Exploring Opportunities for Performance Improvement in a Global Climate Model

Rhuan E. C. Costa[1], **Celso L. Mendes**[1]

[1]Laboratory for Computing and Applied Mathematics
National Institute for Space Research (INPE)
Sao Jose dos Campos – SP – Brazil

{rhuan.costa,celso.mendes}@inpe.br

Abstract. *The Brazilian Earth System Model (BESM) is a Global Climate Model (GCM) developed by the Brazilian National Institute for Space Research (INPE). The main purpose of a GCM is to simulate Earth's climate in a decadal or centennial scale. The simulations usually include representations of the main elements of the Earth, such as atmosphere, ocean, ice and land. Since its first release, BESM has provided support materials for contributions to the Intergovernmental Panel on Climate Change (IPCC).*
This paper evaluates BESM's performance and explores optimization possibilities, aiming to speed up the model execution. Our study started with a detailed analysis that characterized the performance of BESM executions on hundreds of processors, which served to reveal the major performance bottlenecks. Next, we worked on schemes to mitigate some of those bottlenecks. The changes made so far resulted on performance gains up to a factor of 4 in some cases, when compared to the way it was previously being executed in production. We also describe ongoing work towards additional performance improvements. Despite presenting results only for BESM, our optimization techniques are applicable to other scientific, multi-physics models as well.

1. Introduction

The main tool to study future climate changes is a Global Climate Model (GCM), also known as Earth System Model (ESM). In order to represent the climate variability, GCMs incorporate the complexity of various components of the Earth, such as ocean, atmosphere, and land surface [Chou et al. 2014].

Various climate centers around the world are developing their own GCMs, either by adapting an existing GCM, or combining ocean, atmosphere and surface models, from different sources. The last one was the option chosen by Brazil's Center for Weather Forecasting and Climate Studies (CPTEC, a division of INPE) when creating the Brazilian Earth System Model (BESM) [Nobre et al. 2013].

The implementation of GCMs requires the representation of various components of the planet (atmosphere, ocean, land, etc). Each component involves multiple elements (e.g. temperature, speed, pressure, among others) that interact with each other. Thus, the implementation of GCMs is very complex, usually developed by combining isolated models that represent each component, known as modules. These modules are then coupled to represent the whole Earth system. Each module is very likely developed by a different team, specialized on that component modelling. So, an intrinsic characteristic is that each

implementation can target a different approach of code optimization and parallelization. Thus, making the modules to work in harmony, in an efficient way, with minimum waste of computational resources, turns out to be a complex task. It involves a deep analysis of how each module behaves during the execution, and the identification of parallelization techniques that are efficient for all modules.

In this paper, we present a study that aims to optimize the computing performance of a particular GCM that is currently used in production by Brazilian climate researchers. These optimizations include alternative ways to build and run the model, to more efficiently explore the parallelism available on the execution platforms. Despite showing results only for this specific GCM, our techniques can be employed by other GCMs that contain multiple physical modules as well. Our results indicate that these techniques enable concrete performance gains to be achieved, without changing the physical modelling originally implemented by the GCM developers.

The remainder of this paper is organized as follows. In Section 2, we describe the GCM that is the object of our study: the Brazilian Earth System Model (BESM). Section 3 shows the characteristics of BESM's performance. In Section 4, we present our performance improvements made to BESM. Section 5 concludes our presentation.

2. BESM Climate Model

The Brazilian Earth System Model (BESM) is an effort from various institutes and researchers, led by the National Institute for Space Research (INPE), through the Brazilian Center for Weather Forecasting and Climate Studies (CPTEC). The purpose is to build a multidisciplinary research framework aiming to understand the causes of global climate changes, its effects and social impacts [Nobre et al. 2013].

The first major version of the model was called Brazilian Earth System Model version 2.3 (BESM-OA 2.3), and it was an evolution of previous versions of CPTEC's ocean-atmosphere coupled model. BESM was developed following the criteria to participate on the 5th phase of the Coupled Model Intercomparison Project (CMIP5) [Taylor et al. 2012]. The objective was to simulate the behavior of the ocean-atmosphere coupled system, in decadal time scale, under variation of the greenhouse gases concentration on the atmosphere [Nobre et al. 2013].

CMIP5 participants provided data to the Intergovernmental Panel on Climate Change (IPCC) and parts of the dataset were used on the Fifth Assessment Report (AR5) of the IPCC. The dataset was analyzed by the scientific community around the world, and is still available for climate studies [Emori et al. 2016].

