A Fast Navier Stokes Flow Simulation Tool for Image Based CFD

Bilel Hadri,[*] Marc Garbey[*]

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
`http://www.cs.uh.edu`

Technical Report Number UH-CS-07-07

July 17, 2007

Keywords: Domain Decomposition, Aitken Schwarz, Navier Stokes, Penalty Method, Parallel, Fast Solver

Abstract

The objective of this paper is to present a fast parallel incompressible Navier Stokes (NS) simulator that has enough versatility to apply to various endovascular situations with ease in medical image integration. Our long term goal is to bring NS simulation to surgeons as an additional routine image modality that can complement existing medical imaging channels. Given a medical image describing the geometry of an artery, we are able to simulate the blood flow through an artery close to clinical conditions. To achieve this goal, our interface relies on the combination of the L_2 penalty method to deal with complex geometry for solving the NS equations. We present here a high performance portable domain decomposition solver based on a tuning of the fastest linear solver depending on the subdomain and the processor architecture. This paper provides a detailed proof of concept in the two dimensional case. The three dimensional results will be reported in a companion paper.

[*]Department of Computer Science, University of Houston, Houston TX, 77204, TX
[*]Department of Computer Science, University of Houston, Houston TX, 77204, TX

A Fast Navier Stokes Flow Simulation Tool for Image Based CFD

Bilel Hadri,[*] Marc Garbey[*]

Abstract

The objective of this paper is to present a fast parallel incompressible Navier Stokes (NS) simulator that has enough versatility to apply to various endovascular situations with ease in medical image integration. Our long term goal is to bring NS simulation to surgeons as an additional routine image modality that can complement existing medical imaging channels. Given a medical image describing the geometry of an artery, we are able to simulate the blood flow through an artery close to clinical conditions. To achieve this goal, our interface relies on the combination of the L_2 penalty method to deal with complex geometry for solving the NS equations. We present here a high performance portable domain decomposition solver based on a tuning of the fastest linear solver depending on the subdomain and the processor architecture. This paper provides a detailed proof of concept in the two dimensional case. The three dimensional results will be reported in a companion paper.

Index Terms

Domain Decomposition, Aitken Schwarz, Navier Stokes, Penalty Method, Parallel, Fast Solver

I. Introduction and Motivation

To make clinical progress, it is important to accumulate a large data base of hemodynamic studies and get fast patient specific simulations. Our goal is to provide to medical doctors (MD) a computational environment that may bring hemodynamic simulation at the same level as medical imaging, (i.e, a modality that can be used routinely in clinical environment). Current simulators relying on the finite element method provide blood flow simulations but it usually takes several hours to get the results. Mesh generation and Finite Element (FE) [1], [2] solution procedures are time consuming and require tuning for each specific situation. This delay may not be acceptable to incorporate the flow data into the pretreatment planning process.

We present an integrated approach to quickly compute an incompressible Navier Stokes (NS) flow in a section of a large blood vessel using medical imaging data. The goal is essentially to provide a first order approximation of some main quantities of interest in cardiovascular disease: the shear stress and the pressure on the wall. While there is a large number of parameters influencing blood flow conditions, those quantities seem to be critical in the understanding of aneurysm formation, plaque building, and wall plasticity. The set of incompressible NS equations seems to be an acceptable level of approximation to build shear stress and pressure indicators for *large* artery flow in clinical conditions, while the non Newtonian flow model should definitively be used for smaller vessels [3].

The NS solver presented in this paper relies on the L_2 penalty approach pioneered by Caltagirone [4] and co-workers and combines nicely with a level set method based on the Mumford-Shah energy model [5]. The discretization grid can be a Cartesian grid no matter the complexity of the artery shape. We can use fast domain decomposition solvers such as the Aitken-Schwarz method [6] to get portable performance on a wide range of parallel computer architectures. A general description of the method developed in the context of Matlab-MPI was published in [7]. This paper further discusses toward real time simulation. We present here a systematic way to optimize the choice of the subdomain solver that can be borrowed from any existing numerical libraries. The fastest subdomain solver on modern scalar processors is not always the one

[*]Department of Computer Science, University of Houston, Houston TX, 77204, TX
[*]Department of Computer Science, University of Houston, Houston TX, 77204, TX

that has the least arithmetic complexity. Thanks to our Domain Decomposition (DD) framework, we can get performance portability by switching automatically from one numerical library to another based on systematic preprocessing tests and/or track records of previous runs. A surface response technique complements the design of our adaptive software by modeling a posteriori the performance as a function of the rectangular subdomain size, choice of algorithm and processor type.

This paper is a proof of concept based on a two dimensional incompressible NS solver that provides close to real time simulation for arbitrary shaped geometry. Preliminary results for carotid bifurcation in three space dimensions are reported in [8]. This simulator is part of a global hemodynamic software suite that has a data base and web interface to make the system as easy and practical as possible for MDs [9].

The plan of this paper is as follows. We describe in section 2 the formulation of the problem and the method. In section 3, we present the adaptive DD elliptic solver for the pressure equation. Section 4 reports on the performance portability of this elliptic solver. In section 5, we discuss the parallel efficiency of our blood flow simulator. Section 6 is our conclusion.

II. Method

In this section, we describe the penalization method, discretization and time stepping of the incompressible NS code.

A. PDE Formulation

We consider the velocity pressure formulation of the NS equations in a rectangular domain $\Omega = [0, L_x] \times [0, L_y]$:

$$\begin{aligned} \partial_t U + (U.\nabla)U + \nabla p - \nu \nabla.(\nabla U) &= f \text{ in } \Omega \\ div(U) &= 0 \text{ in } \Omega \\ U &= g \text{ on } \partial\Omega \end{aligned}$$

$U(x, y, t)$ stands for the velocity with its x and y component respectively u and v. We denote $p(x, y, t)$ as the pressure of the fluid. ν represents the kinematic viscosity of the fluid.

Considering an immersed boundary approach, the domain Ω is decomposed into a fluid subdomain Ω_f and a wall subdomain Ω_w. In the L_2 penalty method the right hand side f is a forcing term that contains a mask function Λ_{Ω_w}

$$\Lambda_{\Omega_w}(x, y) = 1, \text{ if } (x, y) \in \Omega_w, \ 0 \text{ elsewhere},$$

and is defined as follows

$$f = -\frac{1}{\eta}\Lambda_{\Omega_w} \{U - U_w(t)\}.$$

U_w is the prescribed velocity of the moving wall and η is a small positive parameter.

A formal asymptotic analysis helps us to understand how this penalty method matches the no slip boundary condition on the interface $S_w^f = \bar{\Omega}_f \bigcap \bar{\Omega}_w$ as $\eta \to 0$. Let us define the following asymptotic expansion:

$$U = U_0 \ + \ \eta \, U_1, \ p \ = p_0 \ + \ \eta \, p_1.$$

Formally we obtained at leading order,

$$\frac{1}{\eta}\Lambda_{\Omega_w} \{U_0 - U_w(t)\} = 0,$$

that is

$$U_0 = U_w, \text{ for } (x, y) \in \Omega_w.$$

The leading order terms U_0 and p_0 in the fluid domain Ω_f satisfy the standard set of NS equations:

$$\partial_t U_0 + (U_0.\nabla)U_0 + \nabla p_0 - \nu\nabla.(\nabla U_0) = 0, \text{ in } \Omega_f$$

$$div(U_0) \;=\; 0, \text{ in } \Omega.$$

At next order we have in Ω_w,

$$\nabla p_0 \;+\; U_1 \;+ Q_w = 0, \tag{1}$$

where

$$Q_w \;=\; \partial_t U_w + (U_w.\nabla)U_w - \nu\nabla.(\nabla U_w).$$

Further the wall motion U_w must be divergence free.

