Hindawi
Scientific Programming
Volume 2020, Article ID 8148373, 13 pages
https://doi.org/10.1155/2020/8148373

Research Article

Analysis of a New MPI Process Distribution for the Weather Research and Forecasting (WRF) Model

R. Moreno, E. Arias, D. Cazorla, J. J. Pardo, A. Navarro, T. Rojo, and F. J. Tapiador

University of Castilla-La Mancha, Albacete, Spain

Correspondence should be addressed to J. J. Pardo; juanjose.pardo@uclm.es

Received 24 September 2019; Revised 28 November 2019; Accepted 14 December 2019; Published 17 January 2020

Academic Editor: Antonio J. Peña

The standard method used in the Weather Research and Forecasting (WRF) model for distributing MPI processes across the processors is not always optimal. This circumstance affects performance, i.e., execution times, but also energy consumption, especially if the application is to be extended to exascale. The authors found that the reason why the standard method for process distribution is not always optimal was an imbalance between the orthogonality of the communication and the proper cache usage, and this affects energy consumption. We present an improved MPI process distribution algorithm that increases the performance. Furthermore, scalability analyses for the new algorithm are presented and the energy use of the system is evaluated. A solution for balancing energy use with performance is also proposed for cases where the former is a concern.

1. Introduction

Weather forecasting is becoming increasingly important in people's everyday lives. It is as important for a person who wants to have a good weekend as it is for an agency that has to plan a world-class event like the Winter Olympics. Moreover, increasingly higher resolution forecasts are being demanded, which involves the need to increase the computational power used for these forecasts, even reaching exascale. From an operational point of view, performance is the most important computational aspect of systems providing weather forecasts, as these must be generated in a short period of time, without losing sight of the energy consumption of these computational resources. Thus, a forecast for the next 12 hours should be computed in less than an hour in order to be useful for operations. For this reason, the use of more computational resources is demanded. Increasing the number of processors used on a weather simulation allows the problem to be split into smaller subproblems, but at the cost of an increased communication throughput. This increase in computational resources cannot be sustained indefinitely; at some point, the computational workload of the subproblems will be so low that communications will become a bottleneck, avoiding further reduction in the computation times.

The Weather Research and Forecasting (WRF) model package is a well-known weather forecasting software used extensively around the world. WRF makes use of parallel computing, which has allowed it to be executed on many supercomputers. The authors used the Advanced Research WRF (WRF-ARW) to provide 24-hour-ahead forecasts every 12 hours for the interest points of the events of the Winter Olympics 2018.

WRF performs a domain-based distribution of the workload, using the Message Passing Interface (MPI) as the communication protocol, where the domain is the target region on Earth to be simulated. The domain is partitioned among the available MPI processes, distributed on the available processors. The processing complexity arises from two issues: (i) the simulation resolution, which governs the complexity of the integration step; and (ii) the grid/mesh dimensions, which determine the size of the subdomains.

Each of these questions is influenced by different properties of the underlying processing platform. For

example, the processing capacity of every processing unit greatly determines the quality and performance of integration step (i). As the resolution increases, i.e., the mesh subdomains become smaller, the integration step is shortened causing an increase in the required total integration steps.

This paper focuses, on the one hand, on the second issue (ii), which establishes the number of subdomains, each subdomain having a fixed number of cells. WRF evenly splits the general domain into n subdomains, n being the number of available MPI processes, which are arranged in a 2D grid $(x \times y)$.

The process distribution parameters x and y greatly determine the overall performance of the simulation depending on the dimensions of the subdomain mesh, i.e., the dimensions $(x \times y)$ of the domain grid. For example, with 25 MPI processes, the domain can be decomposed into three possibilities: (25×1), (5×5), and (1×25). It may be thought that distributing processes in a (25×1) layout performs similarly to distributing them in a (1×25) layout, but we show that this was not the case with WRF. In fact, there was an enormous difference in performance between the two layouts. For this reason, we studied the impact of the different process distributions on the simulation times and the reasons for this impact and proposed a new distribution algorithm that works better than the one implemented by WRF.

On the other hand, we also studied how increasing the number of used processing resources decreases the wall time of the simulation at the cost of losing efficiency for computing the same workload, dramatically increasing the energy consumption. In consequence, we found a balance between overall performance and energy consumption, which indicated the best process distribution when energy consumption was also a factor. Note that exascale platforms will be composed of thousands of processors, which means that a slight reduction in the energy consumption of each of them would substantially reduce the energy consumption of the platform as a whole.

The rest of this paper presents an overview of the current state of the art in Section 2. All the methods used in this work are detailed in Section 3. In Section 4, we study the effects of different process distributions on the overall performance and propose a new method to distribute them in Section 5. In Section 6, we study how an increase of the available MPI processes, as well as the variability of the processing capacity of each of these processes, affects the efficiency of the distributed computations and the energy consumption, thinking on an implementation of WRF for exascale. Finally, we present our conclusions and future work in Section 7.

2. Precedents and Related Work

WRF performance may be affected by different factors. Some are related to the software used in the compilation and execution phases (C and Fortran compiler, MPI library, and use of threads) and others to the configuration of a certain case study (physical model used, resolution requested, domain mesh, ...). In this paper, we will focus on three topics: how scalable WRF is when the number of available MPI processes increases; how the dimensions of the domain mesh affects performance; and how we can save energy while achieving good performance. In the literature, there are a number of papers addressing these topics.

Malakar et al. [1] focused on improving and analyzing the performance of nested domain simulations. They showed a significant reduction (up to 29%) in runtime via a combination of compiler optimizations, mapping of processes to physical topology, overlapping communication with computation, and parallel communications. They also concluded that high-resolution nested weather simulations are a challenge in terms of scaling to a large number of processors and considered it critical that practitioners choose a good nesting configuration.

Christidis [2] concluded that significant performance improvements, due to better cache utilization, can be obtained with a proper choice of the parameters nproc_x, nproc_y, and numtiles. Smaller contiguous arrays fit more efficiently in local caches, especially in the cases of "thin" decompositions (nproc_x < nproc_y) which allow computations with minimal cache misses.

In the same line, Johnsen et al. [3] investigated a "best fit" node placement scheme when using 2 OpenMP threads per MPI rank, 8 MPI ranks on each Cray XE6 "Blue Waters" node. By default, the XE6 job scheduler places MPI ranks in serial order on the machine, but halo exchange partners are not mapped this way in WRF. Using an alternate placement allows 3 communication partners to be obtained for most MPI ranks on the same node. At very high scales, this strategy improves overall WRF performance by 18% or more. The WRF grid is decomposed into rectangles with latitudes longer than longitudes for each subdomain. The optimized placement employed has the benefit of sending smaller east-west direction exchanges off-node and keeping as many larger north-south messages on-node as possible.

Shainer et al. [4] concluded that although interconnect type was the greatest determinant in improving WRF scalability, it was also observed that overall cluster productivity could be improved by up to 20% by running simultaneous jobs on the cluster rather than allocating the entire cluster to a single job. This increase in productivity was the result of two factors: (i) core and memory affinity, which reduces the remote memory access penalties and increases cache hits, and (ii) parallel jobs with smaller core counts help reduce the synchronization overhead for each application.

Kruse et al. [5] studied WRF scalability to several thousand cores on commodity supercomputers using Intel compilers and found that total time decreased between 512 and 2K cores and increased beyond 2K cores. While the computation time scaled well with increasing numbers of cores, the time to complete operations involving I/O increased, outweighing the gains in simulation speed at 2K cores and beyond.