The current BESM version, 2.6, comprises the Atmospheric Global Circulation Model (AGCM) [Nobre et al. 2013] from CPTEC/INPE, coupled to the Modular Ocean Model version 5 (MOM5) [Griffies and Zadeh 2019] from Geophysical Fluid Dynamics Laboratory (GFDL). The coupling between those two models is made by the Flexible Modular System (FMS), also from GFDL.

An ice model, called Sea Ice Simulator (SIS), is included in MOM5. SIS acts as an interface between the atmospheric and oceanic models, being responsible for calculating the fluxes ocean/ice and ice/atmosphere, and for communicating ocean/atmosphere fluxes on regions where there is no real ice [Casagrande et al. 2016].

Strictly speaking, BESM is not considered a complete GCM yet. To achieve this, there are efforts to add to the model (i) the atmospheric chemistry and aerosols model, ECHAN - HAMMOZ from Max-Planck Institut [Hammoz 2018], and (ii) the surface vegetation model, IBIS, from Univ. of Wisconsin [Univ. of Wisconsin-Madison 2018] [Miguel and Monteiro 2015]. There is also a plan for improvement of the model through inclusion of new parameterization of atmospheric convection that better represents the precipitation over South America [Nobre et al. 2016].

There is currently great motivation to optimize BESM's performance, in particular regarding its parallel scaling properties. On a presentation made by the BESM development team [Kubota 2015], the need to optimize the model to enable simulations with more refined resolution was highlighted. They showed tests made on CPTEC's supercomputer where the execution made with 30,000 CPUs had only 34% performance improvement over the execution with 8,400 CPUs. This means an efficiency of only 37% between these two cases, indicating an expressive waste of computational resources.

3. Characterization of BESM Performance

3.1. Original BESM Implementation

BESM is prepared to use the Message Passing Interface (MPI) as its major parallelization method. This approach launches various tasks that are managed at the operating system level. Each task is responsible for processing a fraction of the problem, typically corresponding to a sub-region of the domain being represented in the model.

A key execution entity of the BESM model is the FMS coupler. It initializes the individual physical modules, manages their execution sequence and the information sharing among them. The FMS implements an interface for parallelization called MPP. Its purpose is to abstract parallelization-related tasks, such as synchronization and communicator selection. The MPI calls are implemented underneath the MPP.

Each module (i.e. ocean or atmosphere) executes its own timesteps by an arbitrary number of times, updating its internal state. After the specified number of internal timesteps, the modules update each other by sharing their internal state through the coupler. This execution point where the information is exchanged among the modules is called coupling timestep.

3.2. BESM Execution Platform

One of the platforms used to run BESM is the Santos Dumont system [LNCC 2019], from the Brazilian National Laboratory of Scientific Computing (LNCC). That system has a total processing capacity of 1,1 Petaflops, constituted of 18,144 CPU cores, distributed across 756 compute nodes (24 cores per node - 2x Intel Xeon E5-2695v2), with 64GB of DDR3 RAM each. Our tests and improvements were all conducted on Santos Dumont. All runs had one CPU core dedicated to each MPI task.

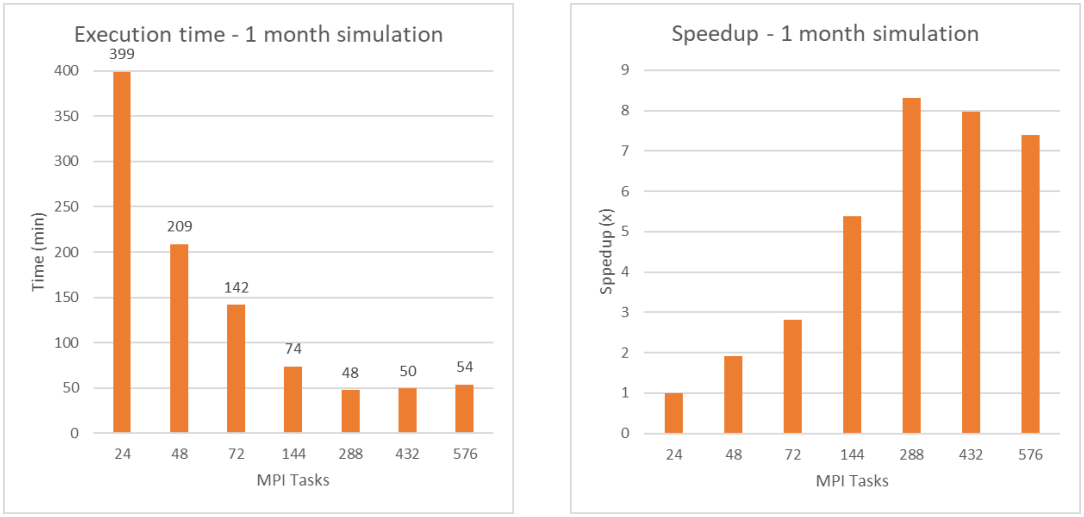
3.3. Original BESM Performance

As an initial study object for our work, CPTEC provided the source code of BESM v2.6. That package also included input files to initialize the model and scripts for compiling and running the model on the Santos Dumont environment. All tests were performed based on

