A Low-Cost Multicomputer for Teaching Environments

Carles Aliagas, Montse García-Famoso, Roc Meseguer, *Member, IEEE*, Pere Millán, and Carlos Molina

***Abstract*— We propose a teaching resource that uses HardKernel boards to build an MPI server with 256 cores. Although this system has a relatively low performance, the aim is to provide access to hundreds of cores for carrying out scalability analyses, while obtaining a good trade-off between performance, price, and energy consumption. Here, we give details about the implementation of this system at both the hardware and software levels. We also explain how it was used to teach parallel programming in a university degree course, and discuss the teachers' and students' comments about using this new system.**

***Index Terms*— Parallel machines, multicore processing, multiprocessor interconnection, low-power systems, educational activities.**

I. Introduction

TO ACQUIRE parallel programming and computer skills in the Degree in Computer Engineering it is necessary to practice and have access to parallel and distributed computer systems. The most common standards are OpenMP, MPI, and CUDA. To be able to use these, it is essential to have specific hardware for doing the appropriate exercises and for practicing. These systems tend to be expensive and the budget often limits the number of processors available in a classroom. Although university departments have high-performance systems for research, it is difficult to access these systems for teaching, and therefore low-performance systems are generally used.

In the subject *Parallel and Mass Computing* in the fourth year of the Degree in Computer Engineering of the University Rovira i Virgili, the subject of this study, a queue system with 24/7 access (24 h a day, 7 days/week) is used in dedicated servers. Therefore, students always have access to the teaching servers, both during teaching time and outside class time.

Manuscript received May 23, 2020; accepted June 2, 2020. Date of publication July 9, 2020; date of current version August 5, 2020. (Spanish version received November 19, 2019; revised January 5, 2020; accepted February 7, 2020). This work was supported in part by the Government of Spain under Contract TIN2016-77836-C2-1-R, Contract TIN2016-77836-C2-2-R, Contract TIN2016-75344-R, and Contract DPI2016-77415-R and in part by the Government of Catalonia as Consolidated Research Groups under Grant 2017-SGR-688 and Grant 2017-SGR-990. *(Corresponding author: Carles Aliagas.)*

Carles Aliagas, Montse García-Famoso, Pere Millán, and Carlos Molina are with the Departament d'Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili, 43007 Tarragona, Spain (e-mail: carles.aliagas@urv.cat; montse.garcia@urv.cat; pere.millan@urv.cat; carlos.molina@urv.cat).

Roc Meseguer is with the Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain (e-mail: meseguer@ac.upc.edu).

There exists a Spanish version of this article available at http://rita.det.uvigo.es/VAEPRITA/V8N3/A6.pdf

Digital Object Identifier 10.1109/RITA.2020.3008098

In particular, selected practice exercises are performed in shared memory machines (SM) and distributed memory machines (DM). In the first case (SM), programming is performed with the OpenMP standard with a machine with two Xeon E5-2660 processors at 2.2 GHz, each of which have eight cores with multithreading capacity (two threads per core). Therefore, up to 32 processes can be executed simultaneously. In the second case (DM), the parallel programming MPI model is studied with an 8-node system with two Opteron 2210 processors at 1.8 GHz, each of which has two cores with multithreading capacity (two threads per core). Therefore, again, up to 32 processes can be executed simultaneously.

In general, in OpenMP environments with shared memory systems, it is relatively easy to get systems of 16–32 cores for a price that can range from 5,000 € to 10,000 €. Similarly, a CUDA environment is also relatively affordable. A high-end graphics card can be purchased for a price between 1,000 € and 2,000 € and can be incorporated into a server to perform the necessary CUDA exercises. The same server could even be used for OpenMP and CUDA approaches. On the other hand, MPI programming environments, being distributed memory systems, make it possible to study the scalability of a problem better because they provide access to a much larger number of nodes, processors, and cores. However, it is too expensive to acquire systems that exceed four or eight nodes, and therefore, scalability studies remain within the theoretical scope.

A solution for improving scalability is to use computer rooms (or laboratories) and thereby obtain a number of nodes greater than 30 computers. This solution has a serious problem: exclusivity. Although the computers could be reserved for several hours, they would have to be left after that time so that other subjects could use them. With this restriction, students would not be able to do the exercises outside class hours, and therefore the complexity of the exercises would have to be significantly reduced. Moreover, the interconnection network used by the computer rooms would be shared with other systems of the educational center, which would undoubtedly significantly damage the MPI communication of the exercises to be performed. This would make it very complex to obtain invariable execution times according to the network load. Given this problem, it is unreliable to study speedup and scalability on nondedicated and nonexclusive networked computers.

Our goal is to build a system for an MPI programming environment with exclusive dedication, of more than 250 cores,

which is relatively economical (between 5,000€ and 6,000€). In addition, this resource will also follow the current trend of working with systems with a good relationship between performance and low energy consumption. Therefore, apart from learning to use the MPI standard, students must adapt to the resources offered by the system in terms of computing power, RAM, and communication network. They can also perform scalability studies by varying the number of processes from a few units to hundreds of units.

This work is an expanded and revised version of the article "Minimulticomputador de Bajo Coste" [1] published in the conference XXV Jornadas sobre la Enseñanza Universitaria de la Informática (JENUI 2019) that was held in Murcia (Spain) in July 2019.

The main differences between the articles are the following:

- *Title*, *Abstract*, and Section I (*Introduction*) have been improved according to the new material included.
- Section II *Related Work* has been added, and new bibliographic references have been included and analyzed
- Section III (*Proposal:Hardware*) and Section IV (*Proposal:Software*) have been reorganized, thus improving the description of the proposal, both in terms of the hardware and software.
- Section V (*Proposal:Verification*) has been reorganized to focus on verifying the system, adding a better and deeper analysis of the consumption, temperature and performance potential of the proposed solution. In addition, we consider a system with eight nodes instead of the 4-node system initially assumed for distributed memory machines, and show the updated results and analysis.
- Section VI (*Use of the System in the Context of the Subject*) is a totally new section. It shows how the proposed low-power system was used in a parallel programming subject. The model for evaluating the subject is described and the performance of the system is verified with initially sequential algorithms that the students parallelize in their practical exercises. These algorithms were executed and analyzed in all the systems considered in this work (Xeon, Opteron, and Odroid).
- Section VII (*Added Value of the New System*) is also a totally new section. It describes the added value (quantitatively and qualitatively)of the low-cost computing system proposed in this work for both the students and teachers of the subject.
- Section VIII (*Conclusions*) has been also improved according to the new material included.

II. Related Work

One of the first approaches, developed in the late 1990s, for implementing parallel systems in teaching environments was to connect personal computers through a local area network. Chhabra and Singh [2] were among the first to apply this approach, proposing the combined use of personal computers available on the intranet of university campuses. From there, many universities used this approach in their parallel programming subjects. For example, Apon *et al.* [3] describes several experiences of using computer clusters (based on PCs/Workstations) in several universities, depending on the specific objectives of the subjects taught (computer architecture, parallel processing, parallel systems, etc.). Even very low-cost approaches with a few personal computers (barely four) have been proposed, such as the system proposed by Xu and Su [4] for the Shanghai University of Engineering Science. In general, all these systems focus on working on parallel processing in distributed memory environments, but in some cases, the systems can also be applied to shared memory environments.