For a very long time, computing performance was the only metric considered when launching a program. Scientists and users were only concerned about the time it took for a program to finish. Though still often true, the priority of

many hardware architects and system administrators has shifted to an increasing concern for energy consumption. High-performance computing consumes ever-larger volumes of electricity, and the reduction of consumption saves an appreciable amount of money.

One group of methods to reduce energy consumption focuses on how to distribute the workload among the cores of the computer. In this area, Lagravière et al. [6] compared the performance and power efficiency of Unified Parallel C (UPC), MPI, and OpenMP by running a set of kernels from the NAS Benchmark. They focused on the Partitioned Global Address Space (PGAS) model, and their main conclusion is that UPC can compete with MPI and OpenMP in terms of both computation speed and energy efficiency, but the data show that OpenMP consumes less energy than the others.

Igumenov and Žilinskas [7, 8] measured the power consumption of multicore computers with different computing loads: when the computer is idle and when some cores are fully loaded. The mean power consumption per core decreases when the computing load increases. Hence, running computers with greater loads is preferable to distributing parallel tasks among separate multicore computers.

Aqib and Fouz [9] compared the time and energy consumption of different tasks using different parallel programming models (OpenMP, OpenMPI, and CUDA). Their results, which can be generalized, outline the effect of choosing a programming model on the efficiency and energy consumption when running different codes on different machines. The parallel programming models obviously only improve the efficiency and reduce the energy consumption if there are blocks of codes that can be parallelized. Their conclusion is that OpenMPI performs much better than the other parallel models considered.

To summarize, the first works mentioned in this section seek to improve performance by modifying different parameters, both software and hardware, but unlike our work, in no case have they presented an exhaustive analysis that provides a heuristic to determine a distribution of processes close to optimal. In all these works, there is no concern for energy consumption. In the following works, different techniques are presented that allow reducing the consumption of energy for certain established test benches. Although they show the way forward, we have carried it out through software in a real situation.

3. Materials and Methods

3.1. Application Case. This work was conducted within the ICE-POP 2018 project, where the collaborating agencies were tasked with supporting the Winter Olympics by providing different kinds of weather information. A mid-resolution simulation with WRF has approximately 1 to 4 km of resolution per cell, with an integration step in the order of seconds. For the ICE-POP 2018, a resolution of about 300 m was required, which imposed integration steps lower than a second. Figure 1 shows the three nested domains with different resolutions that were computed in every simulation over the Korean peninsula. Different parameters from WRF forecasts, such as the visibility or the humidity,

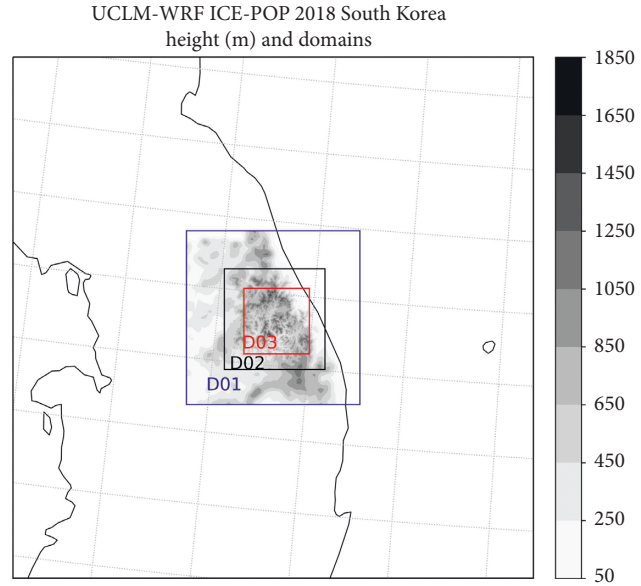

Figure 1: WRF domains used in our simulations over the Korean peninsula. Three nested domains D01, D02, and D03 with different resolutions. Terrain heights in meters are plotted inside the domains.

were extracted for the points of interest of the Olympics. This kind of information was extremely useful for planning the events. We performed the same kind of simulations over the same geographical region for all our studies. WRF-ARW version 3.9.1 with customized configuration (UCLM-WRF) was used by the authors to obtain the weather forecasts. WRF is open-source, so the source code can be obtained from the National Center of Atmospheric Research (NCAR) website (http://www2.mmm.ucar.edu/wrf/users/download/get_source.html). The configuration made use of state-of-the-art P3 microphysics [10], the Rapid Radiative Transfer Model scheme [11] for radiation, and the Noah Land Surface Model [12] for the surface.

3.2. Test Platform. We used the GALGO Supercomputer to perform all the tests in this work. GALGO is located at the Albacete Research Institute of Informatics, Spain, and hosts all kinds of scientific research. GALGO is a cluster of approximately 1200 processing cores, half of which are provided by *Intel Xeon E5450 3.0 GHz* processors. Each processing node is dual-socket, mounting two processors with shared DRAM and a 40 Gb/s dual-port *Mellanox ConnectX-2 Infiniband* interface. We used up to 40 of GALGO's processing nodes (320 processing cores) for our tests. The topology of the network is depicted in Figure 2; it consists of a first level of 24-port DDR switches and a second level of one 36-port QDR switch, with link rate of 20 Gb/s. Sixteen computing nodes are connected to each 24-port switch.

3.3. Compilation Options. The choice of compiler, compilation options, and MPI implementation has a big influence on the runtime of a simulation. In this work, we used the

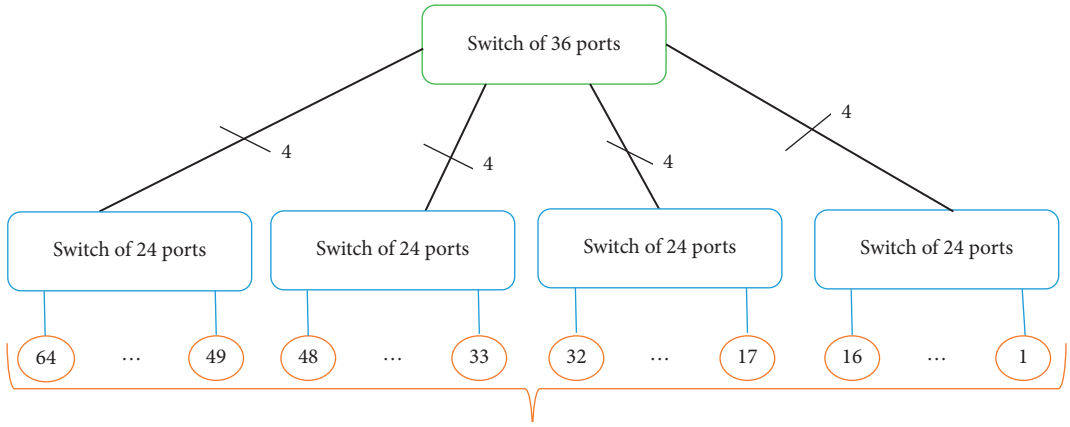

Figure 2: Topology of the network.

fastest binary codes, which are the best combination we can get, that is, the executable that allows us to run a simulation in the shortest possible time of all combinations. In our case, we used Intel compilers with Intel MPI libraries (ver. 2018.0.128) to compile all the required programs and dependencies. For best performance, we used the -O3 compilation option to activate the most aggressive optimizations available. We also used the -xHost compilation option to use the SIMD capabilities of the processor (SSE4). We empirically verified that the hybrid MPI-OpenMP (dm + sm) WRF compilation performed better than the other options and hence used this configuration in our tests.

