, thus, difficult to incorporate sequential classes into such programs. Parallel algorithms often require that more than one thread accesses the same object at the same time, possibly requiring partial internal replication of the object and caching of accesses. Such structures are not always easy to express in concurrent object-oriented languages. As was pointed out by Chien [9], many concurrent object-oriented languages serialize hierarchical abstractions. Synchronization on the single objects and, thus, coupling the granularity of the parallel computation to the decomposition of data into objects is still a performance problem despite recent improvements of the implementation technology [40].

The third approach is to treat object-orientation and parallelism as orthogonal language concepts. Such languages generally enrich a sequential object-oriented programming language with explicit parallelism constructs like thread forking and dedicated synchronization objects. Advantages of this approach include that sequential code can be reused without change as a part of larger parallel computations, that granularity and data structure issues can be treated as separate problems, and that complex distributed data structures can still be hidden from the user by employing the flexible abstraction mechanisms of modern object-oriented languages. A possible disadvantage is that the integrity of object contents is not automatically guaranteed in case of accesses by concurrent threads. Synchronization is explicit.

Threads and and fine-grained synchronization constructs are too low level abstractions for massively parallel computers. We may use the encapsulation and abstraction capabilities like those of pSather to build higher-level abstractions like distributed sets, workbags, simulation frameworks, etc. Among these higher level constructions one stands out as particularly important: "Apply the same operations to a whole collection of objects, where the computation is executed locally with the data." Note that this is a more general notion of *data-parallel* computation than the synchronous single instruction multiple data model (SIMD) proposed by others (e.g. [19]). Yet, in our opinion it embodies the essential concepts of data-parallel computation. Single program multiple data (SPMD) characterizes this kind of computation more appropriately than SIMD.

The traditional array-based data-parallel model[19] has been integrated into a number of different programming languages (e.g. FORTRAN 90[27], C* [18], etc.). This includes extensions to object-oriented languages (pC++ [22]). Recognizing the performance penalty of a completely synchronous data-parallel constructs on modern MIMD machines, asynchronous (SPMD) FORALL-constructs have been introduced into more recent languages[41, 17].

The design of class libraries is fundamental to the object-oriented approach. The object-oriented languages Smalltalk [15], Objective-C [11], and Eiffel [28] each provided class libraries as a part of the system. While they each introduced some interesting concepts, they are not very extensive. There have been several class libraries developed for C++, (eg. libg++ [21] and ET++ [42]). These libraries primarily provide simple container classes and user interface classes. The most developed algorithmic class library for C++ of which we are aware is LEDA [26] and [32]. LEDA is concerned with theoretical performance guarantees and often chooses algorithms with a provable worst case performance over algorithms which are faster in practice.

As yet, general purpose parallel libraries do not exist. Special purpose parallel numerical libraries exist (eg. LAPACK) but primarily for FORTRAN. These are almost entirely restricted to array and vector operations, however. A major aspect of the UHPC agenda will need to be in the area of library development. Although this essay has focussed on programming language design, all of the concomitant systems also will need much greater research and development effort. Compilers, debuggers, performance monitoring and hardware and operating systems support will all play a crucial role in UHPC.

The U.S. is facing many critical problems over the next decade and Computer Science is being asked to play a central role in addressing them. There is enormous pressure to make every Computer Science project serve an immediate national need. But the most important practical advances in computing have come from GENERAL advances in hardware, algorithms and software. The nation must be careful not to kill the goose that has been delivering the golden eggs of economic and social progress. The National Science Foundation must lead in the effort to preserve the core of generality-oriented strategic computer science and engineering research. The quest for Universal High Performance Computing should be a central part of this agenda.

References

[1] Communications of the ACM: Special issue on concurrent object-oriented programming, September 1993.

[2] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* The MIT Press, Cambridge Massachusetts, 1986.

[3] G. Agha, P. Wegner, and A. Yonezawa, editors. *Research Directions in Object-Based Concurrency.* The MIT Press, Cambridge Massachusetts, 1993.

[4] E. Albert, J. D. Lukas, and G. L. Steele Jr. Data Parallel Computers and the FORALL Statement. *Journal of Parallel and Distributed Computing*, 13:185–192, 1991.

[5] F. Alizadeh, R.M. Karp, D. Weisser, G. Zweig, 'Physical Mapping of Chromosomes Using Unique Probes', Proc. Fifth ACM/SIAM Symp. on Discrete Algorithms (January, 1994)

[6] P. America. *Issues in the Design of a Parallel Object-Oriented Language.* Philips Research Laboratories, Eindhoven and University of Amsterdam, March 1 1989. Part of POOL2/PTC Distribution Package.

[7] D. Caromel. Toward a method of object-oriented concurrent programming. *Communications of the ACM*, 36(9):90–102, September 1993.

[8] R. Chandra, A. Gupta, and J. L. Hennessy. Integrating Concurrency and Data Abstraction in a Parallel Programming Language. Technical Report CSL-TR-92-511, Computer Systems Laboratory, Stanford University, February 1992.

[9] A. A. Chien. *Concurrent Aggregates (CA): An Object-Oriented Language for Fine-Grained Message-Passing Machines.* PhD thesis, MIT, July 1990. Also available as: MIT Artificial Intelligence Laboratory, Technical Report 1248.

[10] T. M. Corporation. *The Connection Machine CM-5 Technical Summary.* Thinking Machines Corporation, Cambridge Massachusetts, October 1991.

[11] B. Cox, "Object-Oriented Programming - An Evolutionary Approach", Addison-Wesley, Reading, Massachusetts. (1987)

[12] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurty, S. Lumetta, and K. Yelick. Parallel programming in Split-C. In *Proceedings of Supercomputing'93*, 1993.

