Why Computational Science and Engineering Should be of Interest to Computer Scientists

Lasse Natvig
Group for Computer Architecture and Design, Department of Computer Systems and Telematics,
Norwegian Institute of Technology (NTH), University of Trondheim, N-7034 Trondheim, Norway.
(E-mail: Lasse.Natvig@idt.unit.no)

Abstract: This paper briefly introduces the field computational science and engineering (CS&E), and is an attempt to get other computer scientists more interested in CS&E related activities. It starts by giving a short outline of the increased international activity in the field. Several of the definitions of CS&E that have been given are presented, with an emphasis on how the field is related to computer science. The role of supercomputers is discussed, and we try to identify important challenges for future CS&E education and research, again with a hope to attract computer scientists.

Keywords: computational science and engineering, scientific computing, supercomputing, high performance computing (HPC), education, computer science.

1. Introduction

Motivation

As part of a committee work on a possible building up on computational science and engineering at the University of Trondheim we have tried to identify what Computational Science and Engineering (CS&E) really is, and how it is related to other more established fields such as mathematics and computer science. During this work, being a computer scientist in a computer science department, the author has to his own surprise found a very low extent of interest in CS&E among computer scientists. This has motivated for writing the paper. It tries to explain computer scientists (including myself) what computational science is, and how it contains a lot of interesting challenges for computer science research. I also argue, perhaps a bit provoking for non-computer scientists, that the computer science community has a responsibility for participating on the CS&E arena to ensure quality.

When reading related publications it is clear that the lack of interest in CS&E among computer scientists has been observed by many authors, both nationally and internationally [BW93, Rice93, misc94]. At the University in Trondheim, researchers from the various engineering departments have expressed opinions like; "*the computer scientists are playing in their own field ignoring the use of computers to solve important real world problems in science and engineering"*. This is far from entirely true, but it seems clear that computer science at many places has "grown away" from problem solving in natural sciences and engineering. We try to sketch some possible reasons for this kind of dissatisfaction in Section 5. The paper may be regarded as a small attempt to improve the collaboration between our computer science department and other departments.

Increased international activity

We have lately seen a world-wide increased activity in high-performance computing in natural sciences and engineering. Initiatives such as HPCC (High Performance Computing and Communication) in the U.S. and HPCN (High Performance Computing and Networking) [HPCN94] in Europe are to a large extent focusing on computational science. A substantial part of the fourth framework programme of the European Union [EC94] is devoted to HPCN.

Although a small part of the HPCN activity in Europe may be termed as "pure computer science" projects focusing on topics such as programming environments and message passing libraries, the main focus of HPCN in Europe is on *applications*, with a substantial involvement shown by the large industrial companies. Such projects typically have a significant computing component involving the use of traditional vector based supercomputers and parallel computers. It is the goal of this paper to argue that computer science should play an important role also in this kind of projects.

There have also been an increased activity on the educational side, perhaps most evident in the U.S. John Rice has described thirteen CS&E (or similar) educational programs [Rice93]. These programs try to give their students the knowledge needed to solve computational problems in engineering and science. It is reported that the situation today is that we too often have engineers or scientists working on CS&E projects with too little knowledge about computing, or computer scientists with too little know-how in engineering and science.

Disclaimer
The author has little experience in computational science and engineering. Thus readers from that field may find imprecise or computer science biased formulations. It is my hope however that parts of the paper may be useful also for that community, and all kinds of comments are highly appreciated.

Acknowledgements
I would like to thank Bjørnar Pettersen, Syvert P. Nørsett and Karstein Sørli for numerous discussions that have been instructive and motivating for writing this paper.

2. Computational Science and Engineering

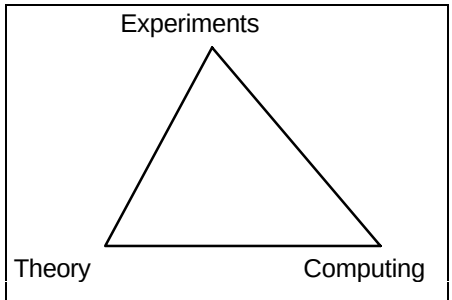

Fig. 1. Three methods of investigation in natural sciences and engineering. (Computing includes, in this figure, numerical computations, simulation and visualisation.)

