Saiph, a Domain Specific Language for Computational Fluid Dynamics simulations

Sandra Macià, Vicenç Beltran, Daniel Mira and Sergi Mateo
Barcelona Supercomputing Center (BSC-CNS)
sandra.macia@bsc.es, vicenc.beltran@bsc.es, daniel.mira@bsc.es and sergi.mateo@bsc.es

***Abstract*—Nowadays, High-Performance Computing (HPC) is assuming an increasingly central role in scientific research while computer architectures are becoming more and more heterogeneous and using different parallel programming models and techniques. Under this scenario, the only way to successfully exploit an HPC system requires that computer and domain scientists work closely towards producing applications to solve domain problems, ensuring productivity and performance at the same time. Facing such purpose, Saiph is a Domain Specific Language designed to ease the task of solving couple and uncouple Partial Differential Equations (PDE's), with a primary focusing on Computational Fluid Dynamics (CFD) applications. Saiph allows to model complex physical phenomena featured by PDE's, easing the use of numerical methods and optimizations on different computer architectures to the users.**

I. Introduction

THIS project aims to ease the development of scientific applications by allowing domain experts to transcribe their equations into the code Saiph and then generating HPC-ready code that efficiently exploits the computational resources of modern heterogeneous supercomputers while dealing with all the specific aspects of solving systems of PDE's. To achieve that, Saiph provides a high-level syntax that directly maps with concepts of the domain, hiding from the user all the complexities related to numerical methods and HPC systems. Users only have to translate their equations to Saiph language and specify some physical and numerical parameters; initial and boundary conditions and post-processing strategy. Later, domain optimizations can be internally applied providing extra efficiency and correctness boosting the workflow productivity. The final specialized system of equations is then solved in parallel using MPI and applying intra-node parallelization techniques (using OpenMP/OmpSs) to achieve high computational performance.

II. Saiph overview

This section introduces the design and underlying technology used for the development of Saiph and the resulting high-level language. We briefly describe the Saiph project and its state of development.

A. Saiph design and underlying technologies

Saiph, as a DSL, has been designed to be simple, efficient, largely applicable in Computational Mechanics problemas and in particular on CFD applications. It has been implemented as an embedded compiler in Scala[1] using the Lightweight Modular Staging (LMS)[2] as a DSL development platform and the Scala Virtualized Compiler[3].

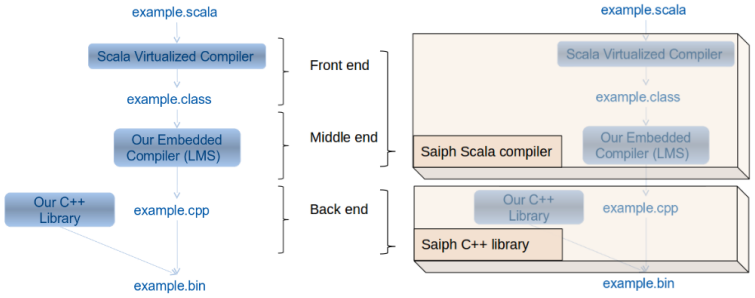

Fig. 1. Underlying design and technologies of the compilation process

Saiph applications are compiled at the front end with the LMS and the Saiph implementation together using the Scala Virtualized Compiler. At the middle end, the output of the first phase is compiled using our embedded compiler; the domain specific optimizations implementations are applied at this point and the LMS generates the corresponding IR nodes. Finally, the output of the embedded compiler (C++ file) is compiled and linked with our low-level C++ library that handles the numerical and parallelization issues, producing a binary ready to be executed in parallel . Figure 1 shows the compilation process and the internal design structured by layers.

Saiph has two main layers: the Scala compiler and the C++ library. This separation eases the DSL development, as in each of them the efforts are devoted to the developments naturally belonging to the layer. In that way, we can take advantage of each tool being used at its natural layer. High-level domain-oriented syntax and domain specific optimizations are thus implemented at the Scala compiler, while MPI and OpenMP libraries and auxiliary, mainly numerical methods, are integrated into the C++ library.

B. Saiph as a language

Saiph offers a high level syntax to unambiguously define a complete system of PDE's used for the characterization of a physical phenomena. The main components are presented in this section.

1) Units: The basic component of Saiph, they represent a physical magnitude through a value and its dimensionality information. `Units` can be combined and compared, and are internally used to check the consistency of the equations.

```
def L = 1 * Meters
```

2) Mesh: Saiph works with Cartesian meshes that can be defined in 1D, 2D or 3D. The sizes are specified by the users using physical continuous space.

```
val mesh = CartesianMesh(L, L)
```

3) Terms and Consterms: Saiph offers these components to represent dimensional variables and constants for the problem.

```
val T1 = Term(Temperature)("Temp1", mesh, 300 * Kelvins)
```

4) Operators: There is a complete set of numerical operators available to combine the different terms in order to build the system of discretized equations.

