The Journal of Supercomputing (2025) 81:445
https://doi.org/10.1007/s11227-024-06893-1

Singularity to deploy HPC applications: a study case with WRF

Pierre-Simon Callist Yannick Tondreau[1] · Juan C. Perez[1] · Juan P. Díaz[1] · Vicente Blanco Pérez[2] · Jonatan Felipe[3]

Accepted: 23 December 2024
© The Author(s) 2025

Abstract
This study evaluates the performance and portability of the Weather Research and Forecasting model in high-performance computing environments, comparing traditional baremetal deployments with containerized executions using Singularity. Experiments were conducted on the TeideHPC system, focusing on execution time, the impact of different compilers (Intel vs. GCC), and parallelization strategies (MPI and OpenMP). The results indicate that while Singularity introduces a performance overhead of 11 to 15%, it offers significant advantages in portability and reproducibility. Additionally, the study highlights the importance of compiler choice and the influence of container image size on startup times, emphasizing the need for careful optimization in containerized HPC workflows.

Keywords HPC · Performance · Docker · Singularity · WRF

✉ Juan C. Perez
jcperez@ull.edu.es

Pierre-Simon Callist Yannick Tondreau
ptondrea@ull.edu.es

Juan P. Díaz
jpdiaz@ull.es

Vicente Blanco Pérez
vblanco@ull.edu.es

Jonatan Felipe
jfelipe@iter.es

[1] Grupo de Observación de la Tierra y Atmósfera, Universidad de La Laguna (ULL), Santa Cruz de Tenerife, Spain

[2] Computer Science and Systems Department, Universidad de La Laguna (ULL), Santa Cruz de Tenerife, Spain

[3] Area de Tecnologia, Instituto Tecnologico y de Energias Renovables (ITER), Santa Cruz de Tenerife, Spain

Published online: 28 January 2025

1 Introduction

The Weather Research and Forecasting (WRF) model [1] is one of the most widely used numerical weather prediction systems, developed to meet both research and operational forecasting needs. Its extensive adaptability across various scales, from meters to thousands of kilometers [2–4], and its application in diverse atmospheric phenomena [5–7] have made WRF indispensable in meteorology, climate studies [8–10], and other earth sciences. According to Powers et al. [11] , the broad applicability of WRF is demonstrated by its use in institutions worldwide, where it supports tasks ranging from academic research to real-time operational forecasting .

The computational demands of WRF [12], particularly when configured for high-resolution simulations in large domains, require the use of high-performance computing (HPC) environments. The computational intensity arises from the complex physical processes being modeled, including detailed atmospheric dynamics, land surface interactions, and radiative transfer processes, which must also be simulated with a temporal resolution proportional to the spatial resolution. Since these are limited-area models, to account for processes occurring outside their region of interest, these models must be periodically fed with data from global-scale prediction or reanalysis models, such as those provided by the Global Forecast System (GFS) [13] or ERA5 reanalysis [14], to update boundary conditions and ensure accurate predictions. Therefore, due to their high computational complexity, as well as storage and communication requirements, the necessity of HPC systems for the execution of WRF limits its use to organizations with a significant computational infrastructure, such as national weather services and large research institutions [15].

As HPC environments evolve, there is a growing interest in enhancing the flexibility and reproducibility of deploying models like WRF. Containerization technologies have emerged as promising solutions that provide a way to encapsulate applications and their dependencies into portable, reproducible containers. Docker [16], the most popular containerization platform, has revolutionized software deployment across various fields by simplifying the management of application environments. However, Docker's reliance on privileged access to system resources raises security concerns, particularly in multi-user HPC environments where resource management is critical [17, 18]. Consequently, alternatives such as Singularity [19] have been developed specifically for HPC, offering better integration with HPC infrastructures while maintaining security by allowing users to run containers without root access [20].

Singularity has gained traction in the HPC community because of its ability to deliver near-native performance, a crucial requirement for performance-sensitive applications such as WRF. Studies have shown that while Singularity introduces some overhead compared to baremetal execution, it offers significant benefits in terms of portability, ease of deployment, and reproducibility [21].

Despite advances in containerization, the deployment of WRF remains a complex task due to the intricacies of the model itself and the architecture of the machines on which it runs. Various factors affect the performance and scalability of WRF, including the choice of compiler, the configuration of the problem to be solved, and the strategies used for parallel execution through MPI and OpenMP [22]. Previous studies, such as those of Christidis, have highlighted the importance of optimizing these factors to improve the performance of WRF on different HPC systems [23, 24]. Furthermore, the management of input/output (I/O) operations, which can become a bottleneck in large-scale simulations, plays a critical role in overall performance [15].

Given the complexity of WRF and the diverse architectures of HPC systems, this study aims to evaluate the performance and portability of WRF when deployed using Singularity containers. By systematically analyzing the effects of different compilers, execution strategies, and containerization approaches, this study aims to provide insights into the trade-offs between performance and portability. The findings will contribute to the ongoing efforts to optimize WRF deployments in modern HPC environments, balancing the need for computational efficiency with the benefits of containerized applications.

This paper is organized as follows. Section 2 provides an overview of the WRF model and the weather forecast case study used to test different configurations analyzed in this work. Section 3 details the hardware configuration used in our experiments. Section 4 outlines the experimental setup, including the specific configurations tested. Section 5 presents the results, highlighting key performance metrics and comparisons between different setups. Finally, Section 6 concludes the paper, summarizing our findings and suggesting directions for future research.

2 WRF model

The Weather Research and Forecasting (WRF) model [1] is a widely used numerical weather prediction system that serves both research and operational forecasting needs. Developed through a collaborative effort among multiple institutions, WRF provides a flexible framework for atmospheric simulations across scales ranging from meters to thousands of kilometers and with time horizons varying from a few hours to several decades. This flexibility has made WRF one of the most popular models in the meteorological and climate community, with applications ranging from academic research to real-time operational forecasting.

The extensive use of WRF in various fields, including climate studies, air quality research, and emergency response planning, underscores its versatility. For instance, the model's ability to simulate high-impact weather events has made it a critical tool for forecasting agencies worldwide, particularly in regions prone to extreme weather conditions.

2.1 Problem description

In this study, we employ WRF to perform a weather 24 h hindcast simulation (model run that is conducted using historical data to recreate past conditions) over a multi-domain setup, which includes three nested domains in the North Atlantic region centered over the Canary Islands. These domains are configured to progressively increase spatial resolution from the outermost to the innermost region, capturing finer atmospheric details as the simulation area narrows. The spatial resolutions of these domains are set to 27, 9, and 3 km, respectively, as shown in Fig. 1. The initial and boundary conditions are provided by the GFS data set [13] for the time interval between, where the boundary information is updated every six hours. The forecast runs for a period of one day, with output generated at hourly intervals, providing detailed temporal snapshots of the atmospheric state.