A decade later, this type of environment is still the one used to teach subjects that introduce students to parallel programming, now using GPU boards (Graphics Processing Units) together with commercial general-purpose processors. Giménez [5] describes his experience in the course "Introduction to Parallel Programming" of the Department of Computing and Systems of the University of Murcia, focusing on parallel programming environments and tools such as OpenMP, MPI, and CUDA. The author uses a cluster of six nodes with a total of 64 cores and with GPU and Xeon Phi cards for each node (ten GPUs and two Xeon Phi). Santamaría *et al.* [6], in the course of Computer Architecture of the Telecommunications Engineering Department of the University of Jaén, use computers with NVIDIA GeForce 8800 GT GPU boards, as well as the CUDA 2.3 toolkit, but their experience is also suitable to other parallelization alternatives (MPI, OpenMP).

An attractive alternative, that contrasts with physical environments, is the cluster systems implemented in the cloud. Ivica *et al.* [7] presented StarHPC, an environment developed by MIT for learning algorithms and parallel programming that provides a virtual machine image configured for parallel programming in OpenMP and OpenMPI technologies. MIT also developed the StarCluster [8] project, which offers a dynamic cluster hosted in the Amazon Elastic Cloud Computing (EC2). This project makes it possible to use distributed and parallel applications and systems. Similarly, Gomez-Folgar *et al.* [9] proposed using cloud computing for learning MPI, in this case, based on CloudStack. The advantage of these types of systems is that they provide great flexibility and high availability. In addition, educational institutions can reduce costs by eliminating the need for dedicated servers or storage and cooling systems as well as minimize the system administrators' tasks.

The use of low-cost systems for improving some features of university classrooms and laboratories that offer some type of engineering education is an increasingly common practice. Due to the lack of funding, teachers tend to sharpen their ingenuity. In this sense, and without being exhaustive, there are a lot of examples that follow this approach. For example, Prieto and Mendoza [10], from the Department of Electronics of the University of Alcalá, developed a robotic platform that facilitates learning without the need for face-to-face laboratories, by using low-cost devices and boards developed in the actual department. Rodriguez *et al.* [11] used Raspberry Pi boards in the teaching classrooms of the National University of La Matanza (Buenos Aires) to implement a system of network access points. Students can answer questionnaires about the subject from their mobile devices without the need for Internet

access. More focused on solutions for Computer Engineering subjects, Catalán Cantero and Blesa Gascón [12] from the Department of Computer Science and Systems Engineering, University of Zaragoza, proposed a programming model for Raspberry Pi. To perform this, they connected a desktop computer and two Raspberry Pi boards through the Internet, to show how these boards work.

Ortega *et al.* [13], from the Department of Computer Science of the University of Almeria, used Raspberry Pi and Arduino boards to demonstrate the potential of this type of low-cost system and, above all, to motivate and awaken the students' interest in the subject. Therefore, they propose experiments such as activating a relay with a phone call, building a piano with fruit, or even setting up a cluster with 4 Raspberry Pi. More recently, Johnston *et al.* [14] discussed the potential of SBC-based clusters (Single Board Computer) as an alternative to traditional clusters in terms of cost and consumption. However, these proposals are not aimed at developing or analyzing applications based on parallel programming.

Several proposals focus on parallel MPI programming and the use of low-cost systems. Pfalzgraf and Driscoll [15], intending to obtain a low-cost cluster (less than 1,000 US$) for training in high-performance computing (HPC, *High Performance Computing*), built a system of 25 nodes (Raspberry Pi Model B) interconnected through an Ethernet switch. The system is managed by free software (open source), which means it has an affordable final cost. To test the system they used three Python + MPI codes with basic linear algebra operations, varying the n size of the problem (vector size, matrices). The results show that it has a very similar behavior toward much more expensive HPC systems.

Similarly, Wina Rachmawan *et al.* [16] used a low-cost cluster consisting of 32 PandaBoard ES, two switches, and two adapted power supplies, with an OpenMPI communication library. The novelty, in this case, is that students also participate in designing and assembling the hardware, and not only in the programming. The HPC training is organized in seven courses, and the surveys carried out by the students show their preferences for the practical training using this cluster. A problematic point of these low-cost systems for HPC training, which is not reflected in most articles, is the configuration of the different hardware and software components. Doucet and Zhang [17] described in detail the assembly and configuration of the hardware, software, and communications of an educational cluster with 20 Raspberry Pi.

Many technical articles, such as Cloutier *et al.* [18], Aroca and Gonçalve [19], and Göddeke *et al.* [20], although they are not directly associated with teaching experiences but rather with parallel performance, have studied the feasibility of assembling a cluster using SoC (System on Chip) systems. These studies compare performance, energy consumption, and cost, but as they are based mainly on 32-bit ARM processors, which have low memory (from 256 KB to 1 GB), the performance they provide is quite low. Although they have a low cost and consume little energy, by establishing a performance/cost and performance/energy relationship, as Cloutier *et al.* [18] did, they do not surpass the relationship offered by traditional systems based on Xeon and Opteron high-performance processors. Aroca and Gonçalves [19] concluded that ARM systems are energy efficient when they run applications based on database services such as MySQL, but they are not energy efficient with applications that require floating-point calculations. Finally, Göddeke *et al.* [20] compared the execution of HPC applications in two clusters, one with 32 nodes with Xeon X5550 processors and one with 96 nodes with ARM Cortex-A9 processors. They conclude that, despite the low consumption of the ARM processors, these require more runtime as they have less computing power, and are therefore not competitive at the energy level compared to Xeon processors. In addition, they concluded that more than twice the floating-point calculation power would be necessary for the system to start being competitive in the performance/energy ratio.

In this work we use processors (ARM Cortex-A53) with more computing power and, going one step further, we aim to build a system of several tens of nodes that would allow students to analyze the scalability of MPI development environments at a reasonable price. We did not consider the assembly of the SoC cluster system until the market offered systems with enough power, RAM, and connectivity to obtain a better performance/cost ratio. Therefore, we have taken advantage of the most recent developments in low-cost boards, resulting in greater benefits than those obtained in related works. We use more powerful Odroid boards, with twice the RAM (2 GB), with eMMC memory (faster than other SD cards) and Gigabit Ethernet connections (10 times faster than Raspberry Fast Ethernet), connected to two Gigabit switches. The four cores of each Odroid (more powerful than those of the Raspberry Pi) allow students to perform scalability tests with up to 256 processes, which is generally not possible in teaching-related studies.

III. Proposal: Hardware Aspects

A. Alternatives

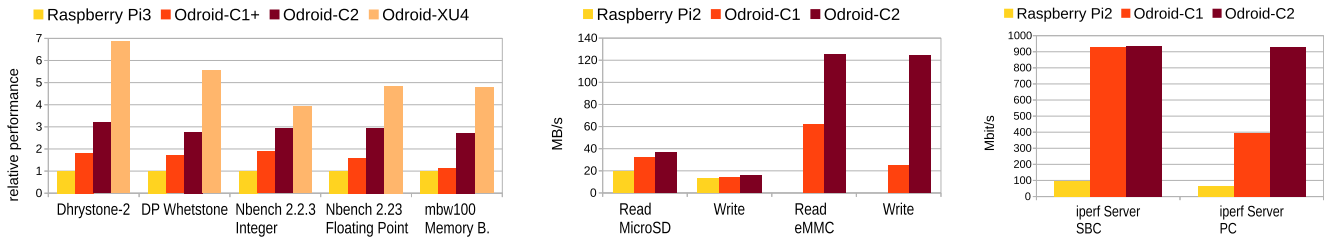
Our proposal focuses on low-power processor systems with a reasonable calculation capacity that can run a complete operating system (OS) from the beginning. Although personal computers can do this, we discarded this possibility because we need a system that can be linked easily with others, which occupies the minimum possible space and that has low energy consumption. Similarly, embedded systems were discarded because they cannot be easily used to run a full Linux OS. Therefore, we decided to analyze SoCs, which have similar features to those based on Raspberry Pi [21].

