Introducing computational thinking, parallel programming and performance engineering in interdisciplinary studies

Eduardo Cesar, Ana Cortés, Antonio Espinosa, Tomàs Margalef*, Juan Carlos Moure, Anna Sikora, Remo Suppi

Computer Architecture and Operating Systems Department, Universitat Autònoma de Barcelona, 08193 Cerdanyola del Vallès, Spain

Abstract:
Nowadays, many fields of science and engineering are evolving through the joint contribution of complementary fields. Computer science, and especially High Performance Computing, has become a key factor in the development of many research fields, establishing a new paradigm called computational science. Researchers and professionals from many different fields require knowledge of High Performance Computing, including parallel programming, to develop fruitful and efficient work in their particular field. Therefore, at Universitat Autònoma of Barcelona (Spain), an interdisciplinary Master on ‘‘Modeling for Science and Engineering’’ was started 5 years ago to provide a thorough knowledge of the application of modeling and simulation to graduate students in different fields (Mathematics, Physics, Chemistry, Engineering, Geology, etc.). In this Master’s degree, ‘‘Parallel Programming’’ appears as a compulsory subject because it is a key topic for them. The concepts learned in this subject must be applied to real applications. Therefore, a complementary subject on ‘‘Applied Modeling and Simulation’’ has also been included. It is very important to show the students how to analyze their particular problems, think about them from a computational perspective and consider the related performance issues. So, in this paper, the methodology and the experience in introducing computational thinking, parallel programming and performance engineering in this interdisciplinary Master’s degree are shown. This overall approach has been refined through the Master’s life, leading to excellent academic results and improving the industry and students appraisal of this programme.

1. Introduction
Many fields of science and engineering are applying techniques and recent advances in complementary fields. In this interdisciplinary context, researchers and professionals with greater knowledge of problem modeling and High Performance Computing (HPC) are in high demand from companies and research centers. Since 2011, Universitat Autònoma of Barcelona hosts a Master’s degree in Modeling for Science and Engineering to provide these kinds of professionals to those companies and centers.

The Master’s involves an interdisciplinary collaboration among professors from various departments; mainly Physics, Mathematics, and Computer Architecture and Operating Systems. The main objective is to provide the average science graduates with mathematical and computational tools to treat different types of scientific and/or technological problems. It covers a large range of problems, introducing many different approaches and tools. In particular, the students are provided with the basic knowledge to be able to model a physical system involved in some problems, to represent the model mathematically and to solve the problem applying different methods such as differential partial equations, optimization, time series, and related methods.

This work has been partially supported by MINECO-Spain under contracts TIN2014-53234-C2-1-R and TIN2014-53172-P.
* Corresponding author.
E-mail addresses: eduardo.cesar@uab.es (E. Cesar), ana.cortes@uab.es (A. Cortés), antoniomiguel.espinosa@uab.es (A. Espinosa), tomas.margalef@uab.es (T. Margalef), juancarlos.moure@uab.es (J.C. Moure), anna.sikora@uab.es (A. Sikora), remo.suppi@uab.es (R. Suppi).

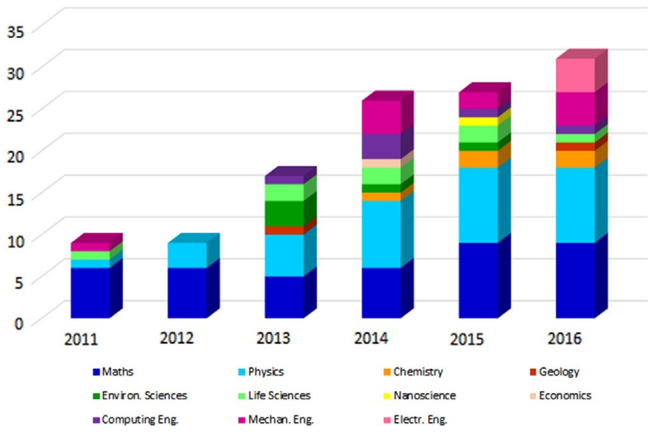

Fig. 1. Evolution of the number of students and their background degrees.

The students learn how to analyze their particular problems and think about them from a computational perspective, i.e. formulating a problem and expressing its solution(s) in a way that a computer can effectively carry out.

Finally, they have to think in parallel solutions and learn HPC technology to evaluate and improve the performance of models, applications and simulators. In order to include all these different issues in the Master's, we define three training pillars:

• Definition of complex systems
• Mathematical representation and resolution of these systems
• Computational thinking for parallel systems and performance engineering.

In this master, advanced programming topics such as performance engineering are specifically brought to science background postgraduate students. These skills are traditionally out of a science degree and are not commonly found in master's studies elsewhere.

Students perceive the extensive and interdisciplinary training offered by the Master's as a significant asset in their curricula. This is enforced by the presentations that several companies, which apply the introduced techniques and methods in their everyday business processes, make to students as part of the subjects regular activities. In this way, students can be aware of the significant impact of such techniques and methods on the productivity of a specific company. Moreover, most of these companies offer short internships and often hire students from the Master's. This fact encourages students to enroll in the Master's and is crucial in its success.

Students enroll from different degrees, such as Mathematics, Physics, Chemistry, Engineering or Computer Science. The studies in this Master's have been very successful and are attractive to students from many different countries and with different backgrounds. The number of applications for this Master's is growing every year, and in the current 2016–2017 course, several student applications were rejected since the maximum number of registered students has been reached. Fig. 1 shows the evolution in the number of enrolled students from year 2011–2012 until the current year 2016–2017. The Master's started with just 8 students, but it has been growing continuously and now it has more than 30 registered students. Fig. 1 shows a clear indication of the high impact of the studies, perceived from the point of view of the students enrolled every year.
Among the students, the most common backgrounds are mathematics and physics; but other backgrounds, such as chemistry, life ming, they usually do not show any significant skills on computational thinking or performance engineering. So, we plan to show them a complete view, from the problem definition, to the performance analysis and tuning.

• We want to introduce a complete view of the existing parallel programming paradigms, analyzing the performance aspects involved. We do not focus on a single approach, but we present all the most commonly used approaches.
• To make it useful, it is necessary to apply the presented concepts, methods and tools to a set of real cases from different fields of science and engineering.

In this context, we aim to provide good training in parallel programming and a solid experience of efficiency analysis of the implementation of several real applications. For that purpose, the Master's contains, among others, two subjects, namely, Parallel Programming and Applied Modeling and Simulation. These two subjects provide a base for computational thinking and for developing efficient solutions addressed to complex scientific problems. Moreover, the contents and activities in both of them have been planned using a well-established teaching methodology.

The outstanding academic results obtained in these subjects throughout the Master's life, as well as their excellent reception among students, have encouraged us to share their structure, contents and successful strategies in this work.