one-month long simulations, with resolution T062L28 ($1,8758^0$ of latitude and longitude - 200 km at Equator line, and 28 vertical layers) for the atmosphere component (AGCM); and 1^0 of longitude, latitudinal resolution from 0.25^0 on the tropical region to 2^0 on the hemispheres, and 50 vertical layers for the ocean component (MOM).

3.3.1. Scalability

Our first analysis was on how BESM was being built, including the employed compiler, its version and optimization flags. The model was originally compiled with GCC 5.3.0 and OpenMPI 2.0.1. The only optimization flag set by the compilation scripts was `-O2`. We compiled the model using all the default values that came on the scripts, and measured the time spent to run one-month long simulations for different numbers of CPUs allocated to execute the model. The results of this experiment are shown on Figures 1a and 1b.



(a) Execution time **(b)** Speedup

Figure 1. Original BESM's performance for one-month simulation

Figure 1a shows the time (in minutes) spent to run the simulation for a varying number of MPI tasks. Starting with 399 minutes to simulate one month of global climate using 24 MPI tasks, the minimum execution time was reached with 288 tasks (48 minutes for the same type of simulation). After this point, when we increased the number of tasks, the execution time started to rise, reaching 50 minutes for 432 tasks and 54 minutes for 576 tasks. This behavior indicates that the BESM model, in this case, did not take advantage of more than 300 CPUs, and any extra hardware allocated to run it would be a waste of computational resources. This also marks the speedup limit of the model, considering the way it was originally implemented and run. Figure 1b shows the speedup related to each number of MPI tasks. On this graph, it is easier to see the speedup peak at 288 MPI tasks. This measurement took the run with 24 tasks as reference, representing a speedup of 1.0. These measurements revealed that BESM was considerably limited in terms of scalability,

After this point, we proceeded the subsequent analysis using the Intel compiler. In addition to better application performance, the Intel compiler also offers better integration with the Intel PSXE tools for performance analysis. To use the Intel compiler, however, we had to make changes to the build process, as explained in Section IV.

3.3.2. OpenMP Parallelization

After selecting compilation flags appropriate for the Intel compiler, we analyzed how the model was being parallelized. We found that MPI was the only parallelization method being effectively employed in production executions. Although there were OpenMP directives in the AGCM module, they were only enabled by CPTEC when running the isolated model, not coupled to other modules. After fixing compiling errors when enabling OpenMP, we conducted performance measurements.

In these tests, the same number of CPUs was used on cases with and without OpenMP. However, with OpenMP enabled, a fraction of the CPUs was reserved to run only OpenMP threads; meanwhile, with pure MPI, each CPU had an MPI task assigned to it. For example, an execution with 72 MPI tasks and 2 OpenMP threads per task (MPI + OMP) was compared to an execution with 144 MPI tasks (pure-MPI). In this way, we can see which configuration is more efficient for a given computational resource (i.e. number of CPUs). As Table 1 shows, the execution times are better when allocating all CPUs to run MPI tasks, instead of reserving a fraction of them for OpenMP threads.

Table 1. Execution time comparison: MPI+OpenMP vs Pure-MPI

MPI+OpenMP			Pure-MPI	
Tasks	**Threads/Task**	**Time (minutes)**	**Tasks**	**Time (minutes)**
72	1	46	72	46
72	2	40	144	28
72	4	37	288	28

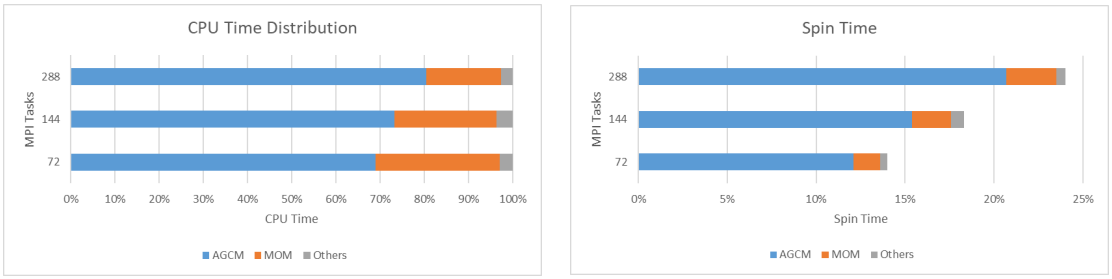
Based on the results of these tests and some code investigation, we concluded that only a small portion of the code was prepared to take advantage of OpenMP threads. As a result, it was not worth to reserve CPUs to run only threads. Instead, we obtained better performance disabling OpenMP and running only MPI tasks on all CPUs.