Background
The power and availability of modern computers have made numerical calculations, computer simulations and other use of high performance computers to an important tool in almost all disciplines of science and engineering. Many researchers claim that *computing has become a third main method for doing investigations, besides theory and experiments*, [Rice93, BW93, CSEP94]. There is probably no consensus about the relative importance of computing compared to the other two parts of this triad (Fig. 1.). However, there is no doubt that computing is being used in more and more disciplines as a main tool. Computing may be used *instead of theory and experiments* by providing insight in many phenomena that are too complex to be handled by analytical methods, too expensive, too dangerous or impossible to study through experiments [CSEP94]. Computing may also be a *supplement to theory and experimentation* resulting in an interplay among the three main methods. This is often the case in CS&E projects.

And we should not forget that computing can be used to help pure theoretical work (e.g. symbolic derivation) and traditional experimentation (e.g. data collection, analysis and visualisation).

Proposed Definitions

First of all, it is important to state that *computational science and engineering* is not the same as *supercomputing*, which we understand as the use of supercomputers. More and more often, other kinds of computers than supercomputers, e.g. workstations, are being used in CS&E projects. Thus, although there in practice has been little difference between the two terms, the difference may increase in the future, and CS&E can be regarded as independent of the HW-platform.

At present, the computational approaches used by scientists and engineers are very similar [CSEP94], so *computational science* is often used as a shorthand for computational science *and engineering*. A very short (informal) definition of computational science is given by D. E. Stevenson in his article *Science, Computational Science and Computer Science: At a Crossroads* [Stev93]:

> *"We describe computational science as an interdisciplinary approach to doing science on computers."*

The difference between the term *scientific computing* and computational science is presently not completely understood by the author. Golub and Ortega give the following working definition of "scientific computing" [GO93 p. 2.]:

> *"Scientific computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in science and engineering."*

Golub and Ortega describe the difference in the following way:

> "... The techniques used to obtain such solutions [of mathematical models that represent some physical situation] are part of the general area called *scientific computing*, and the use of these techniques to elicit insight into scientific or engineering problems is called *computational science* (or *computational engineering*).

In my view, the difference expressed in the second part of this quotation may become a bit clearer when we consider another definition

> ..... *"use of HPC technology to advance the state of knowledge in a particular applications discipline"*

Thus, informally, scientific computing may be perceived as more focused on "obtaining a solution on a computer", while CS&E can be said to be more focused towards using such solutions to increase the understanding of problems in science or engineering. In other words, in the term *computational science* it is important to realise that *science* is a variable *x*, representing any natural science or engineering discipline. Thus we have computational chemistry, computational physics, computational solid mechanics etc. Computational science is *not* the science of how to do computations, covering topics such as computational complexity theory (algorithm analysis, computability, *NP*-completeness etc.). Those topics are part of the well-established field theoretical computer science (TCS). CS&E experts need a certain body of knowledge in such topics just as

they need it in mathematics, but computational science must not be misunderstood as a new name on such relatively traditional "computational topics" within computer science.

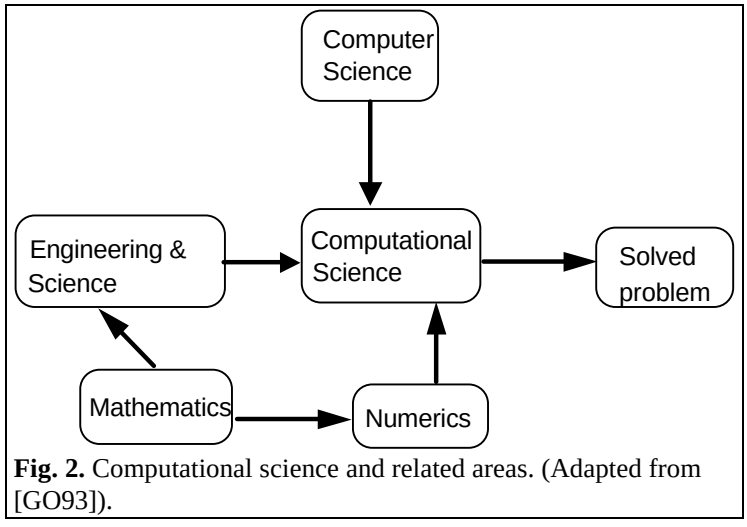

Fig. 2. Computational science and related areas. (Adapted from [GO93]).

