TASK QUARTERLY 4 No 2 (2000), 171–178

AN INTRODUCTION TO HIGH PERFORMANCE COMPUTING: TECHNOLOGY, TOOLS AND APPLICATIONS

JANUSZ S. KOWALIK

The Boeing Company,
PO Box 3707, MS 7L-44, Seattle, Washington 98124-2207, USA
and University of Washington,
Department of Computer Science and Engineering,
FR-35, Seattle, Washington, Seattle 98195, USA
janusz.s.kowalik@boeing.com

(Received 18 February 2000)

Abstract: Traditionally High Performance was applied to very computationally demanding problems in science and engineering. They were known as the Grand Challenge Problems that required supercomputers equiped with very large computer memories and capable of high rates of computation measured by Megaflops (Million Floating Point Operations per second). In the last two decades the world of business and industry has recognized the enormous potential of large distributed computer systems for their business enterprise applications. Today most of the existng high performance computers are employed in the enterprise environments. A typical system has three layers of servers: user interface, applications and database. These Client/Server architectures are the working horse of the large enterprise information processing. This introductory paper describes both areas of High Performance Computing applications that differ in the nature of workload, performance objectives, design methodology and scientific focus.

Keywords: supercopmputing, client/server systems, enterprise data processing systems, performance, parallel computing, cluster computing

1. Preliminaries

High Performance Computing (HPC) is a subdiscipline of Computer Science and Computing Technology dealing with theories, methods, tools and infrastructures for solving computationally most demanding problems in science, engineering and business. Traditionally HPC was applicable mainly to large-scale number crunching

problems and used a variety of advanced mathematical techniques and most powerful supercomputers. Typical applications included computational fluid dynamics, structural mechanics, weather and climate modeling, electric power distribution, design of nuclear weapons, and petroleum engineering, etc. In the last two decades of the century the HPC technology started to penetrate the area of large enterprise applications, such as: manufacturing resource planning, product data management, and other business processes required for running large industrial and business enterprises.

These two broad areas of HPC applications are scientifically and technologically distinct and will be described in the following sections of the paper.

2. Scientific/Engineering applications of HPC

2.1 Supercomputers

The supercomputer technology dates back to the mid-1960s and its development has been fueled by the computation and requirements of large scientific and engineering problems. The rapid progress of this technology can be illustrated by comparing the peak processing rate of CDC 6600 in 1964 which was 1Mflops (million floating point operation per second) with a current supercomputer CRAY T90 that can deliver 1.8 Gflops (billion floating print operation per second).

The early supercomputers derived their speed from vector processing which performed well for problems rich in matrix and vector operations. Later parallel computing has been added as a key technique for increasing computational speed. Some parallel supercomputers had vector processors other used RISC (Reduced Instruction Set Computers) processors. Parallel supercomputers with vector processors have offered two levels of parellelism, (*i*) large grain parallel tasks executed on different processors and (*ii*) low level parallelism executed by arithmetic pipeline units of vector processors.

The RISC processors have achieved their superior performance by fast clock cycles and by executing fundamental instructions in a single CPU clock cycle. They have become cost-effective building blocks for architecting parallel supercomputers with dozens or hundreds of processors. There have been also other advancements in supercomputer technology that contributed to faster processing: faster caches, larger prime memories, and better compiler code optimization techniques.

2.2 Supercomputers for a poor man

Low-cost personal computers (PC) linked by fast networks to form a coordinated cluster system can be a fast and inexpensive alternative for scientists who have no access to multimillion dollar supercomputers. It can also be a more convenient alternative to heavily loaded time-shared supercomputers. These collections of PCs are called Beowulf Clusters and may include hundreds of processors and impressive amounts of total memory. Beowulfs are programmed using FORTRAN or C and use MPI (Message Passing Interface) paradigm [1] to establish cluster communication and coordination.

