Parallel Computing in Ada: An Overview and Critique

Marcin Paprzycki & Janusz Zalewski
Dept. of Science & Mathematics
University of Texas–PermianBasin
Odessa, TX 79762-0001
(915)552-2258
paprzycki_m@utpb.edu, zalewski_j@utpb.edu

Abstract. The aim of this paper is to present the basic issues related to the use of Ada in parallel computing. It is argued that there exists a gap between the most popular area of parallel computing research, scientific computing, and the interests of Ada community. A number of new research directions and curricular changes are proposed.
Keywords: Ada, parallel computing, efficiency, performance, experimental studies

1 Introduction

Ada'83 [1] was created as a programming language with a particular purpose. This purpose was very well summarized by Kok [13]:

"The language Ada was primarily designed for the production of large portions of readable, modular, portable, and maintainable software for real-time applications" (p. 100)

Bearing this in mind, one can expect that there exist a number of constructs in the language that were designed to support these goals. When Ada was being designed, only the first glimpses of realistic parallel computing appeared on the horizon and parallel computers (as we know them now) existed only in minds of their future creators and in very few university and government laboratories. This explains in part why no broad range support for parallel computing was introduced into the language definition. It should be pointed out, however, that since the language was supposed to address the design of software for real-time systems some of the constructs in the language can support parallel computing. These tools are the concept of a task (allowing the separation of the threads of execution into independent units) and the mechanism of rendezvous that supports the communication of these independently executing units and can also work as a very powerful synchronization mechanism ([13], p. 101)

Since the original Ada 83 [1] was introduced, a large number of changes in computing, in general, have occurred. The computer architectures have evolved in an extremely rapid fashion, diverging into vector computers, SIMD (single instruction multiple data) parallel computers, as well as two models of MIMD (multiple instruction multiple data) computers: shared memory and message passing. For these computers, the initial area of applications were large computationally intensive mathematical problems. Even today, computational mathematics (understood very broadly) is the primary area where parallel computers are used (followed by the use of parallel computers in database applications). Interestingly enough, Ada is not the language in which the software for these computationally intensive mathematical problems is written: the two leading languages in this area are Fortran and C – even though their definitions provide absolutely

no support for parallelism.

In this paper we argue that Ada contains all the appropriate tools for it to be used in the development of software for parallel computers. This claim is based on our analysis of the reported research and applications, published in literature before 1996, that is before Ada '95 standard was approved [2]. We also suggest that the new direction of research is necessary if Ada is supposed to gain ground in the area of parallel scientific computing. In Section 2, we summarize the existing research in developing the tools for the support of parallel computing. Section 3 provides a critical analysis of the identified problems and introduces a possible solution for a research program to be pursued. It also discusses the connection between the proposed research and the computer science curricula.

2 Ada Support for Various Models of Parallelism

Our analysis indicates that in the past there was a substantial amount of research completed in support of Ada-based parallel computing. This research has been directed toward various models of parallelism. Starting from basic vector computing, addressing arrays of processors (the SIMD model of computing), the research ventured into very special forms of parallel computing, such as wavefront processing. There was research done in the area of loop parallelism (the basic model for shared memory MIMD paradigm) as well as in hypercube-based parallel computing (message passing MIMD paradigm). There was also some work done in the extremely important area of performance analysis. Obviously, not all the work will (and can) be cited here. We believe though, that what is shown is a good representation of the work done and contains a clear indication of the nature of the problem.

2.1 Vector Computing in Ada

A typical research in the area of Ada-based vector computing is represented by [9, 17, 24]. As a definition of vector computing we can utilize, for instance, the one proposed by Forest and Arnaudo: "A vector machine is (...) a machine able to execute N identical floating operations in less than N times the single operation time" ([9] p. S21). Vector computers are aimed at performing extremely efficiently operations on long vectors. Typical examples of computers in this category are: Cray Y-MP, Convex C-220, IBM V-3900. In each of the papers referenced above, a package (or a collection of packages) was (were) introduced to provide an efficient support for various operations on vectors. Interestingly enough, authors of the first two papers recognize the need for very efficient computing and suggest that the body of the package should be implemented in an assembly language (or interfaced to the Fortran executables that were earlier compiled by an efficient vectorizing compiler) to take full advantage of the underlying architecture. An extension of the initially proposed package(s) into matrix-vector operations was also suggested (in all three papers). It can be observed that some of the influences here come, first of all, from the development of the levels 1 and 2 of BLAS (Basic Linear Algebraic Subroutines), and secondly, from the work that was being done at that time on the definition of Fortran 8X.