At next order we have in Ω_f,

$$\partial_t U_1 + (U_0.\nabla)U_1 + (U_1.\nabla)U_0 + \nabla p_1 - \nu\nabla.(\nabla U_1) = 0,$$

with

$$div(U_1) = 0.$$

In the simplest situation where $U_w \equiv 0$, we observe that the motion of the flow is driven by the pressure following a classical Darcy law. η stands for a small permeability. To summarize as $\eta \to 0$, the flow evolution is dominated by the NS equations inside the artery, and by the Darcy law with very small permeability inside the wall. This actually corresponds to a standard multiscale model of blood flow in the main arteries. From the analytical point of view it was shown in [10] for fixed wall, i.e. $U_w \equiv 0$, that the convergence order of the penalty method is of order $\eta^{\frac{3}{4}}$, in the fluid domain, and $\eta^{\frac{1}{4}}$ in the wall.

The mask function Λ_{Ω_w} is obtained *directly* by a level set method based on the Mumford-Shah Model [5] that deliver the image segmentation. We refer to [7] for more details. Because this Mask function is approximated on a regular Cartesian grid, the method can be only first order in space. Let us introduce now the discretization of the PDE problem.

B. Discretization & Time Stepping

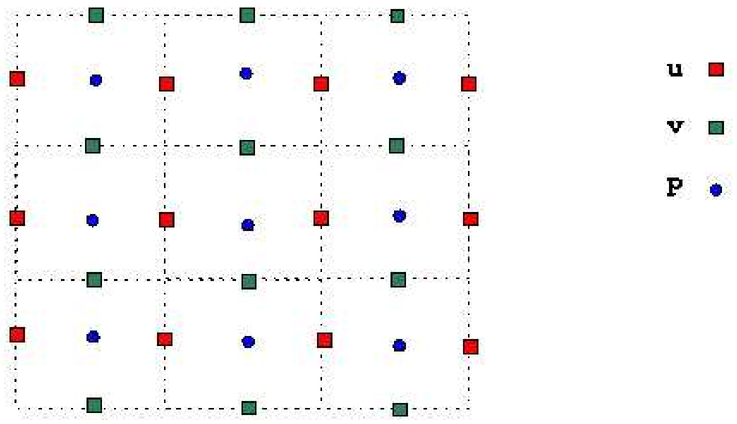

Fig. 1. Spatial discretization

The discretization of the NS equations is done with finite differences on a Cartesian grid following the standard staggered grid method as described in Figure 1 .

The grid approximation for velocity and pressure are defined as follows

$$u\left(i\,h_x,\left(j+\frac{1}{2}\right)\,h_y\right),\ i=0\ldots N_x,\ j=0\ldots N_y-1,$$

$$v\left(\left(i+\frac{1}{2}\right)\,h_x, j\,h_y\right),\ i=0\ldots N_x-1,\ j=0\ldots N_y,$$

$$p\left(\left(i+\frac{1}{2}\right)\,h_x,\left(j+\frac{1}{2}\right)\,h_y\right),\ i=0\ldots N_x-1,\ j=0\ldots N_y-1$$

on the staggered mesh, of space step $h_x=\frac{L_x}{N_x},\ h_y=\frac{L_y}{N_y}$.

Regarding the resolution of the equation, we use a standard projection method [11] as follows :

- Step 1: prediction of the velocity $\hat{u}^{k+1}$ by solving either explicitly:

$$\frac{\hat{u}^{k+1}-u^{k,*}}{\Delta t}-\nu\Delta u^k=f^{k+1}-\nabla p^k, \tag{2}$$

or implicitly

$$\frac{\hat{u}^{k+1}-u^{k,*}}{\Delta t}-\nu\Delta u^{k+1}=f^{k+1}-\nabla p^k, \tag{3}$$

in $(0,L_x)\times(0,L_y)$ with the boundary condition $\hat{u}^{k+1}=g$ on $\partial\Omega$. We have in (2) or (3) a first order approximation in time of the total derivative. The convective term is taken care by the term $u^{k,*}$ that is obtained from the method of characteristics. We use a second order interpolation scheme in space and a backward Euler scheme in time to reconstruct the root of the characteristic. This technique satisfies a maximum principal and the emphasis is on the robustness of the method.

- Step 2 : projection of the predicted velocity to the space of divergence free functions.

$$-div(\nabla\delta p)=-\frac{1}{\Delta t}div(\hat{u}^{k+1});u^{k+1}=\hat{u}^{k+1}-\Delta t\ \delta p \tag{4}$$

$$p^{k+1}=p^k+\delta p. \tag{5}$$

These two steps of the NS solver have very uneven weight in the simulation elapsed time. In the explicit scheme (2), the momentum equations are readily solved. The implicit version (3) requires few steps of a relaxation scheme or possibly a multigrid algorithm, because the corresponding Helmholtz operator is close to the identity operator. Overall Step 1 of the projection scheme has order $N_x\times N_y$ complexity.

In theory one can solve the Poisson problem thanks to full multigrid with the same arithmetic linear complexity. In practice things are very different on parallel system. It is generally agreed that the most time consuming routine in the code is the resolution of the linear problem for pressure. We will now concentrate on this practical issue and present a solver that scales on parallel systems quasi linearly.

III. Elliptic Solver for domain decomposition

In this section, we will recall first the Aitken-Schwarz decomposition technique [6] that is at the core of the method.

A. Aitken-Schwarz Algorithm

This DD solver for elliptic problems that was first introduced in [12] provides a framework to build high performance algorithms that are tolerant to low bandwidth and high latency. This technique can be combined with standard DD that scales well on uniform parallel architecture, where CPU power is well balanced by network performance. Our method post-processes standard block-wise relaxation schemes such as the Schwarz method, [13], [14], [15] with Aitken like acceleration of the sequences of interfaces produced by the Schwarz method. We briefly describe the numerical ideas behind the Aitken-Schwarz (AS) method. We refer to [6] for a detailed description of this method. Let us give an overview of this algorithm with a problem that corresponds to the pressure correction (4). We apply homogeneous Dirichlet boundary conditions along the vertical sides and homogeneous Neumann boundary conditions on the horizontal sides of the rectangular domain.

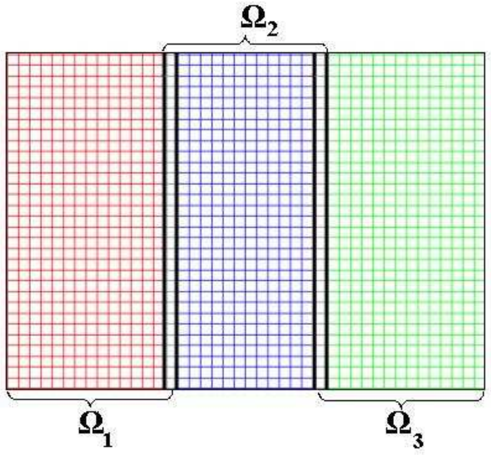

Fig. 2. Domain Decomposition

As described in Figure 2, the domain is decomposed into a unidirectional partition of overlapping domains.