Due to the complexity of the Earth–Atmosphere–Ocean system, numerical models such as WRF used for its simulation need to perform a set of approximations. This is primarily because there are physical processes that occur at a much smaller spatial or temporal scale than the resolution chosen for the simulation. These include the microphysical processes involved in the formation of clouds or precipitation, the turbulence governing the atmospheric lower layer, or the processes governing the interaction of electromagnetic radiation in the system. In order to represent these processes not explicitly resolved by the model, a set of parameterizations is used that attempt to approximate these processes based on the variables that the model does resolve. Table 1 shows this set of unresolved processes as well as the scheme chosen in the simulations carried out in this work, which are crucial to the accuracy of the simulation and are carefully chosen based on the specific requirements of the study area [4]

Under these conditions, both the preprocessing of the input data and the 24-hour weather forecast are run in the HPC environment and the hourly outputs are saved in the storage system. These outputs include a large number of two-dimensional

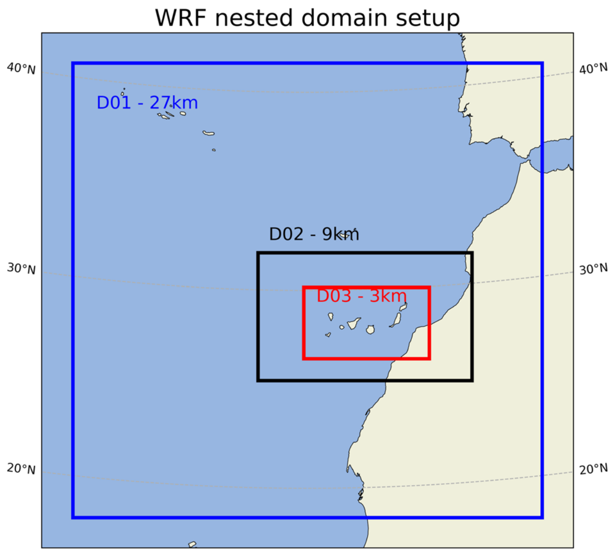

Fig. 1 WRF domain configuration. Domain 01: 100 x 97 grid points at 27 km resolution, Domain 02: 136 x 82 cells at 9 km, Domain 03: 238 x 136 cells at 3 km

Table 1 WRF model parameterizations used in the experiments

Physical process	Parameterization scheme	References
Cumulus parameterization	Betts–Miller–Janjic (BMJ)	[25]
Planetary boundary layer (PBL)	Mellor–Yamada–Janjic (MYJ)	[26]
Land surface model (LSM)	Unified Noah land surface model	[27]
Radiative processes	CAM shortwave and longwave schemes	[28]
Microphysics	Thompson microphysics	[29]

(latitude, longitude) and three-dimensional (latitude, longitude, height/depth) variables. As a reference, for the selected configuration, the simulation generates about 600 MB of data.

2.2 Compilation

Compilation of the WRF model is a critical step that directly impacts the performance of the simulations. For this study, two different compilers were used: the GNU Compiler Collection (GCC) and the Intel Classic Compiler. The choice of compiler can significantly affect the execution speed and efficiency of the WRF model, as each compiler optimizes code differently [30]. In our compilation process, we utilized the default optimization flags provided by each compiler, avoiding any custom optimizations to focus on the model's portability across different environments.

Load imbalances, a common issue in WRF simulations, occur when different processors complete their tasks at different times, leading to inefficiencies [23]. This imbalance can be exacerbated by the complex physical processes simulated and the varied computational demands in different domains [23]. Managing these imbalances is crucial to optimizing performance, particularly in large-scale simulations. However, in our study, we prioritize portability over performance optimization, in order to evaluate how well WRF can be deployed across different HPC environments using containerization technologies.

To manage the complex library dependencies required by WRF, we employed the Spack package manager. Spack [31] simplifies the installation and management of software packages in HPC environments, allowing reproducible WRF builds on different systems. This approach ensures that our focus remains on evaluating the portability of the model rather than optimizing it for a specific system configuration.

2.3 Configuration

The configuration of WRF simulations involves several parameters that directly influence performance, in addition to those affecting the representation of physical processes. One of the most critical aspects is domain decomposition, where the computational domain is divided among multiple MPI processes (or "tasks"). This decomposition is controlled by the nproc_x and nproc_y parameters, which determine the number of MPI processes allocated to each dimension of the domain grid. The choice of these parameters plays a pivotal role in balancing communication overhead and computational efficiency. In our experiments, we used WRF's default settings for nproc_x and nproc_y, which typically assigns nproc_y to be greater than or equal to nproc_x, with minimal difference between the two. This configuration is designed to optimize cache usage and reduce communication costs.

In addition to dividing the domain between MPI processes, WRF further subdivides the computational grid based on the number of OpenMP threads assigned to each MPI process. Each MPI process is responsible for a specific section of the domain, and within that section, the grid is further divided into smaller "tiles" that are distributed across the OpenMP threads. This approach allows for a hybrid parallelization strategy, leveraging both MPI for inter-process communication and OpenMP for multi-threading within each process. By adjusting the number of OpenMP threads, we effectively altered the distribution of WRF tiles, which impacted performance depending on the computational resources available.

Another crucial configuration aspect is the Input/Output (I/O) management, which can become a bottleneck in large-scale simulations. WRF offers several options to optimize I/O, such as asynchronous I/O and parallel I/O, which can significantly reduce the time spent writing output files. This is especially critical when the number of computational cores used in the simulation is very high [22]. However, to maintain the focus on portability, we opted to use the default I/O settings, which are more broadly applicable across different HPC environments.

By configuring WRF with a balance between default settings and necessary adjustments for our study, we aimed to create a portable and reproducible simulation environment. This approach allows us to assess how WRF performs in various containerized setups, providing insights into the trade-offs between portability and performance in HPC applications.

3 Hardware

The experiments carried out in this study were executed on the Teide High-Performance Computing (TeideHPC) system, a state-of-the-art supercomputing facility managed by the Instituto Tecnológico y de Energías Renovables (ITER) in Tenerife, Spain [32]. TeideHPC is designed to support a broad spectrum of computationally intensive scientific and industrial applications, making it an ideal platform for the execution of demanding simulations like those of the WRF model.

TeideHPC's architecture is based on a cluster of compute nodes, each equipped with dual Intel Xeon E5-2670 processors from the Sandy Bridge generation, as shown in Fig. 2. These processors provide 16 cores per node (8 cores per processor), offering substantial computational power that is crucial for the parallel processing requirements of high-resolution weather simulations. Each node is also outfitted with 32 GB of RAM, sufficient to handle the large memory demands of the WRF model and similar computational tasks.

