processes

MDPI

Article

TopADDPi: An Affordable and Sustainable Raspberry Pi Cluster for Parallel-Computing Topology Optimization

Zhi-Dong Zhang [1,*], Dao-Yuan Yu [1], Osezua Ibhadode [2], Liang Meng [1,*], Tong Gao [1], Ji-Hong Zhu [1] and Wei-Hong Zhang [1,*]

[1] State IJR Center of Aerospace Design and Additive Manufacturing, School of Mechanical Engineering, Northwestern Polytechnical University, Xi'an 710072, China
[2] Multi-Scale Additive Manufacturing Lab, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada
* Correspondence: zhidong.zhang@nwpu.edu.cn (Z.-D.Z.); liang.meng@nwpu.edu.cn (L.M.); zhangwh@nwpu.edu.cn (W.-H.Z.)

Abstract: Parallel-Computing Topology Optimization (PCTO) has gained importance, especially with the advancement of additive manufacturing (AM), due to its ability to tackle high-dimensional, high-resolution challenges. PCTO is highly relevant to sustainable manufacturing processes and technologies, enabling resource-efficient designs, reduced emissions, and advancements in Industry 4.0 integration. However, PCTO poses difficulties for newcomers or researchers, mainly because of its reliance on non-traditional computing environments and the limited availability of high-performance computing (HPC) resources. Addressing this, the study introduces TopADDPi, a Raspberry Pi-based cluster system, which has been purpose-built to facilitate learning and research in PCTO. It provides detailed instructions for assembling and configuring a Raspberry Pi cluster, with a focus on cost-effectiveness and ease of use. The study thoroughly investigates how different hardware and software configurations affect computing efficiency. In addition, through extensive numerical testing, the performance, energy consumption, and environmental impact of the Raspberry Pi cluster are benchmarked against conventional computing systems. The findings demonstrate the cluster's advantages in handling parallel computing, its indispensable role in debugging, its remarkable energy efficiency, and its significantly reduced carbon footprint compared to conventional systems. These attributes establish the Raspberry Pi cluster as an invaluable tool for both educational and research applications in structural engineering, offering an affordable, sustainable, and indispensable solution for PCTO.

Keywords: topology optimization; raspberry pi cluster; parallel computing; arbitrary design domain; sustainability; 3D printing

check for updates

Academic Editor: Raul D. S. G. Campilho

Received: 15 January 2025
Revised: 17 February 2025
Accepted: 21 February 2025
Published: 24 February 2025

Citation: Zhang, Z.-D.; Yu, D.-Y.; Ibhadode, O.; Meng, L.; Gao, T.; Zhu, J.-H.; Zhang, W.-H. TopADDPi: An Affordable and Sustainable Raspberry Pi Cluster for Parallel-Computing Topology Optimization. *Processes* **2025**, *13*, 633. https://doi.org/10.3390/pr13030633

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Topology optimization (TO) has become essential in designing high-performance structures, especially with the rise in additive manufacturing (AM) [1–4]. This method optimizes material distribution within a design space and is highly relevant to modern manufacturing processes, transportation, and technologies, enabling resource-efficient designs, reduced greenhouse gases emissions, and advancements in Industry 4.0 integration [5]. The detrimental impact of greenhouse gasses on the environment has attracted the attention of researchers in both industry and academia [6,7]. However, TO faces challenges when it comes to addressing high-dimensional, high-resolution problems [8]. High

resolutions are preferred for detailed features and smoother surfaces are suitable for AM, a key aspect of advanced manufacturing and precision engineering. In contrast, low resolutions can lead to suboptimal designs due to artificial length-scale constraints from the coarse grids [9]. Therefore, managing high-resolution TO is critical, despite substantial computational demands.

The need for increased computational resources due to larger numbers of finite elements has led to the widespread adoption of parallel computing in TO [10]. Parallel computing divides large problems into smaller, concurrently solved segments, enabling an efficient data exchange and faster iterations. Parallel-Computing Topology Optimization (PCTO) has been under investigation for over two decades, with Borrvall and Petersson [11] being pioneers in applying parallel computing to large-scale TO. Subsequent studies, such as that of Aage et al. [9], have developed flexible, open-source frameworks for PCTO, contributing significantly to the field of sustainable processes and green manufacturing. Researchers have expanded these frameworks for various applications, including designing high-resolution support structures [12,13], and incorporating features like minimum and maximum feature size [14] and arbitrary design domain initialization [15], all of which are available as open-source code.

These open-source programs are invaluable for newcomers and researchers focused on PCTO, offering hands-on experience with code and the opportunity to obtain optimized structures [9]. However, the predominance of C++ code and the requirement for a Linux operating system (OS), e.g. Ubuntu, with application software installations can pose challenges, especially for those who are only familiar with Windows environments. In addition, debugging these programs often requires a distributed-memory computational environment like an in-house cluster or high-performance computing (HPC).

Raspberry Pi (RPi) clusters present an affordable solution for learning and research in parallel computing, cluster management, and distributed systems [16]. These clusters have been used for cost-effective experimentation in computing, offering an efficient and eco-friendly option for multi-node system applications [17–20]. However, there is limited research on using Raspberry Pi clusters specifically for TO.

This study presents TopADDPi, an RPi-based cluster system specifically developed to support the open-source TopADD code [15] and other codes to enhance the learning, programming, and mastery of PCTO. Firstly, this study will demonstrate TopADDPi's usefulness to users with limited resources or experience in high-performance computing environments. Then, it will offer detailed guidelines on constructing an RPi cluster tailored for topology optimization tasks. Additionally, the study will thoroughly evaluate the cluster's effectiveness and efficiency. This evaluation will focus on how TopADDPi stands as a practical and eco-friendly solution for handling the computational demands of PCTO. Furthermore, the TopADDPi cluster is suitable for additive technologies like 3D printing, enabling exploration of sustainable supply chains and process designs.

2. Motivation

The increasing complexity of large-scale topology optimization and corresponding computational tasks have underscored the critical need for a dedicated learning and research platform in the field of parallel computing. As modern applications leverage parallelism for the purpose of performance enhancement, mastering the intricacies of parallel computing becomes imperative for structural engineers. The need for an in-house Raspberry Pi (RPi) cluster is demonstrated, followed by two specific requirements for the cluster.

2.1. Need for an In-House Cluster to Learn PCTO

The development of PCTO program necessitates access to an actual computing cluster. While debugging in a serial or single-node setup is feasible, it is crucial to thoroughly familiarize newcomers or researchers with the parallel-computing environment. Newcomers or researchers often face limited computational resources; they are typically restricted to a single computer, frequently one running a Windows operating system. These users would need to install a Linux system or a virtual Linux machine on their existing Windows machines to perform parallel computing tasks. This setup presents several challenges: dual-boot complexity, loss of Windows functionality, and resource limitations in virtual machine.

Firstly, installing Linux alongside Windows often requires setting up a dual-boot system, which can be technically challenging for users who are unfamiliar with system partitioning and bootloaders. Secondly, when running Linux, users lose access to their familiar Windows environment, which they may rely on for tasks such as searching for solutions to technical problems, accessing documentation, or using Windows-specific tools. Thirdly, virtual machines are often used as an alternative to dual-boot setups; however, they are not native systems and suffer from performance overheads, limited hardware access, and incomplete functionalities. These hurdles can significantly impede users' learning and research process, leading to frustration and potential abandonment of their educational and research journey, as shown in Figure 1.

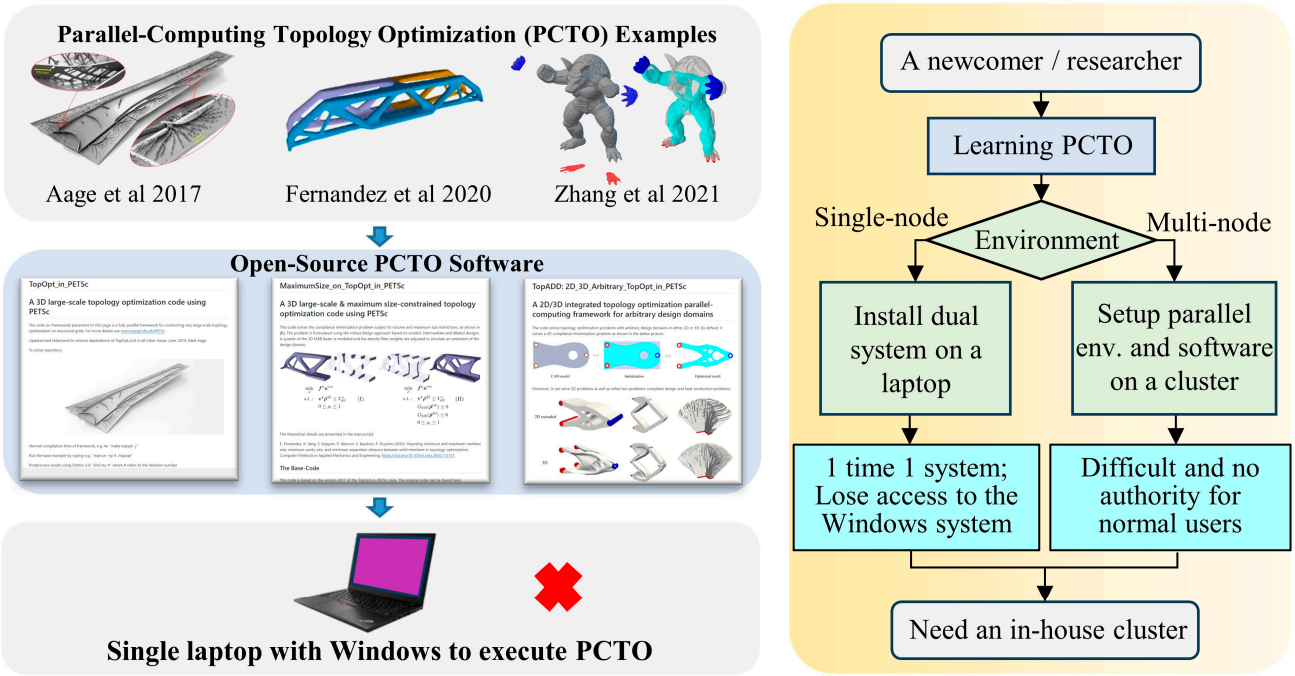


Figure 1. Schematic depicting the complexities and challenges of mastering parallel-computing topology optimization and operating its related open-source software, culminating in the need for an in-house cluster. Adapted with permission from Refs. [9,14,15].

Additionally, accessing high-performance computing (HPC) resources can be daunting for newcomers or researchers, especially when faced with an unfamiliar HPC environment and the lack of skilled technicians for support. The process of setting up complex application software like PETSc—a parallel numerical software library for partial differential equations and sparse matrix computations [21]—can be overwhelming, even for a normal programmer, let alone a beginner.

Therefore, having an affordable and accessible real cluster is vital for newcomers or researchers. A Raspberry Pi-based cluster can be an excellent solution. It not only prompts the

learning and research process for PCTO but also provides hands-on experience with parallel computing with particular focus on distributed-memory systems. This practical exposure is instrumental in helping people gain a deeper and more realistic understanding of parallel computing, thereby enhancing their ability to develop more effective PCTO programs.

2.2. Need for ARM Computers

The Advanced RISC Machines (ARM) architecture is renowned for its power efficiency, simplicity, and flexibility, and is particularly suitable for energy-efficient mobile devices like smartphones, tablets, and laptops [22]. Given the pronounced architectural differences between ARM and common platforms like ×86, which are prevalent among desktop computers and laptops, it is crucial to conduct thorough testing on both ×86 and ARM platforms for software development targeted at various devices. The Raspberry Pi is a widely acclaimed single-board computer leveraging ARM architecture. It serves as an exemplary tool for debugging and testing code tailored for ARM Central Processing Units (CPUs) [23]. Figure 2 illustrates the process of utilizing a Raspberry Pi to identify a bug that is challenging to detect on an ×86 platform. The bug, arising from casting negative float to an unsigned int in C++ [24], results in architecture-dependent behavior. The corresponding code is provided in Appendix A.