Although there is no consensus about a single definition of computational science, most researchers emphasise and agree on its interdisciplinary nature. In their book about scientific computing, Golub and Ortega describe that field and its relation to other well-known sciences, see Fig. 2. According to my view of the CS&E field presented above, CS&E projects involve scientific computing. I have therefore taken the liberty to adapt a figure given by Golub and Ortega by substituting scientific computing with computational science. Figure 2 explains how computational science is based on mathematics, numerical analysis and computer science to solve problems from any science or engineering domain on a computer system.

Computer Use in CS&E Projects

One might argue that using computers today does not require much computer science knowledge, so that computational scientists need not a large amount of computer science background, and the computer scientists may therefore "remain in their playground". Below we argue that the nature of the computer use in CS&E projects requires more general computer science knowledge than what is required for more standard or traditional "computer" use.

There seem to be a relatively broad consensus that some kind of a small package of basic knowledge about computers and their use are needed in an educational programme for scientists and engineers that will use computers to solve problems. In this context the need for "*low level*" computer knowledge may decrease as the computers continue to be more and more user friendly. However, there are several aspects of the computer use typically involved in CS&E projects that imply a need for computer science knowledge.

- The computing involved in CS&E projects has traditionally been done on vector computers like the Cray computers, where traditional sequential programming languages have been dominating. During the last ten years there has been a clear trend towards using *parallel computers of various kinds*, and lately the use of workstation clusters has become popular. The use of these computer architectures has in general made it more frequently necessary for the programmer to use *explicit parallelism* and to know the underlying computer architecture. It is also a fact that the number of various computer architectures that may be used for the computations has increased [Panc93].

- CS&E projects are typically requiring high performance computers to finish the numerical computations fast enough, or to solve a problem with enough precision within a given time

[Thom93].There has always been the case that *striving for high performance* on a given computer requires detailed knowledge about that computer. The general trend towards an increased portability of programs is (hopefully?) also seen on the CS&E-arena, but the use of compiler directives and other ways of improving performance are still in common use. The need for knowledge about the underlying computer will decrease more slowly for programs where performance is in focus, than for programs where the development time is more crucial. Detailed knowledge about the actual computer used in a CS&E project should probably be achieved through that project, but to avoid starting from scratch, a broader basic knowledge about relevant computers should be part of the curriculum for a CS&E-specialist.

- Although there has been a rapid improvement during the last years, the state of the art in programming environments for parallel computers are lagging behind those for sequential programming. There is currently a substantial research activity in this area, but there are at least two problems facing this research. One is the rapidly changing menu of different parallel computer architectures. This makes it more difficult to decide what should be reflected through the interface. It does also make it both more important and difficult to develop portable programming environments. A second obstacle is that the computers and its system software in this market typically are less mature than for traditional sequential computers. A programmer on a CS&E project must therefore be willing to live with an incomplete and possibly changing programming environment, and be prepared to detect, locate, understand and correct or circumvent errors and weaknesses in the computer system.

To summarise, CS&E projects typically involve a more advanced use of a wider diversity of less stable and developed computing environments. This implies a need for a broad basic knowledge in computer systems and architecture, operating systems, and parallel programming.

3. Curricula

The reader should consult the references [Rice93, CSEP94, Same93] for an overview and examples on typical contents of existing educational programs in CS&E. In the context of the engineering profile given to students at the Norwegian Institute of Technology (NTH), our proposal for curriculum for CS&E specialist consists of four main parts.

1. *Common NTH engineering base*: Basic courses in physics, chemistry, mechanics, mathematics, computer science, social studies, business administration etc.

2. *Mathematical courses relevant for CS&E:* mathematical modelling, numerical methods and analysis, differential equations, linear algebra, statistics, optimisation, validation, error analysis, numerical algorithms, computational complexity theory, **mathematical and numerical program-development (Maple, Mathematica, and others).**

3. *Computer science courses relevant for CS&E:* programming methods, software engineering, modelling, simulation, parallel algorithms, (high-level) computer architecture (with emphasis on supercomputers), computer graphics, visualisation, **programming**

environments, tools, languages and standards for supercomputing (Fortran-90, HPF, MPI etc.)

