The role of mini-apps in weather and climate models performance optimization

Giovanni Aloisio[*,§], Jean-Claude André[‡], Italo Epicoco[*,§], Silvia Mocavero[§]
(*) University of Salento, Lecce, Italy
(§) Euro-Mediterranean Centre on Climate Change – CMCC, Italy
(‡) Jca Consultance & Analyse, Toulouse, France

Introduction

Today, modeling the climate system requires the simulation of an extremely large number of interacting and complex processes taking place in various fluids (atmosphere, ocean) and in various physical systems (soil, ice caps and ice sheets, vegetation layers, …), as well as their analysis at different time and spatial scales. Modeling the atmosphere for weather prediction purposes may require fewer efforts on the detailed simulation of other-than-atmospheric subsystems, but requires instead a very large effort for determining the initial state, usually done through data assimilation techniques.

Advanced numerical models are used to solve the inherently non-linear governing equations, while huge computational resources are needed to (i) calculate billions of individual variables describing the physical processes at different scales and to (ii) perform the optimization steps necessary for assimilating the huge amount of available observational data. Efforts in a number of different domains are then needed to analyze and understand the computational performance of weather and climate applications and to design new numerical approaches at exascale. The development of mini-applications appears to be a viable solution to introduce new numerical methods, new algorithms and to drive the hardware development following a co-design approach.

Computational issues

Today, one of the main issues for extreme computing is the exploitation of the new emerging architectures to execute legacy applications, sometimes designed and developed over a decade or more. They have to be re-shaped and/or re-designed at different levels, starting from the new physical phenomena to be modeled at high resolution, through the choice of new alternative mathematical (*e.g.*, ensemble data assimilation techniques), numerical (*e.g.*, grids, parallelization in time) and algorithmic methods, up to a co-design process between technology providers and computational scientists. The starting point of the process is the analysis of both the state-of-the-art and the main scalability bottlenecks of the stand-alone subsystems on the one hand and of the complete, coupled models on the other hand.

Firstly, modelling of the atmosphere and the ocean is based on the Navier-Stokes equations, solved either by finite difference methods in the case of ocean models, or very often by spectral methods in the case of atmospheric models, executed in parallel due to spatial domain decomposition. The new architectures allow increasing the spatial resolution, but one key problem is related to the sequential nature of the time-stepping algorithm (*cf.* CFL condition), even with perfect weak scaling. Some parallelization-in-time techniques have been developed in order to overcome this limit. Moreover, the parallelization is often performed by message-passing, which on massive parallel architectures leads to a very low ratio between computational and communication loads. New parallel programming languages allow to better exploit the shared memory and then to reduce the communication weight. Also at algorithmic level, the efforts aim at the development of new communication-avoiding and communication/computation-overlapping algorithms.

Secondly, the improvement of both spatial and time resolutions causes the increasing of the data read/written by the application, making the I/O a bottleneck for the scalability and requiring consequently new parallelized instead of more classical sequential methods, not to mention the fact that the management of a huge quantity of data produced by a single simulation could be a problem on its own.

Finally, in most actual weather prediction models, the data assimilation schemes are based on a minimization of

the distance between the model trajectory and the already observed data during a time interval which is of the order of 12h (the so-called assimilation window). During this window the minimization is performed by backward integration of the adjoint model, a technique that is difficult to parallelize due to the fact that the adjoint model cannot be integrated at full resolution, asking then for strong scaling. In the future, knowing the sensitivity of the prediction quality upon the length of the assimilation window, this problem will become more and more crucial, with assimilation windows increasing to 24h or even up to 36 or 48h. Alternative mathematical methods with expected much better scaling properties, like ensemble Kalman filters, have to be carefully designed and possibly implemented.

Mini-applications for weather and climate

Although benchmarking applications is a key part of the design and implementation on new platforms, it is strongly limited by the complexity of the complete software code. On the other hand the characteristics that impact on performance should be deeply understood to better evaluate the performance portability on different architectures and drive the design of new computers. To address these issues, two important properties of many weather or climate models can be considered: (i) although a model may have one million or more source lines of code, its performance is often dominated by a small subset of lines included in a single or few computational kernels; (ii) the remaining part of the code simulate distinct physics phenomena but have similar performance characteristics.

Mini-apps, as lighter versions of complex applications identifying their bottlenecks and their computational characteristics, can be used to guide the flexible software development, allowing designing complex applications, taking into account how they will interact with future computing architectures. They are also highly valuable to system designers and architects as long as they represent the behavior of the complete workload. There are two ways to build mini-apps: (i) to start from the real application and applying the needed changes or (ii) to write a new code from scratch, adopting disruptive approaches to break specific barriers to extreme scale and construct the mini-app around it. Both approaches, evolutionary *vs.* revolutionary are valuable but not interchangeable.

Corresponding to the three earlier-mentioned computational issues, examples of such mini-apps could be defined as follows:
(i) Solver mini-app. In the very-high-resolution NEMO oceanic model, one of the most computationally intensive kernels is represented by the tracer advection tendency term that represents the divergence of the advective fluxes, requiring the use of the continuity equation. Performance of the discretization schemes of this advection term are driven by the choice made in the space and time interpolation to define the value of the tracer at the velocity points. The mini-app built around this kernel allows to evaluate the performance portability on several computing platforms and estimate the computational performance achievable by hybrid parallel approaches, as well as to evaluate how new parallel architectures, based on accelerators (GPU, MIC), can improve the code scalability;
(ii) I/O mimi-app. Flexible methods for the management of I/O are necessary for, e.g., either minimizing the call of subroutines related to I/O definition (file creation, axis and dimensions management, adding and output field…) or minimizing arguments of I/O call. Such methods should not be invasive but allow high performance targeted for massive parallel simulation (10 000 or more cores): writing data must not slow down the computation, by, *e.g.*, using one or more "server" processes dedicated exclusively to the I/O management;
(iii) Data assimilation mini-app. The data assimilation method used in the oceanic NEMO model is treated as a separate part of the code, referred to as the NEMOVAR software. It can be used to test either 4-dimensional assimilation method, where the adjoint code is integrated backward at a resolution lower than the direct code (the so-called incremental method), or ensemble methods based upon the Kalman filter theory, or even hybrid method.

More detailed information about these mini-apps will be provided.