

**Teaching Parallel Programming in Containers: Virtualization
of a Heterogeneous Local Infrastructure**

Naylor Garcia Bachiega

Tese de Doutorado do Programa de Pós-Graduação em Ciências de
Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Naylor Garcia Bachiega

Teaching Parallel Programming in Containers: Virtualization of a Heterogeneous Local Infrastructure

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Paulo Sérgio Lopes de Souza

USP – São Carlos
January 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

B123t Bachiega, Naylor Garcia
 Teaching Parallel Programming in Containers:
Virtualization of a Heterogeneous Local
Infrastructure / Naylor Garcia Bachiega; orientador
Paulo Sérgio Lopes de Souza. -- São Carlos, 2021.
 214 p.

 Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2021.

 1. Computing Education. 2. Parallel Programming
Teaching. 3. High-Performance Computing. 4.
Containers. 5. Virtualization. I. Souza, Paulo
Sérgio Lopes de, orient. II. Título.

Naylor Garcia Bachiega

**Ensino de Programação Paralela em Contêineres:
Virtualização de uma Infraestrutura Local Heterogênea**

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Paulo Sérgio Lopes de Souza

**USP – São Carlos
Janeiro de 2022**

First of all, I dedicate this doctoral thesis to Adonai for my existence and intelligence, however limited.

My wife, Carla Fernanda de Souza Bachiega, has always been by my side and provided me with a beautiful family.

To my kids, Benyamin de Souza Bachiega and Sarah Vitória de Souza Bachiega, who gave me joy and hope throughout this journey.

My parents, Vitória Garcia Bachiega and Laércio Donizetti Bachiega, worked day and night to supply my studies.

And to my advisor, who was the most understanding and patient person on the planet.

I will never forget this, and I will never be able to pay for what they did for me.

ACKNOWLEDGEMENTS

I thank my sisters Amanda Bachiega Fernandes and Najara Garcia Bachiega for their care, affection, and understanding during these years of study.

I thank my in-laws, Carlos Alberto de Souza and Marilza Pereira Cardozo de Souza, for providing me with a home to complete my work, and Diego Alberto de Souza and Vitor Alberto de Souza for their fellowship and friendship.

I thank my friends Eder Sotto, Otávio Fernandes, Thiago Coutinho, and Denis Contini for their friendship over the years.

I thank Aparecida Dias Rodrigues for taking care of our family at this crucial moment.

The main thanks go to Júlio Estela, Sarita Bruschi, Simone Souza, and the other ICMC lecturers who helped, in some way, in carrying out this work. I also thank the fellowship and the affection of LaSDPC co-workers.

Special thanks go to the Federal Institute of Education, Science, and Technology of São Paulo, Paraná, and Santa Catarina to support my work.

I also want to thank all ICMC employees, without whom this research could not be carried out, such as librarians, office staff, security, cleaning, and administration staff.

“Blessed is the man who finds wisdom, the man who acquires understanding, for she is more profitable than silver, and her gain is better than fine gold. She is more precious than rubies; nothing you desire compares with her.”
(Mishlê; 3:13-15)

RESUMO

BACHIEGA, N. G. **Ensino de Programação Paralela em Contêineres: Virtualização de uma Infraestrutura Local Heterogênea**. 2022. 214 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Fornecer ensino em programação paralela é um desafio emergente, necessitando de abordagens de ensino para fomentar o processo de aprendizagem e de complexa infraestrutura para proporcionar um ambiente adequado para as aulas práticas de laboratório. Não priorizar requisitos de programação paralela no aprendizado dos futuros profissionais em computação pode levar a uma significativa lacuna de formação, a qual impactará negativamente no uso eficiente das atuais plataformas computacionais. Para popularizar o ensino desse tipo de computação, é imprescindível a adoção de abordagens eficazes de aprendizagem e meios para facilitar a configuração de software e da infraestrutura necessária para a prática do ensino em laboratório. Muitas instituições públicas e privadas não possuem um cluster ou uma infraestrutura adequada para execução dos programas paralelos. Além disso, há um custo operacional para criar e manter um ambiente próprio para essas aulas em laboratório. A falta de docentes que atuam em pesquisas relacionadas à computação de alto desempenho e as dificuldades inerentes ao gerenciamento do ambiente de execução são outros dois fatores que criam barreiras ao ensino de programação paralela. Dessa forma, o objetivo desta tese é avaliar se a virtualização de arquiteturas paralelas heterogêneas contribui para o ensino de programação paralela por alunos de computação em instituições de ensino que não dispõem de tais arquiteturas paralelas, nem pessoal qualificado para a gestão desses ambientes. Esta pesquisa teve como ponto de partida um estudo de trabalhos existentes na literatura, para determinar como o ensino prático de programação paralela é realizado atualmente. Após esse levantamento, foi constatado que nenhuma ferramenta atendia às necessidades de virtualização idealizadas. Definidos os requisitos da virtualização, uma ferramenta usando contêineres foi desenvolvida. Posteriormente, foram conduzidos experimentos com profissionais da área e alunos para avaliar eficácia dessa ferramenta no ensino prático de programação paralela. Como resultado, foi criada a Iguana, uma ferramenta de código aberto para o ensino de programação paralela, pensando em alunos de baixa renda que não têm acesso a arquiteturas paralelas. A ferramenta permite que os alunos criem e executem seus códigos paralelos por meio de uma interface web em tempo real, sem a necessidade de acessar terminais por linha de comando ou aguardar seu processamento em lote. Ademais, a Iguana pode funcionar sem Internet em uma simples máquina virtual, exigindo apenas conhecimentos básicos de informática, permitindo seu uso a qualquer aluno do primeiro ano de graduação.

Palavras-chave: Educação em Computação, Ensino de Programação Paralela, Computação de Alto Desempenho, Contêineres, Virtualização.

ABSTRACT

BACHIEGA, N. G. **Teaching Parallel Programming in Containers: Virtualization of a Heterogeneous Local Infrastructure.** 2022. 214 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Providing parallel programming education is an emerging challenge, requires teaching approaches to further the learning process and a complex infrastructure to provide a suitable environment for the laboratory practical classes. Do not prioritize parallel programming requirements in future computing professionals' learning can lead to a significant training gap, negatively impacting the efficient use of current computing platforms. To popularize "parallel thinking," it is essential to adopt practical learning approaches and means to facilitate software configuration and the infrastructure necessary for laboratory classes. Unfortunately, many public and private institutions do not have a cluster or an infrastructure to run parallel programs. Also, there is an operational cost to create and maintain a required environment for these laboratory classes. The lack of lecturers who work in research related to high-performance computing and the difficulties inherent in managing the execution environment are two other factors that create barriers to teaching parallel programming. Thus, this thesis aims to evaluate whether the virtualization of heterogeneous parallel architectures contributes to the teaching of parallel programming by computing students in educational institutions, which do not have such parallel architectures or qualified personnel to manage these environments. This research started with a study of existing works in the literature to determine how the practical teaching of parallel programming is carried out today. After this survey, we found that no tool met the idealized virtualization needs. Next, we defined the virtualization requirements, and a tool was developed using containers. Subsequently, experiments were carried out with professionals in the field and students to evaluate the effectiveness of this tool in the practical teaching of parallel programming. As a result, Iguana was created, an open-source tool for teaching parallel programming, thinking about low-income students who do not have access to parallel architectures. The tool allows students to develop and run their parallel code through a real-time web interface without the need to access command-line terminals or wait for batch processing. Furthermore, Iguana can operate without the Internet in a simple virtual machine, requiring only essential computer resources, allowing its use by any first-year undergraduate student.

Keywords: Computing Education, Parallel Programming Teaching, High-Performance Computing, Containers, Virtualization.

LIST OF FIGURES

Figure 1 – Structure of the Computing Curricula Series	40
Figure 2 – Brazilian institutions according to geographic location	45
Figure 3 – Prerequisites of the topics related to PDC for Brazilian institutions	48
Figure 4 – Total loads of the topics related to PDC for Brazilian institutions	48
Figure 5 – Semester of the provision of the topics related to PDC for Brazilian institutions	49
Figure 6 – Elective and required topics about PDC for Brazilian institutions	49
Figure 7 – Bibliographic information of the topics related to PDC for Brazilian institutions	50
Figure 8 – Total loads of the topics related to PDC for the world institutions	54
Figure 9 – Year of the provision of the topics related to PDC for the world institutions	56
Figure 10 – Elective and required of the topics related to PDC for the world institutions	56
Figure 11 – Bibliographic information of the topics related to PDC for the world institutions	57
Figure 12 – Parallel computing course	59
Figure 13 – Hybrid schematic distribution with MPI, OpenMP, and CUDA	70
Figure 14 – Cluster HPC Beowulf	71
Figure 15 – HPL benchmark for Docker containers	75
Figure 16 – Memory usage in Docker	75
Figure 17 – Comparison between virtualization modes	76
Figure 18 – Docker infrastructure	79
Figure 19 – Docker Swarm architecture	80
Figure 20 – Results for synthetic imbalances	82
Figure 21 – Volume sharing with NFS	84
Figure 22 – Volume sharing and MPI	85
Figure 23 – Volume sharing with Docker	86
Figure 24 – Benchmark for the image filter	87
Figure 25 – Iguana infrastructure	93
Figure 26 – Languages used in the tool development	95
Figure 27 – Locahost architecture	97
Figure 28 – Local network architecture	98
Figure 29 – The Iguana’s main screen	98
Figure 30 – Compile and run parameters	100
Figure 31 – Node selection	100
Figure 32 – Orderly presentation of the result	101
Figure 33 – Result of the lecturer and student code	102

Figure 34 – Exercises solved	102
Figure 35 – Creating virtual nodes	102
Figure 36 – SUS scoring scale	107
Figure 37 – Age of participants in the experiment with professionals and lecturers	108
Figure 38 – Academic degree of participants	108
Figure 39 – Experience in the computing area	109
Figure 40 – Experience in parallel programming	109
Figure 41 – Answers for the ten SUS questions about usability - 1st round	110
Figure 42 – Answers for the five user interface satisfaction questions - 1st round	111
Figure 43 – Answers about running parallel programs in Iguana with focus in teaching - 1st round	112
Figure 44 – Answers for the ten SUS questions about usability - 2nd round	113
Figure 45 – Answers for the five user interface satisfaction questions - 2nd round	113
Figure 46 – Answers about teaching parallel programming through the tool - 2nd round	114
Figure 47 – Students' scores for the first exercise	115
Figure 48 – Students' scores for the second exercise	115
Figure 49 – Comparing the variability of scores in exercises with and without Iguana support	116
Figure 50 – Comparing the distribution of scores in exercises with and without Iguana support	116
Figure 51 – Answers related to usability to teach parallel programming through the Iguana	117
Figure 52 – Answers for Iguana execution effectiveness	118
Figure 53 – Answers for learning effectiveness through Iguana	119
Figure 54 – Hello aCe World	128
Figure 55 – Response time viewer	129
Figure 56 – MPI job submission through the portal	130
Figure 57 – Supercomputing resources available for the students	130
Figure 58 – StarHPC solution architecture	131
Figure 59 – Pilot API	132
Figure 60 – The architecture of the ZawodyWeb system	133
Figure 61 – Queuing system workflow	134
Figure 62 – Screenshot of Open edX navigation bar	135
Figure 63 – Job launch in a three-node cluster	135
Figure 64 – High-level architecture of Everest	136
Figure 65 – Submit form of generic service for running MPI programs	137
Figure 66 – Results of completed job for MPI service	138
Figure 67 – The Let's HPC platform	139
Figure 68 – A screenshot of how the data filtering process is implemented on the Let's HPC platform	139

Figure 69 – A unifying classroom computing environment	140
Figure 70 – Running a simple parallel program on Jupyter Notebook	141
Figure D.1 – Output of hello world in CUDA	181
Figure D.2 – Output of descriptive statistics metrics in CUDA	181
Figure D.3 – Output of thresholding in CUDA	182
Figure D.4 – Output of greatest common divisor in CUDA	182
Figure D.5 – Output of matrix multiplication in CUDA	182
Figure D.6 – Output of multiplication of vectors by a scalar in CUDA	182

LIST OF CHARTS

Chart 1 – Teaching PDC in 2013 curriculum guidelines for computer courses, according to ACM/IEEE	42
Chart 2 – Teaching PDC in 2013 curriculum guidelines for computer courses, according to ACM/IEEE	43
Chart 3 – List of PDC topics covered by BCS with ACM/IEEE	44
Chart 4 – List of Brazilian institutions selected according to ranking scores	47
Chart 5 – Adherence of topics on PDC with the ACM/IEEE and BCS reference curricula for Brazilian institutions.	51
Chart 6 – List of world institutions selected according to ranking scores	53
Chart 7 – Topics and prerequisites related to PDC for the world institutions	55
Chart 8 – Adherence of PDC topics to ACM/IEEE reference curricula for world institutions	58
Chart 9 – Papers related to teaching strategies	64
Chart 10 – Papers related to laboratory tools	66
Chart 11 – Different stress tests	87
Chart 12 – Summary of system use cases	90
Chart 13 – SUS questions for usability	106
Chart 14 – Questions for user interface satisfaction	110
Chart 15 – Questions about running parallel programs in Iguana with focus in teaching .	111
Chart 16 – Questions about Iguana execution effectiveness	118
Chart 17 – Questions about learning effectiveness through Iguana	119
Chart 18 – Input and output of hello world in CUDA	121
Chart 19 – Input and output of descriptive statistics metrics in CUDA	122
Chart 20 – Input and output of thresholding in CUDA	123
Chart 21 – Input and output of greatest common divisor in CUDA	123
Chart 22 – Input and output of matrix multiplication in CUDA	124
Chart 23 – Input and output of multiplication of vectors by a scalar in CUDA	124
Chart 24 – Tools for teaching parallel programming	143
Chart A.1 – Paired t-test - significance for SSD	170
Chart A.2 – Paired t-test - significance for HDD	171
Chart B.1 – Professionals suggestions for improvements in the tool after the first experiment	173
Chart B.2 – Students’ suggestions after performing experiments with the tool	176
Chart C.1 – Feedback from professionals after the features are applied	177
Chart C.2 – Feedback from students about the Iguana tool	178

LIST OF SOURCE CODES

Source code 1 – Example of a simple OpenMP loop	67
Source code 2 – Example of an MPI code	68
Source code 3 – Example of a CUDA code	69
Source code 4 – Sample data returned with JSON structure	94
Source code 5 – Example of using special tags	99
Source code 6 – Script Install Command	103
Source code 7 – Hello World in CUDA	121
Source code F.1 – Iguana installation script	211

LIST OF TABLES

Table 1 – Computing Knowledge	41
Table 2 – Different stress tests	83
Table 3 – Non-functional requirements	91
Table 4 – Selected papers	128

LIST OF ABBREVIATIONS AND ACRONYMS

ACM	Association for Computing Machinery
BCS	Brazilian Computer Society
CC	Communication and Coordination
CD	Compact Disc
CLI	Command-line Interface
CPU	Central Processing Unit
CS/PP	Concurrent Systems/Parallel Processing
CWUR	Center for World University Rankings
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DP	Distributed Programming
DRBD	Distributed Replicated Block Device
DS	Distributed Systems
DV	Docker Volumes
EC2	Amazon Elastic Computing Cloud
ECTS	European Credit Transfer System
EFTSL	Full-time Equivalent Student Load
FM	Formal Methods
FMS	Formal Models and Semantics
FS	File System
FUR	Folha University Ranking
GPGPU	General Purpose Computing on Graphics Processing Units
GPU	Graphics Processing Unit
HDD	Hard Disk Drive
HPC	High-Performance Computing
HTML	HyperText Markup Language
I/O	Input/Output
IDEs	Integrated Development Environments
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol

LaSDPC	Distributed Systems and Concurrent Programming Laboratory
LVM	Logical Volume Manager
LXC	Linux Containers
MAC	Mandatory Access Control
MPI	Message Passing Interface
MPMD	Multiple Program Multiple Data
NAT	Network Address Translation
NFS	Network File System
NIS	Network Information Service
NTP	Network Time Protocol
OS	Operating System
PA	Parallel Architecture
PAAP	Parallel Algorithms, Analysis, and Programming
PCA	Parallel Computer Architectures
PCEs	Parallel Compute Environments
PD	Parallel Decomposition
PDC	Parallel and Distributed Computing
PDT	Parallel and Distributed Templates
PE	Performance Evaluation
PF	Parallelism Fundamentals
PHP	Hypertext Preprocessor
PP	Parallel Performance
PS	Proprietary Software
QoS	Quality of Service
QUIS	User Interface Satisfaction Questionnaire
RAID	Redundant Array of Inexpensive Disks
RFC	Request for Comments
RPC	Remote Procedure Call
SLA	Service Level Agreement
SPMD	Single Program Multiple Data
SPOC	Small Private Online Course
SSD	Solid State Drive
SSH	Secure Shell
SSL	Secure Sockets Layer
SUS	System Usability Scale
TLS	Transport Layer Security
UMA	Uniform Memory Access

VM	Virtual Machine
VMM	Virtual Machine Manager
WS	Web Services
WUR	World University Rankings

CONTENTS

1	INTRODUCTION	33
1.1	Problem Contextualization	34
1.2	Motivation	35
1.3	Research Question	36
1.4	The Main Proposed Innovation	36
1.5	Thesis Structure	37
2	PARALLEL PROGRAMMING TEACHING	39
2.1	Computer Science Curricula	40
2.2	The PDC Teaching in Brazil	45
2.3	The PDC Teaching Around the World	52
2.4	Mapping of Theoretical and Practical Teaching Approaches	57
2.5	Content of Parallel Programming Teaching	67
2.5.1	<i>Shared Memory (OpenMP)</i>	67
2.5.2	<i>Message-Passing (MPI)</i>	68
2.5.3	<i>Accelerator-oriented Massively-parallel Programming (GPUs)</i>	69
2.5.4	<i>Infrastructure for teaching parallel programming</i>	70
2.6	Final Considerations	71
3	VIRTUALIZATION	73
3.1	Container-based and Hypervisor-based Performance	74
3.2	Container-based	75
3.3	Technologies for Containerization	77
3.3.1	<i>Docker</i>	78
3.3.2	<i>Docker Swarm</i>	79
3.3.3	<i>Constraint Isolation in Container</i>	81
3.4	Volume Sharing in Containers	83
3.4.1	<i>Network File System</i>	84
3.4.2	<i>Docker Volumes</i>	85
3.4.3	<i>DV and NFS Performance Evaluation</i>	86
3.5	Final Considerations	88
4	IGUANA CLUSTER SYSTEM	89
4.1	Requirement Analysis	89

4.2	System Design	91
4.2.1	Architecture	92
4.2.2	Platforms	92
4.2.3	Programming	94
4.2.4	Communications	95
4.2.5	Security	96
4.3	Implementation	97
4.4	Testing and Deployment	103
4.5	Final Considerations	104
5	EXPERIMENTAL EVALUATION	105
5.1	System Usability Scale	106
5.2	Evaluating Usability with Lecturers and Computing Professionals	107
5.2.1	Distribution of professionals	107
5.2.2	Results of the experiment - 1st round	109
5.2.3	Results of the experiment - 2nd round	112
5.3	Teaching Parallel Programming with Iguana	114
5.3.1	Iguana and non-Iguana grades	114
5.3.2	Results of the experiment	117
5.4	Experiments with CUDA	120
5.4.1	Hello World	120
5.4.2	Metrics of descriptive statistics	121
5.4.3	Thresholding	122
5.4.4	Greatest common divisor	123
5.4.5	Matrix multiplication	123
5.4.6	Multiplication of vectors by a scalar	124
5.5	Final Considerations	124
6	RELATED WORK	127
6.1	Systematic Mapping	127
6.1.1	aCe C	128
6.1.2	STEADY	129
6.1.3	A Grid Portal	129
6.1.4	StarHPC	131
6.1.5	Pilot Library	132
6.1.6	ZawodyWeb System	133
6.1.7	SAUCE	133
6.1.8	SPOC	134
6.1.9	OnRamp	135
6.1.10	Everest	136

6.1.11	<i>Let's HPC</i>	138
6.1.12	<i>Palmetto/JupyterHub</i>	140
6.2	Final Considerations	141
7	CONCLUSION AND FUTURE WORK	145
7.1	Contributions	146
7.2	Future Work	147
7.3	Papers	147
7.3.1	<i>Journals</i>	147
7.3.2	<i>Conferences</i>	148
BIBLIOGRAPHY		149
GLOSSARY		165
APPENDIX A	DV AND NFS PERFORMANCE EVALUATION	169
APPENDIX B	IGUANA DESIGN SUGGESTIONS COLLECTED FROM THE EXPERIMENTS	173
APPENDIX C	IGUANA FEEDBACK COLLECTED FROM EXPERIMENTS	177
APPENDIX D	CUDA EXPERIMENTS	181
APPENDIX E	IGUANA CLUSTER SYSTEM TOOL MANUAL	183
APPENDIX F	INSTALLATION SCRIPT	211

INTRODUCTION

High-Performance Computing (HPC), unlike other types of computing, addresses particular challenges and conditions. As a result, the teaching addresses large amounts of information, computational problems, and understanding of supported platforms, where students need skills and abilities to operate these environments. ([SHAMSI; DURRANI; KAFL, 2015](#)).

[Zarza *et al.* \(2012\)](#) highlight the importance of HPC as a valuable tool for society. HPC provides the development of a vast number of applications and services. They also address that the study of parallel architectures is one of the key issues in academic computer science students because of computers' general use in many disciplines.

Parallel Programming, a part of the HPC, became active and intrinsic to the currently available technologies, mainly considering the distinct programming models coming from these different architectures.

Nowadays, heterogeneous computing is a reality, with a broader range of low-cost systems using more than one Central Processing Unit (CPU) core or another type of processor (hardware accelerators, Graphics Processing Unit (GPU), cryptography co-processors, programmable network processors, A/V encoders/decoders, and others).

This new trend changes the focus of sequential programming, widely taught and used in past times and routes for parallel programming, enjoying new environments to access this new technology. Currently, there are 2 million developers registered on the NVIDIA website to use CUDA, a parallel computing platform extended from the C language and widely studied to develop applications as part of the teaching and learning process ([RAMEY, 2021](#)).

Addressing parallel programming in the curricular bases of undergraduate and graduate courses is not something new. The Computing Curriculum proposed by Association for Computing Machinery (ACM) in 2001 included a course on High-Performance Computing ([ROBERTS *et al.*, 1999](#)) **apud** ([CARLEY *et al.*, 2013](#)), and the 2008 interim revision included a course on Parallel Computation ([CASSEL *et al.*, 2008](#)) **apud** ([CARLEY *et al.*, 2013](#)). In the year 2013, a

new review addressed current and pedagogical aspects (CURRICULA; SOCIETY, 2013).

Given its importance, parallel programming must be incorporated into the computer science curriculum as necessary knowledge, rather than complementing the unit of knowledge or being offered an elective course (LI; GUO; ZHENG, 2008). Marowka (2008) argues that the "parallel" prefix can be added to any topic (e.g., architecture, operating systems, among others) due to today's current revolution.

However, parallel programming education brings many challenges, especially in two approaches: theory and practice. The theoretical approach is necessary to know if the teaching method effectively acquires competence for a particular subject. In addition, there is a need to define the laboratory environment and tools required to perform the exercises in the practical approach.

Kim, Jiang and Rajput (2016) highlight the importance of laboratories for the teaching/learning process. The experience is an essential component for parallel programming education, providing students with the opportunities to practice and apply the learned and observed concepts.

1.1 Problem Contextualization

However, adding parallel programming in the computer science curriculum brings a new problem for educators and students: the infrastructure necessary for the course to be delivered with the least resources. Baldwin *et al.* (2016) point out that lecturers need specific HPC resources to prepare students for industry and research positions. This concern about the infrastructure was highlighted in the ACM/IEEE guidelines curriculum:

There are implications for institutional resources to support such a significant scaling up of the teaching mission of computer science departments, particularly in terms of instructors and laboratories (CURRICULA; SOCIETY, 2013, 48).

Furthermore, Curricula and Society (2013, 33) cites that "... students to spend a significant amount of additional time outside of class developing facility with the material presented in class". Thus, emphasizing the need to provide infrastructure for students to develop their studies outside the classroom.

Parallel computers and supercomputers infrastructures are expensive and hardly used in a computer science undergraduate course in many countries, such as Brazil. However, these resources are available remotely over a shared infrastructure, management costs, access issues, user creation, and bandwidth (LI; GUO; ZHENG, 2008).

The popularization of multicore machines has helped in the teaching of parallel programming of MIMD machines (FLYNN, 2011) with shared memory (OpenMP (DAGUM; MENON,

1998)) and even with distributed memory (MPI (CLARKE; GLENDINNING; HEMPEL, 1994)). However, the availability of only these multicore machines limits the teaching of parallel programming, as they do not allow practical teaching on clusters of computers or heterogeneous platforms with GPUs or other accelerators (WOLFER, 2015).

According to Ivica, Riley and Shubert (2009), besides these costs, we can still mention hardware costs, costs associated with hosting, power, and cooling for a typical cluster. Another desirable configuration is to change a cluster for use in classrooms, where it may require some differentiated packages, creating students' accounts, and students and lecturers access manuals.

High-performance computing hardware has a high added value and requires non-trivial configuration. Different hardware must be acquired and managed for each system component. Furthermore, ensuring the application is configured correctly is an expensive task that requires specific technical skills (BALDWIN *et al.*, 2016).

Most of the works focused on the use of clusters and complex infrastructures to make available to students. Generally, the lecturer needs to create an account in a particular cluster, and the student needs to receive the connection parameters. Some systems still try to facilitate access, allowing students to send the code through a web interface for execution, needing to wait for the result, making an online/remote class unfeasible, for example.

This is the case of SAUCE (HUNDT; SCHLARB; SCHMIDT, 2017) and OnRamp (FOLEY *et al.*, 2017), one of the most current web systems found in the literature makes available their entire infrastructure through clusters. The student must send his code and wait for the process to return.

1.2 Motivation

Besides challenges inherent in teaching parallel programming, there is also the challenge of using an infrastructure that allows the execution and testing of these codes, both by students and lecturers. Usually, these parallel infrastructures are expensive and require much energy from a technical team to teach correctly; these structures are not generally available for teaching due to their high cost.

While local computers with multiple cores are already usual nowadays, clusters of computers with remote machines and heterogeneous computing (with GPUs, for example) are still not the reality of many computing courses. This infrastructure is often borrowed from a “near” research center or rented in the cloud for practical parallel programming classes.

These existing solutions sit challenges for teaching parallel programming, such as the presence of multi-users (researchers) with their applications in the parallel architecture execution queue and the high cost, the last one for both the acquisition of this equipment and the rental of them (if it is the case).

Besides such aspects, the hardware for high-performance computing usually requires a non-trivial configuration. For example, there is only one microcomputer in some colleges in Brazil for one or more students to share. Thus, parallel programming education's primary challenge is to develop platforms that fit the reality of the available laboratories in undergraduate and graduate courses. [Carley *et al.* \(2013\)](#) address a challenge for lecturers to find a mechanism to provide resources from a cluster inside the classroom. These challenges include the cost, resources availability, and the desired performance for the tests.

[Jiang and Song \(2015\)](#) point out problems in an infrastructure shared with other courses or disciplines: the coexistence of other programs that use part of the storage space and computational resources, limiting the number of resources available for the parallel execution of programs. [Kim, Jiang and Rajput \(2016\)](#) also emphasize that Virtual Machine (VM) has been used intensively in the last decades to teach information technologies. However, according to [Pahl \(2015a\)](#), this virtualization model requires more resources for its support; for example, more extensive disk storage and initialization is slow.

Given the lack of infrastructure in educational institutions and engaged in the popularization of "parallel thinking", this work addresses the use of containers as a solution for low-cost infrastructure and greater accessibility. Furthermore, unlike proprietary solutions, the containers are open source and are available at no cost to use. Virtual machines have a more significant workload than containers due to virtualization, paravirtualization, or full virtualization modes. On the other hand, containers use the host operating system's kernel and are considered lightweight as they have no emulation layer ([DUA; RAJA; KAKADIA, 2014](#)).

Thus, it is essential to provide a light and virtual infrastructure for parallel programming teaching. The containers can currently meet this need and help lecturers and students in the effectiveness of the teaching and learning process. As previously seen, parallel programming is present in the daily lives of students and lecturers. However, in the related work, it is evident the difficulty of teaching this type of programming and searching for the student's interest in learning new methodologies.

1.3 Research Question

This thesis's research question is: to evaluate if developing a tool can help the practical classes of parallel programming and contribute to improving this teaching in institutions without infrastructure or qualified employees.

1.4 The Main Proposed Innovation

This thesis proposes to develop the Iguana, a novel software tool for the practical teaching of parallel programming in educational institutions without dedicated infrastructure,

and to evaluate its effectiveness. For the complete development of the work, several steps are necessary, as described below:

1. Conduct a systematic review of the state-of-the-art literature on teaching parallel programming in Brazil and worldwide;
2. Perform a systematic review of theoretical and practical teaching methods for PDC;
3. Investigate open-source technologies for creating the tool;
4. Develop a specific container management system for parallel programming teaching, allowing the lecturer to configure and control the execution environment for students;
5. Conduct experiments to validate the developed tool effectiveness; and
6. Publish all research results and the results of this thesis, to disseminate new knowledge to the interested community and those who need the teaching of parallel programming.

As a consequence of this work, we can also cite the use of containers to improve the popularization of parallel programming teaching.

1.5 Thesis Structure

Chapter 1 presents the introduction of the work, and the other chapters were distributed as follows:

- The **Chapter 2** shows the teaching of parallel programming and the current overview in institutions around the world;
- The **Chapter 3** addresses the concepts of virtualization and containerization;
- The **Chapter 4** presents the Iguana tool, developed for the practical teaching of parallel programming;
- The **Chapter 5** addresses the performance evaluation carried out with professionals in computing and with students on the tool's effectiveness;
- In the **Chapter 6**, we present a systematic review of the state of the art of the related work to the parallel programming teaching and the infrastructures available for the practical classes; and
- Finally, in **Chapter 7**, we finalize with conclusions of this work and its contribution to education and society.

PARALLEL PROGRAMMING TEACHING

Parallel systems are being adopted due to the physical limitations in manufacturing chips with a single processor. Thus, as parallel systems, it is possible to use several processing cores connected by bus or network ([EL-REWINI; ABD-EL-BARR, 2005](#)).

With this move to parallel systems, it will be necessary for all students to acquire multi-processor programming skills. Some skills involve using distributed memory, hierarchical memory models, and using GPUs to increase performance ([FERNER; WILKINSON; HEATH, 2013](#)).

International institutions, such as the ACM and the Institute of Electrical and Electronics Engineers (IEEE), recommend that Parallel and Distributed Computing (PDC) should be considered one of the significant areas of knowledge in computing, precisely because of its rise in the educational context.

On the same page, the Brazilian Computer Society recommends that the teaching of PDC is one of the main competencies developed in different training axes in Computer courses. Thus, higher-level computing courses need to incorporate these guidelines on PDC teaching into their curricula to prepare students for an increasingly parallelized and distributed world.

Because of the importance of teaching PDC, this section will show a detailed study of parallel programming teaching in Brazil and the world to know the theoretical and practical approaches to teaching and define gaps in the learning effectiveness.

For this purpose, [Section 2.1](#) will cover the computer science curricula. In [Section 2.2](#), we examine PDC teaching in Brazil. In [Section 2.3](#), we explore how parallel programming is addressed in the world's leading universities. [Section 2.4](#) presents a mapping to identify the approaches for teaching theoretical and practical parallel programming. In [Section 2.5](#), we show the contents covered in a parallel programming class. Finally, [Section 2.6](#) presents the section summary.

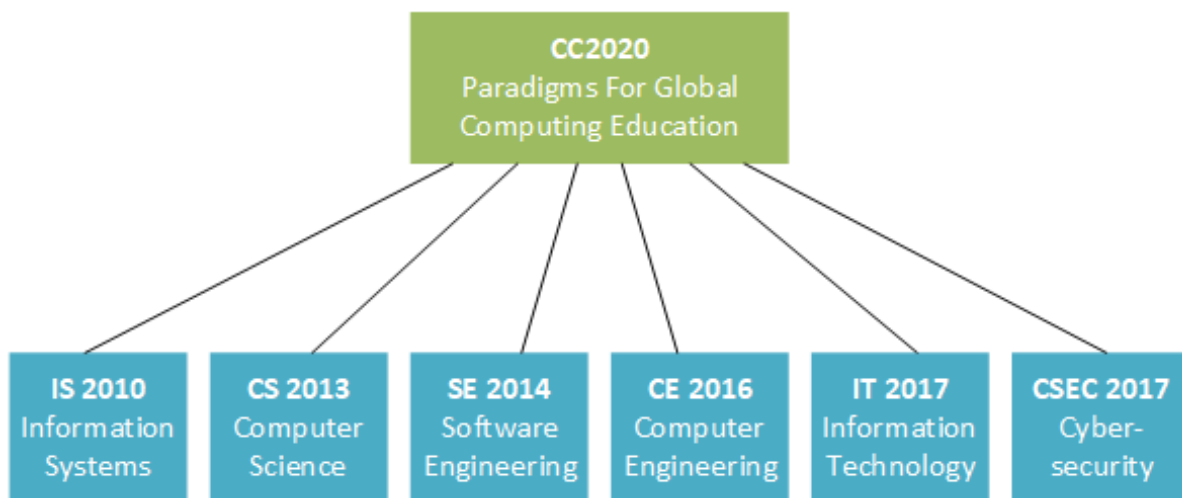
2.1 Computer Science Curricula

The use of single-processor computers in the past has led to programming teaching based on sequential algorithms, in which PDC was usually offered (and at most) as optional ([ACM/IEEE-CS, 2013](#)). However, with the increasing technological development and the consequent adoption of multi-core processors, it was necessary to reorganize the computer curriculum for this new programming need.

The Computing Curricula Series brings a set of curriculum standards related to undergraduate programs in computing. These standards help as a guideline for creating and restructuring courses in the area. [Figure 1](#) shows the design of this report and all other documents used.

For the six-existing discipline specific curricula volumes, each one represents the best judgment of the volunteers representing relevant professional, scientific, and educational associations. Each report serves as a definition of what these degree programs should be and accomplish ([Computing Curricula Series Report, 2020, 19](#)).

Figure 1 – Structure of the Computing Curricula Series



Source: Adapted from [Computing Curricula Series Report \(2020\)](#).

[Table 1](#) presents the minimum (1) and maximum (5) values of the importance of teaching Parallel and Distributed Programming for courses related to computing. These values were scored based on the opinion of the members of the steering committee.

[Computing Curricula Series Report \(2020\)](#) shows the draft competencies to Parallel and Distributed Computing of the disciplines presented in [Table 1](#) to IS 2010, CS 2013, SE 2014, and CE 2016:

- Apply parallel-bases or task-based decomposition to create a scalable parallel algorithm for an IT company;

Table 1 – Computing Knowledge

Discipline	Minimum	Maximum
Information Systems	1	3
Computer Science	2	4
Software Engineering	2	3
Computer Engineering	2	4
Information Technology	1	3
Cybersecurity	1	2

Source: Adapted from [Computing Curricula Series Report \(2020\)](#).

- Write a program containing actors and/or simultaneous processes, locks, and synchronized queues. The program should terminate without error when all concurrent tasks are completed;
- Create a test program that shows a concurrent program error, for example, missing information when two processes try to update a variable;
- Identify and parallelize independent tasks, complete the critical path to a parallel execution diagram, and present the results; and
- Implement a performance evaluation between parallel and sequential solutions by implementing a parallel divide and conquer (and/or graph algorithm).

CS 2013 was created by the ACM and IEEE-Computer Society to help standardize and establish international curriculum guidelines for undergraduate courses in computer science and related fields ([ACM/IEEE-CS, 2013](#)). The guidelines proposed by the 2013 ACM present eighteen knowledge areas, which represent relevant fields of study in computing. PDC is one of these areas, due to its current importance in the development of computational solutions.

Initially, PDC contents were spread among other knowledge areas, including parallel and distributed programming topics. Thus, as shown in [Chart 1](#), the 2013 curriculum revision included PDC contents in a new area, including specific contents such as fundamentals of parallelism, parallel decomposition, parallel algorithms, analysis and programming, and parallel architectures ([ACM/IEEE-CS, 2013](#)).

[Chart 2](#) details the topics needed to be covered within the computer science curricula. The topics of Parallel Performance, Distributed Systems, Cloud Computing and Formal Models and Semantics do not have essential and additional hours, only elective hours.

Teaching PDC in the computing curriculum proposed by ACM/IEEE is quantified in hours. This unit is defined as the time needed to present a traditional reading material. This time does not include extra work, practical classes in laboratories, lectures, among others. [Chart 1](#) shows the essential hours for the Computer Science curriculum; that is, topics need to be inserted

Chart 1 – Teaching PDC in 2013 curriculum guidelines for computer courses, according to ACM/IEEE

Topics	Instructional Hours		Includes Electives
	Essential	Additional	
Parallelism Fundamentals (PF)	2		No
Parallel Decomposition (PD)	1	3	No
Communication and Coordination (CC)	1	3	Yes
Parallel Algorithms, Analysis, and Programming (PAAP)		3	Yes
Parallel Architecture (PA)	1	1	Yes
Parallel Performance (PP)			Yes
Distributed Systems (DS)			Yes
Cloud Computing			Yes
Formal Models and Semantics (FMS)			Yes

Source: [ACM/IEEE-CS \(2013\)](#).

within the curriculum in a specific subject or addressed in another syllabus ([ACM/IEEE-CS, 2013](#)).

The additional hours are usually essential for an undergraduate course in computer science. However, some courses may allow the student to specialize in a specific topic from the third year and do not cover the other topics. Another problem that may influence the application of additional hours concerns administrative issues such as lecturer lack and physical and budgetary constraints.

However, the ACM/IEEE clarify that a curriculum may include elective material as required in those specific courses, specializations, masters, and doctoral degrees, as the essential topics are insufficient for a complete curriculum. Furthermore, a computer science curriculum should cover 90-100% of the add-on topics, with 80% considered a minimum ([ACM/IEEE-CS, 2013](#)).

In Brazil, the Brazilian Computer Society (BCS) is the national academic reference that directs computing's teaching. It defined the document named Training References for the Undergraduate Programs in Computer Science (RF-CC-17) to serve as a basis for Pedagogical Projects ([ZORZO et al., 2017](#)).

BCS recommends, in its training references, that the teaching of PDC is one of the main competencies developed in different axes of formation in Computer courses. Considering this context, higher education courses in Computing in Brazil need to incorporate these guidelines on PDC teaching in their curricula to prepare students for an increasingly parallelized and distributed world.

Chart 2 – Teaching PDC in 2013 curriculum guidelines for computer courses, according to ACM/IEEE

	Essential	Additional	Electives
PF	Multiple simultaneous computations, goals of parallelism, communication, coordination, and programming errors	Not specified	Not specified
PD	Need for communication and coordination. Synchronization, Independence and partitioning	Parallel decomposition concepts, task-based decomposition, data-parallel decomposition and actors and reactive processes	Not specified
CC	Shared memory and consistency	Message passing and atomicity	Barriers, counters, or related constructs, and Conditional waiting (e.g., using condition variables)
PAAP	Not specified	Critical paths, work and span. The relation to Amdahl's law, speed-up, scalability. Naturally (embarrassingly) parallel algorithms, and parallel algorithmic patterns	Parallel graph algorithms, matrix computations, producer-consumer and pipelined algorithms, and non-scalable parallel algorithms
PA	Multicore processors, Shared, and distributed memory	Symmetric multiprocessing (SMP), SIMD, and vector processing	GPU, co-processing, Flynn's taxonomy, instruction level support for parallel programming, memory issues, and topologies
PP	Load balancing, performance measurement, scheduling and contention, evaluating communication overhead, data management, and power usage and management		
DC	Faults, distributed message sending, distributed system design tradeoffs, distributed service design, and core distributed algorithms		
CC	Internet-Scale computing, cloud services, virtualization, and cloud-based data storage		
FMS	Formal models of processes and message passing, parallel computation, shared memory consistency, algorithmic progress, and computational dependencies. Linearizability and techniques for specifying and checking correctness		

Source: Adapted from [ACM/IEEE-CS \(2013\)](#).

[Chart 3](#) points out the topics explained by BCS and are equivalent to the topics referenced by ACM/IEEE in CPD.

Chart 3 – List of PDC topics covered by BCS with ACM/IEEE

ACM/IEEE	SBC
Parallelism Fundamentals (PF)	Concurrent Systems/Parallel Processing (CS/PP)
Parallel Decomposition (PD)	.
Communication and Coordination (CC)	.
Parallel Algorithms, Analysis, and Programming (PAAP)	Parallel and Distributed Programming (DP)
Parallel Architecture (PA)	Parallel Computer Architectures (PCA)
Parallel Performance (PP)	Performance Evaluation (PE)
Distributed Systems (DS)	Distributed Systems (DS)
Cloud Computing	.
Formal Models and Semantics (FMS)	Formal Methods (FM)

Source: Elaborated by the author.

The BCS document does not provide the details of each discipline. It is also possible to note that some areas are not covered, such as Cloud Computing, Parallel Decomposition and Communication, and Coordination. However, these areas may be covered in other topics, such as Parallel and Distributed Programming.

Students need to understand these topics to develop skills and competencies, as guided by computing curricula, to work in these computing environments, increasingly common in the daily lives of companies and universities ([SHAMSI; DURRANI; KAFI, 2015](#)).

Also, students must need to know a considerable range of previous content from other disciplines, increasing its complexity, such as the organization and architecture of computers, operating systems, computer networks, and computer programming ([ZARESTKY; BANGERTH, 2014](#)).

Thus, it is possible to insert PDC topics into existing disciplines in a computer science course. For example, an Algorithms class may include parallel algorithms, a Computer Organization class may cover multicore architectures, and a Programming Languages class may consist of distributed computing message-passing primitives ([ADAMS *et al.*, 2021](#)).

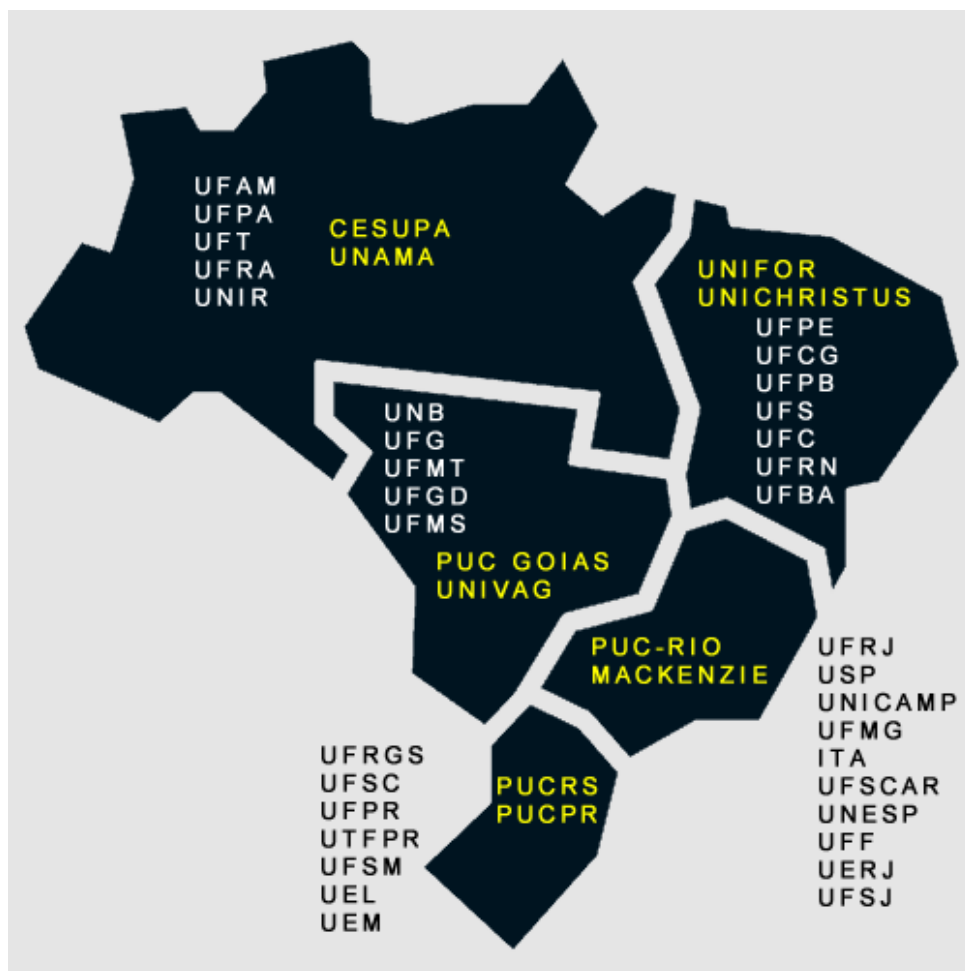
Considering these restructurings in PDC teaching by ACM and IEEE and the importance of parallel/distributed programming in future computer professionals' training, it is essential to determine how universities address parallel programming. The following sections show this scenario.

2.2 The PDC Teaching in Brazil

In this section, we analyze computer education institutions in Brazil, focusing on teaching PDC in some of the most important universities spread across different regions of the country. The survey shows national universities about the teaching of PDC suggested by the ACM, IEEE, and BCS, highlighting convergences and divergences concerning these references, topics not covered, and difficulties in finding official consultation documents.

The 45 universities analyzed were chosen according to the scoring systems available in Folha University Ranking (FUR) ([Folha de São Paulo, 2018](#)) released in 2017 and each region's demographic density ([IBGE, 2010](#)), as shown in [Figure 2](#).

Figure 2 – Brazilian institutions according to geographic location



Source: Elaborated by the author.

FUR was chosen by the considered educational institutions and the use of five distinct indicators: research, internationalization, innovation, teaching, and market. For example, about research, items such as total scientific publications, citations, citations per lecturer, scholarship holders, and productivity, among others, are considered. Regarding teaching, items are considered

a percentage of lecturers with master's and doctorate degrees, full and partial dedication, and Enade's grade (RIGHETTI, 2015).

We selected the number of institutions according to the highest score in the country's ranking and demographic density. Thus, the number of chosen institutions ranged from seven (North and Center-West) to twelve (Southeast). In addition, according to the best positions in the ranking, two private universities were also intentionally selected, whose names are shown in yellow in Figure 2.

The materials consulted for data collection were available on each institution's websites, such as syllabus, teaching plans, lesson plans, and pedagogical projects. The data obtained refer to the semester in which the topic is offered, the number of hours, mandatory or elective, bibliographic references, and prerequisites.

Chart 4 presents the highest-scoring educational institutions for computing and similar courses, topics related to PDC teaching, and ranking position. The discipline of Distributed Systems is the most common, followed by Concurrent Programming.

Considering the researched institutions, only Federal University of Paraíba (2018), Federal University of São João del-Rei (2018), Pontifical Catholic University of Rio de Janeiro (2018), State University of Londrina (2018) do not have topics related to parallelism specifically in their syllabus (indicated as NA) or did not make the information available on their websites; these topics may be inserted into other courses.

Figure 3 shows the prerequisites¹ most used by educational institutions. The topics of Operating Systems, Programming, and Networks are the most common. Of this total, twenty-three institutions have no prerequisites or have not been found. In addition, no institution requires corequisite².

Figure 4 shows the total, practical and theoretical loads for the topics. Of this total, three institutions did not make their hours available, and four did not have these topics. Many universities also show the total loads but do not specify the number of hours reserved for practical and theoretical teaching.

Also, for those institutions that are available information, eight institutions teach only theoretical classes. In contrast, Federal Fluminense University (2018) teaches only practical classes. The average total load is 60 hours, with a maximum of 105 hours (University of São Paulo, 2018) and a minimum of 30 hours (Pontifical Catholic University of Rio Grande do Sul, 2018). The observed average of practical classes is 30 hours and 45 hours for theoretical classes.

Figure 5 provides the semester in which these topics are offered, and eleven institutions did not provide these data. Most institutions offer topics in the 7th semester. It is also interesting to note that none of them offers in the 1st and 2nd semesters, which probably happens due to the

¹ A prerequisite is a topic that needs to be attended to before applying for the PDC course.

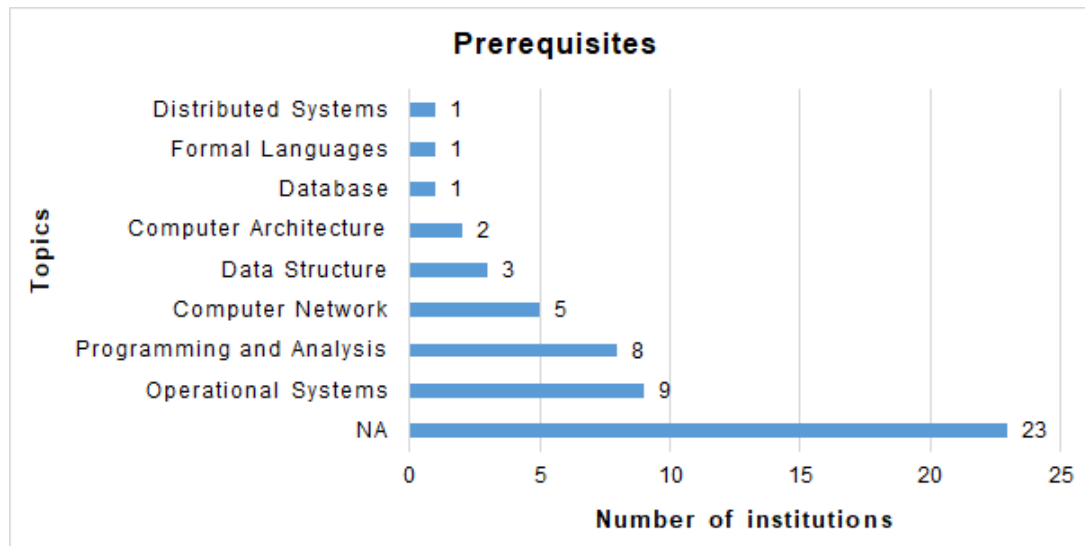
² Corequisite is subject that needs to be taken concurrently with the topic related to the PDC.

Chart 4 – List of Brazilian institutions selected according to ranking scores

	Shortname	Institution	Topic	Ranking
North	UFAM	Federal University in Amazonas (2018)	Distributed Systems	25
	UFPA	Federal University of Pará (2018)	Distributed Systems	30
	UFT	Federal University of Tocantins (2018)	Distributed Systems	46
	UFRA	Federal Rural University of the Amazon (2018)	Distributed Systems	156
	UNIR	Federal University of Rondônia (2018)	Distributed Systems	178
	CESUPA	University Center of the State of Pará (2018)	Distributed and Parallel Programming	34
	UNAMA	State University of Amazonas (2018)	Distributed Systems	58
Northeast	UFPE	Federal University of Pernambuco (2018)	Concurrent Programming Distributed	5
	UFCG	Federal University of Campina Grande (2018)	Concurrent Programming Fundamentals	15
	UFPB	Federal University of Paraíba (2018)	NA	20
	UFS	Federal University of Sergipe (2018)	Distributed Systems	21
	UFC	Federal University of Ceará (2018)	Parallel Computing	26
	UFRN	Federal University of Rio Grande do Norte (2018)	Concurrent Programming	42
	UFBA	Federal University of Bahia (2018)	Distributed Algorithms	44
	UNIFOR	University of Fortaleza (2018)	Distributed Systems	33
	UNICHRISTUS	Christus University Center (2018)	Distributed Systems	61
Midwest	UNB	University of Brasília (2018)	Concurrent Programming	11
	UFG	Federal University of Goiás (2018)	Parallel Computing	17
	UFMT	Federal University of Mato Grosso (2018)	Distributed Systems	51
	UFGD	Federal University of Grande Dourados (2018)	Distributed Systems	149
	UFMS	Federal University of Mato Grosso do Sul (2018)	Parallel Programming	154
	PUC GOIÁS	Pontifical Catholic University of Goiás (2018)	Distributed Systems	60
	UNIVAG	University Center of Varzea Grande (2018)	Advanced Networks and Distributed Systems	100
Southeast	UFRJ	Federal University of Rio de Janeiro (2018)	Concurrent Computing	1
	USP	University of São Paulo (2018)	Concurrent Programming	2
	UNICAMP	State University of Campinas (2018)	Introduction to Parallel Programming	3
	UFMG	Federal University of Minas Gerais (2018)	Projects and Concepts of Parallel and Distributed Systems	6
	ITA	Technological Institute of Aeronautics (2018)	Distributed Processing	8
	UFSCAR	Federal University of São Carlos (2018)	Distributed Systems	9
	UNESP	São Paulo State University (2018)	Distributed Computing	13
	UFF	Federal Fluminense University (2018)	Parallel Programming Laboratory	27
	UERJ	State University of Rio de Janeiro (2018)	Distributed Systems	28
	UFSJ	Federal University of São João del-Rei (2018)	NA	36
	PUC-RIO	Pontifical Catholic University of Rio de Janeiro (2018)	NA	10
	MACKENZIE	Mackenzie Presbyterian University (2018)	Parallel Computing	18
South	UFRGS	Federal University of Rio Grande do Sul (2018)	Distributed and Parallel Programming	4
	UFSC	Federal University of Santa Catarina (2018)	Concurrent Programming	7
	UFPR	Federal University of Paraná (2018)	Parallel Programming	12
	UTFPR	Federal Technological University of Paraná (2018)	Distributed Systems	16
	UFMS	Federal University of Santa Maria (2018)	Distributed Systems	22
	UEL	State University of Londrina (2018)	NA	40
	UEM	State University of Maringá (2018)	Concurrent Programming	48
	PUCRS	Pontifical Catholic University of Rio Grande do Sul (2018)	Models for Concurrent Computing	14
	PUCPR	Pontifical Catholic University of Paraná (2018)	Distributed Systems	24

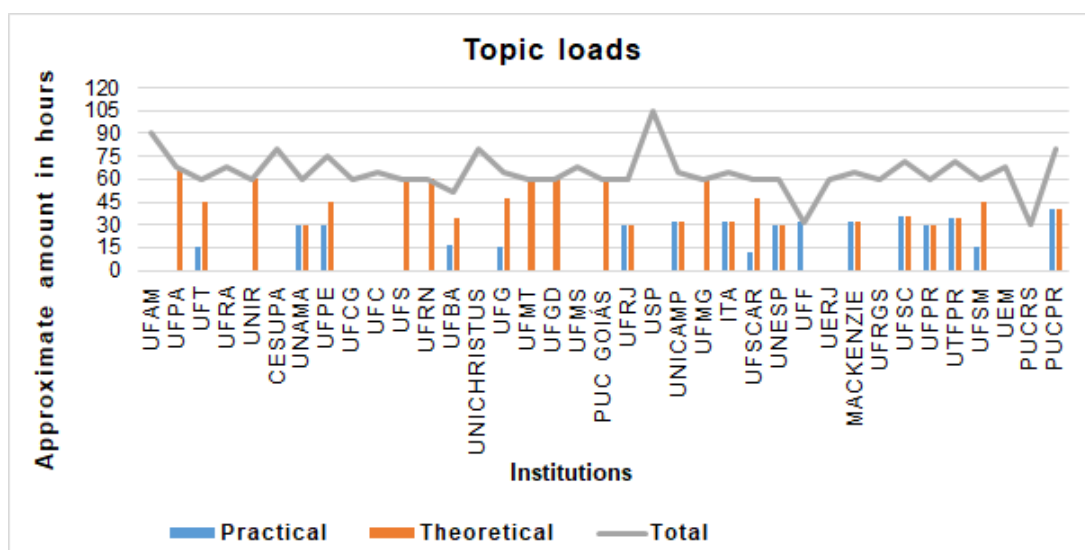
Source: Elaborated by the author.

Figure 3 – Prerequisites of the topics related to PDC for Brazilian institutions



Source: Elaborated by the author.

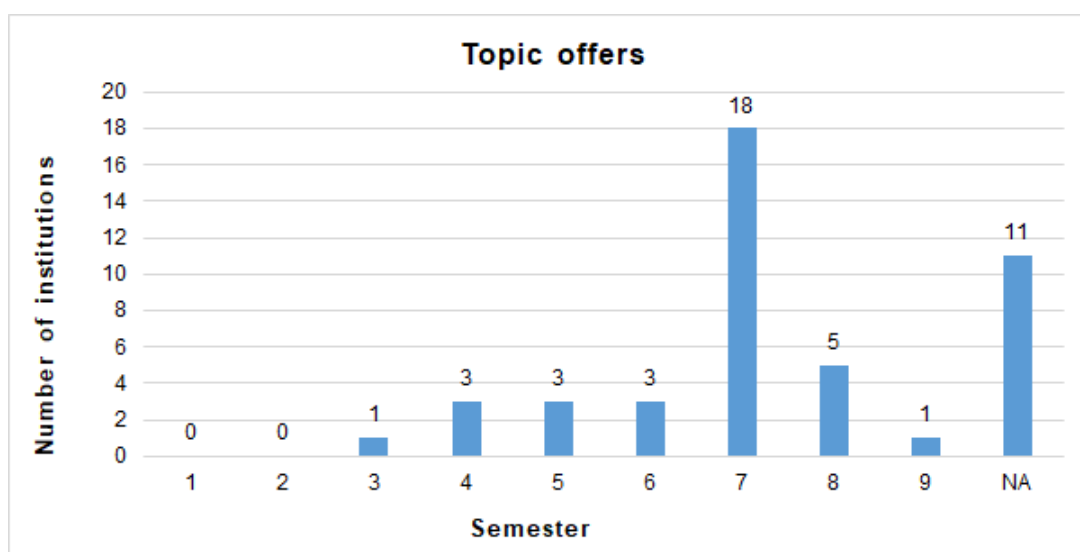
Figure 4 – Total loads of the topics related to PDC for Brazilian institutions



Source: Elaborated by the author.

time required to fulfill the prerequisites.

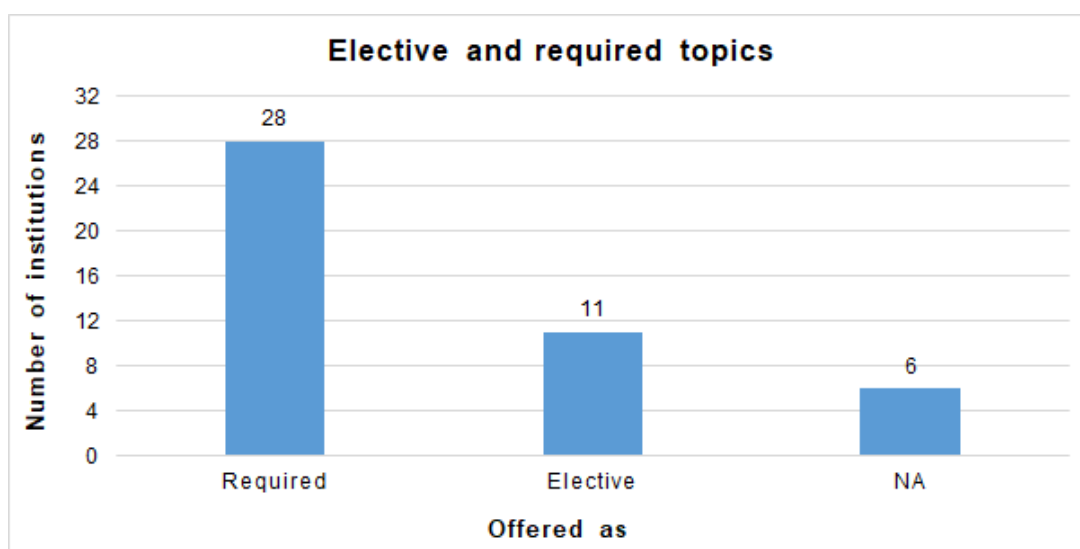
Figure 5 – Semester of the provision of the topics related to PDC for Brazilian institutions



Source: Elaborated by the author.

Figure 6 shows whether the topic is offered as an elective or required. It is usually required, with six institutions not providing the information.

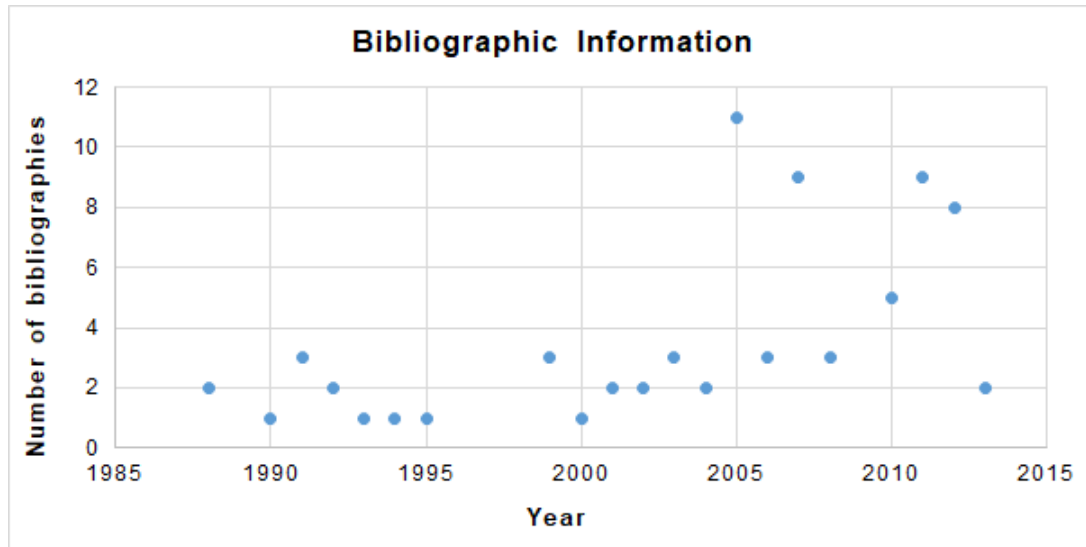
Figure 6 – Elective and required topics about PDC for Brazilian institutions



Source: Elaborated by the author.

Figure 7 shows the grouping of bibliographies by the year used in those institutions' syllabus available online. Most institutions have bibliographies published between 2005 and 2012. However, some institutions work with bibliographies from the 1980s (1988), while the most current is from 2013.

Figure 7 – Bibliographic information of the topics related to PDC for Brazilian institutions



Source: Elaborated by the author.

[Chart 5](#) shows the PDC topics' adherence to the ACM/IEEE and BCS guidelines. We listed in the table only institutions with the syllabus that we found on the official website. Unfortunately, the BCS references do not include the topics' specifications; therefore, we grouped the discipline according to the name and content.

Few institutions teach the last topics of [Chart 5](#), such as Cloud Computing and Formal Methods. Also, 49% of the investigated courses explicitly offer parallel programming in their grades and only 20% address topics about parallel performance.

According to [ACM/IEEE-CS \(2013\)](#), parallel and distributed programming remains more complex in contemporary programming environments. Thus, assimilating these programming models can help students acquire the necessary skills in this area, making parallel teaching vital for computer courses. However, it is possible to note that most courses have chosen to offer this topic in the later semesters due to its high complexity and prerequisites.

Many educational institutions use non-contemporary bibliographies, not addressing relatively new topics, such as Cloud Computing. It is also highlighted that only 62% of these institutions offer the topic as required, while the others do not offer it as required or do not even offer it.

Another critical key is the loads for a topic's success. In this study, we noticed a significant divergence in the number of hours needed. The average indicated it was around 60 hours, while the ACM/IEEE points 80 minimum hours.

Regarding the institutions' adherence to the ACM/IEEE and BCS curriculum guidelines, many institutions go beyond the minimum proposal, covering current topics. However, many universities do not address essential topics, such as Parallel and Distributed Programming, a

Chart 5 – Adherence of topics on PDC with the ACM/IEEE and BCS reference curricula for Brazilian institutions.

	BCS	CS/PP			PDP	PCA	PE	DS		FM
	ACM/IEEE	PF	PD	CC	PAAP	PA	PP	DS	Cloud	FMS
Institution	UFAM	X		X		X		X		
	UFPA	X		X		X		X		
	UFT	X		X		X		X		
	UFRA	X		X		X		X		
	UNIR	X	X	X	X	X	X			
	CESUPA	X	X	X	X	X		X		
	UFPE	X	X	X	X					
	UFCG	X		X		X	X			
	UFS	X		X		X		X		
	UFC	X	X		X		X			
	UFRN	X	X	X	X					
	UFBA	X	X	X	X	X				
	UNIFOR	X		X	X	X		X		
	UFG	X	X	X	X	X	X			
	UFMT	X		X				X		
	UFGD	X		X	X			X		
	UFMS	X		X	X	X	X	X		
	PUC-GOIAS	X		X				X		
	UFRJ	X	X	X	X			X		
	USP	X	X	X	X	X	X			X
	UNICAMP	X	X	X	X	X				
	UFMG	X		X			X	X	X	
	ITA	X	X	X	X			X		
	UFSCAR	X		X	X	X		X	X	
	UNESP	X						X	X	
	UFF	X		X	X	X				
	MACKENZIE	X			X	X				
	UFRGS	X	X	X	X	X		X		
	UFSC	X		X	X					
	UFPR	X		X	X		X			
	UTFPR	X		X		X		X		
	UFSM	X		X				X		
	UEM	X		X	X		X			
	PUC-RS	X		X						X

Source: Elaborated by the author.

subject of extreme importance today.

2.3 The PDC Teaching Around the World

In this section, we analyze the teaching of PDC in the most important universities on their respective continents, according to the main rankings in the area. In addition, we also highlight the divergences and convergences about the teaching of PDC suggested by the ACM and IEEE.

We chose 30 universities according to the scoring systems available in two rankings used worldwide: World University Rankings (WUR) ([The World University Ranking, 2018](#)) and QS World University Rankings ([QS World University Rankings, 2018](#)).

The rankings selected use indicators such as academic reputation, employer reputation, faculty/student ratio, citations by faculty, international faculty index, international student index, among others. In addition, we chose five universities according to their location in North America, Europe, Asia, South America, Africa, and Oceania to limit the results.

Each ranking was filtered by the highest-scoring institutions for computer science courses. Then, we calculated the scores based on position in the two rankings; for example, Stanford University (USA) is #1 in the WUR and #2 in the QS, so its total score was 3. [Chart 6](#) presents the position of each institution according to the sum of the scores for the respective rankings.

For the African continent, we used the ranking called Center for World University Rankings (CWUR) ([Center for World University Rankings, 2018](#)) of equal notoriety; unfortunately, the WUR was not available for that continent.

The collected data have been searched on the official websites of each educational institution, such as teaching plans, curriculum matrices, and syllabus. We filtered information by bibliography used, prerequisites, corequisites, course load, and required or elective. The information might be outdated. Many universities did not provide the full syllabus, only parts they considered significant, and this could affect the results of this study.

For example, [Peking University \(2018\)](#), [Nanyang Technological University \(2018\)](#), [Pontifical Catholic University of Chile \(2018\)](#) do not have information on their websites about the computer science course topics. We sent an email to these institutions requesting this information, but we received no response.

Many topics about PDC may be inserted into other subjects, such as Computer Programming and Operating Systems. It is also possible that PDC topics were made available in other courses/areas or provided as an extension program. However, the focus of the work is undergraduate courses.

[Chart 7](#) shows the names of the disciplines found related to PDC, precisely parallel and distributed programming. In addition, the table shows the offer type and prerequisite for a particular topic. Unfortunately, few institutions offer PDC topics as required, making it difficult

Chart 6 – List of world institutions selected according to ranking scores

	Shortname	Institution	Country	Ranking	Score
North America	Stanford	Stanford University (2018)	EUA	W(1) Q(2)	3
	MIT	Massachusetts Institute of Technology (2018)	EUA	W(2) Q(1)	3
	CMU	Carnegie Mellon University (2018)	EUA	W(6) Q(3)	9
	Harvard	Harvard University (2018)	EUA	W(11) Q(6)	18
	Princeton	Princeton University (2018)	EUA	W(12) Q(8)	20
Europe	Oxford	University of Oxford (2018)	UK	W(3) Q(7)	9
	Cambridge	University of Cambridge (2018)	UK	W(5) Q(5)	10
	ETH	ETH Zurich (2018)	Switzerland	W(4) Q(9)	13
	Imperial	Imperial College London (2018)	UK	W(9) Q(12)	21
	École	École Polytechnique Fédérale de Lausanne (2018)	Switzerland	W(10) Q(18)	28
Asia	NUS	National University of Singapore (2018)	Singapore	W(13) Q(10)	23
	Tsinghua	Tsinghua University (2018)	China	W(20) Q(20)	40
	Peking	Peking University (2018)	China	W(25) Q(17)	42
	Hong Kong	Hong Kong University of Science and Technology (2018)	Hong Kong	W(28) Q(14)	42
	Nanyang	Nanyang Technological University (2018)	Singapore	W(31) Q(16)	47
South America	USP	University of São Paulo (2018)	Brazil	W(?) Q(51)	51
	Chile	University of Chile (2018)	Chile	W(?) Q(51)	51
	Unicamp	State University of Campinas (2018)	Brazil	W(?) Q(101)	101
	UBA	University of Buenos Aires (2018)	Argentina	W(?) Q(151)	151
	PUC Chile	Pontifical Catholic University of Chile (2018)	Chile	W(201) Q(101)	301
Oceania	Melbourne	University of Melbourne (2018)	Australia	W(39) Q(14)	53
	Sydney	University of Technology Sydney (2018)	Australia	W(83) Q(36)	119
	Australian South Wales	Australian National University (2018) University of New South Wales (2018)	Australia Australia	W(83) Q(37) W(101) Q(41)	120 142
	Queensland	Queensland University of Technology (2018)	Australia	W(98) Q(51)	149
Africa	Cape Town	University of Cape Town (2018)	South Africa	CW(223) Q(301)	521
	Witwatersrand	University of the Witwatersrand (2018)	South Africa	CW(230) Q(?)	230
	Stellenbosch	Stellenbosch University (2018)	South Africa	CW(448) Q(401)	849
	Pretoria	University of Pretoria (2018)	South Africa	CW(438) Q(401)	839
	Cairo	Cairo University (2018)	Egypt	CW(452) Q(301)	753

Source: Elaborated by the author.

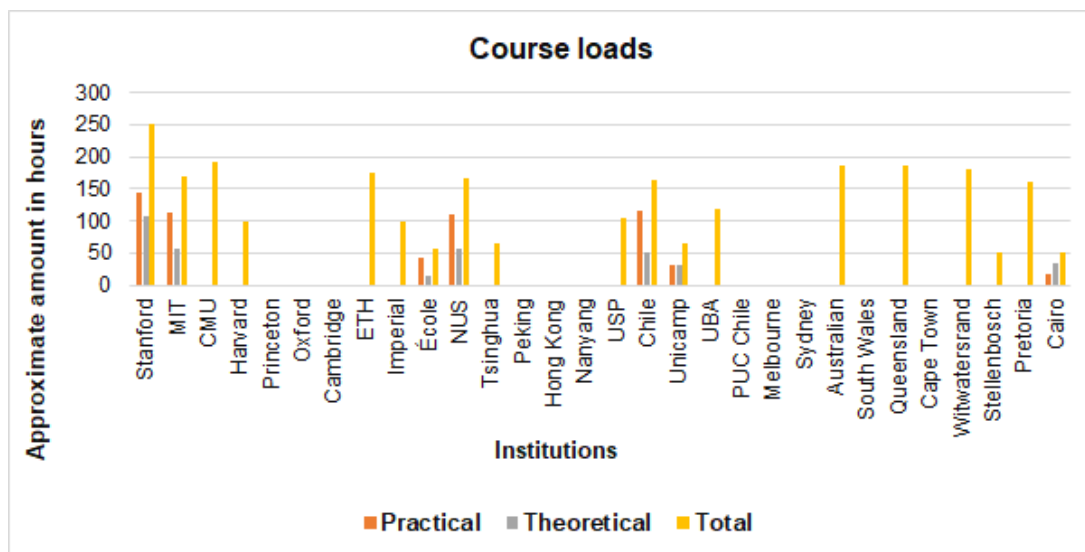
to disseminate "parallel thinking".

We analyzed 30 universities, where 11 institutions did not have prerequisites or were not informed (NI), and six institutions did not present data (NA). Operating Systems, Programming, and Architecture are the most common prerequisites, and no institution presented corequisites.

Figure 8 presents 22 institutions that do not show hours dedicated to practical and theoretical teaching; these institutions only inform the total course load. Only seven universities detailed the number of theoretical and practical hours. Despite the limited amount of information, a large divergence in the total load number between institutions was detected, directly impacting the student's knowledge and skills.

Also, converting different course load formats makes the calculation process a bit subjective. For example, in Europe, the credits can be called European Credit Transfer System (ECTS) or Full-time Equivalent Student Load (EFTSL) and are stipulated in hours. In North America, course loads are described in *units* and can have different values for each university.

Figure 8 – Total loads of the topics related to PDC for the world institutions



Source: Elaborated by the author.

Figure 9 shows the year of offering the topic related to PDC. Unfortunately, most institutions, 22 universities, do not specify which year the students take the discipline. This can happen because students need to fulfill prerequisites before starting a topic or because the course is optional and depends on offers.

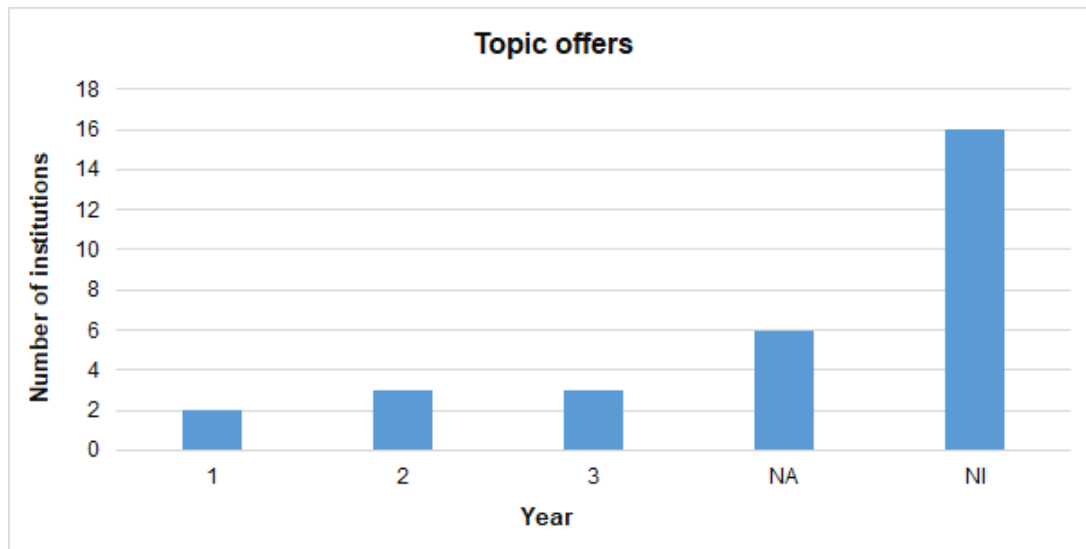
Despite the complexity of PDC topics, some institutions offer these disciplines in the first year, for example, [École Polytechnique Fédérale de Lausanne \(2018\)](#) and [Stellenbosch University \(2018\)](#), preparing students earlier. Unlike École, which is a full bachelor's degree, Stellenbosch is explicitly targeted to Parallel Computing. However, most institutions have chosen to teach PDC in the last years.

Chart 7 – Topics and prerequisites related to PDC for the world institutions

Shortname	Topic	Offer	Prerequisite
Stanford	Parallel Computing	Elective	Compilers; Principles of Computer Systems
MIT	Multicore Programming	Elective	Introduction to Algorithms
CMU	Parallel Computer Architecture and Programming	Elective	Intro to Computer Systems
Harvard	Introduction to Distributed Computing	Elective	Operating Systems
Princeton	Distributed Systems	Elective	Introduction to Programming Systems; Operating Systems; Advanced Programming Techniques
Oxford	Concurrent Programming	Required	Object Oriented Programming
Cambridge	Concurrent and Distributed Systems	Required	Operating Systems
ETH	Parallel Programming	Required	NI
Imperial	Distributed Algorithms	Elective	Concurrency, Maths
École	Parallelism and Concurrency	NI	Functional programming; Algorithms; Computer Architecture
NUS	Parallel and Concurrent Programming	Elective	NI
Tsinghua	Distributed Computing	NI	NI
Peking	NA	NA	NA
Hong Kong	Parallel Programming	Elective	NI
Nanyang	NA	NA	NA
USP	Concurrent Programming	Required	NI
Chile	Parallel Computing and Applications	Elective	Design and Analysis of Algorithms
Unicamp	Introduction to Parallel Programming	Elective	NI
UBA	Complex Systems in Parallel Machines	Elective	NI
PUC Chile	NA	NA	NA
Melbourne	NA	NA	NA
Sydney	NA	NA	NA
Australian	Parallel Systems	Elective	NI
South Wales	Distributed Systems	NI	NI
Queensland	High Performance and Parallel Computing	NI	NI
Cape Town	NA	NA	NA
Witwatersrand	Parallel Computing III	NI	Operating Systems II; Computer Networks II; Analysis of Algorithms II
Stellenbosch	Concurrent Programming 1 and 2	Required	NI
Pretoria	Concurrent Systems	Required	Operating Systems; Data Structures and Algorithms
Cairo	Parallel Processing	Required	Computer Architecture and Organization

Source: Elaborated by the author.

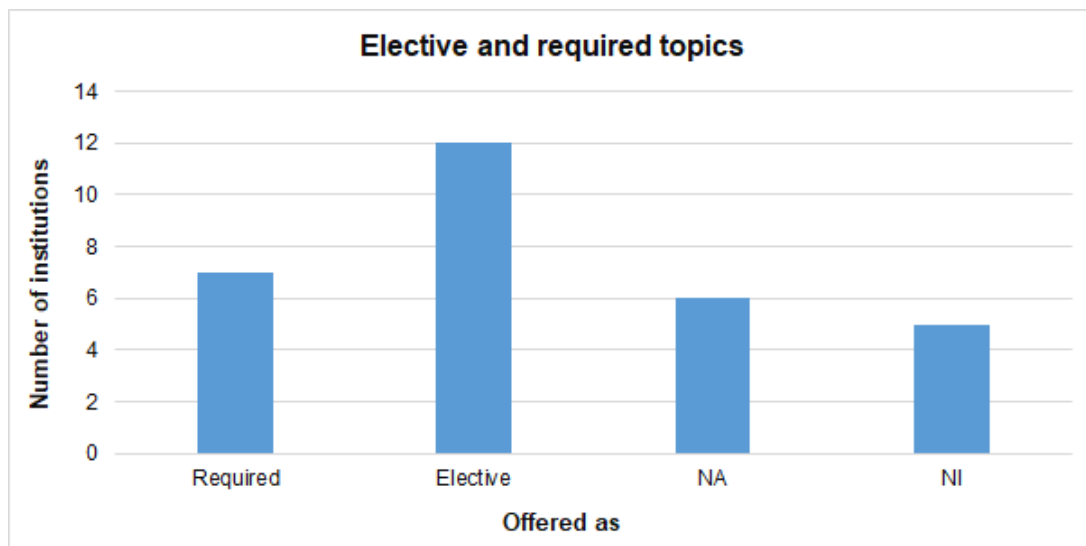
Figure 9 – Year of the provision of the topics related to PDC for the world institutions



Source: Elaborated by the author.

Figure 10 shows 12 institutions in which the discipline about PDC is elective. However, only in seven universities, the subject is required. In addition, five institutions did not present any information (NI), and six only offered the discipline in graduate studies and did not present the course syllabus (NA).

Figure 10 – Elective and required of the topics related to PDC for the world institutions

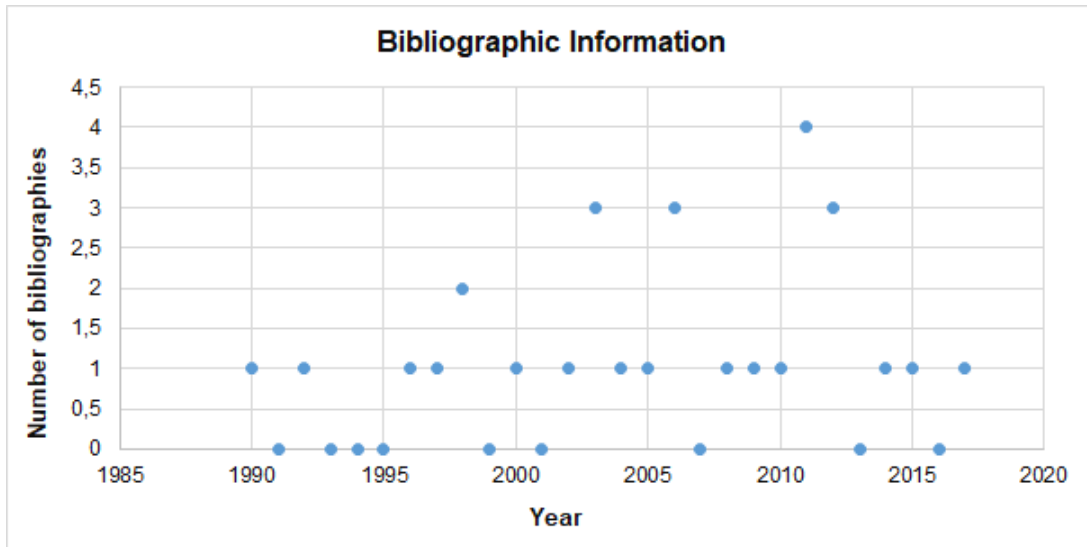


Source: Elaborated by the author.

Figure 11 groups the bibliographies used in the PDC topics. Most bibliographies are between 1990 and 2017, with more than 20 years of publication. However, the information may be incorrect or outdated on the websites of these institutions.

Considering that the information on the websites is correct, many educational institutions

Figure 11 – Bibliographic information of the topics related to PDC for the world institutions



Source: Elaborated by the author.

use outdated bibliographies concerning new technologies, such as Cloud Computing. Unfortunately, this type of approach can sit problems for the student to acquire real-world skills (QIU *et al.*, 2017).

Chart 8 maps the topics that educational institutions cover. Most universities teach essential topics following ACM/IEEE guidelines. However, the subject of Parallel Architecture was inserted in the syllabus only by some institutions.

Also, some institutions, for example, National University of Singapore (2018) insert all topics (essential, additional, and elective) inside one discipline, perhaps not teaching the subject in-depth, increasing the number of hours, and making the course load intense. However, Figure 12 shows a Parallel Computing course that covers all topics in the ACM/IEEE reference curriculum.

Some institutions, such as University of Cambridge (2018), have placed all course material available online for anyone to download, such as annotated lessons, exercises, program codes, exercise solving, and ready-made lessons. This type of action helps to increase the teaching of PDC. However, some courses are still not covered by universities, such as Cloud Computing, Formal Models, and Parallel Performance.

2.4 Mapping of Theoretical and Practical Teaching Approaches

With the study carried out in Section 2.2 and Section 2.3, it was possible to know how the teaching of PDC is approached in Brazilian and world teaching institutions. With this information,

Chart 8 – Adherence of PDC topics to ACM/IEEE reference curricula for world institutions

Institution	Topics								
	PF	PD	CC	PAAP	PA	PP	DS	Cloud	FMS
Stanford	X	X	X	X	X				
MIT	X	X	X	X					
CMU	X	X	X	X	X				
Harvard	X	X	X	X			X		
Princeton	X	X	X	X			X		
Oxford	X	X	X	X					
Cambridge	X	X	X		X		X		
ETH	X	X	X	X					
Imperial	X	X	X	X					
École	X	X	X	X					
NUS	X	X	X	X	X	X	X	X	
Tsinghua	X	X	X				X		
Hong Kong	X	X	X	X					
USP	X	X	X	X	X	X	X		X
Chile	X	X	X	X	X				
Unicamp	X	X	X	X	X				
UBA	X	X	X	X	X				
Australian	X	X	X	X	X	X			
South Wales	X	X	X	X	X	X	X		
Queensland	X	X	X	X	X	X		X	
Witwatersrand	X	X	X	X					
Stellenbosch	X	X	X	X					
Pretoria	X	X	X	X					
Cairo	X	X	X	X	X	X			

Source: Elaborated by the author.

we carried out a mapping of the learning of parallel programming, a part of PDC, to identify the theoretical and practical approaches used for this type of teaching.

Based on the previous sections, but specifically in [Chart 4](#) and [Chart 7](#), we performed a systematic mapping with the PICO³ strategy, where a search string was defined, in the primary databases (ACM, IEEE, Scopus, and Springer): *(teach* OR education OR learn*) AND (parallel or distributed) AND (programming OR computing OR concurrent)*. The search returned 180 articles, and after reading the title and abstract, we selected 24 that addressed the teaching of parallel programming. Below, we present a summary of each selected article.

[Joiner et al. \(2006\)](#) demonstrate tools for teaching high-performance computing through two educational methods. The first method uses a bootable Compact Disc (CD) that temporarily

³ PICO stands for an acronym for Patient, Intervention, Comparison, and Outcome. These four components are the essential elements of the research question in the Evidence-Based Practice discipline and the construction of the question for the bibliographic search for evidence ([SANTOS; PIMENTA; NOBRE, 2007](#)).

Figure 12 – Parallel computing course

Parallel Computing	
Primaries	
<ul style="list-style-type: none">• CS3210 Parallel Computing• CS3211 Parallel and Concurrent Programming• CS4231 Parallel and Distributed Algorithms• CS4223 Multi-core Architecture	
Electives	
<ul style="list-style-type: none">• CS4237 Systems Modelling and Simulation• CS4271 Critical Systems and Their Verification• CS4345 General-Purpose Computation on GPU• CS5207 Foundation in Operating Systems• CS5222 Advanced Computer Architectures• CS5223 Distributed Systems• CS5224 Cloud Computing• CS5239 Computer System Performance Analysis• CS5250 Advanced Operating Systems	

Source: [National University of Singapore \(2018\)](#).

transforms the workstation into a cluster for running parallel code. The second method uses a cluster prototype. For the teaching part, they used practical implementation and collected feedback.

[Marowka \(2008\)](#) shows an introductory course in parallel programming for third-year undergraduate students in Computer Science. The course addresses representative, theoretical and practical topics. It focuses on three points: how to show students that parallel computing is everywhere, how to train students to think in parallel, and which parallel programming model is used to practice this type of programming.

[Li, Guo and Zheng \(2008\)](#) approach a 16-week course introducing parallel and distributed computing principles and techniques to computer students. The course content is divided into parallel programming, parallel algorithm, and parallel computer architecture. Classes are divided into theoretical and practical, using OpenMP and Message Passing Interface (MPI) for the laboratory.

[Ivica, Riley and Shubert \(2009\)](#) present a parallel programming education system consisting of a preconfigured virtual machine image that, through scripts, builds a cluster based on the Amazon Elastic Computing Cloud (EC2). This cluster, called StarHPC, is shared by the class for running parallel codes.

Arroyo (2013) proposes a set of modules for parallel programming teaching, starting with basic computation, advanced high performance, and parallel and distributed programming themes. Also, the paper cites the development of a C++ library with a set of parallel patterns for program development. This library was called Parallel and Distributed Templates (PDT) and focuses on teaching programming through parallel patterns or skeletons.

Adams, Brown and Shoop (2013) show two approaches to parallel programming teaching. The first approach is the use of parallel programming standards based on the professionals' experience. These standards are delivered to students as a collection of problems to be solved. The second approach involves using parallel programming techniques to solve some real issues, motivating students to learn the principles and practices of parallel programming.

Branco *et al.* (2013) propose a course with two approaches for advanced undergraduate or graduate students. First, the method uses books as expository materials, and in the practical part, the students execute prepared codes in the laboratory. Then, after they understand the code, the students could change it. The first approach is to use wireless sensor networks to define topics such as fault tolerance and coordination. The second approach is to insert a new topic to improve understanding of a multitasking level.

Ferner, Wilkinson and Heath (2013) have developed educational materials based on two approaches to parallel programming teaching for graduate students. The first approach uses proprietary software that creates a high-level abstraction for parallel and distributed programming based on the pattern programming approach. The second approach uses compiler guidelines to describe how a program is to be paralleled.

Ferner, Wilkinson and Heath (2014), in another paper, present the results obtained with the materials developed, where it can be concluded that 1) teaching parallel computing in the context of patterns has a positive impact on student learning; 2) teaching the lower-level tools first would be beneficial; 3) the improvements made to the Paragun compiler directives significantly improved the student's confidence in using the tool, and 4) the lower-level tools can still be taught in the context of patterns.

Shafi *et al.* (2014) present an overview of the undergraduate software engineering course. The course was divided into three sections, the first one dealing with parallel programming techniques for shared memory systems. The second section discusses parallel tools used for distributed memory and cluster. The last approach included advanced topics such as MapReduce and the General Purpose Computing on Graphics Processing Units (GPGPU)⁴.

Burkhart, Guerrero and Maffia (2014) show a project-based approach for solving specific problems. In their paper, two methods are defined, standalone and integrated. Students need to configure the development environment in the standalone approach, generate the data, and analyze the results, taking time and learning problems. Also, a pedagogical tool was developed

⁴ GPGPU is the use of a GPU, to perform computation in applications traditionally handled by the CPU.

in the integrated approach that makes the environment preconfigured and integrated. In this environment, the necessary tools are available to solve problems and analyze the results.

Zarestky and Bangerth (2014) describe the use of the flipped classroom in teaching HPC. In this type of approach, the instructor offers the homework and after the classroom activity is performed. The course was administered based on three components: flipped format, research journals (the students should describe what they saw in the homework), and reflective writing (the students should analyze the homework they have learned). Thus, the authors concluded that the students had a greater involvement with the course content, increased motivation, and ease of communication with the instructor.

Shamsi, Durrani and Kafi (2015) take a hands-on approach based on programming and using multiple HPC platforms. The theoretical load of the course is extensive and covers all subjects planned for an HPC course. In addition, teaching methodology, feedback techniques, interactive learning, peer discussion, and group discussions are used.

Dolgopolas *et al.* (2015) present a methodology based on a constructivist approach. This approach is model-centered and comparative learning through programming models based on stochastic simulations provided by a multi-phase queuing system. This queuing system is used for the simplicity and possibilities of parallelization, allowing several experiments to compare and investigate the effectiveness of the parallelized method. Parallelization methods include distributed and shared memory and hybrid parallelization using OpenMP and MPI.

UI-Abdin and Svensson (2015) provide an analytical approach to parallel computing education, focusing on embedded systems' performance and energy efficiency. The course is divided into theoretical and practical, where the practical part includes a small project and seminar. The teaching methods are based on constructivist learning theory, where students are involved in knowledge construction and analysis of real case studies.

Nezu (2015) presents a short work where materials containing parallel programming exercises are used. The basic idea of the material is to allow students to program with their Linux computers.

Schlarb, Hundt and Schmidt (2015) present software for the evaluation and automated for measurement of parallel programs' source code. The "System for AUtomatic Code Evaluation" (SAUCE) system is free, open-source software written in Python, running through a web browser. This tool provides immediate feedback, can be used in the classroom, and supports MPI, OpenMP, and CUDA.

Cuenca and Giménez (2016) use bi-objective optimization problems⁵ to teach parallel programming to students of the Software Engineering course. This problem-based learning

⁵ The bi-objective optimization problem is a multi-criteria decision-making area, which relates to mathematical optimization problems involving more than one objective function to be optimized simultaneously (DEB, 2014).

focuses on practical work and consists of optimizing execution time and energy consumption in a primary heterogeneous cluster. The course lasts one semester and covers topics such as review of parallel architectures, parallel programming paradigms, and parallel programming tools.

Liu (2016) emphasizes the use of coding in his classes. The need to prioritize the coding will also help the student gain experience in acquiring concepts such as process communication, blocking, and load balancing. Clusters are used for laboratory classes with some possible architectures such as Uniform Memory Access (UMA)⁶, GPUs, and clusters Beowulf. Students need to do practical programming projects, as well as research and presentations.

Eijkhout (2016) presents a standard sequence of steps for parallel programming teaching with MPI. The article describes the required exercises and workshops to complete the teaching and learning process, starting with mechanisms that emphasize the symmetry between processes in the Single Program Multiple Data (SPMD)⁷ programming models. In work, it is reiterated that OpenMP should be taught after MPI because of its more intuitive parallelism model. It will be more difficult for students to understand MPI's symmetric model if they acquire concepts like master thread and parallel regions.

Moore and Dunlop (2016) used the flipped classroom's pedagogical approach, in which the typical reading and lesson elements of a course are reversed. For this, small videos and lectures are sent to students before the start of the lesson. After the classroom, the time is dedicated to exercises and discussions about the video's covered concepts. Compared to the traditional teaching approach, the paper concludes that this new approach motivated students and showed better results. Also, to these aspects, they were grouped by mixing more experienced with less experienced students. The class was also composed of undergraduate and graduate students.

Baldwin *et al.* (2016) present a teaching tool called Scholar for computational literacy. This tool is used in classrooms to teach HPC and conduct experiments. To provide access to high-performance computing, Scholar has access to the "Hathi" cluster, a 576-node HPC system, built with 2.6 GHz Intel Xeon E5-2660 processors. Each node has 64 GB of RAM and is interconnected with 56Gb FDR Infiniband. It still provides three software packages: Thin-line, Rstudio Server, and Jupyterhub. Scholar still supports Apache Hadoop YARN and Spark on batch computing nodes.

Gardner and B. (2017) describe a paper report the experience of parallel teaching for undergraduates. In addition, they present a Pilot library, which represents a layer above the MPI to assist students in scheduling message passing.

Grossman *et al.* (2017a) present a parallel programming introductory course, discussing the course structure and teaching approaches. The method uses a partially flipped classroom

⁶ UMA is a shared memory architecture used in parallel programming.

⁷ SPMD describes an execution method with several parallel computing operating units so that the same instruction is applied simultaneously to several data to produce more results.

because it is used in the last minutes of class and uses quizzes, videos, and homework to increase the pedagogical issues. In addition, they used the HJlib and Habanero Autograder tools for laboratory classes.

Capel, Tomeu and Salguero (2017) created a teaching model that involves a demonstrative approach to constructing the parallel program. First, examples are selected through code fragments and a set of standard algorithms. These cases are available to students from Moodle, where students follow specific steps in a work cycle: the teacher explains the particular example in a lecture; students develop the program to solve a specific problem; and finally, they analyze the result and do the exercises.

In Chart 9, we grouped the most relevant points about theoretical approaches to teaching parallel programming. Therefore, we present in the table only the papers that include some teaching-related strategies. Understanding how the teaching of this type of programming is carried out is essential to define teaching strategies.

The vast majority of the teaching strategies presented by the authors address the presentation of PDC topics, exercises based on code standards, and examples. It is essential to highlight that not all papers address theoretical and practical classes in the courses. For example, some works use only theoretical material (ADAMS; BROWN; SHOOP, 2013; ZARESTKY; BANGERTH, 2014), while others only teach based on laboratory practice (IVICA; RILEY; SHUBERT, 2009; SCHLARB; HUNDT; SCHMIDT, 2015; BALDWIN *et al.*, 2016; GARDNER; B., 2017).

Some theoretical teaching strategies provide all PDC materials accessible to students, describing all exercises and workshops necessary for the learning process (EIJKHOUT, 2016).

Chart 10 presents teaching parallel programming strategies for practical laboratory classes. The vast majority of the papers address the use of OpenMP and MPI for code parallelization (JOINER *et al.*, 2006; MAROWKA, 2008; IVICA; RILEY; SHUBERT, 2009; ARROYO, 2013; SHAFI *et al.*, 2014; BURKHART; GUERRERA; MAFFIA, 2014; SHAMSI; DURRANI; KAFI, 2015; SCHLARB; HUNDT; SCHMIDT, 2015; CUENCA; GIMÉNEZ, 2016; EIJKHOUT, 2016; MOORE; DUNLOP, 2016). For the execution of the programs, the largest of the works had a cluster infrastructure (LIU, 2016; MOORE; DUNLOP, 2016; BALDWIN *et al.*, 2016).

Still, many other works use Proprietary Software (PS) for the practice of parallel programming, so few infrastructure details are shown (JOINER *et al.*, 2006; IVICA; RILEY; SHUBERT, 2009; ARROYO, 2013; ADAMS; BROWN; SHOOP, 2013; FERNER; WILKINSON; HEATH, 2013; FERNER; WILKINSON; HEATH, 2014; BURKHART; GUERRERA; MAFFIA, 2014; UL-ABDIN; SVENSSON, 2015; SCHLARB; HUNDT; SCHMIDT, 2015; BALDWIN *et al.*, 2016; GARDNER; B., 2017; GROSSMAN *et al.*, 2017a). Also, we highlight that the Java language is constantly mentioned in papers (ADAMS; BROWN; SHOOP, 2013; FERNER; WILKINSON; HEATH, 2013; FERNER; WILKINSON; HEATH, 2014; SHAFI *et*

Chart 9 – Papers related to teaching strategies

Paper	Approach	Method
Joiner <i>et al.</i> (2006)	Feedback	Examples
Marowka (2008)		Quiz, examples, and presentation
Li, Guo and Zheng (2008)		Examples
Arroyo (2013)	Pattern Programming	Our Pattern Language
Adams, Brown and Shoop (2013)		
Ferner, Wilkinson and Heath (2013)		
Branco <i>et al.</i> (2013)	Project-Based Learning	Presentation and examples
Shafi <i>et al.</i> (2014)		Presentation and examples
Burkhart, Guerrera and Maffia (2014)		Research journals, and reflective writing
Zarestky and Bangerth (2014)	Flipped Classroom	Feedback, interactive learning, peer discussion, and group discussions
Shamsi, Durrani and Kafi (2015)		
Dolgopolovas <i>et al.</i> (2015)	Model-Centered Learning (Constructivist)	Examples
Ul-Abdin and Svensson (2015)	Analytical (Constructivist)	Lecture, laboratory, small project, and seminar
Nezu (2015)	Problem-based Learning	Exercises
Cuenca and Giménez (2016)		Presentations and examples
Liu (2016)		Research, exercises, and presentations
Eijkhout (2016)	Flipped Classroom	Examples and intensive workshop
Moore and Dunlop (2016)		Lecture, exercise, class discussion, individual help, videos, and quizzes
Grossman <i>et al.</i> (2017a)		Quiz, videos, and homework
Capel, Tomeu and Salguero (2017)	Partially Flipped Classroom	Examples and exercises
	Pattern Programming	

Source: Elaborated by the author.

al., 2014; BURKHART; GUERRERA; MAFFIA, 2014; BALDWIN *et al.*, 2016; GROSSMAN *et al.*, 2017a). However, only a few works use GPU for this type of teaching (ARROYO, 2013; BURKHART; GUERRERA; MAFFIA, 2014; SHAMSI; DURRANI; KAFI, 2015; LIU, 2016).

Although CUDA is not an open-source tool, it is still preferred for teaching due to its low complexity imposed by OpenCL, such as discovering platforms and devices, setting the queues and, compiling the kernel (ADAMS, 2021).

Also, we highlight the use of feedback for theoretical (SHAMSI; DURRANI; KAFI, 2015) and practical (SCHLARB; HUNDT; SCHMIDT, 2015) classes; in the latter, the code verification system informs students if the exercise is correct after the program is submitted. Energy efficiency is a concern when using huge infrastructures (CUENCA; GIMÉNEZ, 2016; UL-ABDIN; SVENSSON, 2015), such as a cluster, for example.

Finally, some investigated papers are concerned that students can run parallel programs from their personal computers through scripts to create the environment (JOINER *et al.*, 2006; IVICA; RILEY; SHUBERT, 2009) or web interfaces for program submission (SCHLARB; HUNDT; SCHMIDT, 2015).

Chart 10 – Papers related to laboratory tools

Paper	Infrastructure	PS	Method
Joiner <i>et al.</i> (2006)	Parallel Virtual Machine (PVM)	X	BCCD, MPICH, and OpenMosix
Marowka (2008)			Visual Studio and C++
Li, Guo and Zheng (2008)			OpenMP and MPI
Ivica, Riley and Shubert (2009)	Virtual Machine (VM)	X	StarHPC, OpenMP, and MPI
Arroyo (2013)		X	PDT, OpenMP, MPI and OpenCL
Adams, Brown and Shoop (2013)		X	Java-based, Paraguin Compiler, and Seeds Framework
Branco <i>et al.</i> (2013)			TinyOS
Ferner, Wilkinson and Heath (2013), Ferner, Wilkinson and Heath (2014)		X	Java and Paraguin compiler
Shafi <i>et al.</i> (2014)			Java Threads, OpenMP, Intel Cilk Plus, and MPJ Express
Burkhart, Guerrero and Maffia (2014)		X	Java, Chapel, OpenMP, MPI, Patus, and CUDA
Shamsi, Durrani and Kafi (2015)			MPI, OpenMP, Hadoop, and GPGPU
Dolgopolas <i>et al.</i> (2015)			OpenMP and MPI
Ul-Abdin and Svensson (2015)	Open Virtual Platform (OVP)	X	aDesigner, GT-Board, and C
Nezu (2015)			C and OpenMP
Schlarb, Hundt and Schmidt (2015)	Web	X	System for AUTomated Code Evaluation (SAUCE)
Cuenca and Giménez (2016)	Simulated System		Cuenca and Giménez (2016) Simulated System OpenMP and MPI
Liu (2016)	UMA, Clusters Beowulf, and GPUs		OpenMP and MPI
Eijkhout (2016)			Microsoft's Visual Studio Task Parallel Library and C#
Moore and Dunlop (2016)	Stampede Supercomputer		MPI
Baldwin <i>et al.</i> (2016)	Hathi Cluster	X	C and MPI
Gardner and B. (2017)		X	Scholar, Thin-line, Rstudio Server, and Jupyterhub
Grossman <i>et al.</i> (2017a)		X	Pilot Library
			HJlibparallel and Habanero Autograder

Source: Elaborated by the author.

2.5 Content of Parallel Programming Teaching

As discussed in [Section 2.4](#) of this chapter, several techniques are used for the practical and theoretical teaching of parallel programming. [Chart 10](#) addresses shared memory, message-passing, and accelerator-oriented Massively-parallel Programming (GPUs), which are the leading managed technologies ([CESAR *et al.*, 2017](#); [ADAMS, 2015](#)).

This work proposes developing a learning tool for parallel programming; thus, in this section, we analyzed the teaching needs for the most used technologies: OpenMP, MPI, and CUDA.

2.5.1 Shared Memory (OpenMP)

In shared-memory systems, several computational cores share the same primary memory, through which communication can occur. Today, students can test and run the programs shown in class with their own desktop and laptops with multicore processors. ([CESAR *et al.*, 2017](#); [ADAMS, 2015](#)).

According to [Cesar *et al.* \(2017\)](#), as OpenMP requires minor modification to a simple program written in C, this may be the way to teach parallelism to students initially. With some directives, it is possible to teach parallelized code in the first lab class. First, however, students need to have C skills and basic concepts of parallel algorithms.

The following example, [Source code 1](#), demonstrates how to parallelize a simple loop using the parallel loop construction. The iterative loop variable is private by default, so it is unnecessary to specify it in a private clause explicitly ([OpenMP Architecture Review Board, 2016](#)).

Source code 1 – Example of a simple OpenMP loop

```
1: void simple(int n, float *a, float *b)
2: {
3:     int i;
4:     #pragma omp parallel for
5:         for (i=1; i<n; i++) /* i is private by default */
6:             b[i] = (a[i] + a[i-1]) / 2.0;
7: }
```

OpenMP classes should teach the parallel programming model of MIMD machines with the shared memory communication paradigm. Through compilation directives (`#pragma omp`), OpenMP allows the programmer to specify where parallelism should be applied and built.

Thus, the beginning of the course addresses the concepts of threads, shared and private variables, and the need for diverse synchronizations, such as those that protect critical regions

and ensure mutual exclusion.

2.5.2 Message-Passing (MPI)

MPI is the most used interface for teaching and developing parallel distributed memory programs (CESAR *et al.*, 2017). The interface is very similar to the standard C library specification, and it allows a program written in MPI to be portable from one implementation to another. MPI also specifies a set of command-line programs used to initiate parallel executions or compile MPI programs (BAYSER; CERQUEIRA, 2017).

MPI can automatically run n copies of a program and allocate them to the compute nodes according to user specifications. Each program is assigned an ID numeric called a “rank”. The instance with rank 0 is often used to perform initial configuration tasks or other coordination tasks by convention (BAYSER; CERQUEIRA, 2017).

Once running, an instance can send a message to any other instance identified by classification or even groups of classifications. For example, the following Source code 2 shows an example of a program written with MPI directives for the C language.

Source code 2 – Example of an MPI code

```
1: // Initialize the MPI environment
2: MPI_Init(NULL, NULL);
3:
4: // Get the number of processes
5: int world_size;
6: MPI_Comm_size(MPI_COMM_WORLD, &world_size);
7:
8: // Get the rank of the process
9: int world_rank;
10: MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
11:
12: // Get the name of the processor
13: char processor_name[MPI_MAX_PROCESSOR_NAME];
14: int name_len;
15: MPI_Get_processor_name(processor_name, &name_len);
16:
17: // Print off a hello world message
18: printf("message: %s from processor: %s, rank: %d out of %d processors\n", "
    Hello World", processor_name, world_rank, world_size);
19:
20: // Finalize the MPI environment.
21: MPI_Finalize();
```

In an MPI course, the concepts of parallel computing of distributed memory, the structure of an MPI program (initialization, communication, number and classification of processes and termination), point-to-point communication (MPI_Send and MPI_Recv), blocking and non-blocking (MPI_Wait and MPI_Test), barrier implementation, and performance (MPI_Time) are discussed.

2.5.3 Accelerator-oriented Massively-parallel Programming (GPUs)

After MPI and OpenMP, it is necessary to introduce programming concepts for GPUs such as OpenACC and CUDA. For this, the student must acquire in-depth knowledge in the use of thousands of threads and memory hierarchy to reach performance increases for this type of computation.

Similar to other parallelization models, a CUDA syntax is inserted within a program written in C. In addition, the structure of a CUDA program allows the host (CPU) and one or more devices (GPUs) to coexist on the computer. Therefore, each CUDA output file can contain a combination of host and device code. Also, it is possible to add device resources and data declarations to any C source file through Particular CUDA keywords (SITSYLITSYN, 2020).

The [Source code 3](#) below shows the C and CUDA implementation, where the main difference is the keyword (`__global__`) that indicates the execution of a particular function on the GPU device.

Source code 3 – Example of a CUDA code

```
1: __global__ void cuda_hello(){
2:     printf("Hello World from GPU!\n");
3: }
4:
5: int main() {
6:     cuda_hello<<<1,1>>>();
7:     return 0;
8: }
```

The GPU programming course covers concepts of thread hierarchies (warp, CTA (Cooperating Thread Array) and grid. 3-dimensional thread identifiers), memory allocation, synchronization, execution, and performance.

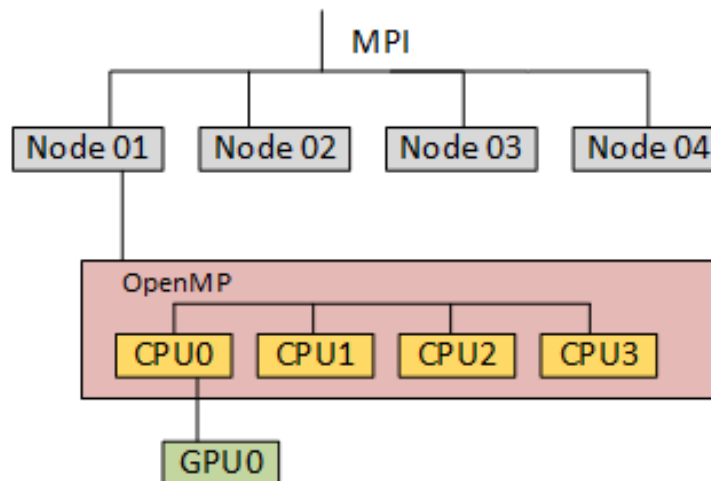
According to [Cesar et al. \(2017\)](#), an entire infrastructure and equipment for practical classes are required for the teaching of parallel programming, especially teaching with GPUs. Furthermore, [Cesar et al. \(2017\)](#), cite in the paper that the Computer Architecture and Operating Systems department at the Universitat Autònoma of Barcelona has received support from computer industry leaders for the design and development of computer labs.

2.5.4 Infrastructure for teaching parallel programming

To teach parallel programming with main tools (OpenMP, MPI in CUDA), it is necessary a cluster infrastructure with various requirements and configurations, which can be exhausting, depending on a technical team to keep it works. [Adams \(2021\)](#) cites that a lecturer must choose an infrastructure based on the student's desired learning outcomes and the budget available to purchase the equipment for PDC teaching.

The basic structure of a regular Linux cluster, for example, has several SMP systems interconnected by a network. Parallelization can be process-based (MPI) and thread-based (OpenMP, required by an SMP machine). On the other hand, MPI can use multiple nodes. [Figure 13](#) shows a hybrid system that uses threads in nodes and MPI processes to communicate between nodes. Furthermore, the GPU can also be accessed by an MPI process and/or OpenMP threads.

Figure 13 – Hybrid schematic distribution with MPI, OpenMP, and CUDA

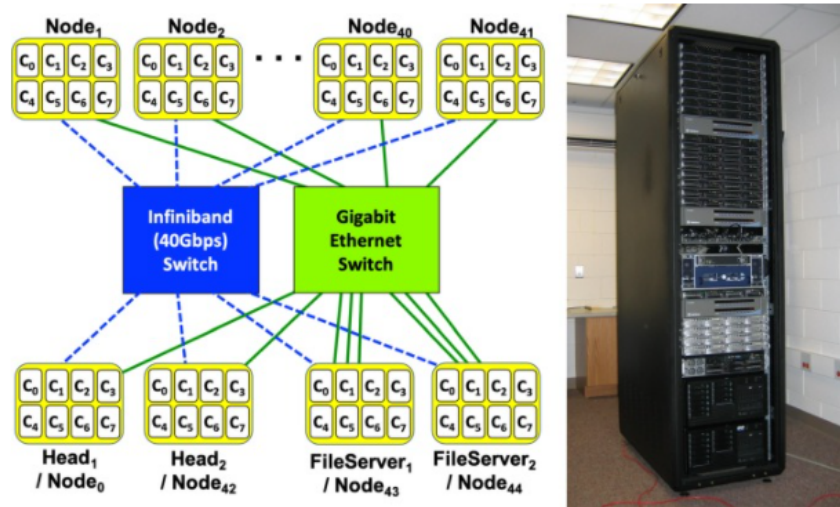


Source: Elaborated by the author.

Configuring a cluster with dedicated servers for teaching parallel computing can be expensive and require specialized operations for construction, maintenance, and support. The initial cost of purchasing the HPC cluster hardware can range from \$150,000 to \$500,000 ([Advanced Clustering Technologies, 2021](#)). In addition, electrical consumption can vary from 60 to 130 MW annually, reaching \$1M annually ([YANG *et al.*, 2013](#)). Another alternative presented in [Figure 14](#) is to build a cluster from conventional computers called Beowulf. However, the educational institution needs to have extra resources for this type of architecture ([ADAMS *et al.*, 2021](#)).

In addition to the cluster, it is also necessary to interconnect the network between computers and create access for students and lecturers. [López and Baydal \(2018\)](#) cite some topics for building a High-Performance service, such as Operating System (OS) installation and configuration, network parameters (Network Address Translation (NAT), Secure Shell

Figure 14 – Cluster HPC Beowulf



Source: [Adams et al. \(2021\)](#).

(SSH), Network Information Service (NIS), Domain Name System (DNS), Network Time Protocol (NTP), and Dynamic Host Configuration Protocol (DHCP), and Network File System (NFS)), disk array management (Distributed Replicated Block Device (DRBD), Redundant Array of Inexpensive Disks (RAID), and Logical Volume Manager (LVM)), and installation and configuration of systems to run parallel applications (CUDA, OpenMP, and MPI).

Remote PDC teaching is another concern of lecturers and students. The Covid-19 pandemic evidenced this type of teaching and induced new challenges to parallel programming about changing the infrastructure access ([ADAMS et al., 2021](#)). Other installations and configurations will be necessary if there is a web system to support students and lecturers. In addition, students will need to be trained to access the system and know commands from the host operating system.

2.6 Final Considerations

Parallel and distributed programming are still challenging in contemporary programming environments ([ACM/IEEE-CS, 2013](#)). However, parallel understanding processing is vital for computer courses, and learning these programming models as soon as possible can increase the students' skills in this area.

The recommendation of the IEEE Technical Committee on Parallel Computing (TCPP) is that **every CS student learns about PDC** ([ADAMS et al., 2021](#)). However, the research presented in the previous sections in [Chart 5](#), [Chart 7](#), and [Chart 8](#) showed that **some educational institutions still do not cover essential areas, do not offer parallel programming, and as far as we know, they do not have the necessary infrastructure** to run these programs.

About the infrastructure, we found many papers that use various solutions based on

proprietary software, which involves risks such as outdated software, high maintenance costs, or project end. Thus, teachers need to consider these risks before adopting these tools in computing courses.

However, some technologies dominate the market and become the standard, such as CUDA, which, although not open-source, makes available for free to students and developers (ADAMS *et al.*, 2021). Thus, lecturers cannot ignore this tool in teaching parallel programming. Other solutions include developing frameworks and libraries, cluster, prototypes, and the use of pre-configured machines. Virtualized solutions, VMs, PVMs, and OVPs, were also found that require setup costs, teacher and student skills, and high consumption of hardware resources.

Therefore, in this chapter, we carried out a detailed survey of the teaching of parallel programming in Brazil and the world. These data allowed us to define the scope of this doctoral thesis. Furthermore, **this research showed a lack of tools for the practical teaching of parallel programming**, an essential part for students to acquire the skills and abilities necessary to work with parallelism.

Most **tools are focused on using clusters** to provide the infrastructure for students and lecturers. However, educational institutions with a lack of resources may not acquire this infrastructure because of the high acquisition and maintenance cost. This way, students will not test their parallel programs. Other tools also try to facilitate access via a web browser; however, internally, the cluster structure persists.

In addition, with this research, we can define some aspects that a tool should have to address some problems highlighted in the papers: avoid using solutions that increase the cost, cluster acquisition or services such as Amazon EC2 (IVICA; RILEY; SHUBERT, 2009; JOINER *et al.*, 2006); provide feedback to the student at the time of submission (SCHLARB; HUNDT; SCHMIDT, 2015); take into account energy efficiency (CUENCA; GIMÉNEZ, 2016; UL-ABDIN; SVENSSON, 2015); allow the student to use their computer (NEZU, 2015); promote coding during classes (LIU, 2016); use of open-source software (FERNER; WILKINSON; HEATH, 2013); optimize response and execution time (CUENCA; GIMÉNEZ, 2016); ease of use of the tool (MOORE; DUNLOP, 2016) and the results visualization (CAPEL; TOMEU; SALGUERO, 2017).

Another critical feature is that tool needs to be lightweight, low-cost, and easy to install. Some courses shown in this chapter are teaching quickly and the tools need to respond fast. One of the technologies that allow this type of response is virtualization, little explored in the papers exposed and included in Chapter 3.

VIRTUALIZATION

Virtualization is an established and prevalent technology in recent years, and its adoption has extended to different areas such as Internet of Things (IoT), cloud environments, and virtualization of network functions. Its growth is due, for example, to increase data center capacity through server virtualization and reducing energy consumption. Among its main benefits are isolation, hardware independence, and scalability, among others ([MORABITO; KJÄLLMAN; KOMU, 2015](#)).

Virtualization development provided flexible and manageable scaling of resources (CPU, memory, filesystem, and networks). An entire environment can be isolated by a virtual machine instance, containing all resources such as an operating system, hardware devices, memory, disk storage, and networking. Also, multiple instances can run on a host; however, this can impact the number of resources needed for the system to support itself ([PAHL, 2015b](#)).

Recently, a new mode of virtualization has become popular in academia and many companies. Called a **container**, this type of virtualization has limited overhead because it works at the operating system level, allowing multiple user-space instances ([TURNBULL, 2014a](#)).

The **container uses the host OS kernel for the virtualization functions having an almost native performance**; therefore, it presents a lower resource overhead than a VM and its virtualization, paravirtualization, or full virtualization methods. For this reason, the container is also called operating system-level virtualization ([DUA; RAJA; KAKADIA, 2014](#)).

There are many differences between these architectures and implementations, for example, differences in security and isolation provided by VMs instead of the near-native performance by containers. Thus, before creating new projects, we need to evaluate these virtualization differences in infrastructure.

This chapter presents the tools and technologies necessary to create the infrastructure proposed in this work. [Section 3.1](#) shows the performance evaluation between containers and virtual machines. [Section 3.2](#) deals with container-based virtualization. [Section 3.3](#) covers the

tools needed for clustering. [Section 3.4](#) shows how volume sharing between containers works, essential for MPI applications. Finally, [Section 3.5](#) summarizes and concludes the chapter.

3.1 Container-based and Hypervisor-based Performance

High-Performance Computing is a term adopted by systems that use resources that require intensive computing. Many **HPC systems are being migrated to virtualization for ease of management and resource allocation**. In addition, virtualization also provides reduced operating costs, high availability, elasticity, and reduced energy consumption ([BESERRA *et al.*, 2015](#)).

Virtualization has an abstraction layer between the host hardware and the virtualized operating system. However, despite the advantages, this technology usually has performance degradation by sharing resources. [Khanghahi and Ravanmehr \(2013\)](#) cite that factors such as storage capacity, availability, usability, data center location, workload, redundancy, latency, bandwidth, buffer capacity, etc., are factors that can affect the virtualization performance of virtual machines and containers.

Performance evaluation techniques are being used in containers to evaluate their effectiveness and efficiency compared to other types of virtualization. Some benchmarks for **containers in HPC environments have shown that the performance evaluation** for this virtualization is almost native to CPU, memory, disk, and network ([XAVIER *et al.*, 2013](#)).

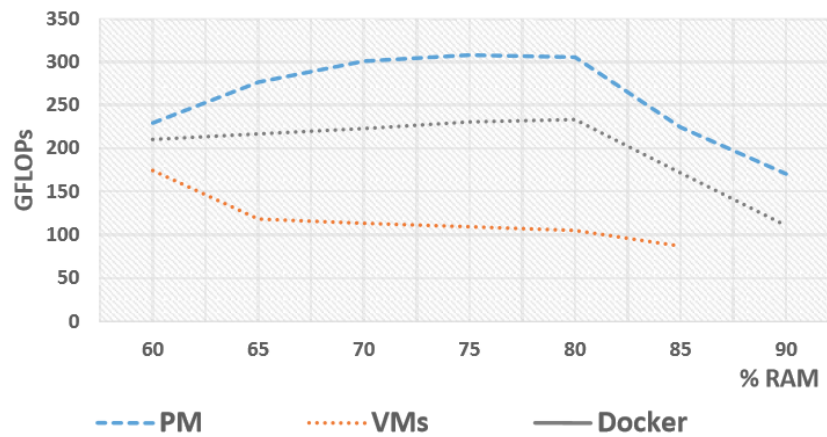
[Beserra *et al.* \(2015\)](#) highlight the importance of evaluating virtualization performance in HPC applications, especially for Message Passing Interface (MPI) systems running in clusters. Thus, another experiment with a benchmark demonstrated that containers perform equal to or better than virtual machines in all cases tested ([FELTER *et al.*, 2015](#)). Furthermore, [Ruiz, Jeanvoine and Nussbaum \(2015\)](#) used a benchmark to demonstrate that container communication is better for virtual devices using MPI processes on HPC systems.

Another research shows the container performance evolution compared to VMs. For example, in [Figure 15](#), VMs perform better with arrays smaller than 65% of RAM, and containers (Docker) perform better with arrays ranging from 75% to 80% of RAM. This result demonstrates an evolution in the benchmark results for processing large amounts of data in containers ([CHUNG *et al.*, 2016](#)).

[Figure 16](#) presents an accurate comparison of memory usage between a virtual machine, container (Docker), where the containers have a performance very close to the physical machine (PM).

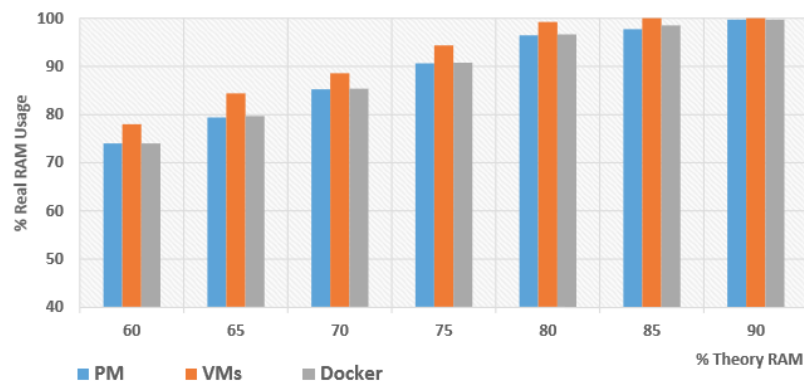
Since the first used benchmarks to evaluate the container performance about conventional virtualization performed by the VMs, there is a distinguished improvement in container performance, which has a superior performance to a VM with some exceptions. Thus, the use of

Figure 15 – HPL benchmark for Docker containers



Source: [Chung *et al.* \(2016\)](#).

Figure 16 – Memory usage in Docker



Source: [Chung *et al.* \(2016\)](#).

containers for HPC applications has become common and very attractive, especially for MPI. [Zhang, Lu and Panda \(2016\)](#) et al. developed a locale-aware HPC cloud MPI library for detecting co-resident containers at runtime.

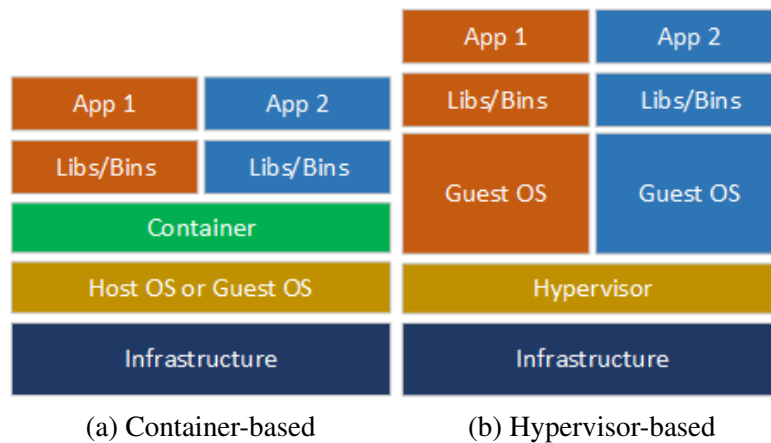
In this work, we chose container-based as the virtualization technology for all the benefits and performance differences already presented, in addition to other advantages such as ease of installation, configuration, and management.

3.2 Container-based

Containers can run applications and systems without the need for a virtual machine instance. This virtualization happens at the operating system level and can isolate multiple systems using just a kernel into one host. This type of virtualization has become popular for its efficiency in running multiple instances on a single OS.

In [Figure 17b](#), the hypervisor-based virtualization has an abstraction layer between the guest operating system and the infrastructure, increasing operation overhead. Containers run directly on the OS, as shown in [Figure 17a](#).

Figure 17 – Comparison between virtualization modes



Source: Adapted from [Jiang and Song \(2015\)](#).

Also, these technologies can work at different problems, even though they are both virtualization techniques. For example, virtual machines perform in IaaS (Infrastructure as a Service)¹, provisioning hardware management and resource allocation. In contrast, containers work in PaaS (Platform as a Service)², allocating and distributing resources to packaged applications and significant capacity to test applications on multiple servers.

Containers are widely used for testing and distributing software, providing all necessary infrastructure like files, network access, and libraries. Multiple instances of containers can run concurrently on a single operating system. In a dispute of containers for resources, the OS is responsible for performing access control to the physical machine, thus creating several isolated systems.

The use of containers supports software distribution facilities for developers to create, develop and distribute applications without worrying about the platform that the software will run. In addition, containers encapsulate applications and their dependencies, thus abstracting environment settings and system dependencies, allowing more time to be spent on other resources such as networks ([MOUAT, 2015](#)).

[Mouat \(2015\)](#) cites some advantages of containers, which would be difficult or impossible with virtual machines, such as sharing host OS resources, accelerating container creation and destruction, and facilitating application portability by avoiding packages for different system

¹ In this model, the service provider offers the entire infrastructure to the customer through virtualization, such as servers, networks, operating systems, and storage.

² The service provider provides all the necessary infrastructure, such as libraries, tools, storage, etc., for consumers to build and test their applications.

configurations. In addition, virtually no overhead, a large number of containers can run on a single host.

Regarding security, [Turnbull \(2014b\)](#) cites that containers were less secure than the complete isolation of hypervisor-based since the containers share the same host operating system. However, lightweight containers have a smaller attack area than the whole operating system required for a VM. Also, regarding virtual machines, the hypervisor layer can be the target to attack.

An alternative to this problem is to create containers inside a virtualized operating system. That way, if an intrusion happens to one of the containers, the attacker will only have access to that VM. Another security alternative is a chroot jail that isolates a directory for a specific process. If there is a security breach in this process, the attacker will be confined in this environment, unable to interfere with other host resources.

Another problem is that containers are less flexible, as they typically run only the same or similar operating system under the host. On the other hand, containers can be created faster than VMs by hypervisors ([TURNBULL, 2014b](#)).

Despite the limitations, containers are being used for various purposes, such as hyper-scale deployments of multi-tenant services³, lightweight sandboxing⁴, and, despite concerns about their security, as process isolation environments.

3.3 Technologies for Containerization

There are many details to check before adopting a containerization technology, especially when building a cluster with containers to run parallel programs. There are several open-source systems for containers, for example, Docker, OpenVZ, LXC, FreeBSD Jails, etc. Although these examples are used to create containers, they are not the same thing.

Linux Containers (LXC) is a tool for OS-Level virtualization, running multiple isolated Linux instances on a single host. Docker previously used LXC to create containers but with additional management commands. Most container tools have equivalent functions, such as memory allocation, CPU, IP addresses, applications, and libraries.

LXC combines several kernel security features to isolate the instance from other running processes at the operating system, such as Mandatory Access Control (MAC)⁵, namespaces⁶,

³ Multi-tenants are architectures that use the same software instance to serve several clients ([CUSATIS; CANNISTA; HAZARD, 2014](#)).

⁴ Sandboxing monitors and limits the program access program to the operating system resources ([AMEIRI; SALAH, 2011](#)).

⁵ Mandatory access control is system-enforced access control to limit access to resources based on authorizations.

⁶ Namespaces are a Linux kernel abstraction feature that makes processes with the same namespace appear to be in an isolated instance of other processes.

and control groups to isolate CPU, file system, memory, and networks (LINUX..., 2016). Container isolation is an essential requirement for running parallel applications. Since a process can interfere with the execution of other processes, such as resource disputes, for example.

HPC environments handle multiple processes or instances of virtualization. In addition, orchestration technology allows the creation and management of container-based clusters. Still, the sharing of resources between the created instances is fundamental for MPI applications. Thus, this section presents some features required in these processes.

3.3.1 Docker

Docker's popularity as a container virtualization technology began with the adoption of large companies such as IBM, Microsoft, and Google. These companies worked together to improve the Docker project. After that, many other companies participated in the project, encouraging the open-source development community (NICKOLOFF, 2016).

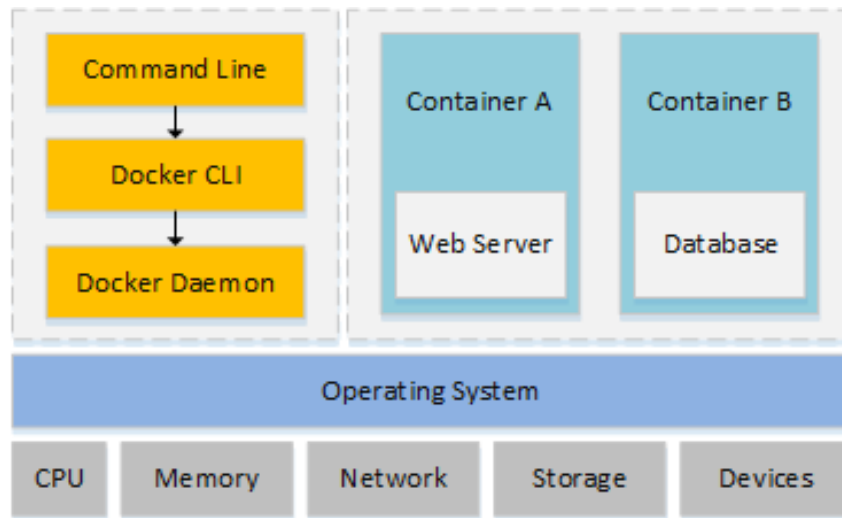
Docker is a command-line program, a background daemon, and a set of remote services that take a logistical approach to solving common software problems and simplifying your experience installing, running, publishing, and removing software. It accomplishes this using a UNIX technology called containers. (NICKOLOFF, 2016, p. 4).

According to Turnbull (2014b), users of the tool can create containers in less than one second; it has easy administration and maintenance of instances quickly since the containers do not need a hypervisor. Below, we have listed the main features of Docker:

- Docker daemon: it is an always running process. Each container instance has a docker daemon child process running in its own userspace and memory.
- Docker CLI: it is a command line interface or an API for accessing the daemon's resources.
- A Docker image: is a system packaged with all the files necessary for a program to run inside a container.
- Registries: it stores public and private images in Docker; and
- Docker containers: they are created from images and can contain one or more services.

Figure 18 shows an example of how Docker works. First, when there is a request to create a container to run a program, the system will check if the request image is already in the registries. Next, Docker will look for the image in an external repository called Docker Hub if the image is not found. Once the image is found in the Docker Hub, the system automatically downloads and registers the image. Afterward, Docker creates the container and starts the requested program.

Figure 18 – Docker infrastructure



Source: Adapted from [Nickoloff \(2016\)](#).

Docker can run on any x64 Kernel Linux host; it stopped using LXC and created its standard library called libcontainer but remained supporting the LXC. Docker isolates file systems, processes, and networks through namespaces, each container having root as owner.

Also, Docker has virtual IP interfaces and, through groups, isolates resources such as memory and CPU. In addition, it has a registration system, collecting information from STDOUT, STDERR, and STDIN⁷. Finally, it also provides a pseudo-tty as an interactive shell.

The containers have all the required software to build a cluster to develop and execute applications, such as network isolation, DHCP and DNS services, and instance management. All these items are necessary because it facilitates the containers creation, users and the execution environment configuration. These programs are referred to as orchestration.

Finally, Docker also has solutions for scalability, such as Docker Swarm, Cloudify, Mesos, and Google Kubernetes, which are open-source orchestrators for containers ([KRATZKE, 2017](#)).

3.3.2 Docker Swarm

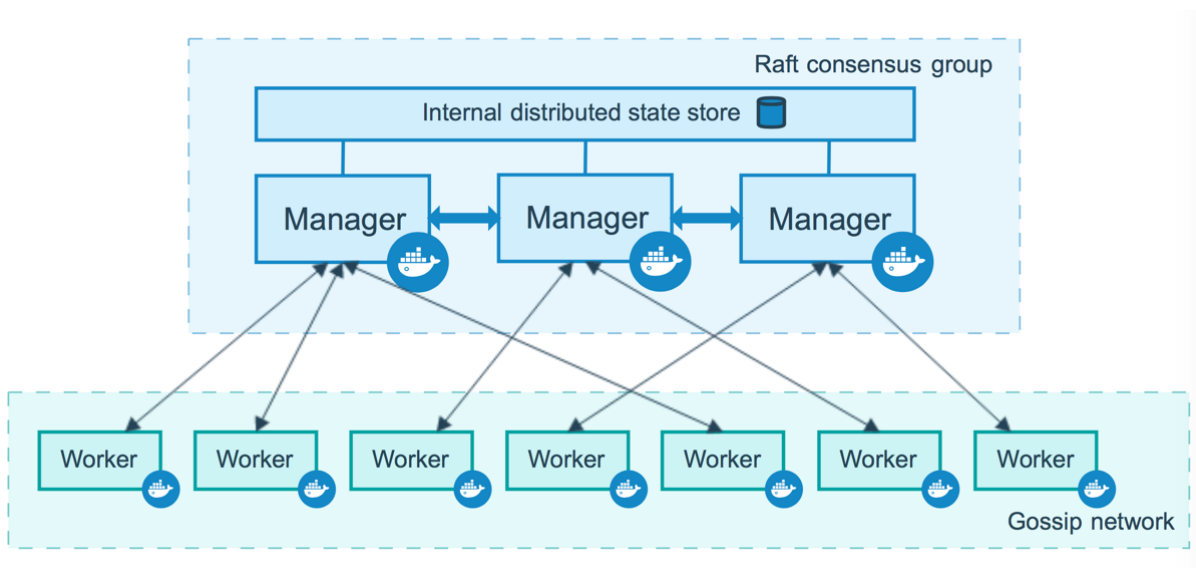
The leading orchestration technologies used in Docker are Kubernetes and Docker Swarm. Kubernetes is an open-source initially developed by Google, which later donated the project to the community. Starting in 2016, Docker created its orchestration tool, called Swarm. The orchestration architecture can incorporate several tools for the use and monitoring of applications under the layer. We also note the similarity with other orchestration tools already conceptualized in hypervisor-based virtualization.

⁷ STDIN (standard input), STDOUT (standard output), and STDERR (standard error) are usual channels between a program and the environment (Linux, a terminal) for sending information.

Although Kubernetes is a complete and highly used tool, we chose to use Docker Swarm for this work because it is a lightweight cluster management tool, already integrated into the Docker Engine, and does not require additional installation, unlike Kubernetes.

Figure 19 shows how swarm is grouped by several Docker hosts that can run as managers (to manage the association of nodes and delegation of services), workers (who perform services), or perform both functions. We can create a service just defining the number of replicas, network, storage, ports that the service exposed. Then, Docker works to maintain the desired state (DOCKER, 2021b).

Figure 19 – Docker Swarm architecture



Source: Docker (2021b).

For example, Docker schedules nodes' tasks that have become unavailable to other nodes that are part of the swarm. Thus, service in Swarm, rather than a standalone container, is a containerized running task. Also, in swarm mode, the system can run standalone containers on any Docker hosts that participate in the cluster.

As previously discussed, Docker offers an orchestration service without the need for additional software through API or Command-line Interface (CLI). Furthermore, the Swarm commands allow the creation and management of containerization behavior. In addition to these points, we can highlight other attributes of this architecture (DOCKER, 2021b):

- Integrated cluster management: orchestrator is part of the Docker package; no additional software is required;
- Decentralized design: it is possible to change the operation of the container at runtime to manager or worker. In addition, we can build an entire swarm from a single image;

- Declarative service model: allows defining the types of applications in the swarm. For example, one application can run a back-end as a database and another as a front-end web server;
- Scheduling: it is possible to define the number of nodes participating in the swarm for each service, which may increase or decrease the number of nodes at run time;
- Reconciliation of the desired state: the manager monitors the swarm state, and if any inconsistencies occurred, it tries to change to the initial desired state;
- Multiple host network: network addresses are automatically assigned to containers through an overlay network;
- Service discovery: each service in the swarm is assigned a unique DNS name;
- Load balancing: internally, the swarm allows distribution services between nodes. In addition, we can provide load balancing through port publishing;
- Secure by default: to protect internal and inter-node communication, swarm uses mutual encryption Transport Layer Security (TLS) and authentication; and
- Continuous updates: offers incremental updates or updates when a service is created.

As seen, Swarm is a lightweight cluster with all the necessary properties for the development of this thesis, which focuses on providing an infrastructure with few resources for practical classes in parallel programming. As analyzed in the survey in [Chapter 2](#) of [Chart 10](#), MPI is the most used application for this type of teaching. Its application involves using multiple processes on nodes in a cluster.

However, we can have several containers on the same machine creating a cluster, significantly reducing the infrastructure needed to run an MPI program for a lab class. Thus, we need to estimate the isolation level between instances, so an MPI process does not interfere with another.

3.3.3 *Constraint Isolation in Container*

The Quality of Service (QoS) is an essential factor for a Service Level Agreement (SLA). Many factors are determinants for virtualization technologies, such as response time, failure rate, resource utilization, among others. Containers are also ruled by these interferences, as external or inter-instance competing for resources.

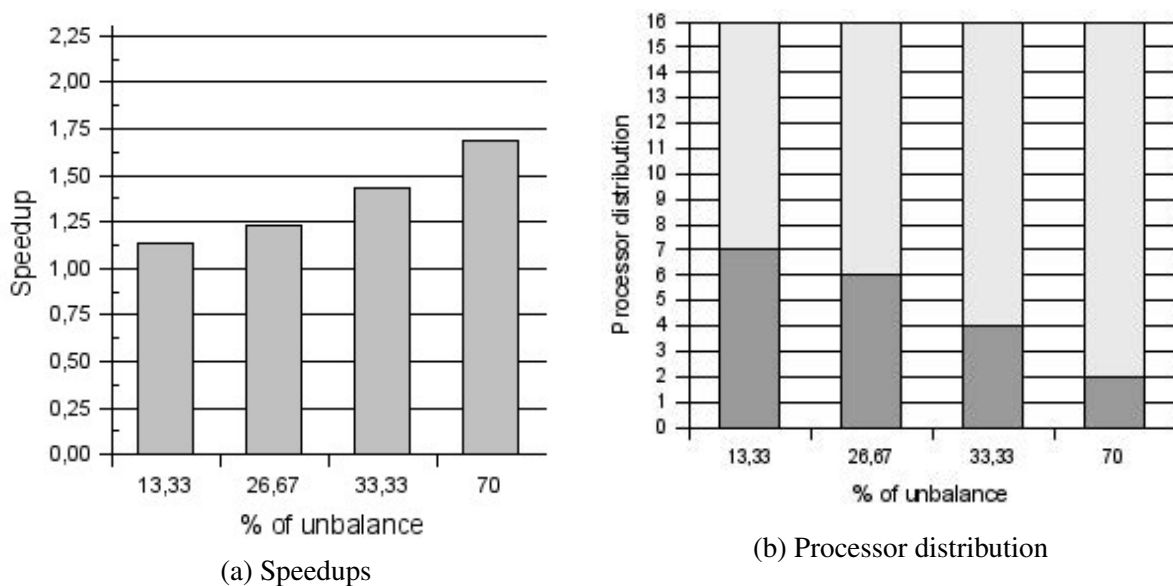
Container isolation is necessary for HPC applications. However, the competition for resources between instances can result in a variable impact in shared environments; once one container gets more resources than another, it can finish processing first. This interference is known as a load imbalance problem, which occurs when nodes that have finished running must wait for the slower node to complete.

[Ma et al. \(2016\)](#) emphasize that an MPI process occupies only one entire container and that interference occurs when applications are running concurrently in other containers on the same physical host. This is a problem for HPC environments. For example, if a student runs the performance evaluation of a particular parallel code versus the sequential code. In that case, the speedup⁸ values may suffer interference if another student is also running some program at that time.

To demonstrate this problem, [Ma et al. \(2016\)](#) emulated a real-world computing environment. Then, they performed a benchmark in containers, generating interference through an MPI library for partitioning unstructured graphs, called ParMETIS ([KARYPIS, 2017](#)).

[Corbalan, Duran and Labarta \(2004\)](#) show in [Figure 20](#) load unbalance scenarios without any load balancing mechanisms for synthetic applications running under containers. They also present in their work a mechanism to balance the load, at runtime, for containers that are executing their applications.

Figure 20 – Results for synthetic imbalances



Source: [Corbalan, Duran and Labarta \(2004\)](#).

The programs run simultaneously by students on a single-host cluster, which is common in a lab class with 40 to 60 students, may have inconsistent results. For example, Swarm delegates a percentage of CPU to each container or distribute the load among them; thus, how the cluster is created will impact speedups.

[Table 2](#) shows a benchmark to assess how the virtualization system can limit the impact of resource disputes among containers running on the same host. For this, the authors stressed

⁸ The speedup is used to measure the performance increase of the relationship existing between parallel and sequential code execution.

one of the instances and performed the measurements. As a result, no CPU impact occurred due to restrictions by cgroup and vCPU.

However, when a particular instance was stressed, the other resources had some impact. The table shows how much influence the well-behaved guests (users who did not have a high consumption of resources) had. Thus, the performance degradation in concurrent instances is evident on a single host, as the kernel needs to respond to instruction calls from stressed guests and well-behaved guests (XAVIER *et al.*, 2013).

Table 2 – Different stress tests

	LXC	OpenVZ	VServer	Xen
CPU Stress	0	0	0	0
Memory	88.2%	89.3%	20.6%	0.9%
Disk Stress	9%	39%	48.8%	0
Network Receiver	2.2%	4.5%	13.6%	0.9%
Network Sender	10.3%	35.4%	8.2%	0.3%

Source: Xavier *et al.* (2013).

Adams (2021) argues that it is not possible to verify the benefits of parallel programming and satisfactorily measure the speedups of MPI programs in a lab class with many students. For example, this happened in a class where the students' MPI applications competed for resources from a single host.

Docker provides ways to control the amount of memory, CPU, or Input/Output (I/O) block a container can use about this problem (DOCKER, 2017). Thus, to avoid the execution of a student interfering with another student's measurement, the container has mechanisms to limit the number of resources used by each container.

Also, we can use queuing systems for an academic class to guarantee exclusive access to the resource at code execution time. In this way, students could submit their work without concurrency and evaluate the performance of parallel programs about sequential programs.

3.4 Volume Sharing in Containers

Some applications require data persistence for information sharing. However, containers do not keep file system data after the state is stopped, unlike virtual machines. For MPI applications, this is a problem as we need to share information between processes. (NATH, 2020).

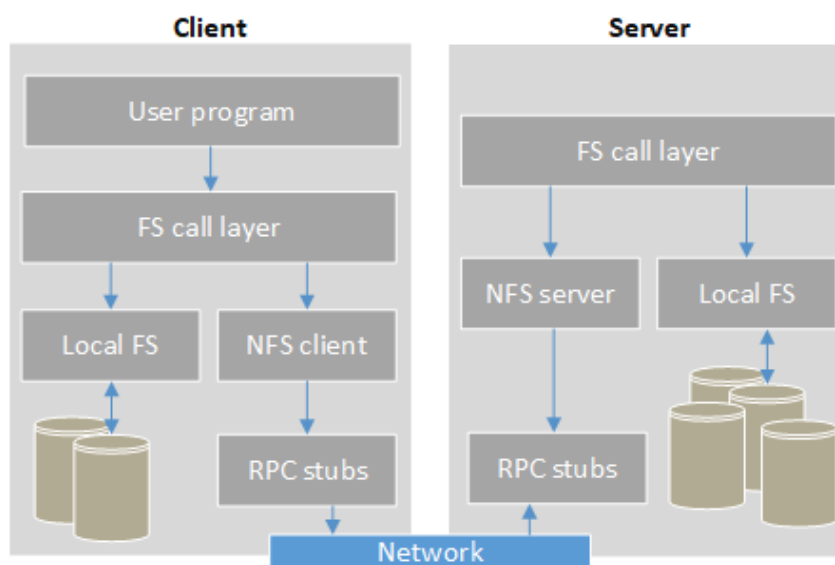
Thus, two technologies are used in the Swarm cluster for persistence and data sharing. Network File System is generally used to share file systems between different hosts. On the

other hand, Docker Volumes, integrated into the Docker package, allows containers to share a directory with the host system or between instances.

3.4.1 Network File System

NFS uses a client/server architecture to share data between different hosts or even different containers, on the same host or not. [Figure 21](#) shows this infrastructure, where the server contains the local file system shared with clients. Clients, through Remote Procedure Call (RPC) stubs, point the remote file system to a local file system.

Figure 21 – Volume sharing with NFS



Source: Adapted from [Sterling, Anderson and Brodowicz \(2018\)](#).

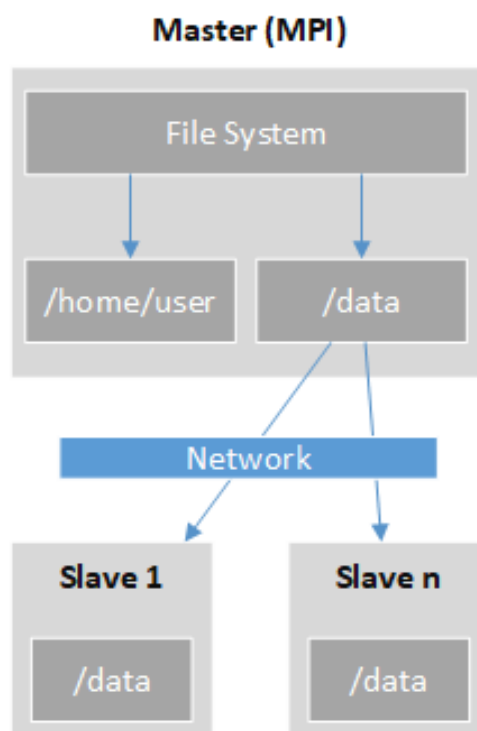
NFS is specified by Request for Comments (RFC) 7530. The FS call layer provides a layer for accessing the local file system. Read and write operations are sent to the NFS Kernel, which interprets and changes these operations into NFS procedures (CREATE, READ, REMOVE, ACCESS, among others). The client uses RPC to execute functions on the remote file system ([JONES, 2019](#)).

This file-sharing system is one of the oldest and most widespread in computing and is also used to share resources with HPC systems. [Figure 22](#) shows the use of NFS on systems with MPI to transfer data between the various nodes involved in the cluster.

The system needs in-depth knowledge of Linux from students and lectures. To install NFS, they need networking skills, disk administration, firewall, and file permissions. In addition, both client and server need to be suitably installed and configured ([UBUNTU, 2019](#)).

Also, NFS operates in two modes. First, the server needs to write all data in the storage and respond to the client in synchronous mode. Second, in the asynchronous mode, the server

Figure 22 – Volume sharing and MPI



Source: Elaborated by the author.

notifies the client before the write data in disk; in this mode, the performance is better, but the reliability decreases since any problem on the server can cause data loss.

3.4.2 Docker Volumes

Docker Volumes (DV) are used to persist data generated by containers or share directories between the host operating system's containers. All share management is handled by Docker, abstracting the infrastructure complexity for students and lecturers (DOCKER, 2019).

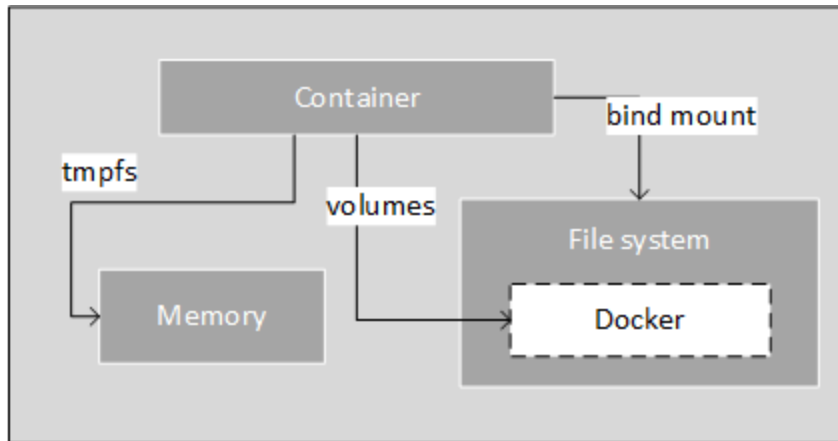
As shown in Figure 23, volumes are stored in disk space on the file system not to use the disk space inside the container. Also, we can use three methods to share volumes. The first is called bind mount, a directory on the host operating system linked into the container.

In the second mode, called volumes, Docker manages the volume created between the file system and the container. Finally, the third mode, called tmpfs, is a temporary space in the host's RAM (DOCKER, 2019).

Different systems require different levels of file system sharing consistency. However, there are cases where temporary consistency is acceptable to achieve better performance. Thus, just as NFS has two synchronous and asynchronous modes, Docker Volumes also have three modes: consistent, cached, and delegated.

Consistent mode, similar to synchronous NFS mode, the container and the host maintain

Figure 23 – Volume sharing with Docker



Source: [Docker \(2019\)](#).

the same view as the file system. However, in cached mode, host views are official, and some container update delays may occur. Finally, delegated mode, container views are official, and some host's File System (FS) update delays may occur.

3.4.3 DV and NFS Performance Evaluation

Considering the proposal of this work to develop a light and low-cost cluster infrastructure, evaluating the performance differences between the volume sharing modes is fundamental for MPI applications, which depend on this type of service. Thus, we planned three experiments with a virtual machine instead of a physical machine. We made this choice based on the expectation that students and lecturers will install the system on a VM, one of the worst performance scenarios.

We performed all benchmarks in a virtual machine with the minimum necessary resources containing 1vCPU, 1GB RAM, and 10GB of disk storage. The physical device has an Intel Core i7 processor (3537U) and 8GB of RAM. In addition, we used two storage disks, Hard Disk Drive (HDD) and Solid State Drive (SSD), with different technologies.

We chose Sysbench for testing as it is the most used benchmark system. [Chart 11](#) shows all tested factors and levels. For all tested modes (A = read, write, concurrency, no concurrency, SSD, and HDD), we run the benchmark 30 times for each factor and level to calculate significance. For example, we used one container for no concurrency tests and ten containers for concurrency tests. As a result, we collected 66,000 events after running the experiments that are available online ([BACHIEGA, 2021](#)).

For the database, we used MySQL for its popularity and easy installation. Sysbench supports this database and evaluates the read and write of queries and the number of transactions performed. We used two file sizes for the FS read and write tests (100MB and 2GB). Finally, we

used an MPI program to apply a filter for small images (SI) and large magnets (LI).

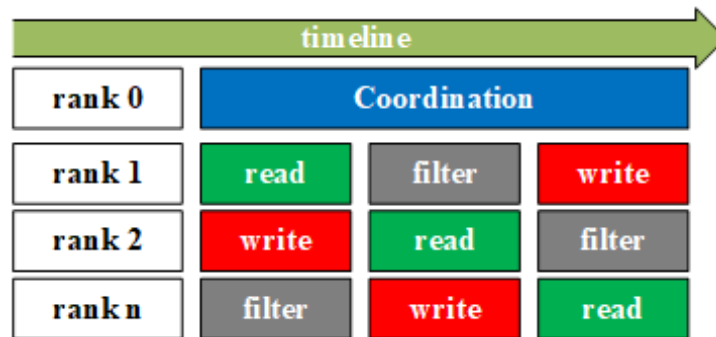
Chart 11 – Different stress tests

Technology	Sharing mode	MySQL	100M	2G	SI	LI
NFS vs Docker	Sync vs Consistent	A	A	A	A	A
NFS vs Docker	Sync vs Cached	A	A	A	A	A
NFS vs Docker	Sync vs Delegated	A	A	A	A	A
NFS vs NFS	Sync vs Async	A	A	A	A	A
NFS vs Docker	ASync vs Consistent	A	A	A	A	A
NFS vs Docker	ASync vs Cached	A	A	A	A	A
NFS vs Docker	ASync vs Delegated	A	A	A	A	A
Docker vs Docker	Consistent vs Cached	A	A	A	A	A
Docker vs Docker	Cached vs Delegated	A	A	A	A	A
Docker vs Docker	Delegated vs Consistent	A	A	A	A	A

Source: Elaborated by the author.

In [Figure 24](#), we illustrated how the algorithm for applying an image filter works. First, the program divides the workload between the nodes, providing resource concurrency. Thus, while one process reads the image, another process applies the filter to the image.

Figure 24 – Benchmark for the image filter



Source: Elaborated by the author.

The tables in [Appendix A](#) present all experiments according to their significance. We used the t-test paired with a normal distribution to compare over 480 test combinations. [Chart A.1](#) presents read (R) and write (W) results for SSD disks, and [Chart A.2](#) presents the same results for HDD disks.

As a result of this experiment, we observed that most tests are significant for comparisons between NFS and Docker volume sharing modes. However, for the comparisons between the Docker methods (consistent, cached, and delegated), the difference in the result was slight, even when significant. About NFS, asynchronous mode showed the best results for both hard disks.

A better understanding of these sharing and performance issues helped define the requirements and configurations necessary for the development of this work. Thus, in this first version

of the system, we will adopt Docker consistent mode for sharing between container and host.

For the volume sharing between separate physical nodes, we will use asynchronous NFS because of its performance benefit. Despite increasing the risk of data loss, in teaching programming, if a physical host crashes, Swarm will remove this host from the cluster. On the subsequent submission of the program by the student, the processes will redirect to others nodes.

3.5 Final Considerations

In this chapter, we showed the importance of virtualization for the Iguana's development. With virtualization, we can quickly provision the infrastructure on-demand and change the horizontal dimensioning. Also, it makes it possible to create a cluster with low resources, minimizing the time of installation, configuration, and complexity involved in this process.

As seen, there are many virtualization technologies, both hypervisor- and container-based. **With containerization, we can create a lightweight cluster infrastructure to run parallel programs in practical classes or by students on their personal computers.**

Docker was chosen for its simplicity of installation among containerization technologies, a complete access API (necessary for the front-end of this work), and having Swarm integrated with the Docker Engine, removing all the complexity of managing the cluster. These items made it possible to develop an application to run parallel code without much complexity.

Other technologies with superior performance than Docker, such as Singularity and Shifter, are used in the HPC environment. However, we did not use them for technical reasons; for example, Singularity can make container portability difficult because it is not compatible with all Linux kernel versions, more complex installation, and various packages than Docker, for example.

According to [Rudyy *et al.* \(2019\)](#), Docker underperforms Singularity and Shifter when there is a large amount of collective communication regarding MPI processes; due to its virtual network's architecture.

However, this work does not propose the maximum performance inherent to real applications but rather a lightweight cluster for teaching parallel programming in the laboratory. As such, we directed this study of Singularity integration into future work to increase the tool's performance. Furthermore, the installation of Singularity is still a challenge by its complexity; this work tries precisely to simplify this process.

Finally, these studies guided us to define the necessary configurations and parameters for creating the proposed work, such as selecting Docker and NFS volume sharing methods and the use of queues to minimize the effects of constraint isolation between containers.

IGUANA CLUSTER SYSTEM

This chapter describes the project, development, implementation, and evaluation of the support tool for parallel programming (called Iguana Cluster System). Its primary function is to provide a low-cost structure for teaching parallel programming.

The construction of software follows a set of steps that includes methods and tools, which keep the layers of technologies cohesive, enabling the development of the software product effectively and efficiently ([PRESSMAN](#); [MAXIM, 2016](#)).

Thus, for this system development, we have separated this chapter into five sections. [Section 4.1](#) shows requirements analysis are covered. In [Section 4.2](#), we describe the system design. [Section 4.3](#) shows the implementation of the system. [Section 4.4](#) describes testing and deployment show the evolution of the system. Finally, in [Section 4.5](#), we present the summary of that chapter.

4.1 Requirement Analysis

The system's main objective is to provide a low-cost infrastructure, with minimal hardware configurations (e.g., a single virtual machine with 1 CPU, 1GB RAM, and 8GB disk space), without the need for specialized labor or involving any costs for the end-user of the tool, students, and lecturers.

We defined the prerequisites based on related studies and proposals for tools from other authors to run parallel codes. Then, through research in [Chapter 2](#) and [Chapter 3](#), we determined the system main functional and non-functional requirements based on theoretical teaching and practice in parallel programming.

[Chart 12](#) presents the summarized description of the system functional requirements. The main features that the tool must have are an authentication system, student registration (create, delete and update), exercise registration (create, delete, change, correct, feedback, score, and

code download), code editor, and compilation and submission online of exercises.

Chart 12 – Summary of system use cases

User case	Name	Short description
UC-01	Login/Logout/Recover Password	System authentication for students and lecturers
UC-02	Add/Edit/Delete User	It allows the lecturer to add, update and delete any registered user
UC-03	Add/Edit/Delete Group	It allows the lecturer to add, update and delete any group
UC-04	Add/Edit/Delete Exercise	It allows the lecturer to add, update and delete any exercise
UC-05	Add Answer to Exercise	It allows students and lecturers to send an answer to an exercise
UC-06	Add Score for Student Response	Allows the lecturer to grade a student
UC-07	Add Exercise Feedback	Allows the lecturer to give feedback to a student
UC-08	Edit/Compile/Run Code	Allows the editing, compilation, and execution of code by any user of the system
UC-09	Download Student Codes	Enables the lecturer to download the codes sent by the students
UC-10	Download Code Reports	It enables the lecturer to download the code reports containing the number of compilation attempts to arrive at the correct result, grade, and feedback
UC-11	Update Settings	Allows the lecturer (admin) to change the default system settings (public host interface, back-end service port, front-end service port, cluster name, cluster password, operation mode, number of virtual nodes, the maximum number of nodes, host user, host password, host self-registration, code execution timeout, queuing system, maximum queue concurrency, debug messages, error messages and SMTP service settings for sending messages by email)
UC-12	Setting the Operation Mode	It allows the admin to change the system's operating mode (master, node, local)
UC-13	Connect to an Available Cluster	It allows the admin to connect to an available network cluster
UC-14	Add/Delete Virtual Nodes in Cluster	It allows the admin to add or remove virtual nodes in the system to execute MPI codes
UC-15	Check the Queue	It allows to monitor the system queue and see what tasks students are compiling/running
UC-16	Check Users Online	It makes it possible to see online users

Source: Elaborated by the author.

Table 3 provides a brief description of the non-functional requirements. For example, the system must allow students to use it on their computers without an Internet connection and no required configurations. Also, the tool should allow the use of multiple compilers, use lightweight

resources, perform an automatic integration with other nodes, and be available online.

Table 3 – Non-functional requirements

ID	Name	Short description
NF-01	User-friendly Graphical Interface	Provide a friendly interface for students and lecturers
NF-02	Authentication	Provide authentication through changes of cryptographic keys
NF-03	No Cluster Infrastructure	The system should allow the execution of MPI codes without the cluster infrastructure
NF-04	No Previous Configurations	The tool should automatically recognize the network settings, defining its back-end and front-end ports, as well as the Ips
NF-05	Multiple Compilers	It should support multiple compilers and allow the inclusion of new ones

Source: Elaborated by the author.

It is essential to highlight that, as this system was developed as part of a doctoral thesis by only one student, the software validation occurred in meetings with his advisor, who also teaches parallel programming disciplines.

The validation with the advisor considered how practical parallel programming classes are taught in different paradigms, such as MIMD machines with shared memory, MIMD machines with distributed memory, and heterogeneous platforms. The validation also considered practical aspects of code development by students and groups of students (intra-class and extra-class), corrective aspects of submitted exercises, and immediate feedback. These and other factors were considered in the validation of the Iguana project.

In addition, we also tested the tool with students (not experiment) in the first half of 2020 to obtain feedback on the design, navigability, and possible changes.

4.2 System Design

We designed a structure to popularize the teaching of parallel programming in environments with limited resources, often without cluster availability and without quality access to the Internet. In addition, this structure provides all the support to lecturers, and students can build and execute their parallelized codes.

Also, the tool allows users with little experience in computing to run their parallel code, like first-year students of a computing course, through virtual machines, using the installation

script on Linux or the website. The system is designed to work even on VMs with 1GB RAM and 8GB of disk space.

4.2.1 Architecture

Iguana is a modular tool. Therefore, we built it in independent parts coupled together to form the structure shown in [Figure 25](#). The front-end is the web interface accessed by students and lecturers that perform all function calls for the back-end. Lecturers and students logged into the system have a different view of the front-end. For the user interface (the student) is only allowed access to their data and the exercises. For the admin (lecturers), all functionalities are delivered.

The admin can create exercises, groups, and other users, basic functions for a practical lab class. Also, the admin can change the method the system works:

- **Localhost:** used when the user has only one physical or virtual machine to run the tool. In this case, to run programs in MPI, Iguana will create containers to simulate nodes in a cluster. The student views these containers in the tool as we do, with their respective Internet Protocol (IP) numbers.
- **Local network:** when the tool is installed on two or more physical or virtual hosts, Iguana (in cluster mode) automatically identifies these computers and groups them together.
- **Remote cluster:** the system can interconnect with remote clusters and send processes to run on those nodes. This item is listed for future work.
- **Cloud:** it allows the system to use the cloud, such as Amazon EC2, Google, or others, to integrate the cluster infrastructure. This item is listed for future work.

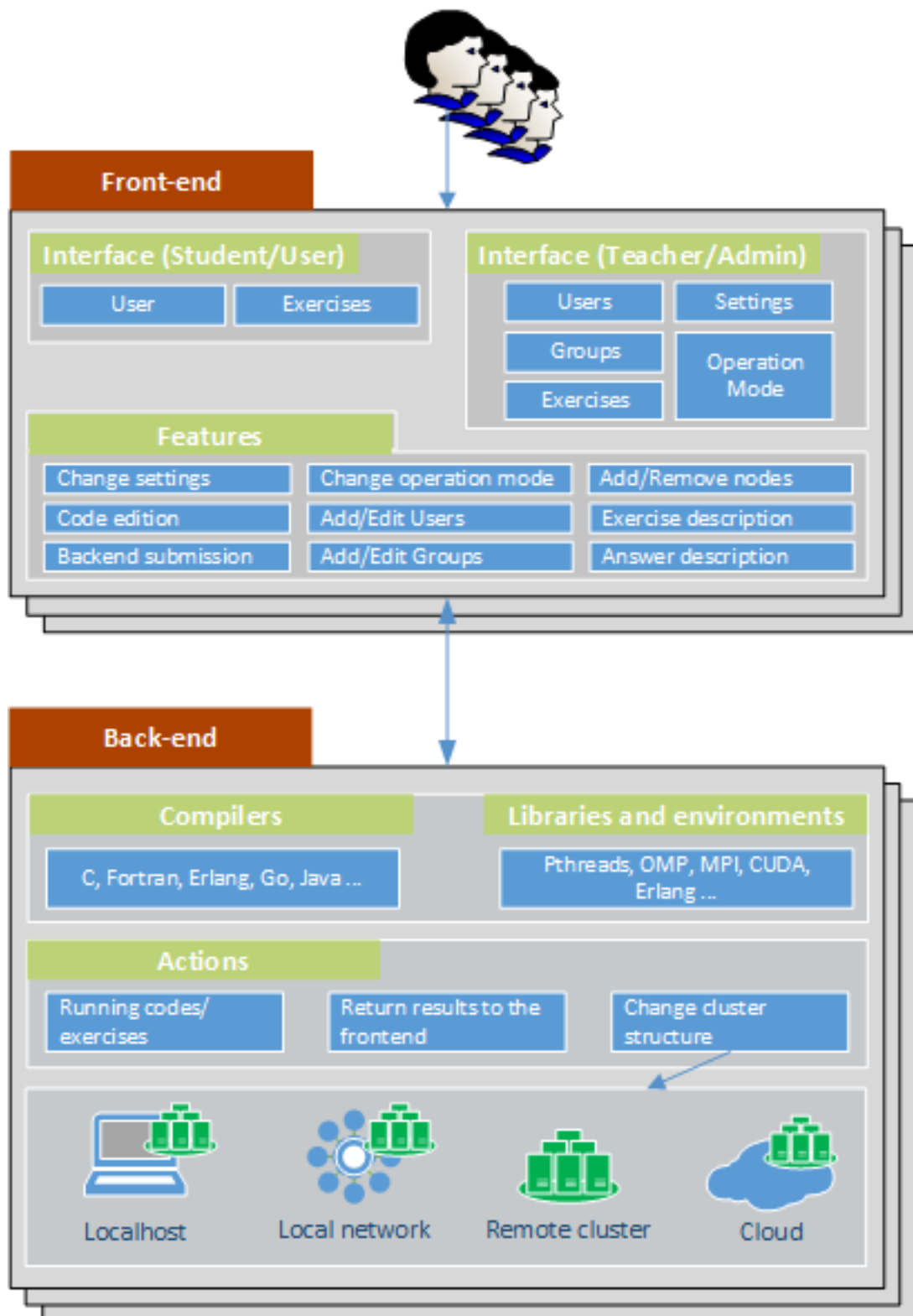
Students will perform the same roles as the admin on the front-end, as long their host is not in a network dominated by a back-end/lecturer. The tool allows this because students must use the infrastructure at home to practice and run exercises. In addition, they will also have an interface to create and test algorithms on a virtual cluster.

With Iguana, students can turn a simple notebook or VM into a cluster to run their experiments with many nodes for their parallel code. Furthermore, they can create networks between one or more VMs, for example, to extend the number of nodes or choose a cloud service.

4.2.2 Platforms

Initially, the tool will be provided to the lecturer through a pre-configured virtual machine image or by downloading an installation script:

Figure 25 – Iguana infrastructure



Source: Elaborated by the author.

- **Pre-configured image:** in this option, available online, the lecturer or student can run the VM and define the operating mode (virtual cluster, local network cluster, remote cluster, or cloud); or
- **Installation script:** the lecturer or student will load an installation script and run it on their already installed operating system. Initially, the script ([Appendix F](#)) supports Linux-based environments (Ubuntu and Debian).

In the case of the preconfigured VM, when it is executed in a virtual machine with minimal configurations, after its initialization, it will inform the user which the number of IPs enabled for access via any web browser. For installation via a script, the user must run it on their operating system.

The base operating system for the construction of this project was Ubuntu 21.04 Linux kernel 5.4, which may be extended in the future to other systems based on Linux, such as Red Hat, Suse, among others. In addition, Iguana is a lightweight infrastructure cluster, meaning the student can run the tool on a Linux-based virtual machine inside a Microsoft Windows host operating system.

4.2.3 Programming

Angular is the language used to create the Iguana front-end. It is an open-source language developed by Google to create dynamic web applications (TypeScript). It is used by more than 44.3% of software engineers to build user interfaces. It is a powerful language, and it is on the increase ([STACKOVERFLOW...](#), 2021).

The language offers excellent documentation; the number of people who use it is supported by a large company and allows modular implementation. Also, the language works with Web Services (WS), which is the communication designed for Iguana's back-end.

Another critical point in adopting Angular is that the application is rendered on the client-side, unlike other web systems, such as Hypertext Preprocessor (PHP). In PHP, the server is responsible for processing and returning the HyperText Markup Language (HTML) page to the client's browser ([ANGULAR](#), 2021).

To Angular, all processing is done by the client's browser. As a result, the back-end can use these extra resources to run MPI programs. Communication is done through a RESTful API that only requests data to the back-end, which it returns encapsulated in JSON structures ([Source code 4](#)) to the front-end. Angular then processes this data and displays the information in HTML pages.

Source code 4 – Sample data returned with JSON structure

```
1: {"Students": [
```

```

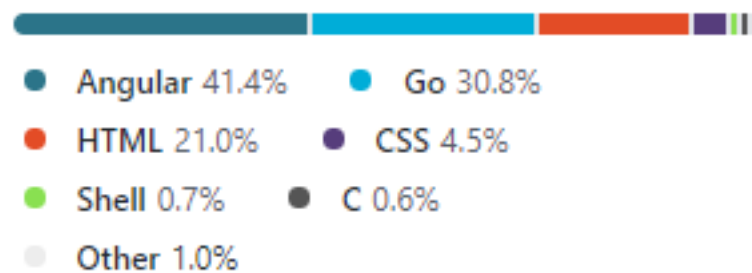
2:    { "name":"Naylor Bachiega","email":"naylor@usp.br"},
3:    { "name":"Paulo Sérgio","email":"pssouza@icmc.usp.br"},
4:  ]}

```

For the back-end, we chose the open-source language called Golang, also developed by Google. In addition to being a high-performance language, Golang is also used for building the Docker Engine and Swarm. Furthermore, the language offers a complete library of access to container management functions by Docker and cluster management by Swarm, essential to create the infrastructure proposed in this work (DOCKER, 2021a).

Iguana is also open source and available for download from GitHub (<https://github.com/iguana-hpc-usp/ICS>). Figure 26 summarizes the division of languages used in the tool, where 41% corresponds to the front-end with 10091 lines and 401 functions and the back-end with 30.8% corresponding to 5669 lines and 187 functions created directly and/or indirectly by the developer (GITHUB, 2021).

Figure 26 – Languages used in the tool development



Source: Adapted from GitHub (2021).

Many lines and functions indicate an extensive system developed in these years as a doctoral project. As a result, the Iguana has many internal and design revisions before the last version used in the experiments. However, as a modular tool, this author highlights the easy modification and customization.

4.2.4 Communications

Iguana was designed to work in four modules to allow different operation modes: local-host, local network cluster, remote cluster, and cloud. Figure 27 shows the operation of Iguana on only one physical machine or VM (localhost). In this case, to run algorithms in MPI, the user can create several virtual nodes on a Docker Volume to share the data and binaries inherent to MPI.

By default, the Iguana engine detects the number of cores on a host. For example, if a host contains four cores, Iguana starts the master container on one vCPU and another three

containers through swarm to the other three remaining vCPUs. This way, the student can test programs in MPI with four nodes. For speedup, for example, the student can run the sequential algorithm on one node and then the parallel algorithm on the others.

However, Iguana is flexible enough to define, for example, that each node can be grouped by two or more vCPUs, in the case of hybrid algorithms with OpenMP. In addition, it is possible to have another configuration model where nodes use processing available at request time, as shown on the right in [Figure 28](#).

The Iguana engine (created with the Go language) controls the entire back-end, receiving requests from the front-end through Web Services (WS) and sending requests to the other layers. For example, when the system starts, the tool sends a request to Docker to start the master container, responsible for executing the program in MPI with the other nodes.

Afterward, the engine sends a request to Swarm to create the cluster according to the default configuration or user customization. Swarm receives the order with the number of nodes needed and makes the dimensioning and management of the requests and creating the volume necessary to interconnect the nodes. Iguana also allows changing the number of nodes in the system at runtime.

Finally, when the user starts the system through the web interface in the browser, the files will be transferred and rendered by Angular. Later, all communication is carried out by web services.

[Figure 28](#) shows the operation through a computer network. It is in this part that Iguana shows its potential for self-configuration. Once in a network, if the tool is in master mode, it sends a message via broadcast through default port 10001.

If there are other machines on the network in node mode, those machines will automatically connect to Iguana. The tool also allows the creation of several clusters in the same network. In this case, the admin can provide a name and password for a particular cluster. Only nodes that know these credentials can be members of the cluster.

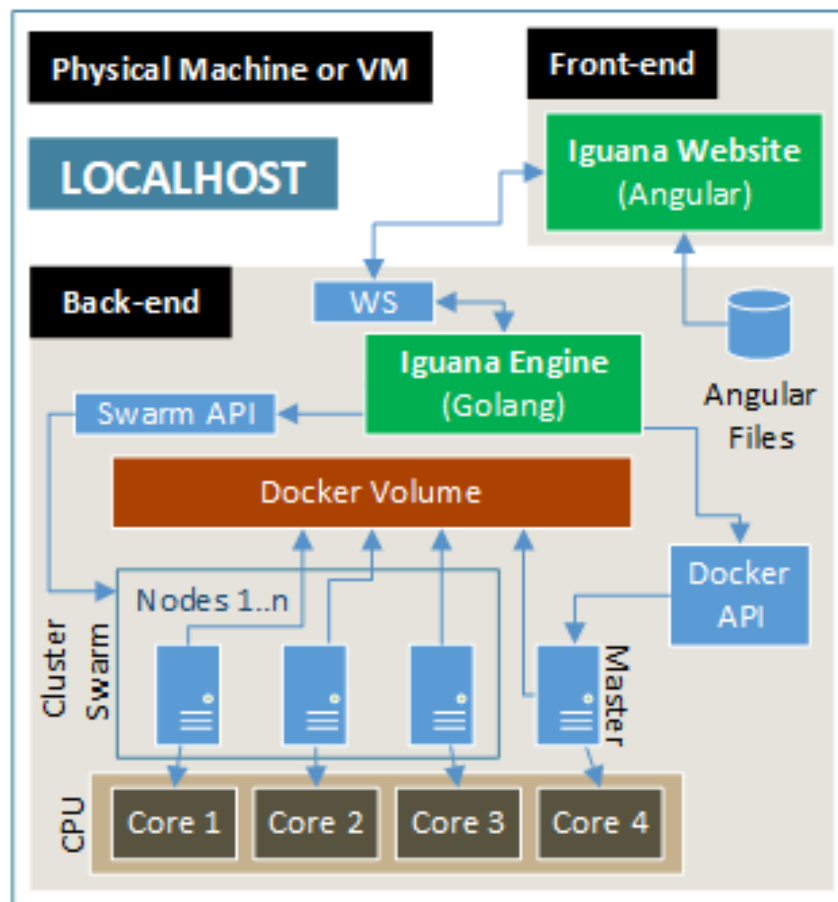
We can have a laboratory with 20 machines, install Iguana, and define one of the machines as a master and the others as a node. Iguana will automatically do all the configuration, insert the machines in clusters and make the necessary shares to execute MPI codes.

Internally, on the host, sharing between containers is performed by Docker Volumes in consistent mode. We have pointed the volume to a logical link on disk, a directory path, and then that path is shared between hosts by asynchronous NFS.

4.2.5 Security

Additional security measures have been adopted, in addition to those provided by Angular and Golang. The front-end remote procedure calls to the back-end were all encrypted, regardless

Figure 27 – Localhost architecture



Source: Elaborated by the author.

of whether the user contracted the Secure Sockets Layer (SSL) service. Many public education companies do not have funds for this type of service.

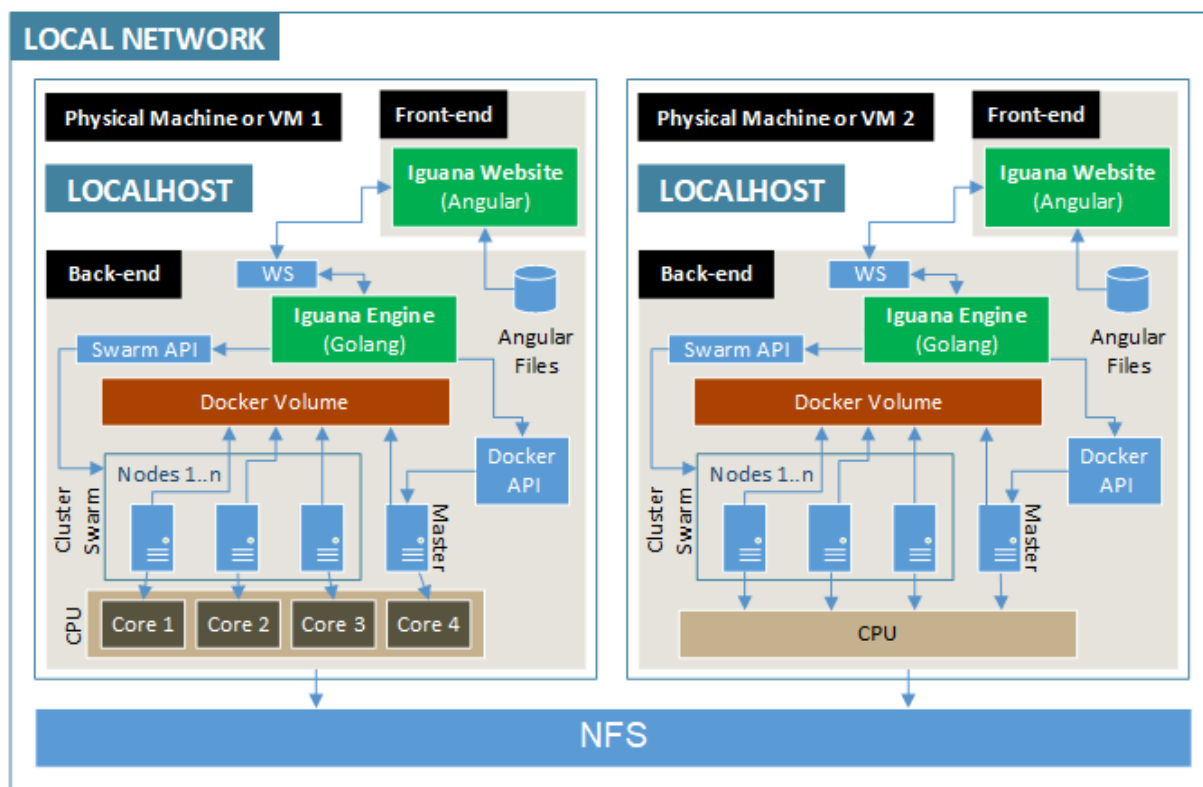
The cryptographic key changes every time the user enters the system. Also, the back-end only receives registered requests from registered users. Therefore, any strange requests are discarded. There are also other levels of security in file system sharing by Docker and NFS. In this case, these mechanisms were sets to allow only the registered host on the network to access the files.

4.3 Implementation

This section only shows the main Iguana screens due to the number of screens in the system. Figure 29 is the tool's main screen, right after the user enters with the correct username and password. On the left panel, the menus appear according to the access profile. In the "My Panel" menu, the admin can access users, groups, exercises, system settings and try a code, which shows the screen on the right.

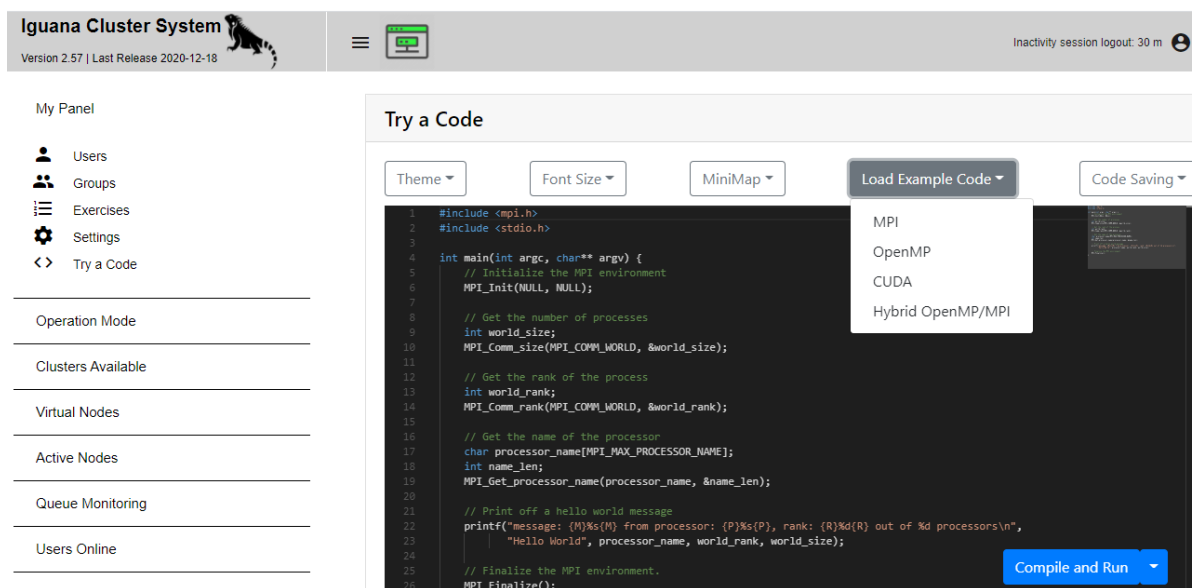
Also, in the left menu, the admin has the options Operation Mode (change the mode

Figure 28 – Local network architecture



Source: Elaborated by the author.

Figure 29 – The Iguana's main screen



Source: Elaborated by the author.

to localhost, local network, remote cluster, and cloud), Clusters Available (shows the clusters already in the local network and allows connecting to one of them), Virtual Nodes (increase or decrease the number of nodes in the cluster), Queue Monitoring (shows which students have code in three states: queued, compiling and running), and Users Online (shows which students are logged into Iguana).

On the right, we have the Monaco code editor (using the VS Code language). It allows the student to write the code directly into the browser with various features of a desktop editor such as color functions, line numbering, auto-complete, search, minimap, theme choice (light or dark), font size, auto-save, sample codes, and C language command palette.

Finally, in the lower right corner is the Compile and Run button. Immediately after the student writes the code and clicks on the button, the system will send the program to the queue. If no other students are in dispute, the program is compiled and run in the cluster, returning the result immediately to the screen. The button is floating, so if the code has many lines, the button will follow the screen, always being available.

Figure 30 shows the customization of parameters for code compilation and running. Iguana automatically recognizes the type of the student's code by the program header (`mpi.h` or `omp.h`) and makes the necessary adjustments to compile and run without human intervention. However, if essential, it is possible to change all parameters manually, adding arguments, replacing compile and run commands, uploading files to use in the running process, adding or removing nodes, view results, and special tags.

Figure 31 shows the nodes' selection for running MPI programs. The student can select one or more specific nodes to run his program or not pick any to includes all nodes. Furthermore, it is also possible to define the number of processes.

Figure 32 shows the result using special tags. In this type of view, the system automatically groups the ranks by the node. Thus, the student can identify and better visualize the distribution of processes after parallel code execution.

As presented in Source code 5, the tags are included in the code prints written by the student. Three labels are used (M=message, P=processor, and R=rank) inserted in the print command like the example.

Source code 5 – Example of using special tags

```
1: // Print off a hello world message
2: printf("message: {M}%s{M} from processor: {P}%s{P}, rank: {R}%d{R} out of %
    d processors\n", "Hello World", processor_name, world_rank, world_size);
```

The next screen shows the result of the exercise for the student. Figure 33 shows code outputs; one is the lecturer output generated when adding the exercise (Admin Result), and the

Figure 30 – Compile and run parameters

The screenshot shows a web interface with a top navigation bar containing tabs: Parameters (selected), Files, Nodes (MPI), Results, and Special Tags. Below the tabs, there are four sections for configuring compile and run parameters:

- Command (compile):** A text input field containing `mpic++`.
- Arguments (compile):** A text input field containing `main.c -o main`.
- Command (execute):** A text input field containing `mpiexec`.
- Arguments (execute):** A text input field containing `-n 8 -host 10.0.1.5,10.0.1.7,10.0.1.8,10.0.1.6,10.0.1.3 main`.
- Extra Arguments (execute):** An empty text input field.

Source: Elaborated by the author.

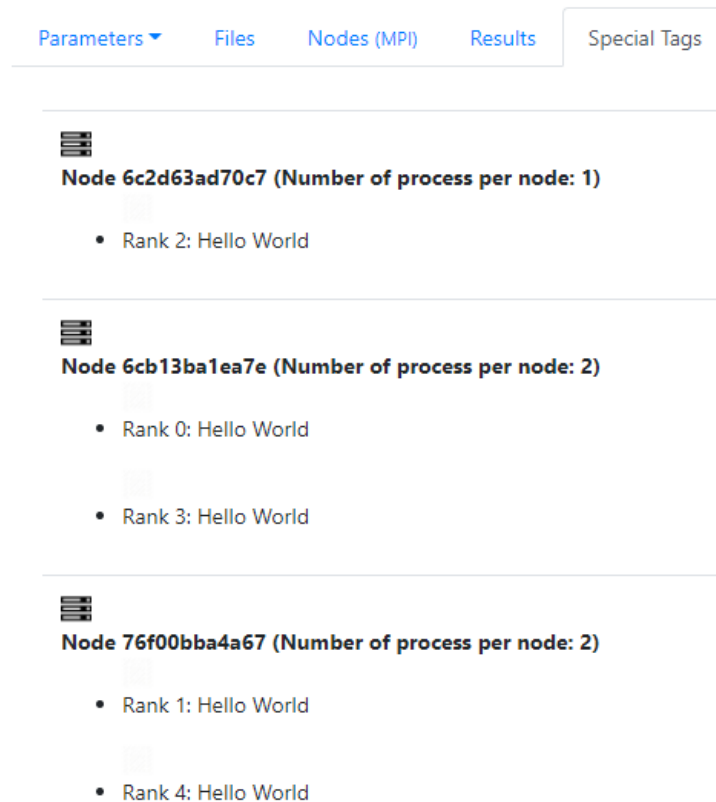
Figure 31 – Node selection

The screenshot shows the 'Nodes (MPI)' tab selected in the top navigation bar. Below the tabs, there are two sections for configuring node selection:

- Number of process:** A text input field containing the value `8`.
- Nodes:** A list of nodes with checkboxes for selection. The nodes are `10.0.1.5`, `10.0.1.6`, and `10.0.1.3`. The checkbox for `10.0.1.6` is checked, while the others are unchecked.

Source: Elaborated by the author.

Figure 32 – Orderly presentation of the result



Source: Elaborated by the author.

other (Your Result) is the student output. For different outcomes, the system informs which row and column there is a divergence immediately after the code running, stimulating learning and facilitating correction. In addition, the teacher can set a time limit for the student to solve the exercise and for the number of submissions for the student to send the code.

Figure 34 shows the exercise screen for students. It is possible to check the last submission date, the previous result, the lecturer's score and feedback, and the number of attempts to solve the exercise. Also, expanding the screen to check all submissions, we have added code analysis, which provides the student with code quality feedback (code smell).

Also, we show the response time of the code execution and the number of events needed to send the code, that is, how many times the student must compile and execute to produce some result. This is important as it can allow the lecturer to measure the difficulty of the exercise or the student's difficulty in solving the problem.

Figure 35 shows which physical or virtual hosts are available. In the figure, there is only one host called "hal11". This host has an 8-core CPU, with the load split between three nodes. As already mentioned in Subsection 4.2.4, Iguana allows other load configurations besides. This type of approach is interesting for students without cluster access; they can create multiple virtual nodes to run MPI programs without any infrastructure.

Figure 33 – Result of the lecturer and student code

Answer: Find Words (Paulo Sergio Lopes de Souza) Answers

Time to finish: 254d 5h 44m 10s

Content Coding Diff

Admin Result Your Result: **NOT EQUAL**

The image shows a side-by-side comparison of two code execution results. On the left, under 'Admin Result', is a list of words and their coordinates: 4 75319,3,4; 5 549,4,3; 6 69,5,1; 7 826,6,4; 8 0101,-1,-1; 9 29292929,8,0; 10 1818181818,9,0; 11 - 3416,-1,-1; 12 - 4444,4,6; 13. On the right, under 'Your Result', is a similar list but with a red 'NOT EQUAL' status. The list on the right is partially obscured by a red box and a diagonal line pattern.

Source: Elaborated by the author.

Figure 34 – Exercises solved

Last Exercise Answers: Find Words

Filter:

ID	Name	Last Submission	Last Result	Score / Feedback	Attempts Number
164	Paulo Sergio Lopes de Souza	2020-12-10 16:45	EQUAL	No/No	1

Number	Submission	Code Analysis	Response Time	Events	Status	Action
1	2020-12-10 16:45		700.04ms	7	EQUAL	

Source: Elaborated by the author.

Figure 35 – Creating virtual nodes

hal11 (Mem: 31958MB | vCPU: 8)

Total cluster: vCPUs: 8 | Mem: 31958MB

Source: Elaborated by the author.

Finally, the lecturer can download all the programs sent and a spreadsheet with the execution data. Furthermore, it is possible to change all system settings, front-end and back-end ports, cluster operation mode, and add and remove nodes. To run programs on GPU, like CUDA, for example, Iguana has some limitations, such as the need for host GPU. However, not all educational institutions or students have this type of hardware. Therefore, we included future work to integrate the tool with NVEmulate and MCUDA (STRATTON; STONE; HWU, 2008) to support GPU emulation.

4.4 Testing and Deployment

The project to create a tool to help teach parallel programming began in 2018. Since the first version, extremely limited and without a graphical interface, Iguana has supported improvements based on feedback from students, professionals in the field, and lecturers. The tool was tested by students from the Federal Institute of Paraná, Instituto Federal Catarinense and in a concurrent programming undergraduate course at USP-ICMC before being used in the experiments.

Since 2020, the tool has been available online under the domain **www.parallelcoding.com** but is still not open to public users. However, the intention is to make it available for anyone to test parallel code. Also, we created an account on GitHub exclusively for Iguana through the link **github.com/iguana-hpc-usp/ICS**, with all the tool's source code (back-end and front-end) and also the preconfigured virtual machine. Thus, once the VM is downloaded and run, no configuration is required.

In addition to the VM, users can install directly on Linux to create an easy-to-configure cluster. When running, the script automatically installs all packages, and configurations are performed. Thus, deploying Iguana in a laboratory with 40 physical machines would not be difficult. For example, it is unnecessary to download the script; just run line 1 shown in [Source code 6](#)

Source code 6 – Script Install Command

```
1: apt install curl -y && curl -s https://raw.githubusercontent.com/iguana/
   doutorado/master/install.sh | bash -s --
```

Finally, the Iguana manual is also available for download with all its features and its use in detail. We separated the manual into three categories: installation process, lecturers, and students. [Appendix E](#) presents the latest version of the manual.

4.5 Final Considerations

This chapter shows the tool development through the requirements discovered in [Chapter 2](#) and the technologies integration presented in [Chapter 3](#). We created Iguana with the limited resources for an educational institution with a lack of resources. Thus, we can install the tool on a simple VM. Furthermore, the system is flexible enough to be installed on a cluster of physical machines taking advantage of all the resources.

The Iguana can run on a single VM with 1GB of RAM, allowing any student to download the tool and have a cluster system run parallelized codes in a few minutes. Also, it promotes another facility that usually requires a lot of time and expertise from professionals, which is to create a cluster on a physical network. Install Iguana on each machine and define one as master and the others as nodes; the system will automatically do all the necessary installations.

The tool offers the student an environment to write and execute his programs without installing any other software from the practical teaching of parallel programming. According to the student's and professionals' suggestions, we redesigned the interface for better usability.

The lecturer has absolute control of the tool, registering users, groups, and exercises. In addition, the tool automatically corrects the code and provides other options to a practical laboratory class.

Finally, we chose to make the tool open-source as we believe in the philosophical ideas involved in sharing code with the community. In addition, we would like to promote "parallel thinking" and Iguana directly contributes to that end. Thus, students no longer need to worry about implementation details and focus only on parallel code development. Likewise, lecturers do not need to worry about providing the infrastructure for lab classes.

EXPERIMENTAL EVALUATION

Iguana is a doctoral project, and it was developed and perfected over three years. The tool is in a solid stage of development and has no bugs discovered, and **we used the version 3.5 in the experiments**. We planned and ran three experiments to assess the use of the tool.

For the first experiment, we invited computer industry professionals and lecturers. We also chose people with no parallel programming experience to check the tool's performance with these users.

In the second experiment, we used the tool at the University of São Paulo with 129 undergraduate students in concurrent programming discipline. The objective of this experiment was to evaluate the efficacy of Iguana as a support tool for teaching parallel programming.

For the first two experiments, including participants, we applied a widely used usability test called System Usability Scale (SUS) and other questions for tool evaluation. Still, we conducted these two experiments with many participants into a simple virtual machine and multiple concurrent submissions of parallel programs.

Also, due to the covid-19 pandemic, students accessed the tool through the web browser in their homes. Iguana's queuing system allowed multiple students to write and test code simultaneously despite minimal virtual machine setup. The cluster used by the students had four virtual nodes, and the entire system was running under a VM with 8GB of storage, 2GB RAM, and one vCPU.

The third experiment consisted of validating tool efficacy for running the CUDA codes. The codes were extracted from [Silva \(2021\)](#) and did not include participants. For the tests, we used an Intel Core i7-3537U, 8GB RAM, 2TB of storage hosted in LaSDPC (Distributed Systems and Concurrent Programming) of the University of São Paulo.

Therefore, [Section 5.1](#) introduces the concept of evaluation with the SUS. Then, [Section 5.2](#) presents the experiment with lecturers and computing professionals. Next, [Section 5.3](#)

shows the experiments with the students. Then, [Section 5.4](#) presents the CUDA codes tested with the tool. Finally, [Section 5.5](#) contains considerations about this chapter.

5.1 System Usability Scale

We use SUS for the survey after experiments with professionals, lecturers, and students. Despite its described as a quick and dirty method, there is a significant correlation between the results indicated by the SUS in direct comparisons with usability tests. Researchers and professionals widely use SUS to conduct usability tests of different websites and tools ([PERES; PHAM; PHILLIPS, 2013](#)).

[Chart 13](#) shows the questions for the usability test (SUS). This SUS scoring system uses ten questions and each with five answer options. The answers start with totally disagree (value 1) and totally agree (value 5). The even-numbered questions (2 and 4) represent negative levels of agreement; that is, the fewer answers, the better the result. Conversely, odd questions (1,3, and 5) represent positive levels of agreement ([BROOKE, 1996](#)).

Chart 13 – SUS questions for usability

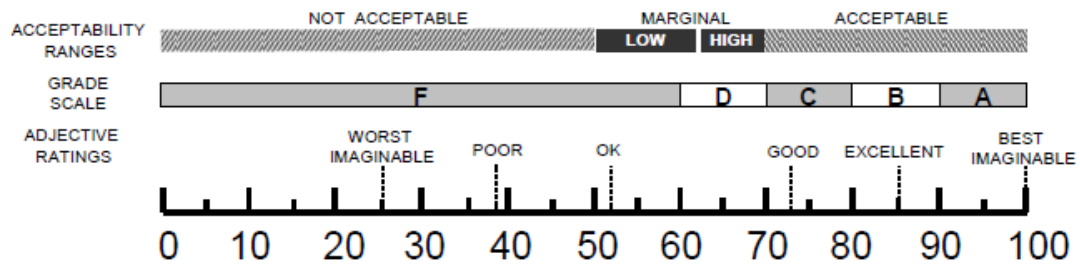
Questions
<ol style="list-style-type: none"> 1. I think that I would like to use this system frequently. 2. I found the system unnecessarily complex. 3. I thought the system was easy to use. 4. I think that I would need the support of a technical person to be able to use this system. 5. I found the various functions in this system were well integrated. 6. I thought there was too much inconsistency in this system. 7. I would imagine that most people would learn to use this system very quickly. 8. I found the system very cumbersome to use. 9. I felt very confident using the system. 10. I needed to learn a lot of things before I could get going with this system.

Source: Adapted from [Brooke \(2013\)](#).

Thus, for even responses, subtract the user responses from 5, and for odd responses, remove one from the user response. This scales all values from 0 to 4. Afterward, we must add up all the converted answers for each user and multiply this total by 2.5. This converts the range of possible values from 0 to 100.

Finally, the score is divided into scales, as shown in [Figure 36](#); 100 is considered the best user experience, equivalent to an A grade. Above 80.3 is rated excellent, and between 68 and 80.3 is classified as good. Ratings below 68 are considered an F grade and are not acceptable ([BANGOR; KORTUM; MILLER, 2009](#); [BROOKE, 2013](#)).

Figure 36 – SUS scoring scale



Source: Elaborated by the author.

5.2 Evaluating Usability with Lecturers and Computing Professionals

The first experiment objective was to evaluate the tool usability with lecturers and professionals in computing. All participants have extensive experience in sequential programming and different knowledge levels in parallel programming. Furthermore, some of them still have experience with teaching parallel programming. They work in the industry as software developers or in academia as lecturers.

We sent the invitation to 36 participants, of which 24 responded. Afterward, we sent an email again with the instructions for experimenting to the participants:

1. First step: a link to a 5-minute training video explaining how to log in and execute an exercise in the tool;
2. Second step: participants had to access the tool and perform four exercises. We sent the answers to the exercise to not overload the guests, as the purpose was to evaluate the Iguana and not the participants' ability to solve exercises; and
3. Last step: after completed step two, we asked the guests to answer a survey with 25 questions.

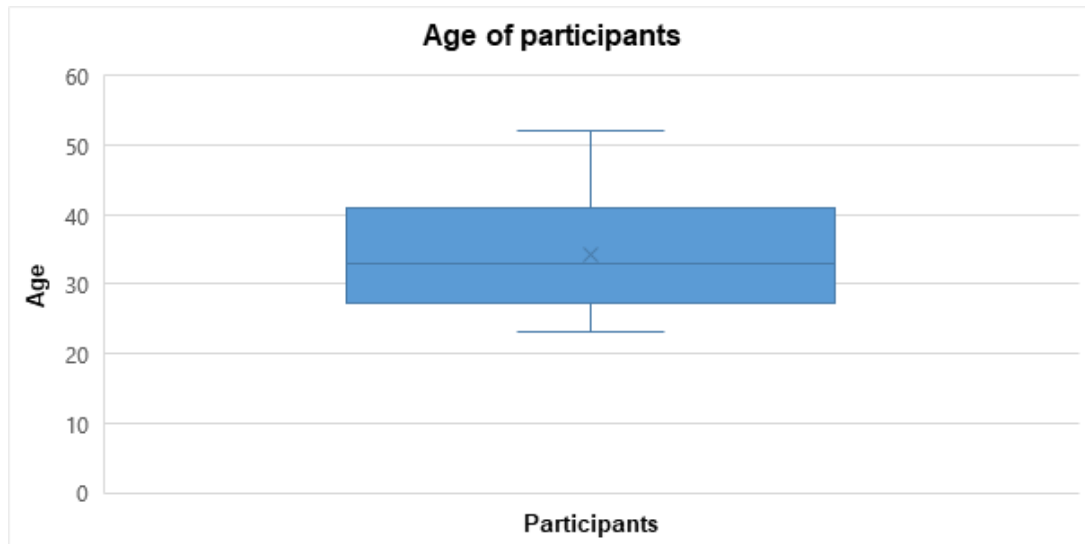
The following ten questions evaluated the tool usability with SUS. Then, another five questions to assess user satisfaction with the interface and finally, ten questions about running parallel programs with Iguana focus on teaching.

5.2.1 Distribution of professionals

The survey has seven questions to analyze the distribution of professionals and lecturers according to age, academic degree (undergraduate, specialist, master or doctor), place of work, guest location, years of computing experience, years of parallel programming experience, and which tool they usually compile and run parallel programs.

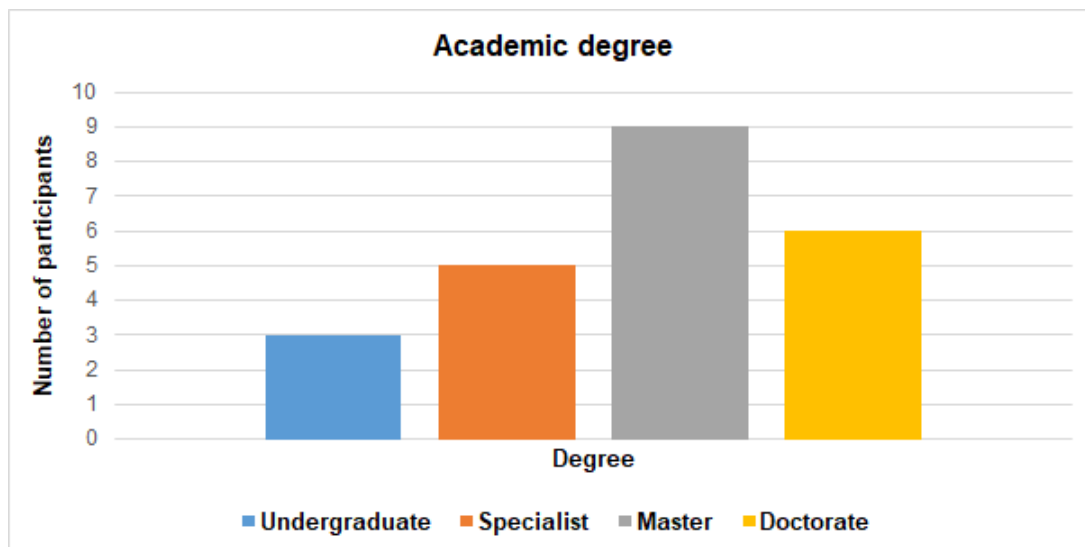
Figure 37 presents the age of the participants, where the minimum was 23, the maximum was 52, and the median was 33. Thus, the principal age ranged (between the first and third quartiles, called the interquartile range) from 28 to 41 years old. Most participants for this experiment had masters and doctoral degrees. Figure 38 shows the academic degree, where 3 participants were undergraduate, 5 specialists, 9 master's degrees, and 6 doctorates.

Figure 37 – Age of participants in the experiment with professionals and lecturers



Source: Elaborated by the author.

Figure 38 – Academic degree of participants

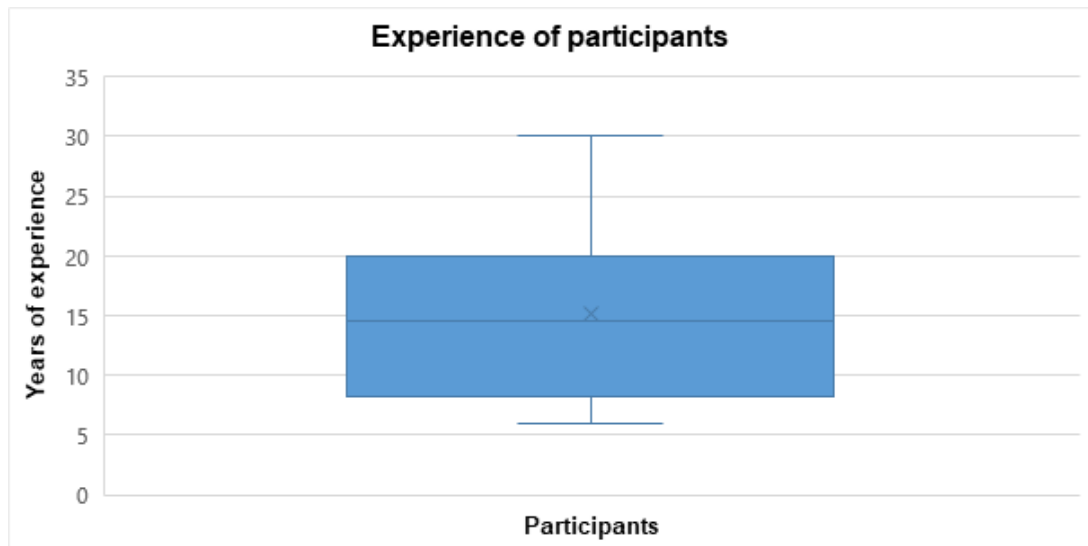


Source: Elaborated by the author.

The boxplot in Figure 39 shows the years of experience of the participants in parallel programming, where the minimum was 6, the maximum was 30, and the median was 14.5. Thus, the interquartile years of the experience were between 8.25 and 20.

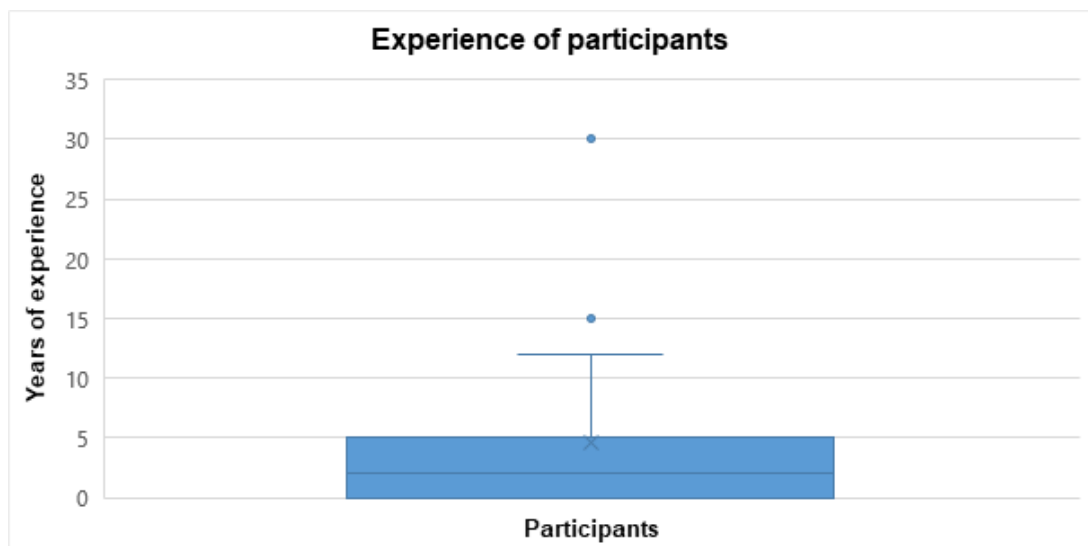
Figure 40 shows the years of experience of the participants in computing, where the minimum was 0, the maximum was 12, and the median was 2. Also, this boxplot features two outliers, 15 and 30. Thus, the interquartile years of the experience were between 0 and 5.

Figure 39 – Experience in the computing area



Source: Elaborated by the author.

Figure 40 – Experience in parallel programming



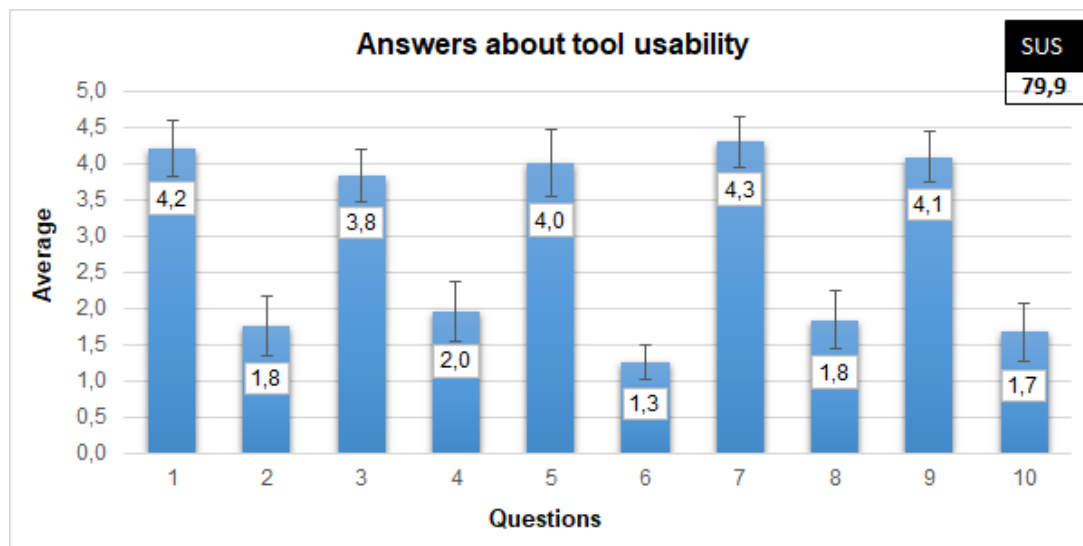
Source: Elaborated by the author.

5.2.2 Results of the experiment - 1st round

Figure 41 shows the participants' responses to SUS questions in Chart 13. The average ranged from 1.3 to 2.0 for negative questions (even) and from 3.8 to 4.3 for positive questions

(odd). The SUS score for this first questionnaire was 79.9 with a 95% confidence level. As shown in Figure 36, this score is classified as acceptable.

Figure 41 – Answers for the ten SUS questions about usability - 1st round



Source: Elaborated by the author.

Chart 14 shows five questions added to identify other subjective responses from the participants about using the system and the information displayed in the tool. These questions were taken from the User Interface Satisfaction Questionnaire (QUIS) (CHIN; DIEHL; NORMAN, 1988) and allowed us to evaluate the Iguana better.

Chart 14 – Questions for user interface satisfaction

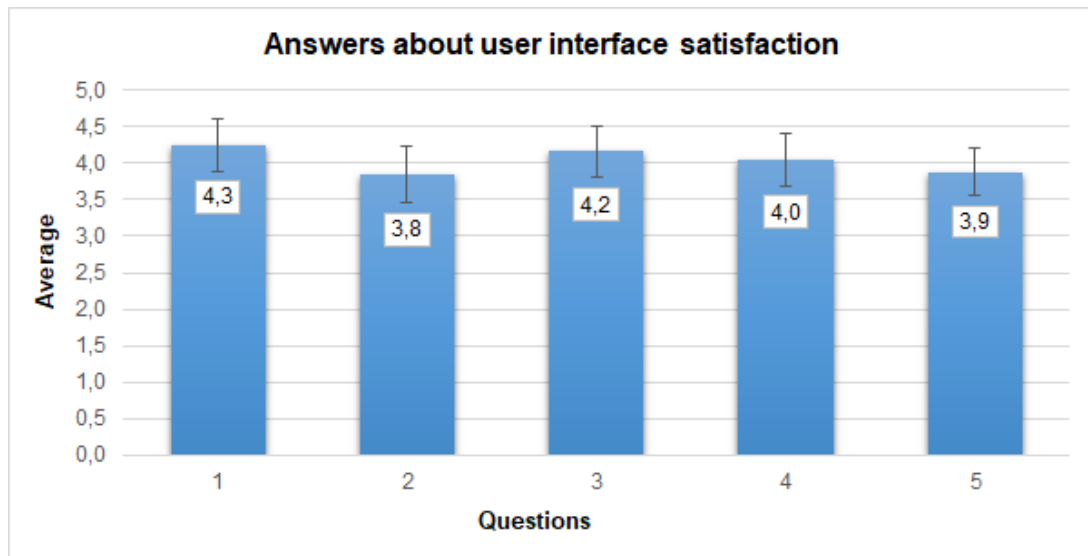
Questions
1. I feel comfortable with this system.
2. It was easy to find the information I needed.
3. I liked using the system interface.
4. The system interface is nice.
5. The organization of information on the system screen is clear.

Source: Adapted from Chin, Diehl and Norman (1988).

Figure 42 shows the responses to the second questionnaire, shown in Chart 14. Again, the average of the responses was 4 (agree), with a confidence level of 95%, indicating that the participants were satisfied with the tool's interface.

Chart 15 presents the last ten questions sent to the participants containing questions about executing parallel programs with the tool. The results allowed us to observe the most prominent items in this type of programming. Figure 43 shows the averages of the responses, again with a 95% confidence level. The average ranged from 4.1 to 4.6 for inexperienced participants and 3.9 to 5.0 for answers from experienced guests. The overall average fluctuated from 4.1 to 4.9.

Figure 42 – Answers for the five user interface satisfaction questions - 1st round



Source: Elaborated by the author.

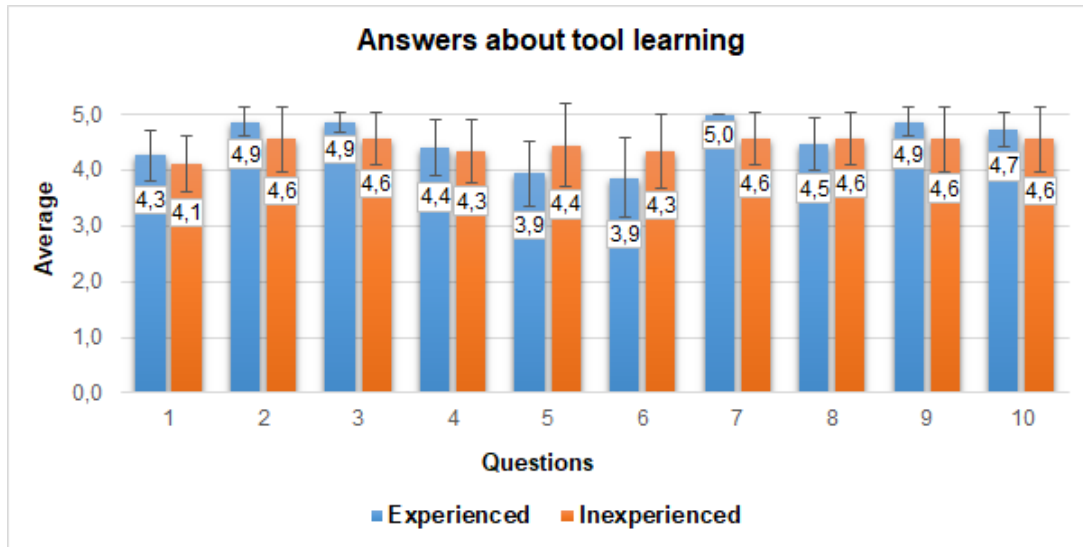
Also, we applied a paired test t to check whether there was a significant difference between the responses of the inexperienced and experienced participants. Again, we found no significant difference between groups of responses ($p > 0.05$).

Chart 15 – Questions about running parallel programs in Iguana with focus in teaching

Questions
1. I find the tool intuitive/easy in program execution.
2. I think the Iguana tool can make teaching parallel programming easier for students with little or no Linux skills.
3. I think that the Iguana tool can encourage students to run parallel programs.
4. I find the tool more intuitive for executing parallel programs than the traditional command-line way.
5. I think that the time to create and execute program in the Iguana tool is less than the conventional method, through the command line.
6. I think editing program directly in the browser is better than having to use an installed application.
7. I think that comparing the result of the lecturer's program helps to produce a correct result.
8. I think the tool is flexible enough to allow me to change my compilation and execution parameters.
9. I would recommend using the tool.
10. I would use the tool in practical classes.

Source: Elaborated by the author.

Figure 43 – Answers about running parallel programs in Iguana with focus in teaching - 1st round



Source: Elaborated by the author.

Finally, we asked participants to submit any suggestions for improvement or feedback regarding the tool. We received many constructive answers as adding auto-recognition of compilation parameters according to the code typed by the student, auto-save of code while writing it in the editor, and some interface improvements, for example.

We included in [Appendix B](#), in [Chart B.1](#), the responses related to Iguana improvements, the risk of implementing these suggestions, and which ones were applied. We registered 31 recommendations, of which twenty-three were used in the tool, and eight were designated for future work.

5.2.3 Results of the experiment - 2nd round

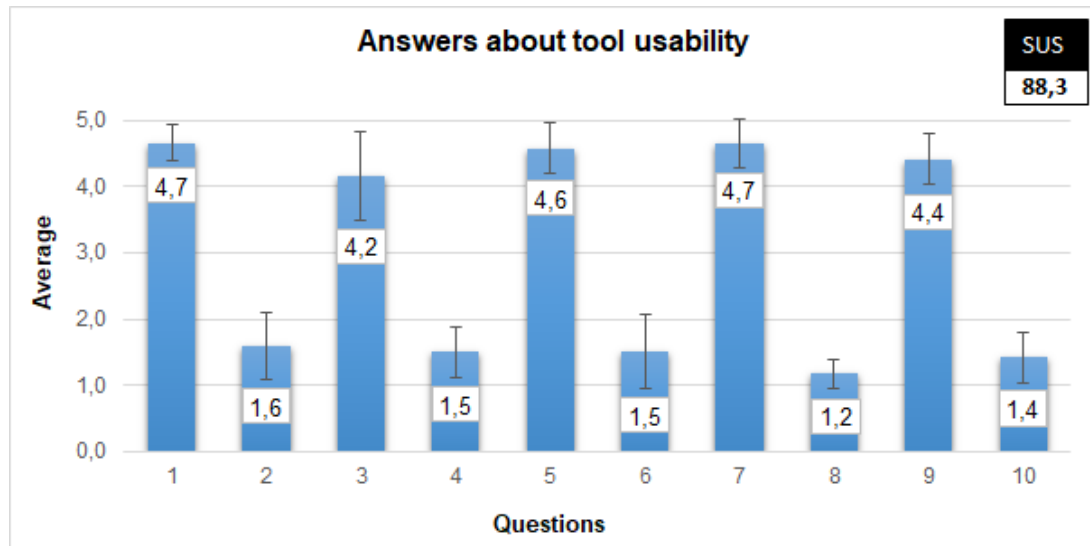
After applying the suggestions pointed out in the previous experiment, we decided to ask the participants for another evaluation with these new features in Iguana. Again we sent 24 invitations and received 12 responses, two undergraduates, one specialist, six masters, and three doctors.

For the second round, the median age of participants was 28.5, with a minimum of 24 and a maximum of 50, and an interquartile between 25.25 and 40.75. For experience in the computing field, the median was 9.5, with a minimum of 6, a maximum of 30, and an interquartile between 7 and 19.25. For experience in parallel programming, the median was 3.5, with a minimum of 0, a maximum of 4.75, and interquartile between 0.25 and 4.75, and an outlier of 15.

[Figure 44](#) now shows the score for responses to the questionnaire SUS. Comparatively, in the first experiment, in [Figure 41](#), the SUS was 79.9 and, after applying the features in Iguana, the SUS increased to 88.3, an excellent rating (acceptable) in the score shown in [Figure 36](#). Also,

the figure still presents the mean variation for negative answers, from 1.2 to 1.6, and positive responses, from 4.2 to 4.7.

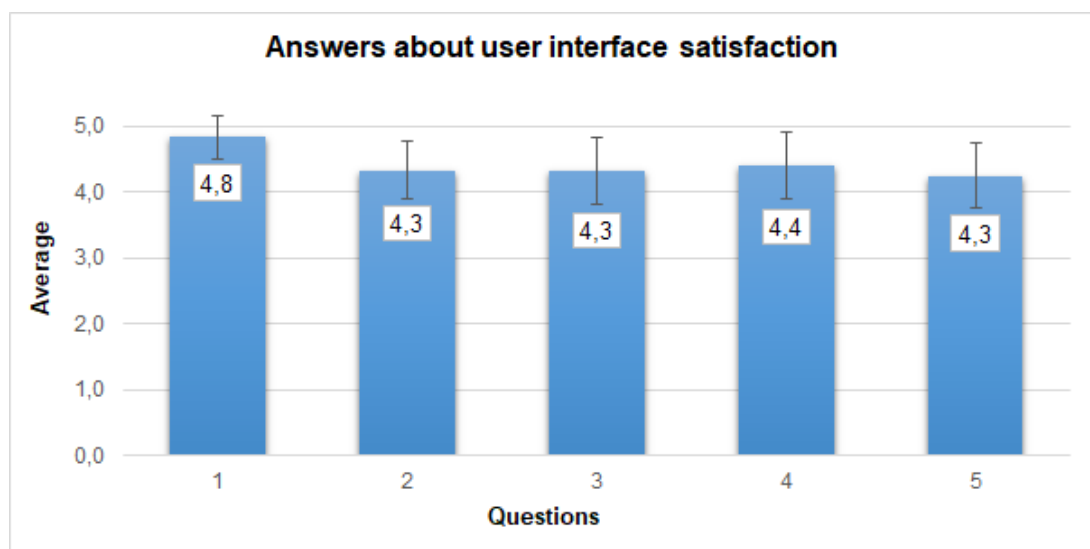
Figure 44 – Answers for the ten SUS questions about usability - 2nd round



Source: Elaborated by the author.

Figure 45 shows the mean responses for the QUIS. In the first experiment, we received an average of 4 and now 4.5 for the second round. The participants showed greater satisfaction with the interface after the features, increasing the score (of agree to strongly agree), evidenced by the responses presented. The confidence level for the SUS and QUIS responses was 95%.

Figure 45 – Answers for the five user interface satisfaction questions - 2nd round

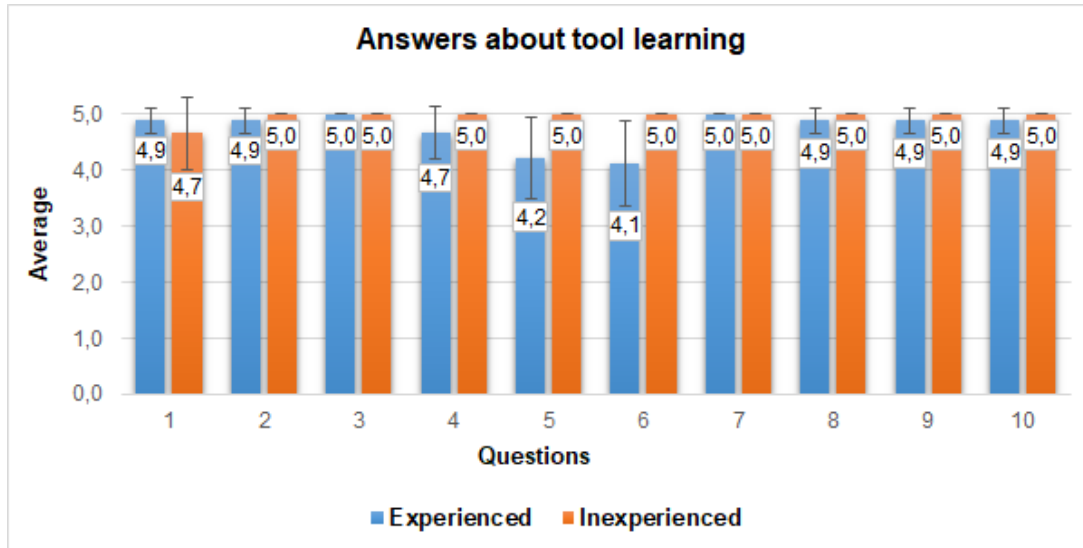


Source: Elaborated by the author.

Figure 46 shows the averages for the last section of questionnaire answers that evaluated the parallel program execution in the tool. The average of responses for inexperienced participants

was 4.7 to 5, and for experienced participants was 4.1 to 5.0. Compared with the same section of the first experiment, we registered a mean increase from 4.5 to 4.7 for experienced participants and 4.5 to 5 for inexperienced participants. Thus, the result approached the borderline of significance for these two groups of responses ($p = 0.07$).

Figure 46 – Answers about teaching parallel programming through the tool - 2nd round



Source: Elaborated by the author.

Finally, in [Appendix C](#), in [Chart C.1](#), we listed all the positive and negative feedback regarding the use of Iguana, and we performed several adjustments to improve usability.

5.3 Teaching Parallel Programming with Iguana

The second experiment was performed during the High-Performance Computing course at the University of São Paulo in the second half of 2020 and involved 129 undergraduate students in Computer Science. The students were divided into 28 groups named A and B to facilitate teacher monitoring. The objective of the experiment was to evaluate the efficacy of the tool in supporting parallel programming.

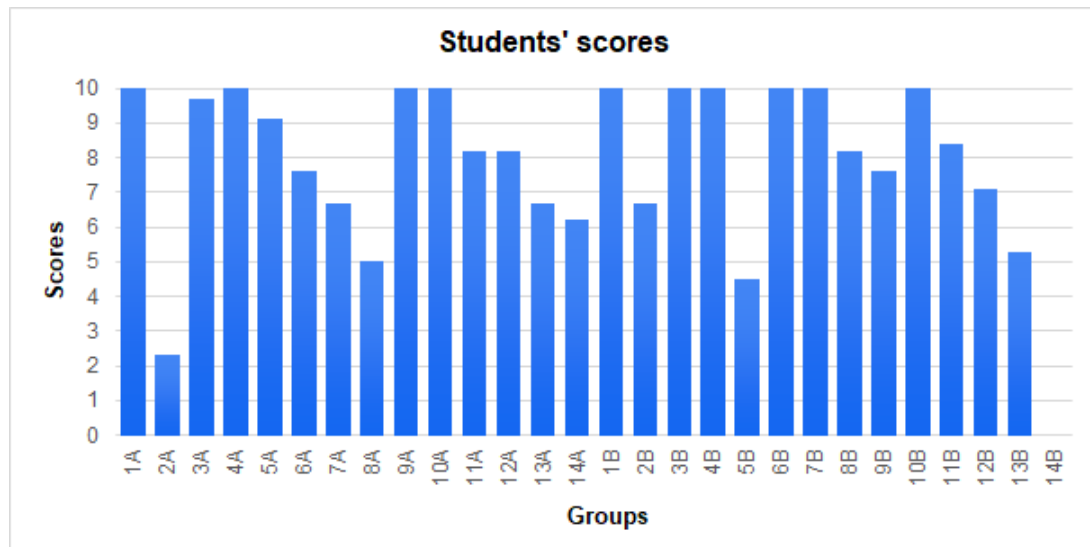
At the end of the course, the 129 students used Iguana to develop two algorithms. The programs used in this experiment were more complex than other algorithms at the beginning of the course. They involved collective message passing, dynamic generation of processes in MPI, and a hybrid model (OpenMP and MPI).

5.3.1 Iguana and non-Iguana grades

The first algorithm objective was to determine how many elements in matrix columns were smaller than the average of the same matrix elements. The second algorithm should find a list of words within strings ordered in an array.

The score for each program is expressed by the weighted average of three parameters: 50% for correct use of MPI resources, 30% for proper program execution, and 20% for source code quality. Figure 47 shows the scores of the 28 groups (sections A and B) for the first exercise.

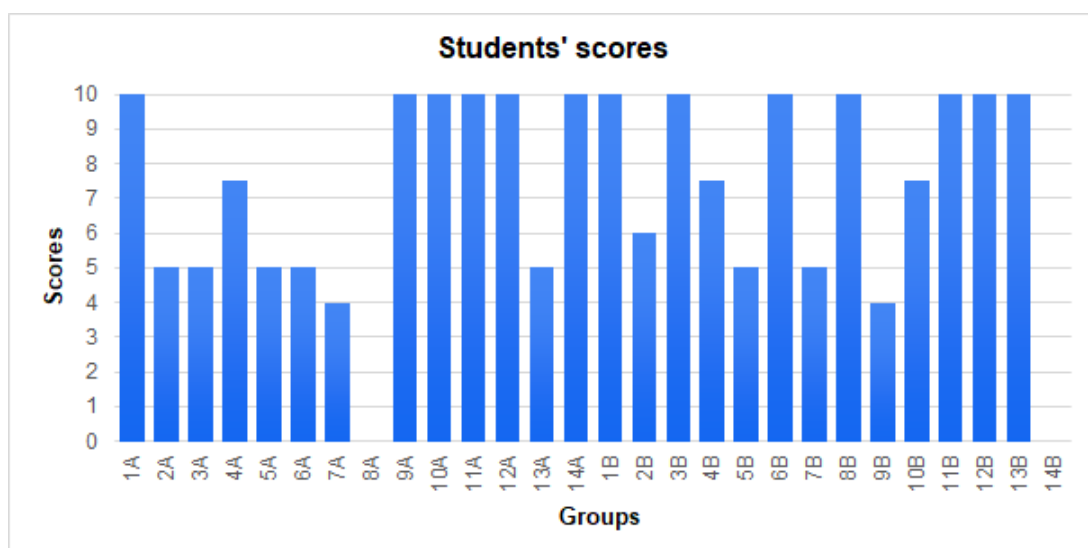
Figure 47 – Students' scores for the first exercise



Source: Elaborated by the author.

The first parameter of score checks if the student correctly used the OpenMP and MPI resources to solving the exercises. The second parameter validates the exercise's correct answer, and the third parameter corresponds to the quality of the source code, such as comments, indentation, and readable structure. Figure 48 shows the scores for the second exercise (Find Words).

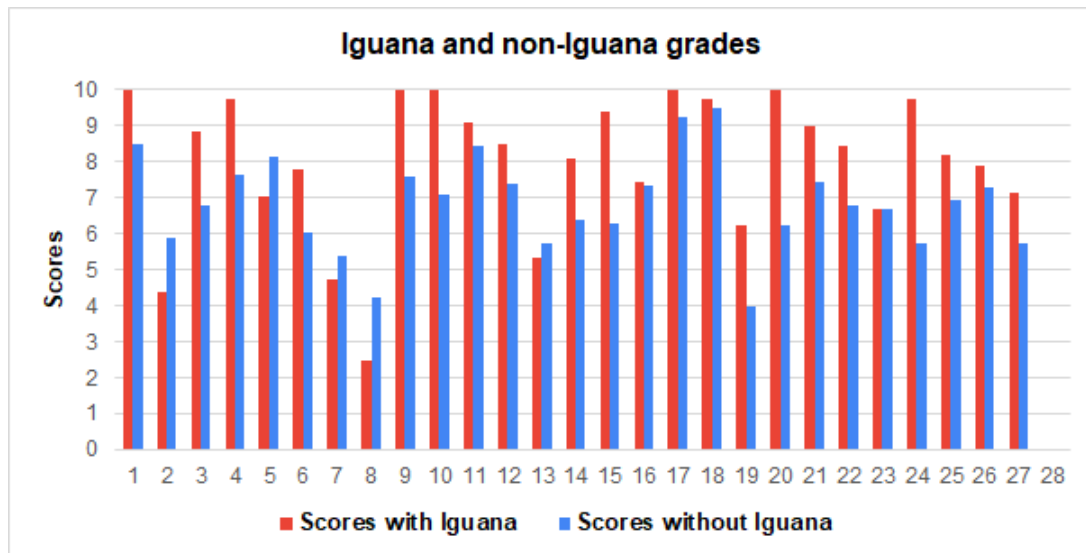
Figure 48 – Students' scores for the second exercise



Source: Elaborated by the author.

Again, the scoring system used was the same as in the first exercise; that is, the weighted average for all 28 groups' scores (sections A and B). Figure 49 shows the averages of the participants' scores for nine exercises conducted before the experiments (non-Iguana grades) and two Iguana activities (Iguana grades).

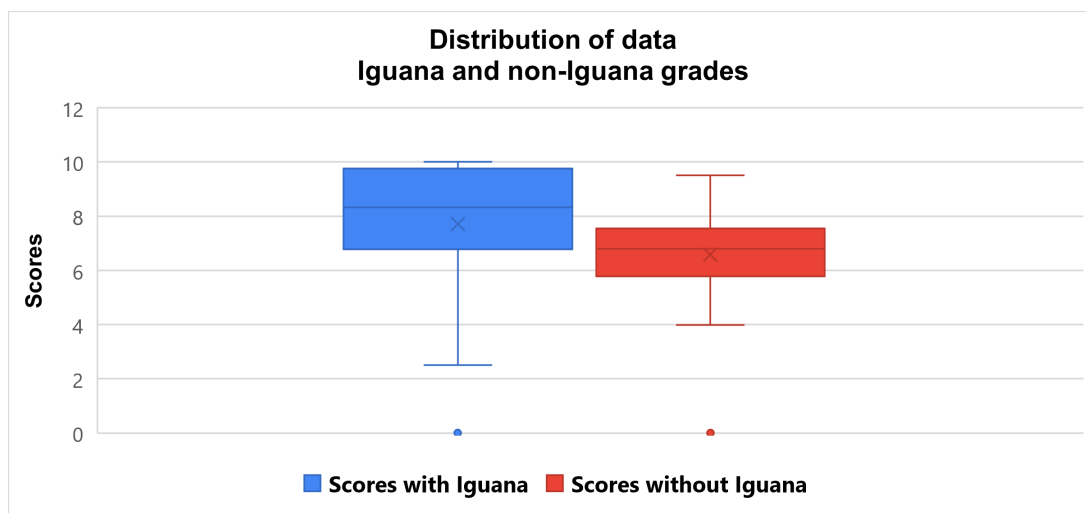
Figure 49 – Comparing the variability of scores in exercises with and without Iguana support



Source: Elaborated by the author.

Figure 50 shows the boxplot between the two means of the previous exercises and the exercises with the tool. Again, we observe similar behavior (variability) in this graph, where scores with Iguana have the median, the first and third quartile above scores without Iguana support.

Figure 50 – Comparing the distribution of scores in exercises with and without Iguana support



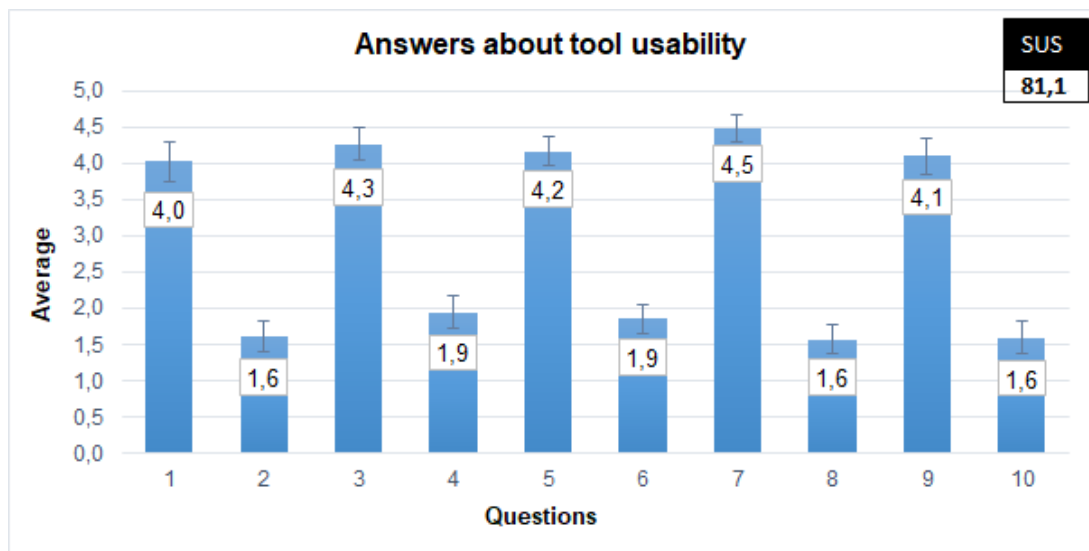
Source: Elaborated by the author.

We performed this comparison to detect if Iguana could add any difficulty to problems solve. The statistical test confirmed significance between the means of two scores ($p < 0.05$), indicating higher scores using the tool. Despite this, we highlighted that two groups (8A and 14B) did not submit the exercises; they failed to develop parallel solutions with OpenMP and MPI for more complex problems. On the other hand, students reported to the lecturer that no problems were using the tool.

5.3.2 Results of the experiment

Students were invited, after completing the two exercises, to answer a questionnaire. However, to improve the confidence level in the responses, we notified the students that the questionnaire was not required and the responses would be anonymous. Thus, 54 out of 129 students participated in the survey. Figure 51 shows the students' responses to SUS questions in Chart 13.

Figure 51 – Answers related to usability to teach parallel programming through the Iguana



Source: Elaborated by the author.

The average ranged from 1.6 to 1.9 for negative questions (even) and from 4.0 to 4.5 for positive questions (odd). The SUS score for this first questionnaire was 81.1 with a 95% confidence level. As shown in Figure 36, this score is classified as acceptable. In addition to SUS, we also requested two other questionnaires for students to answer. Chart 16 shows the questions regarding the effectiveness of the execution of parallel programs in Iguana.

Figure 52 shows the answers averages to the questions presented in Chart 16. For example, question 3, about editing programs directly in the browser, had an average of 2.7, a point of divergence among students.

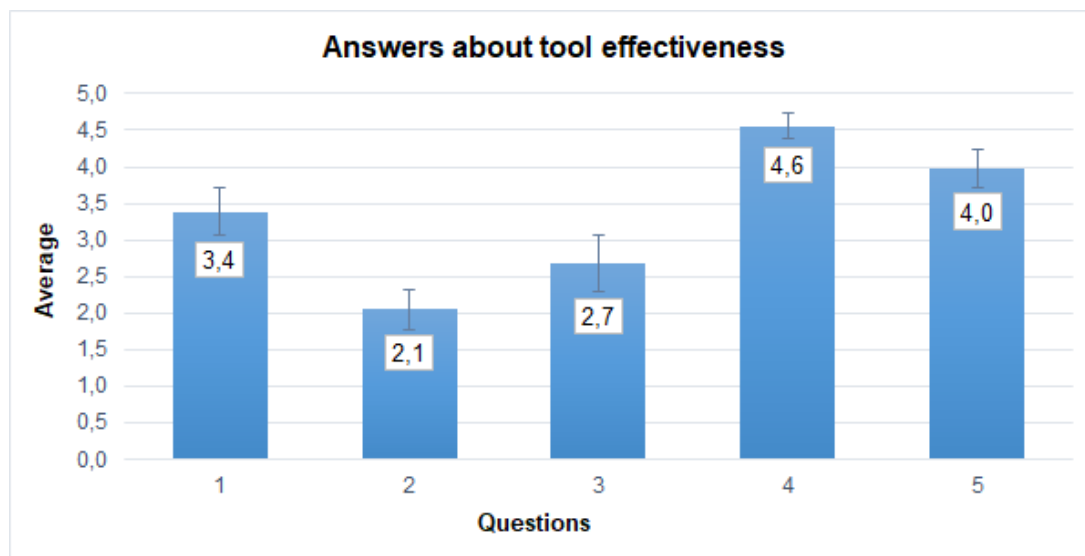
This divergence may have occurred because the students already used their Integrated Development Environments (IDEs) during the course, mainly using shell command lines (editing,

Chart 16 – Questions about Iguana execution effectiveness

Questions
1. I think that the time to create and execute programs in the Iguana tool is less than the conventional method, through the command line.
2. It was NOT easy to find the information I needed.
3. I think editing programs directly in the browser is better than having to use an installed application.
4. I think the tool is flexible enough to allow the compilation and execution of parallel programs on different platforms, just by changing compilation and execution parameters for MPI, Open-MPI or CUDA.
5. The organization of information on the system screen is clear.

Source: Elaborated by the author.

Figure 52 – Answers for Iguana execution effectiveness



Source: Elaborated by the author.

compiling, and executing); they used Iguana at the semester end. For the last section of the student questionnaire, we presented questions about the effectiveness of learning using the tool, shown in [Chart 17](#).

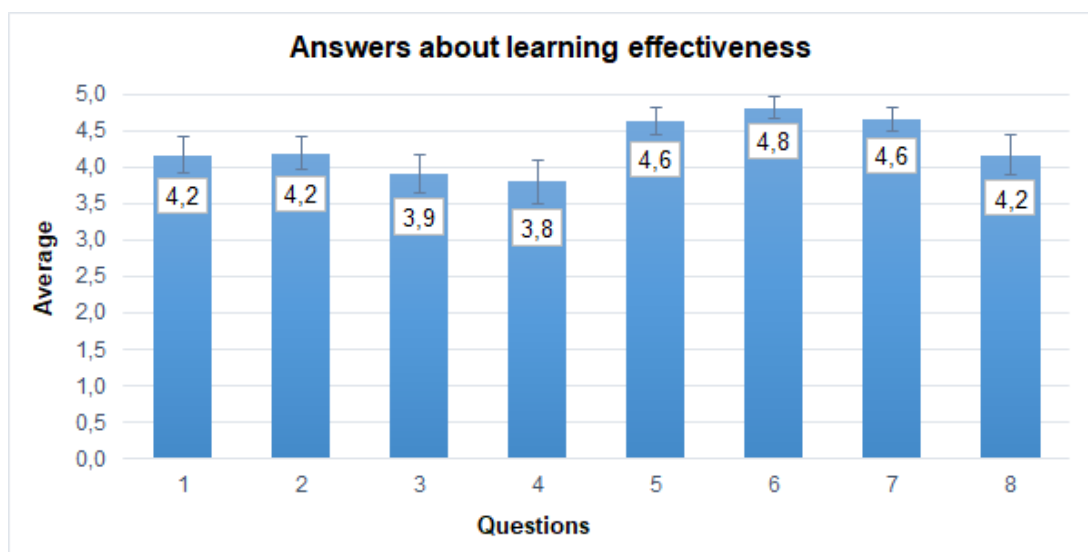
Students evaluated Iguana's strengths and weaknesses compared to their previous experiences, that is, in the context of their learning. In addition, students developed parallel programs in practical laboratory classes in the experiments, with code editing, compilation, execution, and comparison of the lecturer's results through a web browser. Thus, [Figure 53](#) shows how students evaluated these activities offered by Iguana about the questions in [Chart 17](#). The average of the answers ranged from 3.8 to 4.8, indicating a positive evaluation of these topics.

Chart 17 – Questions about learning effectiveness through Iguana

Questions
1. I think first-year students (with little or no experience in programming) can learn parallel programming more easily using Iguana than with command-line shells or other pre-installed applications/IDEs.
2. I think it should be easier to learn parallel programming remotely, without face-to-face classes, using the Iguana tool than with command-line shells or other pre-installed applications/IDEs.
3. I think it should be easier to learn parallel programming in person (with practical classes in person) using the Iguana tool than with command-line shells or other pre-installed applications/IDEs.
4. I think that editing the program directly in the browser without using a pre-installed application or IDE can facilitate the teaching of parallel programming.
5. I think that comparing the result of the lecturer's program helps produce a correct result than submitting an exercise without knowing the expected answer.
6. I think the tool can facilitate parallel programming learning even without the need for a physical and local infrastructure of parallel machines.
7. I think the use of the tool facilitates learning in practical classes.
8. I think that using the tool motivates the learning of parallel programming because I get immediate feedback compared to the traditional submission that allows analyses by the lecturer just after.

Source: Elaborated by the author.

Figure 53 – Answers for learning effectiveness through Iguana



Source: Elaborated by the author.

At the end of the experiment, we asked participants to submit any suggestions for improvement or feedback regarding the tool. We received many constructive answers and excellent feedback, some encouraging ones such as:

I thought the tool made the development of the programs a lot easier, as it saves time in organizing the files and applications needed for compilation and execution (Anonymous, 2020).

The tool is handy because not everyone has machines to execute the program (which might need cards from Nvidia, for example), and the tool helps a lot in this regard (Anonymous, 2020).

This tool was of great help to us (students) because we did not have a device configured at home for the development and execution of parallel programs. Iguana allowed the introduction to the parallel programming in few steps (Anonymous, 2020).

We included in [Appendix B](#), in [Chart B.2](#), the responses related to Iguana improvements, the risk of implementing these suggestions, and which ones were applied. We registered 11 recommendations, of which five were used in the tool, and eight were designated for future work. Finally, in [Appendix C](#), in [Chart C.2](#), we listed all the positive and negative feedback regarding the use of Iguana, and we performed several adjustments to improve usability.

5.4 Experiments with CUDA

For this experiment, we used the CUDA algorithms developed in the undergraduate course ([SILVA, 2021](#)). We executed tests at the Distributed Systems and Concurrent Programming Laboratory (LaSDPC) of the ICMC-USP. The equipment used was a Core i7-4790 3.60GHz processor, 32GB of RAM, and a GeForce GTX 650 video card.

The test consisted of running the selected codes in Iguana on one of the lab machines with a video card. The objective was to determine if the tool would execute the codes correctly. For this, we used 12 ready-made algorithms with test cases and the correct output produced by Silva.

5.4.1 Hello World

This test aims to print a Hello world message in $N \times M$ times, using N blocks and M threads, in CUDA. [Source code 7](#) shows the first code used in this experiment. The other programs are in the cited work due to many code lines. All examples follow the same external file calling pattern, as seen in lines 8 to 11.

Source code 7 – Hello World in CUDA

```

1: __global__ void hello(){
2:     printf("Hello world\n");
3: }
4:
5: int main(int argc, char **argv){
6:     int blocos, threads;
7:
8:     FILE * pFile;
9:     pFile = fopen("file.txt", "r");
10:    fscanf(pFile, "%d ", &blocos); //read the matrix line.
11:    fscanf(pFile, "%d ", &threads); //read the matrix column.
12:
13:    hello<<<blocos, threads>>>();
14:    cudaDeviceSynchronize();
15: }

```

Table 18 shows the contents of the input file used in [Source code 7](#). With this input, the code generated the result shown in the output print column. [Figure D.1](#) in [Appendix D](#) shows the result of [Source code 7](#). Again, we ran the code in Iguana and returned the same output.

Chart 18 – Input and output of hello world in CUDA

Input file	Output print
2 4	Hello world Hello world Hello world Hello world Hello world Hello world Hello world Hello world

Source: Adapted from [Silva \(2021\)](#).

5.4.2 Metrics of descriptive statistics

This algorithm aims to calculate for each sample (column) the following metrics: arithmetic mean, harmonic mean, median, mode, variance, standard deviation, and coefficient of variation, as described below ([SILVA, 2021](#)):

- Arithmetic mean: sum of all elements in the sample, divided by the sample size (sum of the rows in the matrix column divided by the number of rows);

- Harmonic mean: the ratio between the sample size and the sum of the inverse of the samples;
- Median: average element of the sample (average element of the ordered column). For an even number of elements, the median is the mean between the elements in the middle $((n/2+n/2+1)/2)$;
- Mode: most frequent element of the sample (the element that appears the most in the column, if there is more than one, only the first is considered. If not, it returns -1);
- Variance: sum of squares of the differences between the sample element and the calculated arithmetic mean;
- Standard deviation: square root of the variance; and
- Coefficient of variation: the ratio between standard deviation and arithmetic mean.

Chart 19 shows the contents of the input file used to calculate the descriptive statistics metrics. With this input, the code generated the result shown in the output print column. Figure D.2 in Appendix D shows the result. Again, we ran the code in Iguana and returned the same output.

Chart 19 – Input and output of descriptive statistics metrics in CUDA

Input file	Output print
6 4	12.0 17.0 11.5 13.5
9 8 4 5	8.1 12.1 6.6 3.7
4 12 20 40	8.5 12.0 10.5 7.5
8 8 4 4	8.0 8.0 4.0 -1.0
8 12 4 21	110.0 188.4 68.7 217.9
33 44 20 1	10.5 13.7 8.3 14.8
10 18 17 10	0.9 0.8 0.7 1.1

Source: Adapted from Silva (2021).

5.4.3 Thresholding

Thresholding is used to apply a filter and correct brightness to images by defining the threshold and applying it on all pixels (cells of a matrix). The algorithm reads the threshold from a file and compares it with matrix elements. If the element is greater than the threshold, the column value is replaced by 1, and if it is smaller, the value is replaced by 0 (SILVA, 2021).

Chart 20 shows the contents of the input file used to thresholding. With this input, the code generated the result shown in the output print column. Figure D.3 in Appendix D shows the result. Again, we ran the code in Iguana and returned the same output.

Chart 20 – Input and output of thresholding in CUDA

Input file	Output print
3 3	0.0 1.0 0.0
0.5	0.0 1.0 1.0
0.5 0.9 0.3	1.0 1.0 0.0
0.5 0.7 0.7	
0.8 0.8 0.3	

Source: Adapted from [Silva \(2021\)](#).

The first line of the input file defines the number of lines and columns of the matrix, the second line is the threshold value, and the other lines are the matrix elements.

5.4.4 Greatest common divisor

The algorithm reads a vector of n integer elements. Then, it calculates the greatest common divisor between these elements: the largest divisor of two or more natural numbers that can be found by dividing these numbers by the natural numbers greater than zero ([SILVA, 2021](#)).

[Chart 21](#) shows the contents of the input file used for this algorithm. With this input, the code generated the result shown in the output print column. [Figure D.4](#) in [Appendix D](#) shows the result. Again, we ran the code in Iguana and returned the same output.

Chart 21 – Input and output of greatest common divisor in CUDA

Input file	Output print
3	15
45 15 135	

Source: Adapted from [Silva \(2021\)](#).

The first line indicates the vector size (number of elements). The second line shows the vector components, that is, the factors used to calculate the greatest common divisor.

5.4.5 Matrix multiplication

The next program calculates the product of two square matrices of type double ($A * B$) and prints the result into a third matrix (C) ([SILVA, 2021](#)).

[Chart 22](#) shows the contents of the input file used to matrix multiplication. With this input, the code generated the result shown in the output print column. [Figure D.5](#) in [Appendix D](#) shows the result. Again, we ran the code in Iguana and returned the same output.

The first line of the input file defines the matrix dimension (M). The lines between the second line and line number M+1 represent the elements of matrix A. The line M+2 to the end of the file defines the second matrix B.

Chart 22 – Input and output of matrix multiplication in CUDA

Input file	Output print
3	30.0 24.0 18.0
1.0 2.0 3.0	84.0 69.0 54.0
4.0 5.0 6.0	138.0 114.0 90.0
7.0 8.0 9.0	
9.0 8.0 7.0	
6.0 5.0 4.0	
3.0 2.0 1.0	

Source: Adapted from [Silva \(2021\)](#).

5.4.6 Multiplication of vectors by a scalar

The algorithm objective is to perform the multiplication between a scalar and the elements of a vector, that is, given a scalar $x=4$ and a vector $v=(2,4,6)$, where the product of $x*v=4*$, is $(2.4.6)=(8.16.24)$ ([SILVA, 2021](#)).

[Chart 23](#) shows the contents of the input file used to algorithm . With this input, the code generated the result shown in the output print column. [Figure D.6](#) in [Appendix D](#) shows the result. Again, we ran the code in Iguana and returned the same output.

Chart 23 – Input and output of multiplication of vectors by a scalar in CUDA

Input file	Output print
3 4	8 16 24
2 4 6	

Source: Adapted from [Silva \(2021\)](#).

The first element in the first row indicates the vector size, the second element in the first row shows the scalar, and the second row indicates the vector.

Finally, the tests performed show that Iguana is also capable of executing and returning code in CUDA. We also tested many other algorithms and chose some of them for this work. In addition to the compilers experimented with the tool (OpenMP, MPI, and CUDA), Iguana is flexible enough to add new compilers and change the compilation and execution parameters.

5.5 Final Considerations

For the first and second experiments, we used SUS to measure Iguana usability. However, after searching for related work, we did not find any other tool evaluated in this way; therefore, this is the first work to use the SUS to measure the usability of tools for teaching heterogeneous parallel programming.

The tool had usability considered good (SUS of 79.9) in the first experiment with lecturers and professionals in the computing area, with extensive experience in IDE and command-line shells to develop parallel and sequential algorithms. In addition, after applying the features pointed out by these professionals, the SUS score changed to 88.3 on a scale ranging from 0 to 100, causing Iguana's usability is now considered excellent.

In addition to SUS, participants answered other questionnaires about teaching practical parallel programming with the tool: developing, compiling, and executing programs in practical laboratory classes. Iguana received the top 20% survey scores (4.0 and 5.0). Furthermore, participants indicated that the tool is intuitive and simple to use in practical classes; it can produce parallel programs with correct results and facilitate student learning.

In the second experiment, Iguana provided the necessary infrastructure to support the compilation and execution of parallel programs developed by the students during the practical laboratory classes. The participants, involving 129 students, used OpenMP and MPI, which represent separate programming models, for the proposed exercises. The submission of programs by the students showed that the tool is proficient in managing different infrastructures, multicore processors, and computer clusters.

In addition, students used a tool that offers a virtualized parallel infrastructure to run parallel programs without quality infrastructure or Internet access. They can even install on their machines or download a pre-configured virtual machine and run directly on their laptops and computers. Through the feedbacks collected in the experiment, we could observe that the students felt motivated to practice parallel programming using the Iguana, expanding the theory of parallel thinking.

Iguana also proved to be a helpful resource in reducing the time and energy spent on evaluating the exercises presented by the students. The lecturer responsible for the classes in the experiment successfully and quickly carried out all corrections and analyses of the programs submitted in the tool.

Iguana enabled remote education possible. Participants and students accessed the tool from their homes through simple browsers on different devices without an additional installation. In addition, the experiment was performed in the middle of the Covid-19 pandemic, where classes were obligatorily remotely.

This possibility of remote execution was fundamental and is the tool's strength; without Iguana, another alternative could have failed due to the complexity of running parallel programs by students remotely. Furthermore, even if not on a pandemic, many educational institutions do not have the minimum infrastructure and adequate workers to support parallel programs in practical laboratory classes. Iguana is a simple and accessible option to solve this problem.

RELATED WORK

The research conducted in [Chapter 2](#) and [Chapter 3](#) allowed us to define the necessary system parameters to create a lightweight cluster for teaching parallel programming in practical laboratory classes. Unfortunately, many of the approaches found involve developing proprietary software, high-cost clusters, shared infrastructure, skilled workers, and payment services.

Thus, [Chapter 4](#) presented the creation of Iguana, a tool that overcame these problems over the other infrastructures, as evidenced by the experiments conducted in [Chapter 5](#). Finally, this chapter shows a systematic mapping of tools to teach parallel and distributed programming and compares the results with the system proposed in this work. [Section 6.1](#) presents the papers found in the main research bases, and [Section 6.2](#) provides a summary for this systematic mapping.

6.1 Systematic Mapping

It is known that the complexity of parallel programming requires appropriate teaching, with an unequal division between theoretical and practical classes. It is estimated that one-third of the classes are theoretical. In the remaining period, students are trained in laboratories, programming through tools and libraries to acquire the skills necessary to develop parallel programs ([ALOISIO et al., 2005](#)). The tools, therefore, facilitate the peculiar teaching of parallel programming.

Therefore, this section shows the works that present tools dedicated to parallel teaching programming, where we highlight its advantages and disadvantages compared to Iguana. Our methodology for identifying tools can be described as follows, starting from the question: which teaching tools were developed for the execution of parallel programs?

We found 114 papers in the standard digital libraries (ACM DL, IEEE Xplore, Scopus, Science Direct) in 2011-2021. This search used the keyword “*Teach* Parallel Programming*”

Tool". After analyzing the titles, abstracts and reading the full articles, it was possible to select the 12 most relevant articles that present at least one tool for teaching parallel programming (Table 4). In Chart 24, we compare Iguana's resources with the other tools found.

Table 4 – Selected papers

Database journal	Number of papers	Alter filter
ACM Digital Library	5	2
IEEE Digital Library	57	3
Science@Direct	11	6
Scopus	30	0
Springer Link	11	1

Source: Elaborated by the author.

6.1.1 aCe C

The first tool found in the paper (DORBAND; ABURDENE, 2002) is an effort to encourage students to develop parallel applications. The aCe (Architecture Adaptive Computing Environment) is a C-based parallel language for architecture-adaptive programming.

aCe was developed to provide essential parallel programming execution and communication capabilities and make these features more accessible to programmers and students. Figure 54 shows an aCe program. Note that it looks no different from a standard C program. This program will print "Hello aCe World" 10 times because the printf command is executed by each of the ten threads of A (DORBAND; ABURDENE, 2002).

Figure 54 – Hello aCe World

```

/*
                                Bundle of Hello Worlds
                                aCe program
*/
#include <stdio.h>

threads A[10];

int main () {
    A.{ printf("Hello aCe World\n"); }
}

```

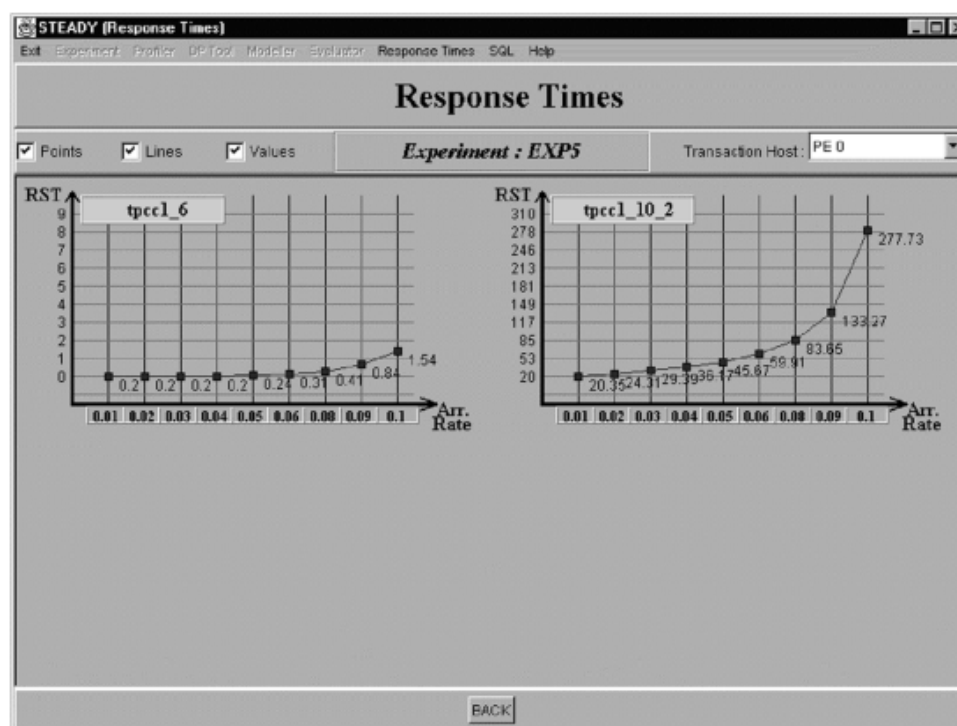
Source: Dorband and Aburdene (2002).

The aCe language facilitates aspects of parallelism such as race condition, process synchronization, and message and parallel consolidation, minimizing the programmer's need to concern about parallel architecture details and developing an algorithm with a high degree of parallelism (DORBAND; ABURDENE, 2002).

6.1.2 STEADY

Dempster *et al.* (2005) developed a system that facilitates the visualization of parallel databases. As a result, STEADY can show students the effects of system performance from different software and hardware configurations, as shown in Figure 55.

Figure 55 – Response time viewer



Source: Dempster *et al.* (2005).

Students can visualize, for example, bottlenecks, response times, utilization, throughput for complex queries through the tool's capabilities to performance prediction. According to the authors, providing access to a complete parallel database structure is very expensive, and the process takes a long time to complete. Thus, the developed tool has been used in the last three years to support the teaching of parallel programming (DEMPSTER *et al.*, 2005).

The tool can complement the learning by students to understand the performance factors that affect the parallel database system. After surveying, the results indicated that the laboratory session had increased students' understanding (DEMPSTER *et al.*, 2005).

6.1.3 A Grid Portal

Grid Portal (TOURINO *et al.*, 2005) is a web interface developed (Figure 56) for undergraduate students to send their parallel programs to a grid containing several computers (Figure 57). In addition, the system also publishes results and feedback.

Figure 56 – MPI job submission through the portal

Submit Job

Current user

Name: Jacobo Tarrío Barreiro
Session expiry: in 00:53:12
[Logout](#)

Menu

- [List resources](#)
- [Submit job](#)
- [List jobs](#)
- [File transfer](#)
- [View profile](#)
- [System status](#)
- [Course links](#)

Submit a job

Job name:
Host / scheduler:
Executable:
Arguments:
A value of zero (0) in the following fields means "do not set"
Number of processors:
Maximum duration: (minutes)
Maximum CPU time: (minutes)
Minimum memory: (MB)
Maximum memory: (MB)
Options: ☒ Submit parallel job using MPI
☒ Send e-mail when job is finished
☐ Wait for job output

Source: Tourino *et al.* (2005).

Figure 57 – Supercomputing resources available for the students

Computer	Location	Programming	Short description
PC Network (sky.des.udc.es)	DES	MPI (MPICH)	10 PCs (PIII 800 MHz), Fast-Ethernet network
Self-made SCI cluster (muxia.des.udc.es)	DES	MPI (ScaMPI)	10 racked dual-processor nodes (PIV Xeon 1.8GHz), SCI network
Compaq Beowulf cluster (bw.cesga.es)	CESGA	MPI (MPICH-GM) HPF	16 racked single-processor nodes (PIII 1GHz), Myrinet 2000 network
SGI Origin 200 (silgar.cecafi.fi.udc.es)	SCS	OpenMP	ccNUMA multiprocessor (4 MIPS R10000 180 MHz)
HP Superdome (sd.cesga.es)	CESGA	OpenMP	cluster of 2 64-processor ccNUMA systems (Intel Itanium2 1.5 GHz)
Compaq HPC320 (sc.cesga.es)	CESGA	OpenMP/MPI hybrid	cluster of 8 4-processor SMP nodes (Alpha EV68 1 GHz)

Source: Tourino *et al.* (2005).

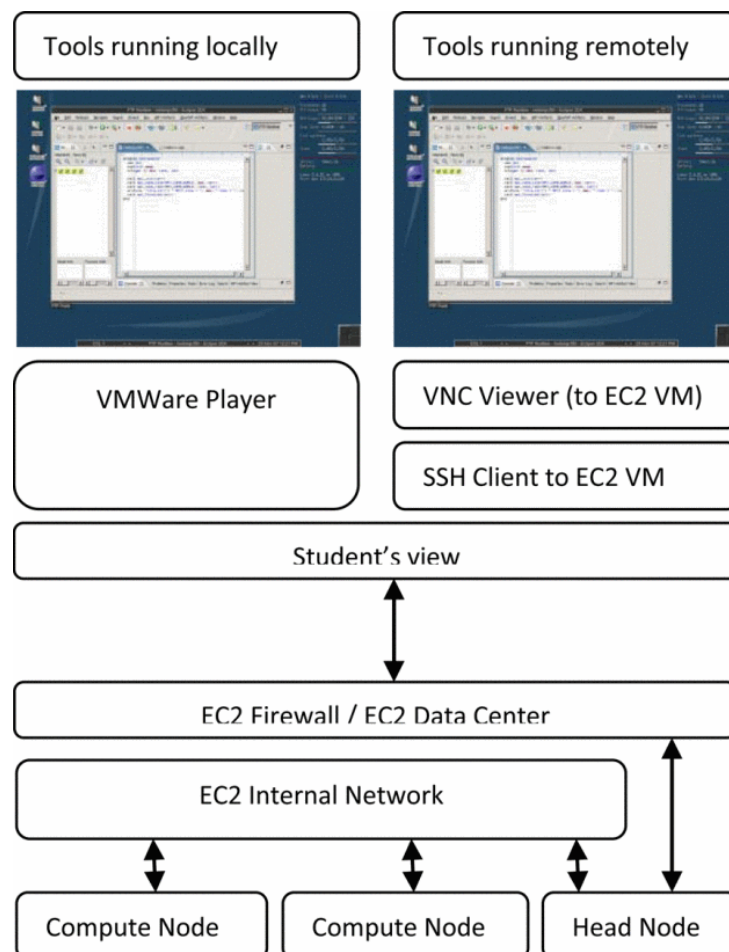
Students provided positive feedback on the design and functionality of the web interface. They pointed out that it is unnecessary to access the machine to run programs, and the system is always available online. For this, they only had a browser to access the computational grid (TOURINO *et al.*, 2005).

6.1.4 StarHPC

StarHPC (IVICA; RILEY; SHUBERT, 2009) is a solution that pre-packages into an Amazon EC2 virtual machine image the scripts used by an administrator and cluster access for students to support parallel programming teaching. This system is used for the Massachusetts Institute of Technology courses and requires a cluster infrastructure for students to run a parallel program.

As Figure 58 shows, StarHPC corrects the problems traditionally associated with the support of classroom computing resources by providing a dedicated, dynamic on-demand computing cluster hosted by Amazon's EC2 web service (IVICA; RILEY; SHUBERT, 2009).

Figure 58 – StarHPC solution architecture



Source: Ivica, Riley and Shubert (2009).

StarHPC architecture can be adopted without increasing high costs in other educational institutions for parallel programming; however, the institution needs to have a dedicated computing cluster for this purpose (IVICA; RILEY; SHUBERT, 2009).

6.1.5 Pilot Library

The Pilot library (GARDNER; CARTER, 2014) can offer parallel programming with MPI enjoyable and teach programming principles with message passing in a simple process and channel-based Multiple Program Multiple Data (MPMD) model. User-defined architecture is applied at runtime, with extensive diagnostics for all types of errors using the built-in deadlock detector.

Figure 59 shows the entire Pilot API, including only 24 functions. The API allows a single Pilot call to execute multiple MPI calls and facilitates variable-length data communication. Even a format code automatically allocates an array of suitable lengths to contain the input data (GARDNER; CARTER, 2014).

Figure 59 – Pilot API

Configuration Phase	Execution Phase
PI_Configure	PI_StartAll
PI_CreateProcess	PI_StopMain
PI_CreateChannel	PI_Write
PI_CreateBundle	PI_Read
PI_CopyChannels	PI_ChannelHasData
Utilities	PI_Select
PI_Set/GetName	PI_TrySelect
PI_GetBundleChannel	PI_Broadcast
PI_GetBundleSize	PI_Scatter
PI_Start/EndTime	PI_Gather
PI_Abort	PI_Reduce
PI_SetCommWorld (for running IMB benchmarks)	

Source: Gardner and Carter (2014).

Typical execution of clustered programs provides low visibility for troubleshooting and error debugging. The Pilot library can help students by delivering extensive verification and diagnosis of usage issues through an integrated deadlock detector (GARDNER; CARTER, 2014).

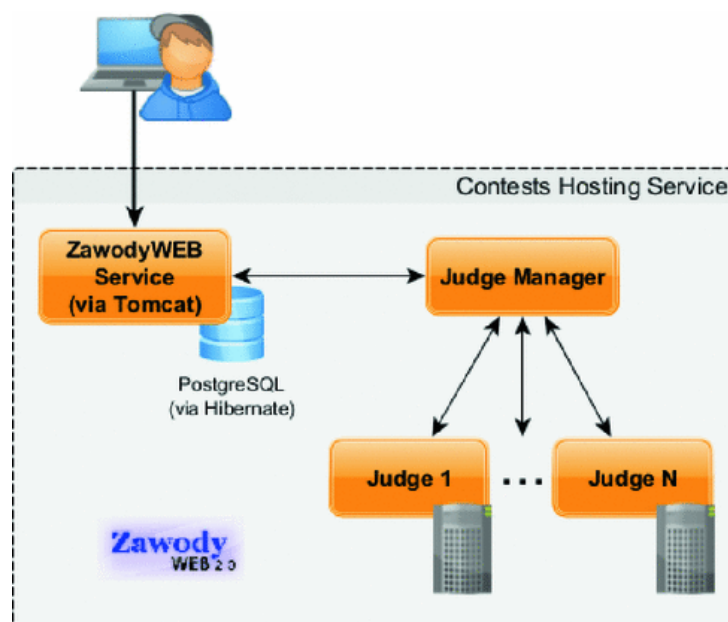
The library was used in a course at the University of Guelph, including 12-week topic programming, organization (assignments, semester project, final exam), textbook, and parallel programming platforms (GARDNER; CARTER, 2014).

6.1.6 ZawodyWeb System

The ZawodyWeb (NOWICKI *et al.*, 2015) system was initially developed for students to submit programs for online validation and later was extended to support parallel programs written in different programming paradigms. With this, the ZawodyWeb system allows the student the possibility of running problems in production systems with thousands of cores, hiding all the complexity of large multiprocessor computers.

Figure 60 shows the system architecture, created in the Java language, using the Spring Framework and JavaServer Faces libraries (Facelets, Richfaces, Restfaces, etc.). The system is hosted on the Apache/Tomcat server. The PostgreSQL database is used for the data store, and Hibernate maps Java objects to the data (NOWICKI *et al.*, 2015).

Figure 60 – The architecture of the ZawodyWeb system



Source: Nowicki *et al.* (2015).

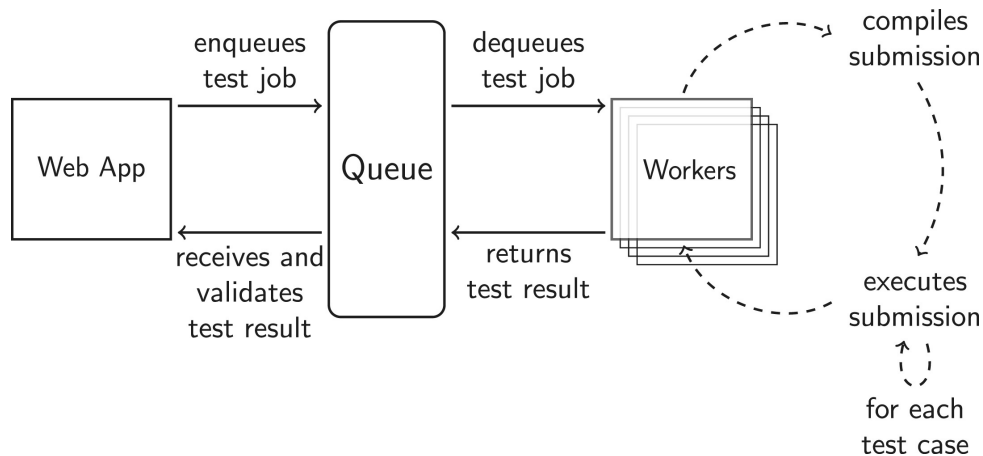
The system was used at the University of Warsaw with undergraduate students in a parallel programming course. The tool's evaluation highlighted the number of submissions by students to complete a given parallel program (NOWICKI *et al.*, 2015).

6.1.7 SAUCE

Hundt, Schlarb and Schmidt (2017) discuss a selection of known parallel algorithms based on C++ 11, OpenMP, MPI, and CUDA threads that can be applied to teach parallel computing or HPC lecture using a unified framework for automated source code evaluation named SAUCE (Automatic Code Assessment System). It is a web system that allows the submission and execution of parallel programs.

Figure 61 shows a schematic overview of the distributed architecture. The system uses a task queue to submit the compilation and execution of test jobs for several workers (HUNDT; SCHLARB; SCHMIDT, 2017).

Figure 61 – Queuing system workflow



Source: Hundt, Schlarb and Schmidt (2017).

The infrastructure has a dedicated web server where the SAUCE web application running and a lightweight queuing system to run tests on worker nodes instead of the host system. A test job is a contiguous unit of work consisting of repeated compilation and execution, one for each defined test case (HUNDT; SCHLARB; SCHMIDT, 2017).

SAUCE has an integrated plagiarism search system that uses all submissions based on the Jaccard index and compares them with students' other codes. In addition, the system has an unlimited number of requests that students can make in a given period, providing immediate feedback for corrective exercises (HUNDT; SCHLARB; SCHMIDT, 2017).

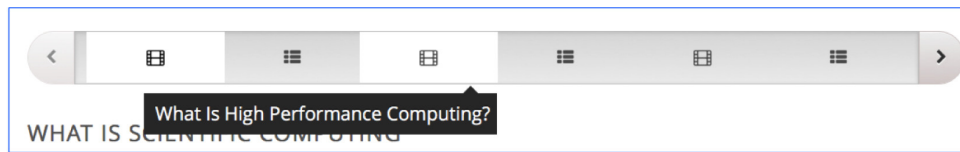
6.1.8 SPOC

Small Private Online Course (SPOC) (MULLEN *et al.*, 2017) presents the design of an HPC course to undergraduate students, precisely parallel programming, and makes it available through MOOCs (Massively Open Online Courses). The theoretical content was developed for students through the usual HPC cases and strategies for developing parallelism.

Figure 62 shows the student interface, where units flow from one tab to another with the mouse button click. For example, a student navigates to a course section and then moves to video, text, illustrations, questions, etc (MULLEN *et al.*, 2017).

The authors also highlight that learning HPC concepts, algorithms, and techniques requires practice in a parallel computing system. Thus, the main challenge in converting SPOC into a MOOC is providing thousands of students access to HPC resources. In addition to the

Figure 62 – Screenshot of Open edX navigation bar



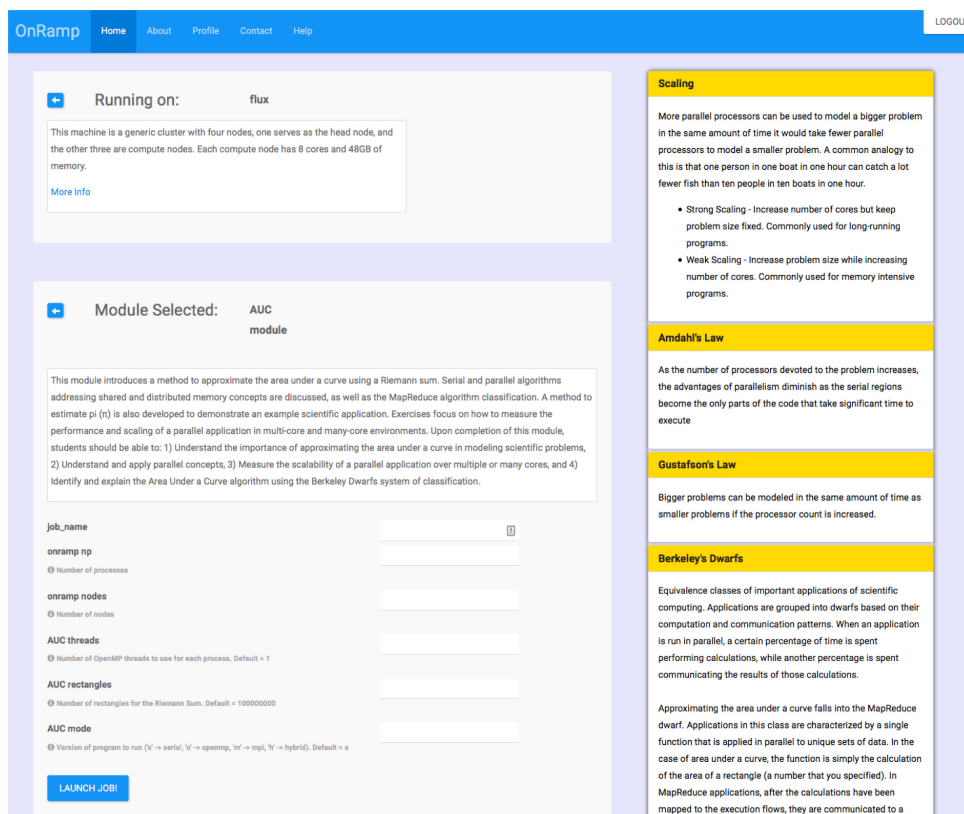
Source: Mullen *et al.* (2017).

theory, the work also presents how programs are executed through a scheduler. For execution by the terminal, the student must know Linux (MULLEN *et al.*, 2017).

6.1.9 OnRamp

OnRamp (FOLEY *et al.*, 2017) is an open-source system that provides parallel computing concepts on day one, such as scaling, decomposition, and parallel architectures through a web portal. The tool abstracts the details of the complex ecosystem of software and infrastructure present in PDC practices. The system is based on interactive modules allowing the exploration of PDC concepts using different Parallel Compute Environments (PCEs). The web system provides the submission of parallel algorithms and the visualization of results (Figure 63).

Figure 63 – Job launch in a three-node cluster



Source: Foley *et al.* (2017).

There are two REST interfaces to provide these features: one for the Web Interface to assign a request to the server and one for the Server to assign a request to the PCE Service (clusters, clouds, and virtual clouds). The PCE service creates a directory structure for managing users, modules, and jobs. Each user has a unique directory (FOLEY *et al.*, 2017).

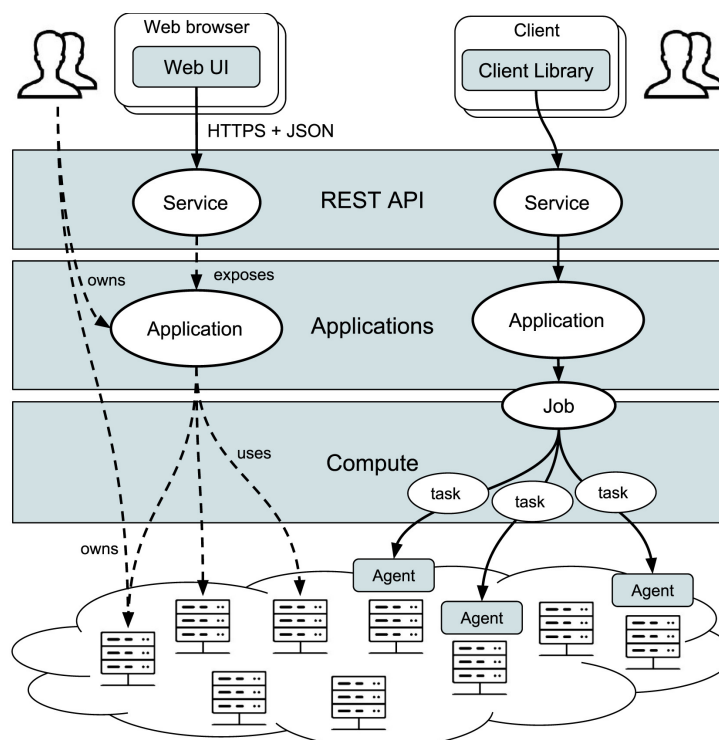
The tool was evaluated in a computer architecture course, with 19 out of 21 students present. The study's main objective was to explore the number of information users wanted to display in the web interface when interacting with OnRamp. As a result, two designs for the Web Interface were implemented. Unfortunately, the data collected for the preferred method choice by students was not statistically significant impact on their learning (FOLEY *et al.*, 2017).

6.1.10 Everest

Everest (SUKHOROSLOV, 2018) is a web-based system built to facilitate the submission of MPI programs by students and lecturers, providing feedback and result of the program execution. Figure 64 shows the architectural design of the tool.

First, Everest creates a new job for each request, consisting of one or more computational tasks generated by the application according to the job entries. Next, the platform on user-specified computing resources executes the jobs. Finally, the completed tasks are sent back to the application and are used to produce work outputs or new jobs, if necessary (SUKHOROSLOV, 2018).

Figure 64 – High-level architecture of Everest



Source: Sukhoroslov (2018).

The submission form is shown in Figure 65 and includes the entries that a student must specify. Some entries can have the default values predefined and also can be configured as mandatory or optional. For example, a user can specify a custom job name and enable email notification when the job is complete (SUKHOROSLOV, 2018).

Figure 65 – Submit form of generic service for running MPI programs

MPI

About Parameters **Submit Job** Discussion

Job Name

Program [+ Add file...](#)
*MPI program as a single *.c or *.cpp file*

Arguments
Command line arguments to pass to the program

Files [+ Add item](#)
Additional input files that are used by the program (optional)

Nodes
Number of cluster nodes to run the program on (maximum is 12)

Processes per Node
Number of MPI processes to run on each node (maximum is 12)

Wall Time
Required wall-clock time in seconds (current limit is 3 minutes)

Email Notification ☐ Send me email when the job completes
 Your job will be automatically shared with: @pdc-instructors

[▶ Submit](#)

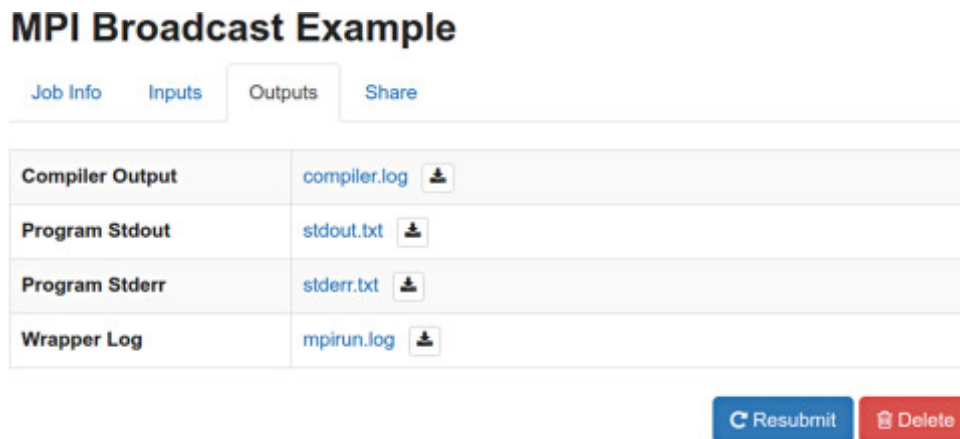
Source: Sukhoroslov (2018).

After the work submission, the student is redirected to the work page that displays dynamically updated information about the task status. Figure 66 contains a screen capture of the completed work for the MPI service (SUKHOROSLOV, 2018).

In 2017, the author conducted a survey of students from the Parallel and Distributed Computing course at the Yandex School of Data Analysis (YSDA, Moscow, Russia) to evaluate the approach presented in terms of student satisfaction. 87 students participated in the survey of 110 enrolled students (SUKHOROSLOV, 2018).

The level of convenience of running programs via the provided generic execution services was rated on a five-point scale as 5 or 4 by 80% of students (5–57%, 4–23%, 3–17%, 2–3%). The level of convenience of testing solutions via the provided assignment evaluation

Figure 66 – Results of completed job for MPI service



Source: Sukhoroslov (2018).

services was rated on the same scale as 5 or 4 by 74% of students (5–37%, 4–37%, 3–18%, 2–8%) (SUKHOROSLOV, 2018).

The survey results indicate that the approach presented is feasible and well received by students. However, the lower rating of assignment evaluation services can be explained by the remaining difficulties in debugging defective programs (SUKHOROSLOV, 2018).

6.1.11 Let's HPC

Let's HPC (CHAUDHURY *et al.*, 2018a) web platform was developed for teaching parallel programming to facilitate the analysis process of HPC/PDC programs. The platform's tools automate the compilation and execution of code and data collection. Figure 67 contains a conceptual scheme that describes the system flow and how users filter data according to their requirements.

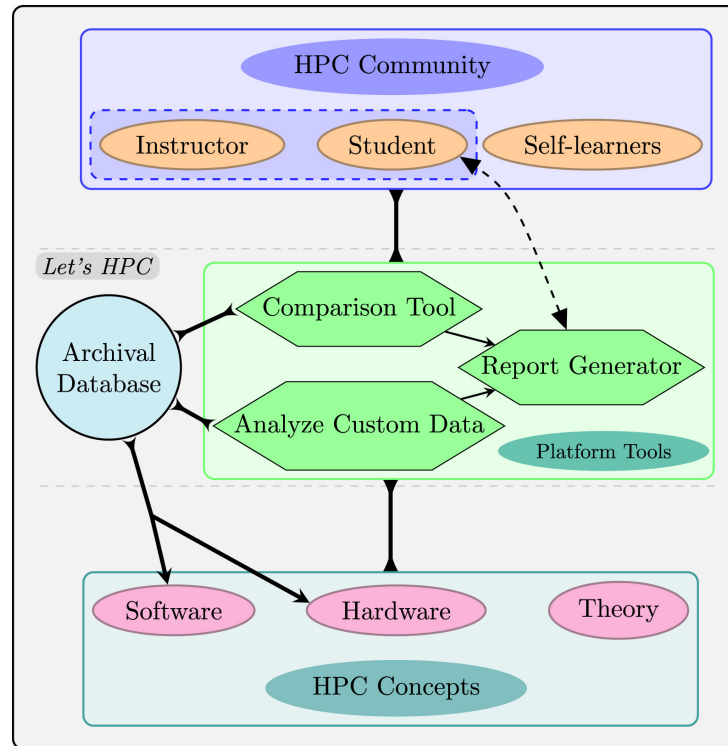
Each tool interacts with the database that uses these data to build charts used to analyze performance. Thus, the user can compare and understand the importance of various deterministic and non-deterministic software and hardware factors that affect the parallels' performance (CHAUDHURY *et al.*, 2018a).

As shown in a screenshot in Figure 68, the user first selects a category and a problem inside this category. After the student chooses the problem, the shared, distributed, or heterogeneous architecture is selected (CHAUDHURY *et al.*, 2018a).

Then, the site gets data from the database and offers three options (compare approaches, machines, or programming environments). The students can select several instances to compare methods, such as matrix multiplications, for example (CHAUDHURY *et al.*, 2018a).

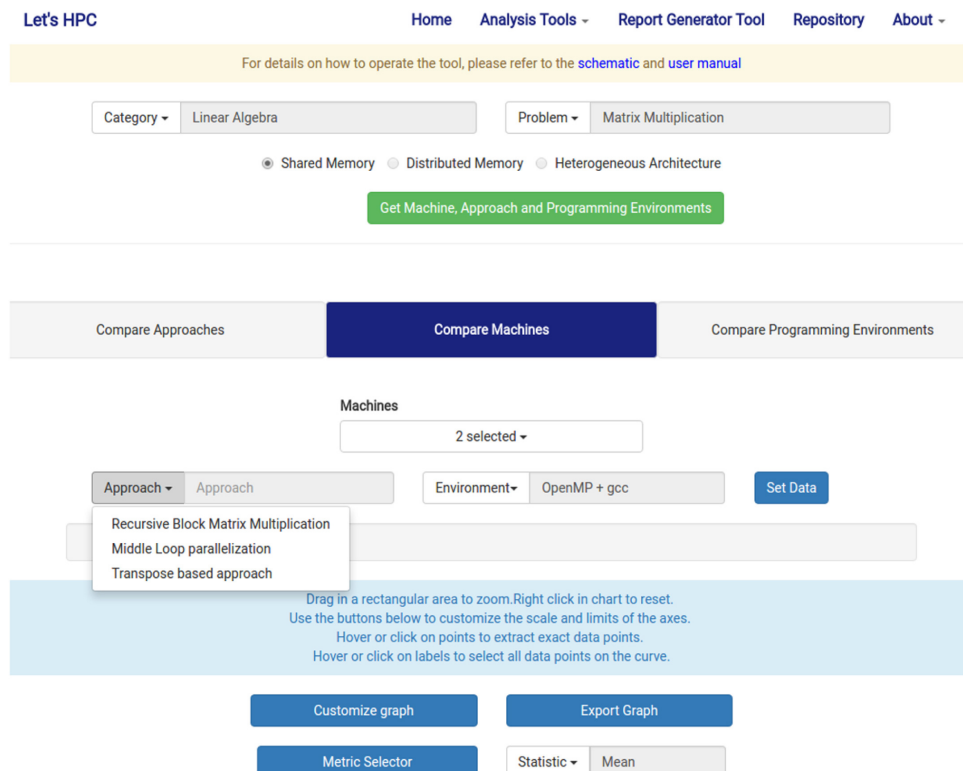
The user can generate graphs from the data retrieved from the instance selection. Furthermore, the data can be adjusted according to the problem size, allowing the user to adjust the plot

Figure 67 – The Let's HPC platform



Source: Chaudhury *et al.* (2018a).

Figure 68 – A screenshot of how the data filtering process is implemented on the Let's HPC platform



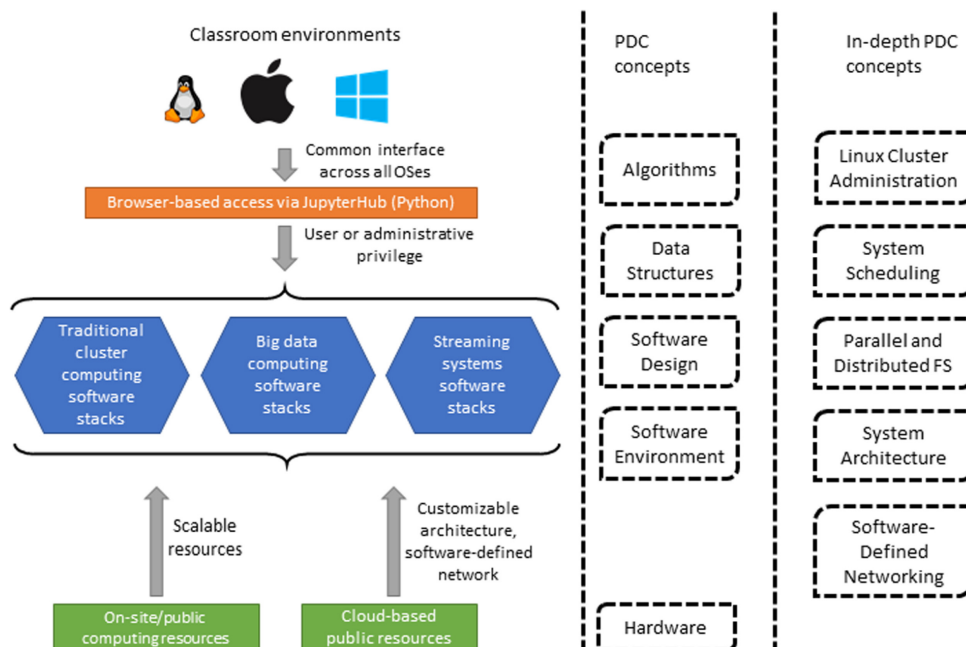
Source: Chaudhury *et al.* (2018a).

according to their preferences (CHAUDHURY *et al.*, 2018a).

6.1.12 Palmetto/JupyterHub

Palmetto Supercomputer (NGO *et al.*, 2018a) features multiple remote and local computing capabilities grouped in one framework to support graduate-level education in parallel and distributed computing. This structure, shown in Figure 69, uses Clemson University’s research cluster, the Palmetto Supercomputer (Palmetto). For advanced PDC concepts, educational modules are developed using CloudLab, a public research computing testbed (RICCI; EIDE; TEAM, 2014).

Figure 69 – A unifying classroom computing environment

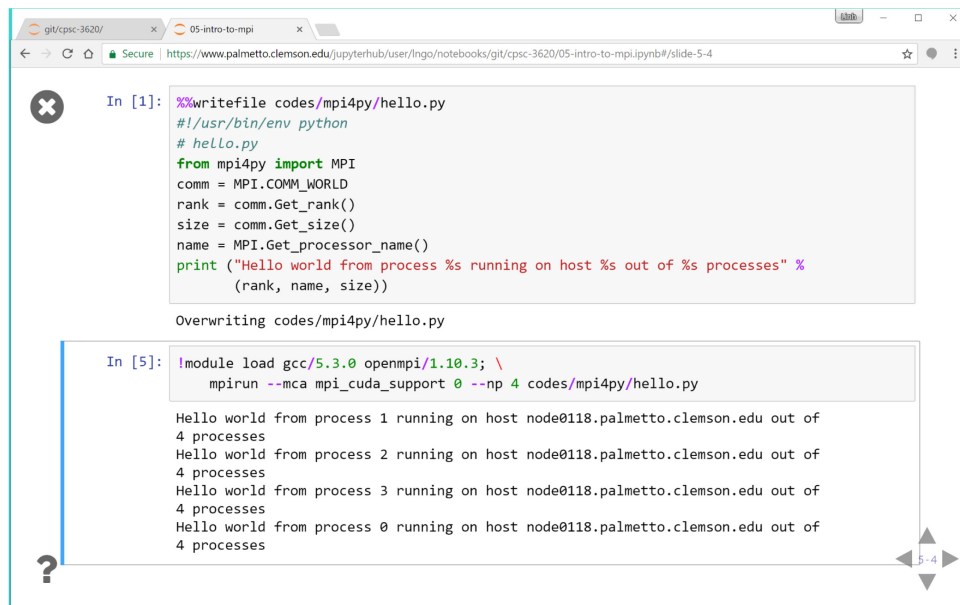


Source: NGO *et al.* (2018a).

The grouping of these features allows teaching PDC, system, and architectural concepts through a standard browser-based interface (JupyterHub, in Figure 70) and a single programming environment (Python and its supported libraries). In this way, students and lecturers can focus more on PDC principles and less on technical aspects of native languages for different platforms (NGO *et al.*, 2018a).

According to the authors, besides interactive lessons, having MPI Python code and classes inside a Jupyter notebook allows instructors to create faster exercises for students. In addition, feedback from instructors and students using the platform has been positive due to new advantages such as reduced technical support required, simplified syntax, and the ability to create lesson slides with a coding interface built into Jupyter notebooks (NGO *et al.*, 2018a).

Figure 70 – Running a simple parallel program on Jupyter Notebook



```

In [1]: %%writefile codes/mpi4py/hello.py
        #!/usr/bin/env python
        # hello.py
        from mpi4py import MPI
        comm = MPI.COMM_WORLD
        rank = comm.Get_rank()
        size = comm.Get_size()
        name = MPI.Get_processor_name()
        print ("Hello world from process %s running on host %s out of %s processes" %
              (rank, name, size))

Overwriting codes/mpi4py/hello.py

In [5]: !module load gcc/5.3.0 openmpi/1.10.3; \
        mpirun --mca mpi_cuda_support 0 --np 4 codes/mpi4py/hello.py

Hello world from process 1 running on host node0118.palmetto.clemson.edu out of
4 processes
Hello world from process 2 running on host node0118.palmetto.clemson.edu out of
4 processes
Hello world from process 3 running on host node0118.palmetto.clemson.edu out of
4 processes
Hello world from process 0 running on host node0118.palmetto.clemson.edu out of
4 processes

```

Source: [Ngo et al. \(2018a\)](#).

6.2 Final Considerations

After presenting the 12 tools in the previous sections, we identified the advantages and principal features these systems offer for teaching PDC. Below we describe what each attribute means, and in [Chart 24](#), we present a summary of the characteristics found in the related work.

- **Works without cluster structure:** the tool allows students to run the parallel programs in a distributed-memory MIMD architecture without a cluster infrastructure.
- **Works without a previous configuration:** it is not necessary to undergo a configuration step before use.
- **Graphical interface:** whether the tool has a graphical interface.
- **Source code editor:** the system's web interface that provides source code editor.
- **Multiple compilers:** it allows the students to use several compiled source codes when executing their programs.
- **Consolidated results:** tool displays results of computations.
- **Report generator:** the tool provides usage reports.
- **Portable:** it can be copied and used without extra installation requirements, by means of the virtualization through containers.
- **Automatic integration:** it is able to recognize and add new nodes in the cluster.

- **Code smell:** if the tool includes source code analysis to indicate its quality (MANTYLA; VANHANEN; LASSENIUS, 2003) and (FERNANDES *et al.*, 2016).
- **Available online:** if the tool is available via web browser.
- **Automatic correction:** whether the tool supports automatic program correction.
- **Queue system:** if it has a queuing system, preventing an executing program from interfering with another or that students compete for resources simultaneously.
- **Run-time limit:** if the tool has a runtime limit to avoid deadlocks.
- **Feedback:** whether the system allows the lecturer to provide feedback for students as comments and grades.
- **Works without internet connection:** can be used without an internet connection (on a student computer).
- **Orchestration:** the tool manager can add or remove nodes for the execution of programs.
- **Lightweight resources:** if the tool does not require heavy computational resources to run, such as clusters and servers.
- **Uses virtualization:** it shows that the tool uses virtualization to create its infrastructure.
- **Heterogeneous computing:** it refers to systems that use more than one kind of processor or cores, such as multicore CPUs, clusters, GPUs, FPGAs, among others.
- **Validation/evaluation:** if the tool has been formally validated and/or evaluated.

The last column in [Chart 24](#) offers the comparison with Iguana. **The usability tests are one of the attributes neglected in the 12 tools found in the related work.** Only two papers present validation or evaluation (DORBAND; ABURDENE, 2002; HUNDT; SCHLARB; SCHMIDT, 2017). Although validated with students, Steady, Grid Portal, StarHPC, Pilot Library, ZawodyWeb, SPOC, and OnRamp do not describe how evaluations were precisely measured or conducted (DEMPSTER *et al.*, 2005; IVICA; RILEY; SHUBERT, 2009; GARDNER; CARTER, 2014; MULLEN *et al.*, 2017; TOURINO *et al.*, 2005; NOWICKI *et al.*, 2015; FOLEY *et al.*, 2017).

Everest, Let's HPC, and Palmetto, performed evaluations of the systems with the students' previous experiences in developing and executing clustered parallel programs (SUKHOROSLOV, 2018; CHAUDHURY *et al.*, 2018b; NGO *et al.*, 2018b). **None of the related works found,** presented in [Chart 24](#), based the evaluation of the tool on widely used and standardized surveys for usability testing, such as the SUS (BROOKE, 1996) and the QUIS (CHIN; DIEHL; NORMAN, 1988).

Chart 24 – Tools for teaching parallel programming

PAPER	aCe	STEADY	Grid Portal	StarHPC	Pilot library	ZawodyWeb	SAUCE	SPOC	OnRamp	Everest	Let's HPC	Palmetto	Iguana
Year of Publication	2002	2005	2005	2009	2014	2015	2016	2017	2017	2018	2018	2018	2021
Works without cluster structure	-	o	o	o	-	o	o	-	o	o	-	o	x
Works without a previous configuration	-	o	o	o	-	o	o	-	o	o	-	o	x
Graphical interface	-	x	x	x	-	x	x	x	x	x	x	x	x
Source code editor	-	o	o	x	-	o	o	-	o	o	-	o	x
Multiple compilers	-	o	x	x	-	x	x	-	x	x	-	x	x
Consolidated results	-	x	o	x	-	o	x	-	o	x	-	x	x
Report generator	-	x	o	o	-	o	o	-	o	o	-	o	x
Portable	-	o	o	o	-	o	o	-	o	o	-	o	x
Automatic integration	-	o	o	o	-	o	o	-	o	o	-	o	x
Code smell	-	o	o	o	-	o	o	-	o	o	-	o	x
Available online	-	o	x	o	-	x	x	-	x	x	-	x	x
Automatic correction	-	o	o	o	-	x	o	-	o	o	-	x	x
Queue system	-	o	x	o	-	x	x	-	o	x	-	o	x
Run-time limit	-	o	o	o	-	o	o	-	o	x	-	o	x
Feedback	-	x	o	o	-	x	x	-	o	o	-	x	x
Works without Internet connection	-	o	o	o	-	o	o	o	o	o	o	o	x
Orchestration	-	o	o	o	-	o	o	-	o	o	-	o	x
Lightweight resources	-	o	o	o	-	o	o	-	o	o	o	o	x
Uses virtualization	-	o	o	o	-	o	o	-	o	o	-	o	x
Heterogeneous computing	-	o	o	o	-	o	x	-	x	x	-	x	x
Validation/evaluation	-	x	x	x	-	x	o	-	x	x	x	x	x

Source: Elaborated by the author.

There are other tools for submitting student programs, in addition to those shown in [Chart 24](#), that can be modified for teaching parallel programming, even not the primary purpose. Mooshak, for example, is a system for managing programming contests on the Web, including answering questions for clarification, automatic judgment, and re-evaluation of programs, among other features.

In addition to Mooshak, two other tools for programming contests (PC2 and BOCA), although not created for teaching parallel programming, are also used for this purpose ([MARTINS *et al.*, 2020](#)). However, these systems may not be proper for this type of teaching. [Grossman *et al.* \(2017b\)](#) mention that Mooshak has web interface limitations that impact the tool's adherence to teaching parallel programming. PC2 presents the same difficulty. According to ([MARTINS *et al.*, 2020](#)), compared to Mooshak, PC2 has limited usability, and the configuration is moderately complicated.

Many educational institutions have limited financial resources for infrastructure acquisition. As shown, **all the mentioned tools require an adjacent infrastructure, cluster, or even cloud services** (e.g., Amazon EC2) to run parallel programs. This cost and infrastructure complexity can make this type of education unfeasible.

In addition to infrastructure resources, the tools also require specialized labor, maintenance, and management costs. Thus, developing a system that uses minimal resources is a valuable requirement; because it allows students to run parallel programs on their computers without the need for clusters, in a simple VM, for example.

Another important aspect is the accessibility of these tools to students and not just allow their use in the lab class or remote access. **None of the systems enable students to instantly download and run in a virtual machine without configuration or installation.** This portability and quickness is an essential stimulus for students adoption the Iguana.

Another fundamental challenge is the need to assist students in socioeconomic vulnerable. [Campbell *et al.* \(2014\)](#), for example, demonstrated through his equation that investing early in childhood, especially in vulnerable children, is the key to increase social development. Thus, **creating tools to assist people in this situation can contribute to their development, giving them the opportunity for learning and future professional growth**, even if late.

Furthermore, the need for the Internet to use these tools is another barrier to development. Brazil, for example, intends to invest R\$ 24 million to serve 900,000 university students without the Internet and in a situation of socioeconomic vulnerability. As such, a lightweight tool that runs on any virtual machine with 1GB RAM and 8GB of storage is decisive for resource-poor environments. To our knowledge, **Iguana is the first proposal** that has all the features present in [Chart 24](#) and is the first desktop student-focused tool because it does **not require remote access, equipment setup, or purchasing (or rental) of a parallel infrastructure** by the educational institution.

CONCLUSION AND FUTURE WORK

The infrastructure needed to support parallel programming can be a barrier to teaching at many universities, as it involves the use of parallel hardware and specialized technical team. However, it is essential to complement the theoretical material with practical programming lab since parallel programming is challenging and unknown to students trained only in sequential programming. Until now, the main alternative to teaching multiple parallel programming models (shared memory, memory passing, and CUDA) is the use of local clusters and cloud-based solutions.

However, these solutions increase costs, as acquiring local computing resources or remote, paid cloud service requires strong Internet connectivity. In addition, in these institutions, even a local cluster infrastructure, teachers often need to train and assist students in configuration and to run the exercises.

These limitations increase the difficulty of teaching parallel programming in institutions with limited financial resources. Moreover, even when resources are available, the lecturer must spend an excessive amount of time configuring and allow access to students to the infrastructure before teaching about parallel programming. On the other hand, the student could run all these services on his notebook; however, the installation and configuration complexity will be difficult.

Thus, this thesis proposed the Iguana software tool to provide a local-virtual infrastructure for teaching heterogeneous parallel programming. Iguana is an open-source tool geared towards students who do not have access to this type of infrastructure, usually low-income students. The tool allows lecturers and students to create and execute parallel programs on a web interface in real-time, without needing terminals or command-line or to wait for batch processing.

In addition, Iguana can operate with no Internet connection in a simple virtual machine and run on student desktops with no costs, requiring only basic computer skills, allowing first-year undergraduates to experiment and run parallel code. We have argued that Iguana has a superset of features common to other such tools and many additional ones. A study that surveyed

both professionals and students provided evidence of usability and effectiveness.

As discussed previously, there are not many tools developed to teach parallel programming. Considering those we found, they require an expensive local or remote infrastructure to run parallel programs. Also, fewer than half are designed for heterogeneous computing, restricting practical laboratory classes. We mitigated the problems related to infrastructure, high service costs, and demand for specialized professionals by employing container-based virtualization.

Iguana not only relieves instructors of having to master infrastructure details, but it also allows hosts to automatically connect in a local cluster, allowing students to develop their parallel programs on these machines, increasing the number of available cores on the network.

Finally, we emphasized that [Chart 1](#) contains some laboratory programming subjects, as described in the ACM/IEEE curriculum guidelines. Iguana can address all topics, except for Cloud Computing, which requires process migration and resource sharing. However, these items have been scaled for future work.

7.1 Contributions

The research conducted to develop this doctoral thesis served several national and international contributions to the scientific and academic world. In addition, the literature review of parallel programming teaching in Brazil and the world identified how this teaching approach is performed in the leading Brazilian universities.

In addition to this research, we mapped the tools used for the theoretical and practical teaching of PDC; this allowed us to discover the methods applied for practical classes in the laboratory. Finally, we investigated the needed infrastructure technologies, publishing results on the performance evaluation between container volume sharing techniques.

Then, we designed and created a tool based on all the research conducted previously. Iguana can be installed on standard computers with resources accessible by an educational institution or student computer. In addition, the tool can even be installed on dedicated servers, increasing its computing processing.

For this, we used containers to virtualize all the necessary infrastructure and packages. We created a back-end that controls the cluster (Docker Swarm), executes the codes, and orchestrates the nodes. In parallel, we have developed a front-end to access virtualized resources.

Iguana can configure itself in a minimal network configuration without a professional's expertise in the area. Also, it allows the student to create, edit and execute his parallel codes directly in the browser without the need for third-party software installations.

All contributions from Iguana's development, orchestration, and professional and student evaluation were submitted for publication in Computer Science Education international journal, pointing to advances in the practical teaching of parallel programming. Also, Iguana will help

educate and spread parallel thinking.

7.2 Future Work

For future work, we can highlight the professionals' suggestions in [Chart B.1](#) and the students in [Chart B.2](#) that were not addressed in this work because of the time available to implement these features.

Among these points, we can highlight gamification's inclusion on the platform. With some changes, we can transform Iguana into a conquest tool, like Mooshask and Boca; but with additional resources, as a self-configuration cluster, no need for installation and lightweight resources.

Another implementation suggestion is to generate comparative graphs of sequential and parallel codes, allowing better visualization of the speedup and balancing the load between the cluster nodes. Also, integrate Iguana with online code repositories to open the source code in the editor direct GitHub, for example.

The tool has some limitations about the code execution by the GPU. In this case, the system needs a graphical interface to run. We intend to add NVEmulate/MCUDA ([STRATTON; STONE; HWU, 2008](#)) to emulated GPU support for institutions without this hardware.

Tutorials about the tool and HPC, such as videos and texts, can also be added. These tutorials can help the student understand the concepts and the tool operation; this type of strategy was addressed in [Mullen et al. \(2017\)](#).

Finally, Iguana was designed to work in 4 methods: localhost, local network, remote cluster, or cloud. This work implemented two strategies: localhost and local network, leaving the other two ways for future implementations.

7.3 Papers

This section lists the papers published based on the results and researches carried out to prepare this thesis.

7.3.1 Journals

- 2021 (Submitted): **Using The Iguana Software Tool to Teach Parallel Programming: Virtualization of Heterogeneous Local Infrastructures with Containers**. Computer Science Education Journal.
- 2019 **Best Paper Award** (Qualis B5): **The World Teaching of Parallel and Distributed Programming**. International Journal of Computer Architecture Education. Porto Alegre:

Sociedade Brasileira de Computação, 2019.

7.3.2 Conferences

- 2020 (Qualis A3): **Research on Parallel Computing Teaching: state of the art and future directions**, 2020 IEEE Frontiers in Education Conference (FIE), Uppsala, 2020, pp. 1-9, doi: 10.1109/FIE44824.2020.9273914.
- 2020 (Qualis B2): **Performance Evaluation of Container's Shared Volumes**, 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Porto, Portugal, 2020, pp. 114-123, doi: 10.1109/ICSTW50294.2020.00031.
- 2019 (Qualis B4): **Avaliação do Docker Volume e do NFS no Compartilhamento de Sistemas de Arquivos em Contêineres**. In: SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD), 20. , 2019, Campo Grande. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019 . p. 446-453. DOI: <https://doi.org/10.5753/wscad.2019.8690>.
- 2018 (Qualis A3): **Container-Based Performance Evaluation: A Survey and Challenges**, 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, 2018, pp. 398-403, doi: 10.1109/IC2E.2018.00075.
- 2018 (Qualis B2): **Um Panorama do Ensino de Programação Paralela e Distribuída em Universidades Brasileiras**. Anais dos Workshops do Congresso Brasileiro de Informática na Educação, [S.l.], p. 480, out. 2018. ISSN 2316-8889. doi:<http://dx.doi.org/10.5753/cbie.wcbie.2018.480>.
- 2017 (Qualis B2): **Mapeamento Sistemático do Ensino Teórico e Prático de Programação Paralela**. Anais dos Workshops do Congresso Brasileiro de Informática na Educação, [S.l.], p. 1089, out. 2017. ISSN 2316-8889. doi:<http://dx.doi.org/10.5753/cbie.wcbie.2017.1089>.

BIBLIOGRAPHY

ACM/IEEE-CS. **Computer Science Curricula 2013**. [S.l.], 2013. Available: <http://dx.doi.org/10.1145/2534860>. Citations on pages 40, 41, 42, 43, 50, and 71.

ADAMS, J.; BROWN, R.; SHOOP, E. Patterns and exemplars: Compelling strategies for teaching parallel and distributed computing to cs undergraduates. In: **2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum**. [S.l.: s.n.], 2013. p. 1244–1251. Citations on pages 60, 63, 64, 65, and 66.

ADAMS, J. C. Patternlets: A teaching tool for introducing students to parallel design patterns. In: IEEE. **2015 IEEE International Parallel and Distributed Processing Symposium Workshop**. [S.l.], 2015. p. 752–759. Citation on page 67.

_____. Evolving pdc curriculum and tools: A study in responding to technological change. **Journal of Parallel and Distributed Computing**, 2021. ISSN 0743-7315. Available: <https://www.sciencedirect.com/science/article/pii/S0743731521001490>. Citations on pages 65, 70, and 83.

ADAMS, J. C.; BROWN, R.; MATTHEWS, S. J.; SHOOP, E. Teaching pdc in the time of covid: Hands-on materials for remote learning. In: **2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.: s.n.], 2021. p. 342–349. Citations on pages 44, 70, 71, and 72.

Advanced Clustering Technologies. **Download Our HPC Pricing Guida**. 2021. Available: <https://www.advancedclustering.com/the-cost-of-hpc>. Accessed: 11/02/2021. Citation on page 70.

ALOISIO, G.; CAFARO, M.; EPICOCO, I.; QUARTA, G. Teaching high performance computing parallelizing a real computational science application. In: SUNDERAM, V. S.; ALBADA, G. D. van; SLOOT, P. M. A.; DONGARRA, J. J. (Ed.). **Computational Science – ICCS 2005**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 10–17. ISBN 978-3-540-32114-9. Citation on page 127.

AMEIRI, F. A.; SALAH, K. Evaluation of popular application sandboxing. In: **2011 International Conference for Internet Technology and Secured Transactions**. [S.l.: s.n.], 2011. p. 358–362. Citation on page 77.

ANGULAR. **Server-side rendering (SSR) with Angular Universal**. 2021. Available: <https://angular.io/guide/universal>. Accessed: 05/02/2021. Citation on page 94.

ARROYO, M. Teaching parallel and distributed computing to undergraduate computer science students. In: **2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum**. [S.l.: s.n.], 2013. p. 1297–1303. Citations on pages 60, 63, 64, 65, and 66.

Australian National University. **Parallel Systems**. 2018. Available: <https://programsandcourses.anu.edu.au/course/comp4300>. Accessed: 18/05/2018. Citation on page 53.

BACHIEGA, N. G. **Shell scripts and results used for the creation of the paper Performance Evaluation of Container Share Volumes for ITEQS 2020**. 2021. Available: <<https://github.com/naylor/ITEQS>>. Accessed: 20/04/2021. Citation on page 86.

Bachiega, N. G.; de Souza, P. S. L.; Bruschi, S. M.; de Souza, S. d. R. S. Performance evaluation of container's shared volumes. In: **2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.: s.n.], 2020. p. 114–123. Citations on pages 170 and 171.

BALDWIN, M. E.; ZHU, X.; SMITH, P. M.; HARRELL, S. L.; SKEEL, R.; MAJI, A. Scholar: A campus hpc resource to enable computational literacy. In: **2016 Workshop on Education for High-Performance Computing (EduHPC)**. [S.l.: s.n.], 2016. p. 25–31. Citations on pages 34, 35, 62, 63, 65, and 66.

BANGOR, A.; KORTUM, P.; MILLER, J. Determining what individual sus scores mean: Adding an adjective rating scale. **J. Usability Studies**, Usability Professionals' Association, Bloomington, IL, v. 4, n. 3, p. 114–123, May 2009. ISSN 1931-3357. Citation on page 106.

BAYSER, M. de; CERQUEIRA, R. Integrating mpi with docker for hpc. In: IEEE. **2017 IEEE International Conference on Cloud Engineering (IC2E)**. [S.l.], 2017. p. 259–265. Citation on page 68.

BESERRA, D.; MORENO, E. D.; ENDO, P. T.; BARRETO, J.; SADOK, D.; FERNANDES, S. Performance analysis of lxc for hpc environments. In: **2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems**. [S.l.: s.n.], 2015. p. 358–363. Citation on page 74.

BRANCO, A.; MOURA, A. L. d.; RODRIGUEZ, N.; ROSSETTO, S. Teaching concurrent and distributed computing – initiatives in rio de janeiro. In: **2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum**. [S.l.: s.n.], 2013. p. 1318–1323. Citations on pages 60, 64, and 66.

BROOKE, J. Sus: a “quick and dirty” usability. **Usability evaluation in industry**, v. 189, 1996. Citations on pages 106 and 142.

_____. Sus: a retrospective. **Journal of usability studies**, Usability Professionals' Association Bloomington, IL, v. 8, n. 2, p. 29–40, 2013. Citation on page 106.

BURKHART, H.; GUERRERA, D.; MAFFIA, A. Trusted high-performance computing in the classroom. In: **2014 Workshop on Education for High Performance Computing**. [S.l.: s.n.], 2014. p. 27–33. Citations on pages 60, 63, 64, 65, and 66.

Cairo University. **Course Specification**. 2018. Available: <<http://fci.cu.edu.eg/userfiles/CS471%20-%20Parallel%20Processing.pdf>>. Accessed: 18/05/2018. Citation on page 53.

CAMPBELL, F.; CONTI, G.; HECKMAN, J. J.; MOON, S. H.; PINTO, R.; PUNGELLO, E.; PAN, Y. Early childhood investments substantially boost adult health. **Science**, American Association for the Advancement of Science, v. 343, n. 6178, p. 1478–1485, 2014. Citation on page 144.

CAPEL, M. I.; TOMEU, A. J.; SALGUERO, A. G. Teaching concurrent and parallel programming by patterns: An interactive ict approach. **Journal of Parallel and Distributed Computing**,

v. 105, p. 42 – 52, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. Available: <http://www.sciencedirect.com/science/article/pii/S0743731517300163>>. Citations on pages 63, 64, and 72.

CARLEY, C.; SELLS, L.; MCKINNEY, B.; ZHAO, C.; NEEMAN, H. Using a shared, remote cluster for teaching hpc. In: **2013 IEEE International Conference on Cluster Computing (CLUSTER)**. [S.l.: s.n.], 2013. p. 1–6. ISSN 1552-5244. Citations on pages 33 and 36.

Carnegie Mellon University. **Bachelors Curriculum - Admitted 2014, 2015 & 2016**. 2018. Available: <https://www.csd.cs.cmu.edu/undergraduate/bachelors-curriculum-admitted-2014-2015-2016>>. Accessed: 18/05/2018. Citation on page 53.

CASSEL, L.; CLEMENTS, A.; DAVIES, G.; GUZDIAL, M.; MCCAULEY, R.; MCGETTRICK, A.; SLOAN, B.; SNYDER, L.; TYMANN, P.; WEIDE, B. W. **Computer Science Curriculum 2008: An Interim Revision of CS 2001**. New York, NY, USA, 2008. Citation on page 33.

Center for World University Rankings. **Center for World University Rankings**. 2018. Available: <https://cwur.org/>>. Accessed: 18/04/2018. Citation on page 52.

CESAR, E.; CORTÉS, A.; ESPINOSA, A.; MARGALEF, T.; MOURE, J. C.; SIKORA, A.; SUPPI, R. Introducing computational thinking, parallel programming and performance engineering in interdisciplinary studies. **Journal of Parallel and Distributed Computing**, v. 105, p. 116–126, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. Available: <https://www.sciencedirect.com/science/article/pii/S0743731517300059>>. Citations on pages 67, 68, and 69.

CHAUDHURY, B.; VARMA, A.; KESWANI, Y.; BHATNAGAR, Y.; PARIKH, S. Let's hpc: A web-based platform to aid parallel, distributed and high performance computing education. **Journal of Parallel and Distributed Computing**, v. 118, p. 213 – 232, 2018. ISSN 0743-7315. Available: <http://www.sciencedirect.com/science/article/pii/S0743731518301205>>. Citations on pages 138, 139, and 140.

_____. Let's hpc: A web-based platform to aid parallel, distributed and high performance computing education. **Journal of Parallel and Distributed Computing**, Elsevier, v. 118, p. 213–232, 2018. Citation on page 142.

CHIN, J. P.; DIEHL, V. A.; NORMAN, K. L. Development of an instrument measuring user satisfaction of the human-computer interface. In: **Proceedings of the SIGCHI conference on Human factors in computing systems**. [S.l.: s.n.], 1988. p. 213–218. Citations on pages 110 and 142.

Christus University Center. **Information Systems Syllabus**. 2018. Available: <https://unichristus.edu.br/graduacao/sistemas-de-informacao-computacao/estrutura-curricular/>>. Accessed: 18/05/2018. Citation on page 47.

CHUNG, M. T.; QUANG-HUNG, N.; NGUYEN, M. T.; THOAI, N. Using docker in high performance computing applications. In: **2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)**. [S.l.: s.n.], 2016. p. 52–57. Citations on pages 74 and 75.

CLARKE, L.; GLENDINNING, I.; HEMPEL, R. The mpi message passing interface standard. In: **Programming environments for massively parallel distributed systems**. [S.l.]: Springer, 1994. p. 213–218. Citation on page 35.

Computing Curricula Series Report. **Computing Curricula 2020: Paradigms for Global Computing Education**. New York, NY, USA: Association for Computing Machinery, 2020. ISBN 9781450390590. Citations on pages 40 and 41.

CORBALAN, J.; DURAN, A.; LABARTA, J. Dynamic load balancing of mpi+openmp applications. In: **International Conference on Parallel Processing, 2004. ICPP 2004**. [S.l.: s.n.], 2004. p. 195–202 vol.1. ISSN 0190-3918. Citation on page 82.

CUENCA, J.; GIMÉNEZ, D. A parallel programming course based on an execution time-energy consumption optimization problem. In: **2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.: s.n.], 2016. p. 996–1003. Citations on pages 61, 63, 64, 65, 66, and 72.

CURRICULA, A. f. C. M. A. Joint Task Force on C.; SOCIETY, I. C. **Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science**. New York, NY, USA: Association for Computing Machinery, 2013. ISBN 9781450323093. Citation on page 34.

CUSATIS, C. D.; CANNISTA, R.; HAZARD, L. Managing multi-tenant services for software defined cloud data center networks. In: **2014 IEEE 6th International Conference on Adaptive Science Technology (ICAST)**. [S.l.: s.n.], 2014. p. 1–5. ISSN 2326-9413. Citation on page 77.

DAGUM, L.; MENON, R. Openmp: an industry standard api for shared-memory programming. **IEEE computational science and engineering**, IEEE, v. 5, n. 1, p. 46–55, 1998. Citation on page 35.

DEB, K. Multi-objective optimization. In: **Search methodologies**. [S.l.]: Springer, 2014. p. 403–449. Citation on page 61.

DEMPSTER, E.; WILLIAMS, M. H.; BURGER, A.; TAYLOR, H. A tool for supporting the teaching of parallel database systems. **IEEE Trans. on Educ.**, IEEE Press, v. 48, n. 2, p. 238–247, May 2005. ISSN 0018-9359. Available: <<https://doi.org/10.1109/TE.2004.842895>>. Citations on pages 129 and 142.

DOCKER. **Runtime options with Memory, CPUs, and GPUs**. [S.l.], 2017. Available: <https://docs.docker.com/engine/admin/resource_constraints/>. Citation on page 83.

_____. **Performance tuning for volume mounts**. 2019. Available: <<https://docs.docker.com/docker-for-mac/osxfs-caching/>>. Accessed: 2019.11.11. Citations on pages 85 and 86.

_____. **Docker Engine API**. 2021. Available: <<https://docs.docker.com/engine/api/>>. Accessed: 15/03/2021. Citation on page 95.

_____. **Swarm mode key concepts**. 2021. <<http://docs.docker.com/engine/swarm>>. Available: <<http://docs.docker.com/engine/swarm>>. Accessed: 02/02/2021. Citation on page 80.

DOLGOPOLOVAS, V.; DAGIENĖ, V.; MINKEVIČIUS, S.; SAKALAUŠKAS, L. Teaching scientific computing: A model-centered approach to pipeline and parallel programming with

c. **Sci. Program.**, Hindawi Publishing Corp., New York, NY, United States, v. 2015, p. 11:11–11:11, Jan. 2015. ISSN 1058-9244. Available: <<https://doi.org/10.1155/2015/820803>>. Citations on pages 61, 64, and 66.

DORBAND, J. E.; ABURDENE, M. F. Architecture-adaptive computing environment: a tool for teaching parallel programming. In: **32nd Annual Frontiers in Education**. [S.l.: s.n.], 2002. v. 3, p. S2F–S2F. Citations on pages 128 and 142.