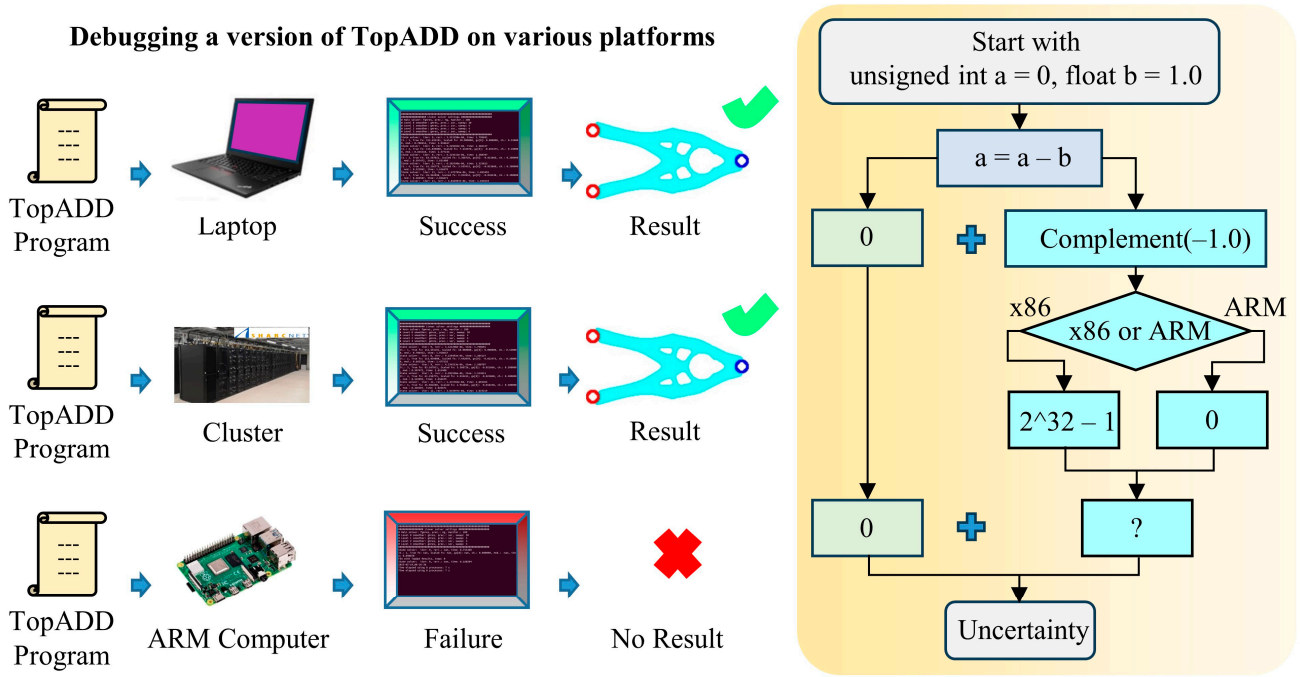


Figure 2. Schematic of utilizing a Raspberry Pi for undefined-behavior bug identification, showing the necessity of bug detection in ARM platforms.

2.3. Need for Distributed-Memory Systems

The escalating demand for computational power and scalability underscores the imperative need for distributed-memory systems. Achieving efficient communication and data sharing among processes is essential. For example, techniques like MPI_Win which can be used for communication on the same node can enhance overall performance and scalability in parallel applications [25]. However, transitioning a program designed for shared-memory parallel computing to a distributed-memory system presents challenges. Despite successful execution on laptops with ×86 architecture and Raspberry Pis with ARM architecture, issues may arise on clusters, as shown in Figure 3. This can be attributed to the malfunctioning of MPI_Win on the cluster without meticulous configuration, which

emphasizes the crucial role of a distributed-memory cluster in code debugging. Therefore, a convenient distributed-memory testbed emerges as indispensable in the development of parallel-computing programs. The corresponding code is provided in Appendix B.

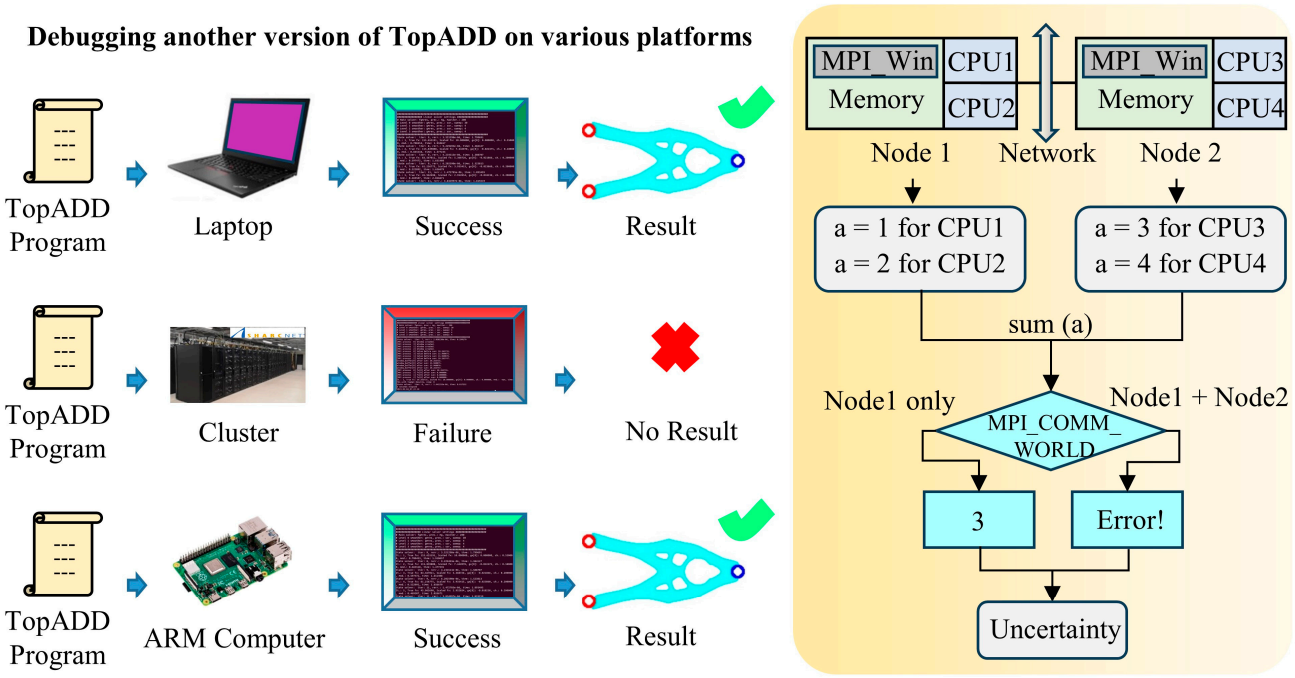


Figure 3. Schematic of utilizing a distributed-memory system for distributed-memory bug identification, showing the necessity of bug detection in multi-node platforms.

3. Building the Cluster for PCTO

This section introduces a Raspberry Pi cluster specifically designed for learning and researching PCTO. Comprehensive details on configuring and setting up this cluster have been made publicly available on GitHub at: https://github.com/wonderfulzzd/Pi_Cluster_for_TopADD (accessed on 1 January 2024).

3.1. Existing Raspberry Pi Clusters

Several projects have previously assembled RPis into a variety of clusters. Table 1 lists nine existing Raspberry Pi clusters developed at different universities and institutions. During the initial release of the first-generation Raspberry Pi Model B (RPi Model B) in 2012, several pioneering projects utilized it to construct RPi clusters. These early endeavors, now over a decade old, were based on the earlier Raspberry Pi models. With the evolution and advancement of Raspberry Pi technology, the exploration of newer models has become increasingly relevant. Consequently, Table 1 includes several recent projects that employ the latest Raspberry Pi 4 Model B (RPi 4 Model B) for building parallel-computing clusters, which was the latest version available during the time of this study.

These contemporary projects delve into various performance aspects of the clusters, examining factors such as cost-efficiency, input/output (I/O) performance, and capabilities in scientific computing. The examples presented in Table 1 represent just a fraction of the myriad projects that have been developed around Raspberry Pi clusters in the literature. This subset highlights an active community of developers and researchers. The ongoing interest in constructing and utilizing Raspberry Pi clusters underscores their potential as a valuable tool in both research and educational spheres. While these initiatives have contributed to the field, it should be noted that none of them were specifically designed for PCTO.

Table 1. Existing Raspberry Pi clusters.

	Year	RPi Version	RPi Number	Purpose
University of Southampton [17]	2012	RPi Model B	64	Testing MPI functionality
Boise State University [18]	2013	RPi Model B	32	Testing MPI functionality
University of Glasgow [19]	2013	RPi Model B	56	Testbed for cloud computing
Free University of Bozen-Bolzano [20,26]	2013	RPi Model B	300	Testbed for cloud computing; Container-based edge cloud
Prometheus Research [27]	2013	RPi Model B	2	Book on building a cluster and exploring application software
Daejin University [28]	2022	RPi 4 Model B	16	Testing cost-performance
Deakin University [29]	2022	RPi 4 Model B	12	Testing performance of scientific computing
Federal University of Sergipe [30]	2022	RPi 4 Model B	9	Testing performance of big data Input/Output (I/O)
Hellenic Mediterranean University [31]	2023	RPi 4 Model B	24	Testing performance of scientific computing

3.2. System Specification of Raspberry Pi 4 Model B

Raspberry Pi 4 Model B will be employed as building blocks of the RPi cluster. Figure 4 shows the most prominent components on a Raspberry Pi 4 Model B board. The heart of the RPi is an integrated circuit Broadcom BCM2711 (System on Chip, SoC), combining a Quad core ARM Cortex-A72 1.5 GHz CPU and a Broadcom VideoCore VI Graphics Processing Unit (GPU). The Random Access Memory (RAM) can be 4 GB/8 GB LDDR4 SDRAM, depending on the choice of users. Moreover, there is an SD card slot for storage, as well as I/O units such as Universal Serial Bus (USB), USB-C as power supply port, a Gigabit ethernet port, audio, and 2 micro-High-Definition Multimedia Interface (HDMI) ports. Table 2 summarizes the hardware specifications of the RPi [32].

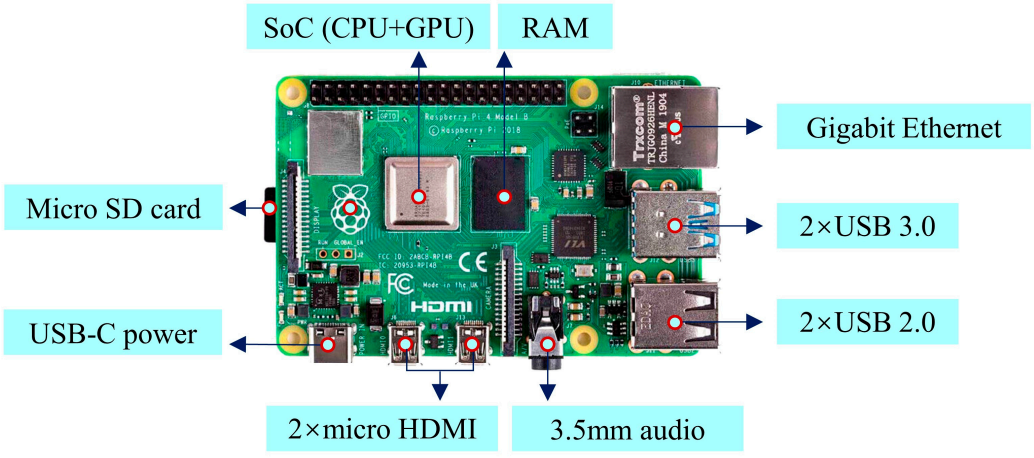


Figure 4. Illustration on the most prominent components on the Raspberry Pi 4 model B board.

For the system software, the Raspberry Pi Foundation typically recommends Raspbian as the OS of choice. Raspbian is a tailored version of the well-known Debian Linux distribution. However, in this case, Ubuntu 20.04 has been installed on the Raspberry Pis to facilitate a more consistent comparison with desktop and laptop environments. Ubuntu is a widely recognized Linux distribution that is also based on Debian and offers a more familiar environment for those accustomed to traditional computer systems. This approach

allows for a more direct and relevant comparison of performance and usability between Raspberry Pi devices and standard computers.

Table 2. Raspberry Pi 4 Model B hardware specifications.

Component Name	Specifications
Integrated circuit	Broadcom BCM2711
CPU	Quad core ARM Cortex-A72 1.5 GHz processor
GPU	Broadcom VideoCore VI 0.5 GHz processor
Memory	4 GB/8 GB LDDR4 SDRAM
Ethernet	Gigabit ethernet port
USB	2 USB 2.0 ports, 2 USB 3.0 ports
Video output	2 micro-HDMI ports
Audio output	3.5 mm port, HDMI
Storage	SD card slot

Details concerning the installation of Ubuntu and other application software on the Raspberry Pis will be provided. This information will include step-by-step guidance for setting up the Raspberry Pi devices with Ubuntu, ensuring the devices are well prepared for the tasks and comparisons intended in this study.

3.3. Cluster Hardware Configurations

The multitude of projects utilizing Raspberry Pis inspired the authors to set up a new Raspberry Pi cluster for PCTO. In this work, an RPi with 8 GB RAM is used as a login node and 4 RPis with 4 GB RAM are employed as compute nodes. Figure 5 illustrates the components of this Raspberry Pi cluster as well as its final assembly.

