Future Generation Computer Systems 67 (2017) 72–82

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

MUSCLE-HPC: A new high performance API to couple multiscale parallel applications

Mohamed Ben Belgacem *, Bastien Chopard

Department of Computer Science, University of Geneva, Battelle, Batiment A, Route de Drize 7, Carouge 1227, Geneva, Switzerland

HIGHLIGHTS

- We proposed a HPC methodology to couple multiscale applications.
- We presented its design and implementation and we measured its performance.
- We showed that its performance is comparable to the MPI native one.

ARTICLE INFO

Article history:
Received 20 April 2016
Received in revised form
29 July 2016
Accepted 22 August 2016
Available online 6 September 2016

Keywords:
Distributed computation
HPC infrastructure
Multiscale coupling

ABSTRACT

Multiscale, multi-physics applications are central to solve the increasing number of important scientific challenges. Computationally speaking, the difficulty is to combine high performance computing with the need to couple various codes or solvers, each representing a different scale or a different physical process. In this paper, we present MUSCLE-HPC a new HPC implementation of MUSCLE-2, a previously developed Multiscale Coupling Library and Environment. We present its design and implementation and we demonstrate its advantages compared to MUSCLE-2. We conduct a performance comparison through a tightly coupled MPI application use-case. Our results indicate that using MUSCLE-HPC to couple submodels within the same HPC cluster can lead to better computing performance comparable to a native MPI execution and can, thus, reduce the coupling overhead.

1. Introduction

Nowadays, the landscape of High Performance Computing (HPC) resources tends to evolve towards heterogeneous architectures, combining within the same machine CPU-optimized cores, large-memory cores and GPU's. At the small scale, cores are organized as shared memory systems, whereas, at large scale they offer a distributed memory model. As HPC resources become more powerful, computational scientist addresses more challenging problems. Multiscale, multiphysics applications are a rising field, that not only require high performance computation, but also new software solutions to match the heterogeneity of the applications with that of the current HPC architectures.

The classical HPC approach, based on monolithic codes can no longer offer the flexibility to build and maintain sophisticated multiscale, multiphysics applications. The reason is that they involve different processes (representing various natural phenomena) which require several multidisciplinary teams collaborating together, and different numerical solvers. A typical multiscale application contains various components interacting with each other and acting at different spatial and temporal scales. These components are often developed with different programming languages and require various computing resource architectures and hardware configurations, such as CPU (shared or distributed memory), GPU, cloud resources, supercomputers, etc. It is therefore clear that, in addition to the computing resources, there is a need for a formalism and a methodology to optimally design, maintain and run this kind of applications on modern heterogeneous parallel infrastructures. This question is central to the recent European project ComPat [1] whose goal is to define and implement multiscale computation patterns on HPC resources.

Most of the existing frameworks to develop and couple multiscale applications are rather application-oriented tools and not generic enough to be applied to other applications. A recent review [2] shows that existing multiscale projects tend to use a variety of approach to develop and couple their multiscale codes,

* Corresponding author.
E-mail addresses: mohamed.benbelgacem@unige.ch (M. Ben Belgacem), bastien.chopard@unige.ch (B. Chopard).
URLs: http://cui.unige.ch/~benbelga/ (M. Ben Belgacem), http://cui.unige.ch/~chopard/ (B. Chopard).

http://dx.doi.org/10.1016/j.future.2016.08.009
0167-739X/

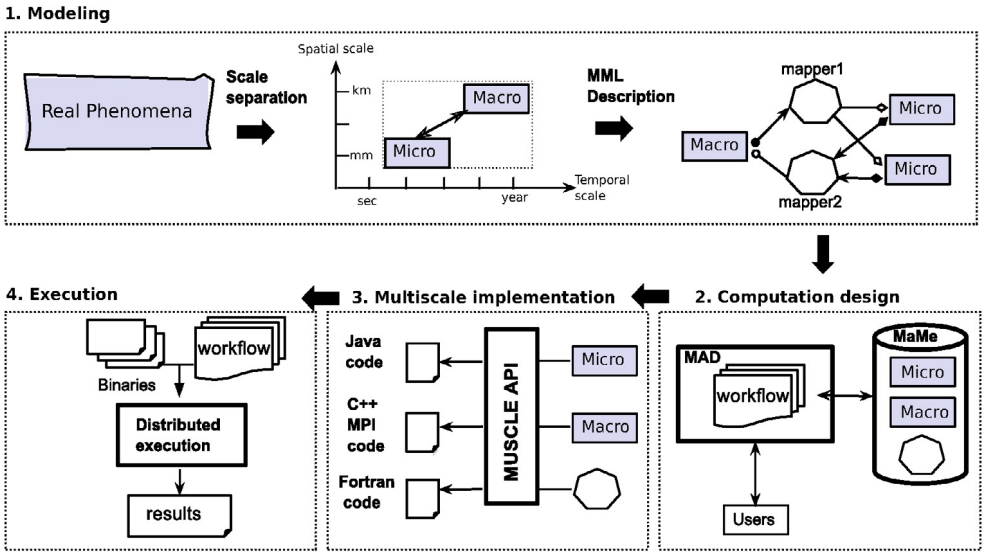

Fig. 1. The different steps to design, program, implement and run a multiscale application with the MMSF developed in the MAPPER project.

mostly using hand-written scripts and run computation on a single machine.