DUA, R.; RAJA, A. R.; KAKADIA, D. Virtualization vs containerization to support paas. In: **2014 IEEE International Conference on Cloud Engineering**. [S.l.: s.n.], 2014. p. 610–614. Citations on pages 36 and 73.

EIJKHOUT, V. Teaching mpi from mental models. In: **2016 Workshop on Education for High-Performance Computing (EduHPC)**. [S.l.: s.n.], 2016. p. 14–18. Citations on pages 62, 63, 64, and 66.

EL-REWINI, H.; ABD-EL-BARR, M. **Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing)**. [S.l.]: Wiley-Interscience, 2005. ISBN 0471467405. Citation on page 39.

ETH Zurich. **252-0029-00L Parallel Programming**. 2018. Available: <<http://www.vvz.ethz.ch/Vorlesungsverzeichnis/lerneinheit.view?semkez=2018S&ansicht=ALLE&lerneinheitId=120165&lang=en>>. Accessed: 18/05/2018. Citation on page 53.

Federal Fluminense University. **Computer Science Syllabus**. 2018. Available: <http://www2.ic.uff.br/RelatorioDeDisciplinasCompleto2018_1527277785161.pdf>. Accessed: 18/05/2018. Citations on pages 46 and 47.

Federal Rural University of the Amazon. **Computer Science Syllabus**. 2018. Available: <https://novo.ufra.edu.br/images/PPC_Licenciatura_Computacao.pdf>. Accessed: 18/05/2018. Citation on page 47.

Federal Technological University of Paraná. **Computer Engineering Syllabus**. 2018. Available: <<http://www.utfpr.edu.br/curitiba/cursos/bacharelados/Ofertados-neste-Campus/engenharia-de-computacao/matriz/matriz-curricular-721-engenharia-de-computacao/view>>. Accessed: 18/05/2018. Citation on page 47.

Federal University in Amazonas. **Computer Science Syllabus**. 2018. Available: <http://icomp.ufam.edu.br/site/phocadownload/ppc_cc_2012.pdf>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Bahia. **Computer Science Syllabus**. 2018. Available: <<https://alunoweb.ufba.br/SiacWWW/ExibirEmentaPublico.do?cdDisciplina=MATA90&nuPerInicial=20072>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Campina Grande. **Computer Science Syllabus**. 2018. Available: <<http://www.computacao.ufcg.edu.br/graduacao/projeto-pedagogico>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Ceará. **Computer Science Syllabus**. 2018. Available: <<http://www.campusrussas.ufc.br/grades/projeto-pedagogico-CC.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Goiás. **Computer Science Syllabus**. 2018. Available: <<http://www.inf.ufg.br/sites/default/files/uploads/Nova%20Matriz%20CC.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Grande Dourados. **Computer Science Syllabus**. 2018. Available: <<https://portalead.ufgd.edu.br/wp-content/uploads/2013/03/PPC-LICENCIATURA-COMPUTA%C3%87%C3%83O.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Mato Grosso. **Computer Science Syllabus**. 2018. Available: <<http://sistemas.ufmt.br/ufmt.ppc/PlanoPedagogico/Download/310>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Mato Grosso do Sul. **Computer Science Syllabus**. 2018. Available: <<http://cphp.sites.ufms.br/files/2013/04/5-RESOLU-O-COEG-COC-RTR-n-520-de-28-10-2014-computa%C3%A7%C3%A3o.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Minas Gerais. **Computer Science Syllabus**. 2018. Available: <<https://ufmg.br/cursos/graduacao/2377/77220/60514>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Paraná. **Computer Science Syllabus**. 2018. Available: <<https://web.inf.ufpr.br/bcc/curriculo/grade-curricular/>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Paraíba. **Computer Science Syllabus**. 2018. Available: <https://sigaa.ufpb.br/sigaa/public/curso/ppp.jsf?lc=pt_BR&id=1626786>. Accessed: 18/05/2018. Citations on pages 46 and 47.

Federal University of Pará. **Computer Science Syllabus**. 2018. Available: <<http://computacao.ufpa.br/DocumentosPublicos/PPC%20de%20Ciencia%20da%20Computacao%202010.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Pernambuco. **Computer Science Syllabus**. 2018. Available: <https://www.ufpe.br/documents/38970/411209/ciencia_computacao_perfil_2002.pdf/09862676-8330-4642-af94-6ec9e8607a62>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Rio de Janeiro. **Computer Science Syllabus**. 2018. Available: <<https://www.siga.ufrj.br/sira/temas/zire/frameConsultas.jsp?mainPage=/repositorio-curriculo/FA9F18A7-92A4-F79B-1A98-293E97D8939B.html>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Rio Grande do Norte. **Computer Science Syllabus**. 2018. Available: <<http://arquivos.info.ufrn.br/arquivos/2014248042a1081902632bfcc77db8587/pp-bcc-2014.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Rio Grande do Sul. **Computer Science Syllabus**. 2018. Available: <<http://www.inf.ufrgs.br/site/ciencia-da-computacao/descricao/>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Rondônia. **Computer Science Syllabus**. 2018. Available: <http://www.dacc.unir.br/uploads/91919191/arquivos/3438_ppc_bcc_1179480801.pdf>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Santa Catarina. **Computer Science Syllabus**. 2018. Available: <<https://planos.inf.ufsc.br/modulos/planos/pdf.php?codigo=1374>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Santa Maria. **Computer Science Syllabus**. 2018. Available: <<http://w3.ufsm.br/prograd/index.php/documentos/12-ppcs-projetos-pedagogicos-dos-cursos-de-graduacao/40-elenco-das-disciplinas-ciencia-da-computacao>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of Sergipe . **Computer Science Syllabus**. 2018. Available: <<https://www.sigaa.ufs.br/sigaa/public/departamento/componentes.jsf;jsessionid=BA223F19700C38DED638A808008A7FAD.canario1>>. Accessed: 18/05/2018. Citation on page 47.

Federal University of São Carlos. **Computer Science Syllabus**. 2018. Available: <http://bcc2.dc.ufscar.br/wp-content/uploads/2016/02/Projeto_Pedagogico_BCC.pdf>. Accessed: 18/05/2018. Citation on page 47.

Federal University of São João del-Rei. **Computer Science Syllabus**. 2018. Available: <https://ufsj.edu.br/portal2-repositorio/File/soces/Res050CONEP2011PPCCienciasComutacao_Anexo.pdf>. Accessed: 18/05/2018. Citations on pages 46 and 47.

Federal University of Tocantins. **Computer Science Syllabus**. 2018. Available: <http://download.uft.edu.br/?d=754389e0-6bdf-4dd2-b948-00ef6e152e15:20_2011_atualizacao_do_ppc_do_curso_de_ciencia_da_computacao_4668.pdf>. Accessed: 18/05/2018. Citation on page 47.

FELTER, W.; FERREIRA, A.; RAJAMONY, R.; RUBIO, J. An updated performance comparison of virtual machines and linux containers. In: **2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)**. [S.l.: s.n.], 2015. p. 171–172. Citation on page 74.

FERNANDES, E.; OLIVEIRA, J.; VALE, G.; PAIVA, T.; FIGUEIREDO, E. A review-based comparative study of bad smell detection tools. In: **Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2016. (EASE '16). ISBN 9781450336918. Available: <<https://doi.org/10.1145/2915970.2915984>>. Citation on page 142.

FERNER, C.; WILKINSON, B.; HEATH, B. Toward using higher-level abstractions to teach parallel computing. In: **2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum**. [S.l.: s.n.], 2013. p. 1291–1296. Citations on pages 39, 60, 63, 64, 65, 66, and 72.

_____. Using patterns to teach parallel computing. In: **2014 IEEE International Parallel Distributed Processing Symposium Workshops**. [S.l.: s.n.], 2014. p. 1106–1113. Citations on pages 60, 63, 65, and 66.

FLYNN, M. J. **Flynn's Taxonomy**. 2011. Citation on page 34.

FOLEY, S. S.; KOEPKE, D.; RAGATZ, J.; BREHM, C.; REGINA, J.; HURSEY, J. Onramp: A web-portal for teaching parallel and distributed computing. **Journal of Parallel and Distributed Computing**, v. 105, p. 138 – 149, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching

Parallel, Distributed and High-Performance Computing. Available: <http://www.sciencedirect.com/science/article/pii/S0743731517300205>. Citations on pages 35, 135, 136, and 142.

Folha de São Paulo. **RUF: Ranking Universitário**. 2018. <http://ruf.folha.uol.com.br/2017/ranking-de-cursos/computacao,18/04/2018>. Available: <http://ruf.folha.uol.com.br/2017/ranking-de-cursos/computacao>. Accessed: 18/04/2018. Citation on page 45.

GARDNER, B., W. Should we be teaching parallel programming? In: **Proceedings of the 22Nd Western Canadian Conference on Computing Education**. New York, NY, USA: ACM, 2017. (WCCCE '17), p. 3:1–3:7. ISBN 978-1-4503-5066-2. Available: <http://doi.acm.org/10.1145/3085585.3085588>. Citations on pages 62, 63, and 66.

GARDNER, W. B.; CARTER, J. D. Using the pilot library to teach message-passing programming. In: **Proceedings of the Workshop on Education for High-Performance Computing**. IEEE Press, 2014. (EduHPC '14), p. 1–8. ISBN 9781479970216. Available: <https://doi.org/10.1109/EduHPC.2014.14>. Citations on pages 132 and 142.

GITHUB. **ICS - Iguana Cluster System**. 2021. Available: <https://github.com/iguana-hpc-usp/ICS>. Accessed: 15/03/2021. Citation on page 95.

GROSSMAN, M.; AZIZ, M.; CHI, H.; TIBREWAL, A.; IMAM, S.; SARKAR, V. Pedagogy and tools for teaching parallel computing at the sophomore undergraduate level. **Journal of Parallel and Distributed Computing**, v. 105, p. 18 – 30, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. Available: <http://www.sciencedirect.com/science/article/pii/S0743731517300047>. Citations on pages 62, 63, 64, 65, and 66.

_____. Pedagogy and tools for teaching parallel computing at the sophomore undergraduate level. **Journal of Parallel and Distributed Computing**, v. 105, p. 18 – 30, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. Available: <http://www.sciencedirect.com/science/article/pii/S0743731517300047>. Citation on page 144.

Harvard University. **Introduction to Distributed Computing**. 2018. Available: <https://courses.my.harvard.edu/psp/courses>. Accessed: 18/05/2018. Citation on page 53.

Hong Kong University of Science and Technology. **BEng in Computer Science**. 2018. Available: http://ugadmin.ust.hk/prog_crs/pdf/ug/comp.pdf. Accessed: 18/05/2018. Citation on page 53.

HUNDT, C.; SCHLARB, M.; SCHMIDT, B. Sauce: A web application for interactive teaching and learning of parallel programming. **Journal of Parallel and Distributed Computing**, v. 105, p. 163 – 173, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. Available: <http://www.sciencedirect.com/science/article/pii/S0743731517300060>. Citations on pages 35, 133, 134, and 142.

IBGE. **Censo Demográfico**. 2010. <http://www.censo2010.ibge.gov.br>. Available: <http://www.censo2010.ibge.gov.br>. Accessed: 12/02/2018. Citation on page 45.

Imperial College London. **CO347 Distributed Algorithms**. 2018. Available: http://intranet.ee.ic.ac.uk/electricalengineering/eecourses_t4/course_content.asp?c=CO347&s=J3. Accessed: 18/05/2018. Citation on page 53.

IVICA, C.; RILEY, J. T.; SHUBERT, C. Starhpc - teaching parallel programming within elastic compute cloud. In: **Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces**. [S.l.: s.n.], 2009. p. 353–356. ISSN 1330-1012. Citations on pages 35, 59, 63, 65, 66, 72, 131, 132, and 142.

JIANG, K.; SONG, Q. A preliminary investigation of container-based virtualization in information technology education. In: **Proceedings of the 16th Annual Conference on Information Technology Education**. New York, NY, USA: ACM, 2015. (SIGITE '15), p. 149–152. ISBN 978-1-4503-3835-6. Available: <<http://doi.acm.org/10.1145/2808006.2808021>>. Citations on pages 36 and 76.

JOINER, D. A.; GRAY, P.; MURPHY, T.; PECK, C. Teaching parallel computing to science faculty: Best practices and common pitfalls. In: **Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming**. New York, NY, USA: ACM, 2006. (PPoPP '06), p. 239–246. ISBN 1-59593-189-9. Available: <<http://doi.acm.org/10.1145/1122971.1123007>>. Citations on pages 58, 63, 64, 65, 66, and 72.

JONES, M. **Network file systems and Linux**. 2019. Available: <<https://developer.ibm.com/tutorials/l-network-filessystem>>. Accessed: 29/12/2019. Citation on page 84.

KARYPIS, G. **ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering**. [S.l.], 2017. Available: <<http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview>>. Citation on page 82.

KHANGHAHL, N.; RAVANMEHR, R. Cloud computing performance evaluation: Issues and challenges. **International Journal on Cloud Computing**, AIRCC, p. 29–41, 2013. Citation on page 74.

KIM, T.-H.; JIANG, K.; RAJPUT, V. S. Adoption of container-based virtualization in it education. In: **ASEE's 123rd Annual - Conference and Exposition**. New Orleans, LA: [s.n.], 2016. Citations on pages 34 and 36.

KRATZKE, N. Smuggling multi-cloud support into cloud-native applications using elastic container platforms. In: **Proceedings of the 7th International Conference on Cloud Computing and Services Science**. [S.l.: s.n.], 2017. p. 57–70. ISBN 978-989-758-243-1. Citation on page 79.

LI, J.; GUO, W.; ZHENG, H. An undergraduate parallel and distributed computing course in multi-core era. In: **2008 The 9th International Conference for Young Computer Scientists**. [S.l.: s.n.], 2008. p. 2412–2416. Citations on pages 34, 59, 64, and 66.

LINUX Containers - Documentation. 2016. Available: <<https://linuxcontainers.org>>. Accessed: 21/01/2017. Citation on page 78.

LIU, J. 20 years of teaching parallel processing to computer science seniors. In: **2016 Workshop on Education for High-Performance Computing (EduHPC)**. [S.l.: s.n.], 2016. p. 7–13. Citations on pages 62, 63, 64, 65, 66, and 72.

LÓPEZ, P.; BAYDAL, E. Teaching high-performance service in a cluster computing course. **Journal of Parallel and Distributed Computing**, Elsevier, v. 117, p. 138–147, 2018. Citation on page 70.

MA, H.; WANG, L.; TAK, B. C.; WANG, L.; TANG, C. Auto-tuning performance of mpi parallel programs using resource management in container-based virtual cloud. In: **2016 IEEE 9th International Conference on Cloud Computing (CLOUD)**. [S.l.: s.n.], 2016. p. 545–552. Citation on page 82.

Mackenzie Presbyterian University. **Computer Science Syllabus**. 2018. Available: <<http://up.mackenzie.br/graduacao/sao-paulo/ciencia-da-computacao/matriz-curricular-2018/>>. Accessed: 18/05/2018. Citation on page 47.

MANTYLA, M.; VANHANEN, J.; LASSENIUS, C. A taxonomy and an initial empirical study of bad smells in code. In: **International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings**. [S.l.: s.n.], 2003. p. 381–384. Citation on page 142.

MAROWKA, A. Think parallel: Teaching parallel programming today. **IEEE Distributed Systems Online**, v. 9, n. 8, p. 1–1, Aug 2008. ISSN 1541-4922. Citations on pages 34, 59, 63, 64, and 66.

MARTINS, G.; SOUZA, P. S. L. de; CONTE, D. J.; BRUSCHI, S. M. Learning parallel programming through programming challenges. In: **2020 IEEE Frontiers in Education Conference (FIE)**. [S.l.: s.n.], 2020. p. 1–9. Citation on page 144.

Massachusetts Institute of Technology. **6.816/6.836 Multicore Programming**. 2018. Available: <<https://stellar.mit.edu/S/course/6/sp14/6.816/?toolset=hidden>>. Accessed: 18/05/2018. Citation on page 53.

MOORE, S. V.; DUNLOP, S. R. A flipped classroom approach to teaching concurrency and parallelism. In: **2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.: s.n.], 2016. p. 987–995. Citations on pages 62, 63, 64, 66, and 72.

MORABITO, R.; KJÄLLMAN, J.; KOMU, M. Hypervisors vs. lightweight virtualization: A performance comparison. In: **2015 IEEE International Conference on Cloud Engineering**. [S.l.: s.n.], 2015. p. 386–393. Citation on page 73.

MOUAT, A. **Using Docker: Developing and deploying software with containers**. [S.l.]: O'Reilly Media, 2015. Citation on page 76.

MULLEN, J.; BYUN, C.; GADEPALLY, V.; SAMSI, S.; REUTHER, A.; KEPNER, J. Learning by doing, high performance computing education in the mooc era. **Journal of Parallel and Distributed Computing**, v. 105, p. 105 – 115, 2017. ISSN 0743-7315. Keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. Available: <<http://www.sciencedirect.com/science/article/pii/S0743731517300217>>. Citations on pages 134, 135, 142, and 147.

Nanyang Technological University. **Computer Science (CS) Programme**. 2018. Available: <<http://scse.ntu.edu.sg/Programmes/CurrentStudents/Undergraduate/Pages/CS.aspx>>. Accessed: 18/05/2018. Citations on pages 52 and 53.

NATH, D. **Running an MPI Cluster within a LAN**. 2020. Available: <<https://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/>>. Accessed: 05/01/2020. Citation on page 83.

National University of Singapore. **Computer Science Focus Areas for BComp (CS)**. 2018. Available: <<https://www.comp.nus.edu.sg/programmes/ug/focus>>. Accessed: 18/05/2018. Citations on pages 53, 57, and 59.

NEZU, N. Teaching parallel programming in 50 minutes. **J. Comput. Sci. Coll.**, Consortium for Computing Sciences in Colleges, USA, v. 31, n. 2, p. 18–24, Dec. 2015. ISSN 1937-4771. Available: <<http://dl.acm.org/citation.cfm?id=2831432.2831435>>. Citations on pages 61, 64, 66, and 72.

NGO, L. B.; SRINATH, A. T.; DENTON, J.; ZIOLKOWSKI, M. Unifying computing resources and access interface to support parallel and distributed computing education. **Journal of Parallel and Distributed Computing**, v. 118, p. 201 – 212, 2018. ISSN 0743-7315. Available: <<http://www.sciencedirect.com/science/article/pii/S0743731518300984>>. Citations on pages 140 and 141.

_____. Unifying computing resources and access interface to support parallel and distributed computing education. **Journal of Parallel and Distributed Computing**, Elsevier, v. 118, p. 201–212, 2018. Citation on page 142.

NICKOLOFF, J. **Docker in Action**. 1st. ed. USA: Manning Publications Co., 2016. ISBN 1633430235. Citations on pages 78 and 79.

NOWICKI, M.; MARCHWIANY, M.; SZPINDLER, M.; BAŁA, P. On-line service for teaching parallel programming. In: HUNOLD, S.; COSTAN, A.; GIMÉNEZ, D.; IOSUP, A.; RICCI, L.; REQUENA, M. E. G.; SCARANO, V.; VARBANESCU, A. L.; SCOTT, S. L.; LANKES, S.; WEIDENDORFER, J.; ALEXANDER, M. (Ed.). **Euro-Par 2015: Parallel Processing Workshops**. Cham: Springer International Publishing, 2015. p. 78–89. ISBN 978-3-319-27308-2. Citations on pages 133 and 142.

OpenMP Architecture Review Board. **OpenMP Application Program Interface**. [S.l.], 2016. Version 4.5.0. Available: <<https://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf>>. Citation on page 67.

PAHL, C. Containerization and the paas cloud. **IEEE Cloud Computing**, v. 2, n. 3, p. 24–31, May 2015. ISSN 2325-6095. Citation on page 36.

_____. Containerization and the paas cloud. **IEEE Cloud Computing**, v. 2, n. 3, p. 24–31, May 2015. ISSN 2325-6095. Citation on page 73.

Peking University. **Undergraduate Programs**. 2018. Accessed: 18/05/2018. Citations on pages 52 and 53.

PERES, S. C.; PHAM, T.; PHILLIPS, R. Validation of the system usability scale (sus): Sus in the wild. **Proceedings of the Human Factors and Ergonomics Society Annual Meeting**, v. 57, n. 1, p. 192–196, 2013. Available: <<https://doi.org/10.1177/1541931213571043>>. Citation on page 106.

Pontifical Catholic University of Chile. **Bachelor: Major in Computer Science**. 2018. Available: <<https://www.ing.uc.cl/ciencia-de-la-computacion/programas/licenciatura/>>. Accessed: 18/05/2018. Citations on pages 52 and 53.

Pontifical Catholic University of Goiás. **Computer Science Syllabus**. 2018. Available: <<http://sites.pucgoias.edu.br/cursos/cienciadacomputacao/wp-content/uploads/sites/18/2013/04/Matriz-Curricular-Ci%C3%Aancia-da-Computa%C3%A7%C3%A3o-2013.1.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Pontifical Catholic University of Paraná. **Computer Science Syllabus**. 2018. Available: <https://www.pucpr.br/escola-politecnica/wp-content/uploads/sites/4/2017/06/matriz_ciencia-da-computacao_.pdf>. Accessed: 18/05/2018. Citation on page 47.

Pontifical Catholic University of Rio de Janeiro. **Computer Science Syllabus**. 2018. Available: <<http://www.inf.puc-rio.br/wordpress/wp-content/uploads/2018/04/PPC-Ciencia-da-Computacao.pdf>>. Accessed: 18/05/2018. Citations on pages 46 and 47.

Pontifical Catholic University of Rio Grande do Sul. **Computer Science Syllabus**. 2018. Available: <<http://www.pucrs.br/politecnica/informacoes-academicas/ciencia-da-computacao/>>. Accessed: 18/05/2018. Citations on pages 46 and 47.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software-8ª Edição**. [S.l.]: McGraw Hill Brasil, 2016. Citation on page 89.

Princeton University. **COS-418, Fall 2016: Distributed Systems**. 2018. Available: <<https://www.cs.princeton.edu/courses/archive/fall16/cos418/>>. Accessed: 18/05/2018. Citation on page 53.

QIU, J.; KAMBURUGAMUVE, S.; LEE, H.; MITCHELL, J.; CALDWELL, R.; BULLOCK, G.; HAYDEN, L. Teaching, learning and collaborating through cloud computing online classes. In: **Proceedings of the 2017 Workshop on Education for High-Performance Computing**. [S.l.: s.n.], 2017. Citation on page 57.

QS World University Rankings. **Computer Science & Information Systems**. 2018. Available: <<https://www.topuniversities.com/university-rankings/university-subject-rankings/2018/computer-science-information-systems>>. Accessed: 18/04/2018. Citation on page 52.

Queensland University of Technology. **Bachelor of Information Technology (Computer Science)**. 2018. Available: <http://pdf.courses.qut.edu.au/coursepdf/qut_IN01_34130_int_cms_unit.pdf>. Accessed: 18/05/2018. Citation on page 53.

RAMEY, W. **2 Million Registered Developers, Countless Breakthroughs**. 2021. Available: <<https://blogs.nvidia.com/blog/2020/08/19/2-million-registered-developers-breakthroughs/>>. Accessed: 11/02/2021. Citation on page 33.

RICCI, R.; EIDE, E.; TEAM, C. Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications. ; **login:: the magazine of USENIX & SAGE**, USENIX Association, v. 39, n. 6, p. 36–38, 2014. Citation on page 140.

RIGHETTI, S. **Fórum Nacional de Educação Superior**. 2015. Available: <http://portal.mec.gov.br/index.php?option=com_docman&view=download&alias=17209-cne-forum-educacao-superior-2015-apresentacao-20-sabine-righetti&category_slug=marco-2015-pdf&Itemid=30192>. Accessed: 13/06/2018. Citation on page 46.

ROBERTS, E.; SHACKELFORD, R.; LEBLANC, R.; DENNING, P. J. Curriculum 2001: Interim report from the acm/ieee-cs task force. **SIGCSE Bull.**, ACM, New York, NY, USA, v. 31, n. 1, p. 343–344, Mar. 1999. ISSN 0097-8418. Available: <<http://doi.acm.org/10.1145/384266.299802>>. Citation on page 33.

RUDYY, O.; GARCIA-GASULLA, M.; MANTOVANI, F.; SANTIAGO, A.; SIRVENT, R.; VÁZQUEZ, M. Containers in hpc: A scalability and portability study in production biological simulations. In: **2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [S.l.: s.n.], 2019. p. 567–577. Citation on page 88.

RUIZ, C.; JEANVOINE, E.; NUSSBAUM, L. Performance evaluation of containers for hpc. In: **In: Hunold S. et al. (eds) Euro-Par 2015: Parallel Processing Workshops. Euro-Par 2015.** [S.l.: s.n.], 2015. p. 813–824. Citation on page 74.

SANTOS, C. M. d. C.; PIMENTA, C. A. d. M.; NOBRE, M. R. C. The pico strategy for the research question construction and evidence search. **Revista latino-americana de enfermagem**, SciELO Brasil, v. 15, p. 508–511, 2007. Citation on page 58.

SCHLARF, M.; HUNDT, C.; SCHMIDT, B. Sauce: A web-based automated assessment tool for teaching parallel programming. In: _____. **Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers.** Cham: Springer International Publishing, 2015. p. 54–65. ISBN 978-3-319-27308-2. Available: <https://doi.org/10.1007/978-3-319-27308-2_5>. Citations on pages 61, 63, 65, 66, and 72.

SHAFI, A.; AKHTAR, A.; JAVED, A.; CARPENTER, B. Teaching parallel programming using java. In: **Proceedings of the Workshop on Education for High-Performance Computing.** Piscataway, NJ, USA: IEEE Press, 2014. (EduHPC '14), p. 56–63. ISBN 978-1-4799-7021-6. Available: <<http://dx.doi.org/10.1109/EduHPC.2014.7>>. Citations on pages 60, 63, 64, 65, and 66.

SHAMSI, J. A.; DURRANI, N. M.; KAFI, N. Novelties in teaching high performance computing. In: **2015 IEEE International Parallel and Distributed Processing Symposium Workshop.** [S.l.: s.n.], 2015. p. 772–778. Citations on pages 33, 44, 61, 63, 64, 65, and 66.

SILVA, G. M. da. **Desenvolvimento de desafios para o aprendizado de programação paralela.** Bachelor's Thesis (Graduação) — Instituto de Ciências Matemáticas e de Computação - USP São Carlos, São Carlos, 2021. Citations on pages 105, 120, 121, 122, 123, and 124.

SITSYLITSYN, Y. Methods and tools for teaching parallel and distributed computing in universities: a systematic review of the literature. In: EDP SCIENCES. **ICHTML 2020: SHS Web of Conferences.** [S.l.], 2020. Citation on page 69.

STACKOVERFLOW Developer Survey Results. 2021. Available: <<https://insights.stackoverflow.com/survey/2017>>. Accessed: 15/03/2021. Citation on page 94.

Stanford University. **CS149 - Parallel Computing.** 2018. Available: <<http://scpd.stanford.edu/search/publicCourseSearchDetails.do?method=load&courseId=8254396>>. Accessed: 18/05/2018. Citation on page 53.

State University of Amazonas. **Computer Science Syllabus.** 2018. Available: <http://blogs.unama.br/sites/blogs.unama.br/files/anexo/gra-mat-0279-c_-_ciencia_da_computacao_fmn_-_estrutura_curricular.pdf>. Accessed: 18/05/2018. Citation on page 47.

State University of Campinas. **MC970/MO644 Parallel Programming.** 2018. Available: <<http://oxent2.ic.unicamp.br/node/41>>. Accessed: 18/05/2018. Citations on pages 47 and 53.

State University of Londrina. **Computer Science Syllabus.** 2018. Available: <http://www.uel.br/prograd/acolhimento/documentos/matriz_curricular/matriz_curricular_acolhimento_semana_retomada_ciencia_computacao.pdf>. Accessed: 18/05/2018. Citations on pages 46 and 47.

State University of Maringá. **Computer Science Syllabus**. 2018. Available: <<http://www.pen.uem.br/deg/apoio-aos-colegiados-aco/graduacao/cursos/PPPIformtica2018.pdf>>. Accessed: 18/05/2018. Citation on page 47.

State University of Rio de Janeiro. **Computer Science Syllabus**. 2018. Available: <http://www.dep.uerj.br/arqs/fluxogamas_cursos/ciencia_da_computacao.pdf>. Accessed: 18/05/2018. Citation on page 47.

Stellenbosch University. **Academic Programmes and Faculty Information**. 2018. Available: <<https://www.sun.ac.za/english/Documents/Yearbooks/Current/Science.pdf>>. Accessed: 18/05/2018. Citations on pages 53 and 54.

STERLING, T.; ANDERSON, M.; BRODOWICZ, M. Chapter 18 - file systems. In: STERLING, T.; ANDERSON, M.; BRODOWICZ, M. (Ed.). **High Performance Computing**. Boston: Morgan Kaufmann, 2018. p. 549–578. ISBN 978-0-12-420158-3. Available: <<https://www.sciencedirect.com/science/article/pii/B9780124201583000186>>. Citation on page 84.

STRATTON, J. A.; STONE, S. S.; HWU, W.-m. W. Mcuda: An efficient implementation of cuda kernels for multi-core cpus. In: AMARAL, J. N. (Ed.). **Languages and Compilers for Parallel Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 16–30. ISBN 978-3-540-89740-8. Citations on pages 103 and 147.

SUKHOROSLOV, O. Building web-based services for practical exercises in parallel and distributed computing. **Journal of Parallel and Distributed Computing**, v. 118, p. 177 – 188, 2018. ISSN 0743-7315. Available: <<http://www.sciencedirect.com/science/article/pii/S0743731518301023>>. Citations on pages 136, 137, 138, and 142.

São Paulo State University. **Computer Science Syllabus**. 2018. Available: <<http://www.fct.unesp.br/Home/Graduacao/CienciadaComputacao/PPP.pdf>>. Accessed: 18/05/2018. Citation on page 47.

Technological Institute of Aeronautics. **Computer Engineering Syllabus**. 2018. Available: <<http://www.ita.br/sites/default/files/pages/collection/ITA-CG-2014.pdf>>. Accessed: 18/05/2018. Citation on page 47.

The World University Ranking. **Computer Science - Times Higher Education**. 2018. Available: <<https://www.timeshighereducation.com/world-university-rankings/2018/subject-ranking/computer-science>>. Accessed: 18/04/2018. Citation on page 52.

TOURINO, J.; MARTIN, M. J.; TARRIO, J.; ARENAZ, M. A grid portal for an undergraduate parallel programming course. **IEEE Transactions on Education**, v. 48, n. 3, p. 391–399, 2005. Citations on pages 129, 130, 131, and 142.

Tsinghua University. **The Undergraduate and Graduate Courses Taught in English and Open to the International Visiting/Exchange Students at Tsinghua University**. 2018. Available: <<http://www.tsinghua.edu.cn/publish/9320/2016/20160122162541542994324/20160122162552835226823>>. Accessed: 18/05/2018. Citation on page 53.

TURNBULL, J. **The Docker Book: Containerization is the New Virtualization**. [S.l.]: James Turnbull, 2014. Citation on page 73.

_____. **The Docker Book: Containerization is the New Virtualization**. [S.l.]: James Turnbull, 2014. Citations on pages 77 and 78.

UBUNTU. **Network File System (NFS)**. 2019. Available: <<https://ubuntu.com/server/docs/service-nfs>>. Accessed: 2019.11.11. Citation on page 84.

UL-ABDIN, Z.; SVENSSON, B. Towards teaching embedded parallel computing: An analytical approach. In: **Proceedings of the Workshop on Computer Architecture Education**. New York, NY, USA: ACM, 2015. (WCAE '15), p. 8:1–8:6. ISBN 978-1-4503-3717-5. Available: <<http://doi.acm.org/10.1145/2795122.2795130>>. Citations on pages 61, 63, 64, 65, 66, and 72.

University Center of the State of Pará. **Computer Science Syllabus**. 2018. Available: <http://www.cesupa.br/Graduacao/Exatas/Doc/BCC/7Per/Programacao_Distribuida_e_Paralela.pdf>. Accessed: 18/05/2018. Citation on page 47.

University Center of Varzea Grande. **Systems Analysis and Development Syllabus**. 2018. Available: <<https://www.univag.com.br/curso/matriz-curricular/27/analise-e-desenvolvimento-de-sistemas/>>. Accessed: 18/05/2018. Citation on page 47.

University of Brasilia. **Computer Science Syllabus**. 2018. Available: <https://cic.unb.br/~lamar/coordenacao/ppp_bcc_final.pdf>. Accessed: 18/05/2018. Citation on page 47.

University of Buenos Aires. **Faculty of Exact and Natural Sciences**. 2018. Available: <<https://www.dc.uba.ar/materias/scmp/sigao.pdf>>. Accessed: 18/05/2018. Citation on page 53.

University of Cambridge. **Computer Science Tripos**. 2018. Available: <<https://www.cl.cam.ac.uk/teaching/1718/cst.pdf>>. Accessed: 18/05/2018. Citations on pages 53 and 57.

University of Cape Town. **Undergraduate Courses**. 2018. Available: <<https://www.cs.uct.ac.za/teaching/undergraduate-courses>>. Accessed: 18/05/2018. Citation on page 53.

University of Chile. **Course Program**. 2018. Available: <<https://www.dcc.uchile.cl/sites/default/files/info-cursos/CC5307%20Computaci%C3%B3n%20Paralela%20y%20Aplicaciones%202012-2.pdf>>. Accessed: 18/05/2018. Citation on page 53.

University of Fortaleza . **Computer Science Syllabus**. 2018. Available: <<https://www.unifor.br/web/graduacao/ciencia-da-computacao>>. Accessed: 18/05/2018. Citation on page 47.

University of Melbourne. **Distributed Computing Project (COMP90019)**. 2018. Available: <<https://handbook.unimelb.edu.au/subjects/comp90019>>. Accessed: 18/05/2018. Citation on page 53.

University of New South Wales. **Distributed Systems - COMP9243**. 2018. Available: <<http://www.handbook.unsw.edu.au/undergraduate/courses/2018/COMP9243.html>>. Accessed: 18/05/2018. Citation on page 53.

University of Oxford. **Concurrent Programming: 2017-2018**. 2018. Available: <<https://www.cs.ox.ac.uk/teaching/courses/2017-2018/concurrentprogramming/timetable.html>>. Accessed: 18/05/2018. Citation on page 53.

University of Pretoria. **Concurrent systems 226 (COS 226)**. 2018. Available: <<https://www.up.ac.za/yearbooks/pdf/module/COS%20226>>. Accessed: 18/05/2018. Citation on page 53.

University of São Paulo. **Concurrent Programming**. 2018. Available: <<https://uspdigital.usp.br/jupiterweb/obterDisciplina?sldis=SSC0143>>. Accessed: 18/05/2018. Citations on pages 46, 47, and 53.

University of Technology Sydney. **42009 Parallel and Multicore Computing**. 2018. Available: <<http://handbook.uts.edu.au/subjects/42009.html>>. Accessed: 18/05/2018. Citation on page 53.

University of the Witwatersrand. **2018 Sci Rules Syllabuses**. 2018. Available: <<https://www.wits.ac.za/media/wits-university/students/academic-matters/documents/2018%20Sci%20Rules%20Syllabuses%20-%20reduced%20size.pdf>>. Accessed: 18/05/2018. Citation on page 53.

WOLFER, J. A heterogeneous supercomputer model for high-performance parallel computing pedagogy. In: IEEE. **2015 IEEE Global Engineering Education Conference (EDUCON)**. [S.l.], 2015. p. 799–805. Citation on page 35.

XAVIER, M. G.; NEVES, M. V.; ROSSI, F. D.; FERRETO, T. C.; LANGE, T.; ROSE, C. A. F. D. Performance evaluation of container-based virtualization for high performance computing environments. In: **2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing**. [S.l.: s.n.], 2013. p. 233–240. ISSN 1066-6192. Citations on pages 74 and 83.

YANG, X.; ZHOU, Z.; WALLACE, S.; LAN, Z.; TANG, W.; COGHLAN, S.; PAPKA, M. E. Integrating dynamic pricing of electricity into energy aware scheduling for hpc systems. In: **SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2013. p. 1–11. Citation on page 70.

ZARESTKY, J.; BANGERTH, W. Teaching high performance computing: Lessons from a flipped classroom, project-based course on finite element methods. In: **2014 Workshop on Education for High Performance Computing**. [S.l.: s.n.], 2014. p. 34–41. Citations on pages 44, 61, 63, and 64.

ZARZA, G.; LUGONES, D.; FRANCO, D.; LUQUE, E. An innovative teaching strategy to understand high-performance systems through performance evaluation. **Procedia Computer Science**, v. 9, p. 1733 – 1742, 2012. ISSN 1877-0509. Available: <<http://www.sciencedirect.com/science/article/pii/S1877050912003122>>. Citation on page 33.

ZHANG, J.; LU, X.; PANDA, D. K. High performance mpi library for container-based hpc cloud on infiniband clusters. In: **2016 45th International Conference on Parallel Processing (ICPP)**. [S.l.: s.n.], 2016. p. 268–277. Citation on page 75.

ZORZO, A.; NUNES, D.; MATOS, E.; STEINMACHER, I.; LEITE, J.; ARAUJO, R.; CORREIA, R.; MARTINS, S. **Referenciais de Formação para os Cursos de Graduação em Computação**. Sociedade Brasileira de Computação (SBC). 153p. [S.l.], 2017. Citation on page 42.

École Polytechnique Fédérale de Lausanne. **Parallelism and concurrency**. 2018. Available: <http://isa.epfl.ch/imoniteur_ISAP/!tffichcours.htm?ww_i_matiere=1887880865&ww_x_anneeAcad=2018-2019&ww_i_section=249847&ww_i_niveau=6683117&ww_c_langue=em>. Accessed: 18/05/2018. Citations on pages 53 and 54.

GLOSSARY

Amazon Elastic Computing Cloud (EC2): is a cloud service that provides scalable computing capacity. This service is designed to make computational scalability at the Web level easier for developers.

Apache Hadoop YARN: is a cluster management technology and processing large data volumes, including attention to fault tolerance.

API: is an interface provided by an application for external access by other applications through pre-established commands.

Cgroup: is a Linux kernel implementation that allows to partition system resources (CPU, memory, I/O) by process group.

Distributed Replicated Block Device (DRBD): generally used in high-availability clusters, offering storage devices available through a computer network.

Docker: is a virtualization system for containers compatible with Linux and Windows Server, provides isolation and security to run and manage applications quickly and reliably.

Dynamic Host Configuration Protocol (DHCP): is a service protocol for granting IP addresses to computers and devices on a network.

ENADE: is a Brazilian national examination that performance evaluates undergraduate students based on defined parameters.

Full Virtualization: provides the OS a replica of the underlying hardware.

HPC: uses computers and parallel processing techniques to handle and analyze large amounts of data or complex computational problems with high speed.

Infrastructure as a Service (IaaS): is a service model that outsources hardware, storage, servers, and networking components to other organizations.

Jaccard index: is used to measure the similarity between two samples.

JavaScript Object Notation (JSON): is a lightweight data-interchange format widely used for exchanging information between systems.

KVM: is a Linux virtualization solution to run multiple virtual machines. Each virtual machine has private virtualized hardware without changes the guest OS image.

Logical Volume Manager (LVM): is a logical volume manager for Linux, managing partitions and file systems.

MapReduce: is a proposed programming model by Google to facilitate the processing of large data volumes (Big Data).

MPMD: is a programming model for achieving parallelism, with many different programs running on different parts of data.

Multi-tenant: is an architecture that provides a particular software to multiple clients in a single shared instance.

NAT: is a network address translation system that assigns a public address to a computer for private network interconnection. In this case, the computers in the private network have public network access through the NAT translation.

Network Information Service (NIS): is a directory service protocol that distributes resource names on the network.

Network Time Protocol (NTP): is a protocol used for synchronizing the clocks of computers and devices in data networks.

Open Virtual Platform (OVP): is a virtual platform that simulates a given system by hardware or software representation.

OpenVZ: is a container virtualization system for Linux; it allows multiple containers inside the host operating system. It also promotes security and isolation, guaranteeing that user applications do not conflict.

OS-level: is a virtualization approach based on OS installations that partition the machine's physical resources, creating multiple users isolated instances over a single host kernel.

Paravirtualization: sends instructions to the host system when it causes a change in the system state. This represents a significant performance increase, unlike full virtualization, where VMM tests all instructions.

Platform as a Service (PaaS): in this service model, the tools and libraries for software creation are offered to the user, who controls the configuration and distribution of the software.

PVM: is a software system that allows heterogeneous computers (shared or local memory multiprocessors, vector supercomputers, graphics engines) used as a simultaneous computational resource that various networks can interconnect.

Redundant Array of Inexpensive Drives (RAID): creates data redundancy between two or more disks.

Sandboxing: is a method to isolate and monitor programs in isolated spaces at the operating system level.

Secure Shell (SSH): is an encrypted secure communication channel for communication between remote hosts.

Spark: is an Apache system that provides high-level APIs for fast cluster computing, optimized for graphics support, structured data processing, machine learning, and graph processing.

Symmetric Multiprocessing (SMP): is composed of two or more identical processors sharing the main memory.

TypeScript: is a JavaScript-based open-source language that adds static type definitions.

vCPU: is used to assign physical processing units to virtual machines.

VMM: is the operating system modified to invoke the Virtual Machine Manager (VMM) when some instruction can change the system state. This modification reduces the VMM needs to test each instruction and represents a significant performance increase.

VPS: is a private virtual server that simulates a dedicated machine through a specific program or system on a physical server.

VServer: is a Linux virtualization system for container creation, management, and manipulation. This system works by creating instances by kernel isolation.

Xen: is an open-source for hypervisor-based virtualization; it allows creating and executing multiple virtual machines over the same hardware.

DV AND NFS PERFORMANCE EVALUATION

This appendix presents the tables containing all results and significance levels for the experiments comparing Docker (consistent, cached, and delegated) and NFS (synchronous and asynchronous) volume sharing, shown in [Subsection 3.4.3](#) of [Chapter 3](#).

[Chart A.1](#) shows all tested factors and levels for all modes (read and write operations with concurrency and no concurrency of resources) for SSD disks. [Chart A.2](#) shows the same data for HDD disks. The blue columns (o) present the results without statistical significance; that is, there is no difference between the averages of the results. Red columns (x) represent results with significance.

Chart A.1 – Paired t-test - significance for SSD

Disk	Benchmark	Concurrency	Operation (read/write)	Sync - DV consistent	Sync - DV cached	Sync - DV delegated	Sync - Async	ASync - DV consistent	ASync - DV cached	ASync - DV delegated	DV consistent - cached	DV cached - delegated	DV delegated - consistent
SSD	MySQL		R	o	o	o	o	o	o	o	x	x	x
			W	o	o	o	o	o	o	o	x	x	x
			T	o	o	o	o	o	o	o	x	x	x
			Q	o	o	o	o	o	o	o	x	x	x
		X	R	o	o	o	o	o	o	o	x	x	x
			W	o	o	o	o	o	o	o	x	x	x
			T	o	o	o	o	o	o	o	x	x	x
			Q	o	o	o	o	o	o	o	x	x	x
	File 100M		R	o	o	o	o	o	o	o	o	o	o
			W	o	o	o	o	o	o	o	o	o	o
		X	R	o	o	o	o	o	o	o	o	o	o
			W	o	o	o	o	o	o	o	o	o	o
	File 2G		R	o	o	o	o	o	o	o	o	o	o
			W	o	o	o	o	o	o	o	o	o	o
		X	R	o	o	o	o	o	o	o	o	o	o
			W	o	o	o	o	o	o	o	o	o	o
	Small Image		R	o	o	o	x	o	o	o	x	x	x
			W	o	o	o	o	o	o	o	x	x	x
		X	R	o	o	o	x	o	o	o	x	x	x
			W	o	o	o	o	o	o	o	x	x	x
	Large Image		R	x	o	x	x	x	x	x	x	x	x
			W	o	o	o	o	o	o	o	x	x	x
		X	R	o	o	o	x	o	o	o	x	x	x
			W	o	o	o	o	o	o	o	x	x	x

Source: [Bachiega et al. \(2020\)](#).

Chart A.2 – Paired t-test - significance for HDD

HDD													Disk
Large Image		Small Image		File 2G		File 100M		MySQL				Benchmark	
X		X		X		X		X				Concurrency	
												Operation (read/write)	
												Sync - DV consistent	
												Sync - DV cached	
X		X		X		X		X				Sync - DV delegated	
												Sync - Async	
												ASync - DV consistent	
												ASync - DV cached	
X		X		X		X		X				ASync - DV delegated	
												DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X		X		X		X		X				DV consistent - cached	
												DV cached - delegated	
												DV delegated - consistent	
												DV consistent - cached	
X													

IGUANA DESIGN SUGGESTIONS COLLECTED FROM THE EXPERIMENTS

This appendix shows the suggestions inserted by professionals [Chart B.1](#) and students [Chart B.2](#) in the experiments carried out with the Iguana tool. The tables show the suggested recommendations. In addition, we defined in column R the risk of implementation of the feature in Iguana as low (L), moderate (M), and high (H), as follows:

- Low: a slight impact on the system, just a change of content or improvement in the HTML.
- Moderate: change in some function or part of the code that does not significantly change the tool operation mode.
- High: potential risk of causing problems due to significant changes in functions or system logic.

In column F, we presented the implementations directed for future work. Finally, in column I, we marked all the features that have been implemented.

Chart B.1 – Professionals suggestions for improvements in the tool after the first experiment

Suggestions	R	F	I
Exercise List: a clear indication that the exercise has been delivered.	L		X
Exercise List: clear indication if the exercise is correct.	L		X
In the "DIFF" tab, after compiling and executing the code, in the "Exercise Result/Your Result:" field, the term "EQUAL" appears twice on the screen.	L		X
In the "Coding" tab, when maximizing the code editor, the button does not change the text to "Minimize" to make the command more intuitive.	L		X

For ease of usability, I would change the "Compile / Compile and Run" button to after the configuration/parameterization block.	L		X
Change "Self-registration" to "create an account" or "register".	L		X
Set "Record updated successfully" when to change password.	L		X
Recover password: change colors (contrast is bad to see the selected option).	L		X
Entering the wrong password returns, "You are not authorized to make this request". Change to "This email or password is incorrect" or something.	L		X
Non-existent user returns the same message as the wrong password: "You are not authorized to make this request". Switch to "This user does not exist on our database".	L		X
Change "add exercise files" to "add input file" or something like that.	L		X
In exercise 3, it was not very clear how to define an input parameter. Maybe put a separate option, like "Extra arguments".	L		X
One of the points is in the tab of the difference between the codes; it is not very clear which is the teacher's code and which is the student's code; this could be clearer without the person having to read the text above the comparison area in full.	L		X
Another point is entering the exercise; when you enter it, you go to the list of answers already given, instead of going into the exercise itself, which is the behavior I expected based on websites like NepsAcademy, Hackerrank, etc. The list of answers could be a tab on the exercise page; in my opinion, it would make it simpler and reduce the number of clicks to reach the exercise.	H	X	
At the end of the run, I could tell if the result worked or not so that I wouldn't have to go to the Diff tab.	L		X
The Action button for each exercise could have a name instead of an icon.	L		X
It bothered me a little that the User button always displayed in the Module div even though I had already created the user.	L		X
You could add Gamification to the Iguana Platform.	H	X	
I missed the tool to save the screens' states; when I changed from exercises to try code and then returned, I lost the code I had done in any of the windows.	M		X
It would be nice to leave guides (hints) explaining a little bit about parallel programming for the more lay. It could be a link to some pages within the system itself.	L	X	
As for the compiler's choice (Parameters): it could be automatic based on the #include files.	M		X

System messages could be more highlights.	L		X
Compile and run could be defaulted, personally I used it the most.	L		X
Comparative graph of parallel vs. serialized processing time for the same task.	M	X	
Color load balancing options, for example, core X, will be responsible for 75% of processing, and core Y will be responsible for 25%.	H	X	
I recommend rethinking the position of the call to compare the code with the registered result. The processed result appears at the bottom of the page, so I believe that there should already be the trigger to access the comparison, which is now at the top of the page.	L		X
Integration with remote/online repositories would make it more attractive because it would be possible to use the online repository and trigger its actions to execute code in parallel.	M	X	
The location where the user/exercise buttons are is a little strange; you could have a smaller or side menu to occupy less of the screen while executing codes.	L		X
At first, navigating between exercises and attempts using the top right buttons is not very comfortable. Perhaps using the buttons at the bottom and a breadcrumb at the top will help in a more intuitive experience.	L	X	
The text editor used has a scroll bar even without content overflow, which gets in the way when using the mouse scroll.	L		X
The interface design can be improved.	L		X

Source: Elaborated by the author.

Chart B.2 – Students' suggestions after performing experiments with the tool

Suggestions	R	F	I
Save the code automatically.	M		X
Increase timeout time.	L		X
Possibility to increase the editor's font.	L		X
Highlight the "Submit" button.	L		X
Monitoring of resources used.	M	X	
Access the tool with the same login from different computers at the same time.	H	X	
Set "Record updated successfully" when to change password.	L		X
As soon as the user logs in, the exercises that are in progress could already be presented.	L	X	
Use stdin.	H	X	
Improve error messages.	M	X	
After we submit an activity, I think it would be interesting if we received an email confirmation.	M	X	

Source: Elaborated by the author.

IGUANA FEEDBACK COLLECTED FROM EXPERIMENTS

This appendix provides feedback after the experiments are performed. [Chart C.1](#) shows the opinions of professionals in the field and lecturers about teaching with Iguana. [Chart C.2](#) shows student impressions after using the tool to create and run parallel code.

Chart C.1 – Feedback from professionals after the features are applied

Feedback
I believe that using the Iguana tool in my learning process would have increased my learning curve and, consequently, would bring me more courage when perfecting the parallel programming techniques.
Congratulations for the work!
After the modifications, the tool was cleaner and more organized. It certainly contributes to the learning of beginners.
I believe that the system has a friendly and attractive interface. Some interface elements could be improved usability (contrasts, shading, etc.), but they are essential elements of .css and not mandatory. The tool features, in themselves, are its differential. The system can promote teaching and learning, especially in environments with limited infrastructure, and motivate students to exercise parallel programming. I congratulate those involved and wish them success in continuing this project.
The tool is very robust and addresses several concepts that facilitate the learning of parallel programs.
I believe that the Iguana tool has the potential to contribute to the teaching and learning of concurrent programming inside and outside the classroom; however, issues with the navigability of the interface need to be improved.

Source: Elaborated by the author.

Chart C.2 – Feedback from students about the Iguana tool

Feedback
I found that the tool made the development of the exercises a lot easier. In addition, it saved time for organizing the files and applications needed for compilation and execution.
I really liked the tool, mainly to compile and run CUDA. The only warning is that I still think that using an IDE to test and debug before sending it to Iguana is a little faster. However, overall the experience was very positive.
I believe that Iguana is incredible because it allows us to execute code in parallel without configuring anything on our machines. In addition, remote access is impressive, especially for those who have few resources available.
I do not think it directly affects learning. But, still, it is really a facilitator, being very easy to use, quick to understand, and helps many students who cannot run parallel programs due to a lack of personal structure.
The tool is helpful because not everyone has machines to execute the code (video cards from Nvidia, for example), and the tool helps a lot in this regard. However, when writing the code directly in the browser, I found the experience a little strange, perhaps because there was no way to keep the entire screen editable.
I found the interface unintuitive for the exercises.
I liked the system, and I think it can make parallel code execution a lot easier. Our group found no problem.
I liked the platform because it makes programming immediately, without going through the journey of installing the necessary packages (I had several problems installing MPI on my computer).
The system is excellent and, in a way, innovative. However, creating and executing parallel programming programs in the terminal of your computer has disadvantages, such as the complicated installation of libraries, hardware often inaccessible (multicore CPUs, GPUs, Clusters, etc.), and some advantages such as better monitoring of resources, the possibility of additional tools (IDEs, Valgrind, gdb, etc.). Thus, this tool is helpful to students who have not had contact with parallel programming. Therefore, they do not have a device configured to create and execute parallel programs, allowing an introduction to this area with fewer problems and pains. Also, I found the tool very important and exciting, and I hope you will continue to work and research it. Congratulations to all the researchers involved!
From the mature Linux user view, I believe that it has exciting potential. Furthermore, part of the difficulties I noticed in my classmates was not related to the parallelism but in accessing the cluster via ssh and handling files remotely. In summary, most students are not experts in the Linux platform, so a web platform seems like an exciting alternative.

I found Iguana very good for allowing simple code execution directly in the cluster, especially when some code was executed correctly on my machine but had a compilation error in the laboratory due to differences in versions. In the following classes, I will have fewer problems like these and will focus my time more on the code.

In general, the platform is excellent. The interface is intuitive, simple, and beautiful. Use is straightforward. Comparing the execution with the available input files is a perfect teaching tool and makes exercises much more accessible.

I found the tool very cool! I believe it will help a lot in teaching parallel programming.

