Parallelizing a finite element solver in computational hemodynamics: a black box approach

Journal Title
XX(X):1–10

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

F. Auricchio[1,2], M. Ferretti[1], A. Lefieux[1,5], M. Musci[1], A. Reali[1,2,3], S. Trimarchi[4], A. Veneziani[5]

Abstract

In the last 20 years, a new approach has emerged to investigate the physiopathology of the circulation. By merging medical images with validated numerical models, it is possible to support the decision-making process of doctors. The iCardioCloud project aims at establishing a computational framework to perform a complete patient-specific numerical analysis specially oriented to aortic diseases (like dissections or aneurysms) and to deliver a compelling synthesis. The project can be considered as a pioneering example of Computer Aided Clinical Trial, i.e. a comprehensive analysis of patients where the level of knowledge extracted by traditional measures and statistics is enhanced by the massive use of numerical modeling. From a computer engineering point-of-view, iCardioCloud faces multiple challenges. First, the size of the problems to solve for each patient is significantly huge – as typical of computational fluid dynamics (CFD) – and it requires parallel methods. In addition, working in a clinical environment demands efficiency as the timeline requires rapid quantitative answers (as may happen in an emergency scenario). It is therefore mandatory to properly implement on high-end parallel systems, such as large clusters or supercomputers.

Here we discuss a parallel implementation of an application within the iCardioCloud project, built with a black-box approach – i.e. by assembling and configuring existing packages and libraries and in particular LifeV, a finite element library developed to solve CFD problems. The goal of this paper is to describe the software architecture underlying LifeV, and to assess its performances and the most appropriate parallel paradigm.

This paper is an extension of a previous work presented at the PBio 2015 Conference. This revision extends the description of the software architecture and discusses several new serial and parallel optimizations to the application. We discuss the introduction of hybrid parallelism in order to mitigate some performance problems previously experienced.

Keywords

Computational Hemodynamics, HPC, Finite Elements, Hybrid Parallelism, Software Optimization

1 Introduction

Scientific computing during the last 20 years has been extended from traditional fields like civil, aerospace, mechanical engineering to new research topics related to sports, environment and life sciences. After taking advantage of the traditional expertise, these new fields raised new exciting challenges, often related to the impact of their effective solution that, beyond technical aspects, may reach out the entire society. This is the case of computational fluid dynamics (CFD) in the study of cardiovascular diseases. As a matter of fact, an accurate quantitative analysis of hemodynamics in patients based on numerical simulations has been proved to provide a terrific insight into physiopathological dynamics with a potential impact on diagnosis, prognosis and ultimately on the entire clinical practice.

In spite of a mature level of the mathematical background[11] and of many successful proofs of concept proving the previous statement, the massive penetration of medical practice by scientific computing tools is not yet real. There are several reasons for this. At a strict scientific level, the computational challenges raised by a patient-specific simulation involves image processing, massive numerical approximations featuring huge algebraic problems and appropriate post-processing procedures. Moreover, the entire work pipeline requires computational facilities that are not typically hosted by healthcare institutions; on the other hand, time requirements due to the large number of patients usually involved, as well as possibly specific emergency situations, demand for high computational efficiency. Altogether, these aspects still challenge mathematicians and computer scientists to search new efficient numerical methods (see e.g.[4;19]) as well as the effective exploitation of parallel architectures – local or remotely located, as in the case of cloud computing[18;22;26].

[1]University of Pavia, Pavia, Italy [2]Istituto di Matematica Applicata e Tecnologie Informatiche (IMATI-CNR), Pavia, Italy
[3]Technische Universität München. Institute for Advanced Study, Garching, Germany
[4]IRCCS Policlinico San Donato, Milan, Italy
[5] Department of Mathematics and Computer Science. Emory University, Atlanta (GA), USA

Corresponding author:
Mirto Musci, Laboratorio DCA, Department of Computer Engineering, University of Pavia, Via Ferrata 4, 27100 Pavia
Email: mirto.musci01@ateneopv.it

All these aspects are becoming part of a new – somehow groundbreaking – concept recently introduced as a breakthrough, the Computer Aided Clinical Trials (CACT). These are clinical studies – i.e. extensive investigations of large volume of patients as an ultimate step to assess the etiology of a disease as well as the effect of a therapy, in controlled environment with an enrollment of patients strictly regulated by specific protocols – where numerical simulations systematically provide a support to enhance the knowledge extracted with a traditional analysis. The iCardioCloud Project is a pioneering initiative founded by Fondazione CaRiPLo, Italy, working in this direction. As a matter of fact, it aims at the definition and establishment of a consolidated pipeline for the routine numerical analysis of patients affected by aortic diseases presented at the Hospital IRCCS San Donato in Milan, Italy.

The work-line starts with the acquisition of images and measurements to the simulation process, finalized by the delivery of charts, diagrams and tables that describe quantitatively the patient status in terms of quantities only partially measurable, including velocity, pressure, wall shear stress, etc.

More precisely, the procedure consists of three steps.

1. *Pre-processing* includes the image elaboration to obtain a 3D patient-specific geometrical model, the preparation of this model for the numerical simulation (the so-called *meshing step*) and the extraction of any data to be used in the numerical simulation from available measures.
2. *Problem solving* is the core of the procedure, where appropriate mathematical models fed by the patient-specific data obtained at the previous step are applied to extract more knowledge as the result of physical principles and constitutive laws. More specifically, the equations that are solved for the aortic diseases are the so-called *incompressible Navier-Stokes Equations* (NSE) that describe the behavior of a fluid like blood under the conditions prescribed by patient-specific data. This is the most intensive step in terms of computational cost.
3. *Post-processing* is the final step where the results are collected and represented in forms that allow an easy translation from mathematical to clinically significant information.

The present paper is clearly concerned with the second step. The NSE represent a challenging problem both in terms of mathematical theory and numerical approximation. In particular, an accurate approximation of NSE requires the repeated solution of linear systems of a size of about 10^7 equations for many time steps within an interval spanning a few heart beats.

The iCardioCloud team basically resorts to the finite element library called LifeV[2], a joint initiative of different groups, including the CMCS Center at the Department of Mathematics of EPFL, Lausanne (CH), the Laboratory MOX at the Department of Mathematics of the Politecnico di Milano (IT) and the Department of Mathematics and Computer Science of Emory University, Atlanta (GA-USA).

The library is intended to be a general purpose tool for the approximation of partial differential equations with the finite element method, however it has a strong and established record of publication in the field of bio-fluid dynamics. In particular, in [17] the library has been successfully validated against a benchmark proposed by the US FDA within the Critical Path Initiative [12;23].

Since the beginning, LifeV has been Object Oriented developed in C++ with advanced programming paradigms. In particular, since 2006 LifeV has been developed for parallel environments, mainly resorting to pure OpenMPI. For this reason, it is a natural candidate for the massive use of CACT, as the iCardioCloud project demonstrated [5;6;24].

However, the assessment of performances on parallel architectures is critical for the successful adoption of the tool on a routine basis. This paper gives a contribution in this direction, following up a previous work presented in the PBio 2015 conference in Helsinki [7]. We investigate the efficiency of the solver and discuss parallel paradigms to maximize the performances, moving from the coarse level profiling in [7] that identified modules to be customized and specific processor level optimizations. As in the previous paper we indicated a hybrid shared memory and message passing approach as a possible solution to mitigate the performance pitfalls, this option is explored here.

More precisely, Section 2 describes the processing component of the iCardioCloud project and introduces the structure of LifeV. Section 3 presents a case study of aorta hemodynamics, to showcase some performance benchmarks of a hemodynamics application on two different parallel systems. The two systems are a small local cluster and the CINECA FERMI [1] BlueGene/Q installation in Bologna (Italy). Section 4 discusses the improvements to the code carried out in this work: the usual compiler and serial optimizations and the introduction of hybrid parallelism with the openMP approach. A discussion of the future perspectives on the project, and the many challenges that lay in the way of a full hybrid and efficient implementation is presented at the conclusion.

2 Architecture

In this section, we provide a short introduction to the numerical procedure required to analyze a patient within the iCardioCloud workline and address some specific feature of the LifeV library.

2.1 The numerical approximation procedure

Partial Differential Equations of interest in real problems seldom admit an explicit analytical solution and numerical approximation procedures are in order. Among other methods, Finite Elements (FE) feature a strong mathematical background. As a matter of fact, the theory of FE has been systematically completed and organized in a "Periodic Table" (`femtable.org`). The mathematical foundations are critical to assess the accuracy and the overall performances of the method for problems of real interest. The method is based on the so called weak or integral formulation of the problem – the natural formulation descending from the virtual work principle. Moving from this formulation, the rationale is to split the region of interest in regions or elements where the solution is assumed to be polynomial. This yields to represent the

approximate solution as a linear combination of polynomial basis functions that have a local support, i.e. they are nonzero only on a limited small region of the domain of interest. This feature has many advantages, in particular: (a) the code organization may be based on a "local-to-global" perspective where local contributions to the problem are first computed on a local basis and eventually mapped to the global problem; (b) the global approximating problem obtained by this choice is typically a sparse algebraic system, where – after suitable linearization procedures – the associated matrix has just a small fraction of the total entries non zero, with a huge storage saving. After the algebraic system is obtained, it is conveniently solved by appropriate methods of numerical linear algebra. This operation needs to be repeated at each time step of a temporal discretization for unsteady problems.

The number of subdivisions of the region of interest into elements is relevant to (a) the accuracy of the solution; (b) the computational cost. More elements lead to a more accurate solution, but larger algebraic systems, generally more expensive to solve. The number of elements as well as the degree of polynomials selected to represent the solution need to be carefully selected as a trade-off between accuracy and computational costs, depending on the application at hand and the computational facilities available. It is worth noting that in CACT the accuracy required is basically associated with the clinical application, and the quality of numerical solutions must guarantee an accurate understanding of the situation of the patient to be translated into clinical actions. Therefore, in many cases accuracy needs not to be pushed to extreme limits, as far as doctors correctly interpret the results and take the consequent actions – in favor of rapid computations compatible with the clinical routine. The literature dedicated to the FE is tremendously large and it is not possible to provide an exhaustive bibliography. Interested readers may refer, e.g., to [10;14] and references therein.

In the applications relevant to iCardioCloud, we need to solve the Navier-Stokes Equations. These are the equations that describe velocity and pressure in a region of interest, as the result of the application of basic principles (conservation of mass and momentum) and empirical constitutive laws – for instance for describing the blood rheology. These equations raise some specific challenges, as the three components of velocity and the pressure are computed all together and the polynomial degree of interpolation for the velocity depends on the one for the pressure. This implies that the size of the algebraic system to solve rapidly increases. In the application of interest for iCardioCloud a typical size of of the order of 10^7. In addition, as for any unsteady nonlinear problems, these systems need to be assembled and solved at each time step, spanning a few heart beats for a number of steps on the order of 10^4.

In aortic diseases, the high velocity induced by the heart pumping action generally triggers dynamics qualified as highly disturbed or even turbulent in some (typically pathological) cases. This requires an extra effort that can lead to finer reticulations, or additional equations that properly describe the energy cascade and the effects of disturbances from the small scales (not resolved) to the large ones. For aortic flows, the latter approach in the framework of the so called *Large Eddie Simulation*, has been proved to be effective and it is used in iCardioCloud [8].

This short summary points out the significantly high computational effort required by a patient-specific analysis. For the sake of clarity, here we list the steps to do.

1. Geometry tessellation or meshing;
2. Time loop (from 3 to 5 heart beats, each is in turn split into an order of thousands of steps)
   (a) *Assemble* the matrix of the algebraic problem with a local-to-global approach
   (b) *Solve* the algebraic problem
   (c) *Store* the solution (when needed)

The LifeV library is mostly concerned with the Assembly step, while the solution of the algebraic problem is outsourced to available libraries. In particular, the Trilinos Library is used as the default choice. The assembly step consists of a series of procedure to (i) compute integrals of the basis functions properly differentiated for the specific problem to be solved; (ii) map each integral from a local computation on a reference element to the global problem. Both the assembly and solving steps greatly benefit from parallel architectures, as we detail in the next section.

2.2 Processing

We summarize hereafter the sequence of steps performed to solve the problem with emphasis on the assembly step and its parallel framework. For a full description of these aspects, the reader is referred to the website `lifev.org` and to [21]. Once the reticulation of the domain of interest is available and all the specifications of the problem to solve are available, all the MPI processes load the global mesh. Then we perform the following steps.

1. The mesh is partitioned in N parts by using ParMETIS. Here N denotes the number of processes or computational units. ParMETIS a parallel library to partition unstructured graphs based on the METIS library. The guideline of partitioning is to guarantee a balanced workload among the different processors, including in this the solution time of each processor as well as the communication time. This is therefore a critical step since the quality of this decomposition has a major impact on the overall performance. At the end of the partitioning procedure, each process retains only the associated partition.
2. *Assembly.* In this step the library assembles the algebraic system as dictated by the mesh and the choice of the degree of the polynomial functions. Notice that the assembly is performed simultaneously also for the vector that forms the right hand side of the algebraic system. Each computational unit loops over the elements (e.g triangles in 2D) assigned by ParMETIS in the previous step. Each local loop on elements can be done in parallel, without communication. The loop – as mentioned above – requires the numerical solution of integrals: (a) each element is referred to template element where the integrals are easily computed. [20]; (b) the contribution brought by each element is then mapped to the global

level, by constructing a matrix for each process, this is called local assembly; (c) after local assembly, a method called `GlobalAssemble()` manages the communication and constructs the pattern of the global matrix, i.e. the map of the nonzero entries in the matrix. The amount of communications in the global assembly depends the granularity of the decomposition. The Epetra methods of the Trilinos library handles all of the underlying complexity.

3. The matrix/right hand side pair obtained after assembly typically does not include the boundary conditions. They are properly and conveniently included at the end of assembly, to avoid conditional jumps inside the assembly loop over elements to select boundary degrees of freedom. The inclusion of boundary conditions typically entails a correction of a low number of rows of the matrix and the right hand side.
4. *Solver*. The solution of the linear system (described by the global matrix) is typically performed by iterative methods. The number of iterations is managed (and minimized) by creating simplified linear problems (called preconditioners) whose solution drives the solution of the problem at hand in a fast way. Ideally, the number of preconditioned iterations when solving a linear system should be independent of the number of processors. LifeV treats the algebra solver as a black box operated by Trilinos, where OpenMPI is used to distribute the computations over the system. Note that the solver decomposition is generally different from the one used in the assembly step. The interested reader may refer to[13] for the documentation of the Trilinos library.
5. Finally, the solution is exported and stored in parallel by using the HDF Format, so to minimize the slowing due to the disk writing of large vectors[21].

It is worth stressing again that the matrices of the algebraic problem are sparse, i.e. only a few nonzero entries are present. For this reason, matrices are conveniently stored in compact format such as Compressed Row Storage (CRS) where only the nonzero entries are tracked. The CRS matrices and the corresponding unknown vectors are properly distributed among the processes thanks to Epetra.

3 Case study: hemodynamics of the aorta

In this section, we propose a case study to analyze the behavior of a LifeV hemodynamics application in a real scenario. In particular, we chose a case study about the blood flow in the aorta of a post-operative patient. From a mathematical standpoint, as already discussed above, the problem is modeled with incompressible NSE. Developing a FE solution for the NSE is a well-studied and complex topic which is outside the scope of this paper. For further information, one may refer to[16] for a physics standpoint introduction to the NSE or to[10] for a discussion on FE methods for the NSE, the numerical method upon which the LifeV-based application is based and thus it follows the general model depicted in Section 2.

Figure 1. Snapshot of the mesh used for the test problem.

The case study presented here is extracted from[5], where is properly discussed from both an engineering and medical point-of-view.

Nevertheless, we provide here a short discussion of the issues related to the discretization and resolution of the NSE, as well as the test problem used in this work. The geometrical input data of the case study is taken from clinical data and it is composed of 6.7 million of tetrahedra (see Figure 1). The method used is the classical Nodal Lagrange Finite Element method. The solvability and stability of the system rely on a careful choice of the polynomial approximations of the velocity and the pressure. As a matter of fact, we use a piecewise continuous linear approximation for the velocity field, stabilized by a quadratic bubble, and thus the number of degrees of freedom for the velocity field is around 23 millions, and a piecewise continuous linear pressure and thus 1.2 millions degrees of freedom for the pressure. As a consequence, the global system is of 24.2 millions degrees of freedom. By no means, such a description represents the number of floating point procedures used to solve the problem solving the system (see[9] for a presentation of the problem using LifeV, but with a different Navier-Stokes solver). Indeed, with such a larger linear problem it is not possible to use direct methods such as Gauss-Jordan. Instead, the code uses an algebraic splitting approach, called High Order Yosida. It is out of the scope of the present work to present in detail the algebraic splitting strategy used in this work. We want to point out that the required time for one time step heavily depends on, in particular, the Reynolds number of the problem (and associated stabilization strategies), the size of the elements, the time stepping, the polynomial order used to approximate the various fields, and, obviously, the time integration schemes and the iterative solver technique. The interested readers may read[26], and the literature cited therein, in particular[25]. Regarding the boundary conditions, the inflow flux has been obtain from the same patient from phase contrast Magnetic Resonance Imaging. More precisely, the

inflow flux is the integration of the projection of the velocity field onto the normal of the plane, which was selected by the radiologist. On the contrary, the outflow boundary conditions have been modeled using 3-elements Windkessel system. The parameters have been taken from the literature[15]. Obviously, the choice of the boundary conditions also impacts the conditioning of the global system, in particular for the outflow ones. However, these critical points are strictly mathematical and, therefore, should not impact the parallelization strategy. As a consequence, to keep the mathematical aspects out of our concern, we maintained, for all our test problems, all the parameters fixed, with the only exception of the number of cores. Consequently, our main concern in the present work is the asymptotic behavior of the code, rather than the absolute results.