In the following, we describe the different alternatives considered (additional details can be found in Table I) for selecting a complete low-cost processor system:

- Raspberry Pi-3B: This is the third version of the original design and the most popular in SoC systems. It has a clear design and high performance; however, it does not have Gigabit Ethernet and only provides low RAM capacity. We, therefore, discarded this possibility.
- Banana Pi-M2-Berry: This is a similar alternative and has Gigabit Ethernet, but it provides less computing power.

TABLE I
COMPARISON OF THE SOC CONSIDERED. RELEVANT TECHNICAL FEATURES

Board	Processor	ARM	cores	Frequency	MEM	Disk	Ethernet
Raspberry Pi-3B	Cortex-A53	v8	4	1,2 GHz	1 GB DDR2	MicroSD	Fast
Banana Pi-M2-Berry	Cortex-A7	v7	4	1,0 GHz	1 GB DDR3	MicroSD	Gigabit
Banana Pi-M3	Cortex-A7	v7	8	1,8 GHz	2 GB DDR3	eMMC	Gigabit
Orange Pi-plus-2E	Cortex-A7	v7	4	1,5 GHz	2 GB DDR3	eMMC	Gigabit
Odroid-XU4	Cortex-A15/A7	v8/v7	4/4	2/1,5 GHz	2 GB DDR3	eMMC	Gigabit
Odroid-C1+	Cortex-A5	v7	4	1,5 GHz	1 GB DDR3	eMMC	Gigabit
Odroid-C2	Cortex-A53	v8	4	1,5 GHz	2 GB DDR3	eMMC	Gigabit



(a) CPU performance (b) Storage speed (c) Ethernet speed

Fig. 1. Comparison of different SoC boards (*HardKernel*® manufacturer source).

- Banana Pi-M3: This is a better option than M2-Berry as it doubles the number of cores, increases memory capacity, and has faster eMMC storage; however, it has a higher price and is not powerful enough.
- Orange Pi-plus-2E: This option has a reasonable price and stands out for improving memory, storage, and communication, but it has low computing power.
- Odroid-XU4: This is the most powerful system with eight cores (4 Cortex A15 2,1 GHz + 4 cores A7 1,5 GHz) but unfortunately it has a high price.
- Odroid-C1+: This system competes directly with the Raspberry Pi-3B. It is a good option, but the same manufacturer offers a more powerful top model.
- Odroid-C2 [22]: This system brings together all the advantages of the previous systems at a reasonable price. It improves memory, communication, storage, and computing power.

B. Selected Board: Odroid-C2

We selected Odroid-C2 to build the system. This board satisfies the minimum criteria for assembling the system: ARM processor with enough performance (four cores per board), enough RAM (2 GB) and acceptable bandwidth (Gigabit Ethernet). Figure 1 compares the performance of the chosen board with the other alternatives: a cheaper board (Odroid-C1 +), a more expensive board (Odroid-XU4) and a very popular board in the market (Raspberry Pi-3B). These values were obtained from the website of the manufacturer HardKernel® [22] and show the CPU performance (Figure 1a), the transfer speed of storage access (Figure 1b), and the speed of communication transfer (Figure 1c).

It can be seen that the chosen board's performance is clearly better than the rest, with the exception of the XU4 board, which, although it has better performance, is more than twice the price. The selected board has a price of 46 US$ with a very good performance/price ratio. Its features are:

- CPU: Amlogic ARM® Cortex®-A53 (ARMv8) 1.5 GHz with four cores.
- GPU: Mali-450 (3 *Pixel* processors + 2 *Vertex shader processors*).
- RAM: 2 GByte DDR3.
- Gigabit-Ethernet.
- Monitor connection: HDMI 2.0 4K/60Hz.
- VPU: H.265 4K/60FPS y H.264 4K/30FPS.
- 40 pin GPIOs + 7 pin I2S.
- Connection to eMMC5.0 HS400 cards and MicroSD UHS-1 SDR50 card reader.
- 4 USB 2.0 connectors and 1 USB OTG (with power).
- Infrared receiver (IR).
- Ubuntu 16.04 or Android 6.0 Marshmallow based on *Kernel* 3.14LTS.

It has many interesting features, especially the computing power of ARMv8 (3 times more powerful than Raspberry Pi3), the 2 GB of RAM (2 times higher), the access speed to eMMC storage (120 MB/s), and the network transfer speed of the Gigabit-Ethernet (900 Mbit/s). Perhaps, the only drawback of this board is that it does not have serial wireless connectivity, since it comes without Wi-Fi or Bluetooth; however, it is possible to add optional add-ons in its USB ports that fulfill this function. Nevertheless, our aim is to connect all the boards with cables.

C. Communication Network

The interconnection network is one of the most important factors that define the performance of parallel distributed memory algorithms. In this system, we chose a solution to obtain the maximum performance of the boards' communication cards without increasing the cost. Although communication via USB can offer good performance, in our

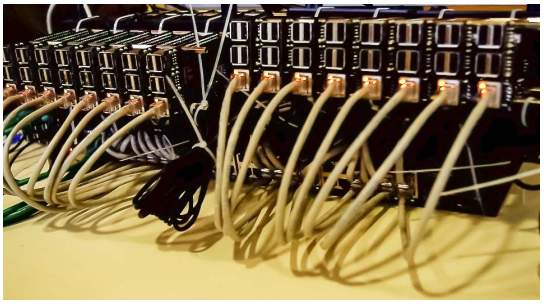


Fig. 2. Operating Odroid-C2 boards.

case it is limited by the USB 2.0 standard, which has a maximum speed of 480 Mbps (the effective speed of a USB 2.0 Gigabit-Ethernet device does not exceed 300 Mbps). Given this limitation, it was decided to communicate the boards through the built-in Gigabit-Ethernet, thus obtaining effective speeds close to 900 Mbps.

The boards were interconnected by dedicated switches (two Zyxel model GS1900-48 switches, which is an appropriate and economical solution).

Their main features are the following:

- Switching capacity (Gbps): 100
- Forwarding rate (Mpps): 74
- Packet buffer (byte): 1.5 M
- MAC address table: 8 K
- Jumbo frame (byte): 9 K

D. Board Assembly

The proposed multicomputer consists of: 64 Odroid-C2 with 8 GB of eMMC storage each, eight power supplies with ten USB outputs each of the sources, two Zyxel switches, and category 5E Ethernet cables. As each Odroid board has 4 cores, it is possible to obtain a system with 256 cores in total.

Assembling the multicomputer consisted of physically joining an Odroid board to another board next to it and connecting each of the Odroid boards to a switch port (Figure 2). As the switches have 48 ports, 32 cables were used to connect the Odroids and an additional cable to connect the switches to each other. An additional Odroid board was used as the manager node to serve as a frontend of the multicomputer. This Odroid makes it possible to launch parallel applications in the computing nodes and provides network access to them. This node has the same components as the other nodes as well as a keyboard, mouse, HDMI connection to the screen, and Gibagit Ethernet via USB to gain access to the multicomputer from the outside.

IV. Proposal: Software Aspects

A. Operating System and Services of the Manager Node

For the proposed multicomputer to run parallel applications, the Linux OS provided by the manufacturer (Ubuntu 16.4) can be written directly on eMMC or MicroSD memory cards. This makes it possible to use a package manager and therefore be able to keep the software updated and install the required packages in a very simple way [23].

The installed Linux system comes with the standard Ubuntu software for desktop machines. This includes a service boot manager and a window manager. We also added the following services: SSH server (*openSSH*), name server and DHCP server (`dnsmasq`), port replication, *tunneling*, and *firewall* (`iptables`), as well as the network file server (NFS).

