THE RIGHT BALANCE: RESTRUCTURING THE PARALLEL AND SCIENTIFIC COMPUTING COURSE*

Lubomir Ivanov
Department of Computer Science
Iona College, New Rochelle, NY 10801
Tel.: 914-633-2342
Email: livanov@iona.edu

ABSTRACT

This paper outlines material presented in the revised offering of the Parallel and Scientific Computing course at Iona College, NY. The course pursues a balanced approach between parallel computing topics and their scientific computing application. The paper describes a number of projects assigned to students throughout the semester which serve to emphasize this balance while developing students' parallel algorithmic and problem solving skills. Finally, the paper discusses some of the challenges and benefits of offering the Parallel and Scientific Computing course.

1. INTRODUCTION

With the advent of multi-core, hyperthreaded CPUs and high-performance GPUs, parallel computing has moved into the mainstream: Powerful parallel hardware resources are now widely available to the average user for a fraction of the cost of a supercomputer. Modern operating systems are designed to recognize and efficiently utilize the parallel hardware, while software applications are increasingly taking advantage of the available parallelism to carry out computation faster and more efficiently. As everyday computer users become more and more dependent on high performance, scientists, emboldened by technological developments, are attempting to solve ever more challenging and computationally demanding problems: From protein folding to weather prediction, from aero-/hydrodynamics to seismology, the scientific community employs parallel systems to tackle questions that, until recently, were thought intractable.

---

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

The rapid advances in parallel hardware and software and the ever-greater need for computational speed have led to an increased demand for trained professionals with skills to develop parallel systems and assist in the use of these systems for solving challenging scientific problems. Unfortunately, the Academia has been somewhat slow in answering this demand: To achieve true proficiency students must encounter parallel computing topics in just about every course in the Computer Science curriculum. Moreover, students should gain a solid understanding of the uses of parallel computing in scientific applications. Thus, parallel computing should be a part of not only the Computer Science curriculum, but that of the Natural Sciences and Mathematics as well. Regrettably, this is kind of all-around exposure to parallel computing and its applications is very rare. Usually, the limited experience which students gain with parallel and scientific computing topics comes from a Parallel Computing course which many departments offer. It is, therefore, imperative, that the course provides a solid and balanced foundation for the further development of students parallel computing skills.

In this paper we present an outline of the latest offering of the Parallel and Scientific Computing course taught at Iona College, NY. We discuss the topics covered in the course and the assignment handed out to students. We emphasize the balance between parallel computing concepts and their scientific computing applications, between traditional techniques and methodologies and the latest developments in this fast paced area of study. We also consider the impact of the course on student education and outline some of the challenges in developing and offering a modern Parallel and Scientific Computing course.

2. THE PARALLEL AND SCIENTIFIC COMPUTING COURSE

2.1 Course Objectives

The main goal of the course is to provide students with a solid foundation for building more advanced skills in parallel and scientific computing. We are fully aware that it is impossible to cover all aspects of parallel computing in the time span of a single one-semester course. Instead, we attempt to provide a balanced treatment of fundamental theory, hardware and software design and implementation to familiarize students with the basic concepts of parallel computing and begin to develop their parallel problem-solving mode of thinking. The specific course objectives are:

- Discuss the benefits, advantages, and drawbacks of parallel computing
- Explain how various parallel computer architectures execute parallel software
- Get students accustomed to estimating the performance increase due to parallelizing the solution to a computational problem.
- Emphasize the design and implementation of parallel algorithms for solving computational and/or scientific problems.
- Introduce modern parallel software techniques, languages, and libraries.

2.2 Course Topics

Parallel computing is one of the most dynamic areas of Computer Science. It is, therefore, imperative that the course material reflects the latest developments in parallel hardware and software design. Each time our course has been offered, the instructor has had to make significant adjustments to the topics covered in the course to accommodate

new advances in parallel computing. This paper reflects the material covered during the latest offering of the Parallel and Scientific Computing course. The course had an enrollment of nine students and met once a week for fifteen weeks for three hour evening sessions. The course prerequisites are Data Structures and Algorithms and Computer Architecture.

The course began with a broad overview of complex scientific problems which require the use of parallel computing to achieve a solution in reasonable time. Many of the specific problems discussed are derivatives of the N-body problem: astronomical simulations, protein folding and other molecular dynamic studies, sub-atomic particle interactions, etc. Other problems we considered included MRI analysis, hydro- and aero-dynamic computations, weather prediction, seismic analysis, and cryptography. Results of real scientific studies were demonstrated through videos and images of simulations of real astronomical and bio-chemical systems. This broad introduction to scientific computing was intended to stimulate student interest and to motivate the study of parallel computing as an aid in solving challenging scientific problems.

Next we turned our attention to the issues of parallel hardware systems. Since Computer Architecture is a prerequisite for the Parallel Computing course, students were already familiar with concepts such as pipelining, hyperthreading, multicore CPUs, and cache coherence. The focus in this course is less on the internal structure and design of such components but rather on the integration of these elements into a parallel system capable of performing massively parallel computation. Flynn's taxonomy was briefly introduced, followed by a discussion of the main differences, advantages and disadvantages of shared-memory, message-passing, and hybrid systems. Emphasis was placed on learning to recognize the suitability of a particular computational model for solving specific types of scientific problems. The role of static and dynamic interconnection networks, cache memories, and GPUs in carrying out massively parallel computations was also considered.

An important aspect of parallel software development is learning to estimate the expected computational speedup as a function of the number and type of hardware resources on which the software will execute. To this end, we introduced certain tools for performance analysis (speedup, sizeup, and efficiency computation techniques) and discussed the implications of Amdahl's law and Gustafson's law for parallel computing. Students learned to calculate the optimal number of processors needed to solve a computational problem of a given size, and, later, apply that knowledge in evaluating the performance of every parallel program they were asked to implement.

The main portion of the course focused on two large areas of parallel software development - CUDA C programming for NVIDIA GPUs and MPI C programming for a homogeneous parallel cluster. CUDA (Compute Unified Device Architecture) was introduced by NVIDIA a few years ago, and has since been implemented in a number of high-end graphics cards including the TESLA series (C1060, C2050, C2070, S2050, S2070). CUDA cards are specifically designed for high-performance computing. The CUDA architecture allows tens-of-thousands of threads to be launched concurrently, each performing an identical (to all other threads) computation on its own portion of the data set loaded in the GPU's memory hierarchy. The CUDA C library provides an extension of the C programming language for developing GPU accelerated parallel computing software. The basic structure of a CUDA C program involves allocating space on the

GPU, copying data from the host (the CPU) to the GPU, executing a GPU "kernel", which the NVIDIA runtime executes concurrently on all threads, and finally copying the results of the computation back to the host for final processing and display/storage. The Parallel and Scientific Computing course introduced students to the fundamentals of CUDA C programming using our recently purchased high-end machines with TESLA C1060 GPUs. Through examples and projects, students learned to optimize the execution of their programs using the GPU's global and shared memories, gained familiarity with thread synchronization and experience in using atomic operations for concurrent access to shared resources.

The Message Passing Interface (MPI) is a message-passing library which has become the de-facto standard for developing high-performance parallel scientific simulations on clusters and supercomputers. The MPI standard has been implemented for a variety of hardware architectures and interconnection network. MPI programs scale easily to larger-sized problems, and offer a rich collection of message-passing communication and cooperative-computation operations. A typical MPI program reads the input data through a single process (usually the "rank 0" process), which then distributes the data to all other processes. Every process determines which portion of the data it is responsible for and carries out the required computation concurrently with the other processes. The results are then either "gathered" or "reduced" on the rank 0 process, which displays/stores the final results. The material presented during our course lectures introduced the most important aspects of MPI programming - message send and receive, broadcast, gather/scatter/allgather, reduce/allreduce operations, data packing/unpacking and custom data type creation, communicators, etc. Students gained hands-on experience with these mechanisms while writing message-passing programs on the departmental cluster consisting of a front node and five backend compute nodes with dual XEON processors per node.

The last portion of the course was dedicated to advanced parallel computing issues such as load balancing, cache interference, and overlapping computation and input/output. Real world examples of load balancing (ORB partitioning, costzones, etc.) were considered along with example of how optimizing parallel code can lead to significant performance improvements.