3.4. Simulations. Our simulations covered the target region where the Winter Olympics were held, simulating 25 January 2018 from 06:00 to 06:30 for all the tests. A typical simulation for the Winter Olympics covered 24 simulated hours, but we used a reduced simulated time in our tests because of the large number of simulations these test involved. We used three one-way nested grids (see Figure 1) with sizes 199 × 199 for the inner grid (300 meters per cell), 103 × 103 for the intermediate grid (900 m), and 60 × 60 for the outer grid (2700 m), each of them with 70 vertical levels. Furthermore, the very high resolution of 300 meters per inner cell required substantial processing power per iteration. Numerical stability greatly depends on the resolution of the input geographical data, and therefore we used a high-resolution dataset of the Korean peninsula provided by the ICE-POP project instead of the default WRF Preprocessing System (WPS) Geographical Input Data.

3.5. Mesh Distribution. WRF performs an automatic distribution or layout of the simulation domain/grids among the available MPI processes, based on a Cartesian topology (MPI_Cart_create). As a result, it splits the domain into the most orthogonal coordinates possible $(x, y) \in \mathbb{N}$ or $(x \times y)$, assigning every coordinate to an MPI process. The x and y values can be overridden, so in order to check whether the most orthogonal layout is the best domain distribution, we performed additional simulations with all the possible $(x \times y)$ combinations for a set of n values, where n represents the number of processing nodes. In our experiments, the number of MPI processes matches the number of nodes n because we only used one MPI process per node. An easy way to change the x and y coordinates in WRF is through the edition of the namelist.input file needed to run WRF. In the namelist.input file, nproc_x controls the x coordinate and nproc_y controls the y coordinate.

3.6. Time Measurement. The basic measurement for our experiments is the average wall time, which is defined as

$$\mu_w = \sum_{i=1}^{z} \frac{W_i}{z}, \tag{1}$$

where W_i is the wall time for the i simulation and z the total number of iterations. Every test is composed of 10 iterations, i.e., executed 10 times, and then μ_w is obtained using (1).

3.7. Speedup and Efficiency. The average wall time μ_w is not appropriate to characterize how scalable parallel software is. For this reason, the analyses are usually done over the speedup (see Kumar et al. [13]) value:

$$S = \frac{\mu_{\text{ref}}}{\mu_{\text{faster}}}, \tag{2}$$

where μ_{ref} is the reference average wall time which depends on the study, e.g., for scalability, it is usually the time for the case with fewer processing units. μ_{faster} is the time of the case we are interested in analyzing. We can also measure the parallel efficiency E of a case as

$$E = \frac{S}{p}; \quad p = \frac{n_{\text{faster}}}{n_{\text{ref}}}, \tag{3}$$

where S is the speedup of the case, n_{faster} is the number of processes corresponding to μ_{faster}, n_{ref} is the number of processes corresponding to μ_{ref}, and p is how many times more processes the selected case has with respect to the reference case.

3.8. Energy Estimation. There is currently great interest in carrying out efficient implementations, both from the computational point of view (less execution time) and from the energy consumption perspective (less energy is needed). The power consumption of the entire platform is measured in Watts, and it is measured in Joules when the energy consumption is considered. In order to approximately estimate the power consumption, we have to know the amount of energy consumed in Watts by each processor. For the processors of the experimental platform, the vendor specifies this number as 80 W on average. In order to estimate the energy consumption in average from the average wall time μ_w, we defined

$$J = \mu_w \times n \times W_c, \tag{4}$$

where J represents the estimated Joules consumed by n nodes and W_c is the amount of Watts per processor according to vendor specifications. Therefore, W_c takes a value of 80 W when considering 2 or 4 cores (1 processor per node) and 160 W when considering 6 or 8 cores (two processors per node). We suppose that idle processors do not consume energy, which is not actually true, as seen in some studies such as Igumenov and Žilinskas [7]. We do this because we only have processors with the same model and a fixed number of cores, so we need to ascertain whether a processor with a reduced number of cores would be more energy efficient.

3.9. Experiment Setup. As a summary of the information shown in this section, the experiment setup is detailed below:

(i) Machines

(1) Up to 40 of GALGO's processing nodes with Intel Xeon E5450 3.0 GHz processors. Each processing node is dual-socket, mounting two processors with 32 GB DRAM and a 40 Gb/s dual-port Mellanox ConnectX-2 Infiniband interface.

(ii) Software

(1) Weather Research and Forecasting (WRF) version 3.9.1, compiled for hybrid MPI-OpenMP platforms (option "dm + sm" in WRF configuration).
(2) Intel compilers with Intel MPI libraries (version 2018.0.128). Compilation options used: -O3 and -xHost.

(iii) Methodology

(1) All WRF simulations were performed with the same parameters; the only change was the number of processes and the distribution of these processes over the nodes.
(2) We performed experiments using different distributions of $n = 9$, 16, 25, and 36 nodes. These values of n were used because they allow us to use an exact orthogonal distribution, that is, (3×3), (4×4), (5×5), and (6×6), which is the default in WRF.
(3) For a given number of nodes n, we considered all the different distributions of nodes in a 2D mesh.
(4) Average wall time was obtained for n processing nodes and 8 cores per node with different domain distributions. Every domain distribution was executed 10 times.
(5) Energy consumption is estimated from wall time and the amount of Watts per processor according to vendor specifications.

4. Analysis over WRF Process Distribution

All our WRF simulations were performed with the same parameters; the only change was the number of processes and the distribution of these processes over the processors. The WRF distribution algorithm assumes that the best layout for distributing the processes is the one that most preserves orthogonality, i.e., for n processes, the distribution is approximately $(\sqrt{n} \times \sqrt{n})$. We performed experiments using different distributions of $n = 9, 16, 25$, and 36 nodes. These values of n were used because they allow us to use an exact orthogonal distribution, that is, (3×3), (4×4), (5×5), and (6×6).

The question is do these orthogonal layouts provide the best times? In our experiments, we took into account all the different distributions of nodes in a 2D mesh with a fixed n. For instance, in the case of $n = 16$ nodes, the possibilities are (16×1), (8×2), the orthogonal (4×4), (2×8), and (1×16). In Table 1, we can see the layout effects on μ_w for $n = 9, 16, 25, 36$. The underlined values correspond to the automatic layouts used by WRF and the best values of μ_w are marked in bold.

The results in Table 1 show that even though the layouts chosen by WRF are usually "good enough," they are not the best. Therefore, based on the results, we state that the most orthogonal layouts are **not always** the best performing layouts. The results also corroborate the theory, that is, $(x \times y)$ is **not equal** to $(y \times x)$ in terms of performance due to the fact that different $(y \times x)$ process distribution involves different communication patterns.

4.1. WRF Communication Behavior. In order to explain the above finding, we looked at the communication behavior. It is known that MPI communications can be a bottleneck in programs with low computational workload and large number of communications among processes. We looked at two main factors that greatly influence the overhead introduced by MPI or any other message passing strategy: (i) the amount of data shared among the MPI processes and (ii) the number of transactions needed to share that amount of data. When these aforementioned factors exceed the capacity of the underlying platform, especially the interconnection network, the performance is greatly degraded.

Because the example of $n = 36$ has more distribution combinations than the others, we used this example to

Table 1: Average wall time μ_w when using n processing nodes and 8 processor cores with different domain distributions.