2.2 Parallel SIMD Computing

SIMD - single instruction multiple data parallel computing model involves a set (typically an array) of processors where in each time step all of them execute the same instruction (some of the processors may possibly execute no instruction at all) on separate data elements ([22], p. 430). Typical computers representing this approach to parallelism are: ICL DAP, Connection Machines 1 and 2, Maspar 1 and 2. Ada support for this model of parallelism was discussed in [22]. It presents the development of an Ada interface to the *LISP language (a proprietary extension of LISP developed at Thinking Machines) to allow efficient usage of the Connection Machine (CM) array of processors. The interface is designed to provide the user with all the

functionality and special features of Ada (like compile-time type checking and the standard data types) as well as with access to the full *LISP functionality and the processing power of the 65,536 CM processors ([22], p. 431). Interestingly enough, the authors make two observations: first, that Fortran 8X standard has been found useful in designing support for array operations (p. 431), and second, that the performance is extremely important (p. 435).

2.3 Ada in Wavefront Computing

Wavefront and systolic array processors are special classes of VLSI-circuit implementation of parallel processing. Wavefront array processor operations are asynchronous, where data transfer is accomplished by exchanges between pairs of processing elements. Even though the systolic and wavefront models of parallel computing are not considered mainstream, it was shown that a number of algorithms can be efficiently represented in this model of computation. The usage of Ada in wavefront computing is discussed in [8]. It is suggested that typical parallel programming languages (including Occam) cannot provide appropriate description of wavefront array processing, while the proper use of Ada tasks allows the description and simulation of this specific model of parallel computing (p. 267). It is shown how Ada task syntax can properly handle timing, local synchronization and data transfer in a wavefront array of processors.

2.4 Ada Support for Shared Memory MIMD Parallelism

Loop parallelism is the most typical model of parallelism for shared memory MIMD parallel computing. In this model, a number of processing elements is connected to a global shared memory. The most typical examples of computers representing this approach to parallelism are: Sequent, Alliant, BBN Butterfly. Implementation of loop parallelism in Ada is discussed in [9, 11]. The authors of both papers point out that the standard task mechanism is too powerful and too awkward for support of the light-weight loop parallelism. Their objections are additionally grounded in the earlier work [5, 6, 25]. To avoid these problems, Forset and Arnaudo suggest the *cobegin* and *coloop* mechanisms whereas Hind and Schonberg introduce a *beacon task* type which provides support for loop parallelism. Hind and Schonberg study also the efficiency issues of their proposal. It is shown that algorithms utilizing the beacon task outperform algorithms using the standard Ada task. It is also suggested that the beacon task based loop parallelism should be more efficient than similar mechanisms in the parallel extensions to C and Fortran.

2.5 Ada Support for Message Passing MIMD Computing

Message passing computers are the second class of parallel MIMD architectures. Here a parallel computer consists of a number of processing units equipped with local memories that are connected over a network (no global memory is present). Typical examples of such machines are: Intel Hypercube and Paragon, Cray T3D, N-cube. Usage of Ada in such an environment (represented by a 1024 processor N-cube) is studied in [7]. Using a hypercube as an example, the development of a run-time system that would support Ada on any distributed memory multiprocessor is presented. In addition, the detailed discussion related to its efficient implementation on a hypercube is included.

2.6 Efficiency and Performance

In most articles of the above survey the authors suggest that the efficiency of implementation and the performance characteristics of implemented tools are the important issues that need to be addressed. These points were raised primarily as related to the parallel scientific computing. There exists additional interest in performance (outside of the scientific computing area) as represented, for instance, in [10, 18]. Nakao and Yogi [18] investigate the performance of the task based parallel binary tree traversal. They derive the

theoretical performance profile and suggest a method of determining an optimal number of processors to be used. The paper by Goforth et al. [10] presents and discusses a large number of benchmarking data and performance measures for parallel Ada tasking applied to the control of an autonomous telerobotic system. The experiments were performed on a 16-processor Sequent Balance 8000. The results suggest that even though the run-time Ada support on the Sequent was less than satisfactory (p. 54) encouraging performance results have been obtained (p. 46-50).