2.3 Student Projects

The use of the parallel computing material presented throughout the course was illustrated with a number of in-class code examples. Based on those, students were asked to design and implement parallel software solutions to a number of scientific problems:

- *Brute-force N-body simulation*: The project asked students to provide a serial implementation of the $O(n^2)$ N-body simulation algorithm.
- *Barnes-Hut N-body simulation*: Students implemented a serial version of the Barnes-Hut O(n log n) algorithm. They were asked to compare the running times of the brute-force and Barnes-Hut simulations for N-body systems of different sizes, and comment on the results.

- *Parallel search*: For their first CUDA C project, students were asked to parallelize the simple Linear search algorithm and compare the running times of the serial and parallel versions on different-sized arrays of randomly generated integers.
- *The Mandelbrot set*: Students were asked to write a parallel CUDA C program to generate the Mandelbrot set. The project illustrated embarrassingly parallel computing and the handling of 2D arrays in CUDA C. Students were asked to estimate in advance the optimal number of blocks and threads per block and to estimate the theoretical speedup achieved when compared to the serial program execution. Comparison of the theoretical predictions and the actual running time measurements were also required.
- *Monte-Carlo approximation of* π: This seemingly simple CUDA C project introduced a number of complex issues: generating random number on the GPU device, barrier synchronization, and reduction computations. As always, students were asked to determine the optimal number of blocks/threads, estimate the expected speedup, then compare the actual running time of the parallel program with the theoretical prediction (and with the running time of the serial version).
- *Numerical integration approximation of* π: This time students had to write an MPI-based implementation of trapezoidal integration and use it to estimate the value of π. They were expected to compare the performance of the numeric and the Monte-Carlo approximation methods and comment on their efficiency.
- *1D continuous Cellular Automata simulation*: Student were expected to write an MPI-based parallel simulation of a 1D continuous CA. For extra credit, student had the option to also implement the simulation in CUDA C and compare the performance of the two implementations.

The *final exam* was a two-week project - the Heat Distribution problem: Given a thin metal plate with a uniform initial temperature onto which heat sources of fixed temperature are introduced at specific spots, simulate the temperature propagation throughout the plate over time (Fig. 1). Students were required to implement a serial, a parallel CUDA-C, and a parallel MPI version of the simulation. For the parallel versions, students had to estimate the optimal number of processes or threads, and compare the theoretical speedup with the actual running time measurements.

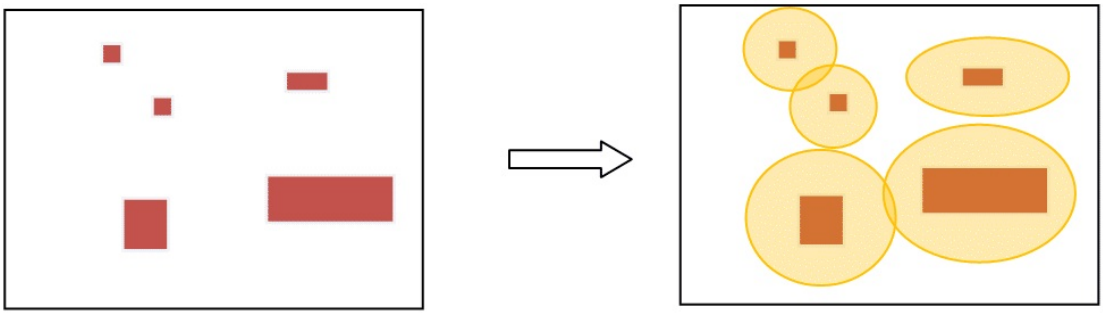

Fig. 1 Heat Distribution Problem

3. CHALLENGES AND BENEFITS

Offering a course on Parallel and Scientific Computing is a challenging endeavor. For the instructor, the main challenge is selecting an appropriate set of topics to cover in

