Technical Report

TR-2015-006

Platform and Algorithm Eff ects on Computational Fluid Dynamics Applications in Life Sciences

by

Sofia Guzzetti, Tiziano Passerini, Jaroslaw Slawinski, Umberto Villa, Alessandro Veneziani, Vaidy Sunderam

Mathematics and Computer Science

EMORY UNIVERSITY

Platform and Algorithm Effects on Computational Fluid Dynamics Applications in Life Sciences

Sofia Guzzetti*, Tiziano Passerini[1], Jaroslaw Slawinski[2], Umberto Villa[3], Alessandro Veneziani, Vaidy Sunderam

Department of Mathematics & Computer Science, Emory University, Atlanta, GA 30322, USA

Abstract

High Performance Computing (HPC) is a mainstream mode of exploration and analysis in different fields, not only technical but also social and life sciences. A well established HPC domain is medicine, and cardiovascular sciences in particular. The adoption of CFD as a tool for diagnosis, prognosis, and treatment planning in the clinical routine is however still an open challenge. The computational analysis of large numbers of patients calls for significant computational resources, and traditional local clusters may be not adequate to deliver the computational needs. Alternative solutions like grids and on-demand cloud resources need to be seriously considered. This paper proposes methodologies and protocols to identify computing platforms for hemodynamics computations that will be increasingly needed in the future. We focus on hemodynamics in patient-specific settings and present extensive results on different platforms. We propose a way to measure and estimate performance and running time under realistic scenarios. In addition, we discuss in detail the optimal (parallel) partitioning of the domain of a problem of interest with different mathematical approaches. We show that an overlapping splitting is generally advantageous and the detection of optimal overlapping has the potential to significantly reduce computational costs of the entire solution process.

*Corresponding author. 400 Dowman Dr, Atlanta, GA 30322, Mathematics and Science Center Suite W401, Ph: 404-268-2789, Email: sofia.guzzetti@emory.edu.

Email addresses: `sofia.guzzetti@emory.edu` (Sofia Guzzetti), `tpasser@emory.edu` (Tiziano Passerini), `jslawin@emory.edu` (Jaroslaw Slawinski), `uvilla@emory.edu` (Umberto Villa), `avenez2@emory.edu` (Alessandro Veneziani), `vss@emory.edu` (Vaidy Sunderam)

[1]Present address: Siemens Medical Solutions, Princeton, NJ

[2]Present address: Microsoft, Seattle, WA

[3]Present address: Center for Computational Geosciences and Optimization, ICES, The University of Texas at Austin, Austin, TX

Keywords: Applied computing in medicine, Platform heterogeneity, Cloud computing, Computational fluid dynamics, Cost characterization, Adaptation, Parallel overlapping Domain Decomposition, Performance analysis

1. Introduction and Background

Computational fluid dynamics (CFD) has been progressively adopted in the last decade for studying the role of blood flow on the development of arterial diseases (see e. g. [1, 2]). Computational investigations - compared to more traditional *in vitro* and *in vivo* studies - are generally more flexible and cost-effective. In combination with appropriate image-processing techniques - see e. g. [3] - CFD can be used in a *patient-specific setting.* This means that the morphological and functional conditions of a specific patient may be reproduced in mathematical terms and quantitative analyses can be performed by solving the corresponding partial differential equations describing the physical and constitutive laws behind the physiopathology. There are several uses for this kind of analysis, including a deeper understanding of the clinical conditions, performing virtual surgery or therapy for predicting outcomes, to a personalized optimization/customization of generic procedures [4, 5, 6, 7, 8].

Adoption of CFD as a tool for the diagnosis, prognosis, and treatment planning in the clinical routine is however still an open challenge. In fact, the time for obtaining results from computational studies is often too long for the fast-paced clinical environment. Furthermore, association of computed blood-flow patterns with outcome is still not supported by large enough sample sizes. On the other hand, computational analysis of large number of patients calls for significant computational resources [9]. In short, we may say that computational hemodynamics is a field with great potential, but currently limited by time and cost constraints.

Increasingly however, scientists and clinicians have access to several classes of available computing platforms which could alleviate the resource bottleneck. While local (owned) resources are faster and cheaper, overall system and operating expenses have led to resource sharing and resource leasing paradigms, i.e. *grids and clouds*, respectively. But it is not trivial to identify the platform that best suits the problem to be solved in each situation. Overall performance depends on two interrelated factors: (1) the architecture of the physical resource and (2) its optimal exploitation for the specific problem to solve. In real production settings, performance must also be balanced with cost.

As for the first aspect, traditionally performance of HPC applications has been measured by a single metric, i.e., time to completion for the particular application at hand, parameterized with respect to problem size and number of processing elements used. Nevertheless, with the advent of cloud computing, the viability of executing parallel applications on the cloud (either through self-assembly or renting a prebuilt cluster) and the actual dollar cost effectiveness of executing HPC applications on different target platforms have become relevant. On the other hand, communication is an issue of paramount concern in the matter of efficiency. On clouds, a great deal of attention has been devoted to data handling but there has been relatively little focus on interconnection network capabilities. For explicit message passing parallel programs, such as those which make use of MPI, data handling and interconnection network capabilities lead to substantial heterogeneity in communication, with significant impact on performance. It is worth stressing that most real-life applications we are interested in are not regular or symmetric and thus their MPI process communication graphs are unevenly weighted.

In this work we explore two viable approaches to tackle these issues.
(a) We re-map the effective topology of the application's interconnection network by managing the allocation of MPI processes to processor cores *before* the execution of the application, so that highly coupled MPI processes are "close", i.e. mapped on cores within a single node. In this way the intra-node communication is maximized and the long-distance inter-node communication is reduced.
(b) We consider well-established methods to associate mathematical formalism to the parallel solution of complex systems of partial differential equations (PDEs). In particular, we resort to *domain decomposition techniques* (DD) to detect the *optimal* splitting of the tasks that minimizes the computational time. This method was historically introduced - well before the advent of parallel computing - to compute manually the solution of PDEs by splitting the process over different subdomains of the region of interest to take advantage of simple geometries (e.g. a L-shape domain was split into rectangles Fig. 2) where simple methods were available. Nowadays, DD is a powerful approach to manage the solution over different computational resources either with or without overlapping of subdomains, depending of the specific problem of interest and the identification of optimal interfaces to minimize inter-node communications.

Our reference application is the solution of problems related to computational hemodynamics, blood flow and solutes like Oxygen. We aim at demonstrating the relevance of all these issues in a realistic context, when dealing with a patient-specific setting. For this reason, we consider the vascular geometry of a patient, so as to discuss our strategies on a case of real interest. We use an object oriented C++ library for the solution of PDEs with the finite element method (FEM) called LiFEV ("Library for Finite Elements 5") [10], extensively adopted in several projects of practical interest - see e.g. [8, 11, 12, 13, 14, 15].

In Section 2 we discuss our experiences with comparing cost and utility on three typical platform types: Infrastructure as a Service (IaaS) clouds, grids, and on-premise local resources, with a particular focus on process-to-node mapping vis-a-vis efficiency.

In Section 3 we consider the work balance in terms of DD and interface handling. We present an automatic procedure to optimize the mapping of the sub-domains to the available processing units based on graph analysis. We first consider a non overlapping strategy, where each domain shares with the others only the interface (e.g. a surface cutting in our case the volume of the artery of interest). However, it is well known that this is not necessarily the best option. In fact, a faster convergence to the desired solution in the iterative-by-subdomain approach can be attained if we allow some overlapping.

In Section 4 we test this option in both idealized and real 3D geometries. We show that the detection of the optimal overlapping in real cases - albeit non trivial - has the potential to significantly reduce the computational costs of the entire solution process.

1.1. The mathematical problem and its numerical solver

Computational hemodynamics requires the study of incompressible fluids. Incompressible fluid dynamics represents one of the most challenging, attractive and impactive problems in modern scientific computing. Fast and reliable numerical solutions of the Navier-Stokes equations (NSE) – the basic mathematical model for incompressible fluid dynamics – are required in several engineering fields, ranging from automotive to geophysical and biomedical engineering [16, 17]. From the computational viewpoint, these equations are very challenging, not only due to the size (it is a vector problem involving four scalar fields, three components of velocity and the pressure), but to intrinsic mathematical features (see, e.g. [17]). In the test we use for our experiments, NSE - completed by appropriate initial and boundary conditions - are solved for computing blood velocity and pressure in an artery affected by a disease, called *cerebral aneurysm.* The latter consists of an abnormal sac in the artery, inducing non-physiological flow patterns that can lead eventually to rupture of the arterial wall and brain hemorrhage.

The application of computational hemodynamics to the study of vascular diseases is time- and cost- sensitive, as it typically entails the generation of large data sets of simulations on patient populations, with the final goal of finding statistical correlations of flow patterns with outcome [18, 8]. Here, in particular, we consider a benchmark problem proposed in the Inaugural CFD Challenge Workshop [19], i. e. the study of blood flow inside a giant brain aneurysm in an internal carotid artery.

The equations are approximated by FEM combined with backward difference formulas (BDF) to handle the time dependence. With FEM, the solution is approximated by a piecewise polynomial function over subdivisions of the artery, called *elements*. The collection of elements is called *mesh*. This step reduces the partial differential equations to a system of ordinary differential equations in time. The latter is finally solved in selected instants by a second order BDF approximation. At each time step a large sparse (i. e. with the majority of entries of the associated matrix equal to 0) linear system needs to be solved. The more elements are introduced in the computational domain and the more instants are collocated for the numerical solution, the higher the computational costs of the procedure are and the more accurate the solution is. In particular, here we consider a mesh with 837,154 elements, such that the total number of unknowns in the linear system is 3,162,146. The equations are collocated in 100 instants within the cardiac cycle (i. e. the simulation time step is $0.01s$). A snapshot of the computed solution is shown in Figure 1.

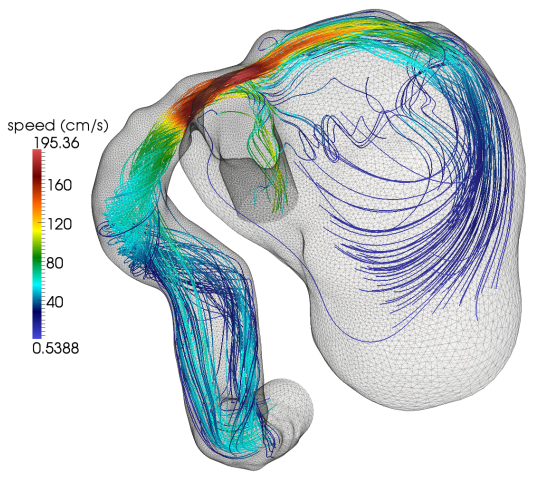


Figure 1: Solution of the problem, based on NSE, when $t = 0.28$s. Streamlines of the velocity field, when the flow rate is maximum over the cardiac cycle.

1.2. Domain decomposition techniques for the solution of Partial Differential Equations

As pointed out previously, DD techniques provide an important framework to associate mathematical formalism to the parallel solution of a complex PDEs system - see e.g. [20, 21]. The PDE problem over a region of interest Ω is decomposed in subproblems to be iteratively solved by single processors or clusters up to the fulfillment of a convergence criterion stating that the solution found is equivalent to the one of the unsplit system. Each subproblem exchanges information with the neighborhood ones by means of *interface conditions*. In non overlapping splittings, these conditions need to be properly chosen to guarantee that the split-by-subdomain solution is equivalent to the global one. In overlapping partitions, less constraints are required since synchronization conditions for each subdomain are prescribed on different space locations. In fact, each subdomain has its own interfaces. Notice that with overlap the PDE problem is solved multiple times on the overlapping regions, with potential computational duplication overhead. However, beyond the more freedom when selecting the interface conditions, the iterative solver requires in general a lower number of iterations to converge. We illustrate the difference between the two approaches for a simple problem in Fig. 2.

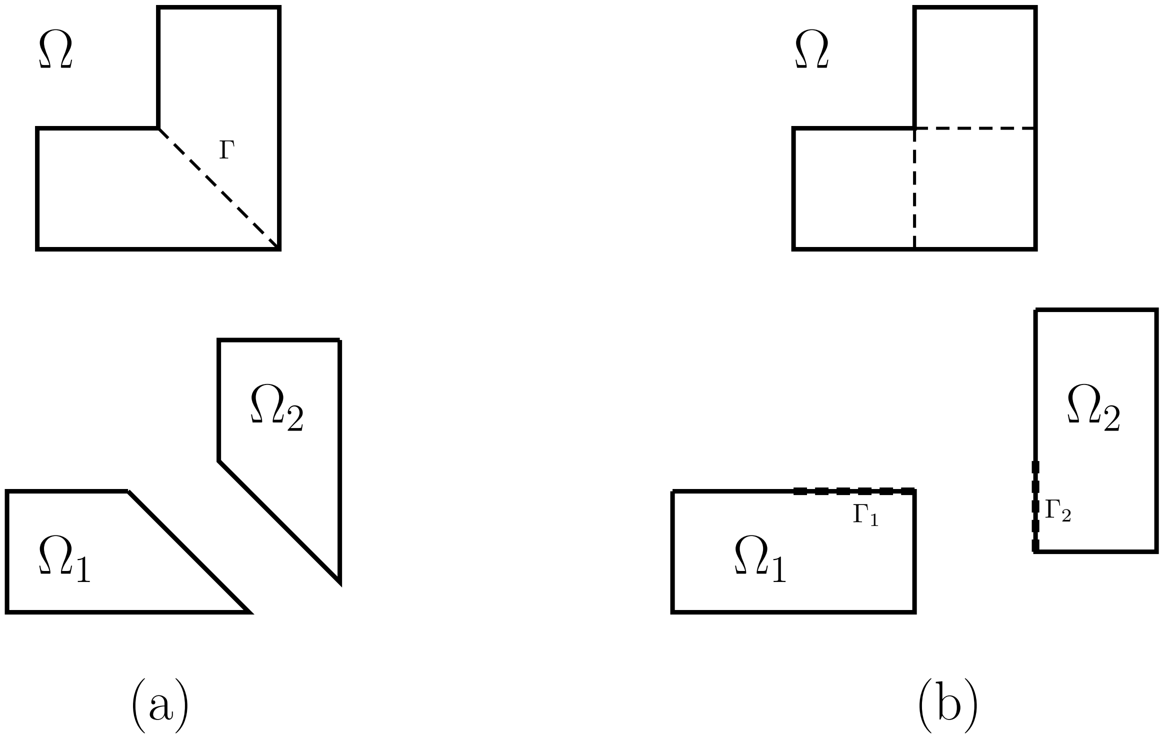


Figure 2: Schematic representation of (a) non overlapping and (b) overlapping DD in a L-shaped domain Ω. In the first case conditions on the interface Γ must fulfill compatibility constraints depending on the nature of the PDE for the split-by-subdomain solution to be equivalent to the unsplit one.

The interplay of (i) additional numerical costs due to the overlap, (ii) efficiency advantages induced by the specific iterative methods and (iii) versatility of the selection of domain interfaces (and the associated conditions) for the communication time, is not trivial in problems of practical interest. In particular, numerical analysis focuses typically on points (i) and (ii), while in the present work we will verify the performances of these overlapping schemes when the geometry of Ω plays a major role, following up previous works [22, 23].

In particular, to this aim we will consider the following differential *Advection-Diffusion-Reaction* (ADR) problem

$$-\sum_{i=1}^{3} \frac{\partial}{\partial x_i}\left(\mu \frac{\partial u}{\partial x_i}\right) + \sum_{i=1}^{3} \beta_i \frac{\partial u}{\partial x_i} + \sigma u = f, \tag{1}$$

for $(x_1, x_2, x_3) \in \Omega \subset \mathbb{R}^3$ with $\mu > 0$ and σ coefficients for simplicity assumed to be constant. Here the unknown u may represent the density of a species in a region where it diffuses with diffusivity μ, it undergoes to a chemical reaction with rate σ and it is convected in the domain by the vector field $\boldsymbol{\beta} = [\beta_1\ \beta_2\ \beta_3]^T$. In particular $\boldsymbol{\beta}$ denotes the blood velocity and it is function of the space coordinates x_1, x_2, x_3. When available, it can be prescribed analytically, as we do in the tests in idealized geometries. More in general, it is retrieved by solving the NSE computed as in the previous Sections. The forcing term f is a given function of space too. Hereafter it will be set to 0 for simplicity. We associate with the equations the boundary conditions $u(\Gamma_D) = g(x_1, x_2, x_3)$, $\dfrac{\partial u}{\partial \mathbf{n}}(\Gamma_N) = 0$, where Γ_D and Γ_N are two disjoint portions of the boundary of Ω such that $\Gamma_D \cup \Gamma_N = \partial\Omega$. This is a simplified model of the dynamics of blood solutes like Oxygen in the arteries [24]. Specifically, we do not consider time dependence, since it does not introduce significant changes for the focus of the present paper. The NSE solution is therefore retrieved in a particular instant of the hart beat, the so called *systolic peak*, corresponding to the maximum opening of the ventricular valve.

To take advantage of domain decomposition, we split the domain Ω into two overlapping subdomains Ω_1 and Ω_2, such that $\Omega_1 \cap \Omega_2 = \Omega_o$ and $\Omega_1 \cup \Omega_2 = \Omega$. Let us denote by Γ_j the interfaces between the two subdomains ($j = 1, 2$), that is the portion of the boundary of Ω_j that is not also boundary of Ω, in short $\Gamma_j \equiv \partial\Omega_j \setminus (\partial\Omega_j \cap \partial\Omega)$. The solution of the problem in each subdomain will be denoted by $u_j(x_1, x_2, x_3)$. We reformulate the original problem in an iterative fashion. Given an initial guess $u_j^{(0)}$ (typically $= 0$), we solve on each subdomain for $k = 1, 2, \ldots$

$$-\sum_{i=1}^{3} \frac{\partial}{\partial x_i}\left(\mu \frac{\partial u_j^{(k)}}{\partial x_i}\right) + \sum_{i=1}^{3} \beta_i \frac{\partial u_j^{(k)}}{\partial x_i} + \sigma u_j^{(k)} = f \qquad \text{in } \Omega_j, j = 1, 2 \tag{2}$$

with boundary conditions

$$u_j^{(k)}(\Gamma_D \cap \partial\Omega_j) = g(x_1, x_2, x_3), \quad \frac{\partial u_j^{(k)}}{\partial \mathbf{n}}(\Gamma_N \cap \partial\Omega_j) = 0, \quad u_j^{(k)}(\Gamma_j) = u_{\widehat{\jmath}}^{(k-1)}(\Gamma_j), \quad (3)$$

(where $\widehat{\jmath} = 2$ for $j = 1$ and $\widehat{\jmath} = 1$ for $j = 2$) up to the fulfillment of the convergence condition. In our case this condition checks that the solution in the overlapping region is not changing significantly along the iterations.

Notice that at each iteration we solve two independent problems in each subdomain, while the communication by subdomain occurs in the latter of boundary conditions (3). The convergence of the iterative scheme depends on the size of the overlapping region. In fact, if the overlapping is 100 % of Ω, convergence is trivially guaranteed as at the first iteration (2-3) we are solving (twice) the unsplit problem. On the other hand, if the overlapping reduces to a volume-zero region, convergence is not guaranteed, as in general the juxtaposition of the two problems does not coincide with the original problem (as pointed out, this occurs only if the interface conditions are chosen properly).

The one presented here is the so called *additive* formulation of the overlapping DD method, where the two subdomain problems can be solved simultaneously - as opposed to the *multiplicative* version, where one subdomain can be solved only when the problem on the other subdomain is completed. In the multiplicative formulation a faster convergence is guaranteed in terms of number of iterations (about one half of the additive scheme), but the advantage of the parallel setting is limited by the sequential structure of the algorithm. From now on, we refer only to the additive algorithm.

The selection of the interfaces Γ_j has the only constraint to guarantee a non empty overlapping. The optimal selection is the result of the trade-off between the computational cost of each subproblem and the reduction of the communication between processors. This will be investigated in Sect. 4.

1.3. Summary of the packages used by the numerical solver

For a more detailed description of the implementation of the numerical solver we refer to [25, 23]. We report here the complete list of required packages:

- LifeV library [10], for the formulation of the algebraic counterparts to differential problems; this library is the direct dependency for our solver application;

- Third-party scientific libraries: (1) Trilinos [26] for the solution of linear systems (data structures and algorithms); (2) ParMETIS [27], used for mesh partitioning; ad hoc MATLAB scripts were prepared to add an overlapping region to an existing non-overlapping partition. (3) SuiteSparse [28], as a support library extending the capabilities of Trilinos; (4) BLAS/LAPACK libraries (generic or vendor-specific implementations); (5) NetGen [29] for generating the mesh to partition.

- General-purpose and communication libraries: (1) Boost C++ libraries [30] 1.44 or above, mainly used for memory management (smart pointers); (2) HDF5 [31], for the storage of large data on file; (3) MPI libraries (e.g., Open MPI);

- Compilers: C++ compiler (e.g., GCC version 4 or above); [optional] Fortran compiler, compatible with C++;

- Deployment tools: (1) GNU make; (2) Autotools; (3) CMake (version 2.8 or above).

2. CFD Experiences on clouds, grids and on-premise resources

Message passing parallel programs are a staple modality of numerical simulations and computational analyses. In addition to the parallel framework (e.g. MPI), codes depend on various other auxiliary components: scientific and mathematical libraries, header files, particular compiler options and flags. These parameters (or subsets thereof) are quite specific to a particular *target platform*; executing the application on different target platforms may require a non-trivial amount of re-building effort (even if the actual application source code is untouched). Hence, applications often continue to be executed only on the default “home” platform, even if other viable and better options are present. However, grids and especially clouds present real opportunities for applications to execute on platforms other than their home environments [32]. In the ADAPT project at Emory, we investigated the feasibility and ease of deploying classes of applications on target platforms other than those on which they normally execute. As a benchmark test, we have experimented LiFEV [10] whose home environment is a 128-core cluster, and ported it on other computational platforms: clusters, grids and Amazon’s EC2 cloud. We conduct both a detailed comparison of platforms based on *utility* of the computational task to the user, function of the wait time and the cost, and further analyze strategies for process mapping when interconnection networks are heterogeneous. [4]

[4] Preliminary results from these exercises were presented in conference papers [25, 33].

User-oriented performance analysis has recently been applied to research on HPC and Grid scheduling strategies. The value that users associate with a completed job is modeled as a *utility function*, with a generally non-trivial dependence on time [34]. The importance of a job to a user can be seen as a function of time, combining an index for the importance of the results and the user sensitivity to delay. It has been shown that a proper job scheduling strategy can significantly increase the performance of HPC systems, measured as the aggregate utility of their users [35, 36]. Several works in the literature discuss an extension to this scenario, in which heterogeneous resources can be discovered and assembled from an arbitrary set of providers. In this case, the *utility* for the user may be defined based on a more detailed analysis of user-specific requirements. For instance, requirements may include the features of the physical resources (memory, processor speed, presence of GPU), presence of installed software or availability of specific services. It is then possible to discriminate between resource providers based on their ability to satisfy the requirements, in full or in part (*partial utility*) [37]. The evaluation of the utility function can be done at runtime, to decide whether or not to dynamically re-distribute resources to obtain an optimal "quality of execution", i. e., an optimal trade off between resource savings and performance degradation [38].

In our approach, the platforms are considered as interchangeable – after the conditioning process. We discuss the effort required to provision each platform with an environment adequate to sustain the user's task. Finally, we identify a basic set of user requirements (minimal cost and minimal execution time of the task) to define a user-based ranking of the tested architectures.

2.1. Heterogeneous Target Platforms

In our study, we compared five heterogeneous computational platforms supporting the parallel hemodynamics simulation. As the starting point for our analyses, we selected the in-house computing cluster `puma`[5] constituting a computational test bed for the LiFEV developer team. As a second platform, we used a larger compute cluster called `ellipse`, provided on a fee-for-use basis within our university. The third platform was the HPC supercomputer `lonestar` made available to the U. S. research community by Texas Advanced Computing Center. Next, we evaluated the usability of on-demand resources. The first such platform was `rockhopper` [39] offered as a part of the Penguin's On-Demand HPC Cloud Service [40] and the second platform was the IaaS cloud provided by Amazon's Elastic Compute Cloud (EC2)

[5]This is the "home" environment where the application is run by default.

	`puma`	`ellipse`	`lonestar`	`rockhopper`	`ec2`
type	cluster	cluster/grid	grid	cloud cluster	IaaS Cloud
cores	2x2	2x2	2x6	4x12	4x4
RAM	8GB	4GB	24GB	2.5GB/slot	66GB
network	SDR IB	1GbE	QDR IB	QDR IB	10GbE
storage	NFS	NFS	Lustre	Lustre	local fs
support	full	very limited	limited	online	none
OS	Rocks 5.1	CentOS 4.8	CentOS 5.5	CentOS 5.6	AMI 12.03
access	user space	user space	user space	user space	privileged
MPI	Open MPI	none	MVAPICH2	Open MPI	none

Table 1: Specification of a single node of the test architectures.

service. From the rich EC2 resource offerings, we picked the most powerful instances `cc2.8xlarge` from *Cluster Compute* (referred to as `ec2` in the following).

The five platforms are heterogeneous in many respects: they differ in hardware configuration, availability (measured as wait-time before execution), access modality (privileged vs. unprivileged user), storage (e.g., size of user disk space and presence of a shared file system), build (e.g., the compilers and system tools availability), computational aggregation (e.g., presence of configured MPI environment), and execution (e.g., interactive shell). Table 1 collects the main features of the chosen targets. We refer to [25, 33] for further details.

2.2. Metrics

We aim to compare different hardware platforms with respect to the execution of the same task, evaluating several different metrics.

As previously mentioned, hemodynamics applications are both time- and cost-sensitive. It is worth noting that optimizing these two aspects separately leads in general to conflicting strategies, as it is often the case that the most expensive hardware resource provides the result in the shortest time, as we will see later on. We therefore consider both traditional metrics ("time to completion" and "cost per simulation"), and a user-specific combination of these two, corresponding to the "perceived cost" of the computational experiment.

Time to completion. This is the wall clock time from program launch to final exit. In the mainstream HPC community, in which it is a primary focus, it is not common to include the queue waiting time. In terms of utility in the sense adopted in this work, queue time is certainly important but it is highly variable and, in fact, our platforms presented few queue delays compared to the execution time of the application. We decided to exclude queue time in our analysis for the sake of simplicity and uniformity, especially since "on-demand" IaaS clouds are practically characterized by zero waiting time.

Cost per simulation. The overall cost for the execution of the job depends mainly on the unit cost of the hardware resource (cost per core-hour), its pricing policy (by core or by node, by hour or prorated), and on the execution wall-clock time. Other factors, that we consider negligible relative to the former (for our application), are the size of occupied storage and/or volume of data staged in and out.

Utility function. The *utility function* expresses the job's value to a user, as a function of time. This has a user-specific, complex dependency on several parameters, including expenditures, time to completion, and significance of the task. Following [36, 41], we consider a simple linear utility function with customizable maximum (starting) value and slope, as shown in figure 3. The equation reads

$$U(t) = \begin{cases} U_{max} & \text{if } t \leq T^* \\ U_{max}\left(\dfrac{T_0 - t}{T_0 - T^*}\right) & \text{if } T^* < t \leq T_0 \\ 0 & \text{if } t > T_0. \end{cases}$$

U_{max} is a measure of the *importance* of the job to the user, and we assume that it can be given a monetary value, as the price that the user would be willing to pay for the simulation. T^* is the expected completion time, which can be estimated in several ways. We use a simple averaging method defined in section 2.3.2, based on the performance of the available platforms. T_0 is the user-defined time at which the utility is zero, while the distance $(T_0 - T^*)$ is a measure of the user's *delay tolerance* and can be measured as a multiple of the expected completion time T^*.

With this formulation, we assume that there is no loss of value during the expected duration of the job (when $t \leq T^*$). An extension of the model could take into account the decrease in the utility function during runtime, reflecting the fact that faster runtime is valuable to users [34].

2.3. Experimental Results

Our experiments on different architectures yielded interesting results. This discussion centers on cost and utility.

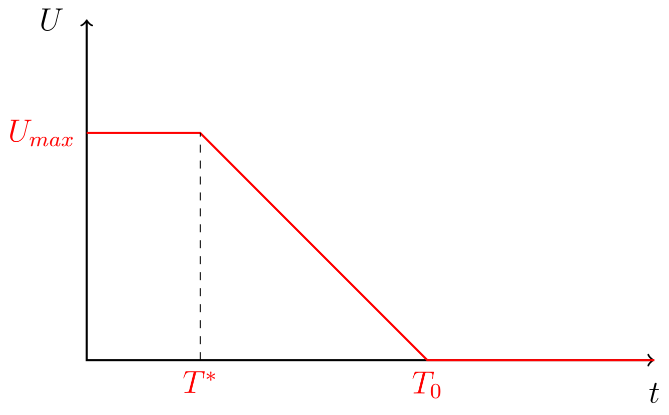


Figure 3: The considered utility function. U_{max} is a measure of the *importance* of the job to the user, T^* is the expected completion time, T_0 is the time at which the utility is zero [36].

2.3.1. Performance, scaling and time to completion

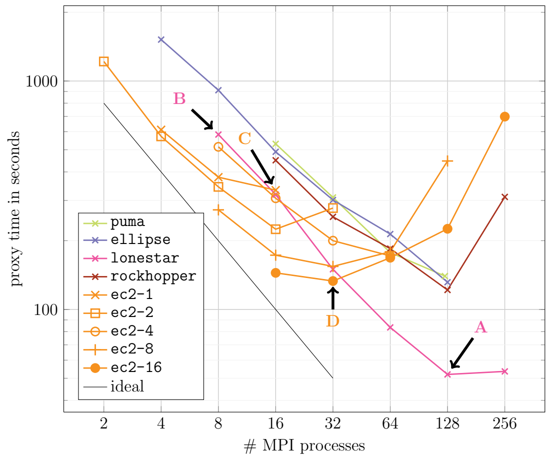
In our study we tested the selected platforms executing a fixed-size simulation (over 3.1M unknowns) with varying numbers of processors, i.e., a strong scalability benchmark. All tested clusters allowed the reservation of computing resources by specifying the number of processes (or slots) used by the parallel job. However, in the case of the Amazon EC2 cloud, we needed to set the execution policy: we assumed that each `ec2` instance can host a maximum of 16 processes (as they have 16 physical cores) and we decided to map the MPI processes onto the physical nodes in round-robin fashion. As Amazon charges users on the basis of running instances, we decided to optimize the cost of the benchmark by testing small assemblies of `ec2` first, and then to increase the number of nodes in the assembly by powers of 2. For this reason, we present several configurations of cloud instances; we label such separate assemblies as `ec2`-i, where i is the number of `ec2` nodes.

The application repeats the same set of operations in each simulated time frame (in our case corresponding to 0.01s intervals). For each considered hardware platform, the time required to compute a single frame was observed to be constant during the course of the simulation. We, therefore, use the average computing time for a single frame as a proxy for the performance of the hardware resource. This facilitates a side-by-side comparison of all platforms, including cases when the simulation could not be completed due to cluster usage policies (e.g., `ellipse` limits the job execution time to 12 hours so for jobs that spanned small numbers of cores only a fraction of the entire simulation could be done).

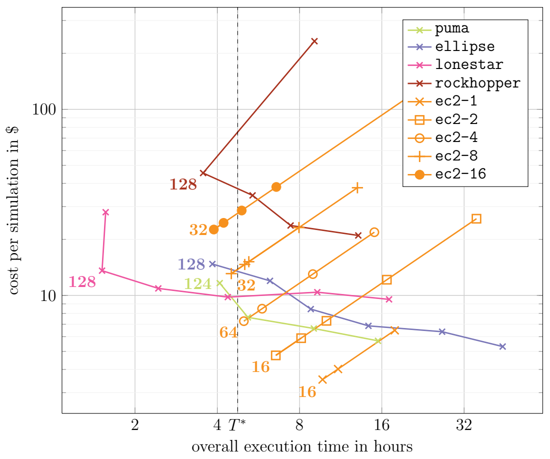
The graph in Figure 4a shows a comparison of the performances of the different platforms, as a function of the number of computing cores. The clusters `puma` and `ellipse`, the grid `lonestar` and the cloud cluster `rockhopper` achieve good strong scaling up to 128 computing cores, while they show a significant decrease in performance for larger numbers of cores. In particular, Point **A** in the figure corresponds to the fastest execution case in our experiment, that is running the simulation with 128 computing cores on `lonestar`. If this metric is used to represent utility or value to the user, it is clear that when using more than 32 cores, `lonestar` is the best platform.

`ec2` resources scale less well. `ec2-1` achieves good scaling only in the range 4-8 cores, `ec2-2` up to 16 cores, `ec2-4` up to 32, `ec2-8` up to 16 cores, while `ec2-16` does not achieve strong scaling in any range. Point **C** in Figure 4a corresponds to the case when the simulation was sustained by 16 computing cores on a single `ec2` instance. It is worth noting that the time to completion in this case matches the time to completion obtained using 16 computing cores on `lonestar`. Most significantly, the time to completion required by `ec2-1` when using 8 computing cores is lower than the time required by `lonestar` with the same number of computing cores. This result suggests that one of the advantages of IaaS clouds is the availability of powerful hardware configurations (both in terms of memory and CPU clock speed), that can match and outperform the computing nodes provided by standard grid resources. This finding is in agreement with previous reports. A study [42] pointed out that when an EC2 user reserves an entire computing node (this happens in our case using `cc2.8xlarge` instances) the impact of virtualization is negligible since processor cores are not shared among users (see also [43, 44]). This results in performance comparable to "bare metal" hardware. As predicted by Iosup and coworkers [45], this is a significant advantage of *Cluster Compute* instances over former offerings by EC2, that were suffering from performance degradation due to concurrency of multiple users or applications using the same processor. On the other hand, the performance of `ec2` platforms seems to be sensitive to overload of the instances, as shown by the poor strong scaling achieved by `ec2-1` when all of the 16 available computing cores are used for the execution of the simulation.

Point **D** corresponds to the fastest execution on `ec2` resources, that is running the simulation with 32 computing cores using `ec2-16`. In this case, we launched 16 EC2 instances and allocated 2 computing cores on each instance in a round robin fashion. The loss of performance of `ec2-16` as the number of cores per instance increases suggests that *when requiring a relatively large number of instances, the physical connectivity of the nodes may become an issue, and the timings seem to be dominated by communication overheads*. The severe impact of network latency and bandwidth on EC2 performance is a known issue, especially for large assemblies of instances [43, 44, 45]. In terms of utility, therefore, individual EC2 nodes offer high performance but when communication across nodes or racks is involved, this platform is less attractive.



(a) The average computation time per simulated time step (proxy), for the benchmarked architectures



(b) The labels indicate the number of MPI processes in the fastest run. $T^* = 4h\ 44m$ corresponds to the average value among the fastest runs.

Figure 4: The average computation time per simulated time step (a) and the relation between the cost and time of the simulation (b).

Based on the metric *time to completion* we can rank the different resources – Table 2 shows the wall clock times for the fastest run on each platform. The grid `lonestar` is by far the fastest resource, while `ec2` is generally the slowest. However, one of the solutions provided by `ec2` (namely `ec2-16`) matches the performance of the clusters `ellipse` and `puma`, using a significantly smaller amount of computing cores. This result further demonstrates one of the strengths of on-demand resources as compared to on-premise resources, i.e., `ec2` can count on a more efficient hardware configuration. The cloud cluster `rockhopper` performs best among the tested on-demand resources and better than the tested on-premise resources. However, it is still significantly slower than the tested HPC cluster.

2.3.2. Utility function

Ideally, users desire to minimize both simulation cost and time to completion but these objectives compete with each other. This is confirmed by our tests where the cheapest resource, namely `ec2-1`, was also the slowest one. In Figure 4b we

rank	time to completion [s]	target	# of MPI proc.
1	1h 31m	`lonestar`	128
2	3h 33m	`rockhopper`	128
3	3h 50m	`ellipse`	128
4	3h 53m	`ec2-16`	32
5	4h 05m	`puma`	124*
6	4h 30m	`ec2-8`	32
7	5h 00m	`ec2-4`	64
8	6h 33m	`ec2-2`	16
9	9h 43m	`ec2-1`	16

Table 2: The performance ranking of the hardware resources based on the metric *time to completion*. * One node is permanently down.

present how the cost per simulation relates to the time to completion for the different architectures. The closest points of the graphs to the origin of the axes represent execution cases that minimize both metrics. Clearly, the decision on which architecture to prefer cannot be made based on a single attribute. The general trend of these characteristics shows an increase in the cost per simulation with the decreasing time to completion. A remarkable exception is `ec2`, for which cost increases with time to completion. In fact, on this platform slower executions achieved with few computing cores are actually more expensive due to the policy requiring the reservation of 16-core instances.

To define a ranking of the tested platforms based on a user-centric performance analysis we evaluate the utility function defined in Section 2.2. We consider three user profiles,

Case 1. The job has high priority, and the user has little delay tolerance;

Case 2. The job has average priority, and the user has average delay tolerance;

Case 3. The job has low priority, and the user has large delay tolerance.

Referring for the sake of example to the results of our benchmark, we assume that the value of a simulation to the user (i.e., the cost the user would be willing to pay) is in the range between \$3.53 (low) and \$22.59 (high). More precisely, we assume that a job with low priority has a value to the user equal to the average cost of the simulation over the tested architectures, i.e., \$10.31. We assign double this value to a high priority job (\$20.62) while an average priority job will have an intermediate value between the previous two (\$15.465). We further assume that for all the user profiles the expected time to completion T^* is the average value of the times measured on the different architectures (cf. table 2), i.e., $T^* = 4\text{h } 44\text{m}$. A user with an average delay tolerance is represented by a utility function that remains non-negative for a runtime up to twice the expected value (i.e., $T_0 = 2T^*$). A user with large delay tolerance accepts twice as much delay ($T_0 = 3T^*$), while a user with small delay tolerance accepts half as much (i.e., $T_0 = 1.5T^*$).

We plot in Figure 5 the user-specific utility functions together with the graphs shown in Figure 4b. As discussed in previous sections, each platform was tested in several use cases (varying the number of computing cores); a case is considered *useful* to the user if it is represented by a point on the cost/time plot located below the graph of the user's utility function. For the sake of example, we reported on the plot the points corresponding to the cases discussed in detail in the previous sections. Point **A** corresponds to the fastest execution of the simulation in our experiment, obtained when using 128 cores on `lonestar`. This case is *useful* to user profiles 1 and 2, for which the *importance* of the simulation is greater than the actual cost. User profile 3 would not consider this case *useful* due to its high cost.

Despite the cost being relatively lower, the use case of `lonestar` represented by point **B** (8 computing cores) is not *useful* for any user profile, because for all of the profiles the time to completion of the simulation exceeds the time T_0 for which the utility function is zero. The use case of `ec2-1` corresponding to the cheapest execution in our experiment (16 computing cores) is represented by point **C**; because of the long time to completion, this use case is only *useful* to user profile 3. The fastest execution achieved on `ec2` resources (2 computing cores on each instance of `ec2-16`) is represented by point **D**. This use case has a cost exceeding the maximum value of the utility function for all the user profiles, so it is *useful* to none of them.

In our experiment, a variety of platforms can meet the requirements of user profile 1. Fast and expensive architectures (e.g., `lonestar`) can be chosen in alternative to slower and cheaper ones (e.g., `ec2`). However, because of a small delay tolerance, a cheap option (`ec2-2`) has to be ruled out, being penalized by high execution times. The second user profile has the largest pool of *useful* choices, including the cheaper (and slower) `ec2-2`. For user profile 3, because of the low priority assigned by the user to the job, most of the fastest options (`lonestar`, `ellipse`) have to be discarded. On the other hand, the on-premise cluster `puma` and some of Amazon's instances (most significantly the very cheap `ec2-1`) can meet the user's requests.

According to this model, one of the on-demand resources (`rockhopper`) is not *useful* to any of the considered user profiles. Amazon's diverse offering allows instead this service to be competitive for a wide range of user profiles, being able to provide reasonably small execution times (`ec2-8`) or extremely cheap solutions (`ec2-1`).

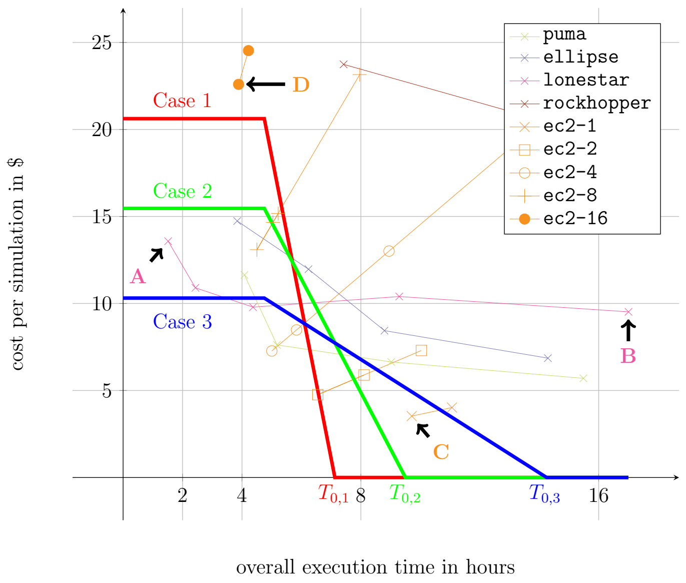


Figure 5: Evaluating the cost/time characteristics of the different platforms against the user-specific utility function. $T_{0,1}$, $T_{0,2}$ and $T_{0,3}$ are the times at which the utility function is zero for user profiles 1, 2 and 3, respectively.

On-premise resources do not perform well compared to HPC machines, being significantly slower, and in most cases they are also outperformed by cheaper on-demand resources. As a result, they are competitive only in specific execution cases (i.e., with the proper choice of the number of computing cores). Finally, `lonestar` is a very strong competitor in the first two user scenarios (average to high job priority), while its performance is matched and outperformed both by on-demand and on-premise resources in the third scenario (low job priority and high delay tolerance).

Notably, the on-demand cutting edge offering by Amazon EC2 has the advantage of availability. In fact, our analysis does not consider queue waiting times that may diminish the attractiveness of shorter execution time on grid resources. This feature would make the IaaS choice even more convenient. Moreover, the cost per simulation on the resources offered by Amazon can be optimized with a proper scheduling policy that takes into account the specific pricing policy of Amazon (per-node rather than per-core). Furthermore, if cost needs to be minimized, it is possible to select cheaper Amazon instances such as `cc2.4xlarge`.

3. Adaptive mapping of parallel components on physical resources

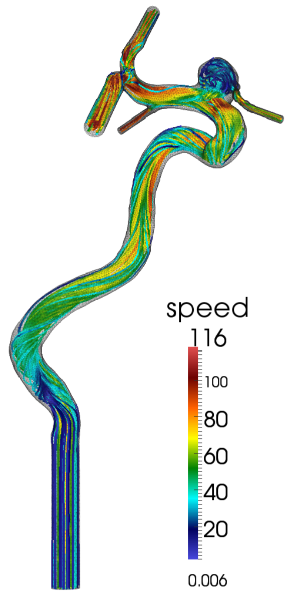
Improving the layout of the tasks of a parallel application on a particular hardware architecture is an attractive research subject as it may increase the performance of the application without requiring modifications to the source code. The advantages may be particularly significant if the supporting computing machines are high performance clusters and the optimized placement harnesses the capabilities of cutting edge network solutions. Rubik [46] is a software toolkit that applies simple geometry transformations (e.g., splits, tilts) to the Cartesian task topology of an application, altering its mapping to the Cartesian network topology. Thanks to the tasks shuffling, the underlying hardware may better support MPI collective operations by utilizing more hardware links while avoiding excessive latency or congestion [47]. Our solution follows a similar approach: we design the layout of MPI tasks before we execute the application. However, as our hemodynamic CFD code mainly uses point-to-point communication and has no statically defined communication topology we need to analyze a data exchange graph for a particular execution use case in order to optimize the tasks mapping.

A successful mapping strategy has to consider also the properties of the network backend. Eliminating unnecessary network hops may improve the overall latency and lead to better performance of the executed application. The project described in [48] considers the homogenous, multilevel IB network and offers an improved MPI implementation that exploits the network topology to increase intra-node communication and reducing the long distance inter-node communication. While this is an end point also for our project, we do not force a different MPI framework implementation. Moreover, even though currently we consider a simple, single hop network topology, we propose methods that can be extended to different scenarios. In particular, when considering wider networks, a more aggressive planning of the mapping can be applied, aggregating machines from even geographically separated data centers to provide the computational platform for the distributed applications. The possibility of the inter-cloud aggregation and the performance of such conglomerate were evaluated by two of the authors in [49].

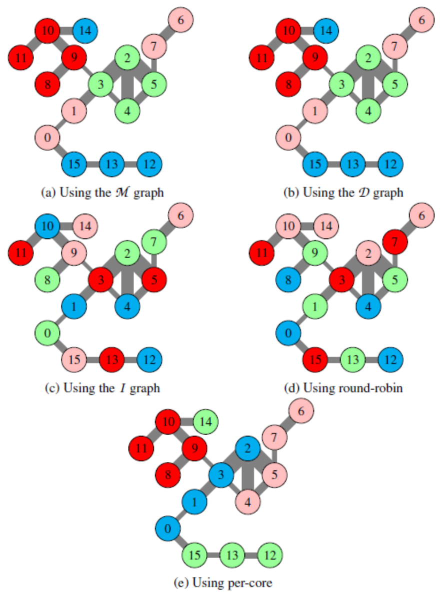
3.1. Test case

The model problem used in our experiment is blood flow in an internal carotid artery affected by a saccular brain aneurysm. Aneurysms are localized dilations of the arterial wall, often times in the form of a blood-filled sac. They may rupture, causing severe brain damage and even death. Fluid dynamics is considered one of the risk factors that may help predict the outcome of the disease [8, 18].

We consider a subject-specific arterial geometry, extracted from medical images acquired and processed during the multi-center research project Aneurisk [50]. This kind of geometries are available for download through the web portal AneuriskWeb (`http://ecm2.mathcs.emory.edu/aneuriskweb`). To compute blood velocity and pressure in the subject specific geometry we solve numerically the NSE equations, using LiFEV. We simulate blood motion under pulsatile flow conditions, representing the pumping action of the heart. Blood is described as a Newtonian fluid with density 1 g/cm3 and dynamic viscosity 0.035 dyn/cm2. For the sake of the analyses presented here, we limit our simulation to a short time interval (0.10 seconds), solving the discretized NS equations at 10 instants (i.e., the simulation time step is $0.01s$). A snapshot of the computed solution is shown in Fig. 6a.



(a) Solution of Navier Stokes Equations for blood flow in an aneurysmatic vessel, for t = 0.05s. Streamlines of the velocity field colored by the blood speed.


(a) Using the $\mathcal{M}$ graph
(b) Using the $\mathcal{D}$ graph
(c) Using the $\mathcal{I}$ graph
(d) Using round-robin
(e) Using per-core

(b) Different mappings of the coarse mesh for four 4-way nodes. Each color represents a part for a single host. Edge thickness represents the number of common vertices between parts. In this case, the partitioning using M and D are the same.

Figure 6

3.2. Offline mesh partitioning

The global mesh consists of the set of all elements, faces, edges and vertices in the tessellation. To each of those entities a unique identifier (global id) is assigned. We will denote by N_{el}, N_f, N_{ed}, N_v the total number of elements, faces, edges, and vertices in the global mesh. The topology of the mesh is described by the relationship between different geometric entities and can be expressed in table format (connectivity tables). For instance, the element-to-face table B_0, with size $N_{el} \times N_f$ is such that

$$(B_0)_{ij} = \begin{cases} 1 & \text{if face } j \text{ belongs to element } i \\ 0 & \text{otherwise.} \end{cases}$$

We have similar definitions for the face-to-edge table B_1, with size $N_f \times N_{ed}$, and the edge-to-vertex table B_2, with size $N_{ed} \times N_v$. Other connectivity tables can be obtained by composition of these.

In the parallel application, the computational domain may be partitioned by subdomains so that each process takes care of only a subset of the global mesh. In this section we consider non overlapping partitions and we refer to these subsets as "local" meshes. The splitting is achieved through the use of graph partitioning algorithms, such as those implemented in the libraries ParMETIS or Scotch, guaranteeing a proper load balancing among processes. The load is measured as the number of mesh elements assigned to each process. When local meshes are not overlapping, each element belongs to one and only one process; however some faces, edges, and vertices are shared among two or more local meshes (interface entities). In any case, high quality partitionings should minimize the edgecut or the number of connections between disjoint partitions. This property is valuable to reduce the communication between processes necessary to synchronize interface unknowns.

For large scale simulations, mesh partitioning is a highly memory intensive operation due to the size of the global mesh and it is usually performed offline on dedicated machines since many times memory on computational nodes is a limiting resource. In more detail, in [51] the following strategy for mesh partitioning was considered.

1. The element adjacency graph A is built from the topological information stored by the mesh as $A = B_0 \wedge B_0^T$. Here and in the following we denote by the symbol $\wedge$ the Boolean multiplication operator between tables. The element adjacency graph is an unweighted symmetric graph such that two elements of the mesh are connected by a link if they share a common face.

2. The element adjacency graph A is partitioned in n_p connected components by using the recursive bisection multilevel partitioning algorithm implemented in

ParMETIS, where n_p is the number of desired processes to run the simulation. The result of the partitioning algorithm is a vector p of integer numbers of length N_{el}, in which the value of entry i ($0 \leq i \leq N_{el}-1$) specifies the partition element i was assigned to.

3. The global mesh is split according to the partitions of the elements induced by p. By introducing the Boolean table P, of size $n_p \times N_{el}$,

$$(P)_{ij} = \begin{cases} 1 & \text{if } p[j] == i \\ 0 & \text{otherwise} \end{cases}$$

the local mesh corresponding to process i is associated to its own set of elements, faces, edges, vertices evaluating the non-zeros entries of the i-th row of the matrices $P_f = P \wedge B_0$, $P_{ed} = P_f \wedge B_1$, $P_v = P_{ed} \wedge B_2$, respectively. The above matrices are also used to define proper mappings between the local meshes and the original global mesh, while local connectivities tables B_i^l are obtained by extracting the appropriate rows and columns from the global tables B_i.

4. Finally, the partition connectivity graph M is computed. This is used to estimate the communication volume due to synchronization of the variables associated to interface entities. In particular M_{ij} is proportional to the number of variables shared by processor i and j, i.e., to the number of shared faces, edges and vertices. Thus, we have $M = \alpha_f P_f P_f^T + \alpha_{ed} P_{ed} P_{ed}^T + \alpha P_v P_v^T$, where $\alpha_f, \alpha_{ed}, \alpha_v$ are constant values expressing the number of unknowns associated to each face, edge, and vertex, respectively. These constants depend only on the polynomial degree of the finite element basis.

3.3. Evaluation procedure and results

To determine the performance of the different process placement scenarios, all simulation configurations were tested, i.e., three mesh resolutions, from 8 to 128 MPI processes, for three target architectures, using five different placement strategies (total 45 benchmark tests). The first three allocation techniques were formed using the information in the communication graph D, M, and an additional inverted communication graph I defined as $I_{ij} = M - M_{ij}$, where M is the maximal entry in M. We will refer to these strategies as *data*, *part* and *invert*, respectively. The partitioning of these graphs was performed using the `gpart` tool from the Scotch 6.0 software package, that implements graph k-way partitioning heuristics. As a result, we obtained three clusterings C_D, C_M, and C_I , such that each cluster included the same number of parts equal to the number of processing units available in a single node of the target machine and the graph cut-sets were minimized. C_D gave us the process placement optimized to the actual communication statistics. C_M was the mapping optimized with respect to the partition connectivity graph M that is a by-product of the partitioning algorithm and does not require any additional work. However, given the strong correlation between D and M, similar results were expected in these two cases. Finally, using C_I, the worst placement choice was expected.

The second group of allocation strategies were methods commonly used to execute parallel applications with the OpenMPI: by-node round-robin (*rr*) and by-cores (*pcore*) placements. The first allocation strategy places processes one per node, cycling by node in a round-robin fashion, while the second uses all CPU cores on one node before moving to the next node (Fig. 6b). All tests were repeated twice and the data were averaged. Figure 7 shows the execution times for different MPI process placements for all configurations relative to the maximal execution time for that configuration (i.e., a chart bar having execution time 1 represents the least effective process placement in the configuration). A detailed description of the results can be found in [51]: a brief summary follows.

As expected, the execution times for different MPI process placements for all configurations demonstrate that deliberate MPI process placement significantly influences the overall performance of the application. In specific situations, the worst mapping in the configuration is almost one order of magnitude slower than the fastest. The standard *pcore* placement mapping is well-suited for processing the CFD application implemented with the LiFEV library. The reason for this behavior has its source in the implementation of the ParMETIS partitioning used by the application. ParMETIS uses recursive bisection, which matches the common 2^i-way multiprocessing architecture of contemporary computers. However, this allocation may be improved by the resource-oriented *part* placement, especially for larger numbers of hosts performing the computation. Moreover, to design such enhanced placement no extra computations - for instance evaluating communication statistics - are needed. The by-product information regarding the pairwise shared mesh entities from the partitioner phase can be utilized instead. As a result, this shows that it is possible to adapt performance of parallel MPI applications by communication-aware placement of their MPI processes if the computation structure, input geometry as well as target architecture and network wiring is known and it may be done automatically by ADAPT.

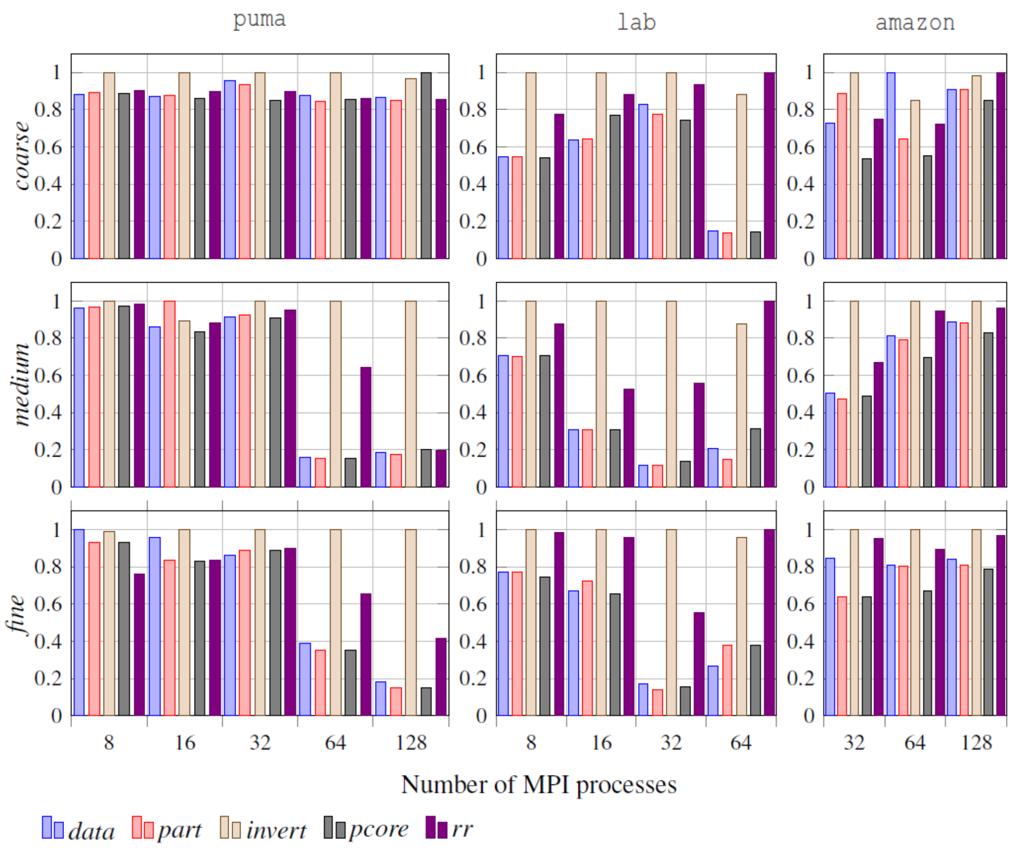


Figure 7: Relative execution times for all simulation configurations. Each value represents the speed-up of the mappings in relation to the less efficient mapping for the configuration.

4. Experimental optimization of parallel 3D overlapping domain decomposition schemes

In the previous section we dealt with process placement. Here we notice an additional opportunity to improve the overall performances on the algorithmic side by resorting to DD techniques. As pointed out, DD methodology relies on the segregated solution of the problem of interest on each subdomain and the iterative synchronization of the partitioned solutions. The advantage of non overlapping splittings is that each local problem is smaller, so we may expect a faster solution of each local solver. On the other hand, the final solution is the result of the iterative synchronization and the number of iterations depends in general on the shape of each subdomains, the interface conditions and the way the subdomains are synchronized. In this respect, the introduction of overlap introduces generally some advantages, since the number of iterations can be reduced. In addition, the interfaces can be positioned in a more flexible way, so to reduce the communication times. We investigate these aspects in a series of tests with different geometries.

In our tests we consider only the iterative-by-subdomain solution in the computational time. Meshing, partitioning and matrix assembly are not included in this analysis, since they are off-line costs that do not depend on the specific solution procedure. The time $T_{it}^{(k)}$ of each iteration (k) is computed as the maximum of the two parallel subdomain solution times $T_j^{(k)}$,

$$T_{it}^{(k)} = \max_{j=1,2} T_j^{(k)}.$$

The single processor time is given by the time for solving the linear system added by the communication time to read from the other processor the last of conditions (3), $T_j^{(k)} = T_{j,sol}^{(k)} + T_{j,com}^{(k)}$. For this particular problem, the computational cost per iteration is constant (denoted by $\overline{T_{sol} + T_{com}}$) , so we get

$$T = \sum_{k=1}^{N_{it}} T_{it}^{(k)} \approx N_{it}(\overline{T_{sol} + T_{com}}).$$

If we denote by p the percentage of overlap in the domain splitting (i.e. the ratio of the volume of the intersection of the domains to the total volume of the geometry), theory of overlapping DD proves that N_{it} *decreases* with p, T_{sol} increases with p while T_{com} depends on the position of the interfaces (precisely on the number of vertexes of the mesh on the interface), so it may change with p in an unpredictable way for a complicated geometry. We therefore expect that the value p has a major impact on the solver performances depending on the different geometries. It is worth noting that the total cost is a function of the mesh size N too. In this case, both factors N_{it} and $\overline{T_{sol} + T_{com}}$ get larger with h, as a price to pay to the improvement of the accuracy of the approximated solution achieved in this way.

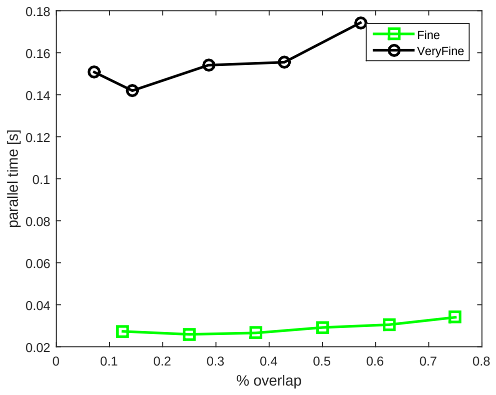
In [23] we have presented several test cases in both academic and nontrivial geometries for a symmetric diffusion reaction problem (i.e. for $\beta_1 = \beta_2 = \beta_3 = 0$). The results point out that a small overlap, symmetric with respect to the non overlapping partition originally determined by ParMETIS, guarantees the best performances in terms of computational time. Here we investigate the same test cases in the more general cases of ADR, when the presence of the "directional" term weighed by $\beta_1, \beta_2, \beta_3$ is expected to have an impact on the detection of the optimal overlap.

We considered the following test cases.
Idealized geometries
(1) *Cylinder* - We consider a cylinder of length $L = 6cm$ and radius $R = 0.5cm$. The coefficients μ and σ are set as in [23] and the convective field has been selected to be constant throughout the domain. We use five meshes with different level of refinement, for each of the sizes of the overlap. We comment only the simulations we ran on the fine and very fine meshes as these are the cases of practical interest.
(2) *Idealized Aneurysm* - We consider an idealized representation of a cerebral aneurysm where a torus with radius 2cm is merged with a sphere of radius 0.5cm, representing the sac of the aneurysm. This test emphasizes the role of communication time. In fact a splitting with an interface intersecting the sac has more vertices than with interfaces involving only the artery. Overlapping DD allows to manage the location of the interfaces so to avoid many vertices on the interface yet preserving workload balance between the subdomains. For more details, see [23]. The convective field has been selected to be tangential with respect to the centerline of the torus and constant in modulus.

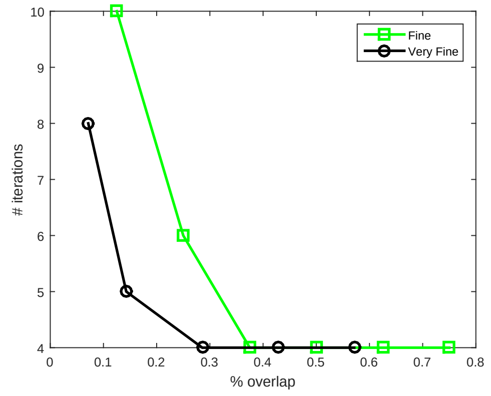
Real geometry We consider the real morphology reported in Fig. 1. The convective field $\boldsymbol{\beta}$ is given by the solution represented there.

4.0.1. Numerical results

Cylinder. Figure 8a shows the parallel running time as a function of p. The varying dependence of number of iterations and cost per iteration on p results in a convex behavior of the computational time. This behavior is expected, since for small p the high number of iterations dominates the cost, while beyond a certain value it does not decrease any longer, while the cost per processor increases. When compared with the similar tests presented in [23], we notice that the performances are not significantly affected by the presence of the convective field $\boldsymbol{\beta}$. Precisely, the optimal size of the overlap is achieved at the same percentage as the one obtained for the problem with $\boldsymbol{\beta} = \mathbf{0}$, around 25% and 15% for a fine and very fine mesh, respectively. In fact the simplicity of the domain makes the presence of the convective field not relevant for the DD iterations. Nevertheless, it is remarkable that the solution of the problem on the finest mesh (black line) takes fewer (or the same) iterations than those needed for a coarser grid (Figure 8b). Indeed, a higher concentration of nodes on the interfaces allows a more detailed exchange of information between the two partitions through the enforcement of the interface conditions and, consequently, it boosts the convergence speed by reducing the number of iterations.



(a) Parallel time as a function of the overlap.



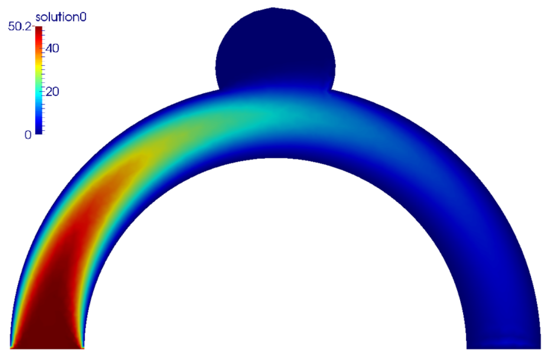
(b) Number of iterations of the parallel solver.

Figure 8: Parallel time performed as a function of p for two levels of refinement of the mesh for the solution of an ADR problem on a cylinder (a). Corresponding number of iterations for fine (□) and very fine (○) meshes (b).

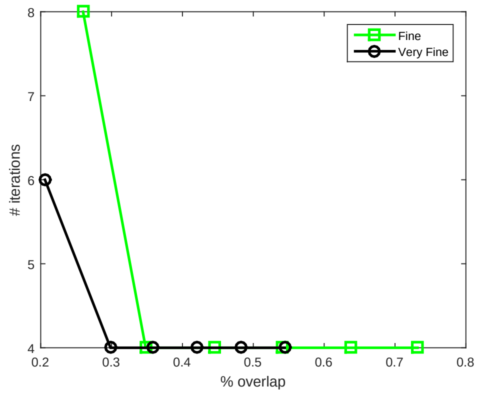
Idealized Aneurysm. Figure 9d shows that the curve related to the very fine mesh features a minimum at a fraction of overlap of $\sim 30\%$, like in the advection-free case [23], i.e., the trend of the estimated parallel time is still invariant with respect to the presence of the convective field.

On the contrary, it is interesting to notice that if we do not have a physical convection ($\boldsymbol{\beta} = \mathbf{0}$) the minimum of the curve for a coarser grid happens at a larger size of the overlap (45% vs. 35%, see Figure 9c). Numerical performances are here explained by physical arguments. In fact the solute concentration u at the inflow is convected through the domain, determining a rapid exchange of information that accelerates the convergence by subdomains (see Figure 9a-9b).

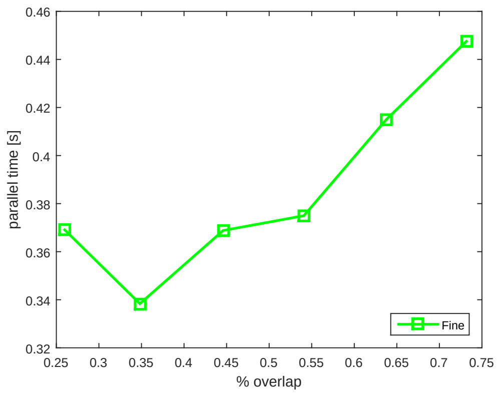
Real Aneurysm. In this case the geometry is twisted and the convective field reproduces the real blood flow into the vessel. As we can see from Figures 10a-10b, the



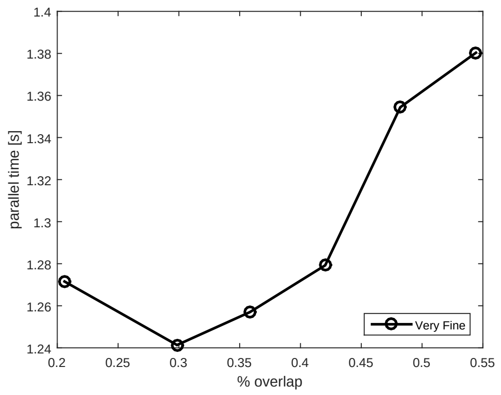
(a) Solution on a slice of the domain.



(b) Number of iterations for a fine (□) and very fine (∘) mesh.

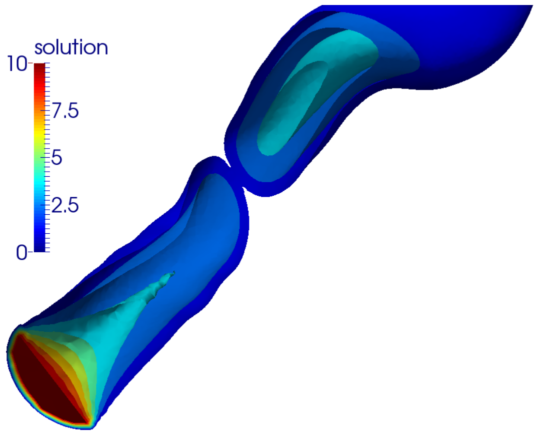


(c) Parallel time for a fine mesh.

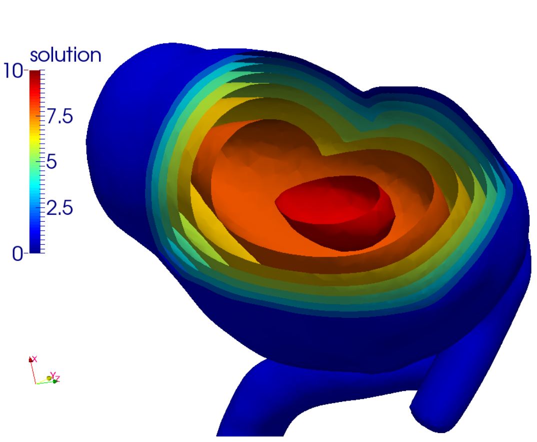


(d) Parallel time for a very fine mesh.

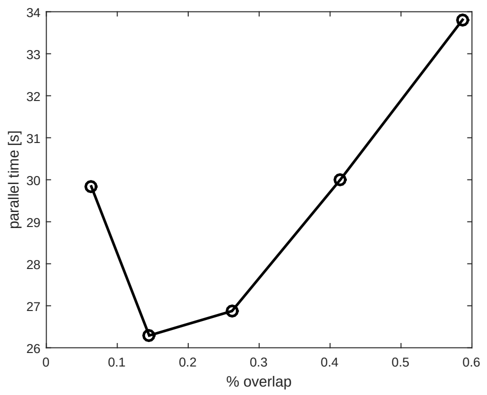
Figure 9: Solution to an ADR problem on an idealized aneurysm (a) and number of iterations (b). Parallel time performed as a function of p for a fine (c) and very fine (d) mesh.



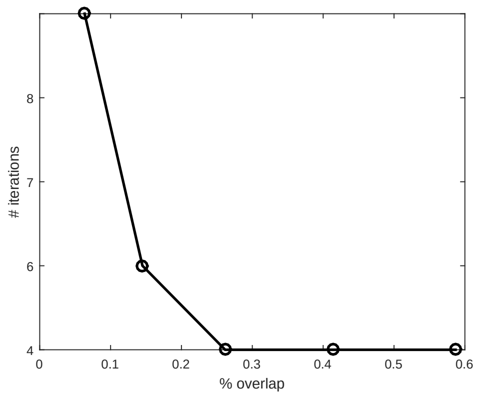
(a) Contour surfaces at inflow.



(b) Contour surfaces in the aneurysm.



(c) Parallel time as a function of the overlap.



(d) Number of iterations.

Figure 10: Solution to an ADR problem on a real aneurysmatic vessel (a-b). Parallel time performed as a function of p for a very fine mesh (c) and number of iterations (d).

% overlap	DoF0	DoF1	InterNodes0	InterNodes1
5%	86,265	85,096	2046	1491
15%	93,255	89,009	3358	1721
25%	104,465	93,791	5068	1947
40%	118,632	97,760	6161	1584
60%	135,228	101,729	6897	1646

Table 3: Number of nodes of each partition (DoF0, DoF1) and total number of nodes on the interfaces (InterNodes0, InterNodes1) for different levels of overlap on a very fine mesh for Test 5.

solute at the inflow section is convected through the vessel and it stagnates into the aneurysmatic sac. Table 3 shows the increment of degrees of freedom when the overlapping is extended. This number is proportional to the computational cost required for solving each subdomain. On the other hand, for the different partitions we have a different number of nodes at the interface, depending on the position of the two cuts. For instance, passing from 25% to 40% the number of nodes at the interface of partition 1 decreases. As the optimal trade-off between the reduction of the iterations attained by a larger overlapping (that however does not improve after 25%), the increased computational cost per subdomain and the communication cost, results indicates 15%. This actually yields a well balanced load for the two subdomains, and a total number of interface degrees of freedom of about 5,000, even if the total number of iterations is 6 vs the minimum of 4 reached with 25% overlapping or more (see Figure 10c-10d).

5. Conclusions

High Performance Computing (HPC) quantification of dynamics traditionally described more in empirical qualitative terms is expected to bring strong improvements for understanding and optimizing processes with a major impact on industry and society. A well established example is medicine and cardiovascular sciences in particular. Numerical analysis of patient-specific settings is becoming a consolidated tool for clinical routine. This allows to improve the level of knowledge available to medical doctors thanks to mathematical models and numerical tools that compute quantities difficult or impossible to measure and overall to enhance the reliability of measures.

However, the intrinsic complexity of the dynamics of interest - reflected by complicated systems of Partial Differential Equations - the constraining timelines of the clinical routine as well as the large volumes of patients typically needed by clinical trials rise formidable challenges in terms of computational resources. Traditional local clusters may be not adequate to afford the computational requests and alternative solutions like grid/cloud resources on demand need to be deeply evaluated. The performance evaluation of these resources is however much more complex for the variety of user needs and availability scenarios that may present. A general recipe for the identification of the optimal strategy is currently out of reach. Nevertheless, in this paper we aim at presenting the results of years of experience in a vital environment like Emory University, where mathematicians and computer scientists routinely assist medical doctors in their daily activity. We focused on a real problem, such as hemodynamics in patient-specific settings and presented extensive results on different platforms. We propose a way for measuring the performances under realistic scenarios. Comparing execution time and cost of the application on on-premise and on-demand targets, we found some evidence to support the claim that IaaS resources may be utilized for scientific CFD simulations possibly at lower cost than incurred locally. In particular, our test with Amazons spot-request feature coupled with availability of cutting edge resources (16-core nodes, 60GB RAM) suggests that small on-demand assemblies may be a viable alternative to local clusters. It is crucial that IaaSs provide resources immediately while local and grid resources are often subject to long queue wait times - an aspect that might offset any additional expense. Furthermore, while a modern local computing cluster with an efficient interconnection network will outperform an on-demand assembly (which is highly vulnerable to network performance), the cloud solution might be useful when cost needs to be minimized.

Among clusters, grids and cloud platforms there is a tremendous variation in communication performance. In order to reduce the heterogeneity due to data handling and interconnection network capabilities, we analyzed performance variations and process placement strategies for a parallel CFD application based on the finite element library LiFEV. The communication profile for this parallel application depends greatly on the partitioning of the mesh representing the physical geometry of the input. Such communication imbalance invites exploration of the possible mappings of the parallel tasks onto diversely performing networks of processors. As parallel target platforms universally present heterogeneous inter-process communication capabilities when nodes are multicore, performance advantages are possible through process placement that exploit this knowledge. We studied five process placement strategies: three of them use problem-related information and the others are typical OpenMPI process allocations. We found that the standard *pcore* placement mapping is well-suited for processing our CFD application. However, we showed that this allocation may be improved by our *part* placement, especially for larger numbers of hosts performing the computation. As a result, we showed that it is possible to adapt performance of parallel MPI applications by communication-aware placement of their MPI processes if the computation structure, input geometry as well as target architecture and network wiring is known and it may be done automatically by ADAPT.

As a complementary approach, we discuss in detail the optimal splitting of a problem of interest with different mathematical techniques. The introduction of overlap in the partition of problems featuring complex morphology gives more freedom in the optimal selection of interfaces and consequently may outperform more traditional nonoverlapping approaches. Different aspects have competitive dynamics resulting in a nontrivial optimization. The dependence of the number of iterations on the iterative-by-subdomain method decreases with the overlap, while the cost of the solution on each subdomain increases. The communication time depends on the location of the interface. Our results in realistic geometries point out the efficacy of an appropriate selection of overlapping to reduce costs in a parallel computing setting. In general, a small amount of overlap results in a good trade-off of all the competitive mechanisms affecting the total computational time. This shows that a more thoughtful positioning of the interfaces can benefit the overall computing performance, at a small additional computational cost on each processing unit. This complements discussion of sections 2 and 3, highlighting the trade-offs that can be achieved on different types of parallel platforms.

This paper has highlighted three dimensions of hemodynamic simulations on clusters, grids and clouds, both algorithmic- and platform-specific. While no universal conclusions can be drawn, our work proposes indications to the identification of protocols for hemodynamics computations in outsourcing that we think are needed - and progressively will be more requested in the next future - by clinical applications. We plan to include overlapping partitions more extensively in the current activities to have a more solid experience on the identification of optimal location of the interfaces and in general of the workbalance.

Acknowledgments

The authors gratefully acknowledge support by the Computational and Life Sciences (CLS) Strategic Initiative of Emory University. This research was supported in part by US National Science Foundation Grant OCI-1124418, OCI-1053575, DMS-1419060, DMS-1412963, Fondazione Cariplo “iCardioCloud”, URC Emory Grant 2015 “Numerical methods and flows at moderate reynolds number in left ventricular assisted devices”.

References

[1] L. Formaggia, A. Quarteroni, A. Veneziani (Eds.), Cardiovascular Mathematics, volume 1 of *M&SA*, Springer, Italy, 2009.

[2] C. A. Taylor, D. A. Steinman, Image-based modeling of blood flow and vessel wall dynamics: applications, methods and future directions., Annals Biomed Eng 38 (2010) 1188–1203.

[3] M. Piccinelli, A. Veneziani, D. A. Steinman, A. Remuzzi, L. Antiga, A framework for geometric analysis of vascular structures: application to cerebral aneurysms, IEEE Trans Med Imaging 28 (2009) 1141–55.

[4] S. F. C. Stewart, E. G. Paterson, G. W. Burgreen, P. Hariharan, M. Giarra, V. Reddy, S. W. Day, K. B. Manning, S. Deutsch, M. R. Berman, M. R. Myers, R. A. Malinauskas, Assessment of CFD Performance in Simulations of an Idealized Medical Device: Results of FDA's First Computational Interlaboratory Study, Cardiovascular Engineering and Technology 3 (2012) 139–160.

[5] C. M. Haggerty, D. A. de Zlicourt, M. Restrepo, J. Rossignac, T. L. Spray, K. R. Kanter, M. A. Fogel, A. P. Yoganathan, Comparing pre- and post-operative Fontan hemodynamic simulations: Implications for the reliability of surgical planning, Ann Biomed Eng (2012).

[6] D. A. de Zélicourt, C. M. Haggerty, K. S. Sundareswaran, B. S. Whited, J. R. Rossignac, K. R. Kanter, J. W. Gaynor, T. L. Spray, F. Sotiropoulos, M. A. Fogel, A. P. Yoganathan, Individualized computer-based surgical planning to address pulmonary arteriovenous malformations in patients with a single ventricle with an interrupted inferior vena cava and azygous continuation, J. Thorac. Cardiovasc. Surg. 141 (2011) 1170–1177.

[7] F. Migliavacca, G. Pennati, G. Dubini, R. Fumero, R. Pietrabissa, G. Urcelay, E. Bove, T.-Y. Hsia, M. De Laval, Modeling of the norwood circulation: effects of shunt size, vascular resistances, and heart rate, American Journal of Physiopathology 280 (2001) H2076–H2086.

[8] T. Passerini, L. Sangalli, S. Vantini, M. Piccinelli, S. Bacigaluppi, L. Antiga, E. Boccardi, P. Secchi, A. Veneziani, An integrated statistical investigation of internal carotid arteries of patients affected by cerebral aneurysms, Cardiovascular Engineering and Technology 3 (2012) 26–40.

[9] B. Gogas, Coronary stents, innovations in 2015, 2015. URL: `https://www.dropbox.com/s/5bz4j92kxr64aqh/Coronary%20Stents%20Innovations%2015.pdf?dl=0`, chapter “The future of coronary stenting, A mathematical view” by A. Veneziani.

[10] LifeV Project, `http://www.lifev.org`, 2012.

[11] T. Passerini, A. Quaini, U. Villa, A. Veneziani, S. Canic, Validation of an open source framework for the simulation of blood flow in rigid and deformable vessels, Int J Num Meth Biomed Eng 29 (2013) 1192–1213.

[12] S. Guzzetti, Hierarchical model reduction for the incompressible navier-stokes equations, 2014.

[13] L. Mirabella, C. Haggerty, P. T., M. Piccinelli, P. J. Del Nido, A. Veneziani, A. P. Yoganathan, Treatment planning for a tcpc test case: a numerical investigation under rigid and moving wall assumptions, Int J Num Meth Biomed Eng 29 (2013) 197–216.

[14] G. H. W. van Bogerijen, F. Auricchio, M. Conti, A. Lefiueux, A. Reali, A. Veneziani, J. Tolenaar, M. Moll, V. Rampoldi, S. Trimarchi, Aortic hemodynamics after thoracic endovascular aortic repair with the role of bird-beak, J Endov Therapy 21 (2014) 791–802.

[15] B. Gogas, L. Timmins, T. Passerini, M. Piccinelli, S. Kim, D. Molony, A. Veneziani, D. Giddens, S. King, H. Samady, Biomechanical assessment of bioresorbable devices, JACC: Cardiovascular Interventions 6 (2013) 760–761.

[16] L. Quartapelle, Numerical solution of the incompressible Navier-Stokes equations, volume 113, Birkhauser Basel, 1993.

[17] H. Elman, D. Silvester, A. Wathen, Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics, Oxford University Press, USA, 2005.

[18] J. Cebral, F. Mut, J. Weir, C. Putman, Quantitative characterization of the hemodynamic environment in ruptured and unruptured brain aneurysms, American Journal of Neuroradiology 32 (2011) 145–151.

[19] I. C. C. Workshop, The asme 2012 summer bioengineering conference, `www.asmeconferences.org/SBC2012/InauguralCFDWorkshop.cfm`, 2012.

[20] A. Quarteroni, A. Valli, Domain Decomposition Methods for Partial Differential Equations, Technical Report, Oxford University Press, 1999.

[21] A. Toselli, O. Widlund, Domain Decomposition Methods: Algorithms and Theory, volume 34 of *Springer Series in Computational Mathematics*, Springer Berlin Heidelberg, 2005.

[22] D. Darjany, B. Englert, E. H. Kim, Implementing Overlapping Domain Decomposition Methods on a Virtual Parallel Machine, in: G. Min, B. Di Martino, L. T. Yang, M. Guo, G. Rünger (Eds.), Frontiers of High Performance Computing and Networking ISPA 2006 Workshops, volume 4331 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 717–727.

[23] S. Guzzetti, V. Sunderam, A. Veneziani, Experimental optimization of parallel 3d overlapping domain decomposition schemes, 2015. To appear in Proceedings of 11th International Conference on Parallel Processing and Applied Mathematics.

[24] A. Quarteroni, A. Veneziani, P. Zunino, Mathematical and numerical modeling of solute dynamics in blood flow and arterial walls, SIAM Journal on Numerical Analysis 39 (2002) 1488–1511.

[25] J. Slawinski, T. Passerini, U. Villa, A. Veneziani, V. Sunderam, Experiences with target-platform heterogeneity in clouds, grids, and on-premises resources, in: 2012 26th International Parallel and Distributed Processing Symposium (IPDPS-HCW), IEEE, 2012, pp. 41–52.

[26] Sandia National Laboratories, The Trilinos Project, `http://trilinos.sandia.gov`, 2012.

[27] G. Karypis, V. Kumar, MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, `http://www.cs.umn.edu/~metis`, 2009.

[28] SuiteSparse, `http://www.cise.ufl.edu/research/sparse/SuiteSparse/`, 2012.

[29] S. Joachim, G. Hannes, G. Robert, NETGEN - automatic mesh generator, `http://www.hpfem.jku.at/netgen`, 2012.

[30] Boost C++ Libraries, `http://www.boost.org`, 2012.

[31] The HDF Group, Hierarchical data format version 5, `http://www.hdfgroup.org/HDF5`, 2012.

[32] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: 2008 Grid Computing Environments Workshop (GCE '08), IEEE, 2008, pp. 1–10.

[33] J. Slawinski, U. Villa, T. Passerini, A. Veneziani, V. Sunderam, Issues in communication heterogeneity for message-passing concurrent computing, in: Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International, IEEE, 2013, pp. 93–102.

[34] C. Lee, A. Snavely, Precise and realistic utility functions for user-centric performance analysis of schedulers, in: 2007 16th International Symposium on High Performance Distributed Computing (HPDC '07), ACM, 2007, pp. 107–116.

[35] G. Cheliotis, C. Kenyon, R. Buyya, A. Melbourne, Grid economics: 10 lessons from finance, GRIDS Lab and IBM Research Zurich, Melbourne, Tech. Rep (2003).

[36] B. Chun, D. Culler, User-centric performance analysis of market-based cluster batch schedulers, in: 2002 2nd International Symposium on Cluster Computing and the Grid, IEEE/ACM, 2002, pp. 30–30.

[37] J. N. Silva, P. Ferreira, L. Veiga, Service and resource discovery in cycle-sharing environments with a utility algebra, Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on (2010) 1–11.

[38] J. Simão, L. Veiga, QoE-JVM: An Adaptive and Resource-Aware Java Runtime for Cloud Computing - Springer, On the Move to Meaningful Internet Systems: OTM (2012).

[39] Penguin Computing On Demand / Indiana University, `https://podiu.penguincomputing.com/`, 2012.

[40] HPC Cloud Service, Penguin Computing, `http://www.penguincomputing.com/Services/HPCCloud`, 2012.

[41] D. E. Irwin, L. E. Grit, J. S. Chase, Balancing Risk and Reward in a Market-Based Task Service, in: 2004 13th International Symposium on High Performance Distributed Computing (HPDC '04), IEEE, 2004.

[42] J. J. Rehr, F. D. Vila, J. P. Gardner, L. Svec, M. Prange, Scientific Computing in the Cloud, Computing in Science and Engineering 12 (2010) 34–43.

[43] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright, Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud, in: CLOUDCOM '10: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, IEEE Computer Society, 2010.

[44] G. Wang, T. S. E. Ng, The Impact of Virtualization on Network Performance of Amazon EC2 Data Center, in: Proc IEEE INFOCOM 2010, 2010, pp. 1–9.

[45] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, D. H. J. Epema, Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing, IEEE Transactions on Parallel and Distributed Systems 22 (2011) 931–945.

[46] Performance Analysis and Visualization at Exascale (PAVE), `https://computation.llnl.gov/project/performance-analysis-through-visualization/`, 2014.

[47] A. Bhatele, T. Gamblin, S. H. Langer, P.-T. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. Levine, V. Pascucci, et al., Mapping applications with collectives over sub-communicators on torus networks, in: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, IEEE, 2012, pp. 1–11.

[48] H. Subramoni, S. Potluri, K. Kandalla, B. Barth, J. Vienne, J. Keasler, K. Tomko, K. Schulz, A. Moody, D. K. Panda, Design of a scalable infiniband topology service to enable network-topology-aware placement of processes, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society Press, 2012, p. 70.

[49] J. Slawinski, M. Slawinska, V. Sunderam, Unibus-managed execution of scientific applications on aggregated clouds, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 518–521.

[50] L. Antiga, T. Passerini, M. Piccinelli, A. Veneziani, Aneurisk web, ecm2.mathcs.emory.edu/aneuriskweb, 2011. URL: `ecm2.mathcs.emory.edu/aneuriskweb`.

[51] J. K. Slawinski, Adaptive Approaches to Utility Computing for Scientific Applications, Ph.D. thesis, Emory University, 2014. URL: `http://gradworks.umi.com/36/34/3634379.html`.