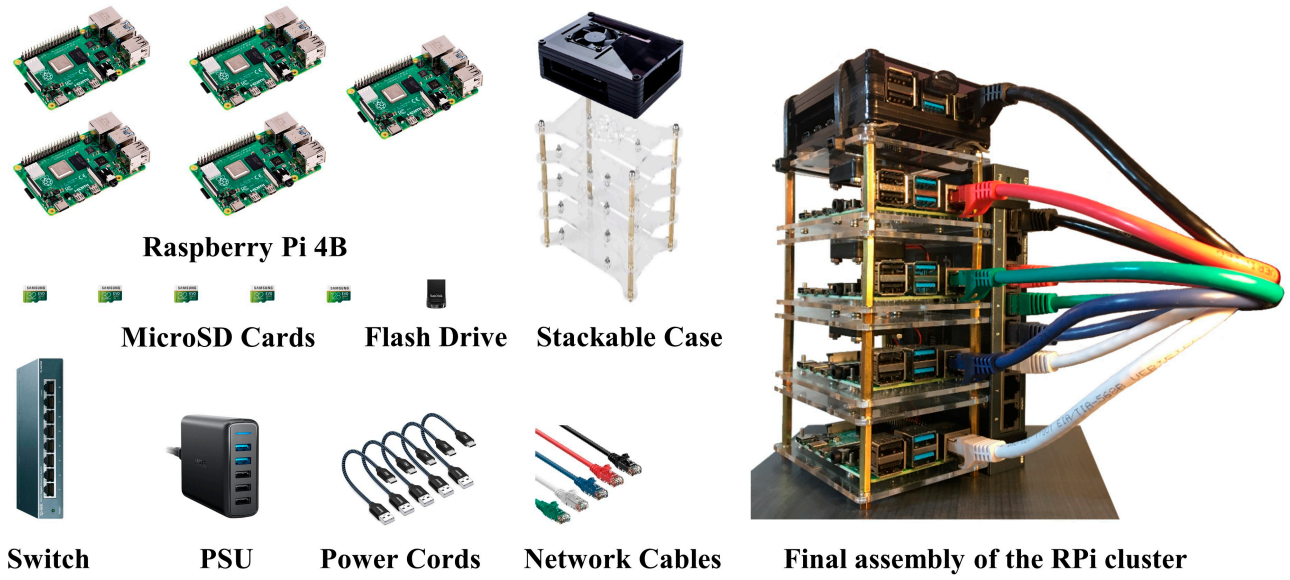


Figure 5. Components of the RPi cluster and its final assembly outlook.

The cluster is designed with the capability to support graphical output, allowing for the connection of a monitor, as depicted in Figure 6a. Such graphical interface support is particularly advantageous for newcomers or researchers who may prefer interacting with graphical operations rather than command-line interfaces. It is important to note that a monitor is not included in the overall expense of the cluster, as its desktop environment can be accessed remotely via a laptop or other computers, eliminating the necessity for a dedicated monitor. Figure 6b presents a schematic of the cluster setup, illustrating the

connections between the monitor, compute nodes, login node, switch, and Network File System (NFS).

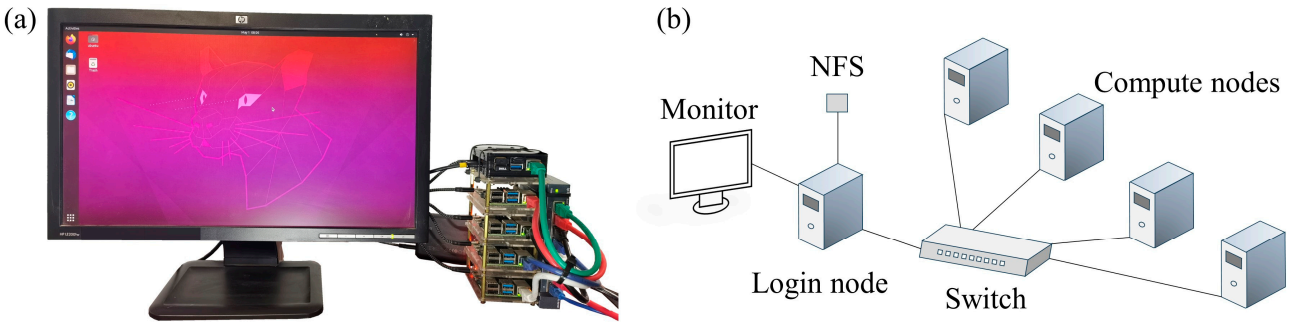


Figure 6. RPi cluster connected with a monitor, (**a**) final assembly outlook, and (**b**) its corresponding setup schematic.

The cost breakdown of the five-node RPi cluster is illustrated in Figure 7. All the components were purchased from Amazon.ca. The prices listed reflect the cost as of December 2019, inclusive of taxes and shipping fees. The total expenditure for the setup is presented in Canadian dollars (CAD), amounting to CAD 730, which is equivalent to approximately USD 550.

	Product	Qty	Unit Price (CAD)	Total Price (CAD)
Raspberry Pi		5	92	460
MicroSD Cards		5	15.4	77
Flash Drive		1	25	25
Switch		1	30	30
PSU		1	50	50
Cable and Cases		1	88	88
Sum				**730**

Figure 7. Cost breakdown of the five-node RPi cluster.

To investigate the impact of hardware configurations on the performance of the RPi cluster, three key aspects will be examined:

(a) Internet cable influence: The study will compare two types of internet cables: CAT6 and CAT8. The CAT6 cable supports data transfer rates of up to 10 gigabits per second (Gbps), whereas the CAT8 cable can handle up to 40 Gbps. This comparison aims to understand how these different capabilities affect the cluster's data transfer efficiency.

(b) NFS influence: The NFS, where all application files are installed and stored, is a critical component. This research will explore the performance differences between two types of NFS hardware: a flash drive with a capacity of 128 GB and a Non-Volatile Memory express (NVMe) solid-state drive (SSD) with 256 GB. The flash drive offers read and write speeds of 130 MB/s and 50 MB/s, respectively, while the NVMe SSD provides significantly faster speeds of 2400 MB/s for reading and 1950 MB/s for writing.

(c) Switch influence: The network switch plays a vital role in the communication among the cluster's nodes, thereby impacting the overall performance of the cluster. This study will investigate the performance differences between switches with 100-Megabit and 1-Gigabit capacities to gauge the extent of their influence on the cluster's efficiency.

3.4. Cluster Software Configurations

Comprehensive details on configuring and setting up this cluster have been made publicly available on GitHub at https://github.com/wonderfulzzd/Pi_Cluster_for_TopADD (accessed on 1 January 2024). Readers interested in constructing their own Raspberry Pi cluster are encouraged to refer to these step-by-step procedures whether for research or educational purposes.

The OS installed on all Raspberry Pis in this project is Ubuntu 20.04, sourced from the official Raspberry Pi website. Each Raspberry Pi's SD card is prepared with this OS image using the Raspberry Pi Imager. Following the image flashing, the systems require configuration, including various application software installation, such as MPI and PETSc software.

Furthermore, the study will briefly delve into the impact of software configuration on the cluster's performance. This includes examining the effects of software's different optimization levels, installation locations, and CPU overclocking settings.

Lastly, a flowchart describing the assembly and configuration process is plotted in Figure 8 to enhance clarity and reproducibility.

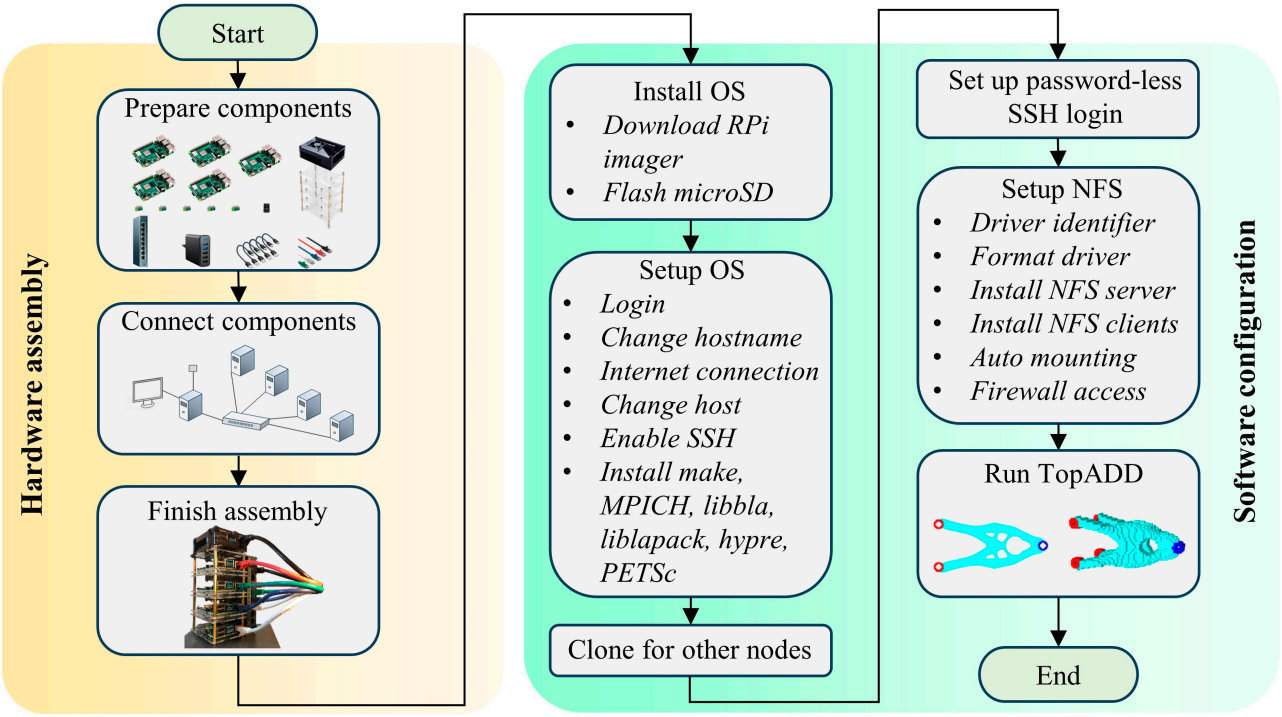


Figure 8. Flowchart describing the hardware assembly and software configuration process.

4. Topology Optimization Problem

The TopADD program [15] encompasses three distinct topology optimization (TO) problems: minimum compliance, a compliant mechanism, and heat conduction. These problems are schematically represented in Figure 9. The primary goal in each TO problem is to determine the optimal distribution of material that minimizes the objective function. This could be the compliance of structures, negative displacement, or heat compliance. They are all subjected to predefined boundary and loading conditions.

$$\begin{aligned} \underset{\boldsymbol{\rho}}{Min.} \quad & c(\boldsymbol{\rho}) = \boldsymbol{L}^T \boldsymbol{U}(\boldsymbol{\rho}) \\ s.t. \quad & \boldsymbol{K}(\boldsymbol{\rho})\boldsymbol{U}(\boldsymbol{\rho}) = \boldsymbol{F} \\ & g(\boldsymbol{\rho}) = \frac{V(\boldsymbol{\rho})}{V_0} = \frac{1}{V_0} \sum_{i=1}^{n} \rho_i \leq \overline{V}_{\mathrm{f}} \cdot \\ & 0 < \rho_i \leq 1, \; i = 1, \, 2, \cdots, \, N_e \end{aligned} \tag{1}$$

where $\boldsymbol{U}$ and $\boldsymbol{F}$ are the solution and load vectors, respectively. The vector $\boldsymbol{L}$ serves as an auxiliary constant vector. In the context of minimum compliance and heat conduction problems, $\boldsymbol{L}$ is equivalent to $\boldsymbol{F}$. However, for the compliant mechanism problem, $\boldsymbol{L}$ is a unique constant vector characterized by zeros at all degrees of freedom except at the output point's degree of freedom, where it is set to -1.

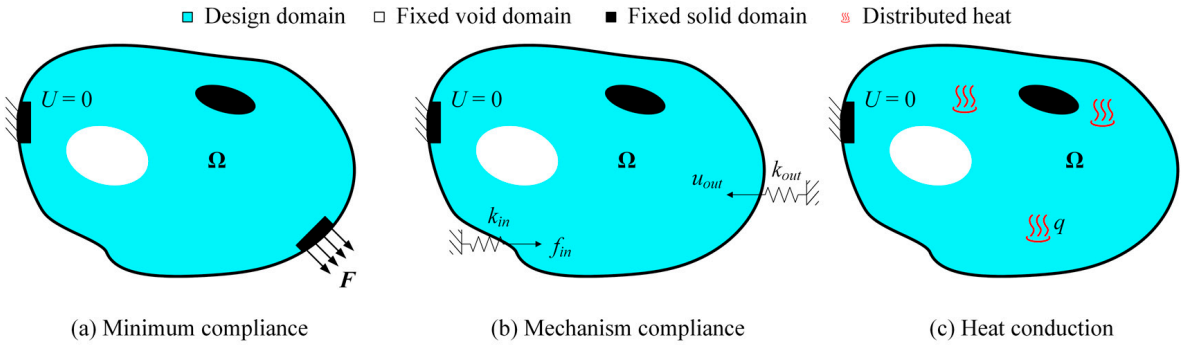


Figure 9. Schematic of three representative topology optimization problems.