An example of a successful Beowulf has been built at Penn State University. It is called Cost Effective Computing Array (COCOA) and has proved to be very cost effective. The system built to study complex fluid dynamics has 50 off-the-shelf PCs, 13 Gbytes of RAM and 100 Gbytes of disk space. An equivalent in power and size supercomputer would cost about $750,000.

In one application the users reported that COCOA took 127 seconds, the CRAY T3E 177 seconds and the SGI ORIGIN 2000 took 90 seconds. This is very impressive considering the prices of all three systems. The authors concluded that PC clusters could be very cost-effective for solving problems such as they dealt within aeracoustics and aerodynamics.

Technical details on COCOA are available at *http://cocoa.aero.psu.edu/.*

2.3 HPC software

2.3.1 Programming styles

From the programming point of view parallel computers fall into two major classes: (*i*) shared memory and (*ii*) message passing distributed memory. In the first case we deal with a single address space memory which can be accessed by different processors in the same fashion as in uniprocessor architecture. Physically shared memory can be centralized or distributed. For example in the NUMA (Nonuniform Memory Access) multiprocessors memories are physically distributed but still there is an illusion of a single shared memory.

In the message passing case programs running on different processors can access each others memories by explicit message passing mechanisms that store and fetch data. Thus the shared memory architectures are closer conceptually to conventional uniprocessors. One class of supercomputers using message passing programming style is clusters. They are often referred to as loosely coupled multiprocessors. Due to latency involved in internodal cluster communication it is usually required to run large-grain parallel codes for better efficiency.

2.3.2 Mathematical software

Application software is an important factor influencing achievable processing performance. This includes algorithms, data structures and programming techniques. Usually there are more than one algorithm to solve a numerical problem. The applicable algorithms may differ computational complexity, i.e. required number of arithmetic/logic operations and amount of storage. It is a good practice to identify an algorithm of least complexity applicable to a problem at hand and attempt to implement it. But this is insufficient for achieving peak performance. We must carefully consider implementation issues such as computer architecture and programming techniques related to the specific implementation computer configuration.

Important implementation issues include handling data movement and recognizing hierarchical memory structure of a particular supercomputer. A good

implementation should maximize local processing with a minimum data transfer to remote memories in supercomputers that have local and global memories. In computers whose shared memory is divided into shared banks avoiding memory bank conflicts contributes to better performance. In general the idea is to maximize processing and minimize data movements to and from memory and between different memory levels.

A very simple but convincing example of memory management can be found in [3]. The example shows that an appropriate programming approach results in the I/O time reduction from many days to seconds.

There is an abundance of experimental data showing that memory management is a key factor influencing supercomputer performance. In the message passing architectures the cost of message transfer has to be taken into account. Even in shared memory machines such as NUMAs accessing remote data may be prohibitively expensive and seriously impact the overall performance.

It is fortunate that at the current state of the supercomputer technology we not only have good understanding of algorithmic efficiency and high performance programming practices but also have access to high quality mathematical software that performs very well across most of the commonly used types of supercomputers. These highly optimized software libraries include:

(1) LAPACK (Linear Algebra Package) for shared memory supercomputers, and

(2) ScaLAPACK similar to LAPACK but applicable to distributed memory machines using message passing.

The software and documentation can be retrieved from netlib *http://www.netlib.org*. The ability to access and use these high quality libraries is a significant benefit for the scientific and engineering community involved in practical application of supercomputers. Also researchers can focus on domain issues related to their discipline of science or technology rather than struggle with numerical analysis and programming problems. This is not to say that enlightened users should not study and understand mathematical and programming aspects of the software that is used.

Quite opposite is true – a better understanding of algorithms and their implementation could help in establishing boundaries between what is practically computable and what is not at the current state of the art in supercomputing.