$(x \times y)$	$n = 9$	$(x \times y)$	$n = 16$	$(x \times y)$	$n = 25$	$(x \times y)$	$n = 36$
9×1	3443	16×1	2596	25×1	2117	36×1	—
3×3	*2403*	8×2	1714	5×5	**1017**	18×2	1152
1×9	**2327**	4×4	1458	1×25	1298	12×3	951
		2×8	**1396**			9×4	861
		1×16	1622			6×6	*824*
						4×9	**801**
						3×12	855
						2×18	908
						1×36	—

The automatic distributions chosen by WRF are underlined and the ones with the best μ_w are marked in bold. Some extreme distributions such as (36×1) crashed the simulations and could not be executed.

analyze the communications, using the statistics provided by the Intel MPI library. Therefore, we plotted the amount of data in MB injected into the intercommunication network (i) by the $n = 36$ distributions in Figure 3 and the number of transactions of every distribution (ii) in Figure 4.

Looking at the results, we can see that the more orthogonal combinations show the minimum amount of data transferred (i); however, they present a higher amount of transactions (ii). With WRF, we see that the amount of MB has a much greater impact on the wall time than the number of transactions. Moreover, if we suppose that communications are the most important factor for performance in our WRF simulations, $(x \times y)$ performance should be similar to $(y \times x)$ performance. If we look at Figure 5, we can see that the observed μ_w for every distribution of $n = 36$ is not the expected μ_w (approximation) in a situation where the communications are the most important factor.

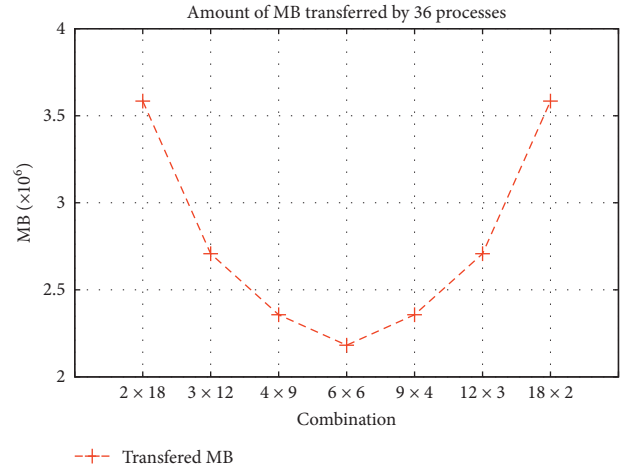

Figure 3: Total amount of data in MB injected into the interconnection network by the MPI processes. The distributions correspond to the case when $n = 36$.

4.2. WRF Integration Step Analysis. Consequently, communications were not the reason for the differences in performance between $(x \times y)$ and $(y \times x)$ distributions. As observed in Christidis [2] and Johnsen et al. [3], combinations where $x < y$ work better due to enhanced cache usage.

The WRF Fortran routine that is executed in every integration step is solve_em(), which is defined in solve_em.F source file. All the MPI and OpenMP functionality is used in this file. As previously stated, every subdomain of the $(x \times y)$ distribution is assigned to a MPI process. As an example, in Figure 6, we can see how WRF divided the inner domain $(199 \times 199 \times 70$ cells) of our simulations when using the (6×6) process distribution, where $x = 6$ and $y = 6$. For the sake of clarity, the cells of the domains are indexed by the i index (latitudes), the j index (longitudes), and the k index (vertical levels). This (6×6) distribution generated a subdomain for every MPI process with 33×33 cells ($i = 33$, $j = 33$, $k = 70$), each of those cells having 70 vertical levels (see the subdomain of case 6×6 in Figure 6).

When using OpenMP, every one of these subdomains is divided into tiles, which can be automatically handled by WRF (usually, tiles = threads) or manually set through the numtiles parameter in the namelist. The solve_em routine contains many loops over the target subdomain of every MPI process, and every loop iterates the corresponding tiles of the subdomain using OpenMP threads (OMP PARALLEL DO). A pseudocode of the WRF integration step (solve_em) is presented in Algorithm 1.

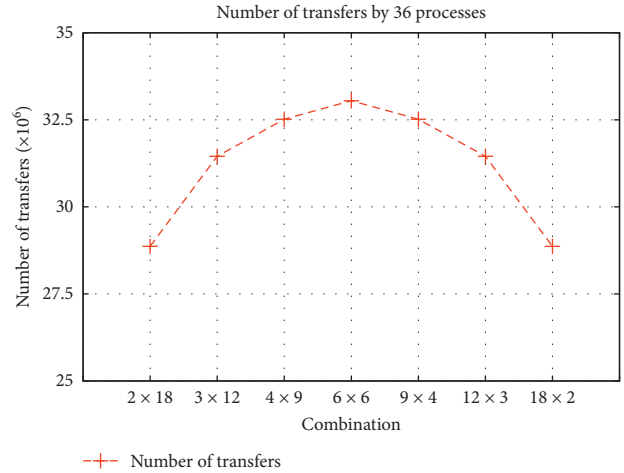

Figure 4: Total number of transactions performed by the MPI processes on the interconnection network. The distributions correspond to the case when $n = 36$.

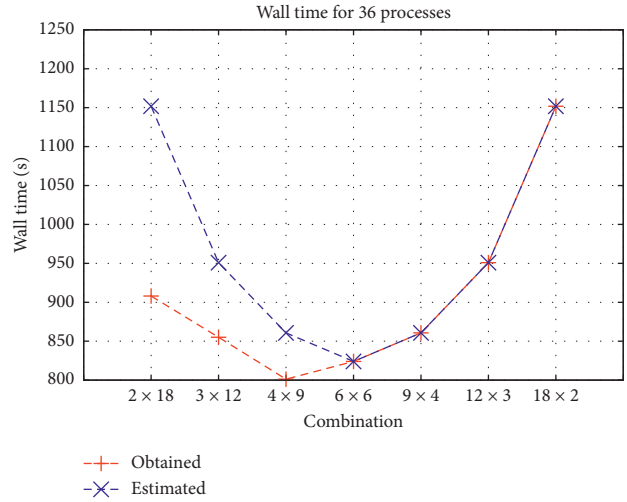

Figure 5: Execution time μ_w obtained on the tests and expected execution time for the different distributions of $n = 36$ processes.

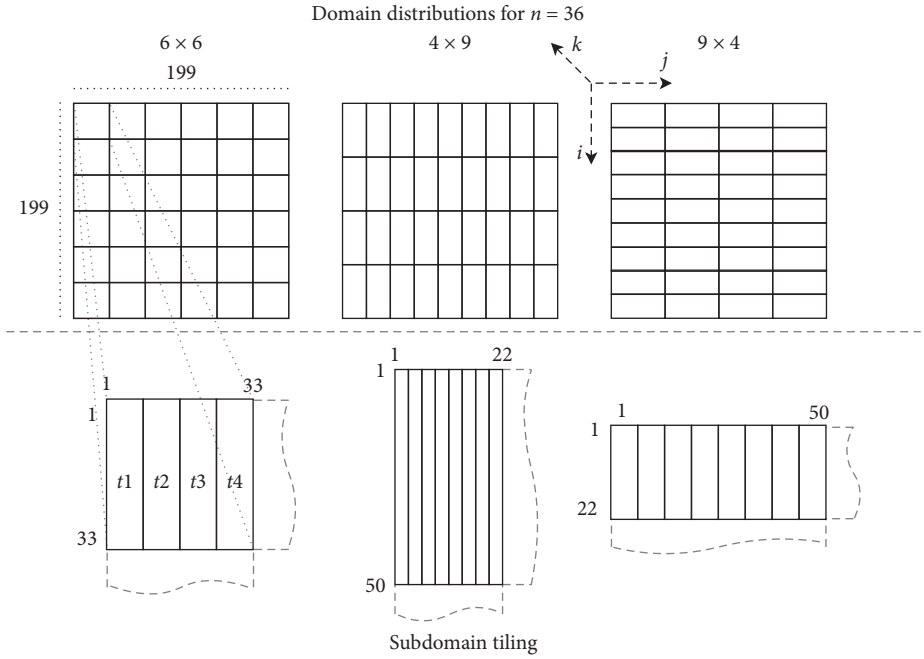

Figure 6: Process distributions over the inner domain when using $n = 36$ processes over the inner domain. The domains had 199×199 cells, each with 70 vertical levels. Every subdomain was divided in a number of tiles that was automatically chosen by WRF.