The term $\boldsymbol{K}(\boldsymbol{\rho})$ denotes the global stiffness matrix in the case of minimum compliance and compliant mechanism problems, whereas it signifies the global heat conductivity matrix in the heat conduction problem. The matrix is dependent on the design variable $\boldsymbol{\rho}$. The variable ρ_i indicates the density of the material in the ith element, where $\rho_i = 1$ represents a solid element and $\rho_i = 0$ indicates an empty element.

Lastly, g($\boldsymbol{\rho}$) symbolizes the volume constraint function. Within this function, $V(\boldsymbol{\rho})$ is the volume of material used, $\overline{V}_{\mathrm{f}}$ is the volume fraction, and V_0 is the total volume of the design domain. This formulation effectively encapsulates the constraints and objectives of the topology optimization problems incorporated by the TopADD program.

5. Results and Discussion

This section will conduct numerical tests on the Raspberry Pi cluster to assess its performance. The impact of hardware and software configurations on the cluster's efficiency will be explored to determine the most effective components for the Raspberry Pi cluster. Additionally, the speedup and efficiency gained through parallelization will be evaluated and compared with standard computing systems. Unless specified otherwise, the default test case will be the 2D arbitrary geometry problem in the TopADD program.

5.1. Hardware Configurations' Influences

5.1.1. Cable Influence

Figure 10 compares the computational time differences when utilizing CAT6 versus CAT8 network cables. Interestingly, the disparity in performance is minimal, despite CAT8 cables being reputed to offer triple the speed of CAT6 cables. To evaluate network connection bandwidth, the "iperf" tool is employed. The procedure involves two primary steps. The first step is the installation and server setup: Install the "iperf" program on the first node (e.g., rpi1) and initiate it in server mode. Secondly, on a second node (e.g., rpi2), after installing "iperf", start the program in client mode, which connects to the server node. The syntax is as follows:

```
ubuntu@rpi1:~$ sudo apt install iperf   #installation on rpi1
ubuntu@rpi1:~$ iperf -s                 #start server on rpi1
ubuntu@rpi2:~$ sudo apt install iperf   #installation on rpi2
ubuntu@rpi2:~$ iperf -c rpi1            #start client on rpi2
```

Network bandwidth tests conducted between various nodes reveal that the connection speed approximates 900 Mbits/s. This speed aligns closely with the 1 gigabit switch's maximum capacity, as anticipated. Consequently, it is evident that the cables do not impact the cluster's performance.

However, CAT8 cables have superior shielding and reduced crosstalk [33], which can improve signal integrity and reliability in high-density cluster environments. While the bandwidth advantage of CAT8 is not utilized here, its enhanced durability and interference resistance provide a more robust infrastructure for consistent performance.

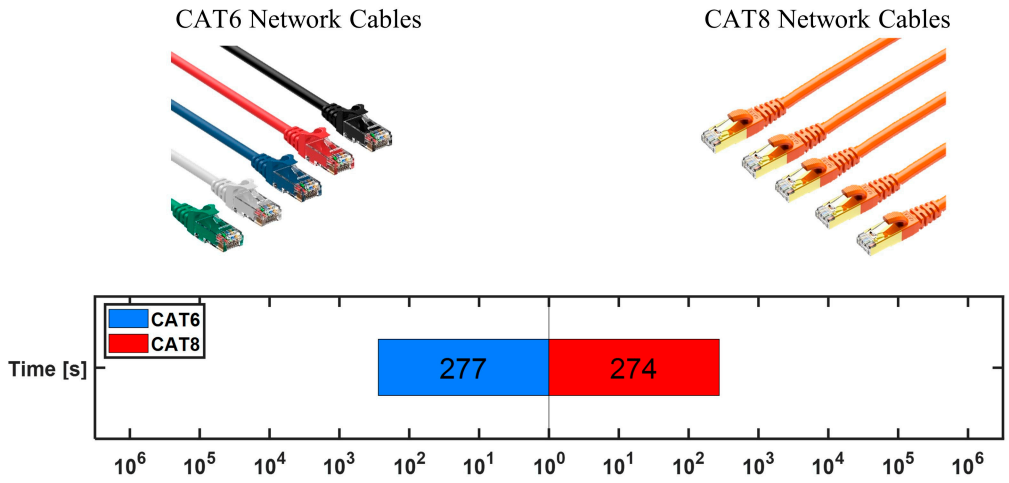


Figure 10. Comparison of computational time using different network cables: CAT6 versus CAT8 network cables.

5.1.2. Network File System Influence

Figure 11 presents a comparative analysis of computational times using different storage solutions, specifically a 128 GB flash drive and a 256 GB NVMe SSD. Despite the NVMe SSD's reported superiority in terms of speed—it is alleged to be over 17 times faster at reading and 38 times faster at writing compared to the flash drive—the differences in computational times are surprisingly marginal. To quantify this, a program named "disk" is employed to offer a user-friendly interface for testing real reading and writing speeds. The results show average reading speeds of 57 MB/s for the flash drive and 333 MB/s for the SSD, with corresponding writing speeds of 48 MB/s and 303 MB/s. While these testing results indicate significant disparities in the reading and writing speeds of the two drives, the final runtime of the TopADD application was nearly identical for both storages. This suggests that the NFS drives are not the bottleneck in the cluster's performance.

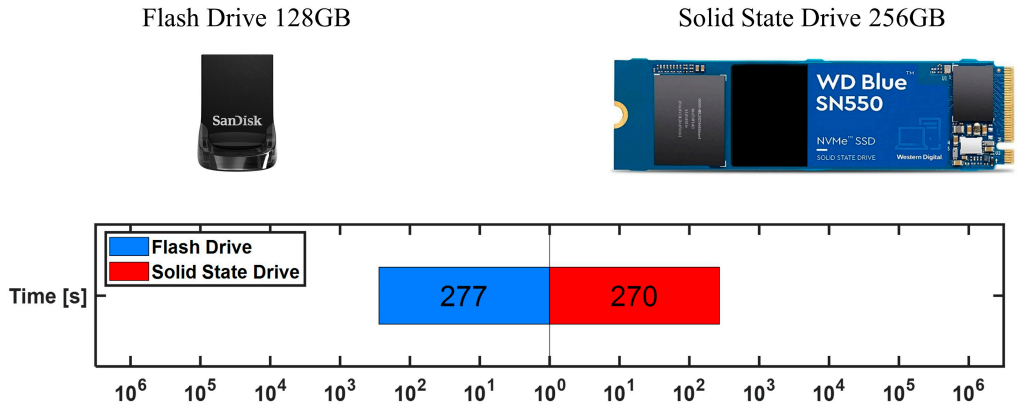


Figure 11. Comparison of computational time using different NFSs: flash drive versus solid state drive network.

5.1.3. Network Switch Influence

Figure 12 compares computational times when using different network switches: a 1-gigabit switch versus a 100-megabit switch. The configuration with the 100-megabit switch shows a 16% slower performance compared to the 1-gigabit one. Network bandwidth was tested using the program "iperf", which has been previously discussed. These tests show that the network bandwidth hovers around 95 Mbits/s, aligning with the maximum capacity of the 100-megabit switch, as expected. However, this comparison reveals an interesting observation: even though the network speed associated with the 100-megabit switch is only a tenth of that achieved with the 1-gigabit switch, the execution time of the TopADD program increases by only 16%. This suggests that although switches do impact the cluster's performance, the effect is less substantial than expected, especially considering the 90% decrease in network speed.

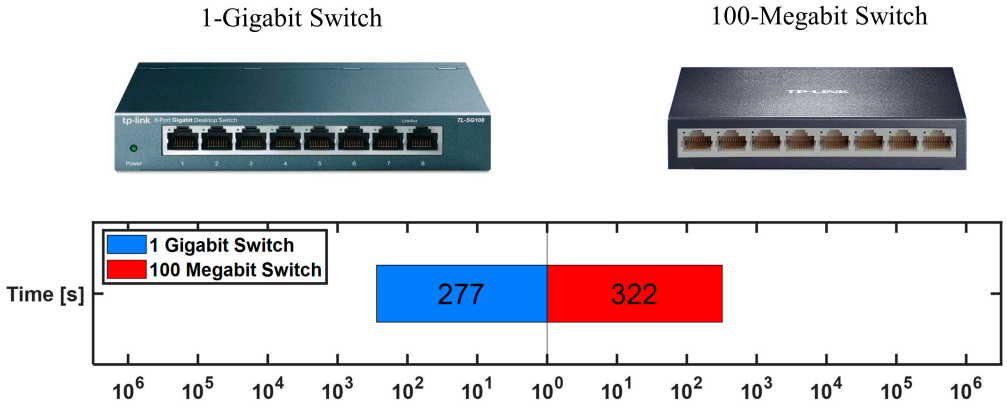


Figure 12. Comparison of computational time using different network switches: a 1-gigabit switch versus a 100-megabit switch.

As noted in the above discussion, peripheral devices, such as network connections and NFSs, have a marginal impact on the cluster's execution time. These devices, including NFSs and network switches, are essentially I/O devices that are connected to the bus of the RPis. However, a more critical component connected to this bus is the RPis' RAM. The RAM's bandwidth plays a significant role in influencing the cluster's performance. To understand this better, the RAM bandwidth has been examined using open-source code form McCalpin [34], available at https://www.cs.virginia.edu/stream/FTP/Code/stream.c (accessed on 1 January 2024). This code can be compiled and executed as follows:

```
ubuntu@rpi1:~$ mpicc -o stream -O3 stream.c  #compiling on rpi1
ubuntu@rpi1:~$ mpirun -n 4 ./stream          #run on rpi1
```

The resulting average memory bandwidth per core is 900 MB/s, with a total bandwidth of 3.6 GB/s. This is notably low, especially when considering the reported total memory bandwidth of 5.5 GB/s and the CPU core frequency of 1.5 GHz [32]. Consequently, it appears that memory bandwidth is a primary limiting factor in the cluster's performance. This topic will receive more in-depth analysis in Section 5.3. Therefore, it is reasonable to infer that investing in high-performance peripheral devices, like advanced network cables or SSD drives, is unnecessary for most users. The identified bottleneck in memory bandwidth suggests that basic versions of these peripherals should suffice.

5.2. Software Configurations' Influences

5.2.1. Compiler Optimization Level Influence

Figure 13 presents a comparative analysis of computational time in relation to the level of optimization in the TopADD program, specifically contrasting the release version against the debug version. In the release version, both the TopADD program and its underlying software, PETSc 3.16.3, were compiled with an optimization level of O3. On the contrary, the debug version is compiled with no level of optimization. The results demonstrate a marked difference: the debug version required over six times the execution time of the release version. This significant disparity underscores the importance of flexibility in configuring a cluster, such as the ability to reinstall optimized versions of software. Such flexibility is crucial for enhancing the performance of parallel computing.

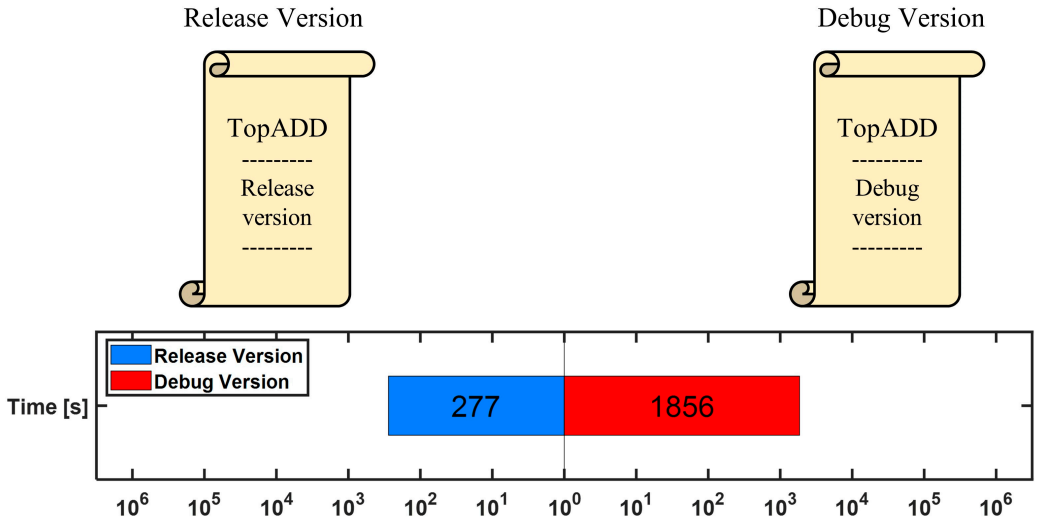


Figure 13. Comparison of computational time using different software versions: the release version versus the debug version.