5) Equations: Defining equations in Saiph involves declaring terms and combine them through operators. An equation is formed by the left-hand side and the right-hand side expressions. Consequently, the units of both sides must match, otherwise, Saiph emits and error.

```
val eq1 = Equation(lhs_expr, rhs_expr)
```

C. Internal features

Internal features are completely transparent to the user.

1) Numerical methods: Numerical evaluations require the discretization of continuous functions, models, and equations which are time-space dependent. The spatial discretization follows explicit high-order schemes based on the finite difference method for which different oprators are available, while the time marching is based on high-order Runge Kutta methods. Saiph uses non-uniform structured meshes.

2) Exploiting parallelism: Inter and intra-node parallelism are harmoniously combined. For the inter-node parallelism, the mesh is partitioned by the last dimension and a similar workload is distributed across the available MPI processes. Each process solve its part of the mesh for the whole simulation. Computations at each time-step are parallel and dependence's free but, after each of them, each MPI process has to exchange its boundaries with its neighbors in order to correctly update all the values to be used for the spatial derivatives of future computational steps. Regarding intra-node parallelism, each equation can be integrated in parallel at any time-step. The iteration space of the loop traversing the mesh is distributed across available OpenMP threads and executed in parallel. Each equation is solved in parallel for all the points of the mesh, one equation after the other.

3) Domain specific optimizations: Saiph has features and components which does not change the user external interface while changing the Saiph internal behaviour. Those optimizations are specific for the resolution of PDEs systems and even more specific for the resolution of CFD problems. As an example of those optimizations, we considered the *advection term* of a convection-diffusion-reaction (CDR) equation. When identified, this term can be treated with different operators depending on the type of problem. For instance, for convection-dominated flows, a low-dissipation *central scheme* can be used, although *upwind differentiation* can be activated when there is no sufficient resolution to capture the sharp of the gradients.

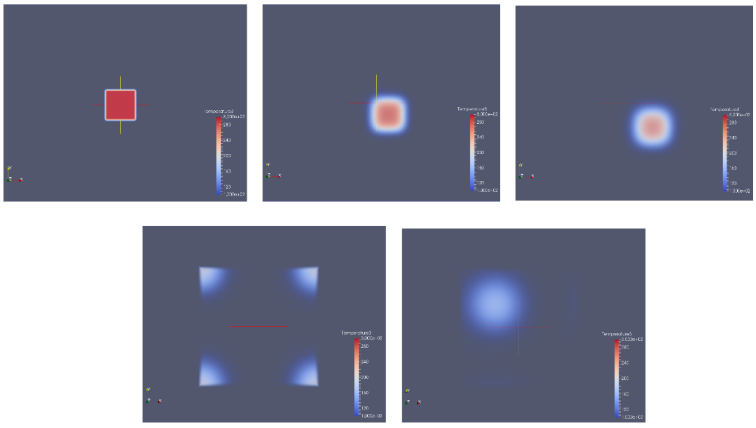

Fig. 2. Saiph - Two-dimensional advection-diffusion simulation

III. Application example

We present an advection-diffusion problem, represented by the following partial differential equation.

$$\frac{\partial T}{\partial t} = -\mathbf{u} \cdot \nabla T + \nabla \cdot (k \nabla T) \tag{1}$$

A small hot 2D cube is being transported within a periodic domain. The following snippet code corresponds to the transcription of this equation into Saiph code.

```
val eq = Equation(dt(T), −u∗grad(T) + div(k∗grad(T)))
```

Figure 2 shows the results from this advection-diffusion Saiph application.

IV. Conclusion

Saiph appears to be a powerful tool that can be advantageously used by scientists without knowledge on numerical methods and High-Performance Computing, while internally providing the advantages of such expertise. Several domain specific optimizations can be implemented as well as new suitable numerical methods and parallelization strategies boosting the efficiency, flexibility and genericity while maintaining its usability.

References

[1] M. Odersky and al., "An Overview of the Scala Programming Language," EPFL, Lausanne, Switzerland, Tech. Rep. IC/2004/64, 2004.

[2] T. Rompf and M. Odersky, "Lightweight modular staging: a pragmatic approach to runtime code generation and compiled dsls," ser. GPCE '10. New York, NY, USA: ACM, 2010, pp. 127–136. [Online]. Available: http://doi.acm.org/10.1145/1868294.1868314

[3] A. Moors, T. Rompf, P. Haller, and M. Odersky, "Scala-virtualized," ser. PEPM '12. New York, NY, USA: ACM, 2012, pp. 117–120. [Online]. Available: http://doi.acm.org/10.1145/2103746.2103769

Sandra Macià studied physics at the UB. After obtaining her degree she enrolled the Master in Innovations and Research in Informatics, on the High Performance Computing specialization, MSc at UPC-FIB, where she obtained the *Severo-Ochoa MSc scholarship*. For her master thesis she joined the *Barcelona Supercomputing Center*, Computer Science Dpt., where she started to work on Domain Specific Languages for the resolution of systems of Partial Differential Equations. Currently she is developing her PhD studies as a Severo Ochoa PhD student, targeting the same subject and with focus on Computational Fluid Dynamics applications.