To be efficient with Fortran memory layout, every time the integration step computes anything over the target tile, it performs three nested loops in the right order (from inner to outer loop: i, k and then j). Thus, the problem lies elsewhere. In Figure 6, we can see the effects on the tiles when using three different distributions for $n = 36$. Remember that in terms of μ_w: $(4 \times 9) < (6 \times 6) < (9 \times 4)$, as we saw in Table 1. If we look at the three cases in Figure 6, the difference is the dimensions of i and j. When WRF computes every tile or slice of the corresponding subdomain, it performs better when the i dimension is large because of better cache usage.

In order to understand this effect, we need to look at how WRF maps the subdomains in memory. The i dimension is contiguous in memory, but not the k and j dimension because the subdomain is mapped on top of a larger memory layout where additional cells are allocated (e.g., the halos). These discontinuities in the three nested loops interfere with the performance of the cache when changing the k or j indexes, slowing down the computation.

Therefore, lowering the j dimension increases the i dimension, resulting in better cache usage and performance. Again, if we look at Figure 6, we can see that in the case of the (4×9) distribution, which is the fastest of the three, the i dimension is larger than in the other two cases, making it better performing than the others.

In the ideal case that the communications did not matter, the (1×36) distribution would be the fastest because a better cache usage ($i = 199$, $j = 5$ per subdomain). In

```
(1) function COMPUTE_SOMETHING(tile)
(2)   for j = 1 to tile_max_y do                    ▷ Iterate the tile
(3)     for k = 1 to max_vertical do                ▷ The iteration order is OK
(4)       for i = 1 to tile_max_x do
(5)         tile[i][k][j] = . . .                   ▷ Do something on the grid
(6)       end for
(7)     end for
(8)   end for
(9) end function
(10) procedure SOLVE_EM(subdomain)
(11)   . . .                                        ▷ Initialization
(12)   !$OMP PARALLEL DO                           ▷ One thread per tile
(13)   for t = 1 to numtiles do                     ▷ Iterate tiles
(14)     compute_something(subdomain[t])           ▷ Pass the tile to the function
       to compute something
(15)   end for
(16)   !$OMP END PARALLEL DO                       ▷ End of parallel region
(17)   if DM_PARALLEL then                         ▷ If MPI is being used
(18)     HALO_EM***.inc      ▷ Send halos to other adjacent processes
       depending on the stencil
(19)   end if
(20)   . . .   ▷ Replicate the same loop structure tens of times for the different
       variables to compute on the subdomain
(21) end procedure
```

Algorithm 1: Integration step (solve_em) pseudocode obtained from WRF Fortran code.

practice, this is not the case because in these extreme cases, communications escalate the computing times (see Figures 3 and 5 and Table 1). As a conclusion, a balance between the next two factors must be found when using WRF:

(1) Communications perform better when the process distribution is orthogonal

(2) Cache performance improves when larger values of i (latitudes) are used in the tiles

5. Improved Distribution Algorithm for WRF

Drawing on the results described in Section 4, we propose an alternative algorithm to distribute the layout of processes based on an α value. We observed that with a lower x dimension than y dimension (better cache usage), performance is much better when the workload of every process is sufficient to negate the communication overhead.

Thus, we devised a method to obtain a better distribution from an α ratio applied to n processes, balancing good cache usage with an acceptable increase in the communication overhead. In fact, the ratio between the values of x and y seemed to be the same, which we defined as

$$\alpha = \frac{x}{y}. \tag{5}$$

Depending on α chosen, we can obtain a good value for x that can be used to obtain an optimal $(x \times y)$ distribution of n MPI processes. In order to derive x, we have the system of equations composed by (5) and

$$xy = n, \tag{6}$$

$$y = \frac{n}{x}. \tag{7}$$

Solving (5)–(7) we obtain

$$\begin{aligned} \alpha &= \frac{x}{y} \\ &= \frac{x}{n} \\ &= \frac{x^2}{n}. \end{aligned} \tag{8}$$

Then, we clear the x from (7) and (8):

$$x = \sqrt{\alpha n}. \tag{9}$$

Equation (9) provides a way to obtain a good x value from a specific n. We also define $f(x, n)$ as a function which returns the divisor x' of n which is the nearest integer divisor to the value of x. Having a specific n, we can use (9) to obtain a near-optimal distribution $(x' \times y')$:

$$(x' \times y') = \left(f(x, n) \times \frac{n}{f(x, n)} \right). \tag{10}$$

5.1. Deriving the Optimal Alpha. Equation (9) needs an α value to be useful, and this value should minimize the overhead generated by the communication and cache misses. We have a domain with $l \times l$ latitude/longitude dimensions, and we subdivide the domain into $x \times y = n$

subdomains. From this, we obtain that the latitude/longitude dimensions of every subdomain are $((l/x) \times (l/y))$.

The communication overhead for every subdomain can be calculated as the length of the perimeter multiplied by the communication overhead A for every point at the perimeter. The communication overhead for the subdomain, using a specific x, is therefore defined by the following formula:

$$o_{\text{comm}}(x) = 2A\left(\frac{l}{x} + \frac{l}{y}\right) = 2A\frac{l}{x} + 2A\frac{lx}{n}. \quad (11)$$

Then, we derive (11) and equate to zero:

$$o'_{\text{comm}}(x) = -2A\frac{l}{x^2} + 2A\frac{1}{n} = 0 \longrightarrow x = \sqrt{n}. \quad (12)$$

The last equation proves that the communications overhead is minimum when $x = \sqrt{n}$, as we already saw in Figure 4.

Following the same line of thought, we derived the overhead of the cache misses. The cache miss overhead could be defined by the dimension of the longitudes (l/y) multiplied by the overhead B introduced for every cache miss. From this, we obtain the cache miss overhead for a specific x as

$$o_{\text{miss}}(x) = B\frac{l}{y} = B\frac{lx}{n}. \quad (13)$$

Equation (13) represents a monotonic increasing function whose minimum value is obtained for $x = 1$. This result supports the finding that the lower the y dimension is, the better the cache usage is. From the previous results, we can obtain the combined overhead $o(x)$ of (11) and (13):

$$o(x) = o_{\text{comm}} + o_{\text{miss}}(x) = 2A\frac{l}{x} + 2A\frac{lx}{n} + B\frac{lx}{n}. \quad (14)$$

After deriving (14) to obtain the x value where the combined overhead is minimum, we obtained:

$$o'(x) = -2A\frac{l}{x^2} + 2A\frac{l}{n} + B\frac{l}{n} = 0 \longrightarrow x = \sqrt{\frac{2An}{2A + B}}. \quad (15)$$

Therefore, from (9) and (15), the optimal α where the combined overhead is minimum is

$$x = \sqrt{\frac{2An}{2A + B}} \longrightarrow \alpha = \frac{2A}{2A + B}. \quad (16)$$

This result clearly shows the influence of both overheads (communication and cache misses) on the final computing performance.