the course. This is no easy task as the field of parallel computing is vast and ever changing. Some instructors choose to emphasize parallel computing theory or to present a more hardware-oriented or software-oriented material. In an effort to provide a more balanced coverage of parallel computing topics and scientific applications to which parallel computing is applied we opted for breadth of topics rather than real depth in a specific narrow area. We believe that this broad foundation better serves the interests of our students as it provides an overall perspective of the parallel and scientific computing and familiarizes students with the latest advances the field. Of course, the instructor must also be willing to adapt to new developments, learn new methodologies, and become familiar with upcoming technologies. Selecting interesting, representative, yet relatively simple scientific problems with which to illustrate the application of parallel computing is a challenging task in its own right. The instructor must be well versed in at least some aspects of the scientific problem areas being presented in the classroom. Inviting a guest lecturer or co-teaching a course with a professor from the Natural Sciences may be interesting alternative which students usually find exciting.

The Parallel and Scientific Computing course requires a significant investment in technology on behalf of the department: A high-performance cluster or a set of high-end machines with NVIDIA GPUs can cost tens of thousands of dollars. A cheaper alternative is to create a cluster by interconnecting old machines and installing the MPI middleware. Under the guidance of a faculty member, this is an excellent project for the Computer Science club or for students interested in getting a more hands-on experience with parallel systems. Yet another alternative is to apply for an educational grant which provides supercomputing time at an NSF supercomputing center. Finally, MPI can be installed even on stand-alone machines, but, of course, the advantage of true parallel computing is lost.

For students, the Parallel and Scientific Computing course is one of the most challenging experiences, which requires that they harness all knowledge and skills accumulated in other courses such as Data Structures and Algorithms, Computer Architecture, Operating Systems, Programming Languages, Calculus, Physics, Chemistry, and Biology. Sometimes, students lack the required background and have to learn new concepts "on the fly", which adds additional stress to an already challenging course. More significantly, the course requires that students adapt to a new way of thinking, a new approach to problem solving, quite different from what they have been trained by all the other courses they have taken. This shift requires a significant investment in time and effort: On their evaluations, many students noted that the Parallel and Scientific Computing was among the most time consuming courses they have taken at Iona. However, many students also commented that they found the course challenging but stimulating and rewarding. Completing a difficult project gives students a true sense of accomplishment, a new confidence in their skills, and a boost to their imagination and creativity.

4. CONCLUSIONS

In this paper we presented an outline of the latest revision of the Parallel and Scientific Computing course offered at Iona College, NY. The course strives to achieve a balance between the parallel and scientific aspects of the material, between the cutting-edge new topics like CUDA C and more traditional parallel approaches like MPI.

Students are engaged in parallel software development through a number of projects, which serve to not only reinforce the new parallel programming concepts but to illustrate their application in scientific computing. While challenging for both the instructor and the students, the course is, nevertheless, a rewarding experience and of vital importance for the professional development of students as they prepare to enter the ever changing modern workplace.

REFERENCES

[1] W.Gropp, E.Lusk, A.Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd edition, MIT Press.

[2] G.Karniadakis, R.Kirby II, *Parallel Scientific Computing in C++ and MPI : A Seamless Approach to Parallel Algorithms and their Implementation*, Cambridge University Press.

[3] D.Kirk and W.Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann, 2010.

[4] P.Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann.

[5] S.Rivoire, A Breadth-First Course in Multicore and Manycore Programming, *SIGCSE'10,* Milwaukee, WI, 3/10.

[6] J.Sanders, E.Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison Wesley Publishing, 2010.

[7] Q.Tran, Teaching design & analysis of multi-core parallel algorithms using CUDA, *Journal of Computing Sciences in Colleges*, Volume 25 Issue 4, 4/10.

[8] B.Wilkinson, Y. Li, Workshop: General purpose computing using GPUs: Developing a hands-on undergraduate course on CUDA programming, *SIGCSE 2011*, Dallas, TX, 3/11.

[9] G.Wolffe, C.Trefftz, Teaching Parallel Computing: New Possibilities, *Journal of Computing Sciences in Colleges*, Volume 25 Issue 1, 10/09.

[10] E.Wynters , Parallel Processing on Nvidia Graphics Processing Units Using CUDA, *CCSC Eastern Conference*, Huntingdon, PA, 10/11.