This paper focuses on the description of the training on parallel programming and applied modeling and simulation offered to the students. In order to introduce the main differences of the proposed teaching approach to similar existing systems, Section 2 analyzes the related work. In Section 3, we present the basic teaching methodology applied to the subjects we focused on and highlight relevant concepts on parallel programming shown to the students in order to establish a common framework. Section 4 describes the parallel programming approaches presented to the students and the different activities planned. Then, Section 5 presents some examples of case study applications that are introduced to the students along with proposals for further developments. Section 6 resumes some global academic results. Finally, Section 7 presents the main conclusions of this teaching experience.

2. Related work and experiences

Already in the 90s, several scientists [11,40] realized the need for training computational scientists due to, among other reasons, the dramatic effects expected from parallel computing development on computers' performance and capabilities. This training should be focused in providing computational skills for solving complex problems to professionals of different areas (Chemistry, Physics, Mathematics, and also Computer Science). After 20 years, parallel computing is having the expected effects and training computational scientists is still a relevant discussion issue [2].

Parallel programming is significantly more complex than sequential one and, consequently, teaching it is a challenge, especially in the case of students with little computer science background. For this reason, general proposals for introducing parallel thinking and programming, such as [20,18], are still presented and discussed in education forums. Contents presented in these proposals include lessons about the main elements related to parallel systems and parallel programming. Moreover, there are also works, such as [32], proposing strategies for simplifying the understanding of parallel computing concepts by non-computer scientists.

Based on this background, and taking into consideration industry and research requirements, many universities have implemented postgraduate programs for training computational scientists. Most of these programs [25,5,29,24,22,16,8] explicitly include subjects on parallel thinking and programming, even though, there are programs which do not include related contents explicitly [4,23].

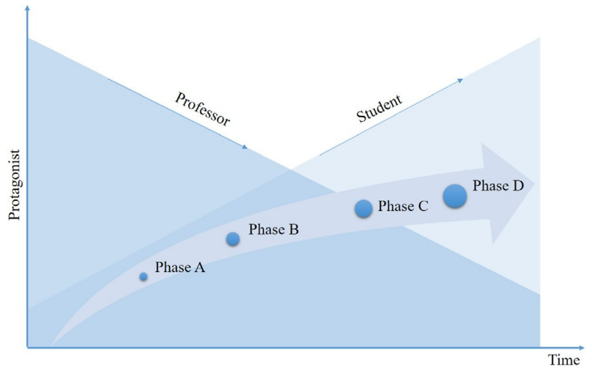

Fig. 2. Two-dimensional teaching methodologies.

Generally, these are two-year programs that offer one single subject dedicated to parallel programming. This topic usually introduces the main characteristics of parallel architectures and heavily relies on practical exercises, since general computational concepts have been introduced in other subjects. In addition, in most cases the course is focused only in one programming paradigm (shared memory [24] or message passing [22]) or a high level language [25].

There are many similarities among these postgraduate programs and our proposal. For example, all the proposals are based on parallel teaching foundations and give great importance to practical training. There are, though, some significant differences. First, it is worth mentioning that we are offering two subjects related to parallel programming and computational thinking in our master program, even though it is a one-year programme. One of the subjects (Parallel Programming) is compulsory, so everybody is getting the foundations, and the other (Applied Modeling and Simulation) is complementary, for the students interested in getting a deeper knowledge. Second, the Parallel Programming subject covers all current parallel programming paradigms, i.e., shared memory (OpenMP and Massive Parallel Processors (GPUs)) and message passing (MPI). We think that this approach gives students a wider view of parallel programming on currently available architectures, although, it may sacrifice some degree of details.

We are also providing innovative content introducing recent parallel programming extensions like Cilk Plus, and new industrial standards like OpenACC, addressed to simplify parallel software engineering. Concerning Modeling and Simulation topics, there are proposals on the subject in the above programs, but they are mostly focused on specific areas. For example, in many of them there are courses on modeling and simulation of non-linear systems, mathematical & numerical modeling and simulation (including simulation on HPC architectures and commercial software) or in specific fields of knowledge (e.g. Biological Systems with Differential Equations, Fluids and Soft Matter, Combat Modeling, Simulation Modeling in Transportation Networks, etc.).

We consider that our modeling and simulation training on high performance computers using ABM allows students to analyze the potential of High Performance Simulation on complex models that are close to their area of expertise.

3. Interdisciplinary teaching methodology

Modern teaching methodologies use two dimensional models [15] to describe the correlations between the degree of knowledge objectivity and the amount of teaching activities done by Fig. 2. Two-dimensional teaching methodologies. teachers and students in the subject. These models typically expose a balance between formal academic and personal student experiences and provide a way to distribute prominence between the teacher and the student. Hence, we can consider different possible orientations in teaching methodologies: teacher-centered expository, balanced interactivity between the teacher and the students, and, finally, student-centered discovery-based.

We have analyzed the most suitable methodology for our interdisciplinary Master's programme. To do so, we have evaluated the learning pyramid definitions [17], the knowledge competences described in the Tuning project [43], and the recommendations given in the white book of Computer Science Higher Education by the Spanish National Agency for Quality Assessment and Accreditation (ANECA) [44]. Moreover, we have also taken into consideration that students enrolled in this Master's programme have different expertise levels of computer science knowledge. Consequently, more experienced students need methodologies that foster student discussion and practical experimentation, while less proficient students need direct teaching methods to effectively be introduced to relevant technical topics.

In this context, both of the subjects that we present in this paper are based on an even distribution of the expository and interactive methodologies. The students' learning process is shown in Fig. 2, where the leading role between teachers and students is balanced throughout the duration of the subject. We define a list of stages (A–D) in each of the subjects to which we apply our teaching methodology. In the early stages, such as A, we apply more direct, classic teacher-centered methodologies. In the last part of the subject, such as D, the general knowledge assimilated by the student is more mature. Then, we apply student-centered methodologies exploring practical activities and promoting interaction between the students. Taking into account the characteristics of the Master's degree, we are using the following teaching strategies and techniques:

- **Lecture session:** the teacher exposes the most essential topics of the subject to provide a broad, common knowledge background for all of the students.
- **Lab session:** practical, interactive experimental activities, usually done in groups. Work is oriented to gain experience in the usage of tools and practically resolve theoretical concepts with the help of guided exercises.
- **Conceptual maps:** used to define strategies to create an information structure that will generate an adequate hierarchy of the concepts taught in the subject.
- **Case studies:** activities based on the study of well-known, practical problems. The analysis of a given problem that is developed by the students is compared with an existing solution to extract relevant conclusions.

• **Bibliography review:** a list of documents is given to the students, so that they have a basic corpus of documents associated with the subject. The list contains a reduced number of books and articles that the students must know and use throughout the subject. An extra reference document list is also provided allowing the students to get more insight on specific interests.