5.2. Obtaining an Alpha Value. Note that A and B are unknown constants in equation (16), which impeded us from calculating the optimal α. On the other hand, we still needed to assign a value to α in order to obtain any distribution using (10). The problem is that it is not trivial to theoretically (or even empirically) calculate these constants and so we attempted another empirical approach to obtain an approximation of the optimal α.

For this purpose, we first defined α_n values for each of our training cases. From Table 1, we obtained the $(x_n \times y_n)$ distributions with the best μ_w of every n. We then used the x_n and y_n from these distributions to define α_n as

$$\alpha_n = \frac{x_n}{y_n}. \quad (17)$$

In the same way, we obtain the values α_n for the distributions presented in Johnsen et al. [3] (Table 2). Finally, we fit a $\sqrt{\alpha \cdot n}$ curve to these data (α_n values) and obtain as result $\alpha = 0.43$.

5.3. Near-Optimal Process Distribution. After applying (10) and $\alpha = 0.43$ to our values $n = 9$, $n = 16$, $n = 25$, and $n = 36$, we obtained the optimal distributions of (1×9), (2×8), (5×5), and (4×9), respectively. In these cases, WRF picked the orthogonal distributions, which were suboptimal. Our process distribution implementation is presented in Algorithm 2. Algorithm 2 admits two call parameters, the number of nodes considered (n) and the alpha value, and returns the value of x which is divisor of n and which is closer to the one calculated by the formula $x = \sqrt{\alpha n}$. To do this, function $F(x,n)$ first looks for values smaller than x (decreasing a unit in each step) until it finds a divisor of n and then repeats the process with values greater than x. Finally, both obtained values are compared and the one closest to the initial x is chosen. Finally y is calculated as $y = n/x$.

We were unable to perform simulations with n higher than 40 MPI processes in our platform, but we applied our distribution algorithm to the n values in Johnsen et al. The resulting x values from our algorithm (marked with ×) are shown in Figure 7 along with the x values used by Johnsen et al. (marked with +). The top solid line represents $x = \sqrt{n}$ (orthogonal), whereas the bottom solid line represents the x values obtained by (9). Using our algorithm achieves a smoother adjustment of x while increasing n.

To perform our tests with different distributions, we used the outputs of Algorithm 2 to change the value of the nproc_x and nproc_y parameters in the WRF namelist file. Nevertheless, the WRF distribution code can be modified to implement our algorithm without the need to externally modify (via automatic scripting or manual way) the namelist.

6. WRF Scalability and Energy Analysis

The improved distribution algorithm allowed us to boost in performance for our simulations, but we wanted to determine how this performance could be further increased. The second part of this work consisted of a study of the scalability of WRF and its efficiency when increasing the available processing power from the perspective of performance and energy consumption. Our target was to increase performance without significantly increasing the energy consumption.

Table 2: Rounded average wall times μ_w in seconds when using n processing nodes and c processor cores per node.

$c \backslash n$	10	20	30	40
8	2024	1182	896	774
6	2288	1333	964	812
4	3049	1733	1220	952
2	3895	2316	1612	1239

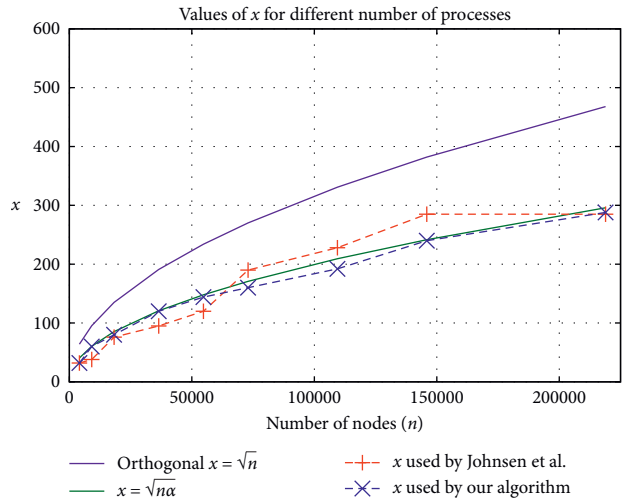

Figure 7: Final x values obtained from our distribution algorithm. The orthogonal $x = \sqrt{n}$ values (top solid line) and raw values from equation (9) (bottom solid line) are plotted. The x values obtained by our algorithm are very similar to those chosen by Johnsen et al.

```
Require: n ∈ N, α ∈ ℝ, n > 0 and α > 0
(1)  function F(x, n)          ▷ f(x, n) implementation
(2)    for i = 0 to n do       ▷Iterate possible decrements
(3)      x_f ⟵ x − i
(4)      r ⟵ n mod x_f
(5)      if r == 0 then
(6)        break ▷ Nearest divisor below x has been found
(7)      end if
(8)    end for
(9)    for i = 0 to n do ▷ Iterate possible increments
(10)     x_c ⟵ x + i
(11)     r ⟵ n mod x_c
(12)     if r == 0 then
(13)       break ▷ Nearest divisor over x has been found
(14)     end if
(15)   end for
(16)   if (x − x_f) < (x_c − x) then ▷ Return the nearest to x
(17)     return x_f
(18)   else
(19)     return x_c
(20)   end if
(21) end function
(22) procedure DISTRIBUTE(n, α) ▷ Distribution algorithm
(23)   x ⟵ √(α · n)          ▷ Get the first candidate
(24)   x ⟵ f(x, n)  ▷ Get the nearest divisor of n to x
(25)   y ⟵ n/x
(26)   Distribute processes using the (x × y) distribution
(27) end procedure
```

Algorithm 2: New WRF process distribution algorithm and $f(x', n)$ implementation.

6.1. Scalability. For our scalability study, we tested different cases using a variable number of computing nodes n and processing cores c from these nodes. After performing 10 simulations of every combination and applying (1), we obtained the μ_w values shown in Table 2. The same results are presented in Figure 8, where we can see the enormous differences in performance when n is low. In contrast, when n is high, the four cases shorten the distance between each other and converge to the same values of μ_w.

The processing cores c were handled by OpenMP threads, which provided good performance and a reduced memory footprint. After applying (2) and (3) to the values in Table 2, we obtained the efficiency values e, presented in Table 3.

From the information in Tables 2 and 3, we inferred the following observations:

(i) As expected, the average time per simulation is reduced when increasing the total number of processing units used $(n \times c)$

(ii) Decreasing the number of c cores per node and increasing the number of nodes n greatly increase efficiency e

(iii) When the total number of processing units $(n \times c)$ is greater than 180, efficiency e plummets

From the first observation, we see that increasing the number of nodes n reduces the wall times in all the cases when the number of c used per node is constant. In the case of using all the cores of every node, efficiency is greatly undermined when increasing the number of nodes. However, when the number of used cores per node is four or less, the efficiency keeps stable. This circumstance is clearly observed in Table 3.

