Parallel Processing Laboratory Experiments for Undergraduate Students

Muhammad Ali Mohammad Javed Khan Daniel Nyatuame

Tuskegee University

Abstract

This paper explains the approach at Tuskegee University for consolidating high performance computing fundamentals and concepts taught to undergraduate students through a series of laboratory experiments. The computing environment is a PC-based cluster with a Linux operating system and running MPICH. A real world problem of grid-generation for computational fluid dynamics has been utilized to bring out the practical aspects of high performance computing through a set of laboratory experiments.

Introduction

Tuskegee University is in the process of developing an academic program in the area of High Performance Computing (HPC). This effort has been broadly divided into three areas i.e. coursework, laboratory infrastructure and experiments to reinforce theoretical concepts. A HPC course has been developed as a technical elective for undergraduates from both science and engineering departments. This course has a prerequisite of an introductory programming course. Details of the course are available in [1]. The laboratory infrastructure is based on commercial off-the-shelf (COTS) PC-server hardware. The Linux operating system chosen for the PC-clusters has allowed access to considerable software resources in the public domain. This approach has made HPC an affordable activity at minority institutions such as Tuskegee University. This paper will restrict itself to the development of experiments for students to understand parallel computations through hands-on experience.

The primary reason advanced in favor of parallel processing is performance, which is normally translated into the single metric 'total execution time'. This single metric however consists of a number of elements e.g. computation time, communication time etc. Thus impacts of memory requirements, algorithmic efficiency of computation (numerical) and parallelization techniques need to be understood.

A series of experiments have therefore been developed to identify and understand various aspects of the parallel computational environment. Since the HPC course is intended not only for computer science majors, but also targets engineering majors, the experiments are based on a simple (yet non-trivial) problem of grid generation in Cartesian coordinates, a typical activity in computational engineering approaches in a variety of disciplines such as fluid dynamics and structural mechanics etc.

Objective

This paper explains the approach adopted at Tuskegee University for developing the laboratory element of an undergraduate course in High Performance Computing (HPC) in a PC-based cluster environment.

Computational Environment

The HPC resource is centered on a Beowulf cluster computer with 4 server nodes. Each server node is a SMP with two 966MHz processors, 256MB of RAM and 20GB Hard disk. Inter-connect between the cluster nodes is a100Mbps Ethernet. The Operating System is Linux 6.2, with several patches for optimal performance on a cluster computer. The cluster computer can be accessed from other departments in the university.

The software package has both commercial and Open Source software, and configured for optimal performance on a cluster computer. The commercial packages include, Portland Group Workstation 3.2 Compilers and MPI-PRO Professional Communication Library. There are two other open source implementations of MPI and PVM. Also included are ScaLAPAK, PVFS, PBS, SCA Linda, Dogsled Administration and Monitoring Tool.

In addition, an effort is underway to build a cluster computer, using OSCAR (Open Source Cluster Application Resource) developed by Oak Ridge National Labs.

Problem Environment

In addition to the typical programming exercises given in a parallel programming course, we at Tuskegee decided to use programming experiments, which firstly would be more exciting and motivating to engineering students and secondly could be expanded upon in a logical manner in subsequent courses. The obvious choice was selecting a computational fluid dynamics (CFD) related experiment since CFD has perhaps had the greatest influence in the development of HPC.

A large number of CFD techniques are based on transforming the governing differential equations into finite-difference equations, and a grid of points needs to be identified in the problem (physical) domain where the finite-differences are evaluated. This process referred to as grid generation is used in other computational techniques, such as the finite element method (FEM), as well. Usually, the physics of the problem dictates a non-uniform grid e.g. near solid boundaries or shock-wave regions where extreme gradients in the physical properties (pressure, temperature, velocity etc) exist, necessitating a fine mesh in these regions to resolve the rapid changes. The uneven grid spacing makes evaluation of the finite-difference quantities complicated. One of the approaches is to transform the

non-uniform grid in the physical domain into a uniform computational grid, and the flow equations, suitably transformed, are solved in this 'computational domain'. The first step then in a CFD solution is the process of 'grid-generation'. In fact grid-generation over the years has spawned into an industry by itself. It was therefore chosen as the programming exercise to illustrate aspects of HPC.