• **Experimental portfolio:** a list of work tasks to be done during the practical sessions which the students can use to auto-evaluate their own knowledge and look for extra information from the teacher or other sources.

• **Invited conferences:** Special lectures given by invited speakers where discussion is open for relevant subject topics. The objective is to foster the interactions between the students and the professional experts.

The rest of the paper describes the objectives, principles and detailed methodology used in two specific subjects of the Master's focused on computational thinking and performance engineering, namely Parallel Programming and Applied Modeling and Simulation. These subjects, which are focused on High Performance Computing, provide the basic concepts to introduce the students to computational thinking, solving a given problem in a parallel and efficient way and learning to apply the principles of performance engineering to scientific or industrial applications. Taking into account the chosen teaching methodology, the Parallel Programming subject starts by providing a general programming background of C language. This is done through the use of introductory lectures and programming labs. Then, we present the basic theoretical concepts of parallel programming by combining lectures on computer architecture and a selected bibliography review. Next, students must survey a list of relevant parallel algorithms with some general introductory lectures and are given an experimental portfolio to analyze matrix multiplication in practice. From here, students must analyze a selected list of case studies of parallel computational patterns like map, reduce and stencil. In this part of the subject, they have to apply the computational thinking concepts to an experimental portfolio with examples like parallel prefix and convex hull. Finally, they receive a conceptual map of programming paradigms: shared memory, message passing and accelerator-oriented massively parallel programming. These lectures are complemented with several lab sessions where students use performance analysis tools to develop a full performance engineering cycle for example applications.

The Applied Modeling and Simulation subject adopts a similar methodological approach. First, the students need to attend a short number of lectures for the development of a simulation model. Then, they are provided with the explanation of several case studies, such as emergency evacuation and meteorological services where they have to compare their own designs with already existing solutions. Finally, the students must apply performance engineering principles in several lab sessions addressed to analyze the performance of the simulation process. In the next sections, we are going to provide more detailed descriptions of the particular objectives, contents and how the planned activities are put into practice for the two subjects, Parallel Programming and Applied Modeling and Simulation. Finally, we provide some conclusions obtained from the implementation of these subjects over the last few years.

4. Parallel programming

Parallel Programming is a core subject in this interdisciplinary Master's. The first challenge to tackle is to set a common practical background for the students. The students of this Master's typically have some programming knowledge of high level languages such as Java or Python, but they usually have a limited knowledge of the C programming language. Since C is at the core of High Performance Computing, the very first part of the subject is devoted to introducing the students to its main concepts and to provide the means for them to work on several programming exercises. Students usually succeed in this initial training, in part due to their high interest and their previous programming experience. Our previous experiences have shown us that devoting some time for setting this basic C knowledge becomes a hard requirement before introducing shared memory or message passing programming.

Once the students have learned the C programming principles, it is necessary to introduce them to the basic concepts of parallel programming. The first point to present is the general idea of parallelism itself and how HPC computing platforms are designed. So, a general introduction to parallel and distributed systems, multi-core processors, memory hierarchy and accelerators, is presented to the students. These objectives present a challenge because it is necessary to provide the students useful, real architecture concepts while avoiding excessively deep details that are complex to relate to programming issues and may become a threat to the assimilation of the relevant knowledge. For this reason, we provide a gentle, summarized introduction with selected further readings for those students particularly interested in the architectural aspects.

The following point in the subject is an introduction to parallel algorithms. The computational aspects of parallel algorithm design must be introduced to the students, showing them different current paradigms and related tools. We provide details on several parallel algorithms for different computational problems. The first problem considered is matrix multiplication, which most of them know very well and have already programmed sequentially.

We start by showing them how the problem is inherently parallel. Several matrix multiplication parallel algorithms are shown and analyzed considering different aspects such as computational complexity, communication requirements, data structure layout and size and memory requirements. These different algorithms are analyzed considering the previously mentioned architectural aspects, showing the implications of computing capabilities, communication network and memory limitations.

Throughout the subject, we identify several important parallel computation patterns [26], which are used in many examples. The map pattern is exemplified by the vector addition algorithm (and the outer loops of matrix multiplication). It is an appropriate pattern to introduce parallelism as it does not involve any dependence or communication among threads. The reduce pattern is studied in the inner loop of matrix multiplication, we use it to introduce the problem of synchronization and the idea of reassociating arithmetic operations to increase parallelism. The stencil pattern is used to simulate the movement of a string, and requires synchronization, sharing, and communications of boundary data.

Two additional parallel computation patterns are studied by means of the exercises proposed to the students. The parallel prefix algorithm (scan pattern) and the convex hull problem (divide and conquer or recursive pattern) are proposed so that students can analyze the problem and find out the sources of potential parallelism in the algorithm. The students compare their proposals considering aspects such as algorithm complexity, memory and communication requirements.

Once the basic concepts of programming and parallelism have been presented to the students, it is feasible to enter the core part of the Parallel Programming subject. In this part, three paradigms are presented: Shared memory, Message passing and Accelerator-oriented massively-parallel programming (GPUs). The rationale for this organization is that developing programs with a shared memory model, such as OpenMP, requires a simple modification of a C sequential program by including just some directives.

So, the students can parallelize their sequential C programs in just one lab session. After OpenMP, MPI is introduced. In this case, it is necessary to think about how to parallelize the algorithm, which processes must be defined, how such processes must communicate, and so on. This implies a greater effort from the students. The last approach introduced is OpenACC and CUDA as programming models for GPUs (accelerators), which requires a more detailed understanding of memory hierarchy and the coordinated use of thousands of threads to reach relevant performance gains.

The programming sessions are complemented with the introduction of performance analysis tools to understand the benefits of parallel programming and to detect and correct performance bottlenecks. Fundamental performance engineering abstractions are introduced, like the speedup concept, Amdahl's and Little's Laws, and the Roofline model.

The steps of the learning evolution shown in Fig. 2 are applied in this subject for each of the aforementioned topics. Consequently, the initial lectures (one in most cases) are used by the professor to introduce the main concepts regarding the topic and, next, students assume incrementally more and more responsibility in the subsequent sessions associated with each topic.
The specific development of these topics is covered in the following subsections.