The analysis has been conducted on two different testing environments. The first one is a relatively small cluster, hosted at the University of Pavia. The second is the FERMI[1] supercomputer (an IBM BlueGene/Q) at CINECA in Bologna, Italy. On each machine we performed two different sets of tests. The goal is to assess the performance and the scalability of the processing steps of LifeV, and especially the assembly and solver phase.

3.1 Local cluster

The first batch of tests has been run on a cluster of four AMD Opteron 6272 nodes running at 2.1 GHz. Each node has four CPUs, each with 16 cores, for a total of 64 cores and 252 GB of RAM per node. Overall, the cluster has 256 cores and about 2 Tb of RAM available. Each node is connected to the others through a Gigabit Ethernet switch.

In the following we present two figures to show, respectively, the performance of the assembly and the solving step of the blood flow analysis of the case study (see Section 2.2). Please note that all data are the average of 20 consecutive runs, where the first is discarded because it includes some initialization costs. Figure 2 shows the scalability results for the assembly phase on the local cluster. By analyzing the figure, it is clear that the assembly phase scales very nicely up to 64 cores. We can draw the same conclusions, even when considering different distributions of the computations across the entire cluster. Nice scalability results, as shown by the differently colored lines in the figure, hold in fact either when using a single machine and even when using the four computational units at the same time. Unfortunately, the scalability quickly degrades if the number of cores become larger than 64. Eventually, after 128 cores, the overall execution times even start to increase. The reason for this behavior is due to the high ratio of communication time over computation time, which is overemphasized by the interconnection network of the local cluster: clearly, a simple Gigabit Ethernet is not enough to keep up with the high volume of communication required by applications such as the case study.

Figure 3 shows the scalability results for the solving phase on the local cluster. Analyzing the figure, it is clear that the solver does not scale very well on the local cluster; we are sure that the lack of scalability is not due to the internal implementation of the Trilinos solver, as proven by the more satisfying results described in the following section on the FERMI supercomputer. To understand the reason for

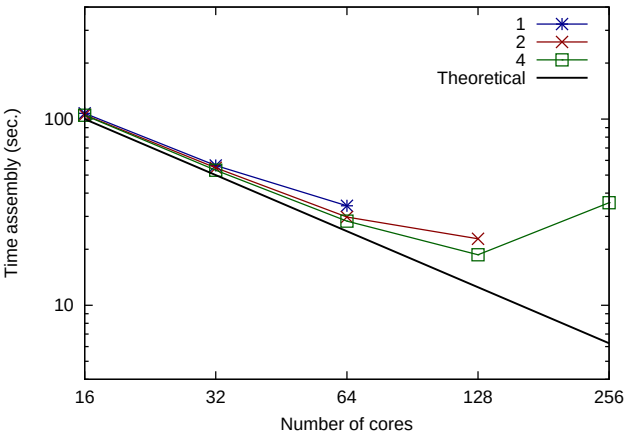

Figure 2. Scalability of the assembly phase of the case study on the local cluster. On the x-axis the overall number of cores concurring to the computation. On the y-axis the execution times in seconds. Both axis use a logarithmic scale. Different plots correspond to different distribution of the computation on the nodes (e.g. the red line refers to a test scenario where the computation is equally distributed on *two* nodes in the cluster).

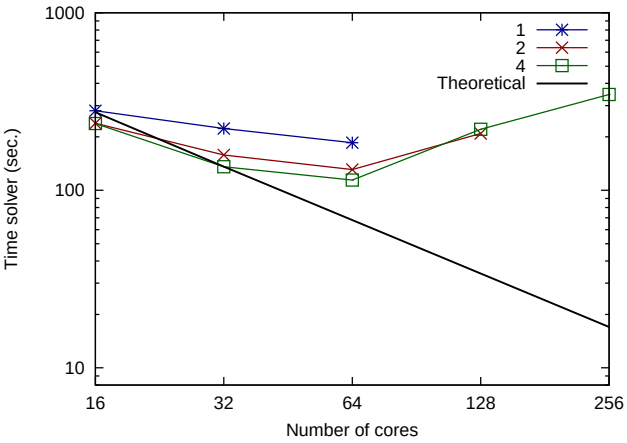

Figure 3. Scalability of the solving phase of the case study on the local cluster. On the x-axis the overall number of cores concurring to the computation. On the y-axis the execution times in seconds. Both axis use a logarithmic scale. Different plots correspond to different distributions of the computation on the nodes (e.g. the red line refers to a test scenario where the computation is equally distributed on *two* nodes in the cluster).

this behavior, it is also important to note that we obtain better performance for the solver when the computation is distributed among different nodes (at least up to 64 total cores). Remember that the computations in the solver are mostly floating point operations and that the Interlagos (AMD 6200 series) architecture shares each floating point unit between two cores: if we use less cores on each node we can easily improve the floating point performance; moreover, less processes on a node also means less contention for the memory bus. These two hardware limitations provide a huge hint about the lack of scaling.

Moreover, as we shall see later on in Section 4.1, more satisfactory results in the solver scalability were obtained using a serially optimized version of the code, which is able to fully exploit the floating point units with efficient vectorization.

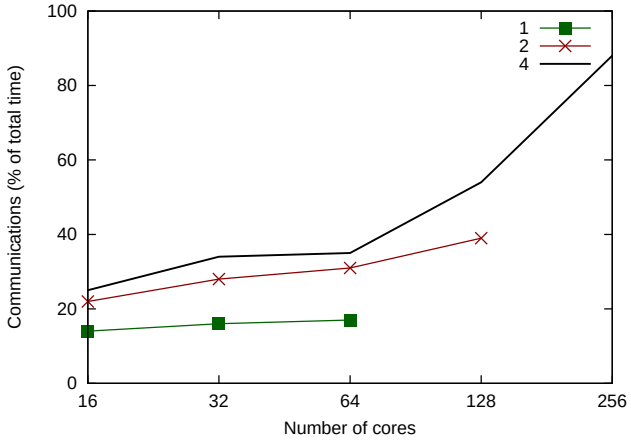

Figure 4. Communication time of the target application on the local cluster. On the y-axis the percentage the time spent in MPI calls with respect of overall execution time. On the x-axis the number of cores. Different plots correspond to different distributions of the computation on the nodes (e.g. the red line refers to a test scenario where the computation is equally distributed on *two* nodes in the cluster).

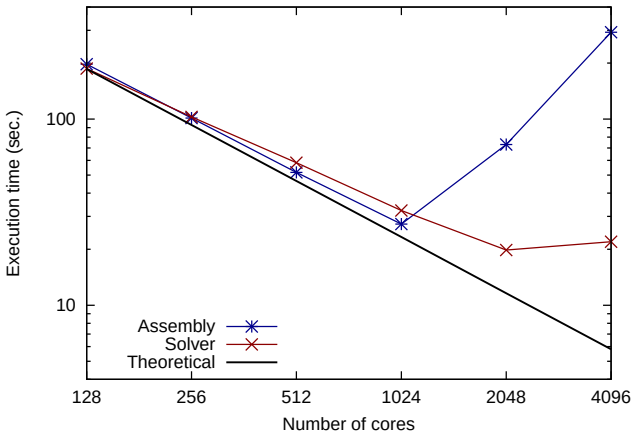

Figure 5. Scalability of the case study on the FERMI supercomputer. The data refer to both the assembly and solving steps. On the x-axis the number of cores concurring to the computation. On the y-axis the execution times in seconds. Both axis use a logarithmic scale.

Finally, we have analyzed the behavior of the application using the TAU profiler[3]. Figure 4 shows the percentage of the overall execution time that is spent in communications as the number of cores increase. Similarly to the previous graphs in this section, different distribution of the workload across the local cluster are analyzed. The figure shows that the percentage increases more than linearly when we use more than one node; in other words the application becomes communication-bound, and this is coherent with the unsatisfactory scalability results discussed above. These results and can be explained with the inadequacy on the interconnection network. Indeed the impact of communication remains more or less constant when using only one node.

3.2 FERMI supercomputer

The local cluster is a necessary tool for evaluating the development of the iCardioCloud project. However, the sheer size of hemodynamics problems and the time requirement of clinical applications demand a more drastic approach. To understand the performance on the hemodynamics application on a high-end machine, we repeated the experiments described in the previous section on the FERMI supercomputer. The main difference in the set-up of the experiment is that we had to partition the input mesh using an offline partitioning strategy. The reason is that the amount of memory available to each node in the FERMI architecture is smaller than the size of the mesh and each thread has to load the whole mesh at the beginning of the application to proceed with the partitioning. On the contrary, with offline partitioning we provide the mesh already divided and thus each thread only load its part of the mesh. However, this process does not impact on the performance in a significant way, as offline partitioning is done only once and off-line; to be fair, the results on the local cluster in the previous section did not include the cost for the initial online partitioning.

This time, we condense in one figure the results for both the assembly and solving steps. All data are the average of 10 executions, expressed in seconds. Again, the first execution is discarded to discard the initialization time.

Figure 5 shows that the both phases and thus the overall application, scale very nicely up to 1024 cores on FERMI. Increasing the available number of cores, however, only the solver performance has any benefit, while the assembly quickly becomes the bottleneck of the application: in the case of 4096 and beyond it completely dominates the execution times.

Having analyzed the case study on both architecture, we can compare the results presented in this section to the ones presented in Section 3.1. First of all, even if the solver did not scale very well on the local cluster, it scales reasonably well on FERMI, up to 2048 cores. FERMI interconnecting network is better suited for high volumes of communication with respect to the local cluster; indeed the ratio of computation over communication remains constant up to 1024 cores. Moreover, the PowerA2 chip is more optimized for floating point operations with respect to the AMD 6272; in particular floating point unit are not shared between adjacent cores, while this does happen on the Bulldozer architecture.

Overall, the most critical point is that as the number of cores increases, the amount of the inter-process communication and synchronization during the global assembly quickly explodes on FERMI. Intuitively, the *perimeter* of the mesh chunks becomes longer, while the inner volume becomes smaller. The smaller the ratio between the volume of a chunk and the length of its border, the smaller become the granularity of the parallel application, up to a point where the communication overhead becomes comparable to the computational time of a single chunk. Due to the relatively small number of cores, this behavior was not observed on the local cluster.

To mitigate the problem, LifeV developers have proposed a new decomposition strategy for the computational mesh but it is not used in our hemodynamics application at present. The approach, called *ghost cells*, amounts to inflate with replicated the data the borders of each computational chunk. In this way, each region need to communicate less with

their neighbors during assembly. Overall this amounts to a reduction of inter-chunk communication. Put in other words, using the ghost cells we can keep the granularity the same and reduce the overhead. The cost of this strategy is twofold: first of all a relatively small portion of the data, that is the ghost cells, needs to be replicated across multiple computational units, thus increasing the memory footprint of the application. On the other hand, the solver needs to be modified to account for a different data format in input.

An alternative, yet more complicated, strategy is possible: indeed, hybrid parallelism (Section 4.2) could probably improve the overall performance of the application beyond the level of the ghost cells approach.

4 Optimizing the performance

The analysis presented in Section 3 has shown several performance issues of our hemodynamics application. Clearly, these issues have something to do with the parallel paradigm of the implementation, and can theoretically be solved by modifying the paradigm and/or its implementation. For instance, we could introduce hybrid parallelism in the iCardioCloud project. This idea will be fully discussed in Section 4.2.