The physical problem chosen was one-dimensional viscous flow in a two dimensional duct. The experiment was to generate a structured grid with refinement near the solid boundaries to capture the velocity gradients normal to the duct surfaces. To achieve this an algebraic grid generation transformation from the computational domain to the physical domain was to be utilized. The transformation relationship between the coordinate normal to the duct in the physical domain (y) and computational domain ($\bar{y}$)is given by (Ref. 1):

$$y = (h)((\beta+ 2\alpha)[(\beta+1)/(\beta-1)]^{(\bar{y} - \alpha)/(1 - \alpha)} - \beta+2\alpha)/((2\alpha+1)\{1+[(\beta+1)/(\beta-1)]^{(\bar{y} - \alpha)/(1 - \alpha)}\})$$

where the stretching parameter β is related to the boundary layer thickness δ and the distance h between the two plates by the relation:,

$$\beta = (1 - \delta/h)^{-1/2} \qquad \text{for: } 0 < \delta/h < 1$$

A value of $\alpha = 1/2$ will result in the refining of the grid near both the walls while for $\alpha = 0$, the refinement will be only near one wall.

In view of the large number of grid points in a realistic physical problem, visualization of the generated grid is an important element so as to ensure the correctness of the grid. For this purpose COTS visualization software will be utilized.

Experiments

The first experiment is aimed at making the students cognizant of the fact that to ensure optimum efficiency, performance of the nodes should be comparable. Otherwise overhead penalty in the form of idle-time would be a consequence. This experiment requires writing of a parallel program, which simply executes a loop on all the processors and notes the compute time on each processor in the presence of un-related activity on the nodes. This allows detection of possible major variations in processing time. Since all nodes are similar, such a variation would indicate unexpected resource consumption. All unnecessary activities would therefore have to be stopped on the cluster nodes before carrying out the subsequent laboratory experiments.

It is generally presumed that parallelizing a serial program would result in reduction of total execution time proportional to the number of processors used. However, this is not always true. In applications with a high content of file I/O, there may be negligible decrease in total execution time despite the parallelization of the problem.

The second experiment is therefore designed to show the effect of I/O on the total execution time. The students will be required to write a serial program, which generates grid-points and outputs the data in text or binary format, as required.

The third experiment is the parallelization of the above, to determine the effect of I/O and computation loads on the execution time of a parallelized program. Applications with a high content of inter-process communication may not benefit significantly from the parallelization. Students would expect considerable reduction in program execution times after parallelizing. However, the results of their experiments will be counter-intuitive. The reason for this of course is that the time for writing out data to a text file is very high compared to the computation time of the grid generation algorithm chosen for the experiment. Thus, reduction in the computation time due to parallelization does not significantly impact the total program execution time. The students will be asked to analyze this apparent contradiction. Hints could be given to the students to guide them in the right direction, but they would be required to individually and collectively analyze the problem, and determine a course of action to collect the relevant data. This entails determining the computation time in addition to the total execution time, and would require modification of the codes to capture these. Finally, to capitalize on the benefits of parallelization, the students would be expected to use binary output of data.

Fig. 1 which shows the computation times and overheads of the serial and parallelized program, with respect to the number of processors the parallel program is executed on, captures basic characteristics of parallelization. These times are plotted for the three cases of text-file output, binary-file output and no-file output. Overhead defines as the difference between the total execution time and computation. It includes inter-process communication and file I/O overheads. While the computation time remains the same in all the three cases, the overheads with text output are an order of magnitude higher than the overheads with binary file output. However, the overheads for binary file output are slightly more than the overheads without file output.

Ideally, the computation time should decrease by half when the number of processors is increased from one to two. Fig. 1 shows a considerably higher decrease. The reason for this apparent anomaly, which the students will be expected to comment on is that wall-time rather than CPU usage time is being clocked, and for the serial program, the wall-time for computation code includes the time during which the computation code is interrupted to perform administrative activities of the operating system. This is a depiction of what students will encounter in real life. Increase in overheads with parallelization is also observable in Fig. 1. This increase is due to the inter-process communication and MPI layer. The decrease in computation time should be significant enough to compensate for this increase in overhead. Insensitivity of the computation time with the increase in the number of processors can be seen in Fig. 1, which is also counter-intuitive. This characteristic of the computation time has been purposely built into the program logic by requiring that it allocates a maximum of 500,000 grid points per processor. For the total number of grid points of 1,000,000 as in this example, no allocation is therefore made to the third and subsequent processors. For much larger problems this inefficiency is eliminated as is seen in the next experiment underscoring the importance of load balancing amongst the processors.