3 Critical Analysis

The summary of research on parallel computing in Ada presented in Section 2 indicates clearly that a broad range of issues related to the usage of Ada in all of the major models of parallelism have been studied. The results of this research indicate clearly that Ada either has the appropriate tools available to provide direct support for all models of parallel computing (typically through the task and the rendezvous mechanisms) or that appropriate tools can be easily provided by creation of appropriate auxiliary packages and/or interfaces. At the same time, the basic issues still remain open, which prevents the postulate formulated in 1990 by Hunter ([12]):

"The only technically rational way of advancing the art of scientific and engineering programming is to abandon Fortran in favor of modern, block structured language as Algol-68 or Ada"

from having been materialized. There are many possible answers to the question why this is so? In the remaining part of this section we provide support for one of the answers, by pointing out to specific problems in the research summarized above. We also make a number of suggestions that could possibly lead to the fulfillment of Hunter's postulate. These suggestions are directed first, toward the research area and, second, towards the educational curricula.

3.1 Problem Identification

From the definition of the language (in both '83 and '95 versions), it is clear that Ada was **not** created with parallel scientific computing in mind. A quick look into the Ada 95 definition [2] reveals that a number of extensions to the language were proposed to support the Object-Oriented paradigm, whereas no additional support was introduced for parallel computing (as opposed to distributed computing). We have also seen that Ada **has** (or **can easily have**) all the right tools and attributes to support development of parallel scientific computing software. The open question remains: why is Ada almost unused to develop parallel programs? We would like to suggest that the primary problem is related to the efficiency and performance issues.

Parallel computing is being practiced primarily by the people who have one of the following two goals in mind (or sometimes both): to solve **larger** problems (typically in the sense of larger input size), or to solve problems (of a given size) **faster** (typically when someone wants to solve a longer sequence of problems of a given size). In both cases the execution time is crucial. These researchers participate in the *MFlop race* (what program and on what machine can achieve a larger number of floating point operations per second), which leads to the creation of relatively small (compared to the industry standards) software artifacts. They have also developed a number of benchmark suites that are designed to study the performance characteristics of hardware, compilers and software environments. The issues of software portability and reliability, addressed through the creation of software libraries and establishing standardized operations (like the BLAS/LAPACK standards), are considered of slightly lesser importance. It can be conjectured, as it is assumed here, that the science of parallel computing is so new and the parallel hardware and software development tools are changing so fast, that the time has not come (yet), to address the issues of building large software systems.

At the same time, the Ada users develop large software systems (very often related to real-time, safety critical computing) and are primarily interested in the issues related to software reliability and portability.

Even if the software systems produced work in a distributed environment and need to satisfy hard real-time constrains (which, among others, require fast computation), the performance issues as described above are not recognized as extremely important.

The second point we need to raise, is that even if there exists research about the performance of Ada based software, it remains mostly unknown to the non-Ada community. There are two basic reasons for this. First, as shown by the research of Goforth et al. [10] which was published in *Ada Letters* and the work of Nakao and Yogi [18] published in the *Transactions of the IEICE*, these publications are not on the list of journals the most popular among researchers interested in performance. Second, the performance studies completed by the Ada researchers do not exactly match the interest of the other research camp. The research by Goforth et al. was a very typical mainstream Ada application – control of autonomous telerobotic system – which is rather atypical for parallel scientific computing applications. The subject of the research by Nakao and Yogi is of relatively low interest for the scientific computing community and, more importantly, was only partially related to Ada. Since all of their analysis is purely theoretical, this work could be easily extended to any other programming language (replacing the task generation time by the time of initialization of a slave process, and the rendezvous time by the communication overhead). This being the case, their results do not introduce any specific information about Ada's performance.

This problem is also visible from the opposite direction. The work by Kok [13] (containing a very important critical analysis of Ada's advantages and disadvantages as a language for parallel computing and discussing a number of future research direction) remains unknown among the Ada researchers, since it was published in *The International Journal of Supercomputing Applications*, the leading journal of the scientific computing research.

Summarizing, there exist two independent research camps. They have separate research programs and interests and publish in journals unrecognized by each other. This seems to suggest that there is almost no chance for reconciliation. However, without reconciliation the Hunter postulate has only a minimal chance of completion.

3.2 Proposed Solution — Extensive Efficiency Studies

We would like to suggest that the only way to close the gap between the two camps is to establish a common area of research – study of a variety of issues related to performance and efficiency of Ada. It needs to be made clear that since one of the goals of the Ada community is to expand the area in which Ada is used, it will be in some sense their goal to show that Ada is a language in which parallel scientific software can and should be written. Our basic assumption is that there exists a certain trade-off between the software reliability, portability and development ease, on one hand, and the performance and efficiency as represented by the execution time, on the other. As a typical example of such trade-off one can argue that, in Ada, in every step the ranges of arrays are being checked, which is very important for software reliability, but has adverse effect on performance [16]. Researchers interested in solving large scientific problems on parallel computers may be willing to trade some speed for the other gains, but at this moment it is unknown how much speed is in question.