4.1. Shared memory: OpenMP

As mentioned above, once students are familiarized with C and basic concepts of parallel algorithms, the most natural way to introduce parallel applications development is by using OpenMP [34]. OpenMP is a portable and flexible directive-based API for shared-memory parallel programming which, for some basic code constructions, allows us to express parallelism in an extremely simple way. Given these characteristics, it has become the de-facto standard for multicore shared-memory architectures. In addition, current laptops and desktop computers have multicore processors and, consequently, students can test all the examples given in class and develop new ideas on their own computers.
After a few motivating examples, such as the one shown in Listing 1, the contents of the theoretical OpenMP lecture (2 h) are structured as follows:

- Introduction. Shared memory model, concept of thread, shared and local (private) variables, and need for synchronization.
- Fork-join model. The #pragma omp parallel clause. Introducing parallel regions. Data management clauses (private, shared, firstprivate, lastprivate)
- Data parallelism: parallelizing loops. The #pragma omp for clause.
- Task parallelism: sections. The #pragma omp sections and #pragma omp section clauses.
- OpenMP runtime environment function calls. Getting the number of threads of a parallel region, getting the thread id, and other functions.
- Synchronization. Implicit synchronization, nowait clause. Controlling executing threads, master, single, and barrier clauses. Controlling data dependences, atomic and reduction clauses.
- Performance considerations. Balancing thread load, schedule clause. Eliminating barriers and critical regions.

The instructor plays a central role in this lecture and, consequently, it has been structured following the corresponding strategy, i.e. theoretical lecture, as presented in Section 3. This structure starts by describing the most general and essential concepts (shared memory model, threads and synchronization).

Next, it introduces different parallel constructs ordered according to their conceptual complexity: all threads doing the same work (parallel construct), all threads executing the same code on different portions of data (parallel for construct), and threads executing different tasks (parallel section construct). Then, it introduces several OpenMP synchronization mechanisms, which naturally leads to a discussion of their negative performance implications and strategies to minimize their use.

Listing 1: OpenMP simple example: adding two vectors.
```
#pragma omp p a r a l l e l for
for ( i = 0; i < N; i ++ )
    c [ i ] = a[ i ] + b[ i ];
```

After this lecture, the student should assume the central role and be able to apply the acquired theoretical knowledge to real cases. Consequently, the concepts introduced in this lecture are reinforced in a lab session (2 h with an instructor and 6 h of autonomous development of practical exercises), where students must use OpenMP to parallelize the code for simulating the movement of a string developed in the C labs (see Listing 2). In this way, students continue their work and can experience the advantages of using the 4 cores available in each piece of lab equipment.

Listing 2: String simulation main computation loop.
```
for ( t =1; t <=T ; t ++) {
    for ( x =1; x<X; x ++)
        U3[ x ] = L2 ∗ U2[ x ] + L ∗ (U2[ x+1]+U2[ x − 1]) − U1[ x ] ;
    double ∗ TMP =U3;
    / / r o t a t e usage o f v e c t o r s
    U3=U1; U1=U2; U2=TMP;
}
```

Parallelizing this code with OpenMP is straightforward, as can be seen in Listing 3. Its only complexity is that the clause firstprivate(T,U1,U2,U3) must be used to ensure that each thread does the same vector rotation using its private copies. This parallelization is specially designed to be done in a short time, leaving students plenty of opportunities to test the code and analyze its behavior.

Listing 3: Parallelized string simulation main computation loop.
```
#pragma omp p a r a l l e l f i r s t p r i v a t e ( T , U1 , U2 , U3)
for ( t =1; t <=T ; t ++) {
    #pragma omp for
    for ( x =1; x<X; x ++)
        U3[ x ] = L2 ∗ U2[ x ] + L ∗ (U2[ x+1]+U2[ x − 1]) − U1[ x ] ;
    double ∗ TMP =U3;
    / / r o t a t e usage o f v e c t o r s
    U3=U1; U1=U2; U2=TMP;
}
```

Several of the strategies presented in Section 3 have been applied in the design of this lab session. First, the lab session strategy has been used to train the students in the use of the most common tools used in the lab: compilers (gcc), monitoring tools (likwid [19], perf [39]), remote access and resource management (SGE [42]).

A detailed manual describing these tools with examples has been elaborated with this objective. In this case, students develop this exercise in groups of two. Second, the case study strategy has been used to work on the problem of parallelizing the string movement simulator previously described.

In this case, students are provided with a very short outline of the problem, so they must explore different approaches to the solution on their own. Third, the conceptual maps strategy has been used to make students organize and summarize the concepts learned. Students must write and deliver a report describing their solution to the problem and the tests done on the application they have developed. Finally, the experimental portfolio strategy is used in order to follow the student's evolution through the set of exercises developed in the lab.

4.2. Message passing: MPI

After explaining parallelism at a multi-core level using shared memory, the next step is to introduce cluster parallelism (distributed memory) using message passing. With this objective, the subject includes two lectures (4 h) on Message Passing Interface (MPI) [27] and two lab sessions (4 h with an instructor and 12 h of autonomous development of practical exercises). MPI is by far the most used interface for developing distributed memory parallel programs, mainly because many libraries have been implemented based on the MPI consortium specification (OpenMPI, MPICH, Intel MPI, etc.). MPI includes plenty of features, but this subject focuses on presenting the basic MPI program structure and the functions for point-to-point as well as collective communication.

The contents of the MPI lectures are structured as follows:
• Message passing paradigm. Distributed memory parallel computing, the need for a mechanism for interchanging information. Introducing MPI history.
• MPI program structure. Initializing and finalizing the environment MPI_Init and MPI_Finalize. Communicator's definition (MPI_COMM_WORLD), getting the number of processes in the application (MPI_Comm_size) and the process rank (MPI_Comm_rank). General structure of an MPI call.
• Point-to-point communication. Sending (MPI_Send) and receiving messages (MPI_Recv). Sending modes: standard, synchronous, buffered and ready send.
• Blocking and non-blocking communications. Waiting for an operation completion (MPI_Wait and MPI_Test).
• Collective communication. Barrier, broadcast, scatter, gather and reduce operations.
• Performance considerations. Overlapping communication and computation. Measuring time (MPI_Time). Discussion on the communication overhead. Load balancing.

The instructor plays a central role in these lectures and, consequently, they have been structured following the corresponding strategy, i.e. theoretical lecture, as presented in Section 3. This structure starts by describing the most general and essential concepts (distributed memory model, processes and message passing). Next, it introduces the basic MPI concepts: program structure, communicators, process identifier and MPI function naming convention. Then, it introduces different types of communication ordered according to their conceptual complexity: point-to-point blocking communication, point-to-point non-blocking communication, and collective communication. This naturally leads to a discussion of the impact of each type of communication on the application performance and programming complexity. This discussion on MPI applications performance is also used to present the load balancing problem and some strategies to overcome it.
Students work around these concepts in the lab sessions by developing a simple program for computing π approximation using the dartboard approach [38]. This approach simulates throwing darts at a dartboard on a square backing. As each dart is thrown randomly, the ratio of darts hitting the board to those landing on the square is equal to the ratio between the two areas,which is $\pi/4$.
A parallel implementation of this algorithm consists of a certain number of processes throwing a fixed number of darts and calculating their own approximation of π , then one of the processes (the master) receives all approximations and calculates the average value. In this solution, workers send their results to the master (process with rank 0) using point-to-point communication.
A second approach consists of distributing the total number of throws among all the processes, and each of them will calculate its own number of hits (darts in the circle) and send it to the master process, which will compute the π approximation. In this case, the master sends the number of throws that must be done by each process and receives the number of hits, always using collective communication functions.
As in the case of OpenMP, several of the strategies presented in Section 3 have been applied in the design of these lab sessions. First, the lab session strategy has been used to train the students in the use of MPI tools: mpicc, mpirun and mpe [31]. Also in this case, students develop this exercise in groups of two. Second, the case study strategy has been used to work on the problem of π computation; again, students are provided with a very short outline of the problem, so they must explore different approaches to the solution on their own. Third, the conceptual maps strategy has been used because students must write and deliver a report describing their solutions to the problem and the tests done on the applications they have developed. Finally, the experimental portfolio strategy is used in order to follow the students' evolution through the set of exercises developed in the lab.

