Unveiling WARIS code, a parallel and multi-purpose FDM framework

Raúl de la Cruz[1], Mauricio Hanzich[1],
Arnau Folch[1], Guillaume Houzeaux[1] and José Maria Cela[1]

CASE Department, Barcelona Supercomputing Center, Spain,
{delacruz, mauricio.hanzich, arnau.folch, guillaume.houzeaux, josem.cela}@bsc.es

Abstract. WARIS is an in-house multi-purpose framework focused on solving scientific problems using Finite Difference Methods as numerical scheme. Its framework was designed from scratch to solve in a parallel and efficient way Earth Science and Computational Fluid Dynamic problems on a wide variety of architectures. WARIS uses structured meshes to discretize the problem domains, as these are better suited for optimization in accelerator-based architectures. To succeed in such challenge, WARIS framework was initially designed to be modular in order to ease development cycles, portability, reusability and future extensions of the framework. In order to assess its performance, a code that solves the vectorial Advection-Diffusion-Sedimentation equation has been ported to the WARIS framework. This problem appears in many geophysical applications, including atmospheric transport of passive substances. As an application example, we focus on atmospheric dispersion of volcanic ash, a case in which operational code performance is critical given the threat posed by this substance on aircraft engines. Preliminary results are very promising, performance has been improved by 8.2× with respect to the baseline code using a realistic case. This opens new perspectives for operational setups, including efficient ensemble forecast.

1 Introduction

Many relevant problems arising in geoscience and Computational Fluid Dynamics (CFD) can be solved numerically using Finite Difference Methods (FDM) on structured computational meshes. Examples include, seismic wave propagation, numerical weather prediction or atmospheric transport. FDM numerical schemes on structured meshes allow peak performances of $\approx 20-30\%$, about 3 times larger than analogous FE (Finite Element) methods.

WARIS is a brand-new multi-purpose framework aimed at solving efficiently in a parallel way this kind of scientific computing problems. The design requirements were to obtain a portable framework (*i.e.* able to run on any hardware platform) suited for accelerated-based architectures, with reusable software components, easily extendible, and able to solve the physical problems on structured meshes explicitly, implicitly or semi-implicitly.

The next sections are organized as follows: Section 2 describes the procedure followed during the development stages of WARIS, and introduces the current state-of-the-art regarding FDM optimizations as well. Section 3 elaborates on the final design chosen for the WARIS framework, detailing also its internals. Finally, Section 4 and 5 expose a case of study and the conclusions respectively.

2 Software Engineering

Software engineering plays an important role when achieving a good software design. The desirable aspects of a software package are reliability, efficiency, and robustness; bestowing accurate results, high performance and solid components respectively. Meeting all those requirements lead to a successful project. In order to succeed in such a challenge, an application development life-cycle methodology must be applied [10]. Besides, life-cycle methodology is particularly important when HPC software comes into play. This importance is due to several inherent needs of the HPC software.

Firstly, given that the numerical software is clearly expensive to develop due to the involved research process, it is intended to reuse code as much as possible. A modular and flexible design of the software framework may help in this approach owing to many important reasons. Software must be flexible and modular whether same code is wanted to be used for different scientific problems and likewise, be adapted to a novel hardware architecture or a different programming model. In this regard, the HPC is far more dynamic in terms of adaptability requirements than the rest of the computational areas, and therefore these reasons have a greater influence. Secondly, the HPC involves large numerical simulations which may require hundreds or thousands of computational nodes during days or weeks. Unsuccessful executions (abnormal end) might arise from time to time, leading to unfruitful runs and a waste of time and resources. Hence, the cost of these executions is utterly high in terms of power consumption and resource usage.

Finally, a modular and robust infrastructure is crucial, but performing the numerical simulations in an efficient way is also a key point. Additionally, considering that each simulation run can last several days or weeks, even a mild optimization in performance of only 5 to 10% may lead to days of savings in core-hours of computational resources. In order to do so, unlike ordinary development life-cycle, an additional stage for the optimization process of the modules is also included. This stage is repeatedly performed in order to successively optimize the performance of each module.