The OpenMPI package was also installed in order to run parallel programs. We use the OpenMPI command `mpicc` to compile and the command `mpirun` to execute. This last command has an option that makes it possible to indicate the addresses of the nodes that will be part of the parallel execution. Thus, a text file is created with the 64 internal DNS names of the nodes.

B. Operating System and Services of the Computation Nodes

The Linux Ubuntu 16.04 OS provided by the manufacturer was also installed in the computing nodes; however, in this case, the majority of packages that were not needed for the parallel execution of applications were eliminated, thus leaving a minimum installation that does not consume a lot of the system's resources. We kept the remote access service (SSH), remote disk access client (NFS), and the MPI execution package (OpenMPI). Any package that occupies considerable disk space was uninstalled, including the Gnome desktop.

In order to manage the nodes, users with privileges (*root* and *sudoers*) have been allowed to execute commands on the nodes from the frontend without having to enter the privileged user's password each time. This was done by generating an RSA key (`ssh-keygen`) and subsequently distributing the public key to the nodes so that it can be accessed without requesting the password (`ssh-copy-id`).

An entry in the file system assembly (`fstab`) was added to the nodes so that, through the NFS service, they can access the user directory of the frontend node (`/home`) and share it among all of the nodes. Therefore, in parallel execution, the cost of transmission of the compiled binary MPI codes is not required. This transfer is carried out by NFS, and this protocol is used to optimize the distribution of the code.

V. Proposal: Verification

A. Temperature

To verify the correctness of the system and that there is appropriate passive cooling, stress tests were carried out with the package `stress` [24]. We found that the temperature of the CPU with the inactive nodes was around 41 °C and that, after a period of CPU stress, temperatures stabilized around 60 °C (some nodes around 56 °C and others around 62 °C).

Given the physical distribution of the boards, it is observed that if they are stacked horizontally, the heat of the lower boards affects the upper boards, so they reach temperatures above 75 °C. Although this value is not a problem in the system (the manufacturer's limit is above 80 °C), we observed that by turning the Odroid stack 90°, and keeping it in a vertical distribution (as in Figure 2), the temperatures were drastically reduced, and they stabilized around 60 °C. Undoubtedly, this is because the heat generated is dissipated by convection and does not affect neighboring boards so much. For instance,

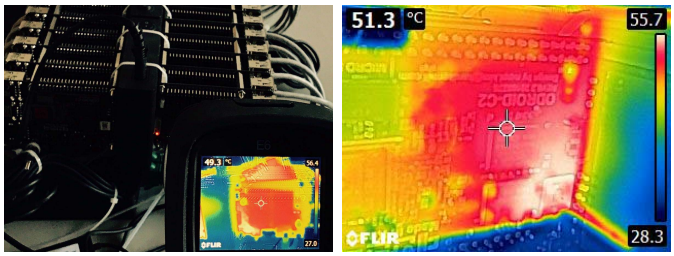


Fig. 3. Odroid-C2 temperature with stress tests.

TABLE II
ENERGY CONSUMPTION OF THE ANALYZED SERVERS

	one node		switch	total system	
Server	***idle***	***full***		***idle***	***full***
MC-Xeon	-	-	-	110 W	418 W
MD-Opteron	144 W	180 W	20 W	1,172 W	1,460 W
MD-Odroid	2.5 W	4 W	40 W	201 W	303 W

Figure 3 shows an outer board with a maximum temperature in stress kept at 55.7 °C.

B. Energy Consumption

In order to verify the consumption of the new Odroid Distributed Memory (MD-Odroid) system, we compared it with the rest of the existing systems (discussed in Section I), that is, the Opteron Distributed Memory (MD-Opteron) system and the Xeon Shared Memory system (MC-Xeon). Therefore, energy consumption is measured with an ammeter that has a power supply from a UPS (Uninterruptible Power Supply). Two operation modes were considered: *idle* and *full*. The first case would be the consumption of the three servers doing nothing, while the second case would be the consumption of the three servers at full capacity when they are executing the *stress* program in each of the three systems.

Table II details the power consumption of each of the three servers, considering the two operating modes described earlier. The individual results of one of the nodes of each system, and the overall results considering all the nodes of the system are shown. In addition, and to be more fair with the comparison, we included the consumption introduced by the different switches needed by each server (two switches for the MD-Odroid system, one switch for the MD-Opteron system, and no switch in the case of the MC-Xeon system).

As can be seen, the MD-Odroid server has the lowest energy consumption when it is at full capacity. It has just 72% of the consumption of the MC-Xeon server and 20% of the total server consumption of the MD-Opteron server. In fact, since only those nodes that are being used will have an increase in consumption (the rest will remain "idle"), the MD-Odroid system option offers an excellent consumption/performance ratio. This analysis reaffirms the viability of this new system in terms of consumption.

C. MPI Code Selection

We used three different types of code to verify the system's performance, three different types of codes are selected: (1) product of sparse matrices, (2) search for an optimal path of the traveling salesman, and (3) computation of clusters based on the k-means algorithm.

The first problem (MD×MD) calculates two matrix products: (1) dense matrix per dispersed matrix, and (2) the product of two dispersed matrices. The exercise does not have many parallelism problems as it is enough to distribute fragments of the matrices between the processes to finally collect the results and compact them in the resulting matrix. The size of the matrices is 5.000×5.000 items.

The second problem is the "traveling salesman problem" (TSP-greedy) in which the salesman must visit a series of points linked by roads, so that he visits them all only once with the minimum possible travel. In this case, a fairly good solution is obtained, but not the optimal solution. It is based on a "greedy" selection of all possible beginnings. The parallel solution involves distributing the possible beginnings, and finally choosing the minimum path that one of the processes has computed. The number of places to visit for this example is 3,000.

Finally, the k-means algorithm tries to create groups of randomly generated elements. It searches iteratively for k-centroids of the elements to be processed, until each of them stabilizes and is assigned to a centroid. The centroids are computed from the average of the elements that belong to them. In this case, there is a recurrence in each iteration, which requires synchronizing the processes to correctly update the centroids. In this exercise, 200 centroids are searched from 400,000 elements.

D. Sequential Performance and Cost

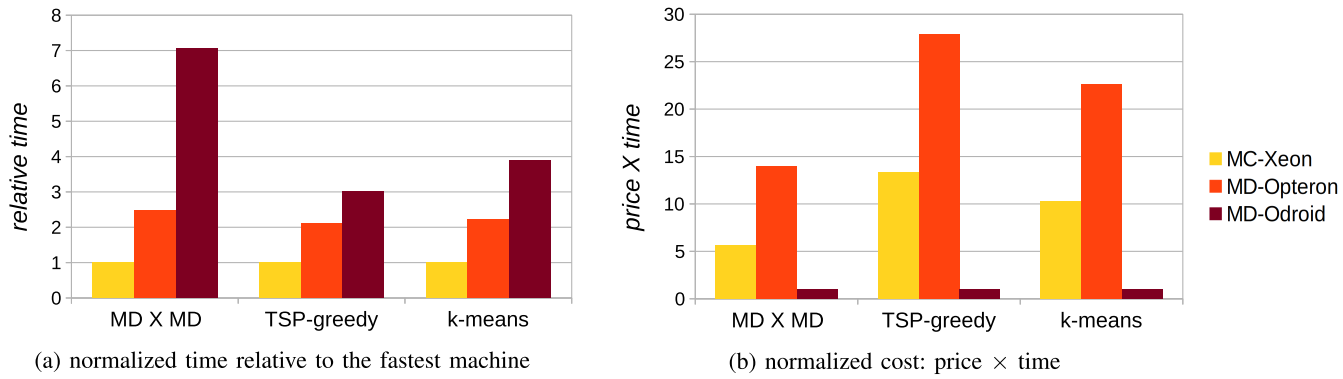
To compare the sequential performance of the algorithms and systems discussed in the previous sections, Figure 4a shows the normalized execution time for creating a single process in each system. The machine with Xeon processors is the fastest: up to 2.5 times faster than the machine with Opteron processors, and up to 7 times faster than an ARM processor on an Odroid-C2 board.