3.3.3. Hotspot Analysis

The next performance analysis made was to check for execution hotspots, which means to check which portions of the code take more CPU time. Ideally, optimizing those regions would have a bigger impact on the overall performance of the program. The goal of this analysis was to identify the reasons for the poor speedup that we had observed.

The first verification was the amount of CPU time used by each module (AGCM/MOM) and their variation according to the number of MPI tasks. On our tests, we found that the AGCM module uses more CPU time than the MOM module. Less than 5% of the total CPU time is spent by functions that do not belong to any of these two modules. Figure 2a shows the behavior of BESM in relation to the number of MPI tasks. The proportion of CPU time taken by AGCM increases proportionally to the number of tasks, while the MOM's share decreases. This indicates that while MOM can scale well with the addition of more tasks, AGCM has deficiencies in this regard.

Before diving into a function-level hotspot checking, we faced an alarming situation. Intel VTune Amplifier, the tool we used for this phase of the analysis, was reporting a high percentage of spin time. In our tests, the total spin time of the model increased proportionally to the number of tasks, as shown in Figure 2b. An expressive part of this spin time comes from the AGCM module, up to 8 times more than MOM's share. For



(a) CPU time distribution

(b) Spin time

Figure 2. Analysis of Intel-compiled BESM by module

instance, on the test with 288 tasks, AGCM wasted 20.7% of the total CPU time, while MOM spent 2.8% on spin time. This is a worrying situation, since almost a quarter of the total model's CPU time is being wasted when running with 288 tasks, and its tendency is to rise while the number of tasks increases.

Knowing that mitigating the spin time could improve BESM's scalability, we explored the reasons of this wasted time. We found that a single MPI call was causing almost the totality of the spin time in the AGCM module. That was an `MPI_Allreduce()` call, which makes an operation over an array distributed across multiple tasks, and sends the result to all participating tasks. So, for the function to complete, there is a need of synchronization of the participating tasks. Other collective MPI calls distributed across the code had a less significant contribution to spin time.

In addition to the spin time from `MPI_Allreduce()`, there was also up to 13% CPU time (varying with the number of tasks) attributed to it. In order to check if the spin time and CPU time from the function were intrinsic characteristics of the distributed operation, or if they were caused by desynchronization of the tasks, we added a `MPI_Barrier()` before the `MPI_Allreduce()` call. Due to the presence of that barrier, when `MPI_Allreduce()` is called all tasks will already be synchronized.

After this modification, all the CPU time and spin time were transferred to the barrier, meaning that those are caused exclusively by the desynchronization of the MPI tasks. This behavior characterizes a possible load imbalance among the AGCM's MPI tasks. Load imbalance is typical in atmospheric models [Rodrigues et al. 2010], and is frequently a major factor limiting scalability of those models. While total removal of this imbalance may require deep restructuring of the model, and is beyond the scope of this paper, we present in the next sections certain techniques that at least mitigate its effects.

4. Performance Improvements

The performance improvements achieved so far are presented in this section: modifications and optimizations related to the compiler, and strategies to execute AGCM and MOM modules concurrently.

4.1. Compiler-Related Optimization

The first task conducted to optimize BESM performance was to identify compile-time optimizations. In order to get better compile-time optimizations and integration with performance analysis tools, we created new compiling scripts to use the Intel PSXE 2019

suite (available on Santos Dumont). That suite includes Fortran and C compilers, an Intel MPI library that implements the MPI-3.1 specification, and tools for performance analysis. In addition to the replacement of the MPI library, we added optimization flags aiming to get a more computationally efficient program. The flags added were:

- `-O3`: enables more aggressive automatic optimizations;
- `-no-prec-div`: increases calculation speed in divisions;
- `-fp-mode fast=2`: enables more aggressive optimizations on floating-point operations;
- `-xHost`: enables vectorization instructions optimized for the target CPU.

