Experience with Teaching PDC Topics into Babeş-Bolyai University's CS Courses

Virginia Niculescu[1] and Darius Bufnea[2]

[1] Faculty of Mathematics and Computer Science
Babeş-Bolyai University, Cluj-Napoca
vniculescu@cs.ubbcluj.ro
[2] bufny@cs.ubbcluj.ro

Abstract. In this paper, we present an analysis of the outcomes of teaching Parallel and Distributed Computing within the Faculty of Mathematics and Computer Science from Babeş-Bolyai University of Cluj-Napoca. The analysis considers the level of interest of students for different topics as being determinant in achieving the learning outcomes. Our experiences have been greatly influenced by the specific context defined by the fact that the majority of the students are already enrolled into a software company either as interns in an internship program or as employees. The level of interest of students for a specific topic is determined by the development of the IT industry in the region. The learning activity is in general influenced by this specific context, and a new, high demanding topic as Parallel and Distributed Computing is even more influenced, when is to be taught to the undergraduate level. This analysis further leads to a more general analysis on the appropriateness of introducing PDC topics, or other relatively advanced topics, to all undergraduate students in CS, or to consider newly defined educational degrees.
Keywords: Parallel and distributed programming, curricula, courses, undergraduate, IT industry, workforce.

1 Introduction

Recent years have brought an explosive growth in multiprocessor computing, including multi-core processors and distributed data centers. The mass marketing of multi-cores and general-purpose graphics processing units induces the possibility for common users to rely on its effectiveness. This enforces the software developers to efficiently use it, and also to contribute to the technology development.

As a consequence, there is a clear need for students general education courses computing related to be aware of the role that parallel and distributed computing technologies play in the computing landscape.

The ACM/IEEE Curricula 2013 Report [9] and the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [5], argue that the undergraduate computer science programs should include topics in parallel and distributed computing (PDC).

This approach implies important changes and their impact should be carefully analyzed.

Babeş-Bolyai University is the biggest university of Romania, even though its host city, Cluj-Napoca, is only the second largest in Romania (after Bucharest). It is also the oldest university of the country, but at the same time it is a dynamic and constructive institution well integrated into society and oriented towards the future.

The Faculty of Mathematics and Computer Science follows the Bologna system of study. In the last decade the number of students attending Computer Science has continuously increased, exceeding in 2016-2017 academic year 2600 students enrolled in undergraduate, graduate and doctoral programmes.

Parallel and distributed computing topics have been studied at our faculty especially at master level programs, but still there were some modules related to concurrency, multi-threading and client-server application, RPC, RMI included into some curriculum courses. In 2015 a dedicated compulsory course *Parallel and Distributed Programming* has been introduced for students in the third year of study. Before this, it was also an elective course *Paradigms and Techniques of Parallel Programming* that aimed to introduce the main concepts and paradigms of parallel programming; the curriculum changes for this course addresses more advanced topics.

In 2014 a master program with the title *High Performance Computing and Big Data Analytics* has been included into academic program of our faculty. This offers the students the possibility of acquisition of theoretical, applicative and practical knowledge in high performance computing but also on using HPC in data analysis.

In terms of PDC infrastructure, the university owns a hybrid (High Performance Computing + Cloud Computing) cluster, acquired in 2015, capable of reaching 40 Tflops in Rmax (sustained) and 62 Tflops in Rpeak (theoretical). The HPC component has 68 nodes with a total of 1360 physical computing cores for the whole HPC component. Also, 6 nodes are hosting an additional Intel Phi coprocessor, while 12 others are equipped with 2 Nvidia Tesla K40 GPU each.

This paper intends to present the evolution of courses that include modules from the topic of PDC. Also a broad analysis of the outcomes of these teaching subjects in correlation to the level of interest of the students for them is presented.

The next section presents the existing courses, and modules, and section 3 describes the particular context of our region that has an important influence on the level of interest of the students for different subjects. Section 4 shows the analysis results and their correlation, and the final conclusions are presented in section 5.

2 The Subject of Analysis

As in other reports on various curricula, we will use Bloom's classification [1] (B class) considering also a correlation to ACM level of mastery. So, we will use the following classes:

- K = Know the term ($\Leftrightarrow$ Familiarity)
- C = Comprehend so as to paraphrase/illustrate ($\Leftrightarrow$ Usage)
- A = Apply it in some way (requires operational command) ($\Leftrightarrow$ Assessment)
- N = Not in Core, but can be in an elective course

The courses from the undergraduate curriculum that address Parallel and Distributed Computing topics are presented in Table 1. The number of students enrolled is about 180 for a compulsory course and varies between 30 and 70 for an elective one.

