TUTORIAL

INTRODUCTION TO PARALLEL COMPUTING

AND PARALLEL COMPUTING EDUCATION

Marcin Paprzycki, Janusz Zalewski
Department of Mathematics and Computer Science
University of Texas of the Permian Basin
Odessa, TX 79762

Recent years can be characterized by definitive advances in computations performed by more than one processor at a time. There are many names for this type of processing. Depending on a particular viewpoint it can be named: high performance computing, supercomputing, parallel computing or distributed computing. To cause further confusion each of these terms may have different meaning to different people. Whichever name we use it is clear that the future of computing lays in multiple processor environments executing user applications. We would like to argue that substantial changes in the computer science education are thus necessary to properly prepare the new generation of software and hardware developers.

The aim of this tutorial is twofold. First, we introduce the basic concepts of parallel computing and, second, discuss the parallel computing education.

Part 1 – Introduction

Introduction to the basic issues of parallel computing. The need for parallel computing. The Flynn's taxonomy. Existing computer architectures. Basic approaches toward parallel computation. Measures of effectiveness of parallel computing.

Part 2 – Advanced topics

Algorithmic issues. Designing programs for parallel computers. Factors hampering performance of parallel programs. Modeling performance of parallel programs (models of parallel computations). Parallel programming environments/languages and language extensions.

Part 3 – Parallel computing education

When in the Computer Science curriculum should we start? What should we concentrate our attention on? Pros and cons of various approaches. Sample model of parallel computing curriculum. Selection of examples and exercises. Discussion.

Proceedings of the Fifth Annual South Central Small College Computing Conference

April 15-16, 1994

Amarillo College
Amarillo, TX

Edited by

George A. Benjamin
Muhlenberg College, Allentown, Pennsylvania

The Journal of Computing in Small Colleges

The Newsletter of the
Consortium for Computing in Small Colleges

Volume 9, Number 5 **May 1994**

INTRODUCTION TO PARALLEL COMPUTING EDUCATION

Marcin Paprzycki, Janusz Zalewski
Department of Mathematics and Computer Science
University of Texas of the Permian Basin
Odessa, TX 79762

0. Introduction

Recent years are characterized by definitive advances in the area of high performance computing. These advances led to the formulation of a number of problems, so called Computational Grand Challenges [12]. Solutions to these problems will require substantial progress in five areas:

- mathematical understanding of the problems posed,
- the development of new algorithms,
- improvement in software construction methods and tools,
- development in implementation techniques,
- development of new hardware.

One of the important issues that needs to be addressed is the level of preparation of our students to face the challenges ahead. We argue that substantial changes in the Computer Science Education are necessary to adequately prepare the new generation of graduates.

Section 1 of this article contains a short overview of current issues in parallel computing presented by addressing the question: what is the established practice in parallel computing developments? In Section 2 a number of approaches toward introducing high performance computing into the curriculum is presented and discussed. Section 3 offers a number of ways to gain access to high performance and parallel computers. Appendix 1 contains references to a number of publications relevant to the parallel computing education.

1. State of the Art of Parallel Computing

It is reasonable to assume that applications are the driving force of most if not all advances in computer science and engineering, including parallel computing. In particular, extremely high computational requirements, in such fields as astronomy, molecular biology, quantum chemistry, fluid dynamics, theoretical physics, structural modeling, atmospheric and ocean research, to mention only a few, lead to the development of high performance and parallel computers.

The need and demand of applications, first of all, enforces the development of new algorithms. Several classes of algorithms can be distinguished for solving typical problems [1, 8, 14, 16, 17, 18, 20]:

- parallel selection and searching, merging and sorting algorithms,
- parallel graph algorithms,
- parallel numerical algorithms,
- computational geometry and image processing,
- parallel combinatorial algorithms,
- parallel optimization,
- neural network and genetic algorithms.