For simplicity, we describe the algorithm with a domain Ω decomposed into two overlapping sub-domains: $\Omega = \Omega_1 \cup \Omega_2$. The Additive Schwarz algorithm consists of repeating the following iteration until convergence:

$$\Delta u_1^{n+1} = RHS \text{ in } \Omega_1, \;\; \Delta u_2^{n+1} = RHS \text{ in } \Omega_2, \tag{6}$$
$$u_{1|\Gamma_1}^{n+1} = u_{2|\Gamma_1}^{n}, \;\; u_{2|\Gamma_2}^{n+1} = u_{1|\Gamma_2}^{n},$$

Because we have homogeneous Neumann boundary conditions on the horizontal sides of the domain, a cosine expansion of the trace of the solution on the interface provides a diagonalization of the trace transfer operator :

$$\left(u_{1|\Gamma_1}^{n}, u_{2|\Gamma_2}^{n}\right) \xrightarrow{T} \left(u_{1|\Gamma_1}^{n+1}, u_{2|\Gamma_2}^{n+1}\right). \tag{7}$$

The Poisson problem decomposes into a set of independent two point boundary value problems:

$$\frac{\partial^2 \hat{u}_k(x)}{\partial x^2} - \mu_k \, \hat{u}_k(x) = \widehat{RHS}_k, \;\; \forall k. \tag{8}$$

We denote T_k as the trace transfer operator for each wave component of the interface:

$$\left(\hat{u}_{1|\Gamma_1}^{n,k} - \hat{u}_{\Gamma_1}^{k}, \hat{u}_{2|\Gamma_2}^{n,k} - \hat{u}_{\Gamma_2}^{k}\right) \xrightarrow{T_k} \left(\hat{u}_{1|\Gamma_1}^{n+1,k} - \hat{u}_{\Gamma_1}^{k}, \hat{u}_{2|\Gamma_2}^{n+1,k} - \hat{u}_{\Gamma_2}^{k}\right), \;\; \forall k. \tag{9}$$

The operators T_k are linear and the sequences $\left\{\hat{u}^n_{1|\Gamma_1}\right\}$ and $\left\{\hat{u}^n_{2|\Gamma_2}\right\}$ have linear convergence. This is expressed by the set of linear equations,

$$\hat{u}^{n+1,k}_{1|\Gamma_2} - \hat{u}^k_{\Gamma_2} = \hat{\delta^1_k}\left(\hat{u}^{n,k}_{2|\Gamma_1} - \hat{u}^k_{\Gamma_1}\right), \quad \hat{u}^{n+1,k}_{2|\Gamma_1} - \hat{u}^k_{\Gamma_1} = \hat{\delta^2_k}\left(\hat{u}^{n,k}_{1|\Gamma_2} - \hat{u}^k_{\Gamma_2}\right), \tag{10}$$

where $\hat{\delta^1_k}$ and $\hat{\delta^2_k}$ are the so called damping factors associated with each subdomain Ω_1 and Ω_2. These damping factors are computed analytically from the eigenvalues of the operators. We apply an Aitken acceleration separately to each wave coefficient in order to get the exact limit of the sequence on the interfaces based on the first Schwarz iterate. It consists of simply solving the 2×2 linear system (10) where $n = 0$, of unknown $(\hat{u}^k_{\Gamma_1}, \hat{u}^k_{\Gamma_1})$. Finally, the exact solution at the artificial interfaces Γ_i is reconstructed in physical space from its discrete trigonometric expansion.

For the non-homogeneous mixed boundary conditions a preprocessing step is added which consists of solving the zero mode equation with appropriate boundary conditions:

$$\frac{\partial^2 \hat{u}_0(x)}{\partial x^2} = \widehat{RHS}_0, \tag{11}$$

This method is easily generalized for an arbitrary number of subdomain and the matrix of the linear system corresponding to (10) has a pentadiagonal structure [6].

Here are the steps of the Aitken Schwarz algorithm:

- step 1: Compute analytically each damping factor for each wave number. This need to be done only once.
- step 2: Solve the zero mode equation (11).
- step 3: Perform one additive Schwarz iterate in parallel with the two dimensional subdomain solver of choice.
- step 4: Compute the cosine expansion of the traces of the solutions of Step 3, on the artificial interfaces.
- step 5: Apply the Aitken acceleration separately to each wave coefficients in order to get the limit expressed in the cosine functions vector basis.
- step 6: Transfer back the interface limit values into the physical space.
- step 7: Compute the solution for each subdomain in parallel using the boundary values obtained from step 6 .

To adapt this DD solver to a broad variety of computer architectures, one needs to adapt the choice of the subdomain solver for each type of processor configuration. We present in the next section a methodology for this purpose.

B. Interface to Subdomain Solver

There are many different classes of elliptic solver based either on direct methods or iterative methods that perform differently depending on the grid size, the architecture of the processor and eventually the physical parameters of the problem.

We have written an interface software [16] to reuse a broad variety of existing linear algebra softwares such as LU factorization, Krylov methods with an Incomplete LU preconditioner and Geometric or Algebraic multigrids solvers. Most of these methods have been implemented respectively in Lapack, Sparskit [17] and Hypre [18] and are gathered in this interface.

We will restrict ourselves to these three libraries that are representative of the state of the art for the resolution of a linear system. It is usually a cumbersome exercise for the programmer to call theses libraries in his code, because of the multiplicity of available options for each iterative solver, and because each of these softwares has a different language and/or style of coding with different storage. This explain somewhat the success of metalanguages like Matlab, that speed up dramatically the code development and hide the complexity of the library calls, but at the expense of performance.

TABLE I

DESCRIPTION OF THE MACHINES

Name of the Machine	**# of Processors**	**Processor Architecture**	**RAM Size**	**Compiler Name**
Atlantis (HP cluster)	308	dual Itanium2 900Mhz	4 GB	Intel8.1
Stokes (Appro cluster)	64	dual AMD Athlon1800	2GB	Intel8.1
Altix (SGI)	16	Intel Itanium 1.3GHz	16GB	Intel8.1
Marvin 8-way	8	AMD Opteron 2GHz	32GB	Pathscale2.1
Darwin (Apple Cluster)	20	dual 2 GHz Apple G5	2 GB	Xlf
Iroise (Dec)	4	Alpha EV6 500 MHz	2 GB	Compac Fortran

Our interface assumes that each subdomain has regular data structure with $n_x \times n_y$ grid points, and therefore that the discrete operator has a sparse multi-diagonal structure. This is a dramatic simplification that makes the software development easy. But it is a relevant assumption for applications based on immersed boundary technique on Cartesian grids.

This interface has the simplicity of Matlab command since it is based on single function calls to solve the subdomain problem with the appropriate parameters and it keeps the efficiency of the original Fortran or C solver library. The main focus is to help the user speed up his code without painful programming. In particular our friendly user interface can guide the user to configure the choice of solver that he wants to fill in a transparent way the numerous parameters that the original linear library function call requires.

For our NS application, the dimension of the subdomain obviously depends on the number of processors used in a run. The method of choice for the subdomain solution process should then depend on the number of processors as well.

We have used a simple surface response methodology [19] to model the performance of each subdomain solver available through our interface as a function of the subdomain dimension and processor type. The idea is to use the surface response to predict for each parallel run what is the best solver, and obtain performance portability.

We have shown in [16] that a least square quadratic approximation of the elapsed time as follows,

$$E(n_x, n_y) = \beta_0 + \beta_1 n_x + \beta_2 n_y + \beta_3 n_x^2 + \beta_4 n_y^2 + \beta_5 n_x n_y. \tag{12}$$

based on a dozen of different configurations runs $n_x \times n_y$ with $n_x,\ n_y \in (10, 200)$ is sufficient enough to predict with few percent accuracy the performance of Lapack and Sparskit solvers with subdomains in that range of dimensions.

Unfortunately this is not true with Hypre, because the algebraic multigrid implementation might be very sensitive to the grid size. It can be noticed that one should use statistical sampling methods to estimate a posteriori the reliability of the surface response model.