Table 1. Undergraduate courses addressing PDC topics

Course name	Semester of study	ECTS	Hours per week (course, seminar, lab)
Operating Systems (OS)	2	5	2,1,2
Advanced Programming Methods (APM)	3	6	2,2,2
Computer Networks (CN)	3	6	2,0,2
Systems for Design and Implementation (SDI)	4	6	2,1,1
Parallel and Distributed Programming (PDP)	5	6	2,1,2
Paradigms and Techniques of Parallel Programming (elective) (PTPP)	6	7	2,0,1

At the master level there are other courses related to PDC from which we may mention the following: 'Formal Models of Concurrency', 'Operating Systems for Parallel and Distributed Architectures', 'Models in Parallel Programming', 'Functional Parallel Programming for big Data Analytics', 'Workflow Systems', 'Grid, Cluster and Cloud Computing', 'Algorithms, Models, and Concepts in Distributed Systems', 'GPU and Distributed Architecture Computing'. Most of these courses are part of the *High Performance Computing and Big Data Analytics* or *Distributed Systems in Internet* graduate programmes' curricula.

The next two tables emphasis (to a great extent but not completely) the PDC topics discussed in these courses. Table 2 presents some general topics with the focus on concepts, and Table 3 shows the topics for which concrete implementations are analysed.

A certain topic could be introduced in a certain course where the corresponding learning outcome belongs to (K) or (C) in Bloom's classification, and then it is discussed to a following course where the learning outcomes are moved to a

Table 2. General conceptual topics. The table emphasizes the main concepts associated to the corresponding courses were they are discussed.

Topic	B class	Courses
SISD/SIMD/ SPMD/ MIMD	K	PDP,PTPP
Computation decomposition strategies	C	PDP, PTPP
Data Distribution	C	PDP, PTPP
Functional Decomposition	C	PDP, PTPP
PCAM methodology	K	PTPP
Synchronisation Concepts	C	OS,PDP,PTPP
processes, pipe, fifo	C	OS
critical section, race condition	C	OS, PDP, PTPP
mutex, semaphore, monitor	C	OS, PDP, PTPP
barriers, conditional variable	C	PDP, PTPP
deadlock, livelock	C	OS, PDP, PTPP
starvation, fairness	K	OS, PDP
Tasks and threads	C	APM, CN, PDP, PTPP
Non-determinism	C	PDP, PTPP
Performance metrics	C	PDP, PTPP
Speedup, Efficiency, Cost	C	PDP, PTPP
IsoEfficiency	K	PTPP
PRAM	C	PDP, PTPP
Brent Theorem	K	PTPP
Dependencies	A	PDP, PTPP
Task graphs	K	PDP, PTPP
Divide & conquer (parallel aspects)	A	PDP, PTPP
Recursion (parallel aspects)	C	PDP, PTPP
Master-slave	A	CN, PDP
Pipeline (parallel aspects)	A	PDP, PTPP
Scalability	K	PDP, PTPP
Granularity	K	PDP, PTPP

more advanced level by Bloom's classification. So, for example, the semaphore concept is first introduced at *Operating Systems* course considering an outcome of class (K), and then is discussed again at *Parallel and Distributed Programming* course where a more deeply understanding is provided and also it is used in the context of the current implementations in Java or C#.

Examples of parallel algorithms are given especially at the *Paradigms and Techniques of Parallel Programming* course. In the curriculum there is a course of *Data Structures and Algorithms* – DSA, but at the moment the possible parallelization of the algorithms is not treated there. The parallelization techniques are introduced at PDP course, and then they are detailed at PTPP course. Still, time-complexity and space-complexity issues for sequential algorithms are analysed at DSA, and so when the parallel programs performance metrics are introduced we may start from some already introduced concepts.

Table 3. Specific topics ⇒ implementation oriented.