2.1 Boosting Numerical Codes

Irregular codes (stencil computations and sparse algebra) are usually limited by memory access (memory bound). Therefore, the ratio of floating-point operations to memory access is low compared with regular codes (FFT and dense linear algebra), which are mainly compute bound. In explicit FDM schemes, the basic structure of stencil computation is that the central point accumulates the contribution of neighbor points in every axis of the Cartesian system. The number of neighbor points in every axis relates to the accuracy level of the stencil, where more neighbor points lead to higher accuracy. The stencil computation is then repeated for every point in the computational domain, thus solving the spatial differential operator.

Two inherent problems can be identified from the structure of the stencil computation. Firstly, the non-contiguous memory access pattern. In order to compute the central point of the stencil, a set of neighbors has to be accessed. Some of these neighbor points are distant in the memory hierarchy, requiring many cycles in latencies to be accessed [5]. Secondly, the low computational-intensity and reuse ratios. After gathering the set of data points, just one central point is computed and only the accessed data points in the sweep direction might be reused for the computation of the next central point [4].

The state-of-the-art in performance optimizations for stencil computation is very prolific. The contributions can be divided into three dissimilar groups: space blocking, time blocking and pipeline optimizations. Space blocking algorithms promote data reuse by traversing data in a specific order. Space blocking is especially useful when the dataset structure does not fit into the memory hierarchy [12,5]. Time blocking algorithms [8] perform loop unrolling over time-step sweeps to exploit the grid points as much as possible, and thus increase data reuse. Finally, low level optimizations at the CPU pipeline include several well-known techniques. These techniques may be categorized into loop transformations [9], data access [2] and streaming optimizations (SMP, SIMD and MIMD).

3 System Architecture

As the number of physical problems that should be supported could be in the order of tenths, the primary system (named Physical Simulator Kernel, PSK) should be flexible enough to accommodate new problems reusing as much code as possible.The PSK framework is divided in two main sets of components. On one hand, there is a *framework* responsible for those tasks that are common to any physical simulation being solved, such as domain decomposition, communications and I/O operations. On the other hand there is a set of *specializations* that are used to configure the framework in order to have a complete solution for a given physical problem. Those specializations depend on aspects such as: the physical problem, the hardware platform (e.g. general purpose, GPU, FPGA, Xeon Phi) and the programming model being used for development. As the PSK main goal is to generalize the way a physical simulation is built, some aspects have to be fixed and restricted, in order to bound the framework functionality and limits.

3.1 Hardware Architecture Model

The first aspect to be considered is the computational architecture model that will be supported by the PSK. Figure 1 shows the concepts in such model and their relation. The main building block of the architecture model is the *Computational Node* (CN), which is built using both the *host* and *device* elements that communicate through a *Common Address Space* (CAS) memory. Examples of such devices include GPUs [15] and brand-new Intel's Xeon Phi [11]. Following this model the PSK framework is executed in the *host* while the *device* processes the specifics of the physical problem being simulated.

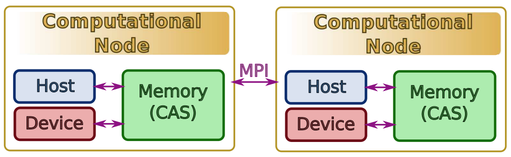

Fig. 1. Hardware architecture model supported by the PSK

In order to run a physical simulation the PSK will construct a defined structure of MPI processes and threads across the CNs to be used for such simulation. Figure 2.Left shows the structure for a simulator using 2 CNs: CN0 and CN1. As the memory address space of CN0 and CN1 are disjoint, the PSK system must provide *Domain Decomposition* (DD) between both CNs (known as extra-node DD), in order to coordinate the simulation process. Moreover, there are some cases (e.g. multi- and many-core architectures), for which the PSK have to provide DD also inside a CN (known as intra-node DD). Notice, that each green and blue nodes into Figure 2.Left represents threads and I/O devices respectively, whereas the brown and red boxes are the MPI processes running on each CN and the intra-node domains in a CN.