It is clear that due to the objectives of the language, where software reliability and safety is of foremost importance, the performance of Ada implemented algorithms on single-processor computers will be in many cases lower than performance of the same algorithms implemented in other languages. At the same time, no substantial research has been done to find out what is really the performance difference for uniprocessors and how does it manifest itself for various classes of algorithms (for limited examples of performance data see [4]). The situation is even less clear how will the picture look like when the performance of parallel programs is to be studied. Research cited above suggests that the answer to the latter question will depend on the overhead associated with task generation and the efficiency of the rendezvous mechanism. However, almost no results estimating these parameters exist (except the work by Hind and Schonberg [11] which is quite encouraging for the Ada community).

We would like to suggest that a number of steps need to be undertaken to present a clear picture of Ada performance and to allow a fair comparison with the performance of other programming languages. This can be achieved by a collection of experimental data. An example of such an experimental work can be found in [4, 19]. It presents a comparative study of a single processor performance of array based sorting algorithms implemented in Ada and C. More studies like this are needed to provide important metrics, for example to be used by compiler designers (see also [3, 16]).

As mentioned earlier, there exist a number of benchmarks suites that were designed primarily to study the performance characteristics of various computers. At the same time, there has been already an indication that these benchmarks can be used as the basis of comparative studies between various programming languages. Why not to use these well known benchmarks to study the performance of Ada and to compare its performance on various machines (single-processor as well as parallel) with software written in other programming languages?

One of the important ways in which parallel scientific software is being developed is through the creation of software libraries. The most interesting project in this area is the collection of mathematical software available form the Oak Ridge National Laboratory. Similar software libraries need to be created in Ada and allow an exchange of research done by various researchers and to support software reuse.

Overall, the above discussion goes hand-in-hand with what was suggested by Kok ([13]):

"Though Ada is now available on many uniprocessor systems, there is still little experience with truly multiprocessor Ada implementations. This experience will grow rapidly in the near future, and it will probably prove the benefits of high level language features for the developments of new methods that successfully exploit the possibilities of parallel architectures." (p. 108)

To achieve this goal, Ada researchers must respond to the needs of people who use parallel computers and their need to have an extremely efficient code. The only way to convince them that programming in Ada is worth considering is by showing that they do not lose too much efficiency and at the same time they gain a lot in the areas of software reliability and reuse. Only through the experimental work and entering the dialog with the parallel computer users to solve their large problems can the Hunter's postulate be fulfilled.

3.3 Effects on the Computer Science Curriculum

The proposed amendment to the research agenda of the Ada community would have some effect on the computer science curriculum. Similar changes have been recently suggested by Lawlis and Adams [15]. Their general recommendation is to return the computer science curricula to the engineering basics. This leads to the increase of the amount of experimental laboratory work – which is so typical for engineering curricula [21].

Our suggestion, which follows the need for experimental benchmarking, referred to earlier in this paper, is typical for engineering and experimental sciences. We designed a number of laboratories [20, 21] during which students study efficiency issues of various algorithms and following these experiments make the best choice of either algorithm or architecture or both.

Following the standard outline of a scientific experiment:

- formulation of a precise hypothesis
- complete specification of the experimental system
- quantitative measurements and use of controls
- analysis of measured data
- report of the procedures and results

also leads to the fulfillment of educational goals of providing students with increased: problem-solving abilities, analytical skills and professional judgment – goals specified as extremely important in the educational process by Lawlis and Adams [15]. In this sense, our proposal goes beyond what has been proposed recently in [14, 23]) where the educational goal was primarily to introduce students to various aspects of parallel code development.

4 Summary

An overview of parallel computing in Ada, as reported in the literature up to 1996, has been presented. This includes the use of Ada in vector computing, SIMD computations, wavefront computing, and both shared memory and message passing MIMD computations. This is followed by a critical analysis of the use of Ada and suggestions for more focused performance studies and respective curriculum changes.

Acknowledgements

This work was supported in part by a grant from the Defense Information Systems Agency, under the Udergraduate Curriculum and Course Development Program: Software Engineering and the Use of Ada, Contract F29601-94-K-0046.