Topic	B class	Exemplification	Courses
Shared memory	A	Java, C/C++, C#	OS ,PDP
Thread/Task spawning	A	Java, C/C++, C#	OS, APM, SDI, CN, PDP
Executors,Threads pools	A	Java	APM, PDP, PTPP
Work stealing	K	Java	PTPP
Synchronisation tools	A		OS,PDP,PTPP
mutex, semaphore	A	Java, C/C++, C#	OS, PDP, PTPP
barriers, conditional variable	A	Java, C	PDP, PTPP
Asynchrony	A		PDP, PTPP
Futures/promises/Async tasks	A	Java, C++	PDP, PTPP
Streams	A	Java	PDP, PTPP
Parallel loop	A	C/C++,OpenMP	PDP, PTPP
Hybrid	C	CUDA/C++	PDP, PTPP
Distributed memory	C		PDP, PTPP
Message passing	C	MPI, C/C++	PDP, PTPP
Broadcast, Scatter/Gather	C	MPI, C/C++	PDP, PTPP
Client Server	A	C/C++, Java, C#	SDI, CN, PDP
RPC, RMI	A	Java, C#	SDI
P2P	K		PDP

3 The Context of the Analysis

Cluj-Napoca is now the most important educational and economic center in Transylvania and the second largest in Romania after Bucharest and it has a long standing tradition in IT development - the beginnings of the computer sciences in Cluj are situated around the years 1960. According to a recent study, done by iTech Transilvania Cluster, Cluj has the highest density of IT employees in Romania, 1 in 25 employees working in this industry [11]. A decade ago, when most of the IT companies were founded, the main activity of Romanian software industry was outsourcing. On the long run this had scalability issues since the number of potential new employees, although raising, couldn't satisfy the increasing market demand. Another important factor was that man-day rates in neighbor countries were very competitive. What tip the balance in favor of Romanian IT specialists is that half of them are software developers and that almost 90% of them speak English. The economic factor also had an important influence in this together with the focus on education proven by students' results in Informatics and Math Olympiads or design competitions over the years [8]. Outsourcing is still the main activity but the current trends are moving towards innovation (startups or developing of own products) and providing high level roles (such as solution architects, business analysts or project managers) and business knowledge to clients in order to achieve added value for the constantly increased rates.

The cooperation between students and companies starts usually with an internship program (required by the academic curriculum), which is followed by real employment before graduation. So, when we discuss the impact of some changes in the academic curricula, we have to consider the fact that the feedback that we obtain from students, includes also, indirectly, feedback from industry.

There is a known gap between academic world and the industry. The industry is productivity oriented with some expense in the software quality. Consolidated frameworks, libraries and APIs are frequently used in the development, alongside development tools that are required in a productive environment.

This is why there are companies that can afford to hire students even from their first years of study, and encourage them for early employment with the promise that they will learn "all they have to know" at the workplace. (Of course their perspective is on the present day, without considering the future.). This comes with the drawback that students focus less on obtaining general knowledge in computer science and they start learning/using only specific fields of computer science (database, user interface development, etc.). Often enough students that are not yet employed are reluctant to learn things that wouldn't help them during an internship or job interview.

The development is very often based on *"applying patterns... "* but the meaning of the term – pattern, in this context, is not the same with the one used in [3], where it is used to emphasise the situation when a design pattern (a well defined solution) is used in a new context in a creative way. Here, we have to understand that the software is built using specific framework and technologies by composing components based on some specific recipes.

So, many times the developers build the software by using some tools and without a deep understanding of what they are really doing. The leading questions are: *"how to do"*, *"what to apply"* and not *"why"*, or *"what is hidden behind"*.

It is important to say that the described situation has a large spreading, but it is not generalized. Not all companies adopt this kind of development, but there is a large majority that has an important influence.

The university purpose is to prepare the young minds for whatever is out there in the industry without limiting the knowledge to a specific area. The graduates need to acquire enough information from all the fields in such a way that they can face the industry switches without too much effort, having the basics in place.

4 The Analysis

In order to move from "traditional" development to distributed development, the students need to posses the most basic knowledge of development. It is always easier to 'build' on top of something that has solid ground. The challenges that come from the current industry context (students start focusing on employment rather than finalising their studies) trigger different approaches regarding teaching techniques:

– Before moving to a topic that requires specific background, we need to validate that students have this background; this comes with the drawback that some of the students that already have the background cannot move faster to the specific distributed programming topics, and they become distrustful.
– Some of the basic courses have been condensed or made optional in order to accommodate the students needs to have the bare minimum knowledge for employment.