4.3. GPUs: CUDA and OpenACC

After introducing OpenMP and MPI programming models, our objective is to teach students the principles of effectively using computational accelerators like GPUs. As we did previously in the case of multi-core systems, we start by presenting the OpenACC toolkit to the students, then providing them with a deeper view of accelerators with CUDA. OpenACC [3] is an open specification for compiler directives for parallel programming. With the use of high level directives, similar to OpenMP, applications can be accelerated without losing portability across processor architectures. CUDA is an extension for massively parallel programming of GPUs (or accelerators). We choose CUDA instead of OpenCL because of the existence of efficient and mature compiling, debugging and profiling tools, and because of the extensive information available. The contents of the lectures are structured as follows:

• Introduction. Hierarchy of threads: warp, CTA (Cooperating Thread Array) and grid. 3-dimensional thread identifiers.
• Model of an accelerator: host and device. Moving data between host and device. Allocating memory on the device and synchronizing the execution.
• Architectural restrictions. Warp size. Maximum CTA and grid dimensions.
• Memory space. Global, local and shared memory.
• Synchronization. Warp-level and CTA-level synchronization.
• Performance considerations. Excess of threads to tolerate the latencies of data dependences. Increasing work per thread to improve instruction-level parallelism.

The lecture uses vector addition as an example to introduce the OpenACC and CUDA syntax. Four implementations are provided and evaluated using: (a) one single thread, (b) one CTA, (c) a grid of CTAs where each thread performs a single addition, and (d) a grid of CTAs with more work per thread.

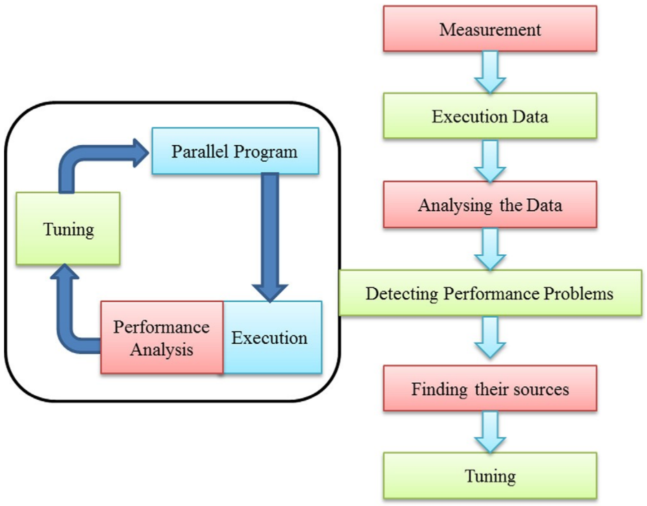

Fig. 3. Performance analysis in the application development cycle.

We show the performance results (deceiving, for the first implementations) to motivate the different solutions and the need for developing good performance engineering skills.

We also present Thrust [1], a high-level parallel algorithm library written in C++, to show the students the benefit of learning object-oriented programming and software engineering concepts.

However, due to the limited background of our students and obvious time limitations, it is out of the scope of our subject to provide further information on Thrust usage. Students must use OpenACC and CUDA in the lab sessions to parallelize the code that simulates the movement of a string. They explore, step by step, the different obstacles they must face to exploit the full potential of GPUs and increase performance ≈ 10x with respect to the multicore CPU code.

The methodological strategies used in this part of the subject are very similar to those on the previous parts. First, lecture sessions provide a general introduction to the concepts defined above.

Then, lab sessions train the students in the use of Nvidia software development tools. Students receive the vector addition case study and must provide incremental solutions that rely on performance-oriented design decisions.

4.4. Performance analysis: tools

It is not just important to be able to develop applications using the different approaches taught throughout the subject. In general, parallel programming's main goal is to improve application performance and, consequently, performance analysis should be introduced to students.

During the subject labs, students use basic tools, such as Intel VTune, nvprof and nvvp [6], perf Linux command [39], jumpshot [37] and likwid [19] to visualize and analyze the behavior of their applications. These tools are enough for the simple applications developed and the small cluster used in this subject.
However, our students will likely participate in the development of real parallel applications during their professional life. Consequently, a lecture (2 h) is used to describe the performance analysis cycle shown in Fig. 3 and introduce the main tools currently available for supporting each of these steps.

The instructor plays a central role in this lecture and, consequently, it has been structured following the theoretical lecture strategy presented in Section 3. In this case, the contents are naturally guided by the performance analysis cycle presented in Fig. 3. Consequently, measurement and monitoring concepts and tools are presented first. For example, Performance API (PAPI) [36] related to parallel programming in the student's field of study. We expect that our students will be able to integrate the knowledge acquired in this subject into their current and future projects.

• Apply specific methodologies, techniques and resources to conduct research and produce innovative results in the area of specialization. Using parallelism could allow our students to obtain better results (better precision, more results, faster results), which could help to achieve new innovative objectives in their specific
projects.
• Continue the learning process, to a large extent autonomously.
In order to achieve this outcome, students have been ''forced'' in this subject to tackle practical problems autonomously, which includes thinking about the problem's solution, but also consulting bibliographical sources looking for the proper functionality or tool to implement it.
• Identify sources of parallelism in a computational problem. The theoretical foundations of the process for designing a parallel solution to a given problem are presented at the beginning of this subject. Then, these foundations are taken into consideration when discussing each subject topic and subsequently applied in the lab sessions.
• Design and develop the parallel solutions to a computational problem taking the characteristics of the available hardware into account. To achieve this outcome, the subject includes techniques and tools to implement parallel applications on multi-core, cluster and accelerator architectures.
• Interpret information from performance-analysis tools and be able to consider application-specific design decisions to improve performance. Students use appropriate tools to analyze the performance of an application and are asked to include the results and impact of their analysis in the lab reports.