The compute nodes in TeideHPC are interconnected via a 100 Gbps Infiniband [33] EDR network, which ensures low-latency and high-throughput communication

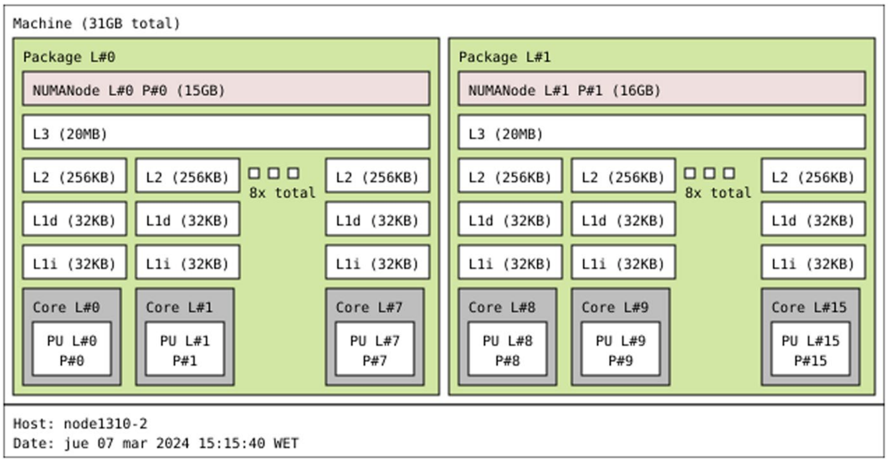

Fig. 2 Architecture of a compute node featuring dual Intel Xeon E5-2670 processors, showcasing the core distribution and memory organization. The node is divided into two processor packages: Package L0 with 16 GB of memory and Package L1, also with 16 GB of memory, each corresponding to a NUMA (Non-Uniform Memory Access) node. Both packages have 8 cores each (L0 to L7 for Package 0 and L8 to L15 for Package 1). Each core has dedicated L1 and L2 caches, with 32KB for both instruction and data at the L1 level and 256KB for L2. A shared 20MB L3 cache is accessible by all cores within each package

between nodes. This network configuration is essential for minimizing the time spent on inter-node communication during parallel computations, thereby optimizing the overall performance of the HPC environment.

TeideHPC also features a NetApp storage system with a capacity of 2.6 petabytes, configured in a clustered setup with redundancy and spare disks. This storage infrastructure is designed to prevent data loss and ensure high availability, which is critical for long-running simulations and data-intensive applications.

For the experiments described in this study, exclusive access to a subset of the total compute nodes available was granted, ensuring that the hardware resources were fully dedicated to our work. This exclusivity was key in eliminating potential performance variability due to resource sharing, thereby providing consistent and reliable benchmarking results.

In terms of software, the system operates on a Linux-based operating system, which is standard in HPC environments due to its stability and optimization capabilities. Singularity was used for running containerized applications, offering a secure and performance-optimized alternative to Docker, which was initially used to build the container images. As mentioned before, the management of software dependencies was handled by Spack.

4 Experimental setup

The experimental setup for this study was designed to assess the performance and portability of the Weather Research and Forecasting (WRF) model across various configurations in high-performance computing (HPC) environments. The experiments focused on comparing the performance of WRF in traditional baremetal deployments versus containerized environments using Singularity, analyzing the impact of different compilers, and exploring various parallelization strategies. This section outlines the key components of the experimental setup, including the containerization approach, automation of deployment, and the specific experiments conducted.

4.1 Containerization with Singularity

The WRF model was initially containerized using Docker. Dockerfiles were created to define the environment, including the base operating system, necessary libraries, MPI configurations, and the WRF model itself. Once the Docker images were built, they were converted into Singularity images using the *docker2singularity* tool. This process ensured that the containerized application retained all the benefits of Docker's extensive ecosystem while being optimized for execution in an HPC environment using Singularity.

One key aspect of the container build was the use of Spack to manage the installation of the WRF model and its dependencies. The Dockerfile employed a multi-stage build process, beginning with a Spack pre-installed image where necessary tools like GCC, OpenMPI, and WRF were compiled and installed. The Dockerfile specified the installation and concretization of WRF (version 4.5) and WPS (Weather Preprocessing System) through Spack, ensuring that all software packages were optimized for the target architecture and installed into a uniform environment. Furthermore, Spack allowed us to activate the environment with the necessary configurations through a profile script, ensuring that the required paths and environment variables were correctly set at runtime.

To minimize the size of the final container image, all binaries were stripped of unnecessary debugging symbols and metadata, further improving execution efficiency by reducing startup overhead.

While Singularity provides significant benefits in terms of portability and ease of deployment, there is an inherent performance overhead due to the additional layer of abstraction between the application and the hardware. This study aimed to quantify this overhead and determine whether the advantages of containerization, such as improved reproducibility and deployment flexibility, justify the associated performance costs. Previous studies have shown that Singularity can achieve near-native performance, although the degree of overhead varies depending on the specific application and HPC system [34].

4.2 Automation of deployment

To streamline the deployment and execution of the experiments, we developed custom automation tools. They were designed to manage the entire workflow, from configuring the HPC environment to running the simulations and collecting performance data. The automation ensured that each experiment was executed under identical conditions, minimizing human error and variability. The scripts also facilitated the efficient deployment of the Singularity containers across multiple nodes, automatically configuring the necessary environment variables, and handling the input/output (I/O) operations required by WRF.

The automation software was developed using a combination of Bash scripting and Python, designed to integrate seamlessly with the Slurm workload manager on TeideHPC. All experiments were executed using the *srun* command, with the *–exclusive* attribute to ensure whole-node allocation for optimal resource usage. This approach provided several benefits: by allocating entire nodes exclusively to each job, it minimized potential resource contention from other processes, allowed full control over node resources (such as CPU cores and memory), and ensured more predictable performance. This setup also facilitated efficient scaling across multiple nodes and ensured uniform configuration for both baremetal and containerized environments.

Table 2 Experiment execution configurations

Conf.	Env.	Comp.	Nodes	Tasks x Node	CPU x Task
			1	1	8
8	Singularity	Intel	1	2	4
Cores	Baremetal	GCC	1	4	2
			1	8	1
			1	1	16
			1	2	8
			1	4	4
16	Singularity	Intel	1	8	2
			1	16	1
Cores	Baremetal	GCC	2	1	8
			2	2	4
			2	4	2
			2	8	1
			2	1	16
			2	2	8
			2	4	4
32	Singularity	Intel	2	8	2
			2	16	1
Cores	Baremetal	GCC	4	1	8
			4	2	4
			4	4	2
			4	8	1

4.3 Experiments

A comprehensive set of experiments was conducted to assess the performance of WRF under various conditions, as shown in Table 2. The experiments were designed to compare the following.

- Baremetal vs. Containerized Execution: Evaluating the performance difference between running WRF natively on the hardware (baremetal) and within a Singularity container.
- Compiler Impact: Analyzing how different compilers (e.g., Intel vs. GCC) affect the performance of WRF, considering aspects such as compilation time, execution time, and optimization levels. Note that the optimization levels analyzed in these experiments are specific to the WRF configuration options rather than compiler-specific optimizations
- Execution Strategies: Assessing the impact of different execution strategies, such as varying the number of tasks per node and the use of CPUs per task, which will be used to compare the performance of MPI vs. OpenMP.

Several configurations were tested by varying the number of cores, the environment (baremetal vs. containerized), the compiler, and the task allocation strategy. These configurations are detailed in Table 2.

- Nodes: Refers to the number of physical machines used for each experiment. In this case, the experiments used 1, 2, or 4 compute nodes.
- Tasks per Node: Denotes the number of parallel tasks (MPI processes) assigned to each node. Different configurations varied the number of tasks, ranging from 1 to 16, in order to examine the scalability of WRF under different parallelization strategies.
- CPUs per Task: Specifies the number of CPU cores allocated to each MPI task. This value was varied to observe how dividing resources among tasks affected overall performance.