However, we are not only interested in parallel performance and scalability on supercomputers, but also in the overall execution time. In particular, even if it is relatively small, the local cluster presented in Section 3.1 is extremely important for our day-to-day operations, as it is not easy to obtain a satisfying level of access to more powerful machines. In practice, obtaining good performance on the local cluster is of paramount importance for our research. To this end, we can apply several black-box serial optimizations to the application, which are discussed in Section 4.1.

4.1 Serial optimizations

From a software engineering point-of-view, the iCardioCloud framework we see LifeV as a black-box, thus the amount of serial optimization we can introduce is limited by our choice not to modify the code in-depth.

As is it common with many frameworks, every major revision of Trilinos brings new tools and capabilities to the library. In particular the transition from Trilinos 10 to Trilinos 11 has brought to the table many optimizations; among them, the most significant are the new internal implementations of the solver modules. Unfortunately, our version of LifeV is Trilinos 10 based. The first task we did has been to upgrade our version of the code to Trilinos 11. Moreover, upgrading to Trilinos 11 is a necessity in perspective for the parallel optimization of the application (Section 4.2). Upgrading to Trilinos 11, obviously, also brought us to upgrade (and sometime substitute) most of the other libraries in the iCardioCloud framework. For instance, we decided to move from the ACML linear algebra library to MKL on the local cluster, as testing showed that the latter had better performance when comparing their respective most recent releases.

The serial optimization also included an in-depth study of possible compile-time optimizations. In particular, we profiled the application to discover that almost 60% of the computation time, is spent inside the underlying mathematical library, such as ACML or MKL. Both of these libraries support system-specific hardware instructions. Namely, as we are working on AMD nodes on the local cluster, recompiling the code using the FMA4 vector instruction has the potential to greatly improve the overall performance of the application. In addition, we studied the effect of many other possible compiler optimization, and selected the best and most stable ones; in particular, we strived to obtain automatic vectorization of the code.

Run	Assembly	Preconditioner	Solver	Overall
1	81.14	82.88	70.06	234.08
2	29.32	36.76	61.96	128.04
3	29.21	35.98	80.77	145.96
4	29.57	36.61	67.88	134.06
5	29.38	36.88	76.53	142.80
6	29.53	36.60	73.92	140.04
(etc.)	...	...	...	...

Table 1. Raw data for the 64 cores performance of the case study on the local cluster. “Old implementation” as in the previous version of this paper. Note that these data is the same shown in Figure 2.

Run	Assembly	Preconditioner	Solver	Overall
1	13.76	36.75	57.73	108.25
2	37.03	37.49	50.55	125.08
3	37.15	37.41	50.97	125.53
4	37.36	37.45	53.45	128.25
5	37.41	37.56	43.85	118.82
6	37.73	37.64	43.91	119.28
(etc.)	...	...	...	...

Table 2. Raw data for the 64 cores performance of the case study on the local cluster. “New implementation” with compile time optimizations and Trilinos 11 porting.

The selected string of optimizations flags on the local cluster is the following: `-O3` (standard optimization flag), `-march=bdver1` (compile optimized code for the AMD Bulldozer architecture), `-funroll-all-loops` (force loop unrolling; this flag is recommended by AMD), `-mfma4` (enable FMA4 instruction set), `-mavx` and `-mprefer-avx-128` (enable AVX instruction set to induce vectorization outside of the already optimized math libraries). A more in-depth discussion of these compiler optimization, can be found in[9].

To prove the effectiveness of the serial optimizations, we executed several tests on the case study described in Section 3. Table 1 shows in tabular form the same data presented in a synthetic way in Figure 2, but is limited to the most interesting scenario in practical applications; that is, due to the results in Figure 4 showing the poor interconnections on the local cluster, we run the application on a single node, exploiting all of its 64 cores. Table 2 is similar to Table 1, but shows the results of the new optimized implementation. All the values are expressed in seconds.

Each row of the two tables show a different time step of the computation. The measured times are subdivided in the different step of processing. As previously, we can safely rule out the first rows in the tables as outliers. First of all we can note that the assembly times seems to be almost the same in the two versions, and thus are unaffected by the serial optimizations. The most interesting results, however, is that the solver takes much less time in the new version of the

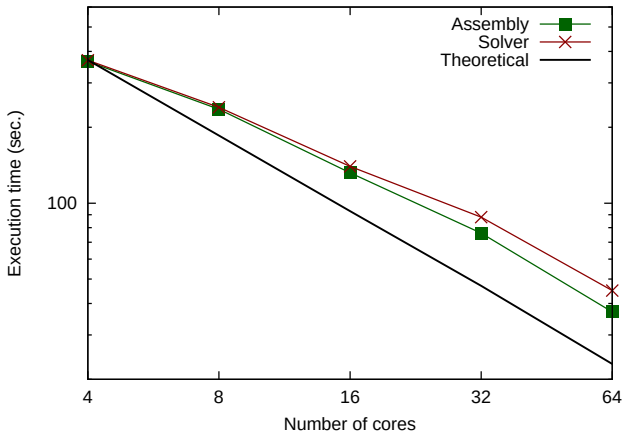

Figure 6. Scalability of the new implementation for the case study on a single node of the local cluster. On the x-axis the number of cores concurring to the computation. On the y-axis the execution times in seconds. Both axis use a logarithmic scale.

code. As stated before, this was expected as most calls to the underlying mathematical library are performed during the solution of the matrix, and this calls are now heavily optimized.

Averaging the results, we can state that the serial optimizations amount to a measured reduction in the overall execution time of almost 16%, and up to 20% in some cases. Bear in mind that a one-fifth reduction in the execution time amounts to several days when a complete run on a real patient could take up to two weeks of computations. Finally, Figure 6 shows the extremely good scalability of the new implementation on a single node; it seems that the serial optimizations have solved the previous problems about the scalability of the solver (see Section 3.1).