[13] D. Culler, R.M. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian and T. von Eiken,' LogP: Towards a Realistic Model of Parallel Computation', Proc. ACM Symp. on Principles and Practices of Parallel Programming. (May, 1993)

[14] J. Feldman, C. C. Lim, and F. Mazzanti, "pSather monitors: Design, Tutorial, Rationale and Implementation", International Computer Science Institute Technical Report TR-91-031, (September 1991).

[15] A. Goldberg and D. Robson, "Smalltalk-80, The Language and its Implementation", Addison-Wesley, Reading, Massachusetts. (1983)

[16] A. S. Grimshaw. An introduction to parallel object-oriented programming with Mentat. Technical Report TR-91-07, University of Virginia, Charlottesville, Va., 1991.

[17] S. U. Hänßgen, E. A. Heinz, P. Lukowitz, M. Philippsen, and W. F. Tichy. The Modula-2* environment for parallel programming. In *Proceedings of the Working Conference on Massively Parallel Programming Models*, Berlin, Germany.

[18] P. J. Hatcher, A. J. Lapadula, R. R. Jones, M. J. Quinn, and R. J. Anderson. A Production-Quality C* Compiler for a Hypercube Multicomputer. Technical Report TR 90-80-3, Oregon State University, Computer Science Department, 1990.

[19] W. D. Hillis and G. L. Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, 1986.

[20] R.M. Karp, 'Mapping the Genome: Some Combinatorial Problems Arising in Molecular Biology', Proc. ACM Symp. on Theory of Computing. (May, 1993)

[21] D. Lea, "libg++, The GNU C++ Library", in "USENIX Proceedings C++ Conference", Denver, Colorado, 17-21, Berkeley, California, 243-256 (1988).

[22] J. K. Lee and D. Gannon. Object Oriented Parallel Programming Experiments and Results. In *Proceedings of Supercomputing '91*. IEEE Computer Society Press and ACM SIGARCH, November 1991.

[23] C.-C. Lim. *A Parallel Object-Oriented System for Realizing Reusable and Efficient Data Abstractions*. PhD thesis, University of California at Berkeley, 1993.

[24] C.-C. Lim, J. A. Feldman, and S. Murer. Unifying control- and data-parallelism in an object-oriented language. In *Proceedings of the 1993 Joint Symposium on Parallel Processing*, Tokyo, Japan, May 17 - May 19 1993.

[25] C. C. Lim and A. Stolcke, "Sather Language Design and Performance Evaluation," Technical Report TR-91-034, International Computer Science Institute, Berkeley, Ca., May 1991.

[26] K. Mehlhorn, S. Naher, "LEDA, a Library of Efficient Data Types and Algorithms", TR A 04/89, FB10, Universitat des Saarlandes, Saarbrucken, 1989.

[27] M. Metcalf and J. Reid. *Fortran 90 Explained*. Oxford University Press, 1992.

[28] B. Meyer. *Object-oriented Software Construction*. Prentice Hall, New York, 1988.

[29] E. Moss and W. Kohler. Concurrency features for the Trellis/Owl language. In *Proceedings of ECOOP'87*, 1987.

[30] S. Murer, J. Feldman, and C. C. Lim, "pSather: Layered Extensions to an Object-Oriented Language for Efficient Parallel Computation", Technical Report TR-93-028, International Computer Science Institute, Berkeley, Ca., June 1993.

[31] S. Murer, S. M. Omohundro, and C. Szyperski, "Sather Iters: Object-oriented Iteration Abstraction", submitted to ACM Letters on Programming Languages and Systems, (1993).

[32] S. Naher, "LEDA2.0 User Manual", Technischer Bericht A 17/90, Fachbereich Informatik, Universitat des Saarlandes (1990).

[33] S. M. Omohundro, "The Differences Between Sather and Eiffel", *Eiffel Outlook*, **1:1**, (April 1991).

[34] S. M. Omohundro "Sather's Design", *Eiffel Outlook*, **1:3**, (June 1991).

[35] S. M. Omohundro. The Sather 1.0 Specification. Technical report, International Computer Science Institute, Berkeley, Ca., 1992 (in preparation).

[36] S. M. Omohundro and C. C. Lim, "The Sather Language and Libraries," Technical report TR-92-017, International Computer Science Institute, Berkeley, Ca., (1991).

[37] S. M. Omohundro, "Sather Provides Nonproprietary Access to Object-Oriented Programming", *Computers in Physics*, Vol.6, No. 5, p. 444-449, (September 1992).

[38] S. M. Omohundro, "The Sather Programming Language" *Dr. Dobb's Journal*, Volume 18, Issue 11, p. 42, (October 1993).

[39] H. W. Schmidt and S. M. Omohundro, "CLOS, Eiffel, and Sather: A Comparison", in Andreas Paepcke, ed. *Object Oriented Programming: The CLOS Perspective*, MIT Press, Boston (1993).

[40] K. Taura, S. Matsuoka, and A. Yonezawa. An efficient implementations scheme of concurrent object oriented languages on stock multicomputers. In *Proceedings of the 4th ACM Symposium on Principles and Practices of Parallel Programming*, pages 218–228, 1993.

[41] H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald. Vienna Fortran - a language specification (version 1.1). Technical Report ICASE Interim Report 21, NASA Langley Research Center, Hampton, Va., 1992.

[42] A. Weinand, E. Gamma, R. Marty, "ET++-an Object-Oriented Application Framework in C++" in "OOPSLA 88 Conference Proceedings", September 26-30, San Diego, CA, published as Special Issue of SIGPLAN Notices, Vol. 10, No. 2, p. 63-87 (June 1989).

[43] A. Yonezawa (ed). *ABCL – An object-oriented concurrent system.* MIT Press, Cambridge, Massachusetts, 1990.