Those changes resulted on a performance improvement of up to 3.5x, according to the number of CPUs used to run the model. Figure 3 shows the execution time before and after the compiler changes, for different numbers of MPI Tasks. The best execution time changed from 48 minutes with GCC to 28 minutes after our modifications. On the other hand, the maximum speedup point moved from 288 tasks to 144 tasks. Overall, we had better execution times on all test cases.

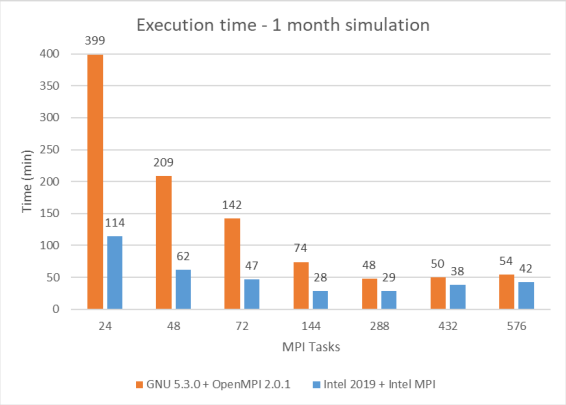


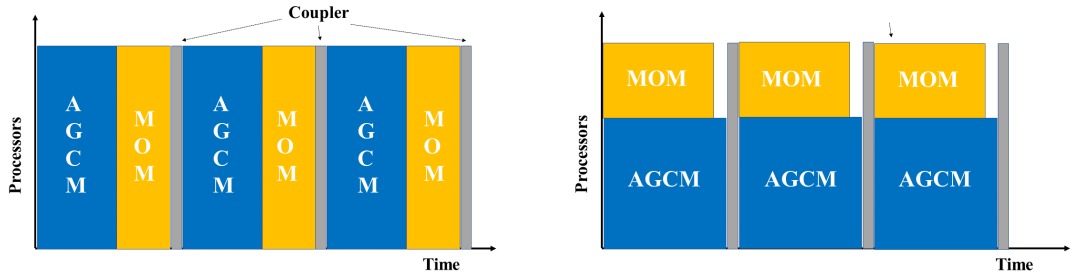
Figure 3. Execution time comparison after compiler and MPI library changes

4.2. Concurrent-Module Execution

One of our main goals was to explore the performance implications of executing the AGCM and MOM modules concurrently, rather than one after the other (i.e. in "alternate mode") as currently done at CPTEC. Figure 4a shows a functioning schematic of the alternate mode, where AGCM executes its internal timesteps on all MPI tasks. Then, MOM executes its internal timesteps, also using all MPI tasks. After both modules execute a set of timesteps, a coupling timestep is reached, when the modules exchange information.

With concurrent mode (Figure 4b), a portion of the MPI tasks is dedicated to run each module. In this way, AGCM is executed on a set of tasks, and at the same time, MOM is executed on another set. After both modules finish their execution, the coupling timestep is reached. One drawback found on this execution mode is that the faster module will have its tasks waiting the coupling timestep until the slowest module finishes its execution. This may result on a waste of computational resources, but it can be mitigated by properly balancing the number of MPI tasks assigned to each module.

The FMS coupler is designed to give support to concurrent execution of the GCM modules. MOM, which came integrated to FMS, is also prepared to run concurrently. On



(a) Alternate mode **(b)** Concurrent mode

Figure 4. Execution mode schematics

the other hand, AGCM was not originally implemented aiming to be part of a coupled model, and even less to run in a concurrent way with other modules. Thus, we had to modify the AGCM code to make it work with the concurrency method available on FMS.

While MOM uses an interface for parallelization called MPP, AGCM makes MPI calls directly. Thus, among other minor changes, the main modification we made was the creation of a dedicated MPI subcommunicator to replace the global communicator used on AGCM. This required less effort than changing the whole AGCM to use MPP.

The AGCM code used the `MPI_COMM_WORLD` communicator in its MPI calls. This means that its collective functions expected participation of all MPI tasks of the program. When running in "alternate mode", this was not a problem, since on AGCM's turn to run it had participation of all tasks. But when running concurrently, MOM's tasks did not reach AGCM's functions; then, the MPI collective calls in AGCM were stuck waiting for all executing tasks to reach that MPI call, which never happened. We solved this problem creating a subcommunicator of `MPI_COMM_WORLD`, called `COMM_ATMOS`. This subcommunicator contains only the tasks belonging to AGCM. We also made the surrounding code changes for the new communicator to work properly. Since this subcommunicator is created on the FMS code (where all the tasks are available), we had to modify a set of functions to pass its reference up to the points where it is used in AGCM.