4. *Application discipline X (e.g. chemistry):* A thorough general knowledge in the applications discipline, CS&E methods and tools for that specific application (e.g. modelling techniques, numerical methods and visualisation techniques and tools used in computational chemistry).

Whether the courses from the "intersection between mathematics and computer science", e.g. complexity theory, are given by math faculty or computer science faculty is not of importance for the student. However, it is desirable that such courses are given by teachers seeing this interdisciplinarity.

From ad hoc practical courses to standard CS&E methods

The topics written in boldface above can be considered as typical CS&E topics of today.[1] In my view this indicates the relatively immature state of the CS&E discipline. We have no standard CS&E theory or well-established CS&E textbooks at the moment.

It is important to identify general elements of computational science which is not specific for any language, computer system, computer architecture or application domain. Such elements are perhaps hard to identify today, but should form *the core* of a programme for educating specialists in computational science. In addition to this core, there will be more specific, but still partly generic computational topics within each application domain, for instance in computational chemistry.

The choice of computer architecture, operating system or language should not be overemphasised. Much of the problems within computational science and engineering are the same whether the computation is run on a supercomputer or a workstation. However, it is still far too early to expect competitive top-level results on high performance computers merely with a knowledge of well-known high level languages. Explicit parallelisation, message passing or performance tuning is still needed. This gives a need for two main kinds of courses in educational programmes in CS&E.

a) *Practically oriented courses* focusing on the use of those tools that at the moment play a crucial role in the relevant CS&E projects. Also courses like "what to do with system *x* on computer *y* to achieve good performance". Many such courses are given by computing centres today.

b) *Durable CS&E courses* in techniques and theory that are typical for CS&E projects but are less dependent on the current technology. As the CS&E discipline becomes more mature it will be easier to identify themes for this second class of courses, and improvements in high level languages etc. will reduce the need for giving type a) courses. In the future, it is not unlikely that the CS&E-topics are given by a specialised CS&E department.

Three kinds of CS&E specialists

Large CS&E projects might benefit from having CS&E specialists with three different profiles as shown in Table 1. The three kinds of specialists can be characterised by the percentage of courses

[1]Note that one might find standard courses in UNIX and C beeing marketed as CS&E-courses.

taken in mathematics, computer science or a specific application domain. (The common knowledge base, part 1 in the list above, is neglected in this context.) An ideal project team might consist of a number of application-oriented specialists from the actual field(s), one computer science oriented specialist and one mathematics oriented specialist. The computer science oriented specialist should have a general knowledge of CS&E, but the main role will be to help the other team members with the most difficult computer science related subtasks. Similarly, the mathematical oriented specialist will be the main person for solving difficult mathematical subtasks.

Table 1. Three possible profiles of CS&E specialists.

% applic. / % comp.sci. / % maths.	
60/20/20 **Application oriented**	A standard CS&E specialist having application area X as "home department". This is the most common type of CS&E specialist.
20/60/20 **Computer science oriented**	A CS&E specialist with a computer science department as home department.
20/20/60 **Mathematics oriented**	A CS&E specialist with a mathematics department as home department.

Several institutions follow this kind of model with different home departments, e.g. Rensselaer Polytechnic Institute [Rice93]. This way of organising the CS&E education may be regarded as a natural evolution from the previous CS&E activities within the various departments.

4. Supercomputers and CS&E

The role of supercomputers

Traditionally, most CS&E projects have involved the use of vector supercomputers from vendors such as Cray Research, Fujitsu, NEC and others. It has often been necessary to use supercomputers to obtain the results of the computation intensive programs fast enough, or to obtain precise enough solutions. Traditionally, the most powerful computers have been based on vector processing.

For some years various computers based on massively parallel processing (MPP) have shown performance results that are better than those obtained on vector computers for many important applications. Recently, also Cray Research has entered the MPP arena. Powerful workstations, or workstation clusters may also be used in CS&E projects. At present, the diversity of computers used in CS&E projects is probably larger than ever before.

