International Journal of Engineering Education Vol. 32, No. 2(A), pp. 772–781, 2016
Printed in Great Britain

0949-149X/91 $3.00+0.00
© 2016 TEMPUS Publications.

Teaching Parallel Computing Concepts Using Real-Life Applications*

ALI YAZICI and ALOK MISHRA
Software Engineering Department, Faculty of Engineering, Atilim University, 06836, Incek, Ankara, Turkey.
E-mail: ali.yazici@atilim.edu.tr, alok.mishra@atilim.edu.tr

ZIYA KARAKAYA
Computer Engineering Department, Faculty of Engineering, Atilim University, 06836, Incek, Ankara, Turkey.
E-mail: ziya.karakaya@atilim.edu.tr

The need to promote parallel computing concepts is an important issue due to a rapid advance in multi-core architectures. This paper reports experiences in teaching parallel computing concepts to computer and software engineering undergraduates. By taking a practical approach in delivering the material, students are shown to grasp the essential concepts in an effective way. This has been demonstrated by implementing small projects during the course, such as computing the sum of the terms of a geometric series using pipelines, solving linear systems by parallel iterative methods, and computing Mandelbrot set (fractal). This study shows that, it is useful to provide real-life analogies to facilitate general understanding and to motivate students in their studies as early as possible via small project implementations. The paper also describes an overall approach used to develop students' parallel computing skills and provides examples of the analogies employed in conjunction with the approach described. This approach is also assessed by collecting questionnaires and learning outcome surveys.

Keywords: parallel computing; message-passing interface; speed-up factor; Flynn's taxonomy

1. Introduction

Over the past few years, a change of paradigm has occurred in the computer industry due to power dissipation constraints and memory access-time limitations. Rather than increasing the processors' frequency of operation (the usual strategy), computer manufacturers have started introducing more core per chip [1]. Today, general-purpose computers have multiple cores and it is common to see processors with 4 cores; even though soon it is expected to have computers to have 10s and even 100s of cores [2]. It is now imperative that all computer science graduates as well as undergraduates develop the skills to successfully make software programs for systems with multiple processors [2]. Recent advancements in computer hardware are going in the direction of increasing the multiplicity of components executing in parallel rather than increasing the speed of the individual devices [3]. In the same article it is argued that another dimension to the complexity in the field of parallel and distributed computing is the heterogeneity of the components in the current hardware. According to Falcao [1], this has created the potential for increasing the processing performance, but also poses new challenges, namely regarding the extra level of effort required for programmers to exploit these new processing machines. In this context, the need to learn parallel programming topics has become a significant issue due to a rapid growth in the parallel computing field [4]. According to Hartman and Sanders [5] parallel application development has been included in different areas, such as biology, physics, economics, engineering, etc. and this indicates that this course should be included in the teaching curricula of different fields in order to improve student skills and cover the need to learn this topic [6]. Therefore, the need for students to learn parallel programming is growing. In this way, it is easier to deliver the course from a multi-core laptop where lectures can be supplemented with live presentations.

The success of parallel computing in solving real-life, computation-intensive problems relies on their efficient mapping in commercially available multi-processor architectures [7]. According to Giacaman [8], although parallel computing has been around for decades, it still largely remains an under-developed skill amongst programmers. He further argues that parallelism takes the concepts of multi-threading beyond concurrency: the emphasis now is on "real performance" by reducing the overall execution time. In developing interactive and performance GUI (Graphical User Interface) applications for modern multi-core systems, both concurrency and parallelism are required [8]. Today, low-cost, general-purpose desktop parallel machines and parallel programming environments are widely available for students; thus, in a parallel computing course, theory and practice must be taught as early as possible [9].

There is a lack of scientific parallel curricula

* Accepted 12 November 2015.

suitable for highlighting computational science principles in the classroom [10]. Thus, this paper is an endeavor to advance the mode of teaching parallel computing in an even more inspiring manner by adopting real-life applications and examples to complement the theoretical discussions in the classroom. Further, this study adapts small projects and live demonstrations during the course which reinforce the concepts. Various tools were utilized during the course which facilitated students' analysis of their efficiency of codes and the time spent in some portion of the code. It has been observed that a practical approach is a more effective and enthusiastic approach in teaching parallel computing to undergraduate students.

1.1 Background

In the last decade, parallel computing has regained its popularity based on the facts summarized below:

- The Von Neumann architecture has reached its physical limits;
- The cost of CPU is reduced;
- Multi-core PCs and systems are accessible easily; and
- A new area of knowledge in Parallel and Distributed Computing (Computer Science Curricula 2013-Final Report 0.9, Pre-release version, October 2013) has been released, emphasizing parallel computing concepts.