3.2 Software Architecture Model

Once the hardware architecture model is defined, this subsection will depict the software architecture model. That is, which is the PSK framework architecture and what is to be done in order to use it and extend or specialize it for a specific physical problem. The specialization process is done by implementing an interface defined by the PSK. Each of the functions to be implemented are known as a *specialized functions*. Among this functions there are: initialization and finalization routines for managing data structures that belongs to the physical problem at hands, proper functions for the processing at each iteration of the simulation process, or some functions for scattering and gathering data among different domains if they are needed. Figure 2.Right shows the PSK framework structure. The dashed boxes represent the functions to be provided by the user in order to specialize the framework for a specific physical problem.

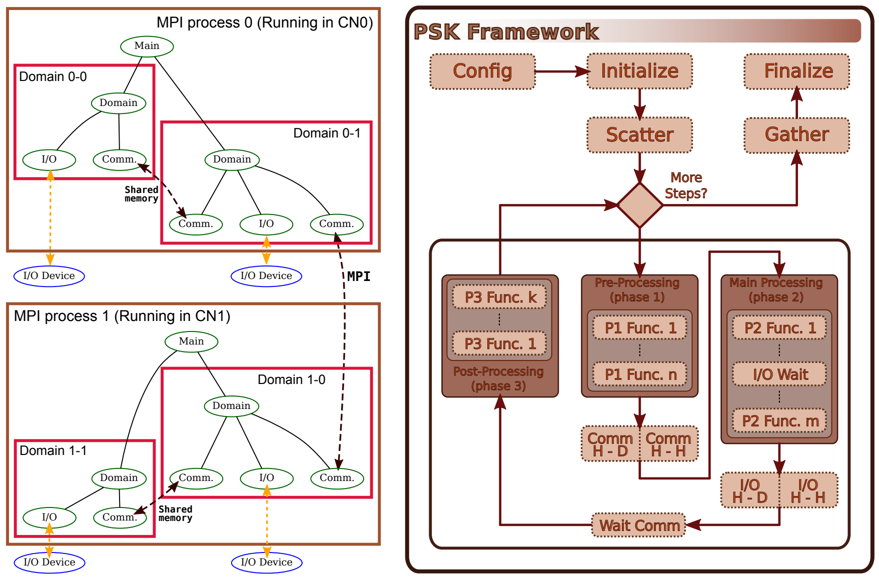

Fig. 2. Left: domain decomposition model. Right: PSK software model.

Regarding the PSK main structure (i.e. inside the main loop), it is divided in three different phases (known as Pre- Main- and Post-Processing), which are separated by some stages such as communications or I/O. The aim of such structure is to provide an environment capable of overlapping computation with communication and I/O. In order to do so, the functions provided by the user for *phase 1* (P1) must process all the physical problem involved in the exchange stage for the current iteration step. Then, an asynchronous communication of these areas is started while the computation (in *phase 2* - P2) processes the remaining domain for the current iteration. Likewise, an I/O operation may be started asynchronously at the end of *phase 2*, enabling as well overlapping with main computation. Finally, an optional *phase 3* (P3) is also considered for such cases where some processing is needed after the communication, but prior to the next iteration.

4 Application Example

Atmospheric transport models [14] deal with transport of substances in the atmosphere, including natural, biogenic, and anthropogenic origin. The physics of these models describes the transport and removal mechanisms acting upon the substance and predicts its concentration depending on meteorological variables and a source term. These models build on the Advection–Diffusion–Sedimentation (ADS) equation, derived in continuum mechanics from the general principle of mass conservation of particles within a fluid.

As an application example, the FALL3D model has been ported to WARIS framework. FALL3D [3] is a multi-scale parallel Eulerian transport model coupled with several mesoscale and global meteorological models, including most re-analyses datasets. Although FALL3D can be applied to simulate transport of any substance, the model is particularly tailored to simulate volcanic ash dispersal and has a worldwide community of users and applications, including operational forecast, modeling of past events or hazard assessment.