We report extensively in the following on the performance of the subdomain solvers that we observed in the framework of our application.

C. Performance Evaluation of Subdomain Solver

We give the descriptions of the machine used for our performance evaluation in Table I.

Altix, Marvin and Iroise are shared memory machines while Atlantis, Darwin and Stokes are distributed memory machines with nodes interconnected by a giga-bit Ethernet network.

In order to compare the different solvers, we performed the surface response analysis of Sect 3.2 on each system.

Figures 3 and 4 are for Iroise. It shows a two dimensional color map in the two dimensional space (nx, ny), where nx on the horizontal axis and ny on the vertical axis are the dimensions of the rectangular grids of the domain. For each map, we compare two methods: each color corresponds to the region where one method is better than the other. From these results, we can notice that LU performs well for small

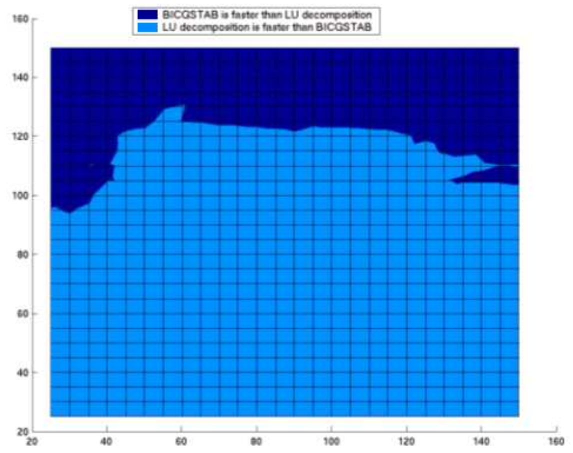

Fig. 3. Comparison between Bicgstab and LU decomposition

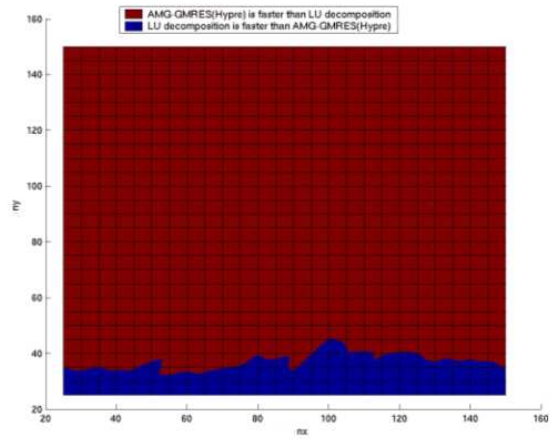

Fig. 4. Comparison between Hypre and LU decomposition

size, while, iterative solvers such as Bicgstab (Krylov method) and Amg-Gmres (multigrid method) give the fastest results for large grid size.

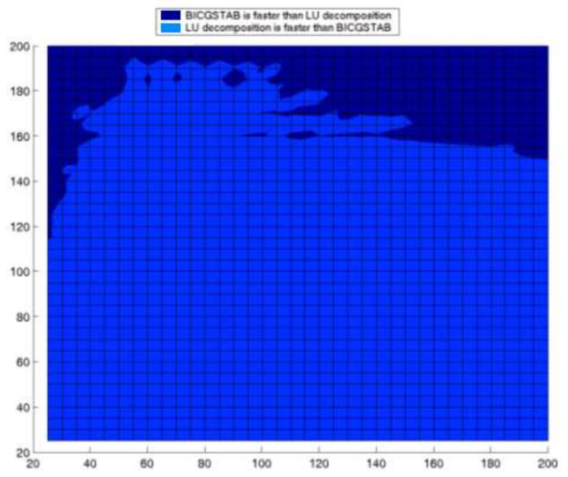

Fig. 5. Comparison between Bicgstab and LU decomposition on Stokes

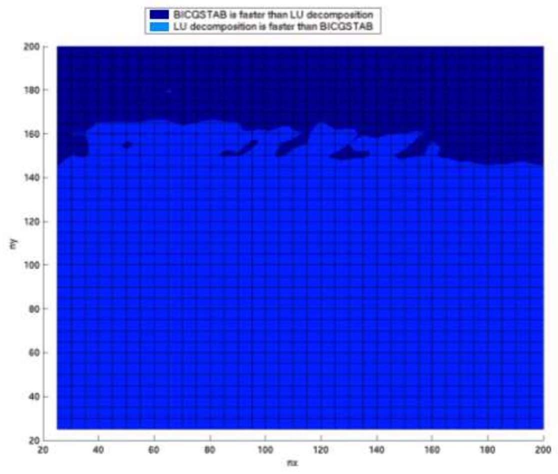

Fig. 6. Comparison between Bicgstab and LU decomposition on Atlantis

Figures 5 and 6 represent the performances on different processors architectures, respectively a dual processors AMD 1800+ with 2GB of RAM and a dual processors 900 MHz Itanium2 with 3 GB of RAM for the same (direct) solver. We clearly see the difference of the region of the area where LU is faster than the iterative method. We can build a lower order approximation of the interface position in the (nx, ny) plan that separates these two areas, and have therefore a simple criterion to select the best solver as a function of the grid size.

Through these experiments, the optimum choice of the solver for each subdomain, i.e., the solver that processes the subdomain in the shortest elapsed time, depends on the type of subdomain, the fact that we reuse or not the same preconditioner or decomposition of the operator, the architecture of the processor and the size of the problem. One can document the performance of the subdomain solver for every configuration by using systematic testing in batch procedure to construct meaningful surface response.

Thanks to the surface response model, and consequently a low order approximation of the interface between the best performances area in Figures 5 and 6, we can compress dramatically the information we need to select which one of the subdomain solver is the best for a specific run configuration. The performance model is built with a limited number of runs per processor configuration. Statistical sampling can build confidence intervals on the surface response model. Let us now study the parallel performances of the AS solver.

IV. Parallel Elliptic Solver

In this section, we will present the parallel performance of the AS algorithm by reporting on the speedup and the scalability of the parallel elliptic solver for the pressure correction problem. Furthermore, we will show that the model presented in the previous section may not be valid for multidomain and that the performance model depends on the number of processors . Finally, we will compare our method with existing parallel libraries such as PETSc and SuperLU.

A. Parallel Performances

The parallel performances of the AS method are based on the Poisson equation in a rectangular domain with Dirichlet boundary conditions. Thanks to our use of immersed boundary techniques we can stick to that simple academic problem.

1) Speedup: First, let us look at the behavior of the speedup.

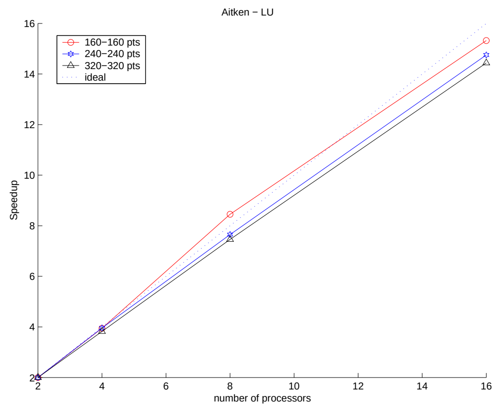

Fig. 7. Speedup of the AS method solved by LU in each subdomain

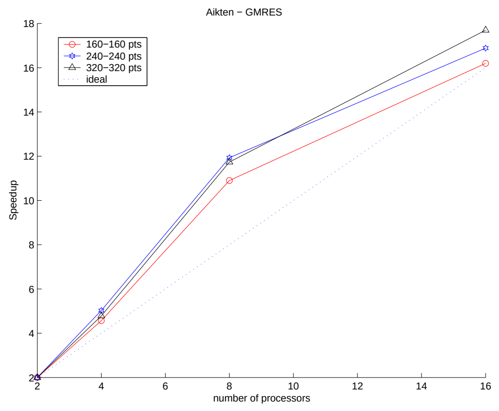

Fig. 8. Speedup of the AS method solved by krylov in each subdomain

We note from Figures 7 and 8 that AS performs very well on different size problems since we observe a quasi linear speedup close to the ideal one. Further, the Krylov method seems to be more sensitive to the cache memory effect, since we have a superlinear speedup.