The primary metric used for evaluating performance was the elapsed wall clock time to complete a 1-day simulation over the 3 nested domains. This metric provided a clear indication of the computational efficiency of WRF under different configurations. Additional metrics included CPU utilization, memory usage, and I/O performance, which were analyzed to provide a comprehensive understanding of how different factors influenced overall performance.

The combination of these tests allowed for a thorough analysis of how these configurations impact the performance of the WRF model in a high-performance computing environment, providing valuable insights into the trade-offs between portability and performance.

5 Results

Performance evaluation of the WRF model was conducted through a series of experiments designed to compare different execution environments, compilers, and execution strategies. The primary focus was on understanding the impact of running WRF on baremetal hardware versus within a Singularity container, as well as assessing the influence of different compilers and parallelization strategies. The results presented here provide insights into the trade-offs between performance and portability in a high-performance computing (HPC) context. Following the configurations described in Table 2, three simulations were performed for each of them and the results presented in this section correspond to the average value of their execution times, in order to avoid spurious results as a consequence of an anomalous execution. In general, as the machine was exclusively dedicated to these experiments, the execution times obtained were quite homogeneous for the 3 simulations.

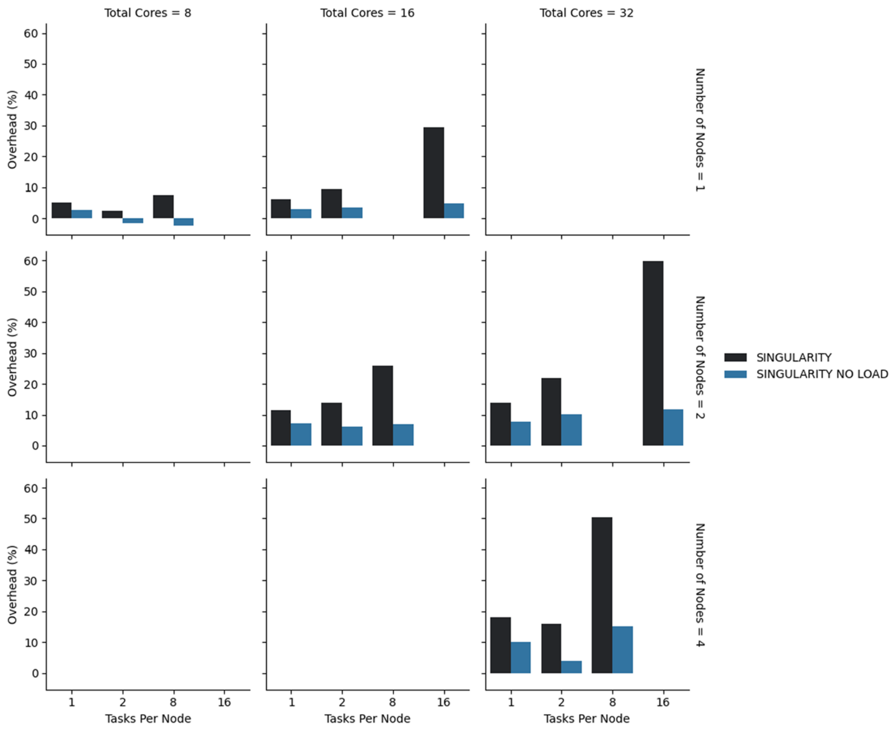

Fig. 3 Singularity WRF time execution overhead compared to baremetal. The results are grouped according to the number of cores used (columns) as well as by the number of nodes used (rows)

5.1 Baremetal versus Singularity container performance

One of the primary goals of this study was to compare the performance of WRF when executed on baremetal hardware versus within a Singularity container. Figure 3 shows the average overhead of different experiments performed using Singularity with respect to the average of the best runs with the same configuration executed directly on the platform. In general, a performance degradation is observed when using Singularity containers, attributed to the overhead introduced by the container load process and the containerization layer.

This comparison, which includes simulations performed with different compilers, shows that, if the initial loading time of the containers on the compute nodes is not taken into account, the degradation introduced by containerization does not exceed 15% and that this degradation, in general, increases with the number of nodes used as well as with the number of MPI tasks executed per node, which reveals the overhead introduced as a consequence of inter-container communications. However, when the container loading time is considered, it is noted that the degradation increases considerably and that this degradation is more pronounced as the number of tasks (containers) per node increases.

Analyzing the initialization process, the overhead in loading Singularity containers is closely related to their size, as it directly impacts both the time required to read them from the storage system where they reside and the time needed to decompress these files and execute their contents. After analyzing various factors that influence the loading time, it becomes evident that the decompression of the container image accounts for the majority of the time consumed. This overhead becomes particularly evident when increasing the number of containers per node, as can be seen in Fig. 4, due to competition for node resources, primarily the memory access penalty. Such contention can significantly degrade the efficiency of container loading, leading to a noticeable drop in overall performance as can be appreciated in Fig. 3 when the number of containers per node increases to 16. Therefore, a critical factor is to minimize the container size by removing all libraries that are not strictly necessary.

5.2 Impact of compilers

The choice of compiler also significantly impacted the performance of the WRF model. Two major compilers were tested: Intel and GCC, each known for their strengths in different aspects of HPC workloads.

Figure 5 shows the results of different experiments grouped by the compiler used. In a configuration with 4 nodes, 8 tasks per node and 2 CPUs per Task, the Intel compiler on baremetal achieved an average execution time of 1694.75 s, whereas the GCC compiler took 1723.94 s. The same configuration in Singularity (excluding load time) resulted in execution times of 2066.85 s with Intel and 1872.43 s with GCC, indicating that while Intel showed superior performance in baremetal, the relative advantage was reduced in the containerized environment due to the added overhead.