4.1 Volcanic Ash Dispersal

Volcanic ash generated during explosive volcanic eruptions can be transported by the prevailing winds thousands of kilometers downwind posing a serious threat to civil aviation. FALL3D models run worldwide operationally to provide advice to the civil aviation authorities, which need to react promptly in order to prevent in-flight aircraft encounters with clouds. Here, we investigate to which extent WARIS-Transport can accelerate model forecasts and could be used for ensemble forecasting [1]. We focus on a paradigmatic case occurred during April-May 2010, when ash clouds from the Eyjafjallajökull volcano in Iceland disrupted the European airspace for almost one week, resulting in thousands of flight cancellations and millionaire economic loss [6].

The following results include several performance techniques carried out in the WARIS-Transport module. These techniques are: *SIMDization*, blocking and pipeline optimizations of the explicit kernels. Furthermore, the parallel I/O operations have been dramatically improved by implementing an active buffering strategy [7] with two-phase collective I/O calls [13]. As an example, Figure 3 shows how a wise choice of the blocking parameter is crucial to reduce the execution time of the explicit kernel. In this particular case ($256\times2048\times64$ domain size), the kernel execution time has been reduced by 24.1%.

All the tests have been conducted in MareNostrum supercomputer (Intel SandyBridge-EP E5-2670) with different number of processors. The Eyjafjallajökull case involves an input dataset of 9 GBytes of meteorological data, corresponding to 8 days of eruption, and 370 MBytes of concentration and dispersal simulated output data for the coarse-grain mesh ($41\times241\times141$). The FALL3D code taken as a reference requires 1h and 58 minutes to complete this simulation on 16 processors with MPI. On the other hand, the WARIS-Transport module only took 14.4 minutes to process it. These times make our implementation $8.2\times$ faster than Fall3D code. Additionally, strong scaling results were obtained for WARIS-Transport using a fine-grain mesh ($41\times480\times280$) of the Eyjafjallajökull case. Scalability results obtained for

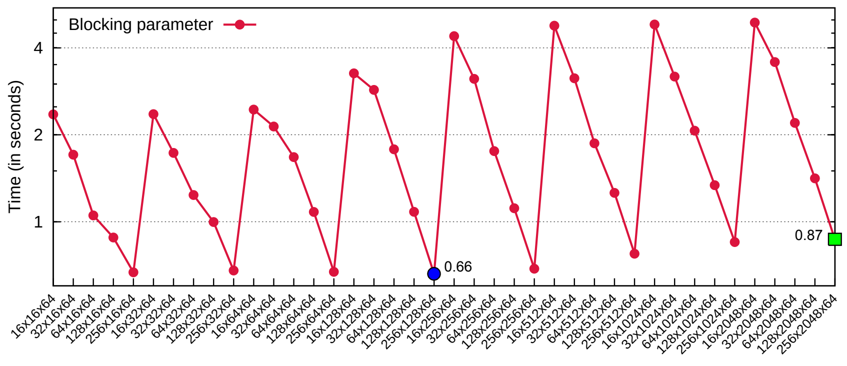

Fig. 3. Blocking impact on the kernel execution time. The green rectangle shows the performance of the naive implementation, whereas the blue circle depicts the best blocking parameter. WARIS automatically selects the best parameter.

Table 1. WARIS-Transport module times (in seconds) with fine-grain mesh of Eyjafjallajökull volcano case. Note that I/O times are overlapped with computation thanks to a double-buffering system and a dedicated I/O thread.

Num.	Explicit Kernel		Others	Meteo data		Output		Total	Speed-up
Proc.	P1 stage	P2 stage	P3 stage	I/O	Postprocess	Preprocess	I/O	Time	
1	0.0	28932.83	4718.99	*302.75*	11108.91	250.06	*133.30*	45010.8	1.0
2	202.21	12004.14	1854.76	*142.79*	5408.69	86.14	*131.25*	19555.9	2.3
4	208.52	6076.23	942.60	*106.75*	2781.53	39.11	*140.52*	10047.9	4.4
8	233.89	3170.07	528.48	*71.08*	1573.25	18.44	*81.86*	5524.1	8.1
16	291.52	2099.81	391.37	*70.15*	917.97	10.93	*29.09*	3711.6	12.1
32	287.49	967.23	192.83	*84.29*	546.03	4.66	*11.81*	1998.2	22.5