It is relevant and crucial for software engineers, engaged in software development for parallel computers, to have good knowledge of parallel algorithms. Then software construction methods and environments can be more effectively built and applied. Unfortunately, software development methodologies very seldom touch parallel computations; they usually focus only on traditional sequential von Neumann machines. In general, a good software development methodology, for developing either traditional sequential or parallel software, should include three aspects: the method, techniques, and tools. The method is a set of paradigms and a notation to express and specify the intended software properties and their relationships. Techniques are mathematical transformations that support the paradigms and enable software development in a systematic and rigorous manner, and tools are software packages that help in automatic development. For traditional sequential systems, there are several methodologies and techniques, well described in literature and supported by tools. At the same time (to the best of our knowledge) no such comprehensive methodology exists that would comprise all three components and aim parallel systems. Regarding methods and techniques, a promising approach is represented by Petri nets, however, there seems to be a lack of commercial tools supporting this approach [4, 11]. Among the environments which offer support for most phases of the software development cycle, although their theoretical basis is unclear, there are two worth mentioning:

- CODE/ROPE [5]
- PPSE [19].

Implementation techniques form one of the best developed segments of parallel computing. This comprises languages for parallel computations and respective operating system constructs. Based entirely on relatively well developed theory of concurrency, they define models of concurrent and parallel computations and define language primitives most suitable from the point of view of such models. The most general notion, a concurrent system, is defined as a system whose multiple activities are performed within the same time interval. A parallel system is a special case of a concurrent system, whose multiple activities are performed at the same time instant. A distributed system is a special case of a parallel system, where communication time among processors is non-negligible. The most important primitives designed for concurrent programming include: critical sections, barriers, semaphores, signals, event flags, monitors, mailboxes, rendezvous, remote procedure calls, and others. Their usage depends on the assumed model of concurrency, whether shared memory or message passing.

There are several languages used for implementation of parallel systems, from traditional languages, such as Fortran and C, through Ada, Occam, Linda [7], Modula-P [4], Erlang [2] and functional, as well as, logic programming languages.

Hardware architectures for parallel computations comprise a variety of approaches, mostly following the perpetual Flynn classification:

- SISD -- sequential in principle; can be extended to VLIW (Very Long Instruction Word) architecture, inherently parallel,
- SIMD -- array processors,
- MISD -- resembling pipeline architectures,
- MIMD -- usually called a multiprocessor,
- other -- e.g. dataflow architectures.

An important issue in multiprocessor machines is a distinction between shared memory (tightly coupled) multiprocessors and message passing (loosely coupled) multiprocessors. Another crucial problem is the type of interconnection, which includes three principal categories:

- bus,
- cube,
- crossbar switches.

Most of the currently existing curricular suggestions (some of which are discussed below) concentrate their attention on particular subareas of parallel computing development and, possibly, correlations between them. We would like to suggest that even though these approaches are valuable individually, a more general approach toward parallel computing education is necessary. This approach is based on teaching parallel computing on all levels discussed above.

2. Teaching parallel (high performance) computing

Current trends in computer science indicate clearly that the future of computing lays in multiple processor environments executing users applications. This can be already observed in top of the line PC's that include the main processor, a video processor, the sound processor, floating point processor and other coprocessors. If this is the case, then when and how are we going to introduce our students to parallel computing? There exists a variety of approaches.

The first one comes from the assumption that parallel computing is rather hard, and that our knowledge about its principles is not well grounded, so we should not introduce it too early (meaning, we will do it only on a graduate level). Instead, we should introduce our senior undergraduate students to vector-computer based high performance computing. Vector computing has been around already for some time and good quality vectorizing compilers exist for a variety of machines. Assuming a prerequisite knowledge of at least one programming language (preferably Fortran or C), a course of this type can consist of (based on [21]; possible textbook [13]):

1. Introduction to Unix (Unicos),
2. Discussion of a vector computer architecture,
3. Performance modeling and evaluation,
4. Designing algorithms for a Cray,
5. Using Cray provided libraries.