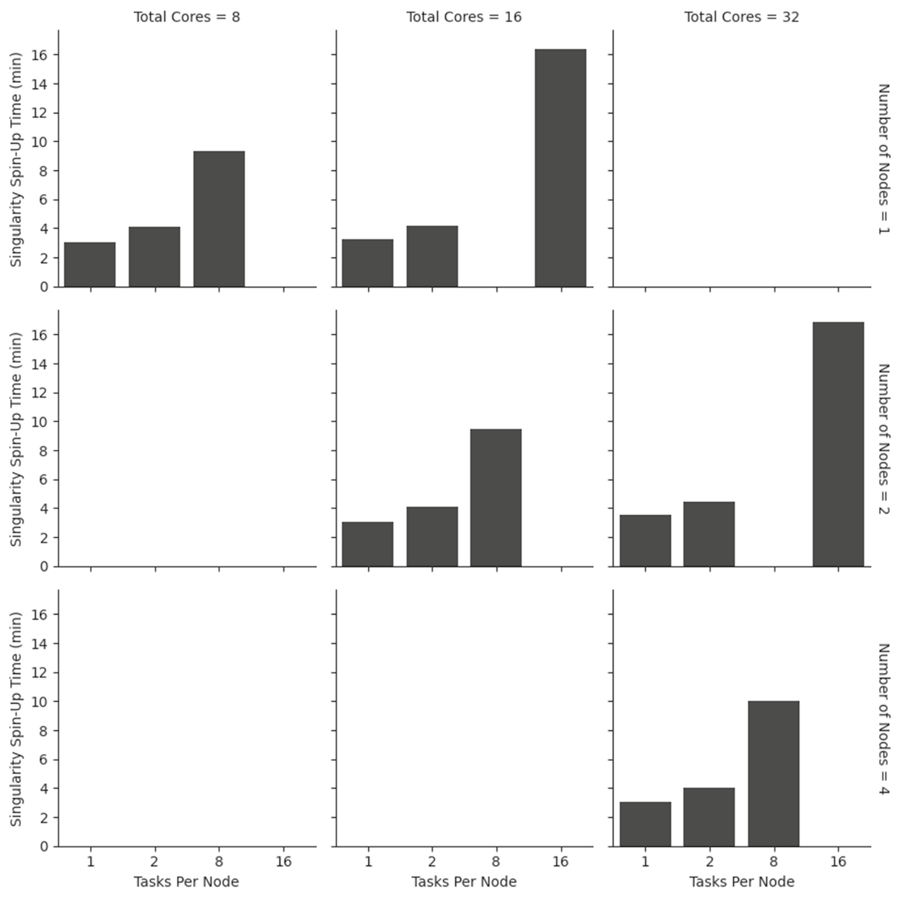

Fig. 4 Singularity container spin-up (load) times for different node and task configurations. The x-axis represents the number of tasks per node (1, 2, 8, and 16), while the y-axis indicates the spin-up time in minutes. The panels are organized by total core counts (8, 16, and 32 cores) and the number of nodes (1, 2, and 4)

In general, the Intel compiler showed an average performance improvement of 1.7% over GCC in baremetal environments, but this margin decreased to 1.2% in Singularity. This trend suggests that the optimizations provided by Intel are somewhat mitigated by the containerization process. The results are consistent with previous studies [30] that have compared the efficiency of these two compilers for WRF and a specific hardware environment, with Intel typically offering better performance in handling complex computations and parallel processing tasks.

A more detailed analysis of the results obtained is shown in Fig. 6, which show the execution times of different experiments for executables generated by the Intel and GCC compilers, respectively, considering also the effect of containerization. It is worth noting the penalty introduced when the simulation is performed in the contanerized environment when using the Intel compiler, which is appreciably higher

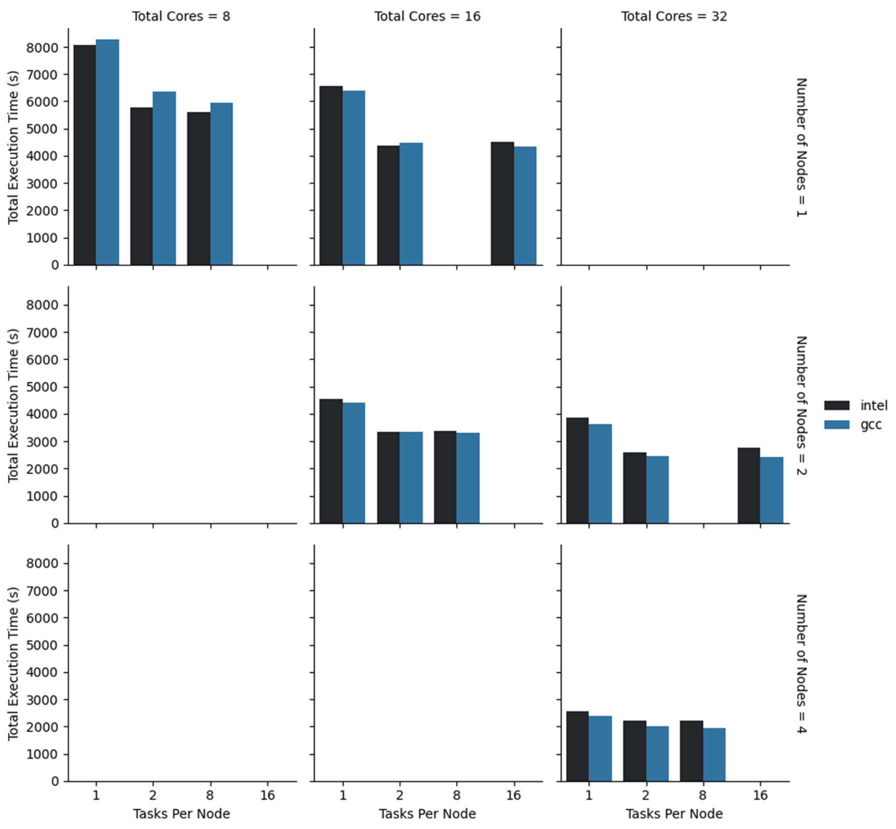

Fig. 5 Execution times based on the compiler used: ICC (black) vs. GCC (blue). The columns group the experiments according to the total number of cores used: 8 (left), 16 (center), and 32 (right), while the rows group the results according to the number of nodes used: 1 (top), 2 (middle), and 4 (bottom) (Color figure online)

than that for GCC executables. It is also observed that this overhead is mainly due to the loading time of the containers, due to the difference in size between the two containers, twice as large in the case of Intel libraries, which justifies that the difference in performances for both compilers is reduced for containerized environments.

5.3 Execution strategies: MPI and OpenMP

The experiments also explored the impact of different parallel execution strategies, specifically focusing on MPI (Message Passing Interface) and OpenMP (Open Multi-Processing). The results indicated that the configuration and tuning of these parallelization strategies play a crucial role in optimizing the performance of WRF.

As expected, the WRF model scales quite well, as has been extensively evaluated by other authors for HPC environments with up to tens of thousands of processors [22]. Focusing on the execution strategy, Fig. 7 shows the execution times for

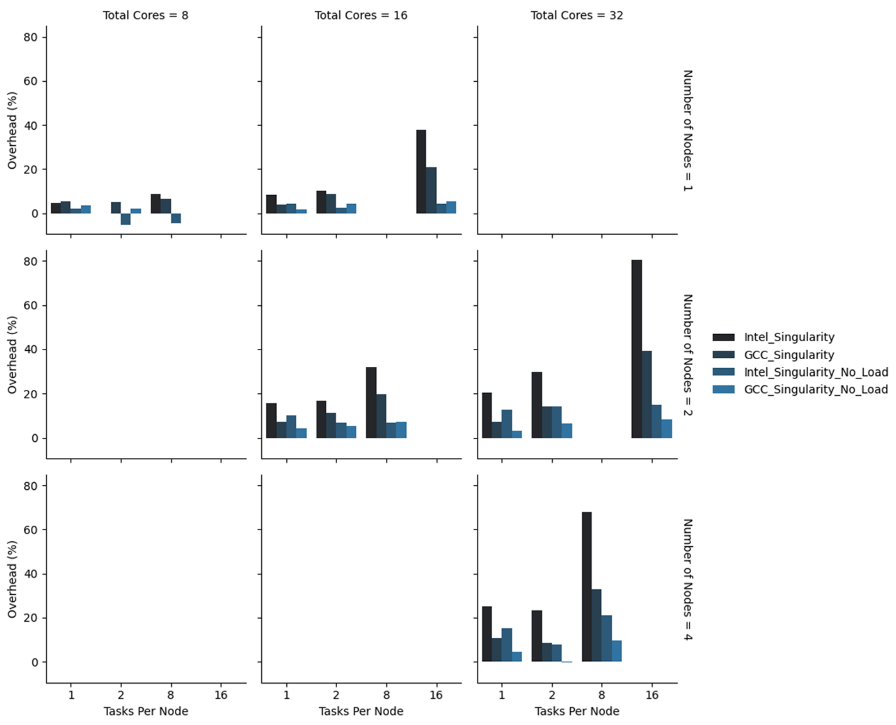

Fig. 6 Execution times overhead based on the compilers in Singularity against baremetal: ICC in Singularity (black) vs. GCC in Singularity (dark blue) vs. ICC in Singularity without load time (blue) vs. GCC in Singularity without load time (light blue). The columns group the experiments according to the total number of cores used: 8 (left), 16 (center), and 32 (right), while the rows group the results according to the number of nodes used: 1 (top), 2 (middle), and 4 (bottom) (Color figure online)

