

Petascale direct numerical simulation of blood flow on 200K cores and heterogeneous architectures

Abtin Rahimian*, Ilya Lashuk*, Shravan K. Veerapaneni[†], Aparna Chandramowliswaran*
Dhairya Malhotra*, Logan Moon*, Rahul Sampath[‡], Aashay Shringarpure*,
Jeffrey Vetter[‡], Richard Vuduc*, Denis Zorin[†], and George Biros*

* Georgia Institute of Technology, Atlanta, GA 30332

[†] New York University, New York, NY 10002

[‡] Oak Ridge National Laboratory, Oak Ridge, TN 37831

rahimian@gatech.edu, ilashuk@cc.gatech.edu, shravan@cims.nyu.edu,
aparna@cc.gatech.edu, dhairya.malhotra88@gmail.com, lmoon3@gatech.edu,
rahul.sampath@gmail.com, aashay.shringarpure@gmail.com,
vetter@computer.org, richie@cc.gatech.edu, dzorin@cs.nyu.edu, gbiros@acm.org

Abstract—We present a fast, petaflop-scalable algorithm for Stokesian particulate flows. Our goal is the direct simulation of blood, which we model as a mixture of a Stokesian fluid (plasma) and red blood cells (RBCs). Directly simulating blood is a challenging multiscale, multiphysics problem.

We report simulations with up to 260 million *deformable* RBCs. The largest simulation amounts to 90 billion unknowns in space. In terms of the number of cells, we improve the state-of-the art by several orders of magnitude: the previous largest simulation, at the same physical fidelity as ours, resolved the flow of $O(1,000-10,000)$ RBCs.

Our approach has three distinct characteristics: (1) we faithfully represent the physics of RBCs by using nonlinear solid mechanics to capture the deformations of each cell; (2) we accurately resolve the long-range, N-body, hydrodynamic interactions between RBCs (which are caused by the surrounding plasma); and (3) we allow for highly non-uniform spatial distributions of RBCs.

The new method has been implemented in the software library MOBO (for “Moving Boundaries”). We designed MOBO to support parallelism at all levels, including inter-node distributed memory parallelism, intra-node shared memory parallelism, data parallelism (vectorization), and fine-grained multithreading for GPUs. We have implemented and optimized the majority of the computation kernels on both Intel/AMD x86 and NVidia’s Tesla/Fermi platforms for single and double floating point precision.

Overall, the code has scaled on 256 CPU-GPUs on the Teragrid’s Lincoln cluster and on 200,000 AMD cores of the Oak Ridge National Laboratory’s Jaguar PF system. In our largest simulation, we have achieved 0.7 Petaflops/s of sustained performance on Jaguar.

I. INTRODUCTION

Clinical needs in thrombosis risk assessment, anti-coagulation therapy, and stroke research would significantly benefit from an improved understanding of the microcirculation of blood. Toward this end, we present a new computational infrastructure, MOBO, that enables the *direct numerical simulation of several microliters of blood* at new levels of physical fidelity (Figure 1). MOBO consists of two key algorithmic components: (1) scalable integral equation solvers for Stokesian flows with dynamic interfaces; and (2) scalable fast multipole algorithms. In terms of

size alone, MOBO’s overall simulation capability represents an advance that is orders of magnitude beyond what was achieved in prior work in blood flow simulation.

We use an algorithmically optimal semi-implicit scheme. Unlike explicit time-stepping schemes that require only near-neighbor communication (but are algorithmically suboptimal for our problem), our solver requires global communication at every time step. Consequently, our solver is much more challenging to scale than an explicit solver. Nevertheless, our results demonstrate that it is possible to successfully scale implicit solvers to hundreds of thousands of cores.

Challenges in direct numerical simulation of blood. Just one microliter of blood of a healthy individual contains approximately four million RBCs. The surrounding plasma, which is a viscous fluid, mechanically couples every RBC to all other RBCs. Furthermore, RBCs are highly deformable (it is the deformability of RBCs which determines the rheological properties of blood). The large number of cells and their complex local and global interactions pose significant challenges in designing tools for high-fidelity scalable numerical simulations of blood.

Due to these difficulties, multiphase blood flow simulations have been restricted to relatively small number of RBCs. For example, the largest simulations today have scaled to 1,200 cells [40] (using boundary integral equations, like us) and 14,000 cells [8] (using lattice Boltzmann methods). The latter work, which is based on an explicit time-stepping scheme, scaled up to 64K cores but requires an excessive number of time steps due to the non-physical stiffness introduced by the numerical scheme. Other methods that model RBCs as rigid bodies have scaled to an even large number of RBCs; but these are crude approximations of the blood flow. Deformable models of RBCs (Figure 1) are critical for accurate blood flow simulations.¹

¹For example, a 65% volume-fraction suspension of rigid spheres cannot flow; blood flows even when the volume-fraction of RBCs in the plasma reaches 95% [2].