We run the code on different clusters described previously in Table 1. Figures 9 and 10 show respectively the elapsed time in seconds for a medium size problem (480 by 240 unknowns) and a large size problem (640 by 320 unknowns).

From Figure 11, one observes that the speedup obtained on different machines are very consistent. Although the machines are different, the AS algorithm keeps a linear and ideal speedup behavior with the LU solver.

2) Scalability: In this section, we do a scalability test that consist to increase the size of the problem linearly with the number of processors. To be more precise we keep the exact subdomain size no matter the number of processors. This may not be realistic in applications because the aspect ratio of the overall domain will change with the number of processors. This will be addressed in Section 5 . It is however a good test to detect the overhead induces by the Aitken acceleration process. To simplify the presentation we present here a scalability test on Stokes and Atlantis with two extremes configurations that are respectively 2 and 36 processors - see Table II. The number of unknowns per processors is 200x200.

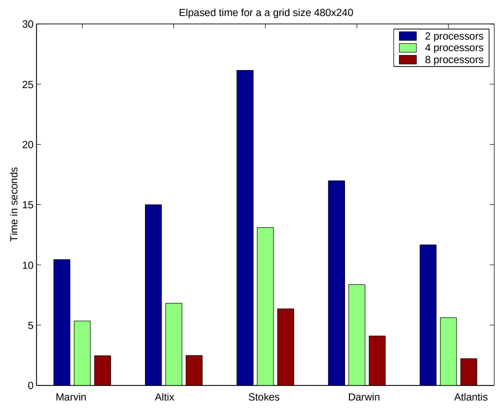

Fig. 9. Elapsed Time for a medium size problem

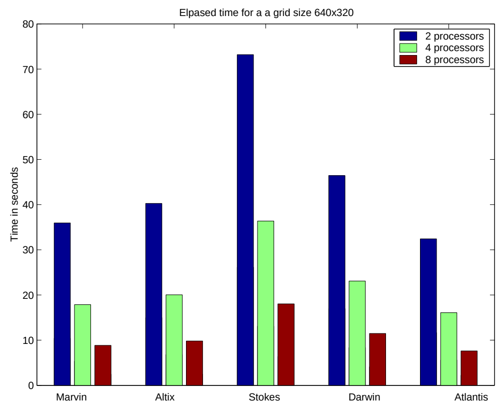

Fig. 10. Elapsed Time for a large size problem

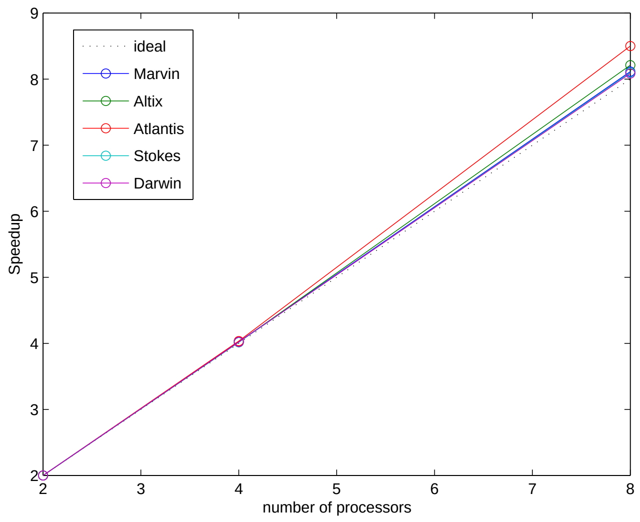

Fig. 11. Speedup on 5 machines for a domain with 640×320 unknowns

We notice that, while the distributed memory system is running a Gigabit ethernet with significant latency, we still get almost perfect efficiency. We are now going to examine how our single processor performance model behaves in parallel runs.

B. On the Limit of the Single Subdomain Solver Performance Modeling

In this section, we check the impact of the choice of the subdomain solver on the overall performance. The goal here is not to obtain the best subdomain solver from all existing methods, but rather to use the best solver from Sparskit(Krylov solver tool developed by Saad [17]) or Lapack only. Of all existing computational algorithms a full multigrid algorithm should be, in theory, the optimum.

The performance modeling on a single processor of the Poisson subdomain solver is given in Figure 12. We checked that the elapsed time with four successive runs stays consistent. Horizontal axis and vertical axis of the figure correspond to the dimension of the subdomain in each space direction. The dark blue region indicates where Sparskit (GMRES) is faster than Lapack (LU decomposition) while the light blue shows where Lapack is faster than Sparskit.

Let us compare this prediction model on a single processor and the performance of the AS algorithm with a fixed size of the subdomain per processor that corresponds to the red points in figure 12. We test

TABLE II

PERFORMANCE OF THE DIFFERENT MACHINE WITH 200 $\times$ 200 PER PROCESS

	Resolution Time		**Communication Time**		**Efficiency**
Machine Name	P=2	P=36	P=2	P=36	Percentage
Stokes	14.88	15.29	2.23E-3	4.21	97.32%
Atlantis	6.29	6.38	1.64E-3	1.92	98.59%

TABLE III

LU AND GMRES ELAPSED TIME DEPENDING ON THE NUMBER OF PROCESSORS

	2 processors		**4 processors**		**8 processors**		**16 processors**	
points per processors	**LU**	**Saad**	**LU**	**Saad**	**LU**	**Saad**	**LU**	**Saad**
100 x 160	**1.44**	2.26	**1.58**	1.87	**1.43**	1.43	**1.79**	1.82
120 x 160	**2.08**	3.19	**2.45**	2.54	2.29	**1.98**	2.75	**2.50**
140 x 160	**3.28**	4.17	3.61	**3.36**	3.54	**2.75**	3.89	**3.45**
160 x 160	**4.67**	5.21	4.81	**4.36**	4.95	**3.49**	5.37	**4.37**
180 x 160	6.46	**6.39**	6.43	**5.63**	6.52	**4.31**	7.72	**5.45**
200 x 160	8.24	**8.18**	8.89	**6.69**	8.27	**5.25**	9.69	**6.54**

the scalability of the AS algorithm, that is: we fix the size of the subdomain per processor once for all, and let the number of processors grows. The global size of the problem grows linearly with the number of processors.

Table III gives the total elapsed time to solve the problem and the number in red gives the best elapsed from both options, i.e., Lapack or Sparskit. These results have been performed on Stokes.

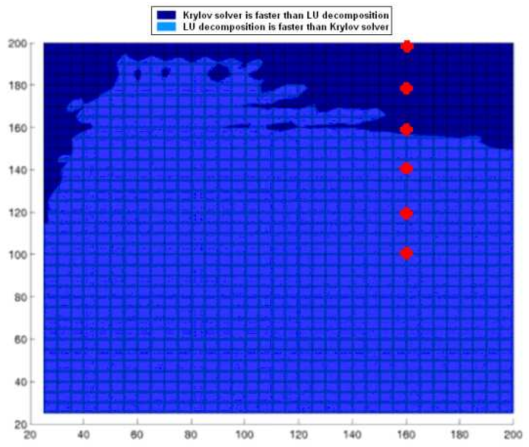

Fig. 12. Comparison between Krylov Solver (GMRES) and LU decomposition (Lapack). The dark blue shows the region where the Krylov solver is faster than LU decomposition, and the light blue shows the region where LU is faster.

