Early Adoption: High-Performance Computing for Big Data Introducing parallel programming and big data in the core algorithms curriculum

Faraz Hussain Narsingh Deo Sumit K. Jha

Abstract

Proficiency in high-performance computing (HPC) is today an essential skill for students in any computer science program. While traditional curricula provide several courses related to parallel programming, we are increasingly including parallel computing topics in our mandatory undergraduate and graduate algorithms classes. We briefly review our recent activities in this direction and outline future plans. We discuss the courses being impacted, the choice of curricular material in our classes, and the evaluation of students on parallel programming problems. In particular, our efforts focus on the use of big data problems as motivational tools for initiating students in the area. Our aim is to teach students not only parallel programming techniques, but also encourage them to think of and formulate real-life tasks in terms of modules that can be solved by exploiting parallel algorithms and multicore computing architectures.

Keywords: big data, high-performance computing, HPC, parallel programming, MPI, multicore, CUDA.

1 Introduction

The University of Central Florida (UCF) has a large undergraduate computer science program: more than 75 students enroll in the *Design and Analysis of Algorithms* class and the enrollment in the *Computer Science II* course exceeds 200 students annually. As these courses are required for successful completion of the graduate and undergraduate computer science degrees respectively, we have identified them as suitable vehicles for reaching out to the general computer science student community at UCF. We are actively engaged in teaching these classes in the current semester, and this short paper describes our ongoing early adoption efforts.

2 COT 5405: Design And Analysis Of Algorithms

COT 5405 serves as a first course in graduate algorithms to graduate (and upper-level undergraduate) students majoring in computer science and computer engineering. Students arrive in the class with a solid background of C programming, object-oriented programming, data structures, discrete structures, formal languages, undergraduate automata theory and undergraduate algorithms. Traditionally, the course has focused on asymptotic complexity, graph algorithms, and greedy, divide and conquer, and dynamic programming algorithms [4].

Evaluation metrics have so far emphasized the ability to solve problems by designing new algorithms, implementing algorithms using object-oriented concepts and analyzing the complexity of given algorithms. The course structure for COT 5405 is quite dense, and we feel there is no hope of adding new lectures to the class without removing the existing lectures on fundamental concepts in the design and analysis of algorithms. Therefore, we are revamping three lectures in this class to include parallel program design and analysis content:

1. The lecture on complexity has been extended to include discussions on parallel complexity [6]. In particular, we now introduce the shared-memory parallel random-access machine (PRAM) model of computation and various read/write conflicts with a special focus on the CREW (concurrent read, exclusive write) concurrent random-access strategy. The lecture also includes a discussion on the **NC** complexity class, and two practical illustrations of this class of problems – parallel sorting and parallel matrix multiplication. This class also places special emphasis on big data problems that arise when matrices are too large to be staged in the memory of any single processing node. Practical examples such as machine learning over large data sets are used to demonstrate the real-world impact of parallel algorithms.

2. Divide and conquer algorithms have traditionally been illustrated using Strassen's matrix multiplication algorithm. Parallel matrix multiplication using naive strides and block-by-block multiplication is presented to understand the variations in speed-up gained by different parallel algorithms [2]. The increasing focus, in recent years, on the design of dedicated matrix multiplication algorithms for different hardware platforms and the pursuit of parallel hardware accelerators is used to emphasize the importance of parallelization on large numerical data sets.

3. Dynamic programming [3] can be a challenging algorithmic technique to get across to students, mostly due to its non-informative poorly chosen nomenclature. The Floyd-Warshall algorithm is usually included in a discussion of this topic. A trivial parallelization of the algorithm where each node computes an entry of the result matrix is introduced to the class. Parallelization of the Floyd-Warshall algorithm provides an opportunity to introduce non-numerical, structured, big data sets to students. The `http://www.opensecrets.org` website, that details campaign contributions to PACs and individual candidates, has been used as an example of the use of big data in the realms of politics, public administration and advocacy.

The goal of changes made to the COT 5405 class is to ensure a non-obstructive introduction to parallel programming in the mandatory graduate algorithms class. We hope that this will permit MS and PhD students in computer science, electrical engineering, physics, mathematics and visiting faculty (who attend COT 5405 lectures) to think about and solve big data problems using parallel algorithms. We believe that this course will serve as an gateway and encourage them to not only enroll in specialized parallel programming classes, but also to pursue research that involves problems that are nearly impossible, or prohibitively expensive, to solve without recognizing and exploiting their inherent parallelism.

3 COP 3503: Computer Science II

Computer Science II (COP 3503) is a mandatory class for computer science undergraduates. Students are expected to arrive with a solid foundation of C and Java programming and have usually been exposed to the fundamentals of data structures and discrete mathematics. However, many students lack sufficient background in formal languages and automata theory. COP 3503 is a gentle introduction to undergraduate algorithms and object-oriented implementations of such algorithms. This is the final core algorithm-related course in our required undergraduate curriculum and hence an ideal venue for the introduction of certain advanced parallel programming concepts.