Clark Thomborson has written an interesting article that may be helpful if you are in doubt whether to do your CS&E project on a workstation or a vector supercomputer [Thom93]. Thomborson claims that it will probably not be worthwhile to attempt porting a program to a vector supercomputer if you answer 'no' to more than one of the following five questions:

1. Would you pay 10 times as much to solve your problem 30 times faster?
2. Can your code be easily vectorised?
3. Will your program be used enough to warrant hand-optimisation?
4. Can you afford to retune your software's floating-point arithmetic?

5. Does your application require more than 200 MB but less than 16 GB of primary memory?

The author has faced arguments like this; "Cray type computers are becoming less popular, and consequently computational science will get a decreasing importance". A possible explanation of this false view is that many current CS&E courses are related to traditional vector processors. It may be right that the Cray-type vector supercomputers will be less dominant in the future, but the reader of this paper should now realise that CS&E is not equivalent with computing on large vector computers. The typical computer use in CS&E projects was discussed at end of Section 2, and a direct consequence of the definition of CS&E is that the actual kind of computers used in a CS&E project is in principle irrelevant.

The role of architectures may be changed but probably not reduced
Above we have argued that CS&E specialists should have a reasonable base knowledge of contemporary computer architectures. The increased availability of different kinds of supercomputers through nation-wide and international high-capacity networks often gives the CS&E specialist the option of choosing what computer is best suited for a given application.

The need to know details of the computer systems will probably continue to be less important because programming environments will continue to make life easier for the programmer. Easier programming will again make it possible to develop even more complex models of real world phenomena. The total complexity is not likely to be reduced, and the need to develop and understand the computations will continue to demand good abstractions and identification of essential *computational structures*.

As an example, a 2-dimensional mesh-structure of parallel (possibly virtual) processors, will continue to be relevant as long as explicit parallelism is used by the programmers. The existence of automated mapping of a logical structure to a given parallel supercomputer will reduce the need for knowing the details of the computer. However, the documentation of the computer and its mapping tools will most probably describe what logical structures that can be mapped in an efficient manner, and those that most likely will result in bad performance. Just as the newer discipline software architecture has emerged lately, a discipline that we propose to call "*computation architecture*" will be increasingly more important as the models become more complex and the computers more easy to use.

5. Why Should Computer Scientists be Interested in CS&E?

Both internationally and in Norway [BW93] it has been noted that the computer science departments typically have reduced their focus on science and numerics. In the start natural sciences and engineering were the main applications of computing. However, the applications of computers grew rapidly to many other areas. In addition, research in computer science started to be more focused on the inherent problems and challenges of computer systems. A lot of challenges in the development of hardware and software systems was identified. We may have had a shift of focus for the computer specialists away from "helping others using computers" towards "attacking central issues for the computer industry". In addition, we have seen a growing number of application areas outside natural sciences and traditional engineering that have attracted the researchers. Se Figure 3.