Finally, it is worth mentioning that the Computer Architecture and Operating Systems department of Universitat Autònoma of Barcelona has received support from computation industry leaders for the design and development of computation labs. We have been appointed by Intel as an Academic Partner with the use of
the Intel Parallel Studio as one of the programming environments for the practical laboratories and we have also been selected as a GPU Teaching Center by Nvidia Corporation for introducing CUDA, OpenACC and GPU technology into computer architecture studies.

5. Applied modeling and simulation

The main goal of the Applied Modeling and Simulation object is to introduce the students to real applications that use modeling and simulation and that must apply parallel programming techniques to improve their performance. It is highly significant to show the students how High Performance Computing is necessary to make these real applications practical.

The main concepts for this subject are developed in two different parts, each with a different methodology:
1. Case studies in collaboration with industry and research laboratories that use modeling and simulation activities every day.
2. Simulation model development and performance analysis.

In the following subsections, we present detailed descriptions of these two parts of the subject.

5.1. Case studies

The first part, Case studies, is conducted in collaboration with industry and research laboratories that use modeling and simulation activities every day. The activities carried out include invited lectures from researchers that work in these laboratories and use modeling to carry out their work.

The first case considered is the paradigmatic example of meteorological services. Everybody watches the weather forecast on TV every day and can imagine the complexity of the models involved, with huge meshes of points with hundreds of variables estimated for every point, and the computing requirements needed to provide a real prediction. However, in this particular case, it is known that weather prediction models show chaotic behavior. The way to keep this behavior as limited as possible is to execute not just a single simulation, but a complete set of scenarios (called ensemble) and apply statistical methods to conform the final prediction. This meteorological modeling and prediction part is presented by members of the Servei Meteorològic de Catalunya (Meteorological Service of Catalonia).

Obviously, it is outside of the scope of the subject to develop a meteorological model, but, the students can use some small specific models such as wind field models (WindNinja [9]) to analyze its execution time, scalability and speedup. In this context, some students (one or two per year) may enroll in an internship in this meteorological service developing code for some particular model or applying parallel programming techniques to some of the existing models.

In a similar way, a collaboration has been established with the IC3-BSC (Institut Català de Ciències del Clima—Barcelona Supercomputing Center), but, in this case, the models and predictions are related to climatological models involving very large time scales. In this case, the real time aspect is not so critical, since the predictions are considered for decades or even centuries. However, the main point is to run hundreds or thousands of simulations with different parameters that make the total amount of computational requirements extremely high. Also in this case, some students carry out an internship in this center, where they have access to very large computing resources and can do studies on speedup and scalability.

5.2. Simulation model development and its performance analysis

In the second part, the students develop a certain simulation model and analyze its performance. In this case, the teaching strategy is based on three well-defined parts: lecture sessions (including conceptual maps), lab sessions and, finally, experimental portfolio. In the last two parts, students develop a project and carry out lab sessions with teacher supervision. At the beginning, the teacher presents the concepts on a particular modeling technique (agent-based modeling -ABM-) and a conceptual map to show the hierarchy of the concepts to be developed. These types of models (ABM) are used to model real systems from different areas of knowledge that are close to the initial knowledge of students.

ABM can represent complex patterns of behavior through simple rules and provide useful information about the system dynamics of the real world. In addition, it is a kind of simulation that needs high computing power when the number of individuals increases which is suitable for the objectives pursued in the area of HPC. As case study, a model of emergency evacuation using ABM is analyzed [14] and the students must perform some practical exercises to extend the model and analyze its performance in lab sessions. There are different aspects for model analysis: the environment and the information (doors and exit signals), policies and procedures for evacuation, and the social characteristics of individuals that affect the response during the evacuation.

Moreover, the following hypotheses are defined as a starting point for the model:
• In emergency evacuation situations, people are generally nervous or even panicking, so they tend to act irrationally.
• Individuals try to move as quickly as possible (more than normal).
• Individuals try to achieve their objectives and may try to push each other in their attempt to exit through a specific door, causing physical injury to other individuals.

Students receive a partial model that includes the management of the evacuation of an enclosed area that presents a certain building structure (walls, access, etc.) and obstacles, with particular signaling and the corresponding safe zones and exits. The model also includes individuals who should be evacuated to safe areas. This model has been developed to support different parameters such as: individuals with different ages, total number of people in the area, number of exits, number of chained signals and safe areas, speed of each individual, and probability of exchanging information with other individuals. The model mentioned is implemented in NetLogo [45] and Fig. 4 represents its main characteristics.

The first practical work for the students requires using a single-core architecture in the lab to analyze the performance of the model and then incorporate a new, not covered, policy: overcrowding in exit zones [12]. Students must then complete a new performance analysis of the new model.
Considering the variability of each individual in the model, a stability analysis is required. For this, the Chebyshev Theorem (also spelled as Tchebycheff) will be used with a confidence interval of 95% and $\alpha = 0.05$, $m = 6$. The result for this analysis indicates that at least 720 simulations must be done to obtain statistically reliable data. Taking into account the 720 executions on one core processor, the simulation time (average) is 7.34 h for 1000 individuals and 27.44 h for 1500 individuals per scenario. In order to use this tool as a Decision Support System (DSS), the students are instructed in necessary HPC techniques and the embarrassingly parallel computing model is presented as a method to reduce the execution time and the decision-making process time [10,13].

Therefore, students must learn how to execute multiple parametric Netlogo model runs in a multi-core system and how to make a performance analysis to evaluate the efficiency and scalability of the method.
Finally, the instructor offers a set of tasks based on the developed model (experimental portfolio), proposing new challenges and additional specifications so the student can assess their knowledge.
These sessions are performed autonomously by the student in the laboratory, but the work is evaluated and represents a percentage of the grade of the module.

5.3. Learning outcomes

The objective of this subject is that the students achieve the following learning outcomes:
• Describe the different components of a system and the interactions between them. Students must analyze a complex system and identify its main components.