The present paper details the experience of teaching parallel computing in a third-year computer engineering course consisting of over 40 students. The prerequisites for this course are two first-year programming courses: the first one being an introductory programming course, while the second one is of being data structures. The aim of this course is to introduce students to a better understanding of parallel computing concepts and developing complex software applications.

1.2 Literature review

Parallel computing has typically been treated as an advanced topic in computer science, and it must now be treated as a core topic [2]. Muresano et al. [4] suggested the need to learn parallel application topics due to the rapid growth in the parallel computing field. Giacaman [8] found that live coding demonstrations and use of analogies are most helpful in learning parallel computing. He argued that analogies are to be used to relate parallel computing concepts to familiar real-life scenarios, and this is especially easy with object-oriented languages, such as Java. According to him, in teaching parallel computing to undergraduate students, it is strongly believed that a practical approach is most effective. Rendell [11] suggested efforts to structure a course in parallel systems using a group-project approach. Joiner et al. [10] presented the result of evaluation by workshop instructors and the materials and tools developed during teaching parallel computing in a science faculty. Chen [12] highlighted the effort taken by several educational institutions to move towards parallel or distributed computing environments and the need for this evolution. Kessler [13] pointed out the significance of teaching the basic understanding of parallel computations and parallel programming early in computer science education. Stojanovic [14] emphasized the benefits of the hands-on approach in teaching parallel computing concepts using different paradigms.

In recent studies, Arroyo [3] discusses how parallel and distributed computing and, specifically, concurrent and parallel programming topics should be taught in a course by using the existing tools and parallel patterns to reduce the gap between sequential and parallel programming. Lin [15] described the experience to teach parallel and distributed computing by using a cluster computing portal. Ferner et al. [2] developed two new approaches (higher level of abstraction and compiler directive) to teaching parallel computing to undergraduates using higher level tools that lead to ease in programming, good software design, and scalable programs.

Section 2 of the paper gives an overview of parallel computers and programming. In Section 3, the details of the approach used in teaching parallel programming are given. Analogies used in teaching the course are also provided in the same section. Section 4 provides the assessment of the course conducted according to the ideas outlined in Section 3. Finally, discussions and future work are included in the last section.

2. Parallel computers and programming

Parallel and distributed programming skills have become a common requirement in the development of modern applications, and there is a general consensus that parallel programming topics should be spread throughout all related undergraduate curricula [3]. In recent years, parallel programming has increased significantly in different knowledge fields [6], and this increase has motivated universities to design parallel programming course contents according to the requirements for students to develop such applications in an efficient manner [4]. The trend of using parallel programming is due to its achieving more accurate results, lower costs, reduced execution time and optimizing computational resources [16].

2.1 Flynn's taxonomy

Flynn's classification [17] of computers is based upon the number of concurrent instructions and data streams available in the architecture. According to this classification, there are four different architectures:

a. Single Instruction, Single Data stream (SISD). In this category, there are traditional uniprocessor machines, such as a PC with a single control unit. The instructions are executed one at a time following the so-called machine cycle (fetch-decode-execute) sequence (Fig. 1a).
b. Single Instruction, Multiple Data streams (SIMD). SIMD computers have a single control-unit performing operations on multiple data streams. The vector processors of the 1970s, array processors or GPUs are in this category (Fig. 1b).
c. Multiple Instruction, Single Data stream (MISD). Multiple instructions operate on a single data stream. This is an uncommon architecture generally used for fault tolerance (Fig. 1c).
d. Multiple Instruction, Multiple Data streams (MIMD). Computers in this category are known as multi-core/multiprocessors (parallel computers), where a number of autonomous processors simultaneously execute different instructions on different data. The distributed systems are generally recognized to be MIMD architectures (Fig. 1d).

2.2 Further categorization of Flynn's taxonomy

Today's computers mostly fall into the MIMD architecture of Flynn's Taxonomy. This categorization is further divided into two most common parallel programming styles referred to as 'SPMD' and 'MPMD'.

a. *SPMD (Single Program—Multiple Data)*

A single program is processed simultaneously by multiple autonomous processors, operating on multiple data streams. This is the most commonly used parallel programming style in use, and today's computers use MIMD architecture to implement this style. Although the word "single" is used, there are at least two different processing units executing the same program and not necessarily the same instruction of that program.

b. *MPMD (Multiple Program—Multiple Data)*

Multiple programs are processed simultaneously by multiple autonomous processors operating on multiple data streams. The two mostly used programming paradigms, the master-worker model and the pipelining model, can easily be implemented using this style. There may be multiple programs running in parallel on the line of pipe and doing their own operations on different data streams.

2.3 Parallel programming paradigms

The two most commonly used parallel programming paradigms are the "master/worker" (also

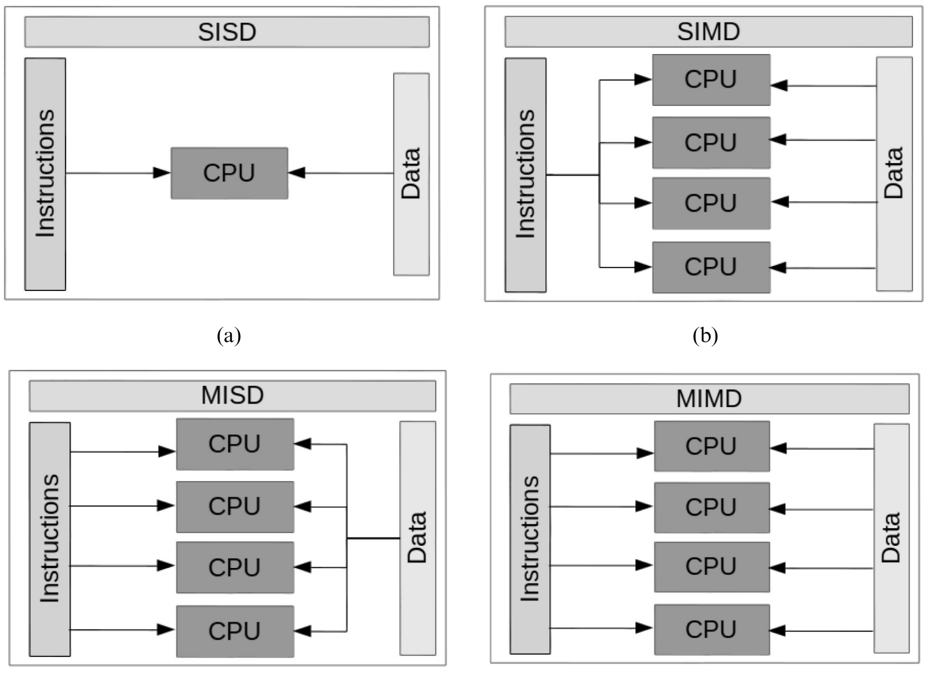


Fig. 1. Flynn's taxonomy of computer systems.

referred to as “master/slave”, or “task farming”) model and the “pipelining” model.

The master-worker model is the simplest and most commonly used parallel programming paradigm. Here, the master is the main computation that either generates the data to be processed by workers, or starts the sub-problem that is to be solved by workers, who in turn take this data or job from the master and return the results to the master process. This paradigm could be used in either SPMD or MPMD. Many other paradigms, such as divide-and-conquer, can also be implemented using this model.

Pipelining is analogous to the production line in a factory, and it is a good application of MPMD. If there are multiple tasks to be done on the input one after another, this technique would be the choice of programming paradigm. Although it seems that, for each task, there is a single input, in reality each process operates on different data, which is the output of the previous task. This can be thought of as many filters used to refresh the water stream. Depending on the point of view, it can be seen as single data or multiple data. If we consider the filter processes as a black box, then single data is a stream coming into this box. From this point of view, one can say that there is a single data stream into this process.

2.4 *Performance metrics*

The performance gained in parallel computing is measured by the speed-up factor, S, which is usually defined as the ratio of the execution time, t_s, of the best sequential algorithm for solving a problem on a SISD computer to the execution time, t_p, of the parallel version of the same algorithm on a parallel computer with p processors.

$$S = \frac{t_s}{t_p} \leq p$$

Efficiency $E = \frac{S}{P}, 0 < E < 1$, is the normalized measure of speed-up that shows how effectively each processor is utilized. A linear speed-up corresponds to an efficiency value of 1 and indicates that all processors are completely engaged in the computing process.

3. Ideas for classroom applications

3.1 *Teaching approach*

This section describes the approach taken in delivering the concepts of pipelining, master-worker model and the SPMD paradigm using an MPI (Message Passing Interface) environment. The primary focus is on communicating the practical significance underpinning parallel computing. This includes delivering the contents in a practical way towards highlighting the necessity of developing parallel computing skills for current and future mainstream computers. This is achieved by delivering the course via a multi-core laptop, where lectures turn into live demonstrations. Some contents are still presented in the traditional way of using slides, while half a lesson is dedicated to supplementing theory with live code demonstrations.

One of the most striking features of the course is the use of real-life analogies and examples to supplement the theoretical discussions and easily introduced the fundamental concepts such as speed-up, efficiency, and Amdahl’s law. A typical coverage of the topics in an introductory parallel computing course along with a set of analogies and algorithmic examples are given in Fig. 2. The pyramid indicates that, as the teaching proceeds, the amount of

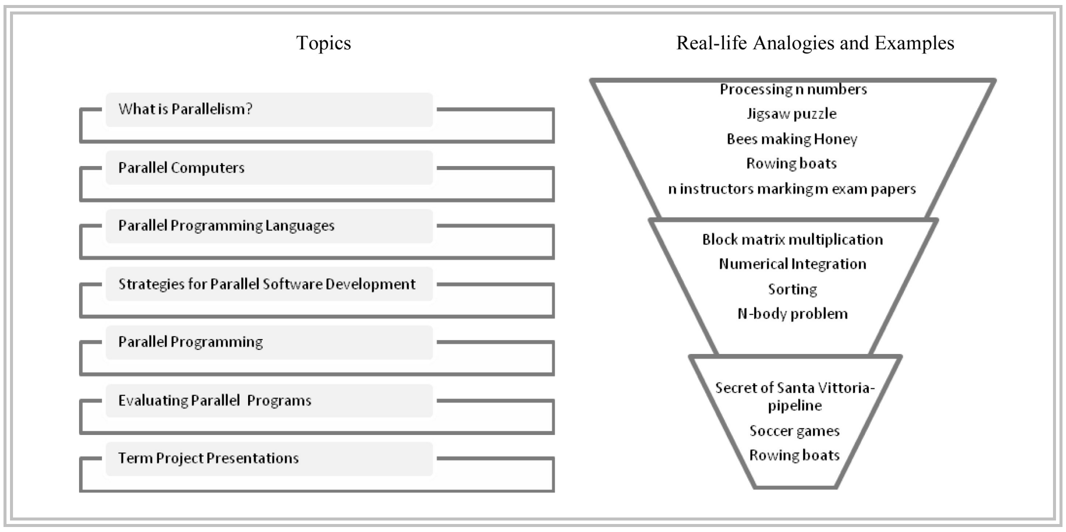

Topics
Real-life Analogies and Examples
What is Parallelism?
Parallel Computers
Parallel Programming Languages
Strategies for Parallel Software Development
Parallel Programming
Evaluating Parallel Programs
Term Project Presentations
Processing n numbers
Jigsaw puzzle
Bees making Honey
Rowing boats
n instructors marking m exam papers
Block matrix multiplication
Numerical Integration
Sorting
N-body problem
Secret of Santa Vittoria-pipeline
Soccer games
Rowing boats

Fig. 2. Main topics in an introductory parallel computing course.

analogies and examples may be reduced. On the other hand, to emphasize critical theoretical discussions, some of the analogies are reiterated in the teaching process.

The instructors of the course are always encouraged to add more intriguing analogies and examples to the existing ones.

Another challenging feature of the course is the term project taken over by a team of 2–3 students. One of the requirements of the project, which almost spans the whole term, is to develop a parallel program using MPI and/or OpenMP. Some of the projects assigned to the students are: Parallel Implementation of Gauss-Seidel Algorithm for Linear Systems using MPI (Distributed Memory Computing) and OpenMP (Shared Memory Computing), Pipelined Sine Calculator using MPI (Pipelining) and Mandelbrot Set using MPI (Embarrassingly Parallel Computation). At the end of the term, the project team presents their work to an audience and writes a comprehensive report to cover all the stages of the development.

A more detailed coverage of the course is outlined in Table 1. During the design of the course syllabus,

Table 1. A typical course coverage

	Content	**Topics**		**Analogies/Applets/ Examples**		**Tools**	
1	What is parallelism? Motivation for parallel computing	• Real-life examples • Definition of Speed-up/Efficiency/Scalability • Amdhal's law		• Processing n numbers (summation) • Solving Jigsaw Puzzles with multi-players • Bees making honey comb • Octuple skull (boat rowing) • N teacher(s) marking M exam papers (M>>N)			
2	Hardware – Parallel computers • Flynn's taxonomy	***SISD***	***MIMD***				
		• Conventional computers	• Shared-memory • Distributed shared-memory	• Message-passing	• Examples of parallel computers		
3	Parallel Programming Strategies	***Partitioning*** • Data partitioning • Functional decomposition	***Divide & Conquer***	• Bucket Sort • Numerical Integration • N-body problem • Block matrix multiplication			
4	Strategies for Parallel Software Development		• Message-passing computing • Pipelined computing • Synchronous computing	• Octuple skull (boat rowing) • The Secret of Santa Vittoria – 1969 movie • Pipelined instructions • Petroleum pipelines • Well-pumping • Bucket Sort • Soccer games			Term Project
5	Parallel Programming	Message passing	• Process creation • Basic send-receive routines • Broadcast • Gather/Scatter	• UML Sequence Diagrams • UML Activity Diagrams • Applet for Mandelbrot set [19]		• PVM • MPI	
		Shared memory	• Creating concurrent processes • Threads • Data sharing and accessing • Critical sections • Synchronization			• OpenMP	
6	Evaluating parallel programs	• Measuring execution time • Ping-pong method for measuring communication time • Profiling • Speed-up		• UML Sequence Diagrams • UML Activity Diagrams • Parallel speed-up and Linear to Parallel demonstrations (K. L. Busbee, Copyright 2009 Creative Commons Attribution License, CC-BY 3.0)		Visualization tools • Upshot for MPI • Projections for Charm++ • Vampire • Jumpshot for MPI	

the general structure of the textbook Parallel Programming [18] was closely followed. The course is divided into 6 main stages and related topics are covered during the lectures supplemented with a rich set of examples and real-life analogies. In the sequel, some of these analogies will be elaborated.

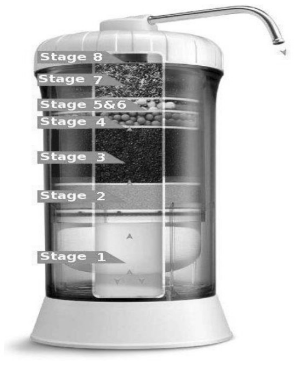
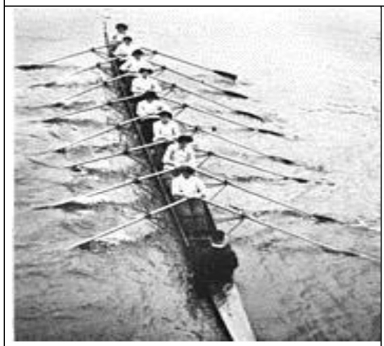
3.2 More on analogies and demonstrations

This section elaborates some of the analogies and examples given in the class to help the students in grasping the concepts such as pipelining, parallel architectures and parallel thinking. During the execution of the theoretical lectures, a movie poster, an image or illustration is utilized in addition to the formal lecture material. A sample list of supplementary content is shown in Table 2.

3.3 Tools for measuring parallel performance

There exists various performance monitoring tools to monitor the execution of parallel programs. For

Table 2. Some examples of real-life analogies

PIPELINING		
In the beginning there was Bombolini the fool, Bombolini the drunk, Bombolini the joke. In the end there was Bombolini the mayor, Bombolini the hero, Bombolini the beautiful. In between is the Secret of Santa Vittoria. THE SECRET OF SANTA VITTORIA "THE SECRET OF SANTA VITTORIA" ANTHONY QUINN VIRNA LISI · HARDY KRUGER · SERGIO FRANCHI ANNA MAGNANI		 Stage 8 Stage 7 Stage 5&6 Stage 4 Stage 3 Stage 2 Stage 1
The Secret of Santa Vittoria (1969 movie) (This image is used for informational or educational purposes only [20]) In the movie, the townspeople hide a million bottles in a cave before the arrival of a German army detachment by forming a pipeline of townspeople and passing the bottles hand-to-hand.	Lotus Cars Assembly Line as of 2008 [21] is a real-life example for pipelining.	Water purifier [22] a multi-stage pipeline
SIMD		
		Octuple skull [23] (boat rowing) and bees making honey [24] are two real-life illustrations for SIMD architecture
MIMD		
		Soccer games are real-life example for MIMD architecture. The Robocup annual competitions is an attempt to realize MIMD architecture for soccer playing robots [25]

example, there is a feature called GEM (Graphical Explorer of MPI), which is a graphical front end for In-situ Partial Order (ISP), a dynamic formal verification tool for MPI C/C++ programs developed at the University of Utah [26]. This tool can be used to visualize the MPI calls and to analyze the codes for errors and warnings of Deadlocks, Functionally Irrelevant Barriers, MPI Object Leaks, Local Assertion Violations, and MPI-Type Mismatches. 'Eclipse' has another feature called ETF (External Tools Framework), which is also known as PTF (Performance Tools Framework) and helps us to integrate external tools for performance analysis. There is such a well-tested tool known as TAU (Tuning and Analysis Utilities), which is a set of plug-ins that can be easily integrated into Eclipse PTP. The integration process is well-documented and more information can be found in paper [27]. The use of such visual tools in an introductory course increases the understanding of message passing orders, processes, and deadlock points. Students can also analyze the efficiency of their codes and the time spent in some portion of the code. Furthermore, utilizing such tools increases their interest in the topics even further.

4. Assessment

In teaching parallel computing to undergraduate students, it is strongly believed that a practical approach adopting real-life analogies helps in understanding the concepts in effective ways. This approach has been put into action in the Parallel Computing elective course taught at Atilim University by two of the authors. The course has two first-year prerequisite courses: the first being an introductory programming course, and the second one being data structures, both requiring a strong background in programming and algorithms.

The effectiveness of the approach and the achievement of the learning outcomes are measured at the end of the term using an on-line survey, containing a set of short questions, inquiring about the level of achievement of the learning outcomes. Furthermore, the actual evaluation of the same course outcomes are also assessed using formative and summative assessment techniques, i.e. by using in-course instructor observations, lab-work, homework, project work, midterms and final exam. The items assessing the individual outcomes are evaluated separately and shown in the same table to see whether the student perceptions are consistent with the instructors' actual measured results. Of the 42 students enrolled in the course, 36 students completed the survey and both the results of student and instructor evaluations are given in Table 3.

Column 2 of the table represents the perception of the students and Column 3 is the actual evaluation of the instructor(s) based on the formative assessments (10%), assignments/project (20%), Mid-term examination (30%) and the Final examination (40%). According to the results, one can conclude that the learning outcomes are almost satisfied.

In addition to the analysis applied to the entire class, the authors tried to find out if there is any relation between the prerequisite programming background and achievement in learning parallel programming concepts. In order to observe this correlation, we also analyzed the instructor evaluation results with respect to students' achievement in the prerequisite courses. It is observed that the achievement of students in the fourth outcome (LO-4) of this course is highly related to their programming background and skills. There was a case of two students with significant achievement in the course, although their grades in the prerequisite courses were relatively less. When we investigated further, we noticed that their programming styles were as good as the other students who achieved higher grades from the prerequisite courses. Hence, it was concluded that this was no exception, and that their poor results in the prerequisite courses could only be attributed to other factors. In general, it can be inferred that students having good programming backgrounds showed a good achievement in implementing parallel algorithms using MPI and OpenMP. The results are given in Table 4.

From Table 4, one can also conclude that, independent from the background (knowledge from the prerequisite subjects) of the students, the teaching techniques used in the course have a significant impact on successful design of parallel algorithms which is perhaps the most important outcome of parallel programming course for their work life.

In addition to the standard learning outcomes

Table 3. Survey results of course learning outcomes

Part I	Course Learning Outcome	Result of Student Survey (1–5)	Evaluation of the Instructor(s) (1–5)
LO-1	Recognize parallelism in computational problems	**4.2**	**4.1**
LO-2	Explain different parallel systems and their classification	**4.2**	**4.4**
LO-3	Design parallel algorithms for different applications	**3.8**	**4.5**
LO-4	Implement parallel algorithms using MPI and OpenMP environments	**3.0**	**3.5**
	AVERAGE	**3.8**	**4.1**

Table 4. Learning outcomes vs. achievement in the prerequisite courses

Part I	Course Learning Outcome	Students with prerequisite grades less than CC 14 Students	Students with prerequisite grades more than CC 28 students	Evaluation of the Instructor(s) (1–5)
LO-1	Recognize parallelism in computational problems	**3.5**	**4.4**	**4.1**
LO-2	Explain different parallel systems and their classification	**3.6**	**4.8**	**4.4**
LO-3	Design parallel algorithms for different applications	**3.9**	**4.8**	**4.5**
LO-4	Implement parallel algorithms using MPI and OpenMP environments	**2.3**	**4.1**	**3.5**
	AVERAGE	**3.1**	**4.6**	**4.1**

Table 5. Effectiveness of real-life examples

Part II	Effectiveness of Real-life examples Survey	1–5
Q-1	Was it useful to relate real-life examples to parallel concepts?	**4.3**
Q-2	How was the effect of real-life examples on designing the algorithm of given problem?	**3.9**
Q-3	To what extent did the real-life examples help you to improve your programming skills?	**4.4**

survey, a set of questions were asked to learn the effectiveness of the teaching style mentioned earlier. These questions are given in Table 5. Out of 36 students who filled the questionnaire, only 24 replied the additional questions. The averages are shown in the third column according to which it is clear that the use of real-life examples and animations has a positive effect in teaching the concepts. However, the lower average in Q-2 suggests that designing a parallel algorithm requires additional skills and real-life analogies and that examples alone are not sufficient to develop design skills.

5. Discussions

In this study, we have investigated and presented the impacts of the practical teaching approach we have applied in teaching a parallel computing course to undergraduate computer engineering and software engineering students at Atilim University. This approach was based on three main pillars:

- *Using real-world analogies*: Certain analogies were used to relate parallel computing concepts, even theoretical concepts, to familiar real-life scenarios and analogies.
- *Live coding sessions*: Students found it helpful to understand the concepts with their query responses during live coding sessions.
- *Small projects*: It is recommended that in teaching parallel computing, a practical hands-on approach may be utilized along with real-life analogies and require small programming projects.

Although it may seem that our study and teaching approach has some similarities to Giacaman [8], such as using real-life analogies, and live code demonstrations during the teaching of the concept, there is a major difference in terms of the concepts taught in the courses. Our study is more dedicated to a comprehensive teaching the parallel programming concept, such as designing and creating algorithms for multi-core systems and for distributed systems, as well as the introduction part of parallelism with multi-threading. Most of the concepts given in our case focuses on core parallel computing and distributed computing using MPI (Message Passing Interface) architecture. Additionally, Giacaman used the threading capability of Java in order to teach responsive and thread-safe implementation using GUI frameworks, which are mostly supported by Java language implicitly. The lecture proposed in Giacaman's work are not entirely dedicated to parallel computing, and last only for three weeks with a content related with concurrency and multi-threaded programming. In this sense, the application of real-life analogies to teach all parallel programming concepts is a novelty of the presented study.

6. Conclusions

This paper presents a practical approach for teaching parallel computing concepts, using real-life applications and tools to undergraduate computer engineering students. We applied live coding sessions related with specific topics along with small projects to further promote students' understanding of the concepts in an effective way. As a result of this approach, it has been observed that students' responses were clearly positive and enthusiastic throughout the process. Further analysis is required by repeating the assessment in subsequent semesters. The authors are currently working on furthering the integration of visual profiling tools and live debugging sessions as well as inclusion of GPU programming examples in their teaching approach. In addition, in future assessment of students we would like to include additional questions in a quantitative manner so as to investigate how learning gains change as a function of different teaching methods and environments.