In our study we went from the premise that the success of introducing new topics in the curriculum, and consequently achieving the desired learning outcomes, depend in a great measure on the level of interest of the students in that topic. In the context described in the previous section, we are aware that the level of interest of the students for one topic and another depend very much if they are working for a company or not, and when they have started to do this.

Table 4. Level of interest for general conceptual topics.

Topic	Level of interest 2nd year	Level of interest 3rd year
SISD/SIMD/ SPMD/ MIMD	1	1,87
Computation decomposition strategies	1	4.5
Data Distribution	1,37	3
Functional Decomposition	1	3,12
PCAM methodology	1	3,25
Synchronisation Concepts	2,56	3,75
processes, pipe, fifo	3,75	3
critical section, race condition	1,5	3,25
mutex, semaphore, monitor	2,25	3,18
barriers, conditional variable	1,62	3,25
deadlock, livelock	3	4,37
starvation, fairness	1	3
Tasks and threads	3,75	4.5
Non-determinism	1,31	3,87
Performance metrics	2,87	3,12
Speedup, Efficiency, Cost	3,06	4,25
IsoEfficiency	1,25	1,75
PRAM	1	2
Brent Theorem	1	1.75
Dependencies	2,37	3
Task graphs	1,62	1,87
Divide & conquer (parallel aspects)	3,68	2,75
Recursion (parallel aspects)	3,81	2,62
Master-slave	1,5	4,25
Pipeline (parallel aspects)	3	3,37
Scalability	2,06	3
Granularity	1,37	2

Table 5. Level of interest for the specific topics.

Topic	Exemplification	Level of interest -2nd year	Level of interest -3rd year
Shared memory	Java, C/C++, C#	2,12	3,62
Thread/Task spawning	Java, C/C++, C#	2,93	4,12
Executors,Threads pools	Java, C#	2,31	4,5
Work stealing	Java(ForkJoin)	1,37	4
Synchronisation tools			
mutex, semaphore	Java, C/C++, C#	2,75	3,75
barriers, cond.variable	Java, C/C++	2,25	3,25
Asynchrony			
Futures/promises		1,5	4,12
Async tasks	Java, C++	3	3,12
Streams	Java	2,06	4,25
Parallel loop	C/C++,OpenMP	3,25	3,75
Hybrid	CUDA/C++	2,25	2
Distributed memory			
Message passing	MPI, C/C++	2	2,12
Broadcast, Scatter/Gather		3,25	2
Client Server	C/C++, Java, C#	4,25	4
RPC, RMI		3,68	2,75
P2P		1,75	2,75

So, the first steps of our investigation was to find out the level of interest of students for the topics specified in Tables 1 and 2. For each topic they have been asked to choose a value between 1 and 5 (1 represents the lowest level of interest and 5 represents the highest level of interest). The students of the second and third year of study have been asked to participate in our analysis. The results are reflected in Table 4 and Table 5.

The differences between the two categories are given by the fact that the students of the second year haven't studied yet some of the questionnaire included topics, but also, by the distribution of their employment per year of study:

- 10% students of the first year of study,
- 25% students of the second year of study,
- 60% students of the third year of study,
- 75% students at the end of the third year.

(The students have a mandatory internship of 3 weeks between the 2nd and the 3rd year, and this is the moment when almost all get hired.)

Parallel programming is not easy if we have to control threads/processes executions, synchronization, communication, etc. As the level of abstraction is increasing, the things could become simpler, but an associated performance degradation could appear, too[6]. So, we may work with frameworks and libraries that make the parallel programming easier and probably more attractive for students.

On the other hand, this way the main concurrency issues will not be well understood. Also, in contexts where the performance is a critical issue, the ability to work only with high level frameworks would not be enough.

There is a large interest from students to learn APIs and tools that implicitly use parallelization without the explicit control from the programmers (Java parallel streams, Scala parallel collection, OpenMP). This approach has the advantage of offering a simple and rapid development and also offers a high degree of confidence in the correctness of the resulted code. It is known that parallel programming is sensitive to hidden errors that are very difficult to detect and hence to debug. On the other hand the programmers are limited to the defined constructions, and also cannot control very well the level of performance.

The analysis includes also the results obtained by the students for the tests and assignments of the curriculum required course *Parallel and Distributed Programming*. The evaluation for this course has been based on the followings tests and assignments:

1. Practical works/assignments (relative short problems that should have been implemented using discussed strategies and technologies);
2. Multithreading practical test (a problem of a medium complexity that had to be solved using threads – explicit thread creation);
3. MPI practical test (a simple problem that had to be solved using MPI);
4. Theoretical test (written exam).