In the spirit of this principle we recommend an excellent book [3] which explains the building blocks of numerical linear algebra and is a practical introduction to solving linear equations and linear eigenvalue problems. The book discusses solution approaches to large sparse linear equations that arise in many applications. It can be used as a reference book or a textbook for self-education or in a classroom environment. It contains a wealth of benchmark material and programming techniques that help to design efficient software for problems in and outside linear algebra.

2.4 Performance analysis

Here we present a rudimentary analysis of parallel processing performance. Let us assume that α is the percentage of a program's sequential running time that can be executed in parallel by p processors. If T_1 denotes the processing time by a uniprocessor and T_p by a parallel machine with p processors then a conventional speedup measure is:

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{\alpha T_1 / p + (1-\alpha) T_1}.$$

Eliminating T_1 and multiplying by p gives:

$$S_p = \frac{p}{\alpha + (1-\alpha)p}.$$

This rather pessimistic results is called the Amdahl's Law and says that speedup is limited by the nonparallelizable portion of the code $1-\alpha$. For example, if only half of the code can be parallelized then the speedup will not exceed 2 no matter how many processors we use.

The basic assumption of Amdahl's Law is that α remains constant regardless of the problem size. In reality large problems have higher degree of parallelism and when we get a supercomputer with large number of processors we tend to solve larger instances of problems.

Now let us assume that the time for parallel computation is α and for sequential computation in $1-\alpha$. The total time is 1. The time for solving the problem on uniprocessor is:

$$T_1 = (1-\alpha) + \alpha p.$$

The speedup is:

$$S_p = (1-\alpha) + \alpha p,$$

and for α close to 1, *i.e.* for a highly parallelizable problem $S_p \approx \alpha p \approx p$.

Which is much more satisfying result derived originally by Gustafson [4].

We have only scratched a surface of the parallel computing performance. Discussions of parallel computing in depth are contained in [5]. The author, David Kuck, introduces concepts of performance stability, scalability and measurement. The book is one of the most complete analytical treatments of high performance computing.

3. Enterprise applications of HPC

3.1 The HPC Enterprise Problem

We consider two versions of the Problem, (*i*) Performance Tuning and Improvement, and (*ii*) Enterprise Systems Design.

In the first case there exists already a distributed client/server system running enterprise applications but it requires modification if its performance is inadequate or

if new workloads need to be implemented. In the second case we have only a workload model and required performance levels, but no system exists. We face a problem of design from scratch. Of the two cases this is more difficult. To design by trial and error would be impractical. Solving a pseudo optimization problem: design a cost-effective system capable of processing the given workload subject to performance constraints can be computationally expensive considering modeling complexities.

We should also mention that a typical workload is a combination of batch processing and transaction processing. The number of servers may be as high as several hundreds, and the number of users over 100,000. In addition to technical difficulties in solving problem (*i*) or (*ii*) we may have other obstacles on our way to a satisfactory solution:

(a) workload predictions tend to be uncertain and inaccurate,

(b) there may be lack of data which is needed for analytical or simulation models,

(c) vendors of hardware and software may have no experience in architecting complex large system,

(d) in multi-vendor environments no individual vendor may be willing to assume responsibility for integrating the entire system.

3.2 The proposed approach

Given the high complexity of the HPC Enterprise Problem we have to use a systematic and accurate method for solving it. The method should be able to take into account all major factors influencing performance. There are many of them as shown in Figure 1.

A method of building and exercising simulation models has been successful for architecting and tuning client/server systems at Boeing [6].

A key to successful system modeling is accurate information describing workloads, and H/W, S/W system characteristics, Figure 2.

We have used Workbench™ and Strategizer™ as our simulation modeling tools. Both are from HyPerformix.

Occasionally it is possible and beneficial to construct and use analytical models for simplified parts of layer systems and take advantage of analytical closed form solution. Such analysis is recommended for preliminary studies which now detect problem areas and point to system subcomponents deserving more detailed simulations.

It is unreasonable to assume that any specific modeling tool is perfect. All have limitations and weaknesses. A way to overcome this is by using an array of tools as appropriate for a problem at hand.