References

1. G. Falcao, How fast can parallel programming be taught to undergraduate students?, *IEEE Potentials,* **32**(4), 2013, pp. 28–29.
2. C. Ferner, B. Wilkinson and B. Heath, Toward using higher-level abstractions to teach parallel computing, *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2013 IEEE 27th International, Cambridge, MA, 20–24 May, 2013, pp. 1291–1296.
3. M. Arroyo, Teaching parallel and distributed computing topics for the undergraduate computer science student, *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2013 IEEE 27th International, Cambridge, MA, 20–24 May, 2013, pp. 1297–1303.
4. R. Muresano, D. Rexachs and E. Luque, Learning parallel programming: a challenge for university students. *Procedia Computer Science*, **1**(1), 2010, pp. 875–883.
5. J. Hartman and D. Sanders, Data parallel programming: a transition from serial to parallel computing, *ACM SIGCSE Bulletin*, **25**(1), 1993, pp. 96–100.
6. M. Paprzycki, Integrating parallel and distributed computing in computer science curricula, *IEEE Distributed Systems Online*, **7**(2), 2006, p. 6.
7. R. Biswas, S. Das, D. Harvey and L. Oliker, Portable parallel programming for dynamic load balancing of unstructured grid applications, *Proceedings of the 13th International Parallel Processing Symposium (IPPS'99)*, San Juan, Puerto Rico, April 12–16, 1999, pp. 338–342.
8. N. Giacaman, Teaching by Example: Using analogies and live coding demonstrations to teach parallel computing concepts to undergraduate students*, Proceedings of Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, 21–25 May, 2012, pp. 1295–1298.
9. A. Marowka, Think Parallel: Teaching parallel programming today, Distributed Systems Online, *IEEE*, **9**(8), 2008, p.1.
10. D. A. Joiner, P. Gray, T. Murphy and C. Peck, Teaching parallel computing to science faculty: best practices and common pitfalls, *Proceedings of the 11th. ACM SIGPLAN symposium on Principles and practice of parallel programming* (PPoPP '06), ACM, New York, NY, 2006, pp. 239–246.
11. A. P. Rendell, A project based approach to teaching parallel systems, in Alexandrof, G. D. van Albada, P. M. A. Sloot, and J. Dongarra (eds), *Computational Science—ICSS 2006*, vol. 3992, Springer, Berlin, Heidelberg, Germany, 2006, pp. 155–160.
12. L. C. Chen, Parallel and distributed computing related issues in education, http://epatcm.any2any.us/EP/EP2002/ATCMA122/fullpaper.pdf, Accessed 30 October 2015.
13. C. W. Kessler, Teaching parallel programming early, *Proceedings of Workshop on Developing Computer Science Education—How can it be done?*, Linkoping University, Linkoping, Sweden, March 10, 2006, IEEE Computer Society, pp. 1–6.
14. N. Stojanovic and E. Milovanovic, Teaching introductory parallel computing course with hands-on experience, *IJEE*, **31**(5), 2015, pp. 1343–1351.
15. H. Lin, Teaching parallel and distributed computing using a cluster computing Portal, *International Symposium on Parallel & Distributed Processing Symposium (IPDPSW), Workshops and Phd Forum*, Cambridge, MA, 20-24 May, 2013, pp. 1312–1317.
16. Y. Pan, Teaching parallel programming using both high-level and low-level languages, in P. M. A. Sloot, C. J. K. Tan, J. Dongarra, and A. G. Hoekstra (eds), *Lecture Notes in computer Science*, **2331**, Part-III, Springer, Berlin, Heidelberg, Germany, 2002, pp. 888–897.
17. M. J. Flynn, Some computer organizations and their effectiveness, *IEEE Transactions on Computers*, **21**(9), 1972, pp. 948–960.
18. B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Stations and Parallel Computers*, 2nd. Edition, Pearson Education, 2005.
19. http://www.ateji.com/px/parallel-implementation-of-the-mandelbrot-set.html, Accessed 21 October 2015.
20. http://www.moviegoods.com , Accessed 1 November 2015.
21. http://en.wikipedia.org/wiki/Assembly_line#mediaviewer/File:Final_assembly_3.jpg, Accessed 1 November 2015.
22. http://vitanero.com/pages/table-water-filter-system.php, Accessed 1 November 2015.
23. http://en.wikipedia.org/wiki/Octuple_scull, Accessed 21 October 2015.
24. http://keepingbee.org/how-does-bees-make-honey, Accessed 21 October 2015.
25. M. K. Sahota and A. K. Mackworth, Can situated robots play soccer?, *Proceedings of Artificial Intelligence 94,* Alberta, Canada, 1994, pp. 249–254.
26. http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.ptp.doc.user%2Fhtml%2F00overview.html, Accessed 1 November 2015.
27. https://www.cs.uoregon.edu/research/paracomp/papers/hpcc06/hpcc06.pdf , Accessed 21 October 2015.

Ali Yazici is a Professor and the Head of the Software Engineering Department at Atilim University, Ankara, Turkey. He received BS, and MS degrees in Mathematics from the Middle East Technical University (METU), Ankara, Turkey. He has completed his PhD dissertation at the Computer Science Department, Waterloo University, Canada. His research interests include Parallel Computing, Big Data and Cloud Computing, Database Systems, and e-Topics. He is the writer of many scientific articles, books and research reports in the field of Computing and Informatics. Dr. Yazici is a founding member of Turkish Mathematics Foundation, an associate member of Turkish Informatics Society, and a founding member of Turkish Informatics Foundation.

Alok Mishra is a Professor in Software Engineering at Atilim University, Ankara, Turkey. His areas of interest and research are software engineering and information systems. He has published articles, book chapters and book-reviews related to software engineering and information system in reputed journals, books and conferences. Dr. Mishra is also editorial board member of many prestigious journals including Computer Standards and Interfaces, Journal of Universal Computer Science, Computing & Informatics, Electronic Government—an International Journal etc. Prof. Mishra has extensive experience in distance and online education related to Computing and Management Information System courses. He had received the excellence in online education award by U21Global Singapore. He is the recipient of various scholarships including the national merit scholarship and the department of information technology scholarship of the Government of India.

Ziya Karakaya is a Dr. Instructor in Computer Engineering Department at Atilim University. He received BS degree in Mathematics, and MS degree in Computer Education and Instructional Technologies, both from the Middle East Technical University. He has completed his PhD at the Modeling of Engineering Systems (MODES) with the main research area of Computer Engineering, at Atilim University. His research interests include Parallel Computing, Distributed Computing, Virtualization, Cloud Computing, Object Oriented Software Engineering and Internet Programming. He is one of the authors of the book about Online Learning and Assessment (in Turkish).He has supervised two MS thesis and has many scientific writings, either published as Journal papers or as conference proceedings.