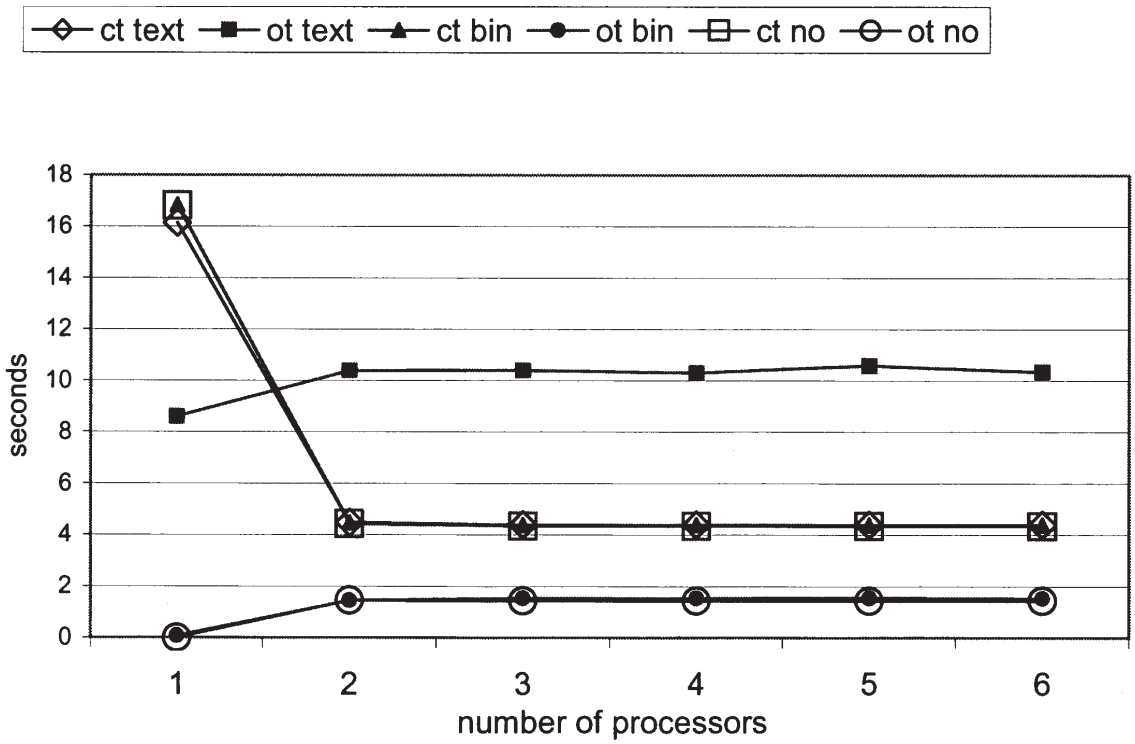


Fig 1. Computation(ct) and Overhead(ot) times with text, binary and no output: 10^6 grid points

Figure 1.

The fourth experiment explores the effect of computational intensity (e.g. flow-field calculations) per grid point. This experiment requires execution of the serial and parallel programs several times with a range of parameters. Figs. 2 and 3 bring out the effect of low and high computational intensity per grid-point computation. When the computation per grid-point is low, there is an increase in total execution time despite the decrease in computation time. This is due to the increase in parallel computation overheads. With high computation per grid-point, the decrease in computation time is enough to compensate for the increase in parallelizing overhead. Figs. 4 and 5 show that, with bigger problem sizes, the performance is better. However, an asymptotic behavior is observable signifying diminishing returns.

Summary

The teaching of HPC can be reinforced with appropriately designed laboratory experiments based on 'real life' applications. The problem of CFD grid-generation has been used as basis to bring out certain minimal but essential aspects of HPC. The experiments designed for undergraduate students at Tuskegee University lucidly bring out the need to understand:

(a) Execution times of the various parts of the code prior to investing into parallelization of the problem
(b) Effect of overheads on performance
(c) Influence of various parameters of a problem

An important element of HPC i.e. load balancing has not been addressed as part of present design due to the inherent uniformity of the structured grid. For this class of problem, the concept of load balancing would have required the next logical step of solving the flow physics equations on the computational grid. This aspect is planned for implementation at a later stage.

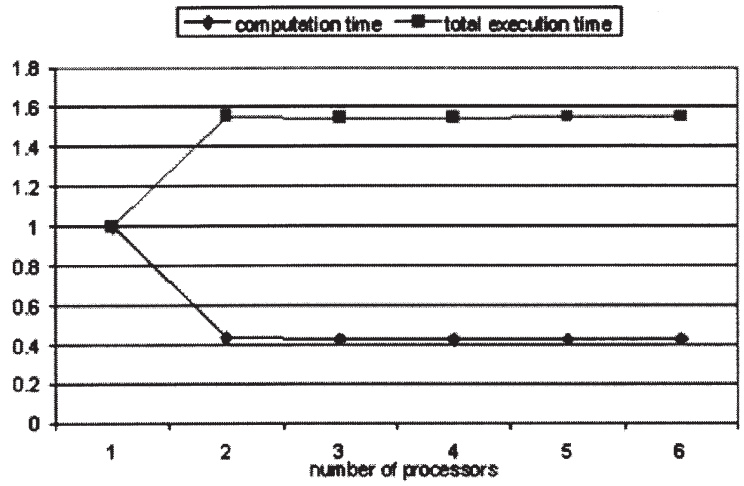


Fig 2. Ratio of parallel to serial computation and total execution times: 10^6 gridpoints Low computation per gridpoint

Figure 2.

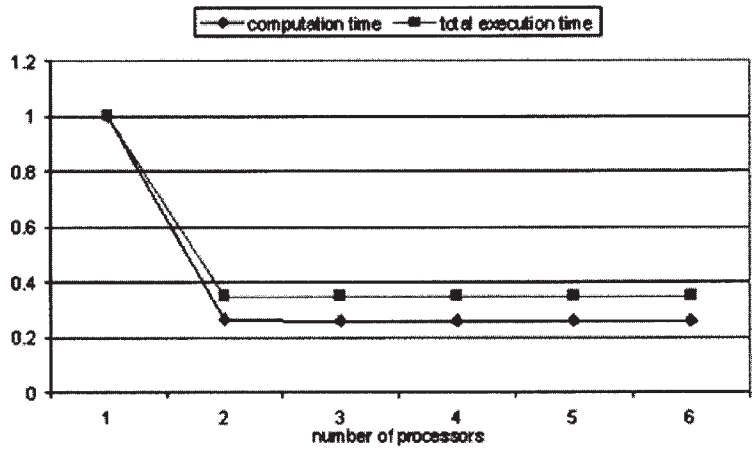


Fig 3. Ratio of parallel to serial computation and total execution times: 10^6 gridpoints High computation per grid point

Figure 3.

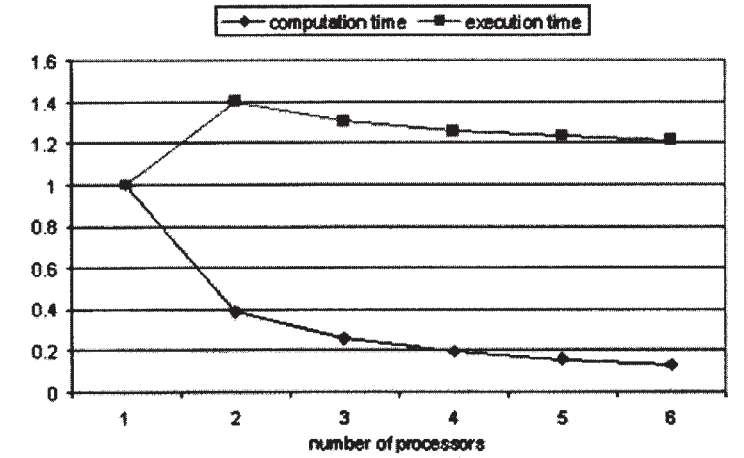


Fig 4. Ratio of parallel to serial computation and total execution times: 10^9 gridpoints low computation per gridpoint

Figure 4.