In addition, the single and double float performance has been compared using the code presented in Appendix D. Single float (12.9455 s) outperforms double float (19.4908 s) in terms of execution time. This is due to the smaller memory footprint and faster memory access associated with float, which allows for better cache utilization and reduced computational overhead. On the other hand, while double float provides higher precision, it incurs additional memory and computation costs, leading to slower performance for large data sizes.

5.2.2. Software Installation Location Influence

Figure 14 illustrates a comparison of computational time when installing the PETSc software 3.16.3 in different locations: on each compute node versus on an NFS. Notably, the execution times are identical in both cases, demonstrating that the installation location does not influence the cluster's performance. This outcome is surprising, given that the NFS-installed version requires loading PETSc via the network each time, and network speeds are typically slower than flash drive speeds. This could be attributed to PETSc being loaded from the network only once, at the beginning of execution, minimizing the impact of loading speed. This demonstrates that conducting practical experiments on an in-house cluster is instrumental in gaining a more comprehensive understanding of parallel computing mechanisms.

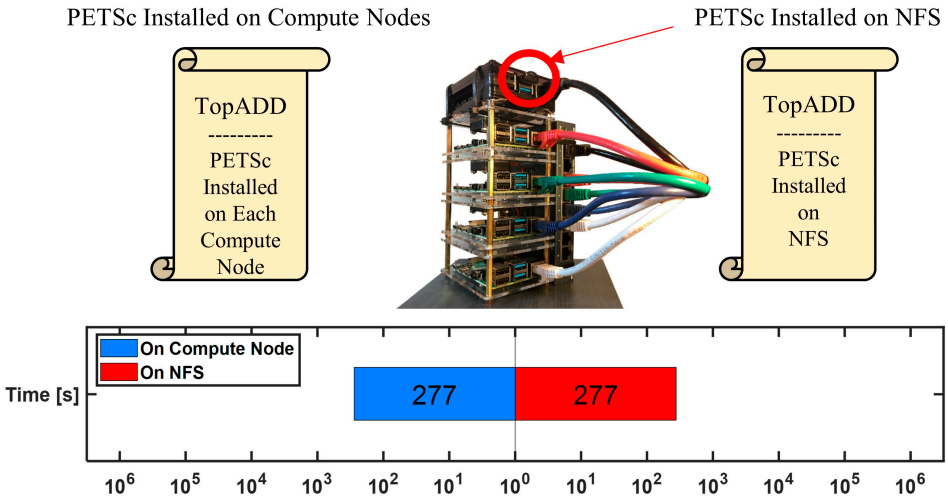


Figure 14. Comparison of computational time using different PETSc installation locations: on each compute node versus on NFS.

5.2.3. Overclocking Level Influence

Figure 15 depicts a comparison of computation times between scenarios with and without overclocking settings. Overclocking refers to the process of increasing the clock speed of a computer's CPU beyond the manufacturer's specified rate. Overclocking can be implemented using the specified syntax, though it is important to note that this is carried out at the user's own risk.

```
sudo nano /boot/firmware/config.txt
# add the following at the end of config.txt
over_voltage=6
arm_freq=2000
```

The results indicate that the difference in computational times due to overclocking is negligible. This aligns with the findings in Section 5.1 which stated that RAM bandwidth has been identified as the primary bottleneck. As the CPU frequency increases due to overclocking, the limiting effect of RAM bandwidth becomes more pronounced. Despite a 25% increase in CPU frequency, the execution time only shows a marginal improvement of 3%. This underscores the minimal impact of overclocking under the current RAM bandwidth constraints.

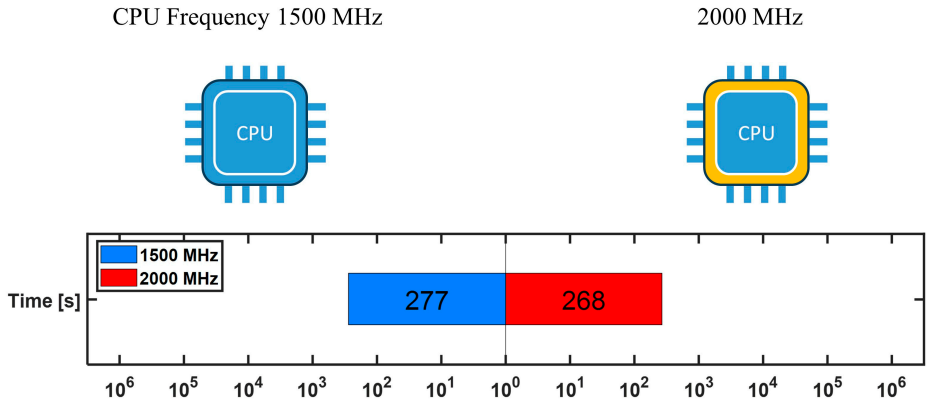


Figure 15. Comparison of computational time using a different CPU frequency: 1500 MHz versus 2000 MHz.

5.3. Parallelization Speedup and Efficiency

The investigation of parallelization speedup and efficiency is conducted. The computational time from a single core was used as the reference. The speedup factor, denoted as S_N, is calculated based on the ratio of the computational time for one core (T_1) to the computational time using N cores (T_N). This is represented by Equation (2):

$$S_N = \frac{T_1}{T_N} \tag{2}$$

Furthermore, the efficiency of parallelization, named as E_p, is a measure of the proportion of time during which a processor is effectively utilized. It is defined as follows in Equation (3):

$$E_P = \frac{S_N}{N} \tag{3}$$

To evaluate these metrics, two benchmarks have been utilized. The first benchmark involves an implementation of parallel computing of pi in mathematics.

It should be mentioned that the core allocation rule in the cluster evenly distributes cores across nodes. For example, when using 4 cores, each node contributes 1 core; for 8 cores, each node contributes 2 cores, and so on. This ensures balanced utilization of resources across all nodes.

5.3.1. Pi Calculation via Integral Method

In the first case study, a method of calculating Pi using the integral method was explored. The integral is represented by

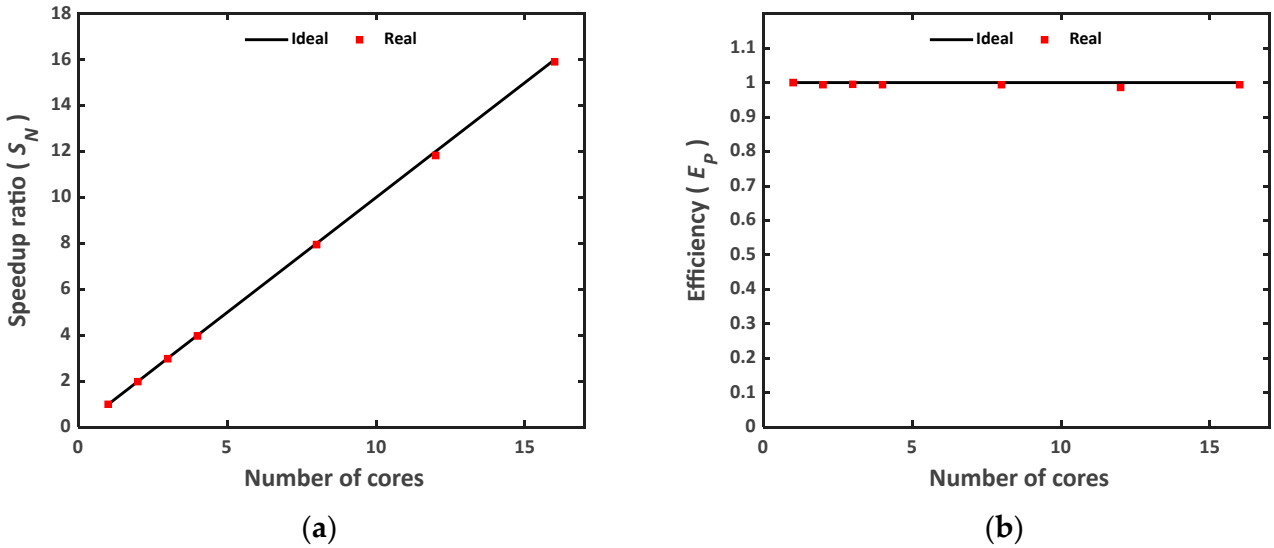
$$\pi = 4\int_0^1 \frac{1}{1+x^2}dx \tag{4}$$

where x varies from 0 to 1, incrementing by an infinitesimally small value dx. The relevant C++ code for this computation is provided in Appendix C.

Table 3 displays the speedup and efficiency results obtained by parallelizing this Pi calculation. Figure 16a,b graphically illustrate these results, showing both the speedup and efficiency achieved through parallelization. The performance of RPi cluster in this context is noteworthy. The observed speedup closely aligns with the theoretical ideal for various core counts. Similarly, the efficiency maintains a high level, reaching up to 0.99 in different scenarios involving varying numbers of cores. This demonstrates the RPi cluster's robust capability in handling parallel computations effectively.

Table 3. Speedup and efficiency of the pi code parallelization.

Core Number	1	2	3	4	8	12	16
Wall clock time (s)	9.70	4.88	3.25	2.44	1.22	0.82	0.61
Speedup (S_N)	1	1.99	2.98	3.98	7.95	11.83	15.90
Efficiency (E_P)	1	0.99	0.99	0.99	0.99	0.99	0.99



(**a**) (**b**)

Figure 16. Visualization of code parallelization for the pi calculation: (**a**) speedup and (**b**) efficiency.

5.3.2. TopADD Default 2D and 3D Models

In another analysis, the performance of TopADD executed with varying numbers of cores is examined. Figure 17 illustrates the initial arbitrary design domain and the corresponding optimized results in TopADD, showing both the default 2D and 3D models. The 2D model features a 240 × 120 mesh, while the 3D model is constructed with a 64 × 32 × 32 mesh. Table 4 provides detailed information about the speedup and efficiency observed when running TopADD on the cluster. Furthermore, Figure 18a,b visually represent these metrics, displaying the speedup and efficiency achieved through parallelization. The data reveal that employing more cores enhances the speedup ratio up to a certain point, beyond which additional cores only marginally reduce the computation time. This trend aligns with findings reported in the literature [29].

Table 4. Speedup and efficiency of the TopADD parallelization on the RPi cluster.

Core Number	1	2	3	4	8	12	16
		2D					
Wall clock time (s)	386	283	219	198	207	221	277
Speedup (S_N)	1	1.36	1.76	1.95	1.86	1.75	1.39
Efficiency (E_P)	1	0.68	0.59	0.49	0.23	0.15	0.09
		3D					
Wall clock time (s)	6593	3844	2662	2119	1948	1808	1853
Speedup (S_N)	1	1.72	2.48	3.11	3.38	3.65	3.56
Efficiency (E_P)	1	0.86	0.83	0.78	0.42	0.30	0.22

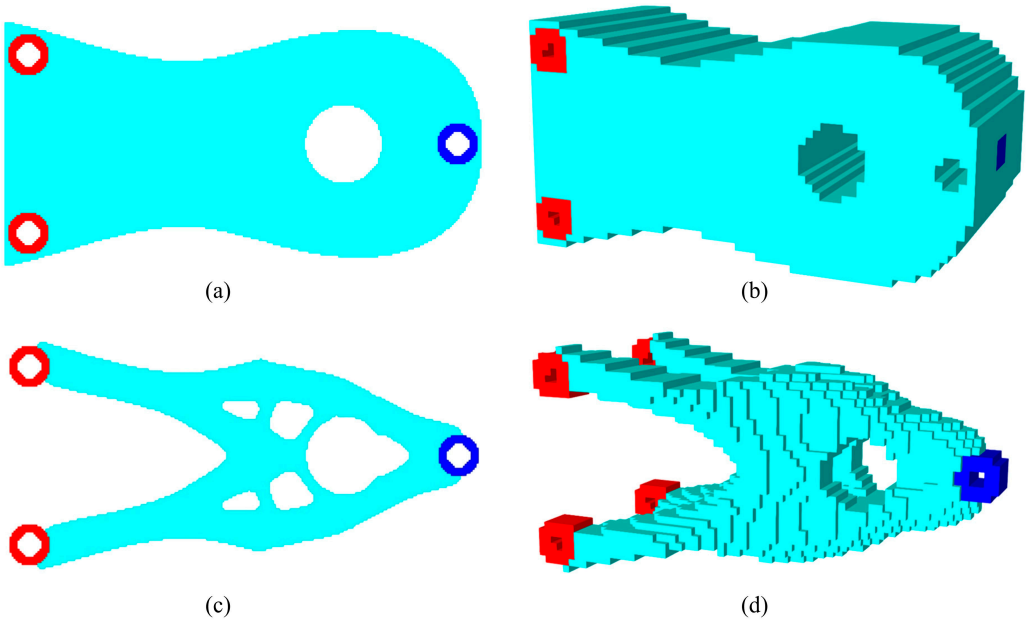


Figure 17. Default domain initialization and optimized results in TopADD, (**a**) 2D design domain with a mesh of 240 × 120, (**b**) 3D design domain with a mesh of 64 × 32 × 32, (**c**) 2D-optimized result, and (**d**) 3D-optimized result. The red color areas are the fixed locations, and the blue areas are the loading positions.