The corresponding results for these evaluations are presented in Figure 1. Practical works included:

- some multithreading examples in C/C++, Java, and C#,
- a very simple CUDA example,
- a client-server application that also includes asynchronous tasks, and
- a simple MPI example.

The students had to solved them independently, at home, and then present them to the instructors.

The practical tests assume solving a given problem in a given period of time, on the students' laptops – if they chosen this way; computers from the faculty laboratories could also be used.

From these results, we may consider that MPI programming have been proved difficult for students. A deeper analysis emphasizes that, in fact, the interest of the students in learning MPI was low.

The students are much more confronted to using multithreading programming, for different types of applications, and this leads to a much better knowledge acquisition. This includes working with threads directly or using APIs such as: OpenMP or Java Streams.

The theoretical evaluation shows the fact that even the students declare that they have certain interest in studying concepts, still either because they don't have enough time (being involved in others activities as working for companies) or because they looses their ability for theoretical approaches, the results are not very good.

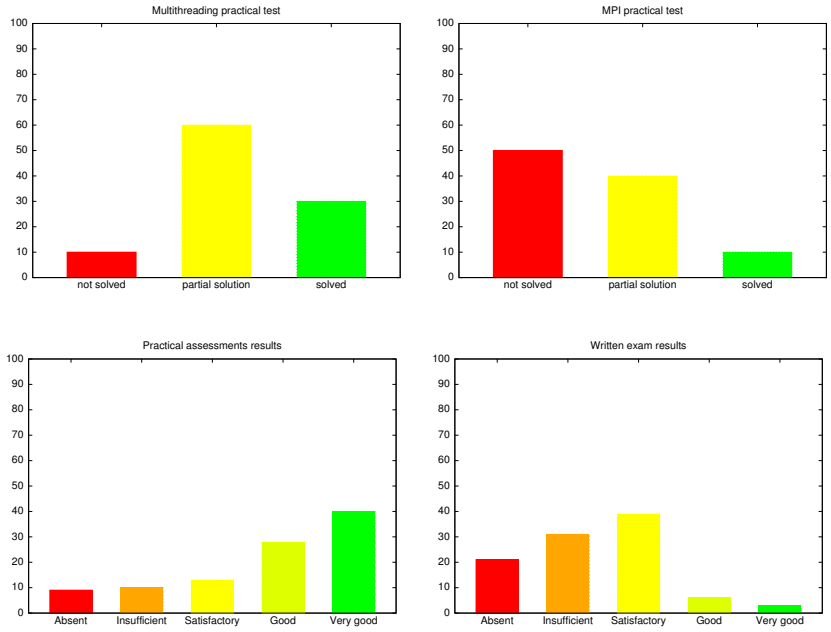

Fig. 1. Evaluation results.

Since the students are soon to be enrolled in productive activities, they are much oriented on practical skills. Hence, the results obtained for practical works in correspondence with the results of the theoretical test confirm this situation.

For the elective course *Paradigms and techniques of parallel programming* (PTPP) the students have been allowed to choose a paradigm and a technology for solving a problem in a parallel way. This problem could have been chosen from a list of proposed problems, but the students also had the possibility to propose new ones. From the 35 students that attended this course in the current academic year (2016-2017), only one has chosen MPI as a programming model. All the others chosen to go on the multithreading paradigm and to use different implementation languages (Java – 23, C# – 5, C++ – 4, Scala –2). The project also required to do a written documentation that includes design pattern oriented analysis of the design decisions, theoretical performance evaluation and results of a set of the empirical testing.

At this elective course, some techniques of algorithm parallelization are discussed, and some concrete examples for well know problems are analyzed (sorting, searching, matrix multiplication algorithms, ...). Even if the level of interest for these was not formally evaluated, we may say that the students consider them interesting. These techniques have been used in a certain measure by the students in the development of their projects.

Even for this elective course, which is chosen by the students that have an increased interest in parallel programming, the students' choices are influenced

by the mainstream technologies and their abilities in working in a specific programming language. These abilities are on their turn influenced by their personal experience, which is, in a vast majority of cases, driven by their employers and industry demands, not by the academic environment.