The second possibility would be to design a multidisciplinary senior level undergraduate course in which students majoring in sciences or engineering would have a chance to get exposure to solutions that modern computers can provide for their respective disciplines. This is a course concentrated primarily on the usage of existing libraries. The prerequisite is at least one course in computer science above computer literacy. It is assumed that each student will work on a term project in his/her discipline. The content of the course can be modified according to the interests of participating students. It can consist of [21]:

1. Introduction to Unix,
2. Introduction to Visualization,
3. Overview of supercomputing applications in Biology,
4. Overview of supercomputing applications in Chemistry,
5. Overview of supercomputing applications in Physics,
6. Overview of supercomputing applications in Geology,
7. Overview of supercomputing applications in Engineering,
8. Presentation of term projects.

The third approach is to introduce parallel computing already in a junior level undergraduate course. Here a typical course may be structured as follows (based on [15]):

1. Introduction to parallel processing (parallelism in what the students know already),

2. Basic parallelizing techniques,
3. Basic parallel architectures,
4. Computational models for parallel processing,
5. Design and analysis of parallel algorithms.

This course could be followed on a senior level by an advanced course consisting of a selection from the topics related to the following areas of parallel computing (based on [15]):

1. Languages,
2. Compilers,
3. Operating systems,
4. Interconnection networks,
5. Heterogeneous parallel computers,
6. In depth studies in parallel computer architectures.

The final approach is to introduce parallel computing already in the early stages of computer science education. This approach most closely matches the entirety of parallel computing as presented in Section 1. In this approach we suggest that no particular course should be devoted to parallel computing, but rather parallel computing should be introduced in a variety of courses (typically considered a part of the core CS curriculum). We propose that as a minimum the following courses should be modified to introduce parallel computing concepts:

1. Computer Science I
   - preparatory concepts: arrays, separate compilation; no explicit parallelism introduced at this stage.
2. Computer Science II
   - elementary concepts of parallelism; parallelization of elementary searching and sorting.
3. Computer Architecture
   - instruction-level parallelism; cache; bus arbitration.
4. Programming Languages (or Operating Systems)
   - implementation issues; multiple processes or tasks; selected interprocess communication constructs.
5. Data Structures
   - concepts of parallel algorithms.
6. Software Engineering
   - parallel/concurrent software development methodologies; use of tools.

If one adopts our strategy then upper level courses (such as programming algorithms, data communication, database systems) could (and should) be modified to include substantial elements of parallel computing. Let us observe that this approach does not require addition of new courses (which may be almost impossible to do without removing some other courses, or increasing the graduation requirements). All that is required is to adjust the content of the existing courses. It can be also suggested that if one of the graduation requirements is a Senior Research Project, then such a project could be conducted to encompass all areas of parallel computing.

3. Availability of Resources

To run the parallel computing course smoothly, it is necessary to gain access to high performance and parallel computers. Not so long ago, high performance computers were extremely expensive and only the big corporations (e.g. Boeing, Exxon, etc.), government

institutions and state supported consortia were able to purchase them. This situation has changed rapidly. There were two factors that led to these changes. The first was the competition between a number of high performance computer vendors (which led to disappearance of many of them, as a side effect). The second was the increased performance of desktop computers combined with the development of software that allows to connect a number of desktop computers and use them as a high performance, distributed memory machine. Right now one can put together a parallel computer for less than $10,000, which is affordable to most colleges. Below we briefly address a question how to provide students with access to high performance environments.

3.1 Access to large computers

There is a number of ways of getting access to high performance computers. Since the cost of high performance workstations starts at about $40,000 and the smallest Cray is about $100,000, some colleges (or consortia of colleges) can afford purchase of such equipment. There exists also a number of NSF-sponsored Supercomputing Centers which may provide computer time for educational purposes:

1. Cornell Theory Center,
2. National Center for Supercomputing Applications,
3. Pittsburgh Supercomputing Center,
4. San Diego Supercomputing Center.

Other sources of supercomputer power can be state supported centers: Utah Supercomputing Center, Ohio Supercomputing Center and others, as well as, national laboratories. It can be also advised to contact a local representatives of the supercomputer vendors. They may be able to arrange some computer time in one of the sites that own their equipment. In each of these cases the only thing required is an Internet connectivity and TCP/IP-like software.