The observed results highlight superior speedup and efficiency in the 3D model compared to the 2D model. This variance is likely a consequence of the increased network communication involved in the 2D scenario, which adds substantial overhead time. Conversely, the 3D model necessitates a larger allocation of CPU time for computation. This shift in resource utilization diminishes the proportion of time spent on network communication, thereby reducing overhead and enhancing overall efficiency and speedup in the parallel computing environment.

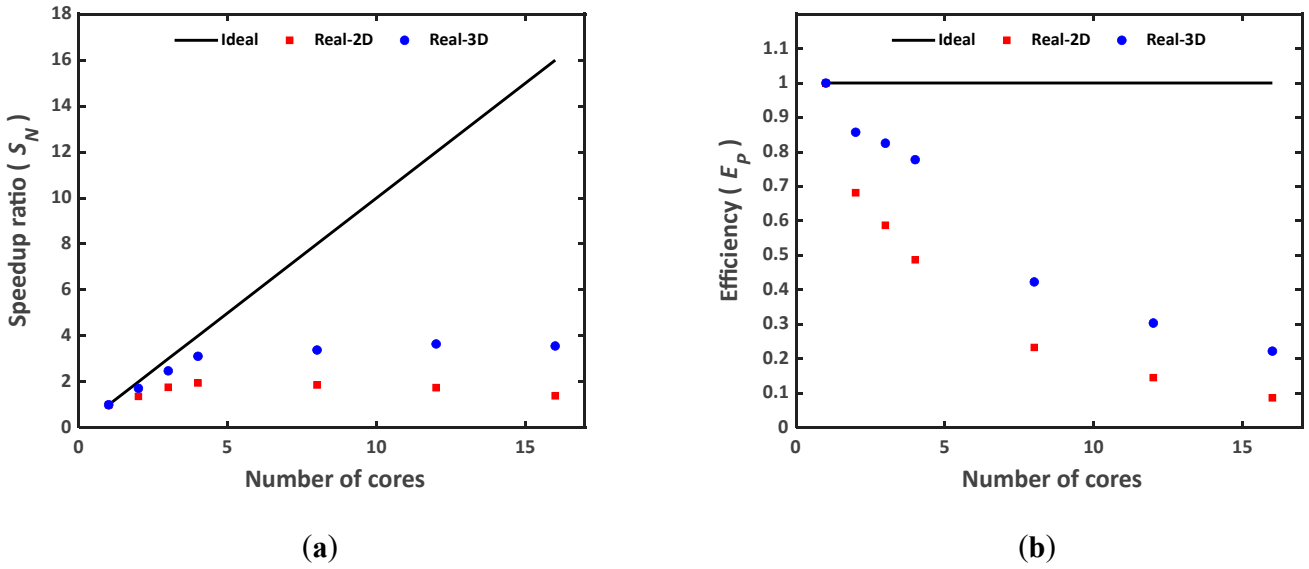


Figure 18. The visualization of the code parallelization for the TopADD default 2D and 3D models: (**a**) speedup and (**b**) efficiency.

It is important to note that performance begins to decline significantly when more than four cores are utilized. This trend may be attributable to the RAM bandwidth limitations of the RPis, as discussed earlier. Consequently, the RAM bandwidth testing code by [34] remains relevant and will be employed again, this time across the entire cluster. The setup involves creating a machine file containing the names of the nodes. The test is then compiled and executed using a varying number of cores, ranging from 1 to 16, as demonstrated in the following example.

```
ubuntu@rpi0:~$ mpicc -o stream -O3 stream.c          #compile on rpi0
ubuntu@rpi0:~$ mpirun -n 16 -f machinefile ./stream #run on rpi1
```

Table 5 displays the total memory bandwidth results. Notably, a decline in bandwidth is observed at 8, 12, and 16 cores. When up to 4 cores are used, each core fully utilizes the bandwidth available on a Raspberry Pi node. However, adding more cores per node leads to a reduction in overall performance. This suggests that the Raspberry Pi 4 Model B is highly efficient at saturating the bus when a single core is used. The addition of more cores does not enhance performance; in fact, it results in a decrease due to increased idle cycles from RAM. This is caused by contention leading to more frequent opening and closing of memory pages.

Table 5. RAM bandwidth with different number of cores used for the RPi cluster.

Core Number	1	2	3	4	8	12	16
Total RAM bandwidth (GB/s)	5.46	10.92	15.82	21.81	26.90	14.68	14.36
Bandwidth per core (GB/s)	5.46	5.46	5.27	5.45	3.36	1.22	0.90

Since parallelization efficiency tends to decrease with an increasing number of cores, a further analysis of network latency, MPI communication overhead, and workload balancing strategies is necessary.

Firstly, the network speed and latency were tested using the method described in Section 5.1.1. The results show a maximum network speed of approximately 900 Mbits/sec, which is close to the theoretical limit of the 1 Gigbits Ethernet on the Raspberry Pi 4 model B. Additionally, the transmission latency between two nodes is around 0.2 ms when the "ping" software is used.

Secondly, the MPI communication overhead was evaluated using the OSU benchmark [35]. The measured latency for transferring 64-byte message is 0.056 ms, indicating good MPI communication performance.

Lastly, the TopADD [15] utilizes the DMDA function in PETSc [21] to generate structured grids, which are evenly distributed across all cores. As a result, factors such as network latency, MPI communication overhead, and workload balancing strategies are unlikely to significantly impact the parallelization efficiency.

From Sections 5.3.1 and 5.3.2, it can be observed that the parallel-computing performance of Raspberry Pi can indeed vary depending on the nature of the application being tested. CPU-bound tasks (e.g., Pi calculation) scale well, while memory-intensive tasks (e.g., TopADD) degrade when more cores are added due to shared memory bus contention, as shown in Table 5. This highlights the value of affordable RPi clusters for testing diverse workloads.

The scalability test on the 4-node Raspberry Pi cluster (4 cores per node) shows parallel efficiency drops beyond 4 cores due to shared-memory bandwidth saturation. However, this is unnecessarily negative, as single-core saturation maximizes performance for non-multithreaded apps, which is necessary for Raspberry as a general-purpose computing device. In the future, scalability can be improved by adding more nodes with one core each, and further multi-node tests will be necessary.

5.4. Comparison to Typical Computers

To benchmark the RPi cluster's performance against the standard computing hardware typically accessible to newcomers or researchers, a laptop and a desktop computer were selected as test systems. The specifications of these devices are detailed in Table 6. Both the laptop and desktop were set up with the same OS to ensure consistency in the testing environment. The following sections will provide a concise evaluation of their performance and energy consumption in comparison to the RPi cluster. This comparison aims to offer a clearer perspective of the relative capabilities and efficiency of the RPi cluster in common computational tasks.

Table 6. Test computers' specifications.

Platform	Cost (CAD)	OS	CPU	Cores	MHz
RPi Cluster	730	Ubuntu 20.04	4 × BCM2711 ARM Cortex-A72	16	1500
Laptop	1500	Ubuntu 20.04	Core i5-8350U	4	1900
Desktop	2000	Ubuntu 20.04	Core i7-11700	8	2500

Figure 19 provides a comparative analysis of the computing and energy performance among three different types of computers: an RPi cluster, a Core i5 laptop (4 cores), and a Core i7 desktop (8 cores). The comparison reveals distinct differences in their computing times and energy consumption. In terms of computing time, as depicted in Figure 19a, the RPi cluster had the longest recorded duration for task completion, whereas the Core i7 desktop demonstrated the shortest computing time. However, when it comes to power consumption, the scenario shifts. As shown in Figure 19b, the Core i7 desktop consumes the most power when operating at full capacity. In contrast, the RPi cluster exhibits remarkably low power consumption. When in an idle state, which is typical during program development and debugging, the RPi cluster can operate with just the login node active, significantly reducing its power usage to merely 6 W. However, the Core i7 desktop consumes approximately 12 times more power than the RPi cluster, as presented in Figure 19c.

The annual energy consumption calculation considers both full-power and idle power states. For the RPi cluster, the calculation assumes an equal split between these two states. For the laptop and desktop, their full-power duration is based on their respective computing time ratios compared to the RPi cluster. For instance, the desktop's full-power time is calculated as $0.5 \times 365 \times (19/198)$, resulting in a total consumption of 763 kWh of electricity annually. Figure 19d summarizes these energy calculations, clearly showing that the RPi cluster is the most energy-efficient option, incurring the lowest annual energy costs among the three.

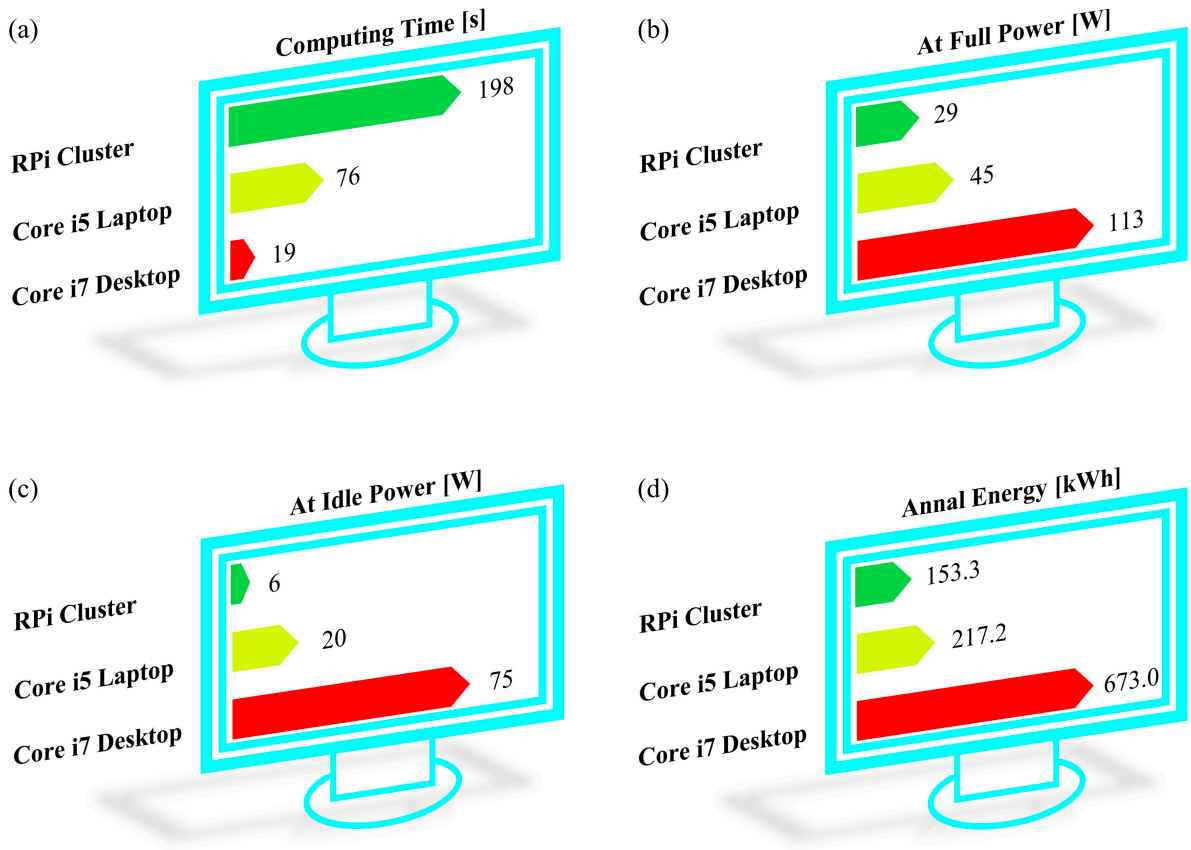


Figure 19. Comparison of computing and energy performance among the three different computers: (**a**) computing time, (**b**) full power consumption, (**c**) idle power consumption, and (**d**) annual energy.

It should be noted that the RPi cluster could be compared to other low-cost clusters in the future. However, comprehensive comparisons between Raspberry Pis, ODROID-XU4, Jetson Nano and other low-cost single-board computers (SBCs) have already been presented in the literature [36], as depicted in Table 7. Table 7 shows the RPi cluster stands out in terms of cost-effectiveness and energy efficiency, although the RPi's shared memory architecture limits its scalability for memory-heavy tasks compared to other SBCs. Furthermore, the Raspberry Pi is one of the best-selling SBCs worldwide, with cumulative sales exceeding 60 million units as of 2024, which demonstrates user satisfaction [37]. Its vibrant ecosystem, including active forums (e.g., raspberrypi.com/forums accessed on 1 January 2024), rich documentation, and a large community, ensures timely problem resolution, making it a trusted platform for both hobbyists and professionals. Finally, although this work uses a Raspberry Pi cluster as the experimental platform, the performance analysis and optimization techniques (e.g., memory bandwidth testing and parallel efficiency evaluation) are universal and can be extended to other SBCs (e.g., ODROID, Jetson Nano, etc.).

