Fall 11- How to Integrate Parallel Computing into Programming Languages Courses

Wilson Rivera

Electrical and Computer Engineering Department
University of Puerto Rico at Mayaguez
Mayaguez, Puerto Rico

***Abstract*—This paper presents the activities and results of the early adoption of the TCPP curriculum initiative at the University of Puerto Rico, Mayaguez Campus (UPRM). This semester we focused on integrating parallel computing topics into a Programming Languages undergraduate course.**

I. Introduction

The Department of Electrical and Computer Engineering at UPRM offers Bachelor of Science and Master of Science programs in Electrical Engineering and Computer Engineering. The Department has about 1,400 undergraduate students and near 100 graduated students. UPRM is a minority serving institution. This paper describes the adoption of the TCPP Curriculum Initiative into a Programming Languages (ICOM 4036) course which is a required course for Computer Engineering undergraduates. The annual enrollment of the course is around 80 students.

This course is a comparative study of programming paradigms, including imperative, object-oriented, functional, logic, and concurrent programming. The measurable learning outcomes of the course include (1) ability to apply different programming paradigms; (2) ability to implement simple scanners and parsers; and (3) ability to select appropriate languages for specific applications. Previous to the adoption of TCPP curriculum the course had the following distribution of topics and hours per topic: Introduction (2hr); syntax analysis (9hr); imperative programming (9hr); object-oriented programming (8hr); functional programming (6hr); logic programming (6hr); concurrent programming (3hr). Previous to the PL course students had taken both Advance Programming (ICOM4015) and Data Structures (ICOM4035). Since these courses provide an extensive exposure to both imperative and object oriented programming it was very reasonable for us to increase the coverage of concurrency by introducing parallel computing topics. Thus, we maintained the number of hours for introduction, syntax analysis, functional programming, and logic programming. We then re-distributed memory management (8hr) and concurrency (9hr). With memory management topics we mean technical aspects of variables, types, binding and scope, parameter passing, data representation in memory, stack and heap allocation, runtime storage management, and garbage collection. With concurrency we addressed issues related to memory access models, parallel programming notations, semantics and correctness, and performance metrics.

II. Integration of Parallel Computing Topics

Table I depicts the parallel computing topics introduced into the course materials. The topics are classified into three categories as algorithms, programming and architecture according to the early adopter TCPP curriculum initiative document [1]

Table 1: TCPP Topics Introduced into ICOM4036 at UPRM

Topics	Bloom	Hours
Algorithm		
Cost tradeoffs		0.5
time vs. space	C	
power vs. time	C	
Graph algorithms		1
Search	A	
Path selection	A	
Programming		
Parallel programming notations		5
OpenMP	A	
TBB	A	
CUDA/OpenCL	A	
MPI	A	
Erlang	A	
Semantics and correctness issues		1
Task and Threads	C	
Critical regions	A	
Producer-consumer	A	
Monitors	A	
Deadlocks	A	
Racing conditions	A	
Performance Issues		0.75
Load balance	A	
Performance metrics	A	
Architecture		
Parallelism and Memory Models		2
Superscalar (ILP)	C	
Flyn's Taxonomy	C	
Shared memory	C	
Distributed memory	C	
Hybrid memory	C	

We introduce the Flynn's Taxonomy empathizing SIMD and MIMD architectures. Then we discuss memory access models (shared, distributed, hybrid). We show examples of parallel computing environments running on local resources (e.g. a GPU accelerated server; a cluster of servers; and multicore laptops). We also show the Intel's "Colony" demo to emphasize the importance of parallel programming. According to students' feedback, this introduction to parallel computing, reasonably covered in two hours of lecture, resulted very interesting and challenging to the students. We then proceeded to elaborate on specific parallel programming notations.

We discussed OpenMP, MPI, CUDA, TBB and Erlang. First, we show examples of OpenMP programming with an emphasis on parallel constructs, synchronization mechanisms, and performance considerations. Second, we discuss examples of MPI programming with emphasis on point-to-point communication, collective communication and synchronization. We also discuss an implementation of the matrix-vector multiplication in CUDA. We also introduce TBB and discuss "colony" demo source code. Finally, we Introduce Erlang programming and show examples compared to MPI. The parallel programming notations can be covered with one hour lecture for each notation. The discussion of parallel programming models is also improved by taking into consideration performance issues. We define speedup and scalability concepts. When discussing OpenMP examples, we introduce Amdahl's law and discuss briefly ompP as a tool for performance monitoring. For example after the OpenMP lecture students are able to differentiate between the embarrassing parallelisms inherent in parallelizing the matrix-vector multiplication and the parallelism considerations when implementing mergesort. We also introduce communication overhead when discussing MPI examples and discuss IPM profiling. In addition we develop awareness of Moore's Law energy management considerations.

Technical discussion on lectures is complemented with programming assignments. For example, this semester students were asked (1) implement a multi-access threaded queue with multiple threads inserting and multiple threads extracting from the queue and use locks to synchronize access to the queue. (2) Implement a simple loop that calls a function dummy containing a programmable delay. Partition this loop across threads using static, dynamic and guided scheduling and compare performance. (3) Implement the Dijkstra's Single source shortest path algorithm using OpenMP and TBB and discuss related performance issues

III. RESULTS

Prior to this course, students have not had major exposure to parallel computing topics. It is surprising that even having into procession multicore computers; none of the students had any knowledge of how to use multiple cores efficiently. Introducing the Intel's "Colony" demo early in the discussion resulted to be of great benefit for further discussion.

Introducing performance monitoring tools as we discuss program examples also resulted if great benefits for students as they went through the assignments, they were able to use the tools and provide soundness results and conclusions.

Students also found very useful to introduce local resources early in the discussion. We show how to compile parallel application on both Linux and windows based systems. Using Rocks-Ganglia interface to visualize cluster status was also very useful to understand cluster configurations and performance.

Most of student did express concerns about Erlang programming. It was very difficult to demonstrate advantages of Erlang versus MPI. The only feature that seems to catch any attention from the students was the hot code upgrade that allows code in a running system to be updated without interrupting the execution. We plan to eliminate Erlang discussion next semester and either expand the CUDA discussion or introduce Unified Parallel C (UPC).

Introducing parallel computing topics into a PL course was very natural and easy to implement. We need to explore on the benefits to connect syntax analysis and parallel computing topics. We also need to assess the benefits of discussing parallel computing previous to memory management.

ACKNOWLEDGMENT

This material is based upon work partially supported by the NSF-TCPP Early Adopter Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] NSF/TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates