Journal of Computational Science xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Journal of Computational Science

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

Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment

J. Borgdorff[a,*], M. Mamonski[b,1], B. Bosak[b], K. Kurowski[b], M. Ben Belgacem[c], B. Chopard[c], D. Groen[d], P.V. Coveney[d], A.G. Hoekstra[a,e]

[a] Computational Science, Faculty of Science, University of Amsterdam, Amsterdam, The Netherlands
[b] Poznań Supercomputing and Networking Center, Poznań, Poland
[c] Computer Science Centre, University of Geneva, Carouge, Switzerland
[d] Centre for Computational Science, University College London, London, United Kingdom
[e] National Research University ITMO, Saint-Petersburg, Russia

ARTICLE INFO

Article history:
Received 4 November 2013
Received in revised form 24 March 2014
Accepted 8 April 2014
Available online xxx

Keywords:
Distributed multiscale computing
Multiscale modelling
Model coupling
Execution environment
MUSCLE

ABSTRACT

We present the Multiscale Coupling Library and Environment: MUSCLE 2. This multiscale component-based execution environment has a simple to use Java, C++, C, Python and Fortran API, compatible with MPI, OpenMP and threading codes. We demonstrate its local and distributed computing capabilities and compare its performance to MUSCLE 1, file copy, MPI, MPWide, and GridFTP. The local throughput of MPI is about two times higher, so very tightly coupled code should use MPI as a single submodel of MUSCLE 2; the distributed performance of GridFTP is lower, especially for small messages. We test the performance of a canal system model with MUSCLE 2, where it introduces an overhead as small as 5% compared to MPI.

1. Introduction

Multiscale modelling and simulation is of growing interest [21], with appeal to scientists in many fields such as computational biomedicine [33], biology [34], systems biology [14], physics [15], chemistry [27] and earth sciences [3]. Meanwhile, there are efforts to provide a more general way of describing multiscale models [26,38,7], including our Multiscale Modelling and Simulation Framework [13,23,24,8]. This framework describes the process of constructing a multiscale model by identifying and separating its scales, defining a multiscale model as a set of coupled single scale models. It then provides a computational modelling language and environment to create and deploy such models on a range of computing infrastructures. For an example of biomedical applications in this context, see [19].

In this paper we present a means to implement multiscale models as described in this theoretical framework: The Multiscale Coupling Library and Environment 2 (MUSCLE 2). It takes a component-based approach to multiscale modelling, promoting modularity in its design. In essence, it treats single scale models as a separate components and facilitates their coupling, whether they are executed at one location or multiple. It is open source software under the LGPL version 3 license and is available at http://apps.man.poznan.pl/trac/muscle. MUSCLE 1 [22] generally had the same architecture and it was based on the Complex Automata theory [23,24] and focussed on multi-agent multiscale computing. The differences between MUSCLE 1 and 2 are discussed in Appendix A.3, and amount to sharing only 4% of their code. The main goal of creating a successor to MUSCLE 1 was to support simulations on high performance computing infrastructures.

Distributed computing is a way to take advantage of limited and heterogeneous resources in combination with heterogeneous multiscale models. There are several motivations for distributing the computation of a multiscale model: to make use of more resources than available on one site; making use of heterogenous resources such as clusters with GPGPUs, fast I/O, highly interconnected CPU's, or fast cores; or making use of a local software license on one

* Corresponding author. Tel.: +31 20 525 7446.
E-mail addresses: J.Borgdorff@uva.nl, joris@jorisborgdorff.nl (J. Borgdorff), bbosak@man.poznan.pl (B. Bosak), krzysztof.kurowski@man.poznan.pl (K. Kurowski), mohamed.benbelgacem@unige.ch (M. Ben Belgacem), bastien.chopard@unige.ch (B. Chopard), d.groen@ucl.ac.uk (D. Groen), p.v.coveney@ucl.ac.uk (P.V. Coveney), A.G.Hoekstra@uva.nl (A.G. Hoekstra).

[1] Mariusz Mamonski (1984–2013) suddenly deceased after the first submission of this article. He had a major role in the development, deployment, tests, support, and software compatibility of MUSCLE 2.

http://dx.doi.org/10.1016/j.jocs.2014.04.004

machine and running a highly parallel code on a high-performance cluster. Projects such as EGI and PRACE make distributed infrastructure available, and software that uses it is then usually managed by a middleware layer to manage distributed computing for users [39].

Quite a few open and generic component-based computing frameworks already exist, for instance the CCA [2] with CCaffeine [1], the Model Coupling Toolkit (MCT) [28,29], Pyre [11], or OpenPALM [12]; see the full comparison by Groen et al. [21]. The Model Coupling Toolkit has a long track-record and uses Fortran code with MPI as a communication layer so it potentially makes optimal use of high-performance machines. OpenPALM uses TCP/IP as a communication layer and it is packaged with a graphical user interface to couple models. Both frameworks provide some built-in data transformations. MUSCLE 2 uses shared memory for models started in the same command and TCP/IP for multiple commands. An advantage over the other mentioned frameworks is that it provides additional support for distributed computing and for Java. However, it has fewer built-in data transformations available and does not provide tools for implementing the contents of single scale models, so it should be combined with domain-specific libraries.