References

[1] Ada Reference Manual. International Standard ANSI/MIL-STD-1815A, 1983

[2] Ada Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995, January 1995

[3] Baker J., Analysis of Machine Code Generation, Journal of Computing in Small Colleges, Vol. 11, No. 7, pp. 162-175, 1996

[4] Baker J., D. Stacy, Comparison of Sorting Algorithms in Ada and C, Journal of Computing in Small Colleges, Vol. 10, No. 5, pp. 185-189, 1995

[5] Blum E.K., Programming Parallel Numerical Algorithms in Ada, pp. 297-304, The Relationship Between Numerical Computation and Programming Languages, J. K. Reid (Ed.), North-Holland, Amsterdam, 1987

[6] Clarson D.R., Proposal for Adding Discriminants for Ada Task Types, Ada Letters, Vol. 8, No. 5, 1987

[7] Clapp R.M., T. Mudge, Ada on a Hypercube, Ada Letters, Vol. 9, No. 2, pp. 118-128, 1989

[8] Cogan K.J., Ada for the Description of Wavefront Array Processors, pp. 267-276, Proc. Ada-Europe International Conference, Dublin, June 12-14, 1990, B. Lynch (Ed.), Cambridge University Press, New York, 1990

[9] Forest F., G. Arnaudo, Parallelism in Ada, Ada User, Vol. 8, Supplement, pp. S21-S26, 1987

[10] Goforth A., P. Collard, M. Marquardt, Performance Measurement of Parallel Ada: An Applications Based Approach, Ada Letters, Vol. 10, No. 3, pp. 38-58, 1990

[11] Hind M., E. Schonberg, Efficient Loop-Level Parallelism in Ada, pp. 166-178, Proc. TRI-Ada '91, San Jose, CA, October 21-25, 1991, ACM, New York, 1991

[12] Hunter G., The Fate of Fortran 8X, Comm. of the ACM, Vol. 33, No. 4, April 1990

[13] Kok J., Parallel Programming with Ada, International J. of Supercomputer Applications, Vol. 2, No. 4, pp. 100-108, 1988

[14] Kortright E.V., Parallel Ada Programming in Ada Under Different Architecture Paradigms, pp. 108-117, Proc. 9th Annual ASEET Symposium, Morgantown, WV, 1995

[15] Lawlis P.K., K. A. Adams, Computing Curricula vs. Industry Needs: a Mismatch, pp. 5-19, Proc. 9th Annual ASEET Symposium, Morgantown, WV, 1995

[16] Lee S., Optimizing Sparse Matrix Multiplication, Journal of Computing in Small Colleges, Vol. 11, No. 7, pp. 176-185, 1996

[17] Lyttle R.W., R.H. Perrott, P. Sritharan, Modeling SIMD-Type Parallelism in Ada, J. Pascal, Ada & Modula-2, Vol. 9, No. 2, pp. 10-16, 1990

[18] Nakao Z., T. Yogi, A Parallel Algorithm in Ada and Its Performance Profile, Transactions of IEICE, Vol. 73, No. 4, pp. 528-531, 1990

[19] Paprzycki M., J. Baker, D. Stacy, Studying Performance of Sorting Algorithms, pp. 165-173, Proc. Conference on Applied Mathematics, University of Central Oklahoma, Edmond, OK, 1995

[20] Paprzycki M., R. Wasniowski, J. Zalewski, Parallel and Distributed Computing Education: a Software Engineering Approach, pp. 187-204, Proc. 8th CSEE'95 Conference, R. L. Ibrahim (Ed.), Springer-Verlag, Berlin, 1995

[21] Paprzycki M., J. Zalewski, Shaping the Focus of the Undergraduate Curriculum, SIGCSE Bulletin, 1996 (to appear)

[22] Park E.K., P.B. Anderson, H.D. Dardy, An Ada Interface for Massively Parallel Systems, pp. 430-435, Proc. 14th COMPSAC, Chicago, October 31 – November 2, 1990, G.J. Knafl (Ed.), IEEE Computer Society Press, Los Alamitos, CA, 1990

[23] VanScoy F., Parallel Computing in Undergreaduate Education, Talk at the 9th Annual ASEET Symposium, Morgantown, WV, 1995

[24] Volksen G., P. Wehrum, Ada Scientific Computations on Vector Processors, J. Pascal, Ada & Modula-2, Vol. 8, No. 6, pp. 16-32, 1989

[25] Yemini S., On the Suitability of Ada Multitasking for Expressing Parallel Algorithms, Proc. AdaTEC Conference on Ada, 1982