We can conclude the the impact of the serial optimizations is quite significant, especially when considering that, due to the black-box approach, we have not modified the internal structure of the code.

4.2 Parallel Optimizations

As shown by the analysis presented in Section 3, the assembly phase of the LifeV implementation has serious scalability issues. To mitigate the problem, we discussed the ghost cell approach (Section 3.2). However, another, more radical approach is possible, that is to modify the parallel paradigm of LifeV.

At the moment, LifeV processes communicate using a pure distributed memory paradigm which means that: (i) the communication overhead is quite high and (ii) the processes cannot share data without explicit communication. Similarly to the alternate domain decomposition of the ghost cell method, a different approach, based on shared memory paradigm, would decrease the overhead and amount of inter-process communication of the overall application. In theory, the approach would work for both the assembly and solving phase and is not mutually exclusive with ghost cells. In general, the disadvantage of the shared memory is that it is required to carefully protect all common resources using some form of synchronization (e.g. locks, critical section, etc.).

In the past few years, research has been focused on the combination of the two paradigms (distributed and shared memory) to take advantage of both; this approach is called hybrid parallelism. On a many-core machine such as FERMI, hybrid solutions are almost mandatory to achieve the best possible performance.

Unfortunately, the porting of iCardioCloud to an hybrid approach is far from being easy or time-efficient. The most recent versions of Trilinos 11 support hybrid parallelism thanks to the Tpetra module, which allows to exploit hybrid strategies to manage the mesh in a concurrent way. The alternative is Epetra, currently used by the LifeV, which is mostly based around a pure MPI, distributed memory approach. The Petra modules are essential because they are the interface to all the other modules of Trilinos, in particular to the linear solvers (see Section 2.2). As discussed in the previous section, we have already upgraded LifeV to work with Trilinos 11. However, the main problem is that, due to the nature of its code, a simple porting of the LifeV source code from Epetra to Tpetra is not feasible.

In fact, LifeV uses a modified and extended version of the Epetra module, which sometimes relies on the details of the internal implementation of Epetra (i.e. the code is not perfectly encapsulated). The main issue is that the `MatrixEpetra` class, the interface from LifeV to Epetra, makes heavy use of advanced capabilities of the `FECrsMatrix` class that are not yet available in Tpetra and thus it renders, at present, cumbersome the implementation of Tpetra into our LifeV code. Nevertheless, even if the Epetra module lacks the advanced hybrid capabilities of Tpetra (such as support for CUDA, PThreads, TBB, etc.) its data structures actually embeds partial OpenMP capabilities, which can be easily activated during compilation.

We ran on FERMI the same test performed in Section 3.2. The results on the local cluster are not showed here, as they are not particularly interesting. In fact, using the hybrid approach on the local cluster has very few advantages and, in some cases, even worse performance. Figure 7 shows the results on the FERMI supercomputer. For a fair comparison, the data still refer to the old implementation (Section 4.1), using Trilinos 10 without the optimizations introduced in Section 4.1. The tests were run with 16 OpenMP threads per MPI process. As each socket in the FERMI architecture has 16 cores[1], this means one MPI process runs per socket or four per node. Again, the data is the average of 10 different runs. In green, we repeat the data from Figure 5 to allow for an easier comparison.

From the figure it is clear that this limited introduction of hybrid parallelism does not drastically modify the asymptotic performance of the application, and the scalability issues have not been solved, as expected. However, we can note that the computational explosion of the assembly phase is less pronounced, and the scalability of the solver has significantly increased. While not conclusive, these data are a clear indication that the hybrid approach is the correct way to improve both the performance and the scalability of a LifeV hemodynamics application.

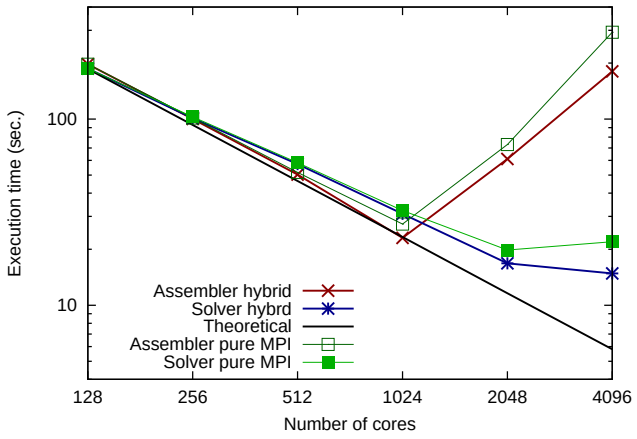

Figure 7. Preliminary results of a hybrid implementation with OpenMP on the the FERMI supercomputer. As before, the data refer to both the assembly and solving steps. On the x-axis the number of cores concurring to the computation. On the y-axis the execution times in seconds. Both axis use a logarithmic scale.

5 Conclusions

In this paper we have discussed an emerging multidisciplinary research topic, that is using numerical methods to study the hemodynamics of a pathological aorta. In particular, we have described the iCardioCloud framework, which uses a finite element software in a parallel context. iCardioCloud is based on LifeV, a pure MPI parallel library. The overview of the software architecture of a LifeV-based application has shown the potential of this approach, but it has underlined present performance limitations, which will dealt in future works.

To validate our assumptions, we have analyzed the performance of a clinical case study with real data on two different systems: a small cluster and a supercomputer. The analysis showed some limitations of the pure MPI approach, such as lack of scalability and overhead of the communication bandwidth.

In the final section we have discussed possible solutions to the performance problems, that is the introduction of a few selected serial optimizations, which allowed to increase the overall execution time by a significant margin, and the introduction of a new hybrid parallel paradigm. Unfortunately, a full porting of iCardioCloud is extremely time-consuming, due to the nature of the LifeV implementation. However, thanks to a limited introduction of OpenMP in the framework, we have shown the potential merit of such porting.