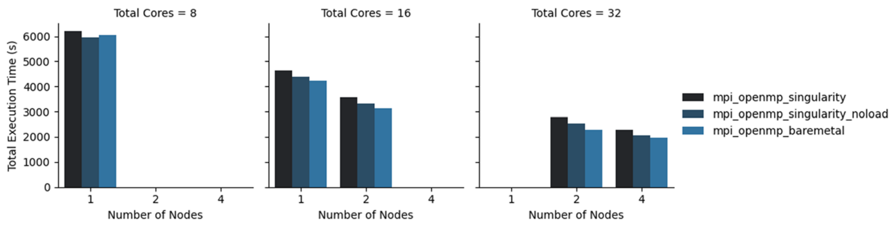

Fig. 7 Execution times for MPI+OpenMP Hybrid strategy: Singularity (black), Singularity without load (blue), and baremetal (light blue). Grouped by core count (8, 16, and 32) and number of nodes (1, 2, 4) (Color figure online)

different configurations in terms of the total number of cores and the total number of nodes used when the mixed MPI and OpenMP strategy is employed. Since during the execution of the tests, the number of threads assigned to a processor never exceeds the maximum number of cores it has, one initial conclusion drawn from these results is that, given the same number of cores used, configurations that utilize more nodes provide better performance than those running on fewer nodes. This

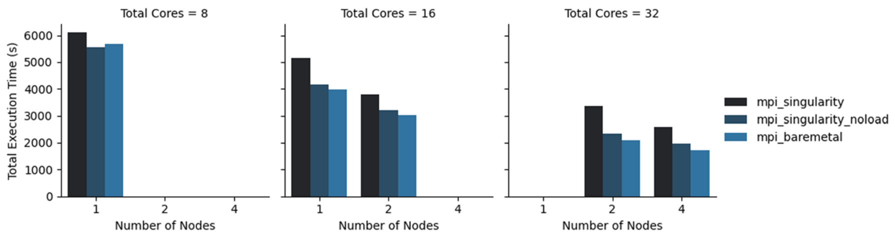

Fig. 8 Execution times for Full MPI strategy: Singularity (black), Singularity without load (blue), and baremetal (light blue). Grouped by core count (8, 16, and 32) and number of nodes (1, 2, 4) (Color figure online)

highlights the importance of memory access for this application, in which, despite the communication overhead between different nodes, the performance improves when the computational domains are distributed across multiple nodes.

If we focus exclusively on the executions performed with the message-passing strategy, Fig. 8, the same behavior is observed regarding the distribution of cores across different nodes. Furthermore, if we also consider the results shown in Figs. 3, 4, 56 concerning the relevance of assigning the number of tasks per node, a significant improvement is observed when moving from 1 to 2 tasks per node. However, this improvement is not as noticeable when increasing the number of tasks per node beyond 2, which aligns with a distribution of the computational domain into as many tasks as processors available.

Finally, if we adopt a strategy based exclusively on OpenMP, Fig. 9, a significant penalty (ranging from 30% to 50%) in execution times is observed compared to hybrid configurations. This increased overhead is due to the combined effects of several factors related to the efficient utilization of the memory hierarchy within compute nodes.

As mentioned above, WRF computationally partitions the domain between the requested MPI tasks. Within each patch, the OpenMP strategy takes over, parallelizing the loops responsible for solving the physical equations using threads. As a result, reducing the number of MPI tasks, thereby increasing the size of the resulting patches, negatively affects the efficient utilization of the memory hierarchy.

To evaluate the memory usage during the execution of the WRF model under different configurations, several performance indicators were analyzed, and the

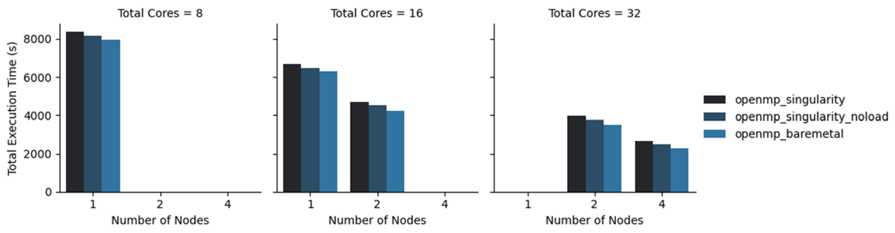

Fig. 9 Execution times for Full OpenMP strategy: Singularity (black), Singularity without load (blue), and baremetal (light blue). Grouped by core count (8, 16, and 32) and number of nodes (1, 2, 4) (Color figure online)

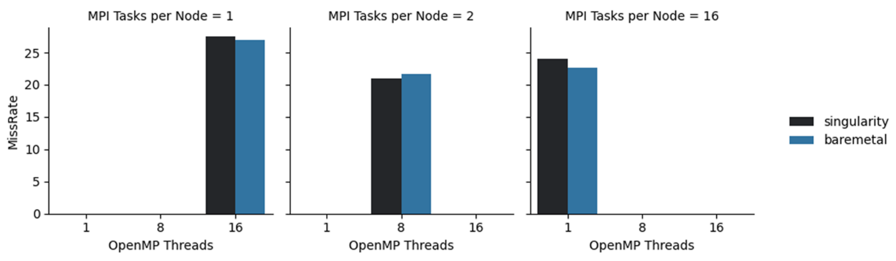

Fig. 10 Comparison of LLC data cache miss rate across varying OpenMP thread counts (1, 8, and 16) for different MPI task distributions per node (1, 2, and 16)

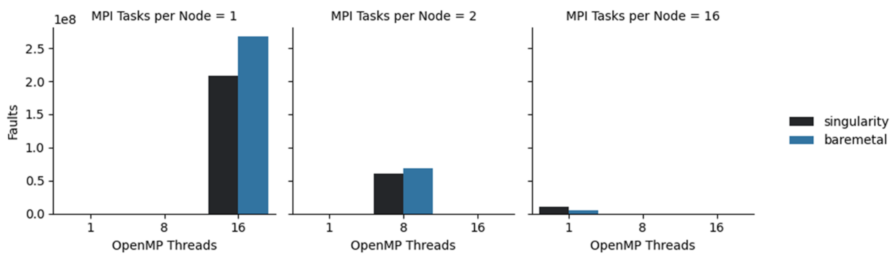

Fig. 11 Page faults registered as a function of the OpenMP thread counts (1, 8, and 16) for different MPI task distributions per node (1, 2, and 16)