3.2 Tightly coupled systems

A very natural way to introduce smoothly parallel computing into computer science curriculum is to extend the basic knowledge of computing into parallel computing. This can be done by building on the traditional concept of von Neumann architecture and introducing multiple processors (MIMD machines) connected via point-to-point links or hooked to a shared bus. In the first case, this can be relatively easy done with transputer equipment which is offered by several vendors at an affordable price, for example [9]. In the latter case, standard multiprocessor bus architectures, such as VMEbus, Multibus I and II, NuBus, or Futurebus+, are gaining more and more popularity and become more affordable. A small scale VMEbus system may initially cost a couple of thousand dollars, and additional processor boards can be attached in subsequent years [22].

The advantage of this solution is that it is comprehensive and allows to teach parallel processing on several levels mentioned in the Introduction: from hardware architecture, through implementation issues (language constructs and operating system calls) to software development methods (especially if the target system is connected to a workstation).

3.3 Networking of existing hardware

The third approach is to use the existing hardware to create one's own parallel computer. It is enough to have a number of PC's or workstations running Unix, networked together, to build a distributed memory (possibly heterogeneous) MIMD computer. There is a number of software tools that can be used to emulate a parallel computer taht way. The most popular are PVM [10], P4 [6], Parallaxis [3, 4] and others. PVM and P4 are available free of charge from the *netlib* (send an e-mail message to *netlib@ornl.gov* containing the word *index*).

4. Conclusion

Parallel computing developments can be envisioned on many levels. We have argued that to adequately educate new generations of students, all these levels should be represented in the curriculum. We have suggested a new approach to introducing parallel computing into the computer science education. It has two main advantages. First, it represents all levels used to conceptualize parallel computing developments. Second, it does not require addition of new courses to the curriculum, only modification of existing ones.

References

1. Akl, S.G., *The Design and Analysis of Parallel Algorithms*, Prentice-Hall, 1989
2. Armstrong, J., Virding, R., Williams, M., *Concurrent Programming in Erlang*, Prentice Hall, 1993
3. Barth, I. et al., *Parallaxis Version 2 User Manual*, University of Stuttgart, Computer Science Report 2/92, February 1992
4. Braunl, T., *Parallel Programming. An Introduction*, Prentice Hall, 1993
5. Browne, J.C., Azam, M., Sobek, S., CODE: A Unified Approach to Parallel Programming, *IEEE Software*, 6 (4), 1989, 10-18
6. Butler, R., Lusk, E., *User's Guide to the P4 Programming Systems*, Report ANL-92/17, Argonne National Laboratory, October 1992
7. Carriero, N., Gelernter, D., *How to Write Parallel Programs. A First Course*, MIT Press, 1990
8. Chaudhuri, P., *Parallel Algorithms: Design and Analysis*, Prentice-Hall, 1992
9. CSA, Computer Systems Architects, 100 Library Plaza, 15 North 100 East, Provo, Utah 84606-3100
10. Geist, A. et al., *PVM 3.0 User's Guide and Reference Manual*, Report ORNL/TM-12187, Oak Ridge National Laboratory, February 1993
11. Ghezzi, C., Jazayeri, M., Mandrioli, D., *Fundamentals of Software Engineering*, Prentice Hall, 1991
12. *Grand Challenges 1993: High Performance Computing and Communications*, A Report by the Committee on Physical, Mathematical, and Engineering Sciences, Federal Coordinating Council for Science Engineering and Technology, Washington, 1992
13. Hennessy, J. L., Patterson, D.A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, 1990
14. Jaja, J., *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992
15. Khan, Z.S., Incorporating Supercomputing in the Undergraduate Computer Science Curriculum, Proceedings of the Eighth Annual Eastern Small College Computing Conference, *The Journal of Computing in Small Colleges*, 8 (2), 1992, 112-121
16. Kronsjo, L., Shansheruddin, D., (Eds.), *Advances in Parallel Algorithms*, John Wiley, 1992
17. Kumar V. et al., *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, 1993
18. Leighton T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*, Morgan Kaufmann, 1991