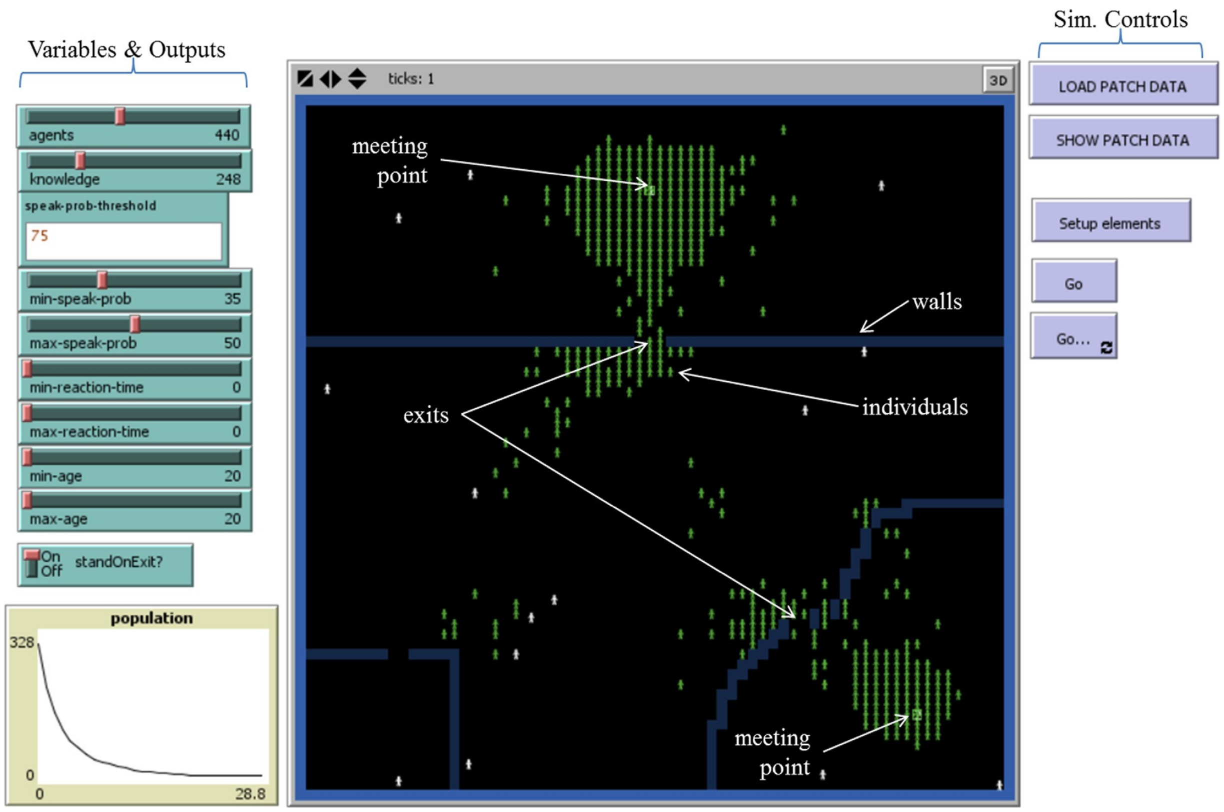

Fig. 4. Agent based modeling for emergency evacuations.

• Identify the parameters that determine how a system works. Students must determine the main parameters of the complex system and analyze their effect on the system behavior.
• Implement appropriate numerical methods to solve models in the particular field under consideration. The mathematical models involved in complex systems usually must be solved by numerical methods. Students must implement different numerical methods efficiently and integrate them into simulation tools.
• Simulate the behavior of complex systems. Students must use the simulation tools to study the behavior of the system.
• Validate the simulation results with the predictions of the models and the behavior of the real system. Students must compare the results obtained from the simulation with the real behavior of the system to validate the correctness of the simulation tools.

6. Academic results

The global result is very positive and, after 5 years of teaching in this interdisciplinary Master's, we have observed that students achieve a much wider range of knowledge, which allows them to tackle problems from disciplines different from their original background. Actually, most of them get jobs in companies where they apply the concepts of computational thinking and performance engineering to improve the applications developed at those companies in areas such as traffic simulation, geophysical simulation, water pollution simulation, spread of disease simulation, and many others, independently of their background.

Currently, the number of qualified candidates exceeds the number of offers and academic results are very high. The success rate (calculated as number students that pass the subject/number of assessed students) and the performance rate (calculated as number of students that pass the subject/number of enrolled students) for both of the subjects presented in this paper are very high, up to 100%. The detailed results of the whole Master's can be seen on the Master's official web page [33].

Quantitative results are relevant to assess the success of the presented subjects, but we consider that the feedback given by students and companies has been very important for building this success. Students answer each semester a survey about the studied subjects. This survey includes subjective questions such as «Tell us what have you liked the most from this course», «Do you have any improvement suggestion?» or «Do you consider the workload of this course has been adequate?». In the latest editions, students have expressed that these subjects are extremely useful and are a great complement to other subjects (specially those related to Big Data) of the Master's programme. Most of them have considered that the workload is adequate, and some have suggested that it is possible to go deeper in certain contents. Overall, they have qualified the 2016 edition of these subjects with 92/100.

Finally, it is important to mention that there are many companies that collaborate in both subjects. They provide certain models that the students can utilize in the lab sessions and deliver invited talks. These conferences are very fruitful for students and enrich their knowledge. Students can see a real application and an actual usage of the theoretical bases taught in the subject lectures.

7. Conclusions

Many fields of science and engineering are evolving through the contribution of complementary fields. This implies that project teams in companies and research centers have significant interdisciplinary components. It is necessary for people from different fields to be able to establish a common ground and understand the requirements and the effects of the problems and solutions for all members of the team. In this sense, it is worth mentioning the performance effects of application design decisions, which, many times, invalidate otherwise valid ideas.

High Performance Computing (HPC), including parallel and distributed programming, becomes a central factor that is applied to many fields from science and engineering. So, it is necessary for students from various fields to receive significant training in HPC. In this way, they will be able to design and develop their own applications and, even more importantly, they will understand the decisions needed to get the most from a given computational platform, such as which is the most suitable programming paradigm, which are the most relevant performance metrics and how to measure them. In this way, they can establish a common language with computer scientists and work together in the development of more powerful and successful applications.

In this interdisciplinary context, we show our experience of teaching parallel programming in interdisciplinary studies at a graduate level. We present a methodological background with the main principles and activities applied to the subject development.
We have used a balanced perspective in which teachers first use direct methodologies to establish the background of the subject; then, students are progressively presented with more interactive, practical activities, and have to understand the problems and propose their own solutions.

The main conclusion is that the experience has been very successful and most students enjoy developing parallel programs, analyzing their behavior and trying to improve their performance.
After this experience, it would be very interesting to introduce similar subjects at the undergraduate level, so that students from different fields are able to apply High Performance Computing techniques to their computational problems from the very beginning.