References

1. Ali, M., Development of a High Performance Computing Course With Low Cost Infrastructure, The Symposium on Computing at Minority Institutions, ADMI 2002, May 30th, 2002, Stetson University, Florida

2. Anderson, D. A., et al., Computational Fluid Mechanics and Head Transfer, McGraw Hill, 1984.

3. Pacheco, P. S., Parallel Programming with MPI, Morgan Kaufmann Publishers, 1997

4. Wilkinson, B., Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers, Prentice Hall, 1999.

5. Foster, I., Designing and Building Parallel Programs, Addison Wesley, 1995

6. Snir M., et al., MPI – The Complete Reference, The MIT Press, 2001

7. Gropp, W., et al., Using MPI: Portable Parallel Programming, 2nd Ed., The MIT Press, 1999

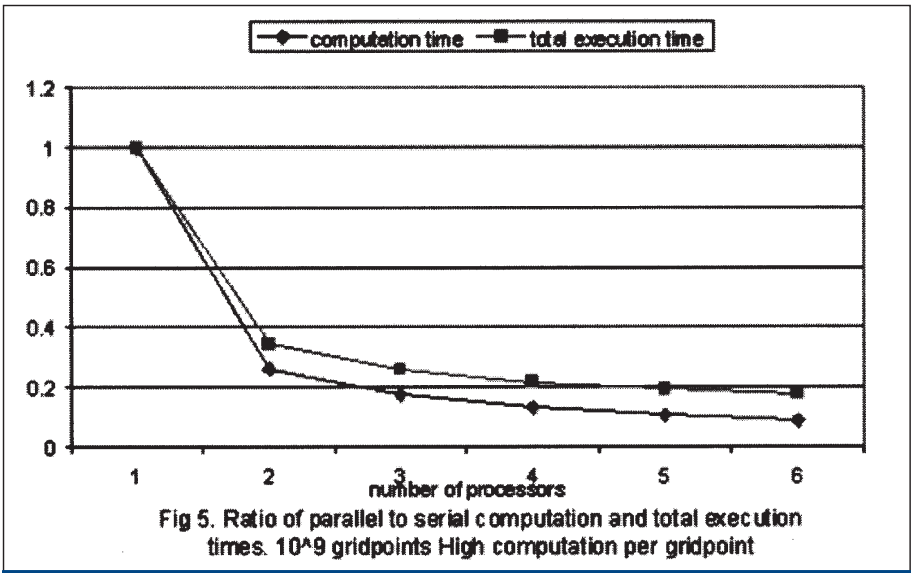
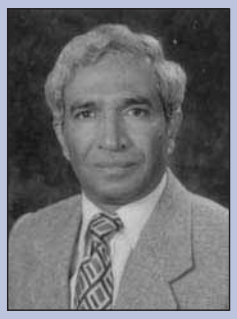


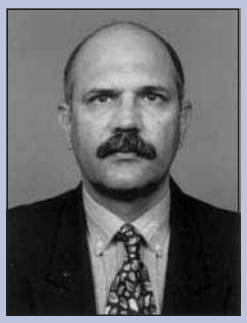
Fig 5. Ratio of parallel to serial computation and total execution times. 10^9 gridpoints High computation per gridpoint

Figure 5.

Muhammad Ali is Assistant Professor in the Dept of Computer Science, Tuskegee University. His area of interest is High Performance Computing, focusing on developing undergraduate courses and experiments, based on low cost laboratory infrastructures. This is of significance to smaller institutions, including minority institutions. He received his DSc(Computer Science) from George Washington University, USA, and MSc(Electronic Eqpt Design) from Cranfield Institute of Technology, UK. He has a BE(Avionics) degree from Karachi University, Pakistan. He is member of Council for Undergraduate Research, and IEEE.



Mohammad Javed Khan is an Associate Professor in the Aerospace Science Engineering Department at Tuskegee University. He has a BE in Aerospace Engineering from Karachi University, Pakistan, an MS in Aeronautical Engineering from the United States Air Force Institute of Technology and a PhD in Aerospace Engineering from Texas A&M University. His research focus is fluid mechanics, aircraft design and introduction of innovative and motivating learning techniques in the classroom. He is a fellow of the Royal Aeronautical Society, UK and member of the American Institute of Aeronautics & Astronautics and American Society of Engineering Education.



Daniel Kwame Nyatuame received the B.S. E.E degree from the University of Science and Technology, Kumasi, Ghana, in 1992, the MBA MIS degree from the Vrije Universiteit Brussel, Brussels, Belgium 1997, and is currently pursuing M.S. degree in Electrical Engineering at Tuskegee University, Tuskegee, AL since 2001.

He worked as Research Assistant with the Department of Computer Science, Tuskegee University, during Summer 2001 and 2002. He was involved in research on High Performance Computing (HPC). Daniel was part of the team that set up a HPC infrastructure based on a Beowulf Linux Cluster and was involved in installation and configuration of various software as well as writing programs to use for performance analysis on the cluster.