After solving this MPI communicator problem and configuring the input files with the appropriate number of tasks to be employed by each module in a concurrent execution, we were able to run the model in concurrent mode and checked if the output files contained results similar to the ones produced by the original model. The results showed differences of up to 5%, similar to the variation faced when we change the number of tasks used to run the original model. With the changes validated, we proceeded to the performance analysis of the model running in concurrent mode.

4.2.1. Even Distribution of CPUs

The first tests we executed were 1-month simulations using half of the MPI tasks for each module (AGCM and MOM). In an ideal situation, one might expect to move the speedup limit to the double of CPUs. This is because, hypothetically, with each module having its saturation at the number P of CPUs, by using $2P$ CPUs (P for each module) both modules would still be at their saturation points. A second ideal-world expectation might be that a concurrent run with $2P$ tasks (P for each module) would take half the time of an alternate run with P tasks. This was only an "ideal scenario", since we know (from the

Hotspot subsection) that the modules do not use equal computational resources.

The concurrent mode with half CPUs for each module did not bring the ideal results. Figure 5 shows the alternate and concurrent execution times for different numbers of MPI tasks. Although we pushed the speedup limit from 144 to 288 MPI tasks and obtained better execution times when running concurrently, those execution times did not improve significantly. We had a maximal time reduction of 26%, for the run with 432 tasks; on the other hand, for 144 tasks the execution time did not change.

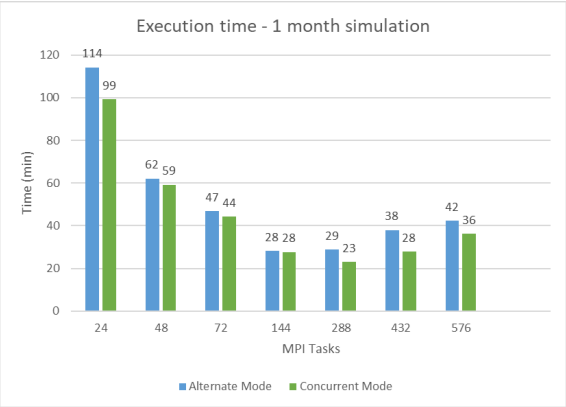


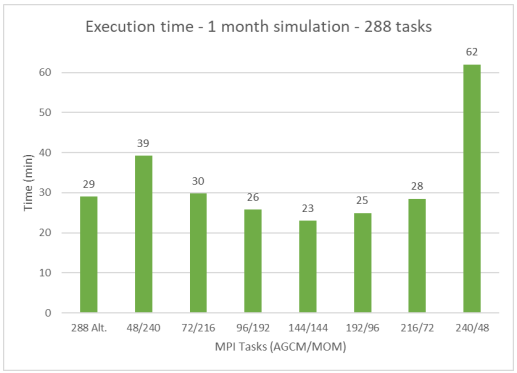
Figure 5. Execution time comparison: alternate vs concurrent mode

4.2.2. CPU Balancing

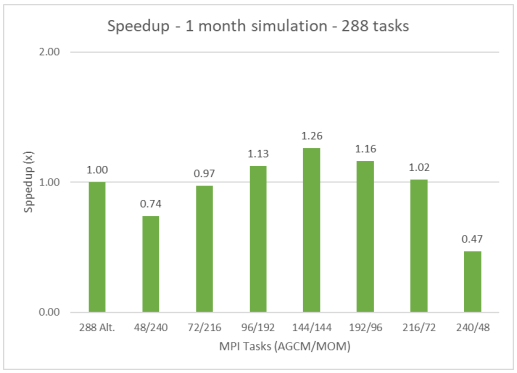
Given that the modules do not consume the same amount of computational resources on an "alternate-mode" run, our next approach was to test different proportions of CPU distribution, to study the effect on performance. We know that AGCM takes more CPU time than MOM on "alternate-mode" runs. Thus, our initial expectation was that assigning more CPUs to AGCM would make it run faster, while MOM, running concurrently with fewer CPUs, could finish its timesteps execution nearly at the same time as AGCM. In this way, both modules might reach the coupling timestep almost simultaneously.

We started by taking the fastest execution mode we had until this point: the run using 288 MPI tasks. We verified the execution times of a set of proportions of MPI tasks for AGCM/MOM, given the same total of 288 tasks. Figure 6a shows the execution times for the distributions that we tested. The best proportion of task distribution we obtained (in terms of execution time) was the 144/144 share. By increasing or decreasing the AGCM share of tasks, the execution time rose. The first bar in Figure 6a shows, for comparison, the execution time of the corresponding alternate-mode execution, with the same total 288 MPI tasks. On Figure 6b, which shows the corresponding speedups, it is easier to see the peak at the 144/144 distribution. For the most uneven distributions (e.g. 48/240 or 240/48), the performance was worse when compared to the alternate mode.