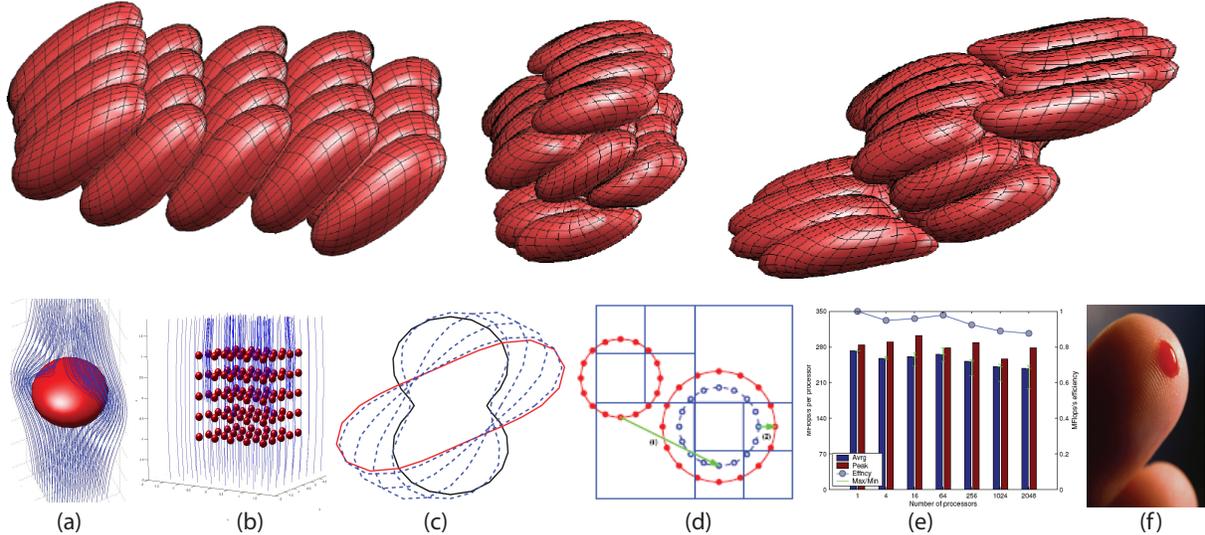


Fig. 1: SUMMARY OF THE COMPUTATIONAL INFRASTRUCTURE FOR DIRECT NUMERICAL SIMULATION OF BLOOD FLOW. *In the top row, we depict a few snapshots from the flow of twenty RBCs that are immersed in plasma. At every time step, a Stokes problem must be solved in the exterior and interior of the RBCs. We have developed computational tools for this problem. The main algorithmic components include: (a) spectral RBC shape representations and quadratures for singular integrals on these shapes; (b) accurate modeling of the hydrodynamic interactions between many-RBCs; (c) nonlinear solvers for the mechanics of RBC deformations; and (d,e) parallel, kernel-independent, tree-based, fast summation methods. The advantage of boundary integral methods is that only the RBC boundary is discretized and no discretization of the space between RBCs is necessary. This is crucial for reducing the number of degrees of freedom and eliminates the need for difficult-to-parallelize 3D unstructured mesh generation. Our tools enable parallel and highly accurate simulations of microcirculation phenomena of blood flow. We have achieved the direct numerical simulation of $\mathcal{O}(50)$ microliters of blood flow. (f) One can think of the volume of a single blood drop as being roughly equivalent to one microliter.)*

Our approach. We model RBCs as deformable viscous sacs with an inextensible, massless membrane that can sustain bending and tension forces. The surrounding plasma is modeled as a Stokesian fluid (we neglect inertial terms). There are several challenges in simulating such a system:

- The evolution of the RBCs requires solving the Stokes equations in the plasma—a very complex geometric region that changes at every time step.
- Computing the bending and tension forces requires accurate geometric description of the shape of the RBCs. Furthermore, these forces introduce significant numerical stiffness.

To address these challenges, we use

- an integro-differential formulation in which we couple a boundary integral formulation for the Stokes equations (plasma) with the RBC’s membrane elasticity;
- a semi-implicit time-stepping scheme that removes the stiffness due to interfacial forces;
- spherical harmonics representations for the shape and the deformation of RBCs;
- the fast multipole method to accelerate the long-range hydrodynamic interactions between cells and plasma; and
- distributed and shared memory parallelism, SIMD parallelism (vectorization), and fine-grained multithreading via GPGPU acceleration, to expose maximum concurrency.

MOBO employs Fourier and Legendre transforms, adaptive fast multipole methods, Galerkin projections, multi-step time march-

ing, fast spherical harmonics rotations, spectral quadratures for smooth and weakly singular integrals, preconditioned Krylov linear solvers, and dense linear algebra.

Our overall formulation can be outlined as follows. We use a spherical harmonics representation for the boundary of every RBC. This choice is mathematically equivalent to tracking a number of points on the surface of the RBC. In our simulations, we typically track either 84 or 312 points. The motion of each such point \mathbf{x} is governed by

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{v}(\mathbf{x}), \quad (1)$$

$$\mathbf{v}(\mathbf{x}) = \mathbf{v}_{local}(\mathbf{x}) + \mathbf{v}_{global}(\mathbf{x}) + \mathbf{v}_{background}(\mathbf{x}).$$

Here, \mathbf{v} is the velocity of the point, which we decompose into three components: local, global, and background velocities. Roughly speaking, the “local” velocity, \mathbf{v}_{local} , accounts for the interactions between the specific point in the RBC under consideration and all of the other points within the same RBC. The “global” velocity, \mathbf{v}_{global} , accounts for all of the interactions occurring across all of the RBCs in the simulation. The “background” velocity, $\mathbf{v}_{background}$, is the imposed flow field. This work builds on our previous work on massively parallel tree-data structures [30], [27], parallel and kernel independent fast multipole methods [36], [17], [7], and fast solvers for particulate flows [33], [25], [34].