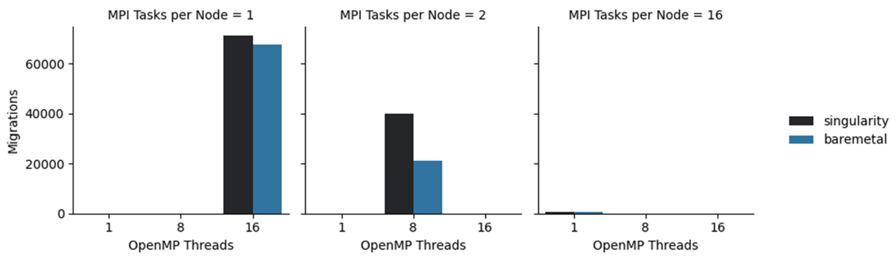

Fig. 12 Migration counts across varying OpenMP thread counts (1, 8, and 16) for different MPI task distributions per node (1, 2, and 16)

results are shown in Figs. 10, 11, and 12, which, respectively, present the percentage of L3 cache misses, the number of page faults, and the number of thread migrations for a single compute node under three distinct scenarios.

When evaluating the cache misses impact, it can be seen that the penalty of a pure OpenMP strategy increases by approximately 8% compared to the pure MPI executions. However, page fault and thread migration metrics exhibit a much more pronounced differentiation. Specifically, in the pure OpenMP strategy, the number of page faults escalates due to the larger domain size, leading to a significant increase in thread migrations. This, in turn, results in the invalidation of the level 1 and level 2 caches, causing a corresponding reduction in performance.

5.4 Proposed solution

From the previous analysis, it can be deduced that to make efficient use of the HPC platform where the model is executed, it is convenient to carefully analyze the size of the resulting patches (MPI processes) to optimize the use of different levels of cache memory available in the architecture. This approach favors the use of MPI tasks over the intensive use of OpenMP threads. However, since the primary goal of this work is to facilitate deployment and portability across different HPC environments, the study conducted allows proposing an easy-to-implement rule that enables the execution of WRF simulations while minimizing performance losses.

For a given architecture with N compute nodes, each equipped with P processors and C cores per processor, we propose the use of Singularity containers combined with a hybrid OpenMP/MPI strategy. In this approach, $N \times P$ Singularity containers are deployed as MPI tasks, with each task utilizing C OpenMP threads. This configuration strikes a balance between domain partitioning into patches assigned to each MPI task and limiting the overhead associated with an increased number of Singularity containers.

Using this rule of thumb, we conducted several simulations using 2 nodes with the same configuration described in Sect. 2, extending the forecast horizon to 3 days to assess the penalties introduced by using containers compared to the native solution. Figure 13 shows the average execution times using the proposed strategy and the best configuration, i.e., running directly over baremetal using a full MPI strategy. The data reveal an overhead of 16.8% when employing the Singularity container-based strategy. However, this penalty is deemed acceptable given the significant benefits provided by the containerized solution, including enhanced portability and ease of deployment.

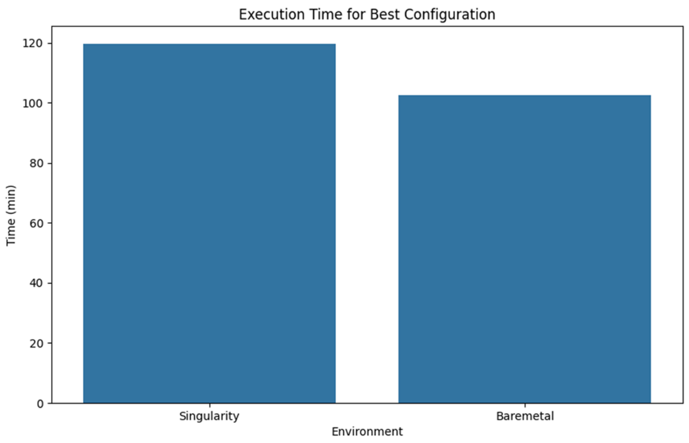

Fig. 13 Comparison of execution times for 3-day weather prediction simulations using the proposed configuration: a) Singularity environment, b) Baremetal environment

6 Conclusions

This paper evaluates the use of containers for the deployment of the WRF application in HPC environments. A set of experiments is designed to evaluate different aspects that affect the performance of the runs of this application and how these effects are transferred to a containerized environment. Thus, different experiments were carried out using different compilers and different execution strategies with MPI and OpenMP both in baremetal systems and using Singularity containers.

The results showed that proper domain decomposition and task allocation were crucial for optimizing performance.

Regarding the compiler choice, although in general runs with the intel compiler provide better results, when migrating the corresponding libraries to a containerized environment, the size of the resulting image causes a penalty in the overall performance.

Another critical aspect is the choice of the execution strategy. While in traditional direct run on baremetal systems, in general, an increase in MPI tasks provides better results, when using Singularity a compromise between the number of containers to be deployed and the number of OpenMP threads assigned to each container must be found due to the high cost of loading the containers on the compute nodes.

These results illustrate the performance trade-offs between compilers and the overhead introduced by containerization.

Another key contribution of this work was the development of an automation framework that simplifies WRF deployment across heterogeneous HPC environments. This automation frees users, particularly those without HPC expertise, from the complexities of compilation and configuration. The ability to deploy WRF easily and consistently on various HPC systems ensures reproducibility and simplifies the overall process.

Despite the observed performance overhead of approximately 15% due to containerization, the benefits of portability, ease of deployment, and reproducibility make Singularity a valuable tool for WRF deployment in environments with varied HPC resources. For example, even with 2 nodes and 2 tasks per node, performance degradation in Singularity was noticeable, but acceptable when considering the convenience of containerization. The ability to run complex simulations in environments that lack dedicated HPC infrastructure democratizes access to high-performance weather simulation tools, allowing broader participation from users, institutions, and countries with limited resources.

Acknowledgements The authors wish to acknowledge the contribution of Teide High-Performance Computing facilities. TeideHPC facilities are provided by the Instituto Tecnológico y de Energías Renovables (ITER), S.A. URL: teidehpc.iter.es. The authors also thank the PLANCLIMAC2 Project (1/MAC/2/2.4/0006) for its support. This Project has been financed by the European Union INTERREG MAC 2021-2027 Program.

Author contributions These authors contributed equally to this work.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no conflict of interest to declare that is relevant to the content of this article.