One clear example to see the impact of using more nodes with limited cores is when comparing the cases that used 80 processing units: $(n = 10, c = 8)$, $(n = 20, c = 4)$, and $(n = 40, c = 2)$. The $(n = 20, c = 4)$ case is $\approx 14\%$ faster than the $(n = 10, c = 8)$ case, even at the expense of an increase in the MPI communications. Again, we see that communications are not the critical factor for performance when using WRF. The gap is even larger in the case of $(n = 40, c = 2)$ where only 25% (2 cores) of the nodes' processing capacity is being used, reducing the wall times by $\approx 38\%$.

After profiling the processor performance when executing $(n = 10, c = 8)$ and $(n = 40, c = 2)$ cases, we found that the number of minor memory page faults in the second case was $\approx 75\%$ lower than that in the first one. Additionally, the number of cache misses was reduced by 6% in the second case. We therefore conclude that the reason for these differences in performance is the better cache usage because of the reduced size of the data structure.

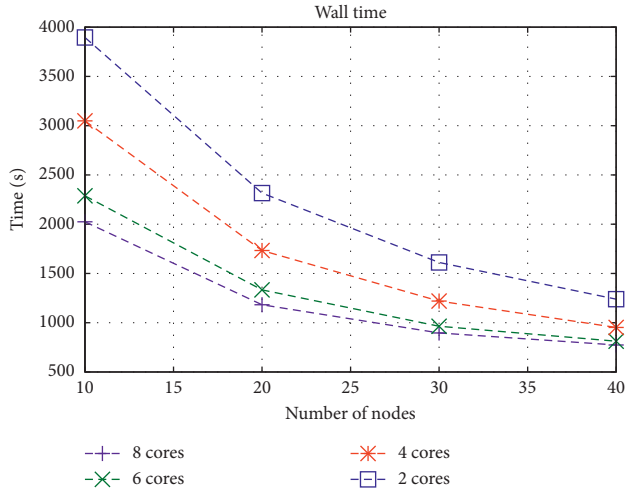

Figure 8: Execution time μ_w for different numbers of processing nodes n.

Table 3: Efficiency e for using n processing nodes and c processor cores, relative to the number of cores.

$c\backslash n$	10	20	30	40
8	1	0.86	0.75	0.65
6	1	0.86	0.79	0.70
4	1	0.88	0.83	0.80
2	1	0.84	0.81	0.79

When we combined the second observation with the conclusions of the first one, we saw that memory management greatly improves when the size of the data structures per node is low enough but also the number of processing cores c is not too high for that size. This is reflected in the efficiency e values obtained in Table 3 for the cases with less than 180 total processing units ($n \times c \leq 180$), which is also supported by the third observation. As a conclusion, we state that in order to maintain a high parallel efficiency, the size of the problem should be proportional to the used processing capacity. For the size of the problem in our simulations, 180 processing units are a good compromise between performance and efficiency. Increasing this number or the number of MPI processes n would divide the domains into small partitions that are too small and cannot be efficiently fed to the processors.

Looking at the previous observations, we conclude and recommend that WRF should be executed in computational resources that prioritize the size and latency of the cache memory more than the complexity and quantity of the computing cores. This recommendation is supported by the better efficiency observed in our simulations with the low values of c, which is in line with the results obtained in Shainer et al. [4]. Moreover, the available processing power should be appropriate to the size of the problem (resolution and domain dimensions), being neither too high nor too low.

Table 4: Estimated energy consumption J for using n processing nodes and c processor cores.

$c\backslash n$	10	20	30	40
8	3238400	3782400	4300800	4953600
6	3660800	4265600	4627200	5196800
4	2439200	2772800	2928000	3046400
2	3116000	3705600	3868800	3964800

6.2. Energy Efficiency. As we stated in the introduction section, when dealing with exascale platforms, a small reduction on performance could become a large reduction on energy consumption. This is why it is so important to balance the performance of the whole system when considering its energy consumption.

There is a limit at which increasing the number of computational resources barely improves performance, at the cost of skyrocketing energy consumption. Therefore, energy consumption is a factor when deciding how many computational resources are needed to satisfy the established requirements. Table 4 shows the estimated energy consumption J after applying (4) to the μ_w times presented in Table 2. The estimated energy consumption J is also presented in Figure 9, where we can see that the greatest energy savings corresponding to the use of 4 cores.

In consequence, it may be thought possible to find a good balance between the time spent on an execution and energy consumption. Evidently, if time is the most critical variable in an experiment, the energy consumption becomes irrelevant. For instance, if the simulations of this paper must be executed in less than 800 seconds, the only feasible configuration corresponds to 8 cores and 40 nodes. But, if the time to execute the simulation is represented by a limit, e.g., 1000 seconds, then we can play with other variables such as the energy consumption. We scatter plotted all the combinations in Figure 10 so we could choose a good combination for the last example. With 1000 seconds as a limit and looking at Figure 10, we have 4 options: $(n = 40, c = 4)$, $(n = 30, c = 8)$, $(n = 30, c = 6)$, and $(n = 40, c = 8)$. In this case, the best option is clearly $(n = 40, c = 4)$ because of the enormous difference in Joules between this and the other options.

In other cases, the best option is not so clear and depends on the priorities. For example, with a limit of 1900 seconds, the reader might agree with us that of all the possibilities, only three are feasible: $(n = 20, c = 4)$, $(n = 30, c = 4)$, and $(n = 40, c = 4)$. The $(n = 20, c = 4)$ case is the least energy consuming but much slower than the other two. $(n = 40, c = 4)$ consumes a little more than $(n = 20, c = 4)$ but, in contrast, is twice as fast. $(n = 30, c = 4)$ is a compromise between both. Depending on our priorities, we would choose

(i) $(n = 40, c = 4)$ for maximizing performance

(ii) $(n = 20, c = 4)$ for maximizing energy savings

(iii) $(n = 30, c = 4)$ if the priority is to harmonize performance and energy consumption

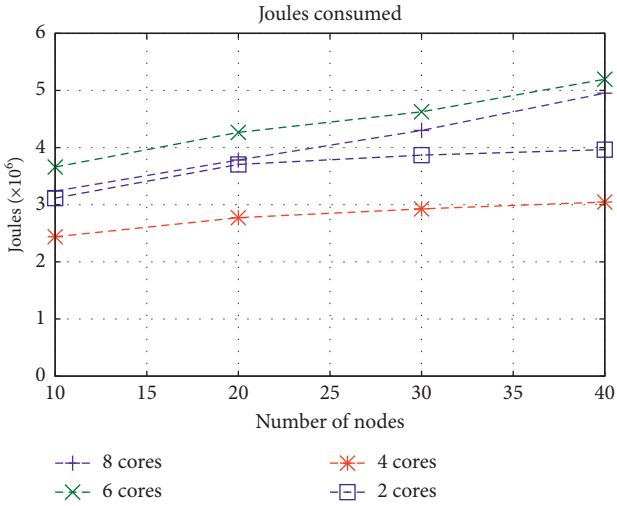

Figure 9: Estimated energy consumption J.

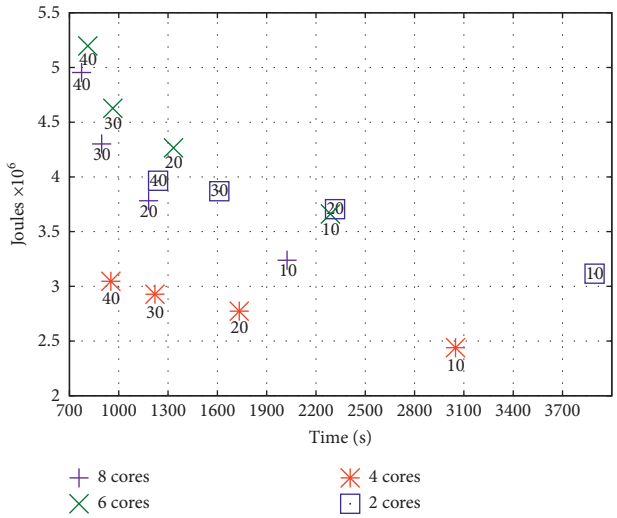

Figure 10: Scatter plot of time (ordinates) and energy consumption (abscissas) for the different combinations of n processes (below the points) and c cores.