Our next tests aimed to check if the model could effectively scale to more than 288 CPUs, taking the maximum number of CPUs we used so far (576) and varying the distribution proportions. Figure 7a shows the execution times of different distributions of the 576 MPI tasks; the first bar shows the time spent by the "alternate-mode" run, for comparison. Figure 7b illustrates the speedup over the "alternate-mode" run. Opposite to what our intuition led us to expect, assigning more CPUs to MOM resulted on better

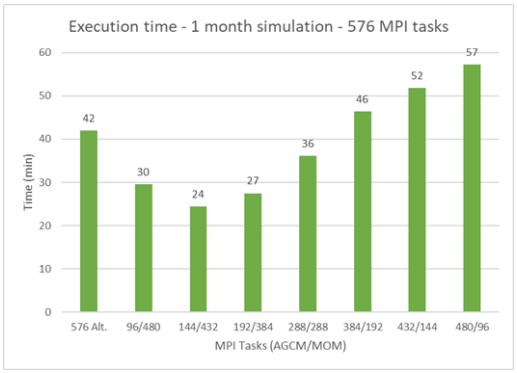


(a) Execution time

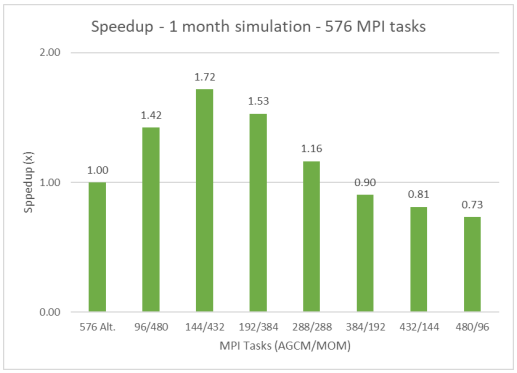


(b) Speedup

Figure 6. Performance on different distributions of 288 MPI tasks



(a) Execution time



(b) Speedup

Figure 7. Performance on different distributions of 576 MPI tasks

performance. What we had not taken into consideration yet was that the AGCM's spin time increases proportionally to the number of MPI tasks in use. On the other hand, MOM shows better scalability with the addition of more CPUs.

We see that performance was lost on all runs of Figures 7a and 7b where the major share of CPUs was assigned to AGCM (i.e. three right-most bars in the figures), while the execution with 75% of the CPUs assigned to MOM (144/432 CPUs for AGCM/MOM, respectively) had a speedup of 1.72. Even with a balanced distribution of the 576 tasks, we did not beat the execution time of the 288 tasks with 50%/50% distribution. We got 24 minutes for 144/432 distribution, and 23 minutes for the 144/144 distribution.

4.3. Overall Results

Even though this work is still in progress, we already have improved BESM's performance by a promising margin. Compared to the way the model was running in production, we obtained a 403% execution time improvement when running with 24 CPUs (from 399 to 99 minutes for a one-month simulation); and reduced the minimum simulation time from 48 to 23 minutes, a 209% enhancement, by properly building and executing the BESM components. Figure 8 shows the execution times from the original model compared to the ones we obtained after our modifications. For 576 MPI tasks, we considered the best result obtained after CPU balancing, explained in the previous subsection.

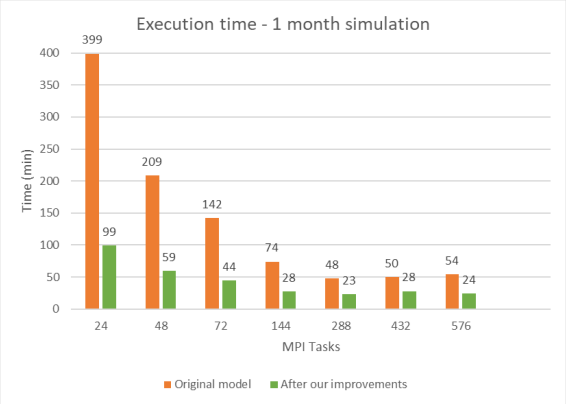


Figure 8. Execution time comparison after all performed changes

5. Conclusion

This paper presented our efforts to optimize the computing efficiency of BESM, a GCM developed by CPTEC/INPE. BESM has shown its relevance by providing material for academic researches and to IPCC reports, which address ongoing concerns related to Global Warming. We started with an analysis of BESM's initial situation. Next, we worked to improve the model performance, based on what we found on the initial analysis.