I really liked Iguana, it was effortless to understand, and it seems to work perfectly. The only objection I have is about the layout, and I believe it could be more intuitive in some parts.

Although we used the platform only at the end of the course, it was an excellent experience. Only a few things in the interface are a little confusing, like selecting input files, but it sure is a great platform, and I imagine that the following high-performance computing classes will have a great starting point.

Source: Elaborated by the author.

CUDA EXPERIMENTS

This appendix presents the results of CUDA algorithms executed in Iguana, as shown in [Section 5.4](#) of [Chapter 5](#).

Figure D.1 – Output of hello world in CUDA

```
1 Hello world
2 Hello world
3 Hello world
4 Hello world
5 Hello world
6 Hello world
7 Hello world
8 Hello world
9
```

Source: Elaborated by the author.

Figure D.2 – Output of descriptive statistics metrics in CUDA

```
1 12.0 17.0 11.5 13.5
2 8.1 12.1 6.6 3.7
3 8.5 12.0 10.5 7.5
4 8.0 8.0 4.0 -1.0
5 110.0 188.4 68.7 217.9
6 10.5 13.7 8.3 14.8
7 0.9 0.8 0.7 1.1
```

Source: Elaborated by the author.

Figure D.3 – Output of thresholding in CUDA

```
1  0.0 1.0 0.0
2  0.0 1.0 1.0
3  1.0 1.0 0.0
4
```

Source: Elaborated by the author.

Figure D.4 – Output of greatest common divisor in CUDA

```
1  15
```

Source: Elaborated by the author.

Figure D.5 – Output of matrix multiplication in CUDA

```
1  30.0 24.0 18.0
2  84.0 69.0 54.0
3  138.0 114.0 90.0
4
```

Source: Elaborated by the author.

Figure D.6 – Output of multiplication of vectors by a scalar in CUDA

```
1  8 16 24
```

Source: Elaborated by the author.

IGUANA CLUSTER SYSTEM TOOL MANUAL

This appendix presents the manual of the Iguana tool, containing three sections: lecturer (admin), student (user), and the guide for cluster installing and using.

IGUANA CLUSTER SYSTEM

PRODUCED BY: NAYLOR G. BACHIEGA
NAYLOR@USP.BR

ADVISOR: PAULO SERGIO LOPES DE SOUZA



THIS MANUAL SHOWS THE IGUANA, A TOOL USED FOR THE PRACTICAL TEACHING OF
PARALLEL PROGRAMMING. THIS SYSTEM CREATES AND MANAGES A CLUSTER
INFRASTRUCTURE TO RUN CODES IN MPI, OPENMP, AND CUDA.

Table of Contents

1	Introduction	2
1.1	Infrastructure	2
1.2	Installation	4
2	User Roles	5
2.1	Login	5
2.2	Modules	6
2.3	Exercises	7
2.3.1	Content Tab	9
2.3.2	Coding Tab	9
2.3.3	Diff Tab	14
2.4	Try a Code	15
3	Admin Roles	16
3.1	Default User Admin	16
3.2	Modules	16
3.3	Users	17
3.4	Groups	17
3.5	Exercises	18
3.5.1	Add an exercise	19
3.5.2	Answers	21
3.6	Settings	22
3.7	Try a Code	23
4	Cluster Operation	24
4.1	Operation mode	24
4.2	Clusters available to connect	25
4.3	Current nodes in the cluster	25
4.4	Active nodes	26
4.5	Active front-ends	26

About This File

This file was created by Ph.D. candidate Naylor Garcia Bachiega, under the supervision of Paulo Sergio Lopes de Souza from USP-ICMC.

Introduction

This is a self-configuring cluster system specially designed for parallel teaching programming, and its goal is to create a low-cost cluster without configuration and maintenance.

Iguana can run on one physical or virtual host and create dozens of virtualized nodes to test and distribute parallelized code. The tool also connects more than one host to instantiate thousands of virtualized nodes to simulate an extensive network or a large cloud.

1.1 Infrastructure

The cluster system consists of two parts, front-end, and back-end, running on the same machine, as shown in Figure: 1:

- **front-end:** consists of the user interface. It has all the functionality of the system with two permission modes: user and admin.
- **back-end:** the part responsible for creating the cluster infrastructure. The back-end uses Docker, the Network File System (NFS), APIs, and other technologies to create a cluster to execute parallel codes. Then, the back-end receives the code, runs it on the nodes, and returns the result to the front-end.

In the front-end, guidelines about users are described in Section 2, and the admin's functions are specified in Section 3. In the back-end, the cluster management and configuration functions are described in Section 4.

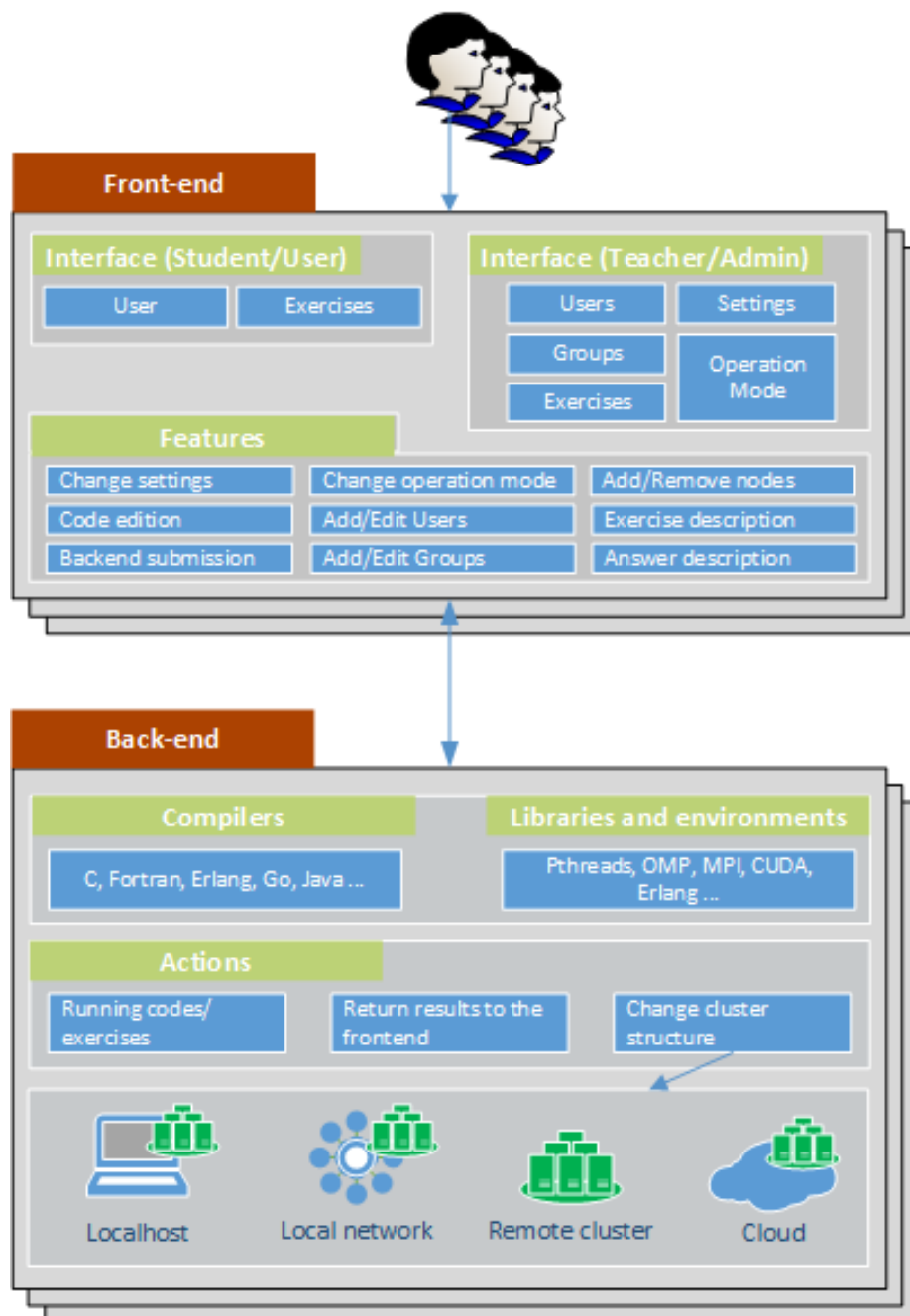


Figure 1: Cluster infrastructure

The base operating system for the construction of this project was Ubuntu 21.04 Linux kernel 5.4. The system is designed to work even on VMs with 1GB RAM and 8GB of disk space.

1.2 Installation

The tool can be installed in two ways:

- 1. Run the command below on Linux to execute the installation script:

```
sudo apt install curl -y &&  
curl -s https://raw.githubusercontent.com/_  
iguana-hpc-usp/ICS/master/install.sh | bash -s --
```

- 2. Download the virtual machine ready and import it into VirtualBox:

<https://github.com/iguana-hpc-usp/ICS/>

If using the VM, do not forget to correctly configure the network adapter for bridge mode (<https://www.virtualbox.org/manual/ch06.html>).

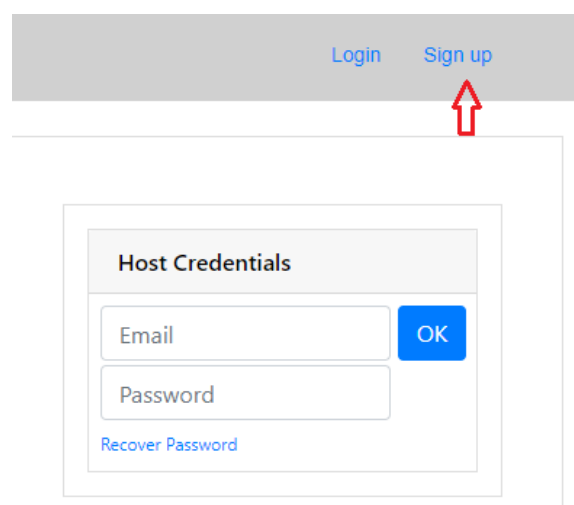
After installation, the system prints the machine's IP for browser access on port 8000, and that's it! See an example:

<http://IP:8000>

User Roles

2.1 Login

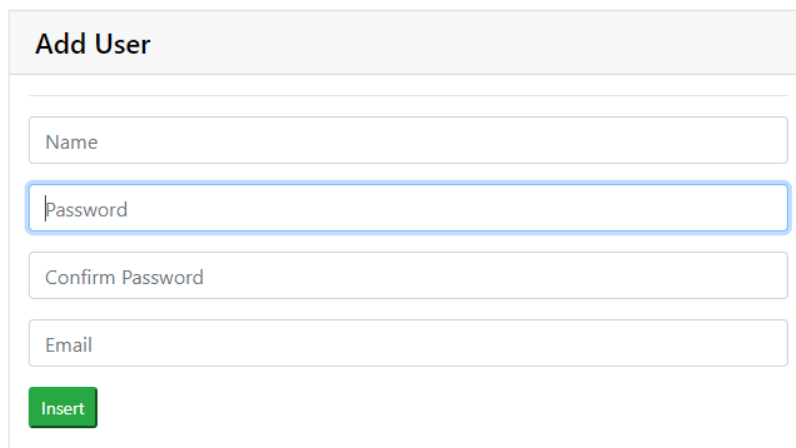
Figure 2 shows the login screen for already-registered users. If the self-registration is enabled in the system, the tool will show the Sign-Up link.



The figure shows a login interface. At the top, there is a grey bar containing the text 'Login' and 'Sign up' in blue. A red arrow points to the 'Sign up' link. Below this bar is a white box with a grey border. Inside this box is a smaller box with a grey header 'Host Credentials'. Below the header are two input fields: 'Email' and 'Password'. To the right of the 'Email' field is a blue 'OK' button. Below the 'Password' field is a blue link 'Recover Password'.

Figure 2: Logging or creating a user

The system delivers the form shown in Figure 3 to the creation of self-registered users.



Add User

Name

Password

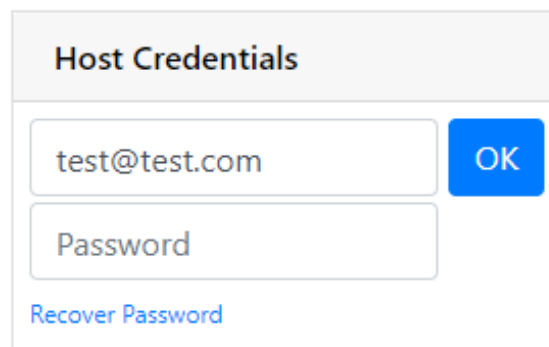
Confirm Password

Email

Insert

Figure 3: Creating a user

Now, with the user and passwords registered, it is possible to log in, as shown in Figure 4.



Host Credentials

test@test.com

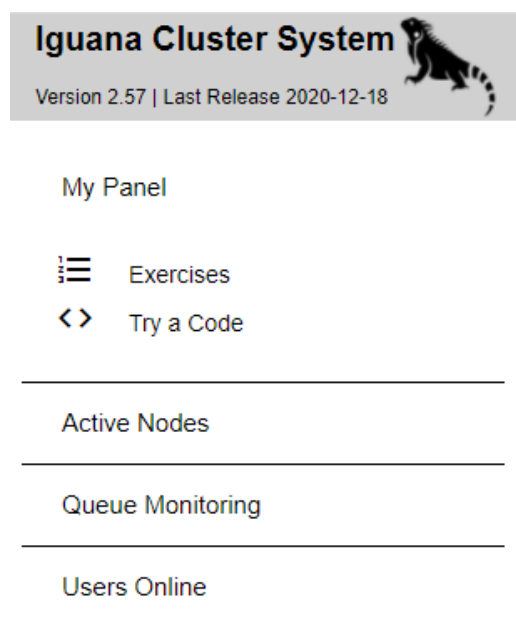
OK

Password

[Recover Password](#)

Figure 4: Login

2.2 Modules



Iguana Cluster System

Version 2.57 | Last Release 2020-12-18

My Panel

Exercises

Try a Code

Active Nodes

Queue Monitoring

Users Online

Figure 5: User modules

When logged in a user mode, the module panel (Figure 5) releases some features, including:

- **Users:** allows changing user data, such as name, password, and e-mail.
- **Exercises:** presents the exercises sent by the admin.
- **Try a Code:** shows the code editor and compiler to run codes quickly.
- **Active Nodes:** presents the physical and virtual nodes of the cluster.
- **Queue Monitoring:** shows processes and users waiting for code execution in the queue.
- **Users Online:** lists users are logged into the tool.

2.3 Exercises

This module shows the exercises registered (Figure 6) by the admin.

Exercises						
Filter: <input type="text"/>						
ID	Title	MNS/T	Expiration Date	RT	Code Analysis	Delivered
5	Higher Value in MPI	10/15s	2021-08-20 18:00 <small>(more than 11 day(s))</small>	463.49ms	<div><div></div></div>	☰ (1)
12	PI Number Calculation	10/30s	2021-07-16 20:00 <small>(expired)</small>	1.55s	<div><div></div></div>	☰ (0)
					Items per page: <input type="text" value="25"/>	1 – 2 of 2

Figure 6: Exercises

- **Title:** displays the exercise title.
- **MSN/T:** displays the maximum number of submissions for this exercise and the maximum execution timeout.
- **Expiration Date:** shows the exercise's expiration date, and delivery after the deadline is no longer possible.
- **RT:** display the response time of the exercise registered by the admin.
- **Code Analysis:** shows the automatic correction of the code semantics (code smell), for more information, see the Clang-Tidy documentation available at <https://clang.llvm.org/extra/clang-tidy/>.
- **Delivered:** to solve and record the answers to this exercise (Figure 8).

When the mouse clicks the icon in the Delivered column, the system will show the screen in Figure 7.

Last Exercise Answers: Higher Value in MPI					
Time to finish: 11d 5h 24m 36s					
Filter: <input type="text"/>					
ID	Name	Last Submission	Last Result	Score / Feedback	Attempts Number
1	Student	2021-07-16 17:38	EQUAL	No/No	1
Items per page: <input type="text" value="25"/> 1 – 1 of 1					

Figure 7: Exercise answers

More details can be seen by clicking on the line containing the exercise answer, as shown in Figure 8.

Last Exercise Answers: Higher Value in MPI

Time to finish: 11d 5h 33m 44s

ExercisesAdd

Filter:

ID	Name	Last Submission	Last Result	Score / Feedback	Attempts Number
1	Student	2021-07-16 17:38	EQUAL	No/No	1

Number	Submission	Code Analysis	Response Time	Events	Status	Action
1	2021-07-16 17:38	<div></div>	463.15ms	2	EQUAL	<>

Items per page: 251 – 1 of 1

Figure 8: Exercise answers detail

- **Last Submission:** records the date and time of the last submission.
- **Last Result:** shows whether the submission result was the same as the submission result made by the admin.
- **Score/Feedback:** displays the score and feedback recorded by the admin.
- **Attempts Number:** shows the number of submissions delivered.
- **Code Analysis:** shows the automatic correction of the code semantics (code smell), for more information, see the Clang-Tidy documentation available at <https://clang.llvm.org/extra/clang-tidy/>.
- **Response Time:** displays the response time of the exercise.
- **Events:** shows the number of builds and runs before submitting the exercise.

- **Action:** displays the code of the response sent.

Click the **Add button** to add a new response attempt. After clicking the button, a new screen will appear with three available tabs (Figure 9):

- **Content:** shows the content and description of the exercise sent by the admin.
- **Coding:** presents the code editor and the compilation and execution options.
- **Diff:** shows the difference in the result of the user's execution and the admin's execution.

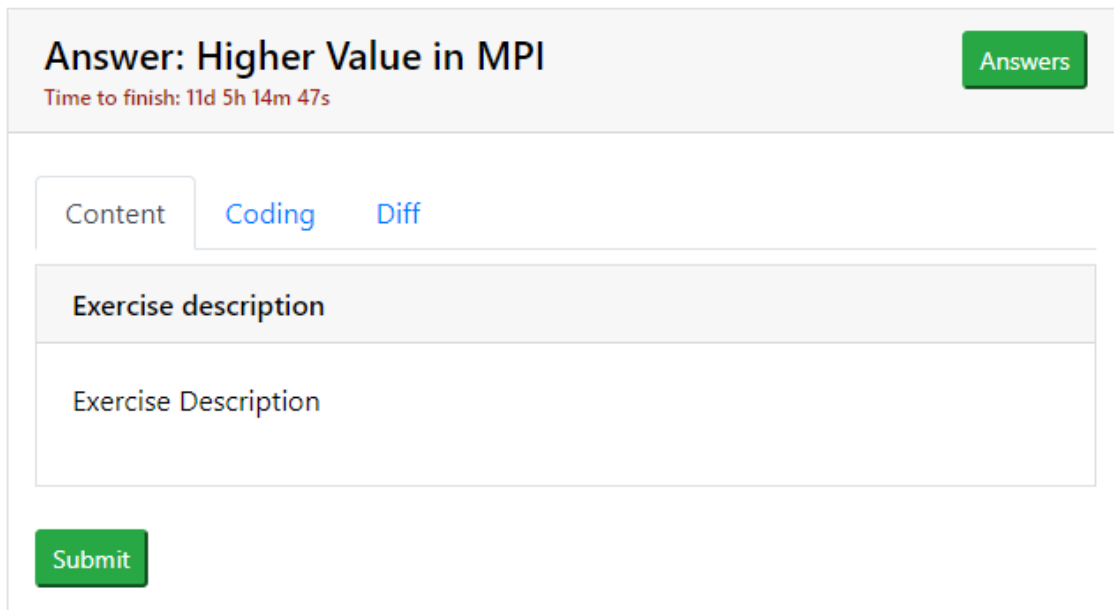


Figure 9: Exercise content

2.3.1 Content Tab

The content tab shows the exercise description registered by the admin (Figure 9).

2.3.2 Coding Tab

The Coding tab shows the system code editor, as shown in Figure 10. There are many options; let's describe them from top to bottom:

- **Theme:** allows choosing between a light or dark theme, according to programmer preferences.
- **Font Size:** the tool offers different font sizes.
- **MiniMap:** shows the minimap on the right side of the editor.
- **Load Example Code:** allows to load an example code.
- **Maximize Window Code:** increases the code editor area.
- **Code Saving:** the system automatically saves the code as the programmer writes in it.

Answer: Higher Value in MPI

Answers

Time to finish: 11d 5h 11m 24s

Content

Coding

Diff

Theme ▾

Font Size ▾

MiniMap ▾

Load Example Code ▾

Code Saving ▾

1

Compile and Run ▾

Queue Off

Parameters ▾

Files

Nodes (MPI)

Results

Special Tags

Command (compile)

Arguments (compile)

Command (execute)

Arguments (execute)

Extra Arguments (execute)

Submit

Figure 10: Code editor

- **Queue Off/On:** when turned on, the user will enter a queuing system and waiting to compile and execute the code. This will ensure that there is no competition for resources from other users simultaneously.
- **Compile and Run:** compiles and/or executes the code.
- **Parameters:** defines the compilation and execution parameters (Figure 11).
- **Files:** allows upload external files to be called in the code or to use the files inserted by the admin in the exercise content.
- **Nodes:** shows the nodes available for execution.
- **Results:** displays the code result or shows the compilation/execution errors.
- **Special Tags:** is a visual way of showing nodes and ranks using special tags.

Parameters Tab Figure 11 shows the Parameters tab to load standard configurations for compiling and executing code in MPI, OpenMP, CUDA, or OpenMP/MPI hybrid. It is possible to change the parameters for other executions and compilations. For that, the compiler needs to be installed on the host system.

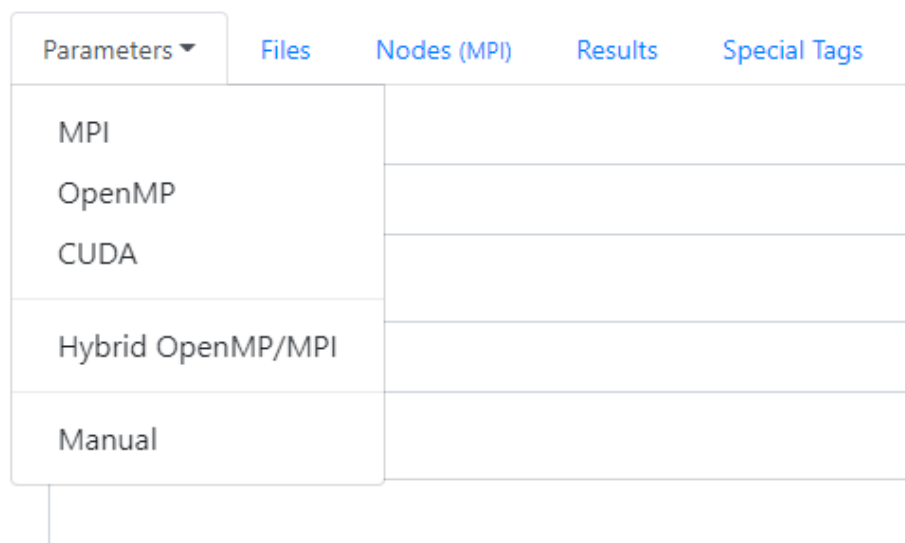


Figure 11: Parameters

After defining the compilation and execution parameters, the system releases the button to compile and run the code written in the editor (Figure 12). Note that the parameters are the same used conventionally in a command prompt.

Parameters ▾ Files Nodes (MPI) Results Special Tags

Command (compile)

mpic++

Arguments (compile)

-fopenmp main.c -o main

Command (execute)

mpiexec

Arguments (execute)

-n 8 -host 10.0.1.3 main

Extra Arguments (execute)

Compile

Compile and Run

Compile and Run

Figure 12: Compile and run

Files Tab The tool allows compiling a program using external files. For this, on the left side of Figure 13, the system has an area for uploading a file. Also, there are cases where the admin has attached a file in the exercise content for compilation. In this example, the middle frame shows the files available for upload and download.

On the right side, the last frame shows an example of how to call the file inside the code. Again, always use the same original file name.

Parameters ▾ Files Nodes (MPI) Results Special Tags

Add a file for this run
(MaxFileSize: 1 KB)

Escolher arquivo Nenhum arquivo selecionado

Upload

Add exercise files

entrada01.txt

Example of code to read an array from a file.

```
FILE * pFile;

pFile = fopen ("file.txt","r");

fscanf(pFile, "%d ", &lin);
fscanf(pFile, "%d\n", &col);
fscanf(pFile, "%lf\n", &limiar);
```

Figure 13: Files

Nodes Tab With MPI, the tool chooses the number of processes and hosts to run. However, if no specific host is selected, the system will determine them all to run.

Parameters
Files
Nodes (MPI)
Results
Special Tags

Number of process
(Use with MPI)

4

Nodes(Use with MPI)

☒ 10.0.1.3
☒ 10.0.1.4
☐ 10.0.1.5
☒ 10.0.1.6

Figure 14: Nodes

Results Tab When clicked on the button to compile or compile/run, the system will automatically check if the execution queue was requested. If yes, the user waits in the queue. If the user does not need a queue, the system will run code immediately, first performing the compilation. Then, it is successful; the system will show the time spent and the compile command line (Figure 15).

Parameters
Files
Nodes (MPI)
Results
Special Tags

Compile and Run ->| Program output

Queue Position: done!

Running timeout: 0

Compile command executed:
mpic++ main.c -o main (time 5.38s)

Run command executed:
mpiexec -n 4 -host 10.0.1.3,10.0.1.4,10.0.1.6 main (time 4.19s)

Code copied to result area.

```

1 message: {M}Hello World{M} from processor: {P}b03a30797b7b{P}, rank: {R}2{R} out of 4 pro
2 message: {M}Hello World{M} from processor: {P}6d69e4bf678b{P}, rank: {R}3{R} out of 4 pro
3 message: {M}Hello World{M} from processor: {P}6d69e4bf678b{P}, rank: {R}0{R} out of 4 pro
4 message: {M}Hello World{M} from processor: {P}f7f7d2535618{P}, rank: {R}1{R} out of 4 pro
5

```

Figure 15: Result area

Afterward, the system will send the code to run according to the user’s choices. If no errors,

the system displays the time spent for execution and the run command line. If no errors, the program return is copied to the result area.

Special Tags Tab Special Tags allow visualizing the code result separated by Nodes and Ranks (Figure 16).

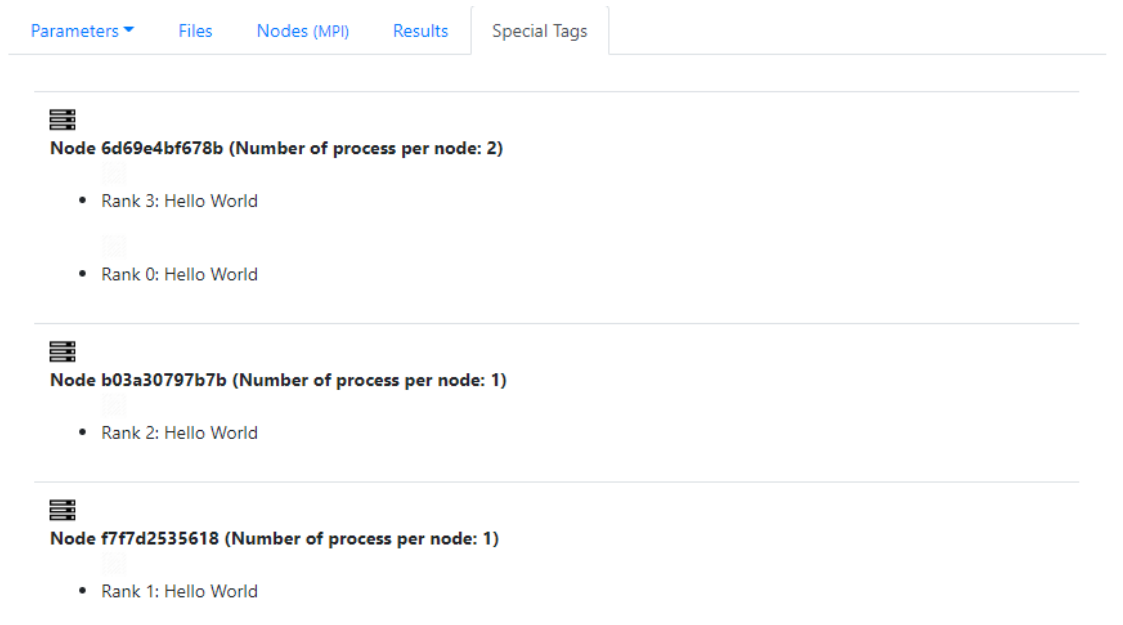


Figure 16: Special tags result

For this, it is necessary to use tags to mark in the outputs, as shown in Figure 17.

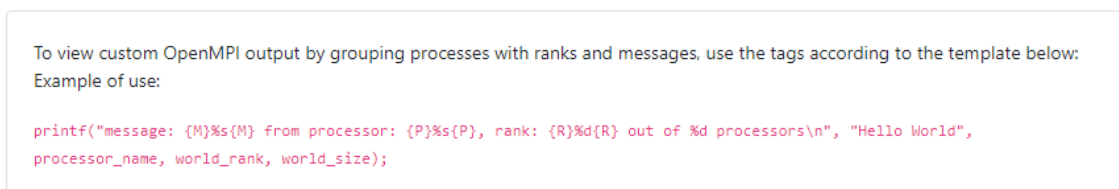


Figure 17: Tags

2.3.3 Diff Tab

The "Diff" section presents the result of the activity. However, the "Diff" section is locked; it is not editable for users and admins to keep the result impartiality.

Figure 18 shows on the left side the answer to the admin's code execution and, on the right, the result of the user's code. If the codes are NOT precisely the same, the message "NOT EQUAL" will be displayed.

The result comparison is made similar to the diffused by the Linux/Unix-based system. For more information, consult the documentation available at <http://man7.org/linux/man-pages/man1/diff.1.html>.

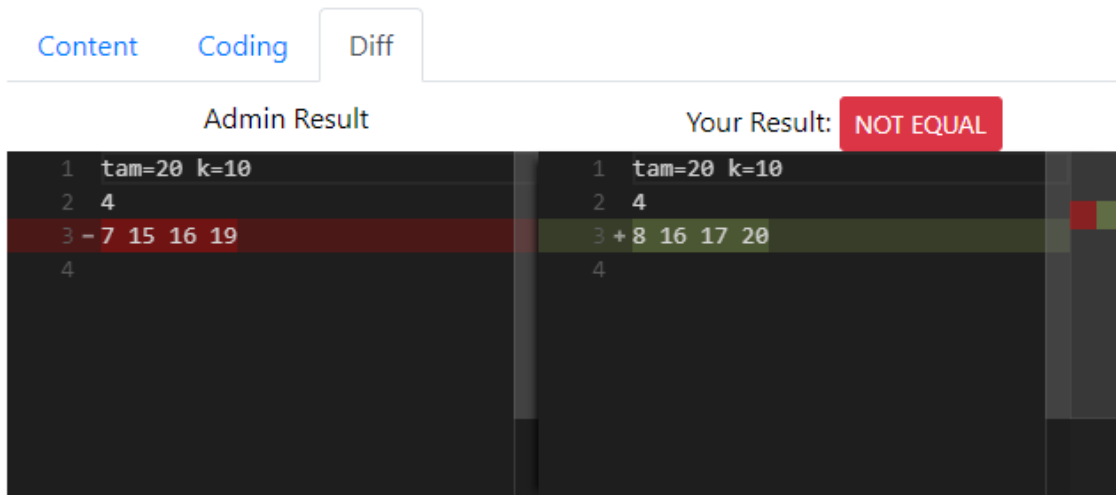


Figure 18: Comparison of user and admin results

2.4 Try a Code

Try a Code is a system screen that allows running a quick code without prior registration. Then, write or paste the program and run. These features are described in Section 2.3.2.

Admin Rules

Admin Roles

3.1 Default User Admin

By default, the system enables the default user and the password. At the first login, it is necessary to change the password for security reasons.

User: **user@user**

Password: **user**

3.2 Modules

When logged in a admin mode, the module panel (Figure 19) releases some features, including:

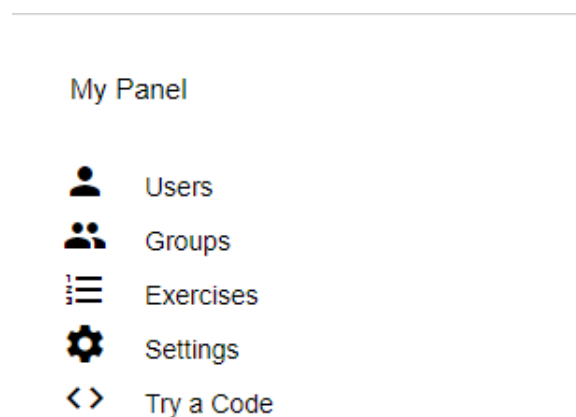


Figure 19: Admin modules

- **Users:** allows to create, change, or delete users.
- **Groups:** allows to create, change, or delete groups.
- **Exercises:** shows the exercises created by the admin.
- **Settings:** lists the main features of the system and the default values.
- **Try a Code:** shows the code editor and compiler to perform tests quickly.

3.3 Users

In this module, it is possible to register, change, and delete users. For example, if the system is configured for Self Registration, registered users will appear in the list shown in Figure 20.

Users

Add

Filter:



ID	Name	Email	Module	Action
2	Paulo S L de Souza	pssouza@icmc.usp.br	Admin	 

Figure 20: Users

3.4 Groups

In this module, it is possible to register, change, and delete groups. After creating a group, it is necessary to add users by clicking on the action button (Figure 21).

Groups

Add

Filter:

ID	Name	Action
1	Group 01	<div><div><div><div></div></div><div>(2)</div></div><div><div></div></div><div><div></div></div></div>

Figure 21: Groups

Type the user name inside the input and press enter to add, as shown in Figure 22.

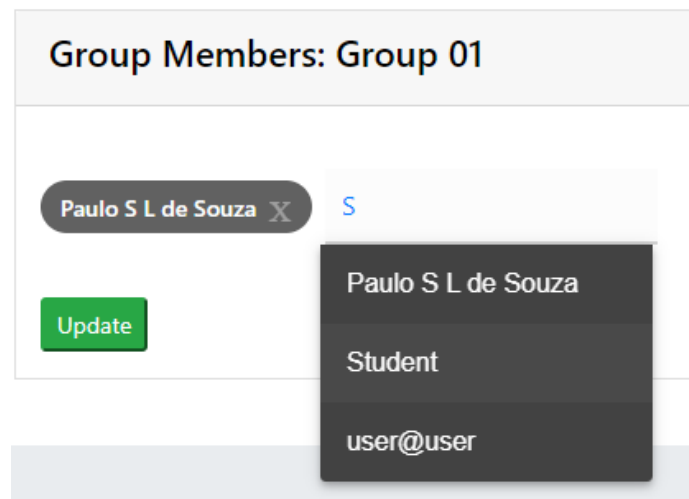


Figure 22: Members

3.5 Exercises

In this module, the system prints the exercises registered (Figure 23) by the admin. The system enables all options to the admin user: add, edit, or delete an exercise. Furthermore, in the Answers button, it is possible to view the activities sent by the users.

Exercises Add							
Filter: <input type="text"/>							
ID	Title	MNS/T	Expiration Date	RT	Code Analysis	Delivered	Action
5	Higher Value in MPI	10/15s	2021-08-20 18:00 <small>(more than 11 day(s))</small>	463.49ms	<div><div></div></div>	☰ (1)	🗑️ ✎️

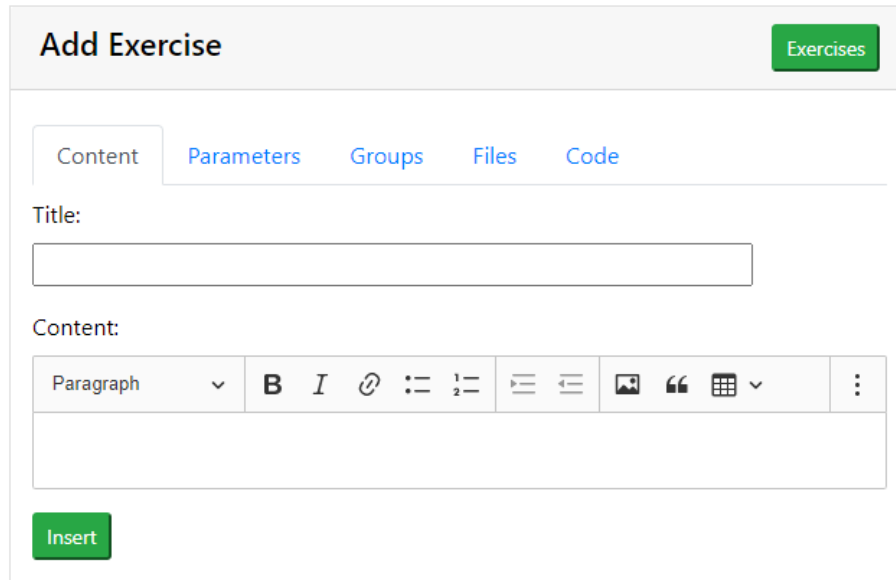
Figure 23: Exercises

- **Title:** displays the exercise title.
- **MSN/T:** displays the maximum number of submissions for this exercise and the maximum execution timeout.
- **Expiration Date:** shows the exercise's expiration date, and delivery after the deadline is no longer possible.
- **RT:** displays the response time of the exercise registered by the admin.
- **Code Analysis:** shows the automatic correction of the code semantics (code smell), for more information, see the Clang-Tidy documentation available at <https://clang.llvm.org/extra/clang-tidy/>.
- **Delivered:** allows to view and score the users' responses for this exercise.

The admin can click Add to enter a new exercise (Subsection 3.5.1) or click Delivered to view the submissions by students (Subsection 3.5.2).

3.5.1 Add an exercise

To submit an exercise, click the Add button. After, in the next screen shown in Figure 24, the admin sets the title, content, expiration date, and code in the next screen.



The screenshot shows a web interface titled "Add Exercise" with a green "Exercises" button in the top right. Below the title is a tabbed interface with five tabs: "Content" (selected), "Parameters", "Groups", "Files", and "Code". Under the "Content" tab, there is a "Title:" label followed by a text input field. Below that is a "Content:" label followed by a rich text editor. The rich text editor has a toolbar with options: "Paragraph" (dropdown), "B" (bold), "I" (italic), a link icon, "List" (bulleted), "List" (numbered), "List" (checkbox), "List" (table), "Image" (insert), "Quote" (insert), "Table" (insert), and a "More" (three dots) icon. Below the rich text editor is a green "Insert" button.

Figure 24: Exercise content

Content Tab In this tab, the admin defines the title and content of the exercise.

Parameters Tab This section defines code execution parameters by users.

- **Expiration Date:** sets the time and date limit for exercise submission by the user.
- **Maximum Number of Submission:** sets the maximum number of attempts a user can submit an exercise.
- **Maximum Timeout (seconds):** sets the timeout that user code can be running.
- **Maximum file size:** sets the maximum file size a user can upload to the code.

Content
Parameters
Groups
Files
Code

Expiration Date:

yyyy-mm-dd

↑

HH

↓

:

↑

MM

↓

Maximum Number of Submissions:

Maximum Timeout (seconds):

Maximum file size that the student can use with input (bytes).
Leave zero to disable:

Figure 25: Exercise parameters

Groups Tab In this tab, the admin inserts the groups to access the exercise.

Content
Parameters
Groups
Files

Save the exercise first to enter a group.

Group 01 X

New group...

Figure 26: Exercise groups

Files Tab In this tab, the admin inserts the test files used by the users when executing the code.

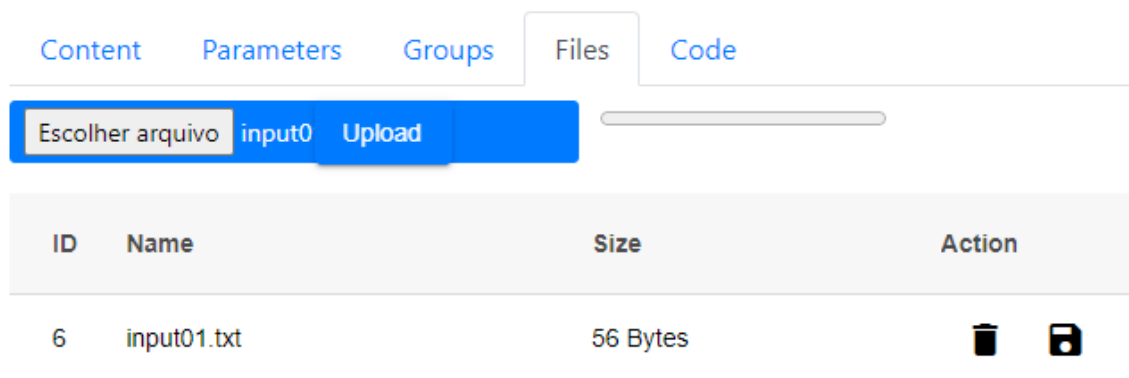


Figure 27: Exercise files

Code Tab For details on this tab, see Subsection 2.3.2.

3.5.2 Answers

In the list of user responses, it is possible to check the user's name, the number of attempts, the execution time, code analysis, status, score, and feedback (Figure 28).

Last Exercise Answers: Higher Value in MPI						Exercises	Add
Time to finish: 11d 3h 26m 46s							
Filter:						Export Results ▼	
ID	Name	Last Submission	Last Result	Score / Feedback	Attempts Number		
1	Student	2021-07-16 17:38	EQUAL	No/No	1		
Number	Submission	Code Analysis	Response Time	Events	Status	Action	
1	2021-07-16 17:38		463.15ms	2	EQUAL	<>	
Items per page: 25						1 – 1 of 1	

Figure 28: User answers

In addition to the parameters already explained in Section 2.3, the admin can define the score and feedback for the exercise on this screen, as shown in Figure 29

Change Score

Please enter the score:

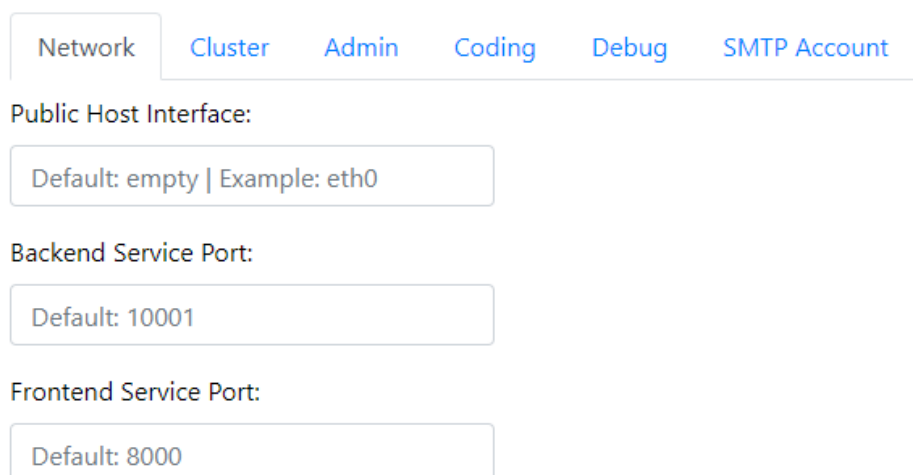
Score	Feedback *(Optional)
-------	----------------------

Yes No

Figure 29: Score and feedback of the answer

3.6 Settings

In the settings module, it is possible to change several system configurations, according to Figure 30.



The screenshot shows the 'Settings' module with the 'Network' tab selected. The 'Cluster' tab is also visible. The 'Public Host Interface' field has a default value of 'empty' and an example of 'eth0'. The 'Backend Service Port' field has a default value of '10001'. The 'Frontend Service Port' field has a default value of '8000'.

Figure 30: Settings

Network Tab

- **Public Host Interface:** defines which network interface back-end and front-end will use to listen to network services.
- **Back-end Service Port:** sets the port value to list the services between the back-ends in the network (default: 10001).
- **Front-end Service Port:** sets the port value to list the services for the front-ends (default: 8000).

Cluster Tab

- **Cluster:** the default cluster name (default: CLUSTER01).
- **Cluster Password:** defines the cluster password for connection between nodes. If the password is left empty, the nodes will connect automatically if their operating modes are set to NODE (default: empty).
- **Operation Mode:** the default operation mode to start on back-end init (default: LOCAL).
- **Number of Nodes:** the number of containers (virtual nodes) to be started on back-end init (default: 0).
- **Maximum Number of Nodes:** the maximum number of containers running as nodes (default: 30).

Admin Tab

- **Host User:** sets the default user for the host and for accessing the front-end Administrator roles (default: user@user).
- **Host Password:** sets the default user password (default: user).
- **Self Registration:** allows users to create their own logins on the server (default: On)

Coding Tab

- **Code Execution Timeout:** defines a timeout to code execution in seconds (default: 20s).
- **Queuing System:** if enabled, a queue system will prevent two codes from running at the same time. If disabled, it will become optional for each run (default: Off).
- **Maximum Queue Concurrency:** defines the maximum number of users in the queue, competing for resources (default: 30).

Debug Tab

- **Debug:** shows debug messages on the back-end (default: Off).
- **Error:** shows error messages on the back-end (default: Off).

SMTP Account Tab

- **Smtp Server:** server address responsible for sending the account registration and password recovery emails.
- **Smtp Port:** mail server port.
- **Smtp User:** registered user for sending emails.
- **Smtp User Password:** user Password registered for sending emails.

3.7 Try a Code

Try a Code is a system screen that allows running a quick code without prior registration. Then, write or paste the program and run. These features are described in Section [2.3.2](#).

Cluster Operation

4.1 Operation mode

As shown in Figure 31, the system has three operation modes that define the cluster behavior. These modes control the connection between hosts. For example, we can have two machines on the same network, one the master and the other the node. These hosts will group in and increase the number of nodes to run parallel programs.

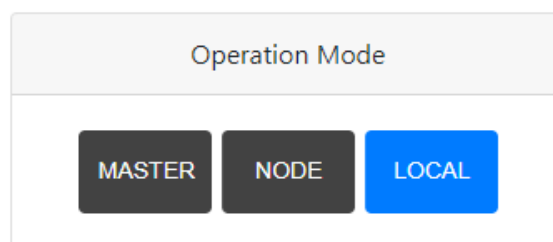


Figure 31: Operation mode

- **Master:** in this mode, the system will be the master and announce via broadcasting, its Cluster ID, and its variables for connection. When selected, the administrator must put a name on the Cluster and, optionally, a password (Figure 32).

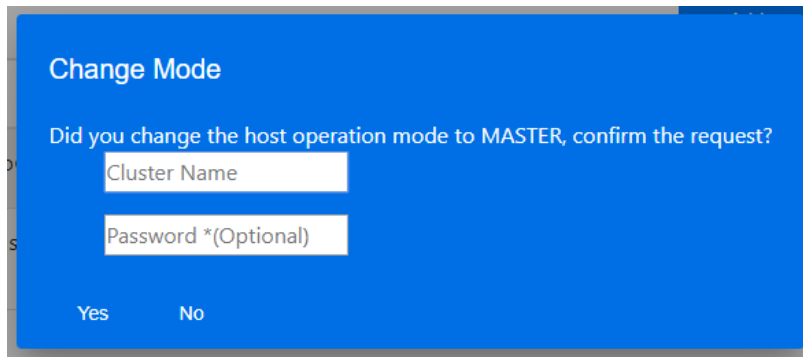


Figure 32: Defining a cluster name and password

- **Node:** in this mode, the host is on standby, waiting for some master on the network. If found, the node will automatically connect if the cluster does not have an entry password; otherwise, the node needs the correct password.
- **Local:** this option allows the user to work outside a cluster on the network or in an isolated virtual machine without conflicting with other hosts.

4.2 Clusters available to connect

This section shows the masters on the network available for connection. If the master has a secret key (Figure 33), the nodes will need the password to connect to the cluster. If no private key is set to cluster, the nodes will connect automatically.

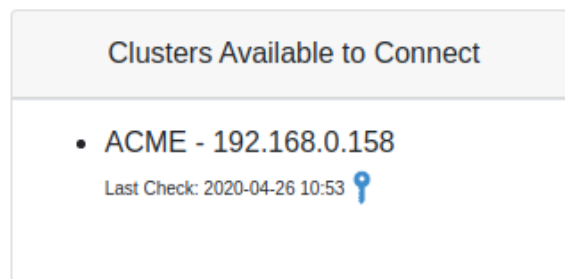


Figure 33: Clusters available to connect

4.3 Current nodes in the cluster

When Iguana is set to master or local mode, it is possible to define the number of nodes available to execute the parallel codes (Figure 34). The admin can increase or decrease the number at any time.

If the network has more than one host connected to the cluster, the nodes are distributed equally and automatically. Therefore, if one host goes offline, the other hosts will continue to function normally without action.

If another host arrives on the network (in node mode) and connects to the cluster, these nodes will run parallel codes in the next change in the node number.

Current Nodes in the Cluster

OK

Figure 34: Current nodes in the cluster

4.4 Active nodes

This section (Figure 35) shows the active hosts and nodes in the system.

hal11 (Mem: 31958MB | vCPU: 8)



Total cluster: vCPUs: 8 | Mem: 31958MB

Figure 35: Active nodes

4.5 Active front-ends

This section (Figure 36) shows all users logged into the system, their IPs, and hosts. Thus, it is still possible to view the user's last activity on the cluster.

Active Frontends

- user@user
OpMode: LOCAL | Last Activity: 2020-04-25 21:32
172.31.19.51 | ip-172-31-19-51

Figure 36: Active front-ends

INSTALLATION SCRIPT

This appendix contains the Iguana installation script. Users can use this script to install the tool directly on the Linux operating system. Tested and valid on Ubuntu 21.04 operating system.

Source code F.1 – Iguana installation script

```

1: #!/bin/sh
2: #####
3: #   Iguana Installation Script           #
4: #   Doctoral Project - ICMC USP         #
5: #   Author: Naylor Garcia Bachiega     #
6: #   Date: 2019-09-10                   #
7: #####
8: NAME=cluster
9: DIR=/usr/local
10: SRC=$DIR/src
11: BIN=$DIR/bin
12: ETC=$DIR/etc/$NAME
13: NFS=$NAME
14: SHARED=$NFS/shared
15: MASTER=$SHARED/master
16: SERVICE=/var/lib/systemd/system
17: RUN=/run/$NAME
18: INIT=/etc/init.d
19: STATIC=/usr/local/share/$NAME
20:
21: echo "\nProgram name: $NAME"
22: echo "Instalation dir: $DIR"

```

```
23: echo "NFS dir: $NFS"
24: echo "Configuration dir: $ETC\n"
25:
26: func_packages() {
27:     LIST_OF_APPS="git golang-go golang docker-compose docker.io nfs-kernel-
        server nfs-common clang-tidy"
28:     apt update
29:
30:     for p in $LIST_OF_APPS
31:     do
32:         if [ $(dpkg-query -W -f='${Status}' $p 2>/dev/null | grep -c "ok
            installed") -eq 0 ];
33:         then
34:             apt-get install -y $p;
35:         fi
36:     done
37: }
38:
39: func_mkdirs() {
40:     LIST_OF_DIRS="$NFS $SHARED $MASTER $ETC $SERVICE $RUN $STATIC"
41:
42:     for d in $LIST_OF_DIRS
43:     do
44:         if [ -d "$d" ]; then
45:             echo "$d find!"
46:         else
47:             mkdir $d
48:             echo "Creating $d directory"
49:         fi
50:     done
51: }
52:
53: func_clone() {
54:     if [ -d "$SRC/$NAME" ]; then
55:         cd "$SRC/$NAME"
56:         echo "start git pull!"
57:         git pull
58:     else
59:         echo "start git clone!"
60:         cd $SRC
61:         git clone https://github.com/naylor/$NAME
62:     fi
```

```
63: }
64:
65: func_build() {
66:     export GOPATH=$SRC/$NAME/
67:     rm -fr $SRC/$NAME/src
68:     cd $SRC/$NAME/backend
69:     cp kill.sh $NFS
70:     go get -d
71:     rm -fr ../src/github.com/docker/docker/vendor/github.com/docker/go-
        connections/
72:     go get -d
73:     go build -o $NAME *.go
74:     cp $NAME $BIN/
75:     chmod u+x $BIN/$NAME
76:     cp ../config/config.yaml $ETC/
77: }
78:
79: func_containers() {
80:     cd $SRC/$NAME/containers
81:     docker-compose build
82: }
83:
84: func_frontend() {
85:     cd $SRC/$NAME/frontend
86:     cp -fr dist/frontend/* $STATIC/
87: }
88:
89: func_clusterService() {
90:     cd $SRC/$NAME
91:     cp $NAME.service $SERVICE/
92:     chmod u+x $NAME
93:     cp $NAME $INIT/
94:     systemctl daemon-reload
95:     systemctl enable $NAME
96: }
97:
98: ##### MAIN #####
99: /etc/init.d/cluster stop
100:
101: dhclient
102:
103: func_packages
```

```
104: func_mkdirs
105: func_clone
106: func_build
107: func_containers
108: func_frontend
109: func_clusterService
110:
111: /etc/init.d/cluster start
112: /etc/init.d/cluster status
113:
114: exit
```