General hardware/software fit; software awareness of the hardware architecture and configuration
Workload characteristics, intensity and time dependence
Application software, programming style, algorithms and data distribution
Operating systems including job schedulers and load balancing algorithms, DBMS
System overall topology, application and database servers, communication speeds and bandwidth

Figure 1. *Factors influencing performance of distributed enterprise systems*

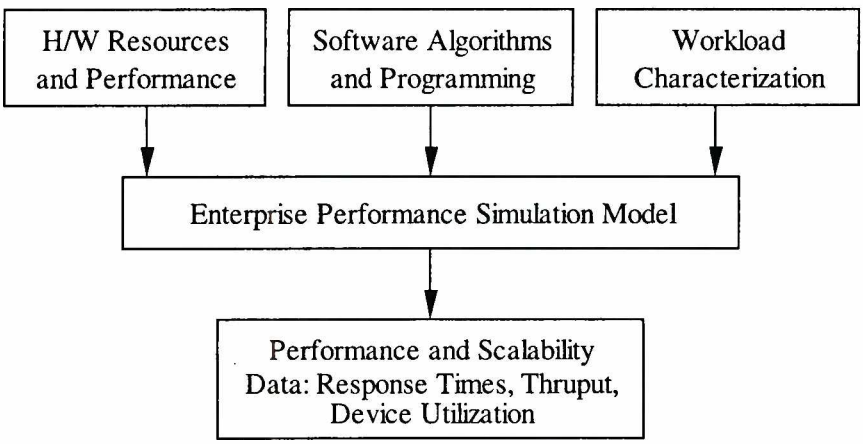

Figure 2. *Input to and output from the enterprise performance simulation model*

3.3 What needs to be done

The problem of designing and running large distributed systems is far from being fully understood and mastered. Many large industrial and commercial companies are struggling to implement enterprise resource planning (ERP) and product data management (PDM) solutions. It would be helpful to divert some research talent and resources from the traditional scientific HPC to enterprise HPC. The rewards for those who will master this technology and use it for improving their competitive position would be significant.

We need research to improve methods and tools for designing and tuning large enterprise systems. The area of modeling economies of designed IS is wide open for innovative ideas and methods.

4. Conclusions

In this paper we attempt to sketch two different areas of HPC applications: the scientific HPC and the Enterprise HPC. What they have in common are: general notions of performance, parallel processing and often hardware that can serve both applications. An area of problem's definition where two applications significantly differ is the technical problem objective. In scientific HPC we are after fast and efficient algorithms for mathematically defined problems. In enterprise HPC we are synthesizing a computing infrastructure that serves geographically distributed users. We refer interested readers to [7] which describes an approach to enterprise computing architecture combining simulation modeling, rule-based reasoning and heuristic classification.

References

[1] Gropp W., Lusk F. and Skjelllum A., *Using MPI: Portable Parallel Programming with the Message Passing Interface*, The MIT Press, Scientific and Engineering Computation Series, 1994

[2] *How to Build Beowulf*, The MIT Press, Scientific and Engineering Computation Series, 1999

[3] Dongarra J., Duff I. and Van der Vorst H. A., *Numerical Linear Algebra for High-Performance Computing*, SIAM Press, 1998

[4] Gustafson J., *Reevaluation of Amdahl's Law*, Comm. ACM, **31**, 531, 1988

[5] Kuck D. J., *High Performance Computing*, Oxford University Press, 1996

[6] Aries J., Banerjee S., Brittan M., Dillon E., Kowalik J. and Lixvar J., *Capacity and Performance Analysis of Distributed Enterprise Systems*, submitted to Comm. of ACM

[7] Nezlek G. S., Jain H. K. and Nazareth D. L., *An Integrated Approach to Enterprise Computing Architecture*, Comm. ACM, **42**, 82, 1999