An algorithm for distributed immersed boundary computations

E. Givelberg*

Department of Mathematics, Courant Institute, NYU, New York, USA

Abstract

The immersed boundary method is a general numerical method for modeling elastic boundaries immersed within a viscous, incompressible fluid. It has been applied to biological and engineering systems, including large-scale models of the heart and cochlea. Despite the popularity of the immersed boundary method and the desire to scale the problems to accurately capture the details of the physical systems, parallelization for large-scale distributed memory machine has proven challenging. The primary reason is a classic locality and load balance tradeoff that arises in distributing the immersed boundary data structure across processors. In this paper we describe a parallelized algorithm for the immersed boundary method that is designed for scalability on distributed memory computers. It is implemented using the Titanium language, a Java-based language designed for high-performance scientific computing. Our software package, called IB, takes advantage of the object-oriented features of Titanium to provide a framework for simulating immersed boundaries that separates the generic immersed boundary method code from the specific application features that define the immersed boundary structure and the forces that arise from those structures. We demonstrate the scalability of our design and the feasibility of large-scale immersed boundary computations with the IB package.

Keywords: Fluid–structure interactions; Immersed boundary method; Distributed algorithm; Large-scale computation

1. Introduction

The immersed boundary method is a general numerical method for computational modeling of systems with elastic (and possibly active) tissue immersed in a viscous, incompressible fluid. Such systems naturally arise in biology and engineering. The method was developed by Peskin and McQueen to study the patterns of the blood flow in the heart [1,2]. The research in immersed boundary computations and its many applications are reviewed in [3]. Immersed boundary models of the heart and the cochlea [4] motivate the work described in this paper. Realistic immersed boundary simulations of complex systems require very large computing resources. Both the cochlea and the heart models have been constructed on shared memory computers, where the parallelization of the serial immersed boundary code was achieved mainly with the help of compiler directives [5,6]. However, large-scale simulations involving hundreds and, perhaps, thousands of processors must be carried on machines with distributed architecture.

The immersed boundary method uses a Lagrangean formulation where the fluid is modeled by a three-dimensional rectangular grid and the immersed material is described by separate computational grids (one-dimensional fibers, two-dimensional plates and shells, etc.). Simulation proceeds in a series of time steps, where during each time step: (i) the elastic forces are computed on the material grids, (ii) spread to the fluid grid, (iii) the fluid equations are solved yielding a new fluid velocity, and (iv) the fluid velocity is interpolated to the material grids and is used to update their position relative to the fluid.

Distributed memory implementation of the immersed boundary method have proved quite challenging. The main challenge is to achieve a load-balanced computation, while keeping communication costs low. The interaction between the fluid and the immersed boundaries, however, may result in a significant amount of irregular communication. The algorithm must perform well for a wide range of possible systems with the immersed boundaries being sparse or dense, mobile or static. A distributed Navier–Stokes solver was constructed in [7] and attempts to design a distributed algorithm for the whole immersed boundary method were reported in [8]. Despite the great need, no scalable distributed memory implementation of the immersed

* Tel.: +1 212 998 3221; Fax: +1 212 995 4121; E-mail: givelber@cims.nyu.edu

Computational Fluid and Solid Mechanics 2005
K.J. Bathe (Editor)

boundary method has been available. Our distributed algorithm, descided below, was implemented in Titanium, an explicitly parallel dialect of Java developed at UC Berkeley to support high-performance scientific computing on large-scale multiprocessors. More details and additional results are described in [9].

2. The distributed algorithm

The solution of the discretized Navier–Stokes equations is the most expensive part of the numerical method. We have implemented a Fourier transform-based fluid solver using the distributed routines of FFTW [10], for which the fluid domain is naturally partitioned into slabs, one slab per processor.

We describe the immersed boundary by a collection of grids, each of which is wholly assigned to one of the available processors, making the force computation phase completely local. (More complicated schemes are possible and have been successfully tested.) The assignment of data structures to processors is constant throughout the computation; neither fluid points, nor material points migrate between processors.

In phase (ii) immersed boundary forces must be communicated to the (possibly very few) processors that own the appropriate portions of the fluid grid. To ensure a load-balanced computation we break phase (ii) into three stages and introduce a specialized cache-like data structure in each processor: First, each processor spreads the force of every point it owns into this cache. Next, it sends the contents of its cache to the processors that own the corresponding portions of the fluid force array. Finally, it uses the received information to update its portions of the fluid force array.

We now briefly describe the cache construction (see Fig. 1). We imagine the rectangular fluid grid as partitioned by a lattice into a collection of small $4 \times 4 \times 4$ cubes. The cache data structure in every processor consists of an $(N_1/4) \times (N_2/4) \times (N_3/4)$ array of pointers to cubes, together with a linked list containing all the indices of the cubes that have already been allocated. In the first stage of force spreading, this data structure is modified. It is easy to update, yet it does not contain an excessive amount of information; at the end of the local force-spreading stage it contains the combined force of the fluid due to the immersed boundary points owned by the processor.