19. Lewis, T.G., Rudd, W.G., Architecture for Parallel Programming Support Environments, *Proc. COMPCON '90*, IEEE CS Press, 1990, 589-594
20. Reif J.H. (Ed.), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, 1993
21. Stewart, K., Presentation at the Faculty Workshop on Curriculum Development -- "Supercomputing and Undergraduate Education", San Diego Supercomputing Center, July, 1992.
22. VITA, VFEA International Trade Association, 10229 N. Scottsdale Rd, Suite B, Scottsdale, Arizona 85253

Appendix 1

Literature on parallel computing education

1. Bachelis, G., Maxim, B., Tutorial: Introducing Parallel Algorithms in Undergraduate Computer Science Course, *SIGCSE Bulletin*, **22** (1), 1990, 255
2. Butler, R.M., Eggen, R.E., Wallace, S.R., Introducing Parallel Processing at the Undergraduate Level, *SIGCSE Bulletin*, **20** (1), 1988, 63-67
3. Fisher, A.L., Gross, T., Teaching the Programming of Parallel Computers, *SIGCSE Bulletin*, **23** (1), 1991, 102-107
4. Guha, R.K., Hartman, J., Teaching Parallel Processing: Where Architectures and Language Meet, *Proc. IEEE Conf. Frontiers in Education*, 1992
5. Hartman, J., Sanders, D., Data Parallel Programming: A Transition from Sequential to Parallel Computing, *SIGCSE Bulletin*, **25** (1), 1993, 96-100
6. Hartman, J., Sanders, D., Teaching a Course in Parallel Processing with Limited Resources, *SIGCSE Bulletin*, **23** (1), 1991, 97-101
7. Hintz, T., Introducing Undergraduates to Parallel Processing, *IEEE Trans. on Education*, **36** (1), 1993, 210-213
8. Hyde, D.C., A Parallel Processing Course for Undergraduates, *SIGCSE Bulletin*, **21** (1), 1989, 170-173
9. Jipping, M.J., Toppen, J.R., Weeber, S., Concurrent Distributed Pascal: A Hands-on Introduction to Parallelism, *SIGCSE Bulletin*, **22** (1), 1990 94-99
10. John, D.J., Integration of Parallel Computation into Introductory Computer Science, *SIGCSE Bulletin*, **24** (1), 1992, 281-285
11. Kitchen, A.T., Schaller, N.C., Tyman, P.T., Game Playing as a Technique for Teaching Parallel Computing Concepts, *SIGCSE Bulletin*, **24** (3), 1992, 35-38
12. Luque, E., Suppi, R., Sorribes, J., A Quantitative Approach for Teaching Parallel Computing, *SIGCSE Bulletin*, **24** (1), 1992, 286-298
13. Meredith, M.J., Introducing Parallel Computing into the Undergraduate Computer Science Curriculum, *SIGCSE Bulletin*, **24** (1), 1992, 187-191
14. Miller, R., *The Status of Parallel Processing Education*, Draft Report, SUNY Buffalo, September 1993
15. Mims, T., Hoppe, A., Utilizing a Transputer Laboratory and Occam2 in an Undergraduate Operating Systems Course, *SIGCSE Bulletin*, **23** (1), 1991, 317-323
16. Nevison, C., An Undergraduate Parallel Processing Laboratory, *SIGCSE Bulletin*, **20** (1), 1998, 68-72

17. Paprzycki, M., Incorporating High-Performance Computers into Mathematics Curriculum, *Proceedings of the Fifth Annual Conference on Technology in Collegiate Mathematics*, Addison-Wesley, Reading, 1993, 862-868

18. Sanders, D., Hartman, J., Getting Started with Parallel Programming, *SIGCSE Bulletin*, 22 (1), 1990, 86-88

19. Torsone, C.M., Introducing Parallel Programming to a Programming Language Concepts Course, *Journal of Computing in Small Colleges*, 9 (2), 1993, 66-70