There are few frameworks that propose generic tools and concepts to develop and run multiscale applications on HPC infrastructure. In [3] a Multiscale Universal Interface (MUI) framework is presented with the aim to facilitate the coupling of different numerical solvers with an independent data interpretation approach. MUI provides a set of C++ classes and interfaces to develop components and facilitates the exchange of information across solvers by letting them push and fetch data. Data transfer is done through using fetch()/push() methods which are MPI based point-to-point non-blocking send and blocking receive methods. OASIS coupler [4] is another framework developed within the European project IS-ENES2 with the purpose to couple several numerical codes (solvers) representing different components of the climate system. This framework offers a coupling library to interface independent parallel codes together in a generic way. Some other works aimed to offer tools to perform cross-site MPI execution over different cluster sites. This amounts to provide a transparent mechanism for inter-sites process communication with a lower performance overhead. MPICH-G2 [5] and MPIg [6] are two successful attempts that used the Globus Toolkit services [7] to perform cross-site executions. They enable the user to run parallel code across multiple and heterogeneous clusters sites using the same command as in parallel machine. QCG-OMPI [8] is another tool which uses the QosCosGrid [9] grid-level extension of its runtime environment to address connectivity issues between cluster sites. It supports MPI run across a federation of clusters. QCG-OMPI allows direct communications between processes whenever it is possible. In the case of Firewall restriction, communications are relayed through a proxy service installed at each cluster site. In the same perspective, MPWide [10] is a C++ light-weight communication library for performing message passing between supercomputers. This API is easy to install and provides a superior communication performance compared to the previous cross-side tools.

The above frameworks propose programming libraries to couple multiscale applications but do not provide a theoretical approach to better describe and understand the multiscale coupling. However, they reflect the current need of the scientific community for new frameworks that target the programming and running of multiscale applications on HPC resources, especially, connecting MPI-based codes together.

The MAPPER [11] European project addressed running multiscale applications on the European computing infrastructure (EGI [12] and PRACE [13]). The objective of this project was to provide a formalism and a methodology to design, couple and run multiscale applications on distributed HPC infrastructures. This project came up with a framework, coined the *Multiscale Modeling and Simulation Framework* (MMSF) [14,15], offering a theoretical methodology and a programming paradigm. Briefly, as depicted in Fig. 1, a first scale-aware separation step consists in decomposing a real phenomena into several single time/space scale processes (called submodels) and defines the interactions between them. A submodel can be seen as the adequate numerical model capable of computing/simulating a given real phenomena on a defined spatial region (called domain) during a given period of time. A given submodel can be abstracted by a generic Sub-Execution Loop (SEL) composed of limited set of operators: Initialization ($\mathbf{F_{init}}$), Solver (**S**), Boundary (**U**), intermediate Observation ($\mathbf{O_i}$) and final Observation ($\mathbf{O_f}$). The initialization of the computation variables is done in the $\mathbf{F_{init}}$ operator. At each iteration the **S** operator solves the submodel domain and **U** provides the boundary conditions. The $\mathbf{O_i}$ and $\mathbf{O_f}$ operators compute and deliver the desired physical quantities from the current values of the computation variables. Therefore, the coupling between submodels amounts to a data communication between their corresponding operators, as illustrated in Fig. 2, for two very common couplings (see [14] for the coupling templates defined in the MAPPER methodology).

The coupling template of the left-hand side of Fig. 2 illustrates the coupling of two submodels having a time scale overlap. In this case, the intermediate observation $\mathbf{O_i}$ of each submodel is needed to update the boundary condition (**U**) of the other submodel. On the right-hand side of Fig. 2, the coupling template illustrates the coupling of two submodels having a time scale separation. It means that for each time iteration of submodel 1, a complete execution of submodel 2 is needed. Therefore, the final observation $\mathbf{O_f}$ of submodel 2 is needed to solve an iteration (**S** operator) of submodel 1. Note that here the information goes to **S** which is typically the case when the domain of submodel 2 is a sub-domain of submodel 1. If the domains are different, the coupling would go from $\mathbf{O_f}$ to **S** to provide a proper boundary condition to submodel 1.

Data communication is performed through high level, reusable, scale-aware software components (called *conduit*s, *filter*s and *mapper*s) that implement the scale bridging methods. A *conduit* is an abstract notion that represents the link between SEL operators. A *mapper* is a component that receives data from one