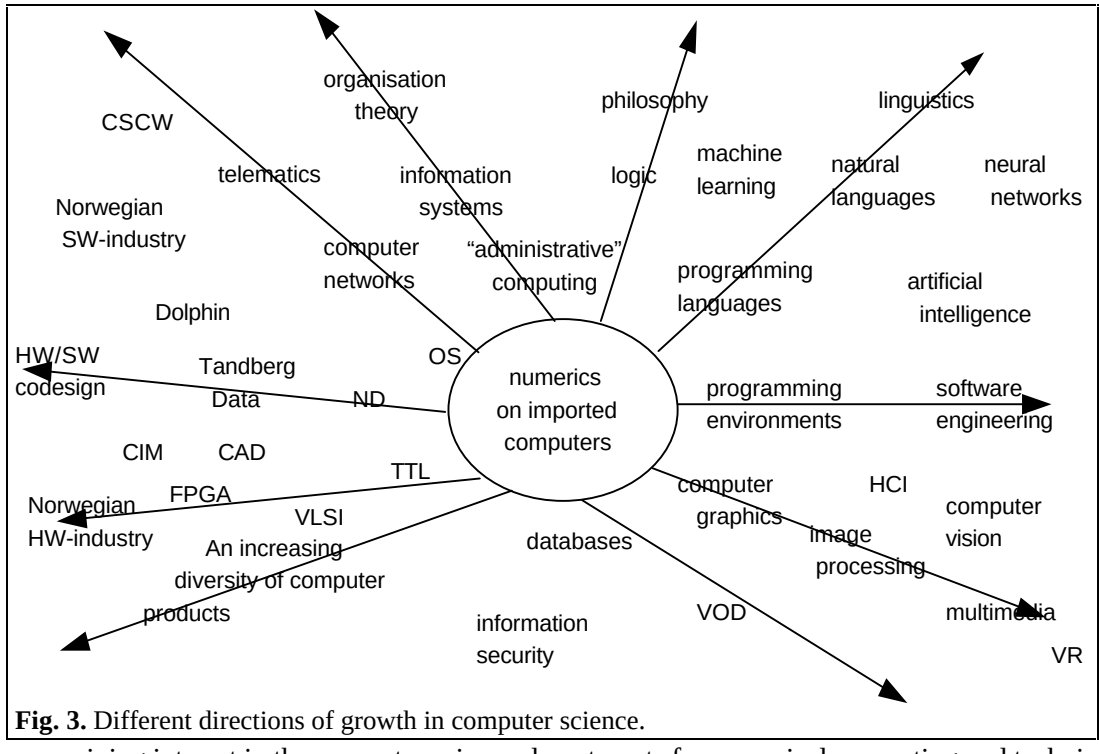

Fig. 3. Different directions of growth in computer science.

The remaining interest in the computer science departments for numerical computing and technical applications is small in many universities. The CS&E initiatives at some institutions are motivated by a wish to bring the computer science departments back to science and engineering. This is also the case at the University of Trondheim.

As discussed at the end of Section 2, the computer use typically involved in CS&E projects is in general more advanced and demanding than "normal" computer use. A result of that is that CS&E specialists need a broad computer science knowledge base. This education should be given by computer scientists, just as mathematics should be taught by mathematicians. A practically oriented CS&E course that currently is needed at NTH is the teaching of FORTRAN-90 or HPF (High Performance Fortran) to CS&E-students. This should ideally be done by computer scientist faculty or other persons having a broad and up to date knowledge of modern programming practices and software engineering.

In my view, computer scientists have a responsibility for bringing the new and important software engineering methods and practices into the CS&E projects. This should be done both through a substantial amount of software engineering themes in the education, and also by having at least one computer science oriented CS&E specialist (See Table 1) in CS&E projects involving development of large software systems. Perhaps a bit provoking, the author believes that this computer science involvement is necessary to ensure or improve quality of the produced software.

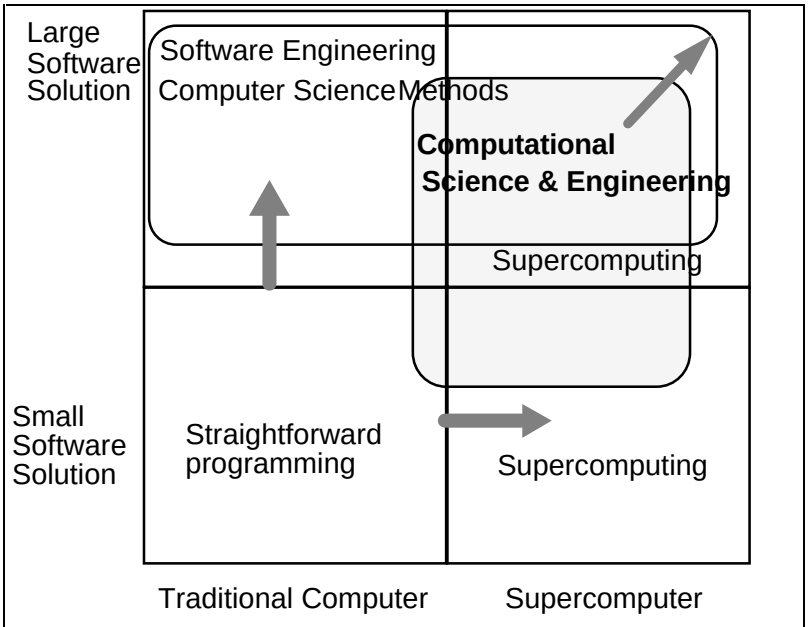

Fig 4. The role of computer science methods in computational science.