However, considering the price of a single node of each system, the cost of executing these exercises is significantly different. An Odroid-C2 is about 10 times better than a system with a Xeon processor, and about 20 times better than a system with an Opteron processor. Figure 4b shows these costs by calculating them as the execution time multiplied by the price[1] unit of a node, which is equivalent to the cost calculated as price divided by performance (as it will be seen in the figures in the next section).

E. Parallel Performance and Cost

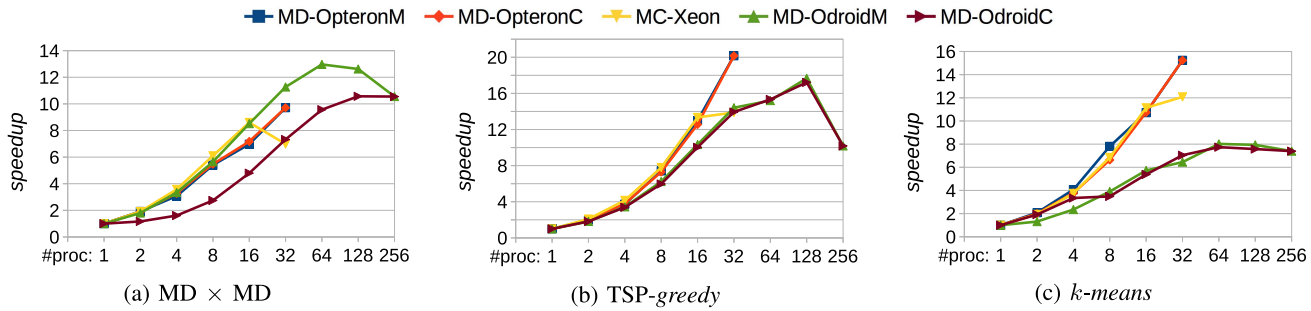
The above codes were parallelized with the MPI standard and were executed using a Slurm queue system (to guarantee exclusivity) on the different machines available for the subject. We only show those that help us assess the differences with MPI: one is a shared memory server with two Intel

[1]A price of 2,000€ is assumed for Xeon and Opteron processor systems. In contrast, the price of an Odroid-C2 board is about 50€.



(a) normalized time relative to the fastest machine (b) normalized cost: price × time

Fig. 4. Cost and performance of sequential execution of verification algorithms.



(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Fig. 5. Evolution of speedup and scalability of verification algorithms in the considered systems.

Xeon E5-2660 processors at 2.2 GHz and 32 GB of RAM (MC-Xeon); one is eight servers that make a cluster of distributed memory, with two Opteron 2210 processors at 1.8 GHz and 4 GB of RAM each (MD-Opteron); and finally, the system object of this work, which is made up of 64 Odroid-C2 boards with an ARM Cortex A-53 processor at 1.5 GHz and 2 GB of RAM each (MD-Odroid).

Figure 5 shows the evolution of speedup as more processes are created in parallel execution. We present two executions with different process mappings for MD-Opteron and MD-Odroid. MD-OpteronM and MD-OdroidM are equivalent to a mapping considering problems that are "Memory-Bound," where there are many memory accesses by the algorithm and the aim is that a process would have all the memory of a node for the process. Thus, for MD-OpteronM an execution of eight processes will assign one process to each node, thus occupying the eight nodes. In MD-OdroidM, each board executes a single process, from the execution of two processes up to 64 processes (two and four processes per board are already assigned for 128 and 256). MD-OpteronC and MD-OdroidC are equivalent to a mapping designed for "CPU-Bound" problems, where there are not too many memory accesses and the bottleneck is due to the use of the CPU. In this case, an attempt is made to assign the maximum number of processes to each node, so that the processes of a node communicate through shared memory without the need to use the interconnection network. For example, for four processes, only one node will be used and, as the number of processes increases, more nodes are used.

In MD×MD (Figure 5a), the machines have a similar evolution but do not achieve a high speedup. The decrease in efficiency is mainly because it is a memory-bound algorithm, and it is necessary to communicate the results of the multiplication to the collecting node. This highlights the loss of performance in 128 and 256 of the MD-OdroidM system, and the low performance obtained in the MD-OdroidC version, due to the competition for the memory that is produced on each board because of the four processes assigned.

With the TSP-greedy algorithm (Figure 5b) we see a similar evolution but with more efficient speedup. This algorithm uses the CPU intensively and has a greater degree of parallelism. Communication is only needed at the end of the execution to decide which process has found the best path and then transmit it to the collecting node. The two types of mapping obtain equivalent results.

Finally, we observed that the k-means algorithm (Figure 5c) behaves in a similar way in MC-Xeon and in MD-Opteron (M and C), but it has a significant loss of performance in the MD-Odroid systems (M and C). In this case, synchronization is needed in each iteration to gather and distribute the new centroids. Here we can see how, when there are multiprocessors in MD-OpteronC and MC-Xeon in which some processes share memory, the communication between them does not use the network. However, the MD-Odroid (M and C) uses the interconnection network more to communicate the centroids, resulting in a significant loss of performance. This highlights the initial benefit (up to 4 processes) of MD-OdroidC over MD-OdroidM since the communication,

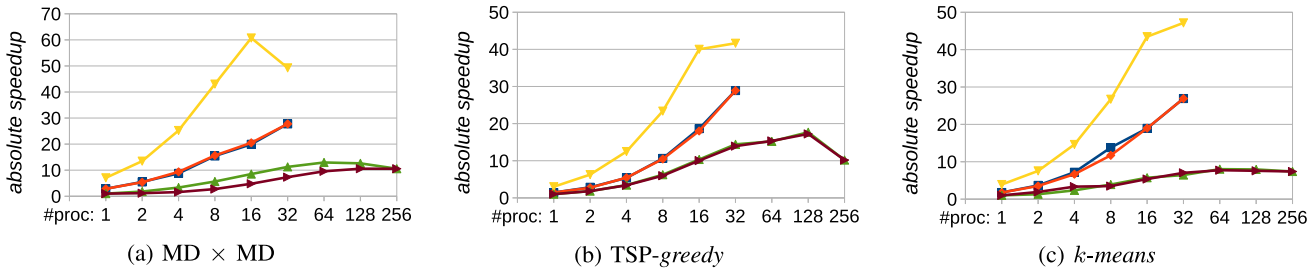


Fig. 6. Evolution of the absolute speedup of the verification algorithms in the considered systems.