3. Software performance

We have constructed a number of simple test models, each consisting of n identical $N \times N$-point plates ($N = 256$ or $N = 512$), evenly distributed among processors,

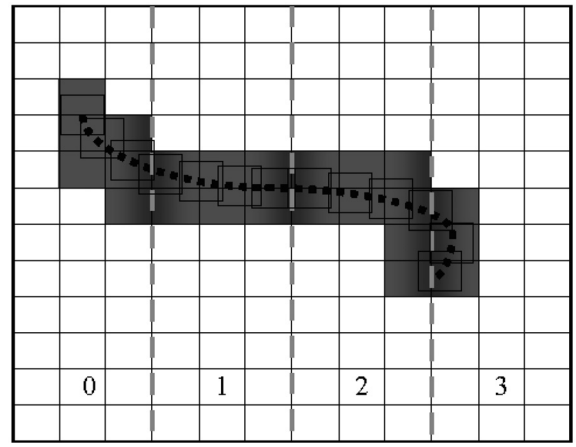

Fig. 1. The fluid cache: each point of the immersed boundary interacts with a $4 \times 4 \times 4$-cube of fluid around it; the shaded region describes the union of the fluid cache structures of all four processors. The cache of processor i is the intersection of slab i with the shaded region.

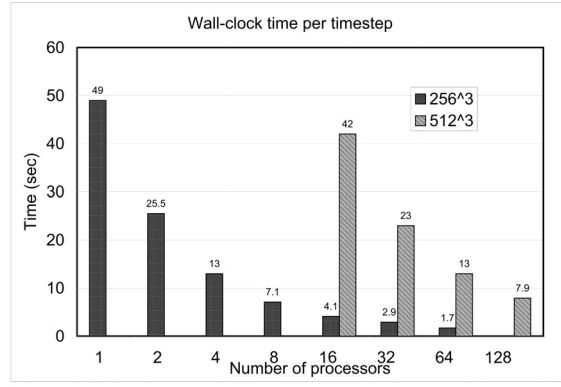

Fig. 2. Wall-clock time (in seconds) per time step for the (256, 1) and (512, 1) models as a function of the number of processors.

with $n = 1$, 16 and $N/8$, and $N = 256$ and $N = 512$. We refer to such models as (N, n)-models below. The sizes were chosen so that $n = 16$ models resemble the cochlea, while the $n = N/8$ models resemble the heart.

Our experiments were carried out on the IBM SP RS/6000 cluster of 16-processor nodes at the National Energy Research Scientific Computing Center (NERSC), Fig. 2 demonstrates the scaling of our software for two test problems. Table 1 summarizes the results for a number of test models, when the maximal number of processors is employed.

Table 1
Wall-clock time per time step for various test models

Model name	Fluid grid size	Total immersed boundary size	Number of processors	Total GFLOPs	Wall-clock time
(256, 1)	256^3	256^2 = 64K points	64	4.34	1.7 sec
(256, 16)	256^3	16×256^2 = 1M points	64	5.5	2.4 sec
(256, 32)	256^3	32×256^2 = 2M points	64	6.78	3.1 sec
(512, 1)	512^3	512^2 = 256K points	128	38.4	7.9 sec
(512, 16)	512^3	16×512^2 = 4M points	128	43.2	9.6 sec
(512, 64)	512^3	64×512^2 = 16M points	128	58.1	15.2 sec

4. Conclusions

We have developed an efficient algorithm for immersed boundary simulations on distributed systems. The classical tradeoff between locality and load balancing arising in fluid–structure interactions is overcome with the help of a specialized cache-like data structure, which is used to communicate information between processors.

We have implemented the algorithm using the Titanium programming language, utilizing its object-oriented features to provide a set of versatile classes for building large-scale immersed boundary applications. We have demonstrated the fesibility of large model constructions by measuring the performance on a number of test models. Subsequently, a 512^3 cochlea model based on the IB package has been successfully tested and Peskin and McQueen's 128^3 heart model has been reconstructed using the IB package. A 256^3 heart model is presently being built. These are the largest models that are possible to compute using the available hardware.

References

[1] Peskin CS. Flow patterns around heart valves: a digital computer method for solving the equations of motion. PhD thesis, Albert Einstein College of Medicine, 1972.
[2] McQueen DM, Peskin CS. Computer-assisted design of pivoting-disc prosthetic mitral valves. J Thorac Cardiovasc Surg 1983;86:126–135.
[3] Peskin CS. The immersed boundary method. Acta Numerica 2002;11:479–517.
[4] Givelberg E, Bunn J. A comprehensive three-dimensional model of the cochlea. J Comp Phys 2003;191(2):377–391.
[5] McQueen DM, Peskin CS. Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart. J Supercomputing 1997;11:213–236.
[6] Givelberg E, Bunn JJ, Rajan M. Detailed simulation of the cochlea: recent progress using large shared memory parallel computers. In: Proc of the 2001 International Mechanical Engineering Congress, New York, November 2001.
[7] Sabbagh HG. Solving the Navier–Stokes equations on a distributed parallel computer. PhD thesis, New York University, 1996.
[8] Yau SM. Experiences in using Titanium for simulation of immersed boundary biological systems, Master's Report, May 2002.
[9] Givelberg E and Yelick K. Distributed immersed boundary simulation in titanium. Submitted.
[10] Frigo M, Johnson SG. The design and implementation of FFTW3. In: Proc of the IEEE 2005;93(2):216–231. Special issue on Program Generation, Optimization, and Platform Adaptation.