At the moment, a full, efficient Trilinos 11 hybrid implementation of LifeV is a work in progress.

Acknowledgements

The authors would like to acknowledge CILEA for the support through the LISA project BIO-FSI, and CINECA for providing access to the FERMI supercomputing facility. Finally, the authors would like to thank M. Conti and S. Morganti (image and data processing), C. Trentin (image processing), F. Braghiroli (software engineer), F. Sardanelli and F. Secchi (radiologists) for their contributions.

Funding

This work was supported by the European Research Council [project no. 259229]; Cariplo Foundation [project no. 2013-1179], Regione Lombardia [project no. E18F13000030007]; Ministero dell'Istruzione, dell'Università e della Ricerca [project no. 2010BFXRHS].

References

1. Fermi Reference Guide. `http://www.hpc.cineca.it/content/fermi-reference-guide`. Accessed: 2015-01-20.
2. Life V Web Page. `http://www.lifev.org`. Accessed: 2016-02-25.
3. TAU profiler. `https://www.cs.uoregon.edu/research/tau/home.php`. Accessed: 2016-02-25.
4. L.A.Mansilla Alvarez, P. J. Blanco, C. A. Bulant, E. A. Dari, A. Veneziani, and R. A. Feijoo. Transversally enriched pipe element method for blood flow modeling. Technical report, Emory University.
5. F. Auricchio, M. Conti, A. Lefieux, S. Morganti., A. Reali., F. Sardanelli, F. Secchi, S. Trimarchi, and A. Veneziani. Patient-specific analysis of post-operative aortic hemodynamics: a focus on thoracic endovascular repair (TEVAR). *Computational Mechanics*, 54(4):943–953, 2014.
6. F. Auricchio, M. Conti, S. Marconi, A. Reali, J. L. Tolenaar, and S. Trimarchi. Patient-specific aortic endografting simulation: From diagnosis to prediction. *Computers in biology and medicine*, 43(4):386–394, 2013.
7. F. Auricchio, M. Ferretti, A. Lefieux, M. Musci, A. Reali, S. Trimarchi, and A. Veneziani. Assessment of a black-box approach for a parallel finite elements solver in computational hemodynamics. In *Third International Workshop on Parallelism in Bioinformatics*, 2015.
8. L. Bertagna, A. Quaini, and A. Veneziani. Deconvolution-based nonlinear filtering for incompressible flows at moderately large reynolds numbers. *Int J Num Meth Fluids*, 2015. (accepted).
9. F. Braghiroli. Configuration, profiling and tuning of a complex biomedical application: analysis of ISA extensions for floating point processing. Master's thesis, University of Pavia, 2014.
10. H. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solvers with applications in incompressible fluid dynamics*. Oxford Press, 2005.
11. L. Formaggia, A. Quarteroni, and A. Veneziani. *Cardiovascular Mathematics*. Springer, 2009.
12. P. Hariharan, M. Giarra, V. Reddy, S. W. Day, K. B. Manning, S. Deutsch, S.F.C. Stewart, M. R. Myers, M. R. Berman, G. W. Burgreen, et al. Multilaboratory particle image velocimetry analysis of the FDA benchmark nozzle model to support validation of computational fluid dynamics simulations. *Journal of biomechanical engineering*, 133(4):041002, 2011.
13. M. Heroux. Trilinos documentation. `https://trilinos.org/about/documentation/`.
14. T. JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.
15. H.J. Kim, I.E. Vignon-Clementel, C.A. Figueroa, J.F. LaDisa, K.E. Jansen, J.A. Feinstein, and C.A. Taylor. On coupling a lumped parameter heart model and a three-dimensional finite element aorta model. *Annals of Biomedical Engineering*, 37(11):2153–2169, 2009.

16. L.D. Landau and E.M. Lifshitz. *Fluid Mechanics*. Elsevier, 1987.
17. T. Passerini, A. Quaini, U. Villa, A. Veneziani, and S. Canic. Validation of an open source framework for the simulation of blood flow in rigid and deformable vessels. *International journal for numerical methods in biomedical engineering*, 29(11):1192–1213, 2013.
18. T. Passerini, J. Slawinski, U. Villa, and V. Sunderam. Experiences with cost and utility trade-offs on IaaS clouds, grids, and on-premise resources. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 391–396. IEEE, 2014.
19. S. Perotto, A. Reali, P. Rusconi, and A. Veneziani. HIGAMod: A hierarchical isogeometric approach for model reduction in curved pipes. Technical report, Emory University.
20. D. A. Di Pietro and A. Veneziani. Expression templates implementation of continuous and discontinuous galerkin methods. *Computing and visualization in science*, 12(8):421–436, 2009.
21. A. Veneziani S. Deparis, L. Formaggia. The LifeV library: engineering mathematics beyond the proof of concept. (in preparation).
22. J. Slawinski, T. Passerini, U. Villa, A. Veneziani, and V. Sunderam. Experiences with target-platform heterogeneity in clouds, grids, and on-premises resources. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, 2012.
23. 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, et al. Assessment of CFD performance in simulations of an idealized medical device: results of fda's first computational interlaboratory study. *Cardiovascular Engineering and Technology*, 3(2):139–160, 2012.
24. G. HW van Bogerijen, F. Auricchio, M. Conti, A. Lefieux, A. Reali, A. Veneziani, J. L. Tolenaar, F. L. Moll, V. Rampoldi, and S. Trimarchi. Aortic hemodynamics after thoracic endovascular aortic repair, with particular attention to the bird-beak configuration. *Journal of Endovascular Therapy*, 21(6):791–802, 2014.
25. A. Veneziani. Block factorized preconditioners for high-order accurate in time approximation of the navier-stokes equations. *Numerical Methods for Partial Differential Equations*, 19(4):487–510, 2003.
26. U. Villa. *Scalable Efficient Methods for Incompressible Fluid-dynamics in Engineering Problems*. PhD thesis, Emory University, 2012.