References

[1] N. Bell, J. Hoberock, Thrust: a productivity-oriented library for cuda, GPU Computing Gems: Jade Edition.
[2] L. Carter, R. Botts, C. Crockett, Computational science programs: The background research, in: 2012 Frontiers in Education Conference Proceedings, 2012, pp. 1–6.
[3] C. Enterprise, C. Inc., NVIDIA, T. P. Group, The openacc application programming interface.
[4] Computational and Mathematical Engineering MS Degree. http://scpd.stanford.edu/programs/masters-degrees [Online; accessed 21.09.16].
[5] Computational Science and Engineering. https://www.seas.harvard.edu/programs/graduate/computational-science-and-engineering/ [Online; accessed 21.09.16].
[6] Cuda Visual Profiler. http://docs.nvidia.com/cuda/profiler-users-guide/index. html#visual-profiler [Online; accessed 18.05.15].
[7] Dyninst API, http://www.dyninst.org/ [Online; accessed 18.05.15].
[8] EPFLs Master in Computational Science & Engineering. http://cse.epfl.ch/ [Online; accessed 21.09.16].
[9] J. Forthofer, K. Shannon, B.W. Butler, Initialization of high resolution surface wind simulations using nws gridded data, in: Proceedings of 3rd Fire Behavior
and Fuels Conference; 25–29 October, 2010.
[10] I.T. Foster, Designing and Building Parallel Programs - Concepts and Tools for Parallel Software Engineering, Addison-Wesley, 1995.
[11] G.C. Fox, Parallel computing and education, Daedalus 121 (1) (1992) 111–118.
[12] A. Gutierrez-Milla, F. Borges, R. Suppi, E. Luque, Individual-oriented model crowd evacuations distributed simulation, in: Proceedings of the International
Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10–12 June, 2014, 2014, pp. 1600–1609. URL http://dx.doi.org/10. 1016/j.procs.2014.05.145.
[13] A. Gutierrez-Milla, F. Borges, R. Suppi, E. Luque, Crowd dynamics modeling and collision avoidance with openmp, in: Proceedings of the 2015 Winter Simulation Conference, Huntington Beach, CA, USA, December 6–9, 2015,
2015, pp. 3128–3129. URL http://dx.doi.org/10.1109/WSC.2015.7408433. [14] D. Helbing, L. Buzna, A. Johansson, T. Werner, Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions, Transp. Sci. 39 (1)
(2005) 1–24. URL http://dx.doi.org/10.1287/trsc.1040.0108. [15] P. Hernandez, Construyendo el constructivismo. Criterios para su fundamentacion y su aplicacion instruccional, Vol. 1, Paidos, 1997.
[16] Interdisciplinary Program in Computational Science, Engineering & Math. https://www.ices.utexas.edu/graduate-studies/ [Online; accessed 21.09.16].
[17] J.P. Lalley, R.H. Miller, The learning pyramid: does it point teachers in the right direction? Education 128 (1) (2007) 64.
[18] G. Lammers, C. Brown, Work in progress - extending parallelism education to the first year with a bottom-up approach, in: 2011 Frontiers in Education
Conference, FIE, 2011, pp. 1–2.
[19] Lightweight performance tools. https://code.google.com/p/likwid/ [Online; accessed 18.05.15].
[20] A. Marowka, Think parallel: Teaching parallel programming today, IEEE Distrib. Syst. Online 9 (8) (2008) 1–8.
[21] A. Martínez, A. Sikora, E. César, J. Sorribes, ELASTIC: A large scale dynamic tuning environment, Sci. Program. 22 (4) (2014) 261–271.
[22] Master of Engineering - Modeling and Simulation. http://catalog.odu.edu/graduate/frankbattencollegeofengineeringandtechnology/ modelingsimulationvisualizationengineering/ [Online; accessed 21.09.16].
[23] Master of Science in Analytics and Modeling. http://www.valpo.edu/grad/compsci/ [Online; accessed 21.09.16].
[24] Master Programme in Computational Science. http://www.uu.se/en/admissions/master/masterprogrammes/ [Online; accessed 21.09.16].
[25] Mathematical Modelling and Scientific Computing MSc. https://www.ucc.ie/en/ckr36/ [Online; accessed 21.09.16].
[26] M. McCool, J. Reinders, A. Robison, Structured Parallel Programming: Patterns for Efficient Computation, first ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012.
[27] Message Passing Interface Forum. http://www.mpi-forum.org/ [Online; accessed 18.05.15].
[28] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C.B. Navarrete, S. Benkner, M. Sandrieser, L. Morin, F. Bodin, AutoTune: A plugin-driven
approach to the automatic tuning of parallel applications, in: Applied Parallel and Scientific Computing - 11th International Conference, PARA
2012, Helsinki, Finland, June 10–13, 2012, Revised Selected Papers, 2012, pp. 328–342.
[29] Modelling and Computational Science MSc. http://catalog.uoit.ca/ [Online; accessed 21.09.16].
[30] A. Morajko, O. Morajko, T. Margalef, E. Luque, MATE: dynamic performance tuning environment, in: Euro-Par 2004 Parallel Processing, 10th International Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, Proceedings, 2004, pp. 98–106.
[31] MPI Parallel Environment (MPE). http://www.mcs.anl.gov/research/projects/perfvis/software/MPE/ [Online; accessed 02.06.16].
[32] H. Neeman, L. Lee, J. Mullen, G. Newman, Analogies for teaching parallel computing to inexperienced programmers, SIGCSE Bull. 38 (4) (2006) 64–67.
[33] Official Master's Degree in Modelling for Science and Engineering. http://www.uab.cat/web/studying/official-master-s-degrees/aster-s-degree-in-figures-1334300576994.html?param1=1307112830469 [Online; accessed 03.06.16].
[34] OpenMP. http://openmp.org/ [Online; accessed 18.05.15]. [35] Paraver. http://www.bsc.es/computer-sciences/performance-tools/paraver [Online; accessed 18.05.15].
[36] Performance API. http://icl.cs.utk.edu/papi/ [Online; accessed 18.05.15].
[37] Performance Visualization. http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/ [Online; accessed 18.05.15].
[38] Parallel Programming in C. http://gribblelab.org/CBootcamp/A2_Parallel_Programming_in_C.html [Online; accessed 20.09.16].
[39] perf: Linux profiling. https://perf.wiki.kernel.org/index.php/Main_Page [On-line; accessed 18.05.15].
[40] G.M. Schneider, D. Schwalbe, T.M. Halverson, Teaching computational science in a liberal arts environment, SIGCSE Bull. 30 (2) (1998) 57–60.

[41] S. Shende, A.D. Malony, The tau parallel performance system, IJHPCA 20 (2) (2006) 287–311.
[42] Sun Grid Engine (SGE) QuickStart. http://star.mit.edu/cluster/docs/0.92rc2/guides/sge.html [Online; accessed 02.06.16].
[43] Tuning Project. Tuning Educational Structures in Europe.
http://www.unideusto.org/tuningeu/images/stories/documents/General_Brochure_final_version.pdf [Online; accessed April 2016].
[44] White Book. Degree in Computer Engineering. ANECA, http://www.aneca.es/media/150388/libroblanco_jun05_informatica.pdf [Online; accessed April 2016].
[45] U. Wilensky, NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.
https://ccl.northwestern.edu/netlogo/index.shtml [Online; accessed 18.05.15].
[46] F. Wolf, Scalasca, in: Encyclopedia of Parallel Computing, 2011, pp. 1775–1785.