Figure 4 illustrates the cases when software engineering and other computer science methods are most important. The development of small programs on traditional computers is, with the existence of high level languages, a relatively straightforward task. In the figure a "small software solution" is roughly smaller than a few thousand source code lines. Many important CS&E problems are solved with such small programs on workstations or on supercomputers. The importance of software engineering and computer science methods increase with the size of the produced software. In the development of systems consisting of tens or hundreds of thousand's source code lines, performance and quality depends heavily on the use of good engineering methods. Experience from other "programming in the large" projects is highly desirable. The situation is analogous with the building of a house. When building a very small house with one room and without water and electricity, you may succeed alone by starting with the joiner's work at day one. If the goal is a huge or complex building, a lot of planning and engineering work is required in the initial phases and throughout the project.

One might argue, as indicated in the figure, that the vertical border moves towards right since more and more problems may be solved without using supercomputers. Similarly, more powerful programming languages and environments make it possible to solve more and more problems with what we might call straightforward programming. A conclusion could be that fewer CS&E projects will need the use of software engineering methods on supercomputers. This is probably not true, since both the demand of computing resources and the complexity of the software solutions are expected to continue increasing also in the future. This is indicated by the arrow in the upper right corner of the figure.

Research challenges in CS&E for computer scientists

In addition to giving computer science education and leading the use of computer science methods in CS&E projects, computer scientists should be able to find a lot of challenging research issues related to computational science and engineering. We present only a few below. The reader is referred to [Stev93, Stev94] for more examples and details.

- *Efficient portable parallel programming*. Parallel programming faces two conflicting goals. On the one side we want programs to be portable from one parallel computer to another, but on

the other side we want to use knowledge of machine-specific details to improve performance. New languages and improved tools may improve the situation.

- *Programming languages and libraries.* A lot of new languages (Fortran-90, High Performance Fortran, etc.) and libraries (Message Passing Interface (MPI), etc.) have been developed lately. There is no sign that this trend will cease in near future.
- *Parallel debuggers and environments.* Parallelism implies that many events occur simultaneously, debugging is more difficult, and programming environments are less mature.
- *Visualisation.* This field is strongly related to human-computer interface research.
- *Standard model for parallel algorithms.* We need a common standard model for teaching, analysing and sharing of parallel agorithms. A large number of models has been proposed, but we are still missing the "von Neumann model of parallel processing".

References

[BW93] Erik Bølviken and Ragnar Winther, *Databehandling og anvendt matematikk*, in "IT neste TI, Informasjonsteknologi de neste ti år" (in Norwegian), Petter Gottschalk (editor), DND og Gyldendal 1993.

[CSEP94] Computational Science Education Project, "E-book" (electronic book), Available through Mosaic or similar programs at `"http://csep1.phy.ornl.gov/csep.html"`.

[EC94] European Commission, Draft proposal for a Specific programme of Research and Technological Development in the Field of Information Technologies (1994-1998), *"Fourth Framework Programme"*, Brussels February 1994.

[Golu93] Scientific Computing, An Introduction with Parallel Computing. Gene Golub and James M. Ortega. Academic Press 1993.

[HPCN94] High-Performance Computing and Networking, Editors: W. Gentzsch and U. Harms, Volume I: Applications, Volume II: Networking and Tools, Springer Verlag, Lecture Notes in Computer Science no. 796 and 797.

[misc94] E-mail discussion following a conference on CS&E education in Albuquerque, Feb. 1994.

[Panc93] Cherri M. Pancake, The Changing Face of Supercomputing, IEEE Parallel & Distributed Technology, Nov. 1993, pages 12-15.

[Rice93] John R. Rice, *Academic Programs in Computational Science and Engineering*, Technical report 93-042, Computer Science Dept., Purdue University, 1993.

[Same93] Ahmed Sameh and John Riganati, *Computational science and engineering*, IEEE Computer / CS&E, Oct. 1993, pages 8-12.

[Stev93] D. E.. Stevenson, *Science, Computational Science and Computer Science: At a Crossroads*, Aug. 1993, 16 pages. (To appear in CACM)

[Stev94] D. E. Stevenson, *A Computational Science Manifesto*, June 1994, 33 pages.

[Thom93] Clark D. Thomborson, *Does Your Workstation Computation Belong on a Vector Supercomputer?*, Comm. of the ACM, Nov. 93, pages 41-49.

Filnavn: Nik94.doc