While big data may seem too specialized a topic to focus on for undergraduate students, we believe that a significant number of professional and research opportunities for our students will demand such skills. This course provides an excellent opportunity to initiate students into understanding big data problems and developing strategies for solving them. We have been designing a number of lectures to introduce parallel programming to students of COP 3503:

1. Introduction to object-oriented programming is

being altered to use the C++ bindings of the MPI (Message Passing Interface) standard [5]. Concepts such as namespaces, and virtual functions will be introduced using the MPI communicator, and the send/receive paradigm. Object-oriented programming concepts such as exception handling and public/private attributes will be discussed in the context of MPI's C++ bindings. Students will also be introduced to large data sets that cannot be placed on a single node and will be required to exchange data between nodes using the MPI extensions of the object-oriented language to facilitate simple computations such as computing the average of a large (about 100 MB) list of numbers.

2. COP 3503 does not include topics in formal languages or automata theory, so it is not feasible to introduce the PRAM model at this stage or discuss the NC complexity class. Instead, hands-on object-oriented programming using the MPI standard will be used to implement simple parallel algorithms such as parallel search. Experimental observations regarding the number of processors and the time taken to solve the problem will be used to motivate a discussion on the class of problems that can be effectively parallelized. Other straightforward problems like computing the sum of a list of numbers will be used to demonstrate the need for high-performance computing in manipulating large data sets.

3. Sorting algorithms are a primary topic of discussion in this class. Using a rudimentary understanding of MPI, we will parallelize sorting algorithms and study the need for read/write synchronization among multiple processes. The goal will be to explain the CREW model to students using a practical example. By sorting massive lists of numbers, the students will gain first-hand experience and confidence in manipulating big data sets.

We believe that this hands-on OO-programming approach can be used to sneak MPI and parallel programming into the COP 3503 curriculum. We will also investigate the possibility of designing a simpler-than-MPI object-oriented parallel programming framework specifically for use specifically in our *Computer Science II* classes. This, however, will require a large effort that cannot be accomplished without significant extramural support.

4 Assignments As Vehicles for Learning

The ancient Hindu archer Eklavya is said to have achieved his training merely by observing the statue of his guru. We believe that a significant fraction of our students are not substantially challenged by our traditional computer science assignments [7] and seek such challenges elsewhere (in ethical hacking clubs, app development forums, etc.). We have been exploring the possibility of using extra-credit assignments and projects as vehicles for inculcating parallel programming proficiency among our students.

In a COP 3503 class, we introduced the satisfiability (SAT) problem [1] and the solution of the problem using survey sampling. We then asked students to implement a simplified version of the algorithm using MPI or CUDA frameworks for an extra-credit assignment. Even though the problem is challenging and students had no formal training in multicore programming or CUDA, we received about half a dozen correct submissions. By introducing C++ bindings of MPI in the curriculum as an OO framework, we expect this number to be significantly larger.

In the COT 5405 class, we held a special lecture on MPI and provided a challenging big data problem from bio-medical sciences to the students. The size of the data provided to the students was more than 150 MB and they were given access to an instructional high-performance computing server. The was large enough that parallellization had an immediate and easily observable impact on the performance of the programs written by the students. In a class of 40 students, we received about 10 submissions. We we consider that a very good start, but are also investigating the barriers that prevented other students from completing this optional assignment.

5 Challenges And Future Work

There are three challenges that we are actively investigating:

- Parallel programming frameworks such as MPI have not been designed as educational vehicles and may not be the appropriate choice for a first course in parallel programming. For giving students gradual exposure to parallel programming, it would be more effective to design a parallel programming library with rich object-oriented features that can be tuned to be as simple as a sequential programming language and as powerful as MPI.

- We currently use a 100-core computing cluster (`euler.i2lab.ucf.edu`) to teach parallel programming to about 150 students. While this is very good for effective classroom demonstrations, the number of accounts is limited, leading to strained resources especially near assignment deadlines. The class will greatly benefit from access to larger educational clusters. We believe that every student should be given a quota on a parallel computing cluster every semester and plan to explore an institutional proposal mechanism using Technology Fee Proposals to make this feasible at the University of Central Florida.

- Big data is a huge area of growth for professional and research opportunities for our students. However, there is a dearth of truly large data sources (gigabytes and terabytes) and adequate hardware where such data can be staged for computation. We have used data from the NIH and `http://www.opensecrets.org` in our classes but they are no larger than 500MB in size. We are actively seeking industrial collaborations that can provide massive, open-source, high-fidelity data for our classes.

6 Acknowledgment

The authors thank Susmit Jha (Intel Strategic CAD Labs) and Mohammad Zubair Ahmed (Akamai Corporation) for useful discussions. We thank the LittleFe program and Dr. Charlie Peck (Earlham College) for providing equipment support to our classes. We also thank NVIDIA Corporation for their generous equipment grant to the parallel computing laboratory. The work reported in this short paper has been made possible by the NSF IEEE TCPP Early Adopter grant to the University of Central Florida.

References

[1] Armin Biere. *Handbook of satisfiability*, volume 185. IOS Press, 2009.

[2] Richard P Brent. *Algorithms for matrix multiplication*. Stanford University Press, 1970.

[3] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*. MIT Press, 2001.

[4] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006.

[5] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104. Springer, 2004.

[6] Ian Parberry. *Parallel complexity theory*. John Wiley & Sons, Inc., 1987.

[7] Paul R Pintrich and Elisabeth V De Groot. Motivational and self-regulated learning components of classroom academic performance. *Journal of educational psychology*, 82(1):33, 1990.