In our simulations, we needed the fastest distribution possible, which was the $(n = 40, c = 8)$ distribution. The other option we considered was the $(n = 40, c = 4)$ distribution, which obtained a 38% energy saving with an increase of 23% in the wall time. The problem with this distribution was that the wall times were far from our target time requirements and therefore not a feasible choice.

7. Conclusions

This paper proposes a new distribution algorithm that works better than the one implemented by WRF. This algorithm was devised from the results of a performance study over different process distributions with a different number of total processes.

We also present a study of the performance of the WRF-ARW model in terms of three main variables: process distribution, execution time, and energy consumption. The authors had to comply with the execution time requirements imposed by the ICE-POP 2018 project, with regard to 2018 Winter Olympics held in Pyeongchang, but also taking into account the appropriate use of resources and energy consumption imposed by the authors' research institution. However, this study is essential for any research group working in this area, so this paper could be considered as a guideline.

Beyond this guideline, the following main contributions emerge from this work:

(i) Orthogonal distributions are optimal for communications and interprocessor performance.

(ii) Latitude dominant dimensions are optimal for cache usage and intraprocessor performance.

(iii) WRF performance therefore depends on the balance between orthogonality (communications) and efficient cache usage (longer latitudes per subdomain). We proposed an algorithm to obtain a balanced distribution.

(iv) In our platform, the WRF model works better when using less cores per node.

(v) WRF consumes less energy using less nodes, achieving a good execution time.

(vi) To execute WRF, it is thus recommended to use simple machines (not so many cores and more energy efficient than complex ones) with quality cache memories.

Related to the grid distribution used in the WRF software package, the authors evidence that for the platform used in this experiment, the best process distribution is not always the orthogonal one. To address this issue, the authors proposed a new distribution algorithm to calculate a better distribution layout than the default one implemented by WRF. As a future work, the authors propose to corroborate the results of this paper in other platforms completely different to that used in this work, especially platforms where the number of processes could be higher than 10,000. In this work, we obtained a good α value from near-optimal distributions used in other works, but we expect to find a better α value from the best process distributions when n is high enough. We also proved that the combined overhead of the communications and cache misses follows a relationship between two constants that we propose to determine in future works. These two constants would allow us to obtain the optimal process distribution for any number of MPI processes.

In order to test our algorithm in our simulations, we externally modified the WRF namelist's nproc_x and nproc_y parameters in our simulations, but the algorithm is easily implementable as an alternative distribution option inside WRF source code.

From the study of WRF source code, we think it is possible to optimize the data locality and hence remove the limitation of having latitude dominant dimensions to achieve the best performance. This could be achieved by modifying the details of the tiling processing code.

Furthermore, and from the previous contributions, we explored different ways of saving energy by changing the process locations. The authors used a graphical method to obtain the best configuration by plotting energy and time together. Depending on the priorities (performance or energy savings), different options could be chosen from this plot. In addition, in this work, we are not considering the use of accelerator due to the fact that the use of GPUs on WRF software is reduced to a short set of functions. Our purpose in this paper was to study the influence of process distribution both in terms of performance and energy consumption without comparing against other WRF implementations that consider GPUs. This comparison (with WRF and without using GPUs) could be interesting as a future work.

Data Availability

The data used to support the findings of this study are included within the article.

Disclosure

An earlier version of this manuscript was presented as a part of RM's PhD dissertation "Some critical HPC improvements in numerical weather prediction workflows."

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Spanish Ministry of the Economy, Competitiveness, Science and Innovation (MINECO) (grant nos. CGL2013-48367-P and CGL2016-80609-R) and the KMA (Korea Meteorological Administration) (grant no. 1365002970/KMA2018-00721). RM acknowledges support from MINECO (grant no. FPI BES-2014-069430) for conducting his PhD. AN acknowledges support from MINECO (grant no. FPU 13/02798) for carrying out his PhD.

References

[1] P. Malakar, V. Saxena, T. George et al., "Performance evaluation and optimization of nested high resolution weather simulations," in *Euro-Par 2012 Parallel Processing*, C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds., pp. 805–817, Springer, Berlin, Germany, 2012.

[2] Z. Christidis, "Performance and scaling of WRF on three different parallel supercomputers," in *High Performance Computing*, J. M. Kunkel and T. Ludwig, Eds., pp. 514–528, Springer, Berlin, Germany, 2015.

[3] P. Johnsen, M. Straka, M. Shapiro, A. Norton, and T. Galarneau, "Petascale WRF simulation of hurricane sandy: deployment of NCSA's cray XE6 blue waters," in *Proceedings of the 2013 SC-International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–7, Denver, CO, USA, November 2013.

[4] G. Shainer, T. Liu, J. Michalakes et al., "Weather research and forecast (WRF) model performance and profiling analysis on advanced multi-core HPC clusters," in *Proceedings of the 10th LCI International Conference on High-Performance Clustered Computing*, Boulder, CO, USA, 2009.

[5] C. Kruse, D. Del Vento, R. Montuoro, M. Lubin, and S. McMillan, "Evaluation of WRF scaling to several thousand cores on the yellowstone supercomputer," in *Proceedings of the Front Range Consortium for Research Computing 2013*, Boulder, CO, USA, August 2013.

[6] J. Lagravière, P. H. Ha, and X. Cai, "Evaluation of the power efficiency of UPC, OpenMP and MPI. Is PGAS ready for the challenge of energy efficiency? A study with the NAS benchmark," Tech. Rep., The Arctic University of Norway, Tromsø, Norway, 2015.

[7] A. Igumenov and J. Žilinskas, "Electrical energy aware parallel computing with MPI and CUDA," in *Proceedings of the 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp. 531–536, IEEE, Compiegne, France, October 2013.

[8] A. Igumenov and J. Žilinskas, "Power consumption optimization with parallel computing," *Jaunųjų Mokslininkų Darbai*, vol. 4, pp. 119–122, 2011.

[9] M. Aqib and F. F. Fouz, "The effect of parallel programming languages on the performance and energy consumption of HPC applications," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, 2016.

[10] H. Morrison and J. A. Milbrandt, "Parameterization of cloud microphysics based on the prediction of bulk ice particle properties—part I: scheme description and idealized tests," *Journal of the Atmospheric Sciences*, vol. 72, no. 1, pp. 287–311, 2015.

[11] E. J. Mlawer, S. J. Taubman, P. D. Brown, M. J. Iacono, and S. A. Clough, "Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave," *Journal of Geophysical Research: Atmospheres*, vol. 102, no. D14, pp. 16663–16682, 1997.

[12] G.-Y. Niu, Z.-L. Yang, K. E. Mitchell et al., "The community Noah land surface model with multiparameterization options (Noah-MP): 1—model description and evaluation with local-scale measurements," *Journal of Geophysical Research: Atmospheres*, vol. 116, no. D12, 2011.

[13] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Vol. 400, Benjamin Cummings, San Francisco, CA, USA, 1994.