We have also received some informal feedback from direct discussions with the students that emphasizes the fact that an orientation on distributed aspects of programming is considered by them much useful than an orientation on parallelization techniques and tools.

5 Conclusions

The main conclusion of our experience with teaching PDC topics is that even if they are very necessary and important to be studied, due to the last development of systems architectures and of the associated programming, it is also very important to take into consideration the latest approaches and paradigms applied in the IT industry. The need for high productivity induces some changes in the way the programming activity and software systems construction are developed. All these lead to a new category of software developers which are not supposed to understand all the components that they usually assemble. An adapted and simplified curriculum should be in this case specified. In such a curriculum some PDC topics should be included, but in a pragmatic, usage-oriented way – how to use parallel programming frameworks/libraries, etc.

Also, there are some topics – as MPI – that have a great importance for the well understanding of some basic concepts of Parallel Programming, but they are not yet very much used in the industry. This leads to a low level of interest for this topic from the undergraduate students.

Also, the acquisition of the theoretical concepts and general principles is not very good, since our students are now, very much oriented on achieving practical abilities.

Master students that choose a specialization that includes High Performance Computing, have of course, a much higher degree of interest and opening to fields as Scientific Computation, Models of Computation, or Correctness and Formal Methods.

The premise of our study was that the success of introducing new topics in the undergraduate curriculum, and most importantly achieving the desired learning outcomes, depend in a great measure on the level of interest of the students in that topic. This premise proved to be correct.
On the other hand, the level of interest on different topics of Parallel and Distributed Computing depends very much on the students' levels. The distinction between undergraduate and master students is very clear, but between undergraduates we may emphasise at least two classes of interest.

A solution could be based on moving more topics on the elective courses.
Another, more complex solution, would involved also other Computer Science fields and introducing a new defined educational degree. A proposal that comes

from Cluj Innovation City Project [10] is to develop *Vocational Studies*. The proposal claims that this way an important part of the IT industry employees could come directly from an IT related vocational curricula, and this would reduce part of the pressure on the employment market, but most importantly would engage young people into the industry in their early 20ties. (The drawbacks of this proposal have not been studied, yet.)

There is an important trend in the software development in using Parallel and Distributed Computing and, at the same time, in using in a more efficient way the present hardware resources. There is also a wide acceptance that "Parallelism is the future of programming". Still, we may paraphrase the title of the paper of Domenico Talia: "Parallel computation still not ready for the mainstream" [7] and say: "Mainstream still not ready for [all kind of] Parallel Computation".

References

1. B.S. Bloom, M.D. Engelhart, E.J. Furst, W.H. Hill, D.R. Krathwohl *Taxonomy of educational objectives: The classification of educational goals*. Handbook I: Cognitive domain. New York: David McKay Company.(1956).
2. David J. John, Stan J. Thomas. *Parallel and Distributed Computing across the Computer Science Curriculum*. Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International. DOI: 10.1109/IPDPSW.2014.121.
3. E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. 1994. Addison-Wesley Longman Publishing Co., Inc..
4. Guoming Lu et al. *Integrating Parallel and Distributed Computing Topics into an Undergraduate CS Curriculum at UESTC*. Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International DOI: 10.1109/IPDPSW.2015.66
5. Sushil K. Prasad et al. 2012. *NSF/IEEE-TCPP Curriculum on Parallel and Distributed Computing - Core Topics for Undergraduates* - Version I, `http://cs.gsu.edu/~tcpp/curriculum/`, 55 pages.
6. David B. Skillicorn, Domenico Talia. *Models and languages for parallel computation*. Journal ACM Computing Surveys, Volume 30 Issue 2, June 1998, pp. 123-169.
7. Domenico Talia. *Parallel computation still not ready for the mainstream*. Communications of the ACM. Volume 40 Issue 7, July 1997, pp. 98-99.
8. Bryan Martin. *The Silicon Valley of Transylvania*. Apr 6, 2016, `https://techcrunch.com/2016/04/06/the-silicon-valley-of-transylvania/`.
9. *** ACM Curricula 2013 Report. `https://www.acm.org/education/CS2013-final-report.pdf`. pp 144-154.
10. *** Cluj Innovation City. `http://www.clujinnovationcity.com`, retrieved May 10, 2017.
11. *** iTech Transilvania Cluster study by ARIES, `http://itech.aries-transilvania.ro/`, retrieved May 10, 2017.