different number of processors (up to 32) are broken down in Table 1. Results are categorized in four groups, explicit kernel (P1 and P2 stages), other computations (P3 stage), meteorological input and ash dispersal output. P1 (boundary elements), P2 (internal elements) and P3 stages refer to the kernel functions in the PSK framework structure. Finally, postprocess and preprocess columns consider the data arrangement computation required after reading meteorological data and before writing ash dispersal results respectively.

5 Conclusions

WARIS framework has shown appealing capabilities by providing successful support for scientific problems using FDM. In the foreseeable future, as the amount of computational resources will increase, more sophisticated physics may be simulated. Furthermore, it provides support for a wide-range of hardware platforms. Therefore, as the computational race keeps the hardware changing everyday, support for specific platforms that will give the best performance results will be supplied for the different simulated physics.

References

1. C. Bonadonna, A. Folch, S. Loughlin, and H. Puempel, *Future developments in modelling and monitoring of volcanic ash clouds: outcomes from the first IAVCE-WMO workshop on ash dispersal forecast and civil aviation*, Bulletin of Volcanology **74** (2012), 1–10.
2. D. Callahan, K. Kennedy, and A. Porterfield, *Software prefetching*, ASPLOS-IV: Proceedings of the fourth international conference on architectural support for programming languages and operating systems, ACM, New York, NY, USA, 1991, pp. 40–52.
3. A. Folch, A. Costa, and G. Macedonio, *Fall3d: A computational model for transport and deposition of volcanic ash*, Comput. Geosci. **35**:6 (2009), 1334–1342.
4. M. Frigo and V. Strumpen, *Cache oblivious stencil computations*, 19th ACM International Conference on Supercomputing, June 2005, pp. 361–366.
5. S. Kamil, P. Husbands, L. Oliker, J. Shalf, and K. Yelick, *Impact of modern memory subsystems on cache optimizations for stencil computations*, MSP '05: Proceedings of the 2005 workshop on Memory System Performance, ACM Press, New York, NY, USA, 2005, pp. 36–43.
6. B. Langmann, A. Folch, M. Hensch, and V. Matthias, *Volcanic ash over europe during the eruption of Eyjafjallajökull on iceland, April-May 2010*, Atmos. Environ. **48** (2012), 1–8.
7. X. Ma, M. Winslett, J. Lee, and S. Yu, *Improving MPI-IO output performance with active buffering plus threads*, , IPDPS '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 68.2–.
8. J. McCalpin and D. Wonnacott, *Time skewing: A value-based approach to optimizing for memory locality*, Tech. Report DCS-TR-379, Department of Computer Science, Rutgers University, 1999.
9. K. S. McKinley, S. Carr, and C.-W. Tseng, *Improving data locality with loop transformations*, ACM Trans. Program. Lang. Syst. **18** (1996), 424–453.
10. W. L. Oberkampf and C. J. Roy, *Verification and validation in scientific computing*, 1st ed., Cambridge University Press, New York, NY, USA, 2010.
11. J. Reinders, *An overview of programming for Intel® Xeon® processors and Intel® Xeon Phi coprocessors*, 2012.
12. G. Rivera and C. W. Tseng, *Tiling optimizations for 3D scientific computations*, Proc. ACM/IEEE Supercomputing Conference (SC 2000), IEEE Computer Society, Washington, DC, USA, November 2000, p. 32.
13. J. M. del Rosario, R. Bordawekar, and A. Choudhary, *Improved parallel I/O via a two-phase run-time access strategy*, SIGARCH Comput. Archit. News **21**:5 (1993), 31–38.
14. A. Russell and R. Dennis, *Narsto critical review of photochemical models and modelling*, Atmospheric environment **34** (2000), 2261–2282.
15. J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose gpu programming*, Addison-Wesley Professional, 2010.