Table 7. Low-cost single board computers' comparisons.

Computer	Cost per Node	Power Draw (W/Node)	Peak RAM Bandwidth (GB/s)	MHz
RPi 4	92 CAD	2–3	4.3	1500
ODROID-XU4	197 CAD	4–6	6.1	1900
Jetson Nano	260 CAD	5–10	12.8	2000

5.5. Environmental Sustainability

The year 2023 has been identified as the hottest on record, according to [38], highlighting the growing urgency of sustainability and carbon-neutral initiatives. With an increasing global focus on reducing carbon footprints, electricity savings have become a critical aspect of sustainable practices. As noted by [39], each kilowatt-hour (kWh) of electricity is equivalent to approximately 0.3 kg of CO_2 emissions.

Given this relationship, the annual carbon footprints of three different types of computers have been calculated and are presented in Figure 20. The results reveal a stark contrast in environmental impact: the RPi cluster requires the carbon absorption capacity of approximately 2 mature trees to offset its emissions, compared to around 9 trees needed for a Core i7 desktop. For context, a mature tree can absorb about 22 kg of CO_2 annually, as noted by the United States Department of Agriculture [39].

The significant difference underscores the sustainability of the RPi cluster, whose carbon footprint is over 70% lower compared to that of the desktop. Its lower power consumption and consequent reduced CO2 emissions make it an attractive choice for those seeking to minimize the environmental impact in the realm of computing.

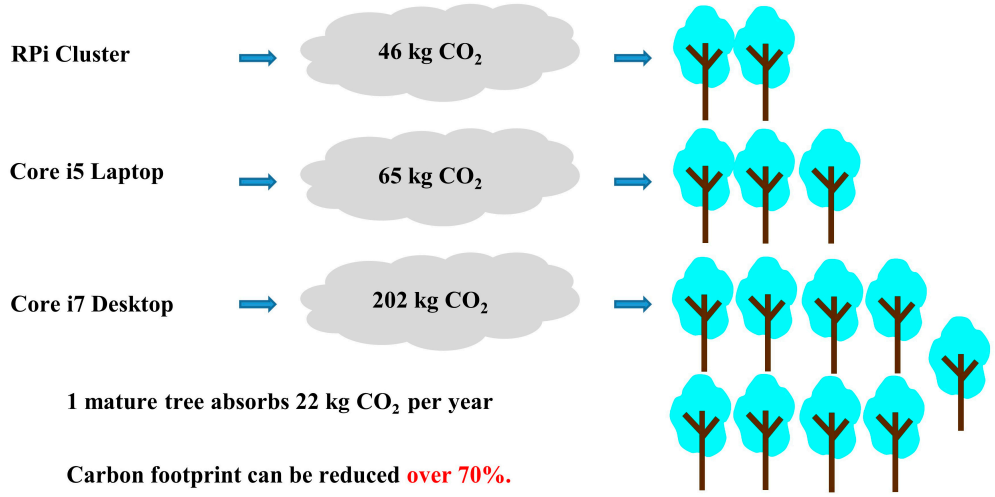


Figure 20. Comparison of the annual carbon footprint among the three different computers.

5.6. Maximum Capability

The cluster's proficiency in high-resolution 2D topology optimization was rigorously evaluated through three physical TO problems, which are outlined in Section 4. The outcomes, as shown in Figure 21, were executed in 2D to optimize efficiency, achieving enhanced resolution while maintaining manageable computational costs. The first scenario involved a compliance minimization problem, utilizing a 2048 $\times$ 1024 mesh that comprised 2 million elements. For the second scenario, the compliant mechanism problem, a smaller 1280 $\times$ 640 mesh was used, resulting in 0.8 million elements. This reduction in mesh size was necessary, as a larger mesh exceeded the cluster's memory capacity. The potential crashes are because of the increased memory requirements of the self-adjoint problem calculation in the compliant mechanism problem. The third scenario addressed a heat conduction problem with a 1648 $\times$ 2048 mesh, accounting for 3.4 million elements. The computational durations for these problems were recorded at 11.9 h, 1.5 h, and 3.6 h, respectively. Notably, the results from these 2D optimizations can be transformed into 3D geometries through extrusion, as shown in Figure 21b. All the procedures were performed exclusively on the login node of the cluster. Furthermore, these optimized designs are suitable for 3D printing, such as Fused Deposition Modeling (FDM), as demonstrated in Figure 21c.

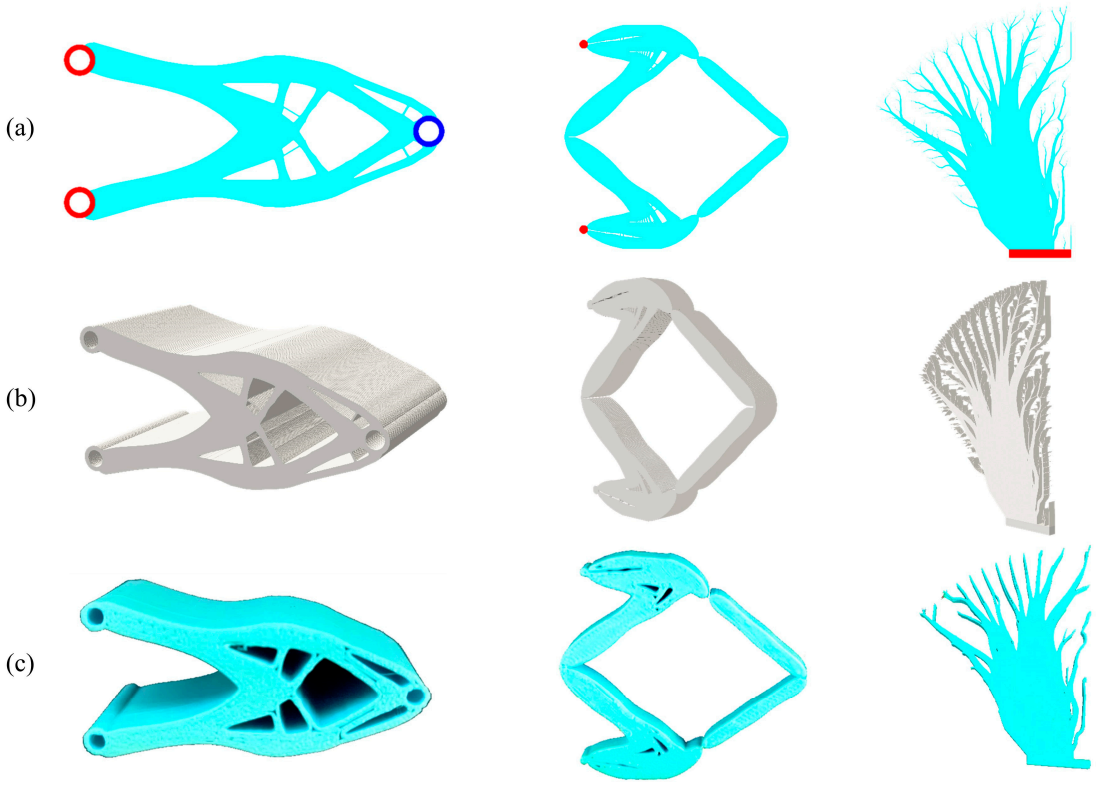


Figure 21. RPi cluster's maximum capacity for the three physical problems in TopADD, from left to right: compliance minimization, compliant mechanism, and heat conduction: (**a**) 2D TO results, (**b**) extruded 3D geometries, and (**c**) parts which were 3D printed via FDM.

6. Conclusions

A Raspberry Pi-based cluster system, TopADDPi, is proposed for the field of PCTO with an arbitrary design domain. TopADDPi addresses the need for cost-effective, accessible distributed-memory computational resources in PCTO. As an affordable, sustainable, and indispensable tool, it offers new opportunities for newcomers and researchers, particularly those with limited access to conventional high-performance computing resources. The main conclusions are as follows:

1. TopADDPi establishes a model in distributed computing and educational resources, presenting a holistic framework that enhances programming and learning capabilities in PCTO.
2. The detailed analysis of hardware and software setups in the study reveals that peripheral devices like internet cables and Network File Systems (NFSs) exert minimal influence on performance. The primary constraint, however, lies in the RAM bandwidth of the Raspberry Pis, which is identified as a critical bottleneck.
3. In tests, the cluster achieves remarkable computational efficiency, approaching 99% parallel efficiency for moderate workloads such as the Pi calculation. For more demanding tasks, the parallel efficiency markedly drops below 10%. To optimize performance and maintain parallel efficiency above 40%, the study recommends using only one core per node, a strategy to mitigate the limitations imposed by RAM bandwidth saturation.
4. When compared with traditional computing systems, TopADDPi emerges as both a cost-effective and environmentally sustainable solution. The study highlights that has a significantly reduced carbon footprint, offering a reduction of over 70% annually compared to standard desktop computers.

5. The proposed method achieves high-resolution topology optimization design results while balancing computational costs and memory constraints. All the procedures were performed exclusively on the cluster. The design results can be processed and manufactured by 3D printing, enabling the exploration of sustainable and digitalized manufacturing processes.

Author Contributions: Conceptualization, Z.-D.Z. and O.I.; methodology, Z.-D.Z.; software, Z.-D.Z.; validation, Z.-D.Z. and D.-Y.Y.; formal analysis, Z.-D.Z. and D.-Y.Y.; investigation, Z.-D.Z. and O.I.; resources, L.M., T.G., J.-H.Z. and W.-H.Z.; data curation, Z.-D.Z.; writing—original draft preparation, Z.-D.Z.; writing—review and editing, all authors; visualization, Z.-D.Z.; supervision, W.-H.Z.; project administration, T.G. and J.-H.Z.; funding acquisition, Z.-D.Z., T.G., J.-H.Z. and W.-H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (12202359, 12032018), the National Key R&D Program of China (2022YFB3402200, 2022YFB4603101) and the Key Project of NSFC (92271205).

Data Availability Statement: The configuration information of Raspberry Pi cluster and further details have been provided at: https://github.com/wonderfulzzd/Pi_Cluster_for_TopADD (accessed on 1 January 2024).

Acknowledgments: The authors would like to thank all the specialists from SHARCNET for providing valuable tutorials and assistance.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

Appendix A. Code Needs a Raspberry Pi to Debug

```
/*
Authors: Zhidong Brian Zhang in Dec. 2023
Copyright (C) 2023-2023

Disclaimer:
The authors reserve all rights but do not guaranty that the code is
free from errors. Futhermore, we shall not be liable in any event
caused by the use of the program.

Usage:
 1. g++ main.cc
 2. ./a.out
*/

#include <iostream>
#include <fstream>

int main () {

  float val = 1.0;
  std::ofstream fs("example.bin", std::ios::out | std::ios::binary);
  fs.write(reinterpret_cast<const char*>(&val), sizeof(float));
  fs.close();

  float buf;
  std::ifstream in ("example.bin", std::ios::binary); // reload the
file stream as binary
  in.read (reinterpret_cast<char*> (&buf), sizeof(float));
  in.close();

  unsigned int a = 0;
  float b = buf;

  a = a - b;
  std::cout << "a - b: " << a << std::endl;

  return 0;
}
```

Appendix B. Code Needs a Cluster to Debug

```
/*
Authors: Zhidong Brian Zhang in Dec. 2023
Copyright (C) 2023-2023

Disclaimer:
The authors reserve all rights but do not guaranty that the code is
free from errors. Futhermore, we shall not be liable in any event
caused by the use of the program.

Usage:
 1. mpic++ main.cc
 2. Create machinefile containing computer name(s)
 3. mpiexec -n 4 --hostfile machinefile ./a.out
*/

#include <mpi.h>
#include <iostream>

int main (int argc, char *argv[]) {

  // Initialize MPI
  MPI_Init (&argc, &argv);

  // Sum up a
  int rank;
  int rankSize;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &rankSize);
  int a = rank;

  int *shared;
  MPI_Win win;
  MPI_Win_allocate_shared(1*sizeof(int), sizeof(int), MPI_INFO_NULL,
                          MPI_COMM_WORLD, &shared, win);
  shared[0] = a;
  a = 0;
  MPI_Barrier(MPI_COMM_WORLD);

  MPI_Aint queried_size;
  int disp_unit;
  MPI_Win_shared_query(win, 0, &queried_size, &disp_unit, &shared);

  for (int i = 0; i < rankSize; ++i)
  {
    a += shared[i];
  }

  std::cout << "sum(a): " << a << std::endl;

  MPI_Win_free(&win);
  MPI_Finalize();

  return 0;
}
```