We can notice that the prediction of the best subdomain solver according to Figure 12, is that one should use Sparskit (respectively Lapack) for the first space dimension above 160 (respectively below 160). This prediction is correct for the 2 processors computation. However, as the number of processors grows, this prediction is incorrect, and one should favor the Krylov solver.

Indeed Figures 13, 14 and 15 show the dependence of the surface response pattern on the number of processors and confirm our statement.

Overall, we observe that

- While the communication schedule in the parallel code is extremely regular, the performance of the parallel

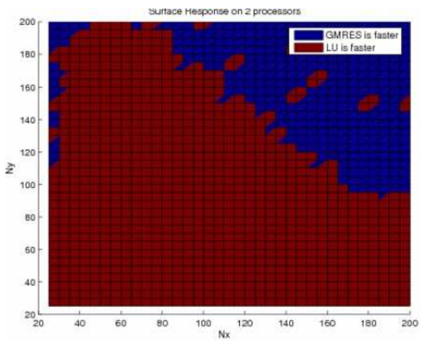

Fig. 13. Surface response with 2 processors

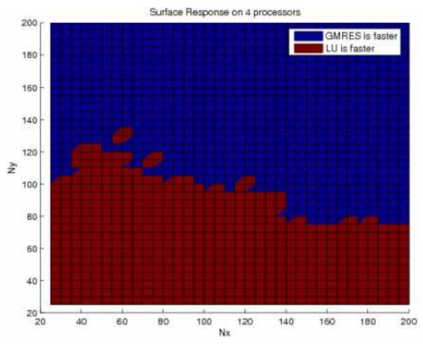

Fig. 14. Surface response with 4 processors

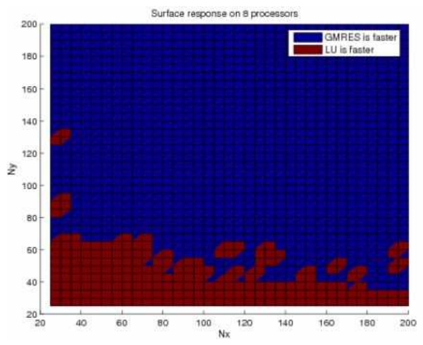

Fig. 15. Surface response with 8 processors

code is non monotonic with respect to the global size of the problem. The optimum choice of the solver is then dependent on the number of processors used.
• If one selects the best of the two subdomain's solvers for each configuration, the scalability of the code is excellent: For the smallest problems, the total elapsed time does not increase significantly as the number of processors grow while the problem size per processor stays constant. For large problems, the elapsed time decreases.

We conclude that the optimum choice of the subdomain solver should use a surface response model that includes as a third dimension the number of processors.

C. Some Comparison with Existing Parallel Software Libraries

There are several parallel software available to solve elliptic problems. We decided to compare our AS for the pressure correction with two high end scientific libraries, PETSc - *http://www-unix.mcs.anl.gov/petsc-* and SuperLU *-http://crd.lbl.gov/∼xiaoye/SuperLU-* that is a general purpose library for the direct solve of linear system. The results in this section are performed for PETSc with the distributed memory systems Stokes, and for SuperLU, with the shared memory system Marvin with the optimization level -O3 for all runs. The idea is not to state that our method is better than the two others software methods: our method has been designed specifically for the pressure correction in the context of immersed boundary method on Cartesian grid while both libraries PETSc and SuperLU are general purpose libraries. But it is not enough to show that our method scales well. We should also compare the elapsed time with modern software libraries on parallel clusters.

1) PETSc: PETSc [20] is an excellent general purposes software for comparison purpose. PETSc consists of a variety of libraries which include many linear solvers such as Lapack, Krylov solver and multigrid solver [21].

In Figure 16 , we report on the speedup performance of PETSc and AS on the same graphic, while in Figure 17, we give the elapsed time. We choose to run PETSc using V-cycle multigrid with 3 levels which seems to be the fastest method to solve the Poisson problem. The preconditioner is of Richardson type to get traditional (non-Krylov accelerated) multigrid. One has two pre and two post smoothing steps of SSOR (running independently on each process) and direct LU on the coarsest grid. To be accurate the option in the PETSc code is:

```
-dmmg_nlevels 3  mg_levels_ksp_type richardson -mg_levels_pc_type sor
 -mg_levels_pc_sor_lits 2 -mg_levels_pc_sor_local_symmetric
```

PETSc, as expected, is faster than AS with two processors and also for 3 processors. We should have used PETSc to solve the subdomain problem in AS then. However as the number of processors increases, one can observe that the multigrid solver of PETSc does not speed up well, while our method performs better. Eventually for more than three processors, AS gives a better elapsed time than the multigrid solver

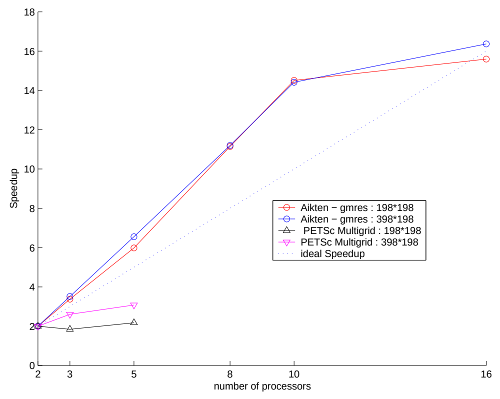

Fig. 16. Speedup between Aitken solved with GMRES and PETSc multigrid

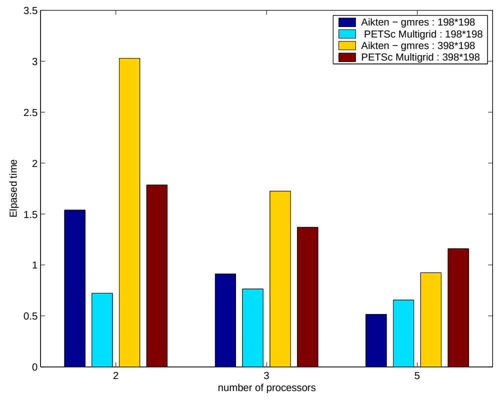

Fig. 17. Elapsed times for Aitken solved with GMRES and PETSc multigrid

and PETSc break down for 5 processors. The fact that PETSc does not have a good speedup is explained by the fact that PETSc is very sensitive to the performance of the network. The Gigabit Ethernet network we use has the same latency as fast ethernet.

We could raise the speedup and scalability of PETSc by solving a much larger problem, but the size of the problem is in principle imposed by the application rather than by the performance result. The comparison was done here with an iterative parallel solver. We will do the comparison with the parallel implementation of a direct parallel solver in the next section.

2) SuperLU: SuperLU [22] is a general purpose library for the direct solution of large, sparse, non-symmetric systems of linear equations on high performance machines.

SuperLU contains a collection of three related subroutine libraries: sequential $SuperLU$ for uniprocessors, the multithreaded version $SuperLU-MT$ for medium-size SMPs, and the MPI version $SuperLU-DIST$ for large distributed memory machines.

In Figure 18, we used the implementation for shared memory parallel machines $SuperLU-MT$, with three different test cases. The small one with a domain discretized into 100×800, the medium with 200×800 and the larger grid with 300×800 points. We compared the elapsed time between SuperLU and AS with either LU or GMRES depending on the number of processors and for small, medium and large size problem. For SuperLU, we compute the factorization and the resolution time.

From these results, we notice that for the small grid size problem, AS is faster than SuperLU and the more processors we have, the better AS performs with either GMRES or LU. For larger problems on the same number of processors we observe the opposite. Regarding the middle size, for 2 and 4 processors, SuperLU is faster however, with more than 6 processors, AS combined to GMRES is faster. With 8 processors, the AS is faster than SuperLU even with LU.

The conclusion so far is that our AS software equipped with the adaptive choice of the subdomain solver provides a fairly competitive parallel solver for the pressure correction in an immersed boundary implementation of a NS code.

We are going to investigate in the next section how this conclusion holds for the complete NS code.

V. Parallel Performances of the NS code

The benchmark problem is for an arbitrary artery tree embedded into a rectangular box. To provide a performance result independent of the exact test case under consideration, we will restrict our performance analysis to the LU subdomain solver in AS. The operation count for the pressure correction will depend only on the size of the domain and number of processors. In the same spirit we will use the explicit scheme (2) for the momentum equation.

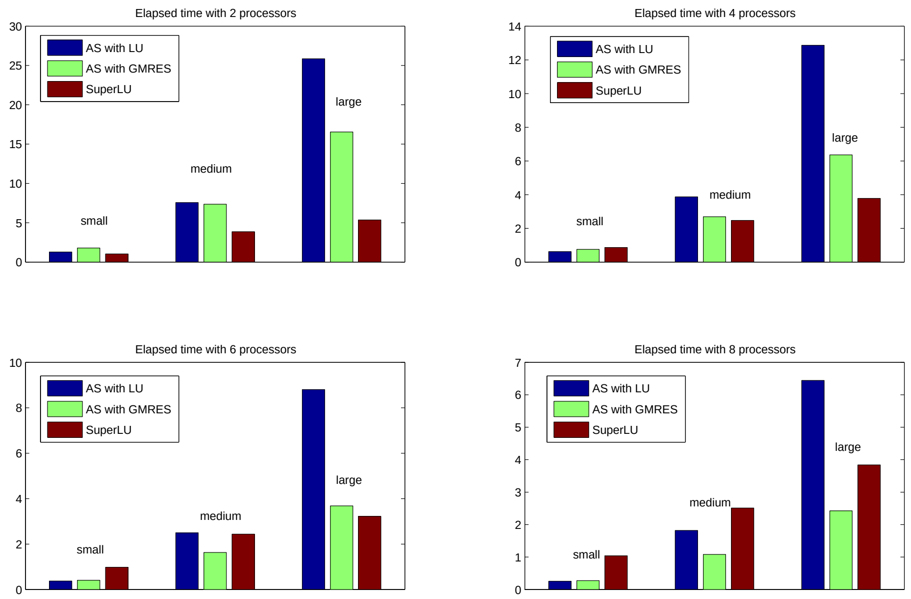

Fig. 18. Comparison of the Elapsed time between SuperLU and AS with either LU or GMRES depending on the number of processor and for small, medium and large size problems

A number of steady and unsteady, two dimensional as well as three dimensional problems have been benchmarked in [7], [8], [9] and compared for verification with a finite element simulation run with the commercial code ADINA.

Figures 19 to 21 show an example of a simple artificial stenose steady calculation. The velocity profile in the inlet of the pipe corresponds to a Poiseuille flow with a velocity of maximum value one. The length of pipe is 70 mm , the width is 10 mm. The obstacle is located at the first third of the pipe and covers 70% of the diameter of the pipe. The Reynolds number is of the order of 200.

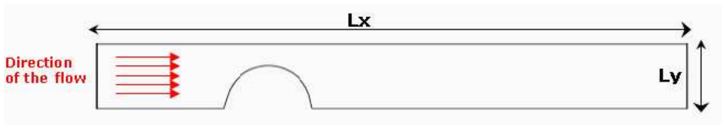

Fig. 19. Benchmark problem with artificial 70% stenose.

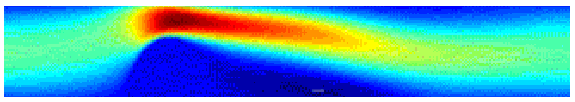

Fig. 20. Contour plot of u velocity component

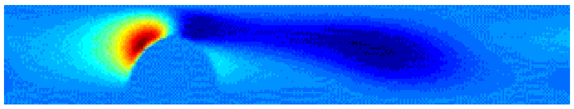

Fig. 21. Contour plot of v velocity component

In Table IV, we report on the average elapsed time per time step with the NS code running on the 8 Way Opteron system Marvin.

Figure 22 shows the speedup of this code with different grid sizes. This two dimensional code is extremely fast and can provide in practice the simulation of a cardiac cycle every few seconds.

The scalability of the NS code is very similar to what we obtained for the pressure solver. In Figure 23

TABLE IV

ELAPSED TIME FOR ONE STEP TIME DEPENDING ON THE NUMBER OF PROCESSORS

$Nx \times Ny$	p=1	p=2	p=3	p=4	p=6	p=8
150x50	0.0171	0.0144	0.0096	0.0072	0.0059	0.0049
225x75	0.0538	0.0431	0.0273	0.0213	0.0161	0.0131
300x100	0.1221	0.0965	0.0623	0.0479	0.0362	0.0288
450x150	0.3993	0.2738	0.1872	0.1368	0.1047	0.0853

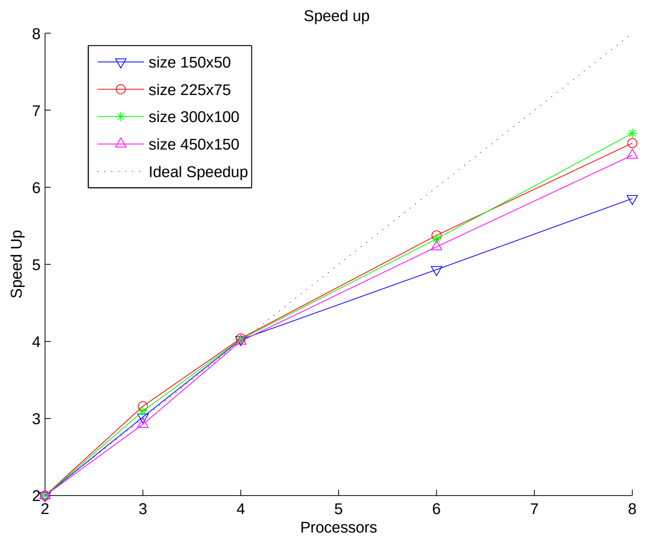

Fig. 22. Speedup of the Navier Stokes code.

we keep the aspect ratio of the grid $\frac{h_x}{h_y}$ the same: three tests have been performed respectively with a problem of size 141×567 for two processors, 200×801 on 4 processors, and 283×1129 on 8 processors. GMRES gives a better scalability result compared to LU because the growth of the bandwidth of the matrix as the number of subdomains increases penalizes the LU solver. On the contrary if we keep the size of the subdomain fixed, here 201×201 in Figure 24, we obtain a very good scalability of the code no matter the subdomain solver.

Compared to the MatlabMPI code implementation of [7], the Fortran code presented in this paper is much faster. We obtain an acceleration by a factor 7 for the medium size problem (400 by 100) that fits the average application needs. This Fortran application is not only faster but it allows a resolution of a larger grid size which is essential for either solution verification and/or three dimensional simulation. Figure 25 shows encouraging results for two dimensional large problems. We obtain a linear speedup on Atlantis up to 80 processors for a 4000×400 problems. This simulation might be used to have accurate results with a larger Reynolds number than usual in endovascular cases.

In the next section we will conclude this study.