Ethical approval Not applicable

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Skamarock C, Klemp B, Dudhia J, Gill O, Liu Z, Berne, J, Wang W, Powers G, Duda G, Barker DM, Huang X (2019) A description of the advanced research WRF model version 4. https://api.semanticscholar.org/CorpusID:196211930
2. Prosper MA, Otero-Casal C, Canoura Fernandez F, Miguez-Macho G (2019) Wind power forecasting for a real onshore wind farm on complex terrain using WRF high resolution simulations. Renew Energy 135:674–686
3. Avisar D, Pelta R, Chudnovsky A, Rostkier-Edelstein D (2021) High resolution WRF simulations for the tel-aviv metropolitan area reveal the urban fingerprint in the sea-breeze hodograph. J Geophys Res Atmos 126(5):33691
4. Pérez J, Díaz J, González A, Expósito J, Rivera-López F, Taima D (2014) Evaluation of WRF parameterizations for dynamical downscaling in the canary islands. J Clim 27(14):5611–5631
5. Yang Q, Yu Z, Wei J, Yang C, Gu H, Xiao M, Shang S, Dong N, Gao L, Arnault J, Laux P, Kunstmann H (2021) Performance of the WRF model in simulating intense precipitation events over the hanjiang river basin, china - a multi-physics ensemble approach. Atmos Res 248:105206. https://doi.org/10.1016/J.ATMOSRES.2020.105206
6. Dulière V, Zhang Y, Salathé EP (2013) Changes in twentieth-century extreme temperature and precipitation over the western united states based on observations and regional climate model simulations. J Clim 26(21):8556–8575. https://doi.org/10.1175/JCLI-D-12-00818.1
7. González A, Pérez JC, Díaz JP, Expósito FJ (2017) Future projections of wind resource in a mountainous archipelago, canary islands. Renew Energy 104:120–128
8. Carrillo J, Hernández-Barrera S, Expósito FJ, Díaz JP, González A, Pérez JC (2023) The uneven impact of climate change on drought with elevation in the canary islands. NPJ Climate Atmos Sci 6(1):31
9. Carrillo J, González A, Pérez JC, Expósito FJ, Díaz JP (2022) Projected impacts of climate change on tourism in the canary islands. Reg Environ Change 22(2):61
10. Expósito FJ, González A, Pérez JC, Díaz JP, Taima D (2015) High-resolution future projections of temperature and precipitation in the canary islands. J Clim 28(19):7846–7856
11. Powers JG, Klemp JB, Skamarock WC, Davis CA, Dudhia J, Gill DO, Coen JL, Gochis DJ, Ahmadov R, Peckham SE, Grell GA, Michalakes J, Trahan S, Benjamin SG, Alexander CR, Dimego GJ, Wang W, Schwartz CS, Romine GS, Liu Z, Snyder C, Chen F, Barlage MJ, Yu W, Duda MG (2017) The weather research and forecasting model: Overview, system efforts, and future directions. Bull Am Meteor Soc 98:1717–1737. https://doi.org/10.1175/BAMS-D-15-00308.1
12. Sobhani N (2017) Applications, performance analysis, and optimization of weather and air quality models. PhD thesis, University of Iowa. https://doi.org/10.17077/etd.gzcokjty

13. National Centers for Environmental Prediction, National Weather Service, (2007) NOAA, U.S. Department of Commerce: NCEP Global Forecast System (GFS) Analyses and Forecasts. Research Data Archive at the National Center for Atmospheric Research, Computational and Information Systems Laboratory, Boulder CO. https://rda.ucar.edu/datasets/dsd084006/
14. Hersbach H, Bell B, Berrisford P, Biavati G, Horányi JA, Muñoz Sabater Nicolas J, Peubey C, Radu R, Rozum I, Schepers D, Simmons A, Soci C, Dee D, Thépaut J-N (2023) Era5 hourly data on single levels from 1940 to present. Technical report, Copernicus Climate Change Service (C3S) Climate Data Store (CDS). https://doi.org/10.24381/cds.adbb2d47
15. Andraju P, Kanth AL, Kumari KV, Vijaya S, Rao B (2019) Performance optimization of operational WRF model configured for Indian monsoon region. Earth Syst Environ 3:231–239. https://doi.org/10.1007/s41748-019-00092-2
16. Merkel D (2014) Docker: lightweight linux containers for consistent development and deployment. Linux J 2014(239):2
17. Benedicic L, Cruz FA, Madonna A, Mariotti K (2019) Sarus: Highly scalable docker containers for HPC systems. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11887, pp. 46–60. Springer, ???
18. Gantikow H, Reich C, Knahl M, Clarke N (2016) Providing security in container-based HPC runtime environments. In: High Performance Computing, vol. 9945, pp. 685–695. Springer, Cham
19. Kurtzer GMea (2018) Singularity 2.5.2 - Linux application and environment containers for science. Zenodo. https://doi.org/10.5281/zenodo.1308868
20. Rudyy O (2019) Viabilidad del uso de contenedores en entornos HPC. Universitat Politècnica de Catalunya
21. Wang Y, Evans RT, Huang L (2019) Performant container support for HPC applications. In: Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (Learning). PEARC '19. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3332186.3332226
22. Kruse C, Del Vento D, Montuoro R, Lubin M, McMillan S (2013) Evaluation of WRF scaling to several thousand cores on the yellowstone supercomputer. In: Proceedings of the Front Range Consortium for Research Computing Conference, vol. 14
23. Christidis Z (2015) Performance and scaling of WRF on three different parallel supercomputers. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9137 LNCS, pp. 514–528. Springer, ???. https://doi.org/10.1007/978-3-319-20119-1_37
24. Moreno R, Arias E, Cazorla D, Pardo JJ, Navarro A, Rojo T, Tapiador FJ (2020) Analysis of a new MPI process distribution for the weather research and forecasting (WRF) model. Sci Program 1:8148373. https://doi.org/10.1155/2020/8148373
25. Janjic ZI (1994) The step-mountain eta coordinate model: further developments of the convection, viscous sublayer, and turbulence closure schemes. Mon Weather Rev 122(5):927–945
26. Janjic ZI (1994) The step-mountain eta coordinate model: Further developments of the convection, viscous sublayer, and turbulence closure schemes. Mon Weather Rev 122(5):927–945. https://doi.org/10.1175/1520-0493(1994)122<0927:TSMECM>2.0.CO;2
27. Tewari M, Chen F, Wang (2004) Implementation and Verification of the Unified Noah Land-surface Model in the WRF Model. In: 20th Conference on Weather Analysis and Forecasting/16th Conference on Numerical Weather Prediction
28. Collins W, Rasch BA, Boville PJ, McCaa J, Williamson (2004) Description of the ncar community atmosphere model CAM 3.0. Technical report, University Corporation for Atmospheric Research
29. Thompson G, Field PR, Rasmussen RM, Hall WD (2008) Explicit forecasts of winter precipitation using an improved bulk microphysics scheme. part ii: Implementation of a new snow parameterization. Monthly Weather Review 136(12), 5095–5115 . https://doi.org/10.1175/2008MWR2387.1
30. Malakar P, Saxena V, George T, Mittal R, Kumar S, Naim AG, Husain SAbH (2012) Performance evaluation and optimization of nested high resolution weather simulations. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012 Parallel Processing, pp. 805–817. Springer, Berlin, Heidelberg
31. Gamblin T, LeGendre M, Collette MR, Lee GL, Moody A, de Supinski BR, Futral S (2015) The spack package manager: bringing order to HPC software chaos. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '15. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2807591.2807623
32. ITER: TeideHPC . [Online; accessed 4-July-2024] (2024). https://www.iter.es/

33. Shanley T (2002) Infiniband. Addison-Wesley Longman Publishing Co., Inc, USA
34. Hu G, Zhang Y, Chen W (2019) Exploring the performance of singularity for high performance computing scenarios. In: Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019, pp. 2587–2593. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.003

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.