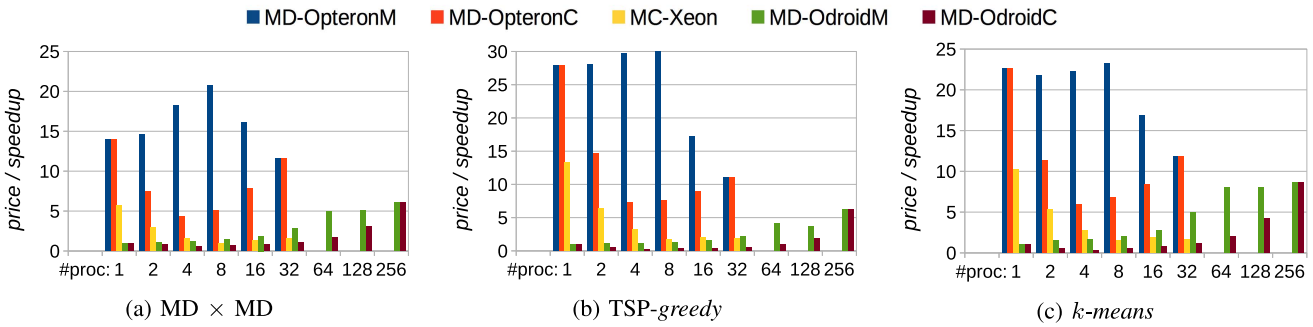


Fig. 7. Absolute price/speedup evolution of the algorithms when using more nodes in the considered systems.

in this case, is also internal to the node, without using the interconnection network.

In any case, it can be observed that, when working with more nodes, the problems and delays produced by the necessary communication between the processes of a parallel algorithm are evident. This behavior is beneficial from an educational point of view because it gives students a more realistic experience.

Figure 6 shows the absolute speedup of the three machines (taking the slowest as baseline). It is clear that the shared memory machine, MC-Xeon, has the best performance, given the computing power of each core of the system, followed by the distributed memory system, MD-Opteron, and finally by the MD-Odroid system. However, when we add the price to the cost function (Figure 7), calculating the cost in money with respect to the absolute performance offered by each system, the results change dramatically.

The MC-Xeon system has the best performance due to its computation power and internal communication, and thus has a very good price/performance ratio, especially for executions of 8 to 16 processes. MD-Opteron systems, in any configuration and mapping, have the highest performance cost of the three. There is no combination which indicates, with respect to the others, that it is an optimal option in terms of the price/performance ratio. For memory-bound mappings (MD-OpteronM) the situation is even worse, because the price of the system to execute eight processes includes the cost of the eight nodes.

Finally, MD-Odroid systems (M and C) are the ones that offer the best price/performance ratio compared to the others mainly due to its low cost. The fact of increasing the cost, adding boards every time you want to run more processes, makes it (in any configuration and mapping) better than the rest by a large difference. These systems have a price/performance ratio equivalent to the other two only for mappings of 4 processes per node and executions of 64, 128, and 256 processes, but in the rest, they are the best option. From the different mapping types, MD-OdroidC obtains a lower performance for memory-bound algorithms (Figure 7a) than the MD-OdroidM option. However, despite this, MD-OdroidC continues to obtain a better price/performance ratio than the others because it takes full advantage of the cores of each board.

VI. System Use in the Context of the Subject

The proposed system has been introduced, validated, and analyzed in the context of the subject Parallel and Massive Computing of the fourth year of the Degree in Computer Engineering at Rovira i Virgili University. This subject studies multiprocessor architectures of shared and distributed memory (MMC, MMD), going deepener into specific techniques for efficient and parallel programming. For evaluating the subject, there are two clearly differentiated parts that have the same weight in the final grade, and which must also be passed separately: the theoretical part and the practical part. The theoretical part consists of two exams that assess the theoretical knowledge that the student has acquired. The practical part consists of a practical exercise involving the parallelization of a series of sequential codes and their subsequent execution to evaluate their speedup. Therefore, the practical part aims to consolidate the theoretical concepts of parallel programming. In each academic year, two different problems are selected that pose a degree of increasing difficulty. The first problem is more affordable than the second. Each of these two exercises needs to be solved with OpenMP (shared memory) and MPI (distributed memory) standards. The resolution of these

two exercises with the two possible memory models involves four different deliveries over the four-month period in which the subject is taught. These deliveries have the same weight in the final grade of the practical part (25% each); however, students must obtain a minimum speedup (established by the teacher of the subject) to be able to hand in their solutions for assessment. If they do not obtain this minimum speedup the students do not pass the practical part. It is also important to note that practical exercises are resolved in groups of two people.

The evaluation of each of the four practical exercises is divided into two phases. In the first phase, an overall ranking is obtained by comparing the work of each group with the rest of the groups. In the second phase, all group members are interviewed personally, in which the proposed solution and the knowledge acquired from resolving the exercise are discussed. Each of these two phases has a weight of 50% of the mark of that practical delivery.

If we focus on resolving the exercises using a distributed memory model, it is important to comment that the exercise is executed on the Opteron and Odroid distributed memory machines by varying the mapping of the processes and the number of processes created. Two approaches are considered regarding the type of possible mapping: (a) mapping oriented to optimizing the amount of memory, and (b) mapping oriented to optimizing the use of the CPU. In (a) you try to occupy the maximum number of possible nodes with a single process, before assigning several processes to the same node (MD-OpteronM and MD-OdroidM options). In (b) you try to map the neighboring processes within the same node, and thus, communication between them will be more efficient (MD-OpteronC and MD-OdroidC options). Finally, for each of the two mapping options, different executions are performed by varying the number of possible processes (starting from a minimum of two processes and doubling the number to a maximum of 32 for the Opteron machine and up to a maximum of 256 for the Odroid machine). It should be noted that these maximum values are the number of cores that each of these machines have.

To evaluate an exercise, in the first phase, a ranking of the different solutions delivered by the students is generated. Therefore, from the two possible mapping options (memory-bound and CPU-bound), the most favorable option (with better speedup) is selected, both for the Opteron and the Odroid machine. After selecting the best mapping on both machines, all the speedups obtained by varying the number of nodes (from 2 to 32 on the Opteron machine and from 2 to 256 on the Odroid machine) are added together. The value of the sums of each of the student groups is normalized to 1 to obtain the ranking. It is important to note that the sum is used instead of the harmonic mean in order to give more weight to the executions with the greatest number of processes, since these will be the ones that will obtain a numerical value of higher speedup, and will have more influence in the order of the ranking of each group of students. This rewards the behavior of the exercise in which executions have a greater number of processes. Based on the ranking, a rating of 10 is assigned to the first group and the rest of the groups receive a rating proportional to the distance they have from the first. Again, it is important to note that if a minimum value is not achieved in this sum normalized to 1, that practical delivery is considered non-evaluable. Finally, in a second phase, a face-to-face interview is carried out with each of the groups to validate their knowledge of the solution presented and the knowledge they have acquired with their resolution. It should be noted here that the grade given to each of the two members of the group may be different, depending on how each student responds to the questions asked in the interview. However, there is a very high correlation between the grades in phase 1 and phase 2. Groups that get a better speedup tend to get a good grade in the face-to-face interview.

Figure 8 shows the results obtained by students in the 2018–2019 academic year in an exercise that uses search-with-pruning to compute all the solutions of a partially started *Sudoku*. In this case, the parallelization is carried out by following a distributed memory model, and therefore using the MPI standard on the Opteron and Odroid machines. Specifically, the results obtained in the Opteron machine are shown with a mapping by optimizing memory (Figure 8b) and with a mapping by optimizing the use of CPU (Figure 8c). The results obtained in the Odroid machine are shown with a mapping by optimizing memory (Figure 8d) and with a mapping by optimizing the use of CPU (Figure 8e). The ranking of all the students' results, elaborated as detailed above, is also shown (Figure 8f). In addition, the results obtained on the Xeon shared memory machine are shown (Figure 8a).

Although the results of the Xeon machine are irrelevant for the evaluation of the distributed memory exercise, they are included to show that the Xeon shared memory machine has a behavior similar to the Opteron distributed memory machine (Figure 8b and Figure 8c). It can be seen that, in both cases, from 32 processes (number of machine cores), it makes no sense to create more processes, since both machines become saturated. However, a greater number of processes can be executed in the Odroid machine, obtaining a good performance.