We have obtained good results by changing the compiler and tuning up compilation options, and by making the necessary changes in the code to enable running the atmospheric and oceanic modules concurrently. Comparing to the way the model was being used in production, we improved the model's execution time by 403% when running with fewer CPUs (24 MPI tasks) and reduced the minimum execution time of BESM from 48 to 23 minutes, a 209% improvement, when running with 288 CPUs. After each relevant modification in the model's code or in execution mode, we validated it by comparing the results of a one-month simulation with the output produced by the original model.

For tests with the new concurrent mode and 576 CPUs, we had better execution times when MOM received the bigger share of the CPUs. This is due to the poor scalability of AGCM, which suffers from an increasing spin time when more MPI tasks are added. Thus, more CPU time is wasted when AGCM has more CPUs than MOM. According to our tests, AGCM cannot get any performance gain with more than 144 CPUs.

Although we have improved BESM's performance by a considerable margin, there is still room for further improvements. We plan to run more tests with different proportions and numbers of MPI tasks, to seek a pattern of CPU distribution among the AGCM and MOM modules that results on better execution times when BESM is run with a higher spatial resolution. These weak-scaling tests, employing spatial resolutions that are still relevant to climate researchers, could alleviate the scaling limits observed in AGCM.

The investigation of the root cause of the observed spin time revealed that load imbalance was the main factor leading to poor scalability of AGCM. Due to this bottleneck, we could not obtain any performance benefits from using more than 288 MPI tasks in BESM. Mitigating this effect is a challenge that remains open. Addressing this load imbalance problem, however, will probably require restructuring of the code of the atmospheric model, which was beyond the scope of this paper.

References

Casagrande, F., Nobre, P., and Souza, R. B. et al. (2016). Arctic sea ice: Decadal simulations and future scenarios using BESM-OA. *Atmospheric and Climate Sciences*, 06(02):351–366.

Chou, S. C., Lyra, A., and Mourão, C et al. (2014). Evaluation of the eta simulations nested in three global climate models. *American Journal of Climate Change*, 03(05):438–454.

Emori, S., Taylor, K., Hewitson, B., Zermoglio, F., Juckes, M., Lautenschlager, M., and Stockhause, M. (2016). Cmip5 data provided at the ipcc data distribution centre.

Griffies, S. and Zadeh, N. (2019). The mom user guide. https://github.com/mom-ocean/MOM5/blob/master/doc/web/user_guide.md.

Hammoz (2018). Echam-hammoz - aerosol & atmospheric chemistry modules. https://redmine.hammoz.ethz.ch/projects/hammoz.

Kubota, P. Y. (2015). Visão geral e estado do desenvolvimento mcga-cptec/inpe. In *Posters*. Workshop BESM 2015: Workshop Anual do Projeto do Modelo Brasileiro do Sistema Terrestre BESM/FAPESP/INCT-MC/Rede CLIMA,.

LNCC (2019). Sdumont - sistema de computação petaflópica do sinapad. http://sdumont.lncc.br/.

Miguel, J. C. H. and Monteiro, M. (2015). Mudanças climáticas, tecnociência e geopolítica: Um modelo do sistema terrestre brasileiro e a soberania na produção de futuros climáticos. In *Seminários Temáticos*, volume 2, page 30. V Reunião de Antropologia da Ciência e da Tecnologia.

Nobre, P., Capistrano, V. B., and Baptista Junior, M. et al. (2016). Modelo brasileiro do sistema terrestre (BESM) para cenários de mudanças climáticas globais. In *Modelagem Climática e Vulnerabilidades Setoriais à Mudança do Clima no Brasil*, volume 1, pages 33–48. MCTI, Brasilia.

Nobre, P., Siqueira, L. S. P., and Almeida, R. A. F. et al. (2013). Climate simulation and change in the brazilian climate model. *Journal of Climate*, 26(17):6716–6732.

Rodrigues, E. R., Navaux, P. O. A., Panetta, J., Fazenda, A., Mendes, C. L., and Kale, L. V. (2010). A comparative analysis of load balancing algorithms applied to a weather forecast model. In *2010 22nd International Symposium on Computer Architecture and High Performance Computing*, pages 71–78.

Taylor, K. E., Stouffer, R. J., and Meehl, G. A. (2012). An overview of cmip5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485–498.

Univ. of Wisconsin-Madison (2018). Ibis - integrated biosphere simulator. https://nelson.wisc.edu/sage/data-and-models/lba/ibis.php.