There are many libraries for local and wide-area communications, apart from MPI implementations and the ubiquitous TCP/IP sockets. MPWide [20], for instance, is a lightweight library that optimises the communication speed between different supercomputers or clusters; ZeroMQ [25] is an extensive communication library for doing easy and fast message passing. To use them for model coupling these libraries have to be called in additional glue code. MUSCLE 2 optionally uses MPWide for wide area communication because of its speed and few dependencies.

So far MUSCLE 2 is being used in a number of multiscale models, for instance a collection of parallel Fortran codes of the Fusion community [17], a gene regulatory network simulation [37], a hydrology application [5], and in a multiscale model of in-stent restenosis [10,6,19].

In this paper, we introduce the design of MUSCLE 2 in Section 2, including the theoretical background of the Multiscale Modelling and Simulation Framework, MUSCLE 2's Programming Interface (API) and runtime environment. The performance and startup overhead of MUSCLE 2 is measured in Section 3 in a number of benchmarks. Finally, in Section 4 two applications that use MUSCLE are described, principally a multiscale model of a complex canal system, for which additional performance tests are done.

2. Design

MUSCLE 2 is a platform to execute time-driven multiscale simulations. It takes advantage of the separation between the submodels that together form the multiscale model, by treating each submodel as a component in a component-based simulation. The submodels individually keep track of the local simulation time, and synchronise time when exchanging messages.

A strict separation of submodels is assumed in the design of MUSCLE 2, so the implementation of a submodel does not dictate how it should be coupled to other submodels. Rather, each submodel sends and receives messages with specified ports that are coupled at a later stage. When coupling, modellers face their main scientific challenge: to devise and implement a suitable scale bridging method to couple single scale models. MUSCLE 2 supports the technical side of this by offering several functional components, described in Section 2.1.

The runtime environment of MUSCLE 2 executes a coupled multiscale model on given machines. It can run each submodel on an independent desktop machine, local cluster, supercomputer, or run all submodels at the same location. For instance, when one or more submodels have high computational requirements or require alternate resources such as GPU computing, these submodels can be executed on a suitable machine, while the others are executed on a smaller cluster. A requirement is that a connection can be established between submodels, and that a message can only be sent to currently running submodels. For some models a local laptop, desktop or cluster will suffice; MUSCLE 2 also works well in these scenarios. Technical details about the runtime environment can be found in Appendix A.

MUSCLE 2 is separated into an API, which submodel code uses, a coupling scripting environment that specifies how the submodels will be coupled, and a runtime environment, that actually executes the multiscale model on various machines. The library is independent from the coupling, which is in turn independent from the runtime environment. As a result, a submodel is implemented once and can be coupled in a variety of ways, and then executed on any suitable set of machines. Additionally, future enhancements to the runtime environment are possible without changing the library.

2.1. Theoretical background

To generally couple multiscale models, a framework describing the foundations of multiscale modelling [13,8,27,26] and its repercussions on multiscale computing [7,16] was conceived. It starts by decomposing a phenomenon into multiple single scale phenomena using a scale separation map as a visual aid. Based on these single scale phenomena, single scale models are created; see Fig. 1. Ideally, these single scale models are independent and rely only on new messages at specific input ports, while sending messages with observations of their state at output ports. By coupling output ports to input ports using so-called conduits, a multiscale model is formed. Assuming a time-driven simulation approach, each message is associated with a time point, which should be kept consistent between single scale models.

The theoretical framework distinguishes between acyclically and cyclically coupled models. In the former, no feedback is possible from one submodel to the other, while in the latter a submodel may give feedback as often as needed. This distinction has many computational implications, such as the need to keep submodels active in cyclically coupled models, or the recurring and possibly dynamic need for computing resources. MUSCLE 2 focusses on cyclically coupled models by keeping submodels active during the entire simulation, whereas workflow systems tend to focus on acyclically coupled models.

Listing 1. Submodel execution loop in MUSCLE 2

```
do {
    init()
    while (!endCondition()) {
        intermediateObservation()
        incrementTime()
        solvingStep()
    }
    finalObservation()
} while (restartSubmodel())
```

To facilitate consistency, submodels each have a fixed submodel execution loop as in Listing 1, consisting of initialisation, a loop with first an observation and then a solving step, and then a final observation. This loop can be restarted as long as a submodel with a coarser time scale provides input for the initial condition. During initialisation and solving steps, only input may be requested, and during the observations, only output may be generated. Although this is the general contract, submodel implementations in MUSCLE 2 may diverge from this loop, for example if it would increase performance.

Please cite this article in press as: J. Borgdorff, et al., Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment, J. Comput. Sci. (2014), http://dx.doi.org/10.1016/j.jocs.2014.04.004