The comparison of the practices delivered by the students shows that group G11 obtained the best performance in the distributed machines (MD-Opteron and MD-Odroid), especially for a high number of processes. It should also be noted that groups G3 and G5 have abnormal executions and do not improve performance no matter how they increase the number of processes. Finally, it is worth mentioning the results of group G4, which has normal performance improvements until the creation of 32 processes (from that value the results stagnate). The ranking obtained by the different student groups (Figure 8f) shows that group G11 obtains the maximum performance, that group G4 only just exceeds the set minimum, and groups G3 and G5 do not reach the minimum and need to improve in the next practical exercise. Therefore, group G11 would obtain the highest grade in this part, group G4 the minimum grade, and the rest a score proportional to the distance from the best, in the following order: G7, G6, G9, G10, G1, G2, and G8.

Finally, analyzing the evolution of the students' grades and considering their opinions about using the system (similar to that of Emiroglu and Sahin [25]) will give us the necessary

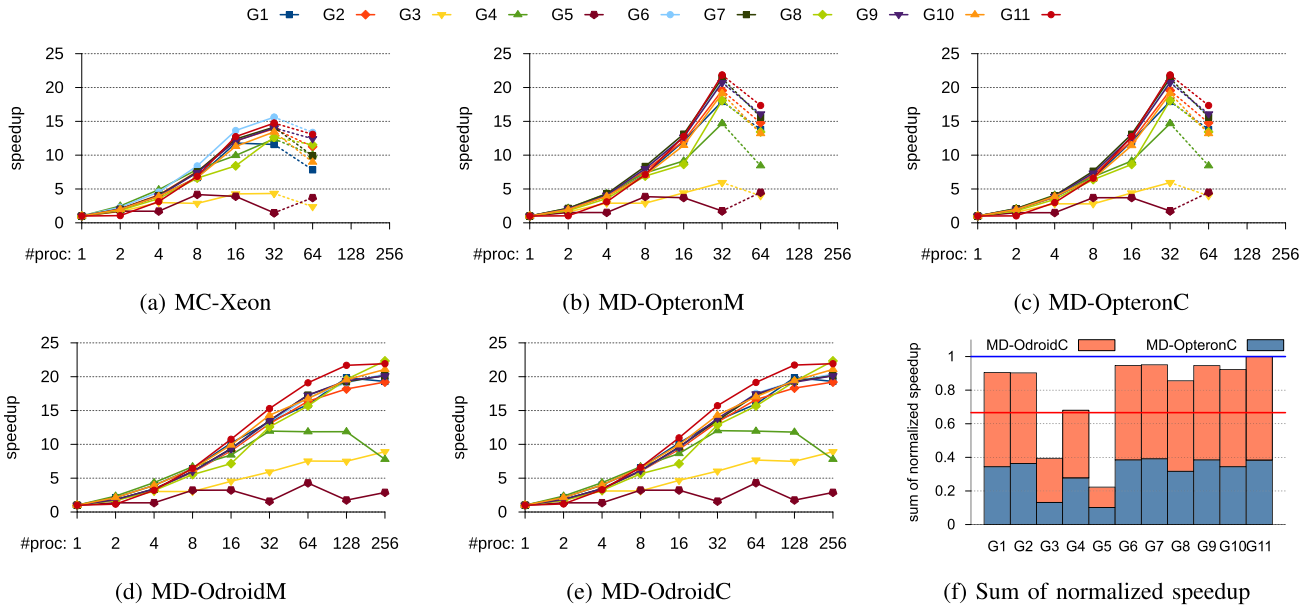


Fig. 8. Comparison of the evolution of speedup of student practices.

feedback to adjust the environment and improve the proposed academic objectives.

VII. Added Value of the New System

The students of the subject use for the first time with a programming environment based on batch queues on dedicated servers available 24/7. This is a new experience with very fast learning, which allows them to work exclusively and thus be able to repeat parallel executions in which the variation in time depends only on their code. The system makes it possible to effectively apply the trial and error methodology studied in the pedagogical theory of behaviorism developed by F. B. Skinner (1904-1990), among others, in the middle of the 20th century. By being able to repeat the different executions, students reinforce their knowledge as they verify that the code improves the result and reduces execution time. Therefore, the teaching experience is positive, and the students evidence this through their comments, because they have systems that usually do not have access.

Students usually have previous experience, academic or personal, with SoC systems (Raspberry Pi and similar). Therefore, when teachers introduce the new server based on Odroid boards, they generally see that students show increased interest. This increased interest did not occur with the previous server, since students were aware of its high price and that it requires demanding installations in terms of refrigeration and energy consumption.

To acquire parallel programming skills, the challenge is to program systems with more than 100 processes (up to 256 processes in this study). While it is easy to access a shared memory machine with 8 or 16 cores, it is not so easy to access machines with hundreds of cores. This extra difficulty forces students to schedule their exercises with sufficiently scalable solutions. This teaching experience would not have been possible if there had not been enough nodes/cores.

Another challenge that students face is the programming of nodes with limited resources. The main cut in resources is the computing capacity and RAM of each node. In this case, the RAM is only 2 GB, which, although it is very small by current standards, is proportional to the size of the exercises to be solved in the subject. This means that in a real parallel programming situation, in which problems must deal with immense amounts of data, the manipulation of the data to be processed must be effectively managed. In these cases, it is necessary to find a compromise that allows solving the problems raised to be solved by adjusting them to the available RAM. In any simulation system, as a general rule, more data implies more precision, but it is not always possible to increase the data due to RAM and computing power limitations. With this new server, students need to be aware that the RAM is just 2 GB and that, if they assign a process to a node, it can have these 2 GB to solve the part of the exercise that corresponds to it (subtracting the RAM required for the operating system). If in that same node as many processes are assigned as there are cores (for example 4), then each process can only have a quarter of the RAM, that is, just 512 KB. This implies a new programming effort to adjust the data volume that each process must deal with.

In summary, we found that the new server offers and extends new paradigms that give students a more realistic parallel programming experience based on increasing cores and restricting resources, such as RAM.

VIII. Conclusion

This work details the construction of a low-cost multicomputer system, both its hardware and software, to be used in practical parallel programming exercises. It is based on Odroid-C2 boards, which have a cost equivalent to that of a small server with shared memory. Therefore, more processors can be included in the system. Furthermore, trying to match

the number of processors in systems with shared memory servers interconnected by a network would cost much more. In any case, the performance obtained in relation to price and consumption is excellent.

We also discuss the use of this proposal in a parallel programming subject of the Degree in Computer Engineering and how it has been included in the assessment model of the practical exercises. The results obtained are very satisfactory, demonstrating this new low-cost system's viability and the added value that it gives to the parallel programming subject. It was observed that this proposal brings students closer to a system with a large number of processors and more intercom options, it allows more realistic scalability studies, and is a good compromise between benefits, price, and consumption.

Finally, we firmly believe that this system is perfectly applicable to other educational fields, where multiple nodes are required to perform practical exercises. Specifically, it could be used in subjects on Distributed Systems (cloud computing …), of System's Administration (`kubernetes` …), and Network Administration (switches …). In addition, we believe that it would be interesting to apply this system in optional university subjects, and even in subjects of the Higher Level Education Cycle (CFGS), as the investment needed to set up the system from scratch is quite acceptable, without taking a large chunk out of the teaching budget.