VI. CONCLUSION

We have presented a versatile and robust engineering solution for the simulation of the incompressible set of Navier Stokes equation in the context of blood flow application. This paper was focussed on the design of the algorithm and software to obtain high performance and portability on a broad spectrum of computer architecture. The results obtained in this paper are a proof of concept, since they have been restricted to two space dimensions. Our preliminary results in [8], [9] seems to show that the conclusion of this paper carry on with true three dimensional space unsteady simulation and complex geometry such as the carotid bifurcation in the neck. However realistic blood flow simulation requires more attention to inlet/outlet boundary condition

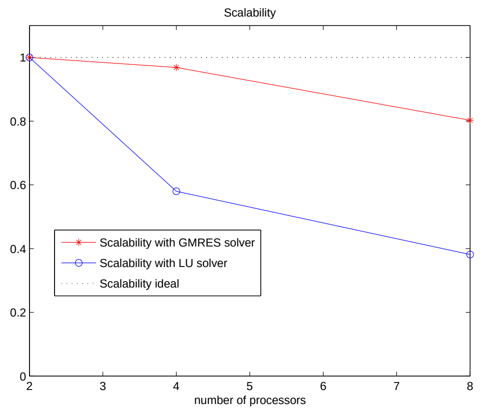

Fig. 23. Scalability of the Navier Stokes code on Marvin

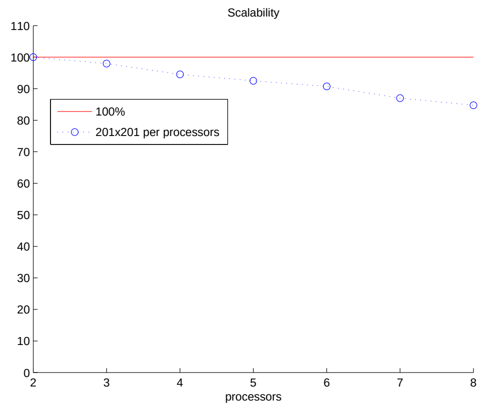

Fig. 24. Scalability of the Navier Stokes code with LU solver on Marvin

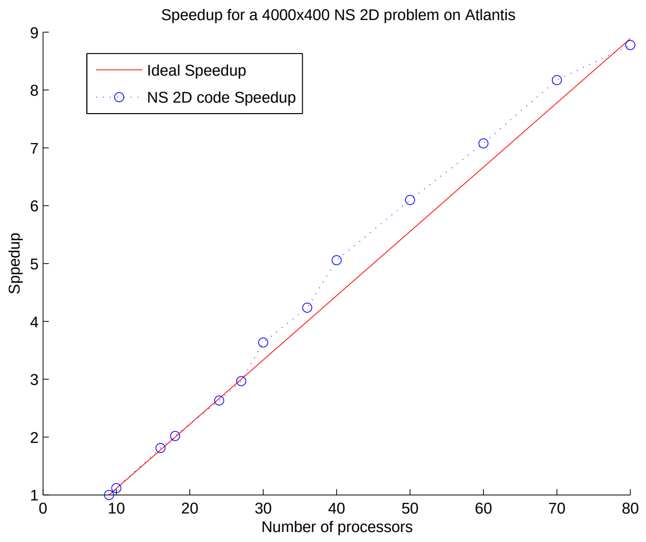

Fig. 25. Speedup of the Navier Stokes code on Atlantis.

and uncertainty assessment than we had in this paper. As a matter of fact there is a lot of variability on blood flow conditions even for the same individual during a short period of time and depending on his level of metabolic activity. We believe that the ability to run close to real time hemodynamic simulation is an important part of the simulation, since one isolated simulations is often meaning less and statistic on dozen of simulation with variable input are really what it is needed.

Acknowledgment This research was partially supported by Awards 0305405 and 0521527 from the National Science Foundation.

References

[1] R. Lohner and J. Cebral, “Parallel advancing front grid generation,” *R. Lohner and J.R. Cebral, Parallel Advancing Front Grid Generation, submitted to 8th International Meshing Roundtable, South Lake Tahoe, California, October 10-13*, 1999.

[2] L. Baghdadi, D. Steinman, and H. Ladak, “Template-based finite-element mesh generation from medical images,” *Comput Methods Programs Biomed*, pp. 77(1):11–21, 2005 Jan.

[3] C. Chen, K. Coyle, J. Hughes, A.B.Lumsden, and D. Ku, “Reduced blood flow accelerates intimal hyperplasia in endarterectomized canine arteries,” *Cardiovascular Surgery*, vol. 5, pp. 161–168(8), April 1997.

[4] E. Arquis and J. Caltagirone, “Sur les conditions hydrodynamiques au voisinage d’une interface milieu fluide-milieux poreux: Application a la convection naturelle,” *CRAS, Paris II*, vol. 299, pp. pp1–4, 1984.

[5] T. Chan and L.A.Vese, "Active contours without edges," *IEE Transaction on Image Processing*, vol. 10 (2), pp. pp 266–277, 2001.

[6] M. Garbey and D. Dervout, "On some aitken like acceleration of the schwarz method," *Int. J. for Numerical Methods in Fluids*, vol. 40, pp. 1493–1513, 2002.

[7] M. Garbey and F. Pacull, "A versatile incompressible navier stokes solver for blood flow application."

[8] M. Garbey, B. Hadri, V. Hilford, and C. Karmonik, "Parallel image-based hemodynamic simulator," *ICSNC'07 , workshop HPC-Bio07, Cap Esterel, France*, 2007.

[9] V. Hilford, M. Garbey, S. Vadakattu, and C. Addison, "Wimi - web interface to medical information, sede-2007, 16th international conference on software engineering and data engineering, best award paper," 2006.

[10] P. Angot, C. Bruneau, and P.Fabrie, "A penalisation method to take into account obstacles in viscous flows," *Numerische Mathematik*, vol. 81, pp. pp 497–520, 1999.

[11] A. Chorin, "The numerical solution on the navier-stokes equations for an incompressible fluid," *Bull. Ameri. Math. Soc.*, vol. 73, p. 928, 1967.

[12] M. Garbey and D. Tromeur-Dervout, "Operator splitting and domain decomposition for multiclusters," in *Proc. Int. Conf. Parallel CFD99*, D. Keyes, A. Ecer, N. Satofuka, P. Fox, and J. Periaux, Eds. North-Holland, 1999, pp. 27–36.

[13] P. L. Lions, "On the Schwarz alternating method I," in *Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, Eds. Philadelphia: Society for Industrial and Applied Mathematics, 1988, pp. 1–42.

[14] Schwarz, "Gesammelte mathematische abhandlungen," *2nd Ed Springer Berlin*, vol. 2, pp. 133–145, 1980, vierteljahresschrift der Naturforschenden Gesellschaft, Zurich, Vol. 15, 1st Edition, 1870, pp. 272-286.

[15] B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[16] M. Garbey, W. Shyy, B. Hadri, and E. Rougetet, "Efficient solution techniques for cfd and heat transfer problems," in *ASME Heat Transfer/Fluids Engineering Summer Conference*, 2004.

[17] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. SIAM, 2003.

[18] R. D. Falgout and U. M. Yang, "Hypre: A library of high performance preconditioners," *Lecture Notes in Computer Science*, vol. 2331, pp. 632–641, 2002.

[19] D. Montgomery and R. Myers, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, 2nd edition, 2002.

[20] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Petsc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 2.1.5, 2003.

[21] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*. SIAM Books, 2000.

[22] X. S. Li and J. W. Demmel, "SuperLU-DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Trans. Mathematical Software*, vol. 29, no. 2, pp. 110–140, June 2003.