Appendix C. Code for Pi Calculation Using Integral Method

```
/*
Authors: Zhidong Brian Zhang in Dec. 2023
Copyright (C) 2023-2023

Disclaimer:
The authors reserve all rights but do not guaranty that the code is
free from errors. Futhermore, we shall not be liable in any event
caused by the use of the program.

Usage:
 1. mpic++ main.cc
 2. Create machinefile containing computer name(s)
 3. mpiexec -n 4 --hostfile machinefile ./a.out
*/

#include "mpi.h"

int main( int argc, char *argv[]) {
    int rank, np, i, nLoc, nTot = 1024000000 ;
    double mypi, pi, detx, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&np);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    double t1, t2;
    // start timer
    t1 = MPI_Wtime ();

    if (rank == 0) { nLoc= nTot / np; }

    MPI_Bcast(&nLoc, 1, MPI_INT, 0, MPI_COMM_WORLD);

    detx = 1.0 / (double) nTot;
    sum = 0.0;
    for (i = rank + 1; i <= nTot; i += np) {
      x = detx * ((double)i -0.5);
      sum += (4.0 / (1.0 + x*x));
    }
    mypi = detx * sum;

    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    t2 = MPI_Wtime ();
    if (rank == 0) { printf("pi: %1.15f, time: %f\n", pi, t2-t1);}

    MPI_Finalize();
    return 0;
}
```

Appendix D. Code for Single and Double Float Comparison

```
/*
Authors: Zhidong Brian Zhang in Feb. 2025
Copyright (C) 2023-2025

Disclaimer:
The authors reserve all rights but do not guaranty that the code is
free from errors. Futhermore, we shall not be liable in any event
caused by the use of the program.

Usage:
 1. mpic++ floatVSdouble floatVSdouble.cc
 2. Create machinefile containing computer name(s)
 3. mpiexec -n 4 --hostfile machinefile ./floatVSdouble
*/
#include <iostream>
#include <vector>
#include <mpi.h>
#include <cstring>
#include <ctime>
using namespace std;

template <typename T>
void compute_sum(vector<T>& array, int chunk_size, MPI_Datatype
mpi_type, int rank, int size, const string& precision) {
    vector<T> locArray(chunk_size);
    MPI_Scatter(array.data(), chunk_size, mpi_type, locArray.data(),
chunk_size, mpi_type, 0, MPI_COMM_WORLD);
    double locSum = 0, totSum = 0;
    for (int i = 0; i < chunk_size; ++i) locSum += static_cast<dou-
ble>(locArray[i]);
    MPI_Reduce(&locSum,&totSum,1,MPI_DOUBLE, MPI_SUM,0,MPI_COMM_WORLD);
    if (rank == 0) cout << "TotSum " <<precision<<":"<<totSum << endl;
}

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int N = 200000000, CHUNK_SIZE = N / size;
    double t1start = MPI_Wtime(); vector<float> aF(N);
    if (rank==0) for(int i=0;i<N;++i) aF[i]=static_cast<float>(i+1);
    compute_sum(aF, N/size, MPI_FLOAT, rank, size, "float");
    double t1 = MPI_Wtime() - t1start;
    double t2start = MPI_Wtime(); vector<double> aD(N);
    if (rank==0) for(int i=0;i<N;++i) aD[i]=static_cast<double>(i+1);
    compute_sum(aD, N/size, MPI_DOUBLE, rank, size, "double");
    double t2 = MPI_Wtime() - t2start;
    if (rank == 0) {cout <<"T1: "<<t1<< " s\n"<<"T2: "<<t2<<" s\n";}
MPI_Finalize();
return 0;
}
```

References

1. Zhu, J.; Zhou, H.; Wang, C.; Zhou, L.; Yuan, S.; Zhang, W. A review of topology optimization for additive manufacturing: Status and challenges. *Chin. J. Aeronaut.* **2021**, *34*, 91–110. [CrossRef]
2. Meng, L.; Zhang, W.; Quan, D.; Shi, G.; Tang, L.; Hou, Y.; Breitkopf, P.; Zhu, J.; Gao, T. From Topology Optimization Design to Additive Manufacturing: Today's Success and Tomorrow's Roadmap. *Arch. Comput. Methods Eng.* **2020**, *27*, 805–830. [CrossRef]

3. Plocher, J.; Panesar, A. Review on design and structural optimisation in additive manufacturing: Towards next-generation lightweight structures. *Mater. Des.* **2019**, *183*, 108164. [CrossRef]
4. Ibhadode, O.; Zhang, Z.; Sixt, J.; Nsiempba, K.M.; Orakwe, J.; Martinez-Marchese, A.; Ero, O.; Shahabad, S.I.; Bonakdar, A.; Toyserkani, E. Topology optimization for metal additive manufacturing: Current trends, challenges, and future outlook. *Virtual Phys. Prototyp.* **2023**, *18*, e2181192. [CrossRef]
5. Bendsøe, M.P.; Sigmund, O. *Topology Optimization: Theory, Methods and Applications*; Springer Science & Business Media: New York, NY, USA, 2003.
6. Eslamipoor, R. Direct and indirect emissions: A bi-objective model for hybrid vehicle routing problem. *J. Bus. Econ.* **2024**, *94*, 413–436. [CrossRef]
7. Eslamipoor, R. An optimization model for green supply chain by regarding emission tax rate in incongruous vehicles. *Model. Earth Syst. Environ.* **2023**, *9*, 227–238. [CrossRef]
8. Aage, N.; Andreassen, E.; Lazarov, B.S.; Sigmund, O. Giga-voxel computational morphogenesis for structural design. *Nature* **2017**, *550*, 84–86. [CrossRef] [PubMed]
9. Aage, N.; Andreassen, E.; Lazarov, B.S. Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. *Struct. Multidiscip. Optim.* **2015**, *51*, 565–572. [CrossRef]
10. Mukherjee, S.; Lu, D.; Raghavan, B.; Breitkopf, P.; Dutta, S.; Xiao, M.; Zhang, W. Accelerating Large-scale Topology Optimization: State-of-the-Art and Challenges. In *Archives of Computational Methods in Engineering*; Springer Nature: Berlin, Germany, 2021; pp. 4549–4571. [CrossRef]
11. Borrvall, T.; Petersson, J. Large-scale topology optimization in 3D using parallel computing. *Comput. Methods Appl. Mech. Eng.* **2001**, *190*, 6201–6229. [CrossRef]
12. Mezzadri, F.; Bouriakov, V.; Qian, X. Topology optimization of self-supporting support structures for additive manufacturing. *Addit. Manuf.* **2018**, *21*, 666–682. [CrossRef]
13. Zhang, Z.D.; Ibhadode, O.; Ali, U.; Dibia, C.F.; Rahnama, P.; Bonakdar, A.; Toyserkani, E. Topology optimization parallel-computing framework based on the inherent strain method for support structure design in laser powder-bed fusion additive manufacturing. *Int. J. Mech. Mater. Des.* **2020**, *16*, 897–923. [CrossRef]
14. Fernández, E.; Yang, K.; Koppen, S.; Alarcón, P.; Bauduin, S.; Duysinx, P. Imposing minimum and maximum member size, minimum cavity size, and minimum separation distance between solid members in topology optimization. *Comput. Methods Appl. Mech. Eng.* **2020**, *368*, 113157. [CrossRef]
15. Zhang, Z.D.; Ibhadode, O.; Bonakdar, A.; Toyserkani, E. TopADD: A 2D/3D integrated topology optimization parallel-computing framework for arbitrary design domains. *Struct. Multidiscip. Optim.* **2021**, *64*, 1701–1723. [CrossRef]
16. Johnston, S.J.; Cox, S.J. The raspberry Pi: A technology disrupter, and the enabler of dreams. *Electronics* **2017**, *6*, 51. [CrossRef]
17. Cox, S. Steps to make Raspberry Pi Supercomputer. *Univ. Southampt.* **2013**. Available online: https://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton_web.pdf (accessed on 1 January 2024).
18. Kiepert, J. Creating a raspberry pi-based beowulf cluster. *Boise State* **2013**. Available online: https://ogervasi.unipg.it/OpSysNet/4Students/Clusters/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf (accessed on 1 January 2024).
19. Tso, F.P.; White, D.R.; Jouet, S.; Singer, J.; Pezaros, D.P. The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures. In Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, Philadelphia, PA, USA, 8–11 July 2013; pp. 108–112.
20. Abrahamsson, P.; Helmer, S.; Phaphoom, N.; Nicolodi, L.; Preda, N.; Miori, L.; Angriman, M.; Rikkilä, J.; Wang, X.; Hamily, K.; et al. Affordable and energy-efficient cloud computing clusters: The Bolzano Raspberry Pi cloud cluster experiment. In Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, UK, 2–5 December 2013. [CrossRef]
21. Balay, S.; Abhyankar, S.; Adams, M.; Brown, J.; Brune, P.; Buschelman, K.; Dalcin, L.; Dener, A.; Eijkhout, V.; Gropp, W.; et al. PETSc Users Manual. 2019. Available online: https://publications.anl.gov/anlpubs/2019/12/155920.pdf (accessed on 1 January 2024).
22. Seal, D. *ARM Architecture Reference Manual*; Pearson Education: London, UK, 2001.
23. Upton, E.; Halfacree, G. *Raspberry Pi® User Guide*; John Wiley & Sons: Hoboken, NJ, USA, 2016. [CrossRef]
24. Stroustrup, B. *The C++ Programming Language*; Pearson Education: London, UK, 2013.
25. Potluri, S.; Wang, H.; Dhanraj, V.; Sur, S.; Panda, D.K. Optimizing MPI one sided communication on multi-core infiniband clusters using shared memory backed windows. In *Recent Advances in the Message Passing Interface, Proceedings of the 18th Eur. MPI Users' Gr. Meet. EuroMPI 2011, Santorini, Greece, 18–21 September 2011*; Springer: Berlin/Heidelberg, Germany; pp. 99–109.
26. Pahl, C.; Helmer, S.; Miori, L.; Sanin, J.; Lee, B. A container-based edge cloud PaaS architecture based on raspberry Pi clusters. In Proceedings of the 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), Vienna, Austria, 22–24 August 2016; pp. 117–124. [CrossRef]
27. Dennis, A.K. *Raspberry Pi Super Cluster*; Packt Publishing Ltd.: Birmingham, UK, 2013.

28. Han, H.; Kim, K.; Jung, S.; Chang, Y. Performance Evaluation on the Parallel Processing System with the Raspberry Pi 4. In Proceedings of the 2022 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 14–16 December 2022; pp. 1372–1374. [CrossRef]
29. Warade, M.; Schneider, J.G.; Lee, K. Measuring the Energy and Performance of Scientific Workflows on Low-Power Clusters. *Electronics* **2022**, *11*, 1801. [CrossRef]
30. Neto, A.J.A.; Neto, J.A.C.; Moreno, E.D. The development of a low-cost big data cluster using Apache Hadoop and Raspberry Pi. A complete guide. *Comput. Electr. Eng.* **2022**, *104*, 108403. [CrossRef]
31. Papakyriakou, D.; Barbounakis, I.S. High Performance Linpack (HPL) Benchmark on Raspberry Pi 4B (8GB) Beowulf Cluster. *Int. J. Comput. Appl.* **2023**, *185*, 11–19. [CrossRef]
32. Raspberry Pi. Raspberry Pi 4 Tech Specs. 2022. Available online: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (accessed on 1 January 2024).
33. Foroughi, T. Evaluation of 10G Ethernet Under Various Environments, Unitec Institute of Technology. 2024. Available online: https://www.researchbank.ac.nz/items/97dab53a-7950-4bc1-8244-0dc3f00db5f6 (accessed on 1 January 2024).
34. McCalpin, J. STREAM: Sustainable Memory Bandwidth in High Performance Computers. 2006. Available online: https://www.cs.virginia.edu/stream/ftp/code/stream.c (accessed on 1 January 2024).
35. OSU Micro-Benchmarks, Ohio State University. 2025. Available online: http://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-7.1.tar.gz (accessed on 1 January 2024).
36. Kaiser, T. sbc-bench. 2025. Available online: https://github.com/ThomasKaiser/sbc-bench.git (accessed on 1 January 2024).
37. Pounder, L. Raspberry Pi Celebrates 12 Years as Sales Break 61 Million Units, Tom's Hardw. 2024. Available online: https://www.tomshardware.com/raspberry-pi/raspberry-pi-celebrates-12-years-as-sales-break-61-million-units (accessed on 1 January 2024).
38. Copernicus Climate Change Service (C3S), Record Warm November Consolidates 2023 as the Warmest Year. 2023. Available online: https://climate.copernicus.eu/record-warm-november-consolidates-2023-warmest-year (accessed on 1 January 2024).
39. Stancil, J.M. The Power of One Tree–The Very Air We Breathe, USDA (Blog). Entry Posted. 2019. Available online: https://www.usda.gov/media/blog/2015/03/17/power-one-tree-very-air-we-breathe (accessed on 17 March 2015).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.