REFERENCES

[1] C. Aliagas, P. Millán, C. Molina, and R. Meseguer, "MiniMultiComputador de Bajo Coste," in *Actas de las XXV JENUI*, vol. 4. Murcia, Spain: Universidad de Murcia, 2019, pp. 31–38.

[2] A. Chhabra and G. Singh, "A cluster based parallel computing framework for performance evaluation of parallel applications," *Int. J. Comput. Theory Eng.*, vol. 2, no. 2, p. 226, 2010.

[3] A. Apon, R. Buyya, H. Jin, and J. Mache, "Cluster computing in the classroom: Topics, guidelines, and experiences," in *Proc. 1st IEEE/ACM Int. Symp. Cluster Comput. Grid*, May 2001, pp. 476–483.

[4] M. Xu and Q. Su, "The realization of small cluster parallel computing environment for college education," in *Proc. 9th Int. Conf. Comput. Sci. Edu.*, Aug. 2014, pp. 861–863.

[5] D. Giménez, "Un curso práctico de programación paralela basado en problemas de concurso español de programación paralela," in *Actas de las XXII JENUI*. Almería, Spain: Universidad de Almería, 2016, pp. 19–26.

[6] J. Santamaría, M. Espinilla, A. Rivera, and S. Romero, "Potenciando el aprendizaje proactivo con ilias&Webquest: Aprendiendo a paralelizar algoritmos con gpus," in *Actas de las XVI JENUI*. Santiago de Compostela, Spain: Univesidade de Santiago de Compostela, 2010, pp. 503–506.

[7] C. Ivica, J. T. Riley, and C. Shubert, "StarHPC—Teaching parallel programming within elastic compute cloud," in *Proc. ITI 31st Int. Conf. Inf. Technol. Interface*, Jun. 2009, pp. 353–356.

[8] Massachusetts Institute of Technology. *StarCluster Home Page*. Accessed: Jan. 13, 2020. [Online]. Available: http://star.mit.edu/cluster/index.html

[9] F. Gomez-Folgar, R. Valin, A. Garcia-Loureiro, T. F. Pena, and I. Zablah, "Cloud computing for teaching and learning MPI with improved network communications," in *Proc. Workshop Cloud Edu. Environ. (WCLOUD)*, vol. 945, 2012, pp. 22–27.

[10] G. A. Prieto and J. P. Mendoza, "Plataforma robótica didáctica de bajo coste basada en la arquitectura software player/stage y en el hardware de la Fonera," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 7, no. 4, pp. 239–245, Nov. 2012.

[11] R. A. Rodriguez, P. Cammarano, D. A. Giulianelli, P. M. Vera, A. Trigueros, and L. J. Albornoz, "Using raspberry pi to create a solution for accessing educative questionnaires from mobile devices," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 13, no. 4, pp. 144–151, Nov. 2018.

[12] C. Catalán Cantero and A. Blesa Gascón, "Enseñanza de sistemas empotrados: De Arduino a Raspberry Pi," in *Actas de las XXII JENUI*. Almería, Spain: Universidad de Almería, 2016, pp. 351–354.

[13] G. Ortega *et al.*, "Procesadores de bajo coste y su aplicación en la docencia de Ingeniería de Computadores," in *Actas de las XXII JENUI*. Almería, Spain: Universidad de Almería, 2016, pp. 343–349.

[14] S. J. Johnston *et al.*, "Commodity single board computer clusters and their applications," *Future Gener. Comput. Syst.*, vol. 89, pp. 201–212, Dec. 2018.

[15] A. M. Pfalzgraf and J. A. Driscoll, "A low-cost computer cluster for high-performance computing education," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, Jun. 2014, pp. 362–366.

[16] I. E. Wina Rachmawan *et al.*, "An embedded system for applying high performance computing in educational learning activity," *EMITTER Int. J. Eng. Technol.*, vol. 4, no. 1, pp. 46–64, Aug. 2016.

[17] K. Doucet and J. Zhang, "The creation of a low-cost raspberry pi cluster for teaching," in *Proc. Western Can. Conf. Comput. Edu. WCCCE*, 2019, pp. 1–5.

[18] M. F. Cloutier, C. Paradis, and V. M. Weaver, "Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance," in *Proc. Hardware-Softw. Co-Design High Perform. Comput.*, Nov. 2014, pp. 1–8.

[19] R. V. Aroca and L. M. G. Gonçalves, "Towards green data centers: A comparison of x86 and ARM architectures power efficiency," *J. Parallel Distrib. Comput.*, vol. 72, no. 12, pp. 1770–1780, Dec. 2012.

[20] D. Göddeke *et al.*, "Energy efficiency vs. Performance of the numerical solution of PDEs: An application study on a low-power ARM-based cluster," *J. Comput. Phys.*, vol. 237, pp. 132–150, Mar. 2013.

[21] Raspberry Pi Foundation. *Raspberry Home Page, Information and Documentation*. Accessed: Feb. 1, 2019. [Online]. Available: https://www.raspberrypi.org

[22] Hardkernel Co., Ltd. *HardKernel Home Page, Odroid-c2 Information*. Accessed: Feb. 1, 2019. [Online]. Available: https://www.hardkernel.com/shop/odroid-c2

[23] Hardkernel Co. Ltd. *Odroid Wiki, Ubuntu*. Accessed: Feb. 1, 2019. [Online]. Available: https://wiki.odroid.com/odroid-c2/os_images/ubuntu/ubuntu

[24] Canonical Ltd. Ubuntu. *Stress Ubuntu Linux Package*. Accessed: Jan. 13, 2020. [Online]. Available: https://packages.ubuntu.com/search?keywords=stress

[25] B. G. Emiroglu and S. Sahin, "Analysis of students' performances during lab sessions of computer networks course," *J. Educ. Technol. Soc.*, vol. 16, no. 3, pp. 329–346, 2013.

Carles Aliagas received the degree in computer science from the Polytechnical University of Catalonia (UPC) in 1991. He is currently pursuing the Ph.D. degree with Rovira i Virgili University (URV). He is also a Lecturer with the Computer Engineering and Mathematics Department, URV, where he teaches and carries out research. His research interests focus on processor microarchitecture, memory hierarchy, and prediction in mobile networks.

Montse García-Famoso received the degree in computer science from the University of Deusto in 1994. She is currently a Lecturer with the Computer Engineering and Mathematics Department, Rovira i Virgili University (URV), where she teaches and carries out research. Her research interests include methodologies and tools for university teaching in computer engineering.

Roc Meseguer (Member, IEEE) received the graduate degree in Telecommunications Engineering and the Ph.D. degree in computer engineering from the Polytechnical University of Catalonia (UPC) in 1999 and 2012, respectively. He is currently a full-time Associate Professor with the Computer Architecture Department, UPC, where he teaches and carries out research. His research interests include distributed systems, particularly resource allocation for large-scale systems, decentralized systems applied to environmental intelligence, and community networks based on bottom–up initiatives.

Pere Millán received the degree in computer science from the Polytechnical University of Catalonia (UPC) in 1992 and the Ph.D. degree in computer engineering from Rovira i Virgili University (URV) in 2018. He is currently a Senior Lecturer with the Computer Engineering and Mathematics Department, URV, where he teaches and carries out research. His research interests focus on predicting quality and improving performance in mobile networks.

Carlos Molina received the degree in computer science and the Ph.D. degree in computer engineering from the Polytechnical University of Catalonia (UPC) in 1996 and 2005, respectively. He is currently a Senior Lecturer with the Computer Engineering and Mathematics Department, URV, where he teaches and carries out research. His research interests focus on computer microarchitecture, quality prediction, and performance improvement in mobile networks, as well as serverless computing.