Contributions. In [25], we presented the details of the formulation and the numerical algorithms that are required to compute \mathbf{v}_{local} and \mathbf{v}_{global} . Here, we focus on the parallelization

and performance analysis for the computation of \mathbf{v}_{local} and \mathbf{v}_{global} . \mathbf{v}_{local} requires nine different kernels. \mathbf{v}_{global} uses the FMM, which in turn has five major computational phases (tree construction, three tree-traversals, and the direct interactions). Our key contributions are:

- We present a hybrid-parallel implementation of nine computational kernels that MOBO uses for the computation of \mathbf{v}_{local} and \mathbf{v}_{global} . The kernels are multithreaded and work-partitioned between CPU and GPU, which execute concurrently, thereby delivering excellent per-node performance.
- The most intensive kernels in our computation have been designed for locality, accuracy, and computational efficiency, capitalizing in particular on highly optimized BLAS3 (GEMM) operations.
- We further improve the performance of our SC'09 FMM algorithm [17]. These improvements include explicit SSE vectorization and multithreading via OpenMP, as described in prior work [7]. In this paper, we add simultaneous asynchronous GPU acceleration.
- We present single-node analysis for computations of \mathbf{v}_{local} and \mathbf{v}_{global} on AMD, Intel, and NVIDIA platforms.
- We present weak and strong scaling results on the Jaguar PF system at Oak Ridge National Laboratory (ORNL).

Performance Highlights. We achieve 780 TFlop/s of sustained performance on the 196,608 cores of the AMD Istanbul-based Jaguar PF system (4 GFlop/s per core), with $160\times$ speedup on strong scaling when moving from 48 to 24,576 cores ($512\times$); and 75% efficiency for the weak scaling. On other platforms, we demonstrate up to 18 GFlop/s per core of sustained performance on the Intel Nehalem-EP; and up to 350 GFlop/s per NVIDIA Fermi C2050 card (both in single precision).

In our largest simulation, we solved a problem involving 8,000 RBCs per MPI process, on 32,768 MPI processes for a total of 196,608 cores. We discretized using 84 points per RBC. This set of parameters results in a total of 262,144,000 RBCs (50 drops of blood) and 90 billion unknowns per time step. (We have four unknowns per point: the three coordinates and a scalar tension.)

Limitations. Despite its capabilities, our method has several limitations. First, MOBO is restricted to very low Reynolds numbers and, therefore, cannot accurately be used to simulate high-Reynolds blood flow (e.g., flow in large arteries). Second, the discretization of the RBCs is not adaptive: all of the RBCs are approximated using the same number of points. Third, the current version of MOBO does not support confined boundaries. The modifications of the method for the confined boundary case has been presented in [25] but we have not yet parallelized the method. Fourth, the memory requirements of the method grow with the cube of the number of points per RBC.

Related work. In spite of advances in understanding the complex behavior of particulate flows [24], only recently have algorithmic advances allowed accurate 3D simulations of Stokes

flows with hundreds of deformable particles using boundary integrals. Attempts to parallelize integral equation solvers have been restricted to low-accuracy discretizations, spatially uniform particle distributions, and have not scaled to large numbers of cores. The main challenges are the parallelization of the hydrodynamic interactions, the stiffness of the RBC deformations, and end-to-end scalability and performance for all of the algorithmic components of a method.

Impressive simulations based on *fictitious/immersed boundary-like methods* have been reported in [1], [2], [11], [14], [18], [32]. However, there is limited work in efficiently parallelizing these methods and no scalings on thousand-core machines have been achieved [13], [19], [28], [37]. In blood rheology simulations, there are at least two examples of simulations with large numbers of particles: one which models a $50\mu\text{m}^2 \times 500\mu\text{m}$ -capillary blood flow with 300 thousand rigid particles [22], and another in which a *dissipative particle dynamics* method was used to model a few thousand deformable RBCs [10]. *Lattice-Boltzmann* based methods for particle simulations have been used for blood flow but are limited to rigid RBCs [29]. An exception is the lattice-Boltzmann approach in [8], that allows for deformability of RBCs. However, lattice Boltzmann methods are low-order accurate in space and require small time steps due to numerical stiffness [20]. Lastly, another class of methods are based on *moving-mesh finite-element methods* [31]. Such methods are difficult to parallelize in particular, for the case of large 3D deformations due to the need for unstructured mesh generation [5].

A different class of methods based on *boundary integral equation* formulations is ideal for blood flow since it only requires discretization of the RBC membrane, which is more scalable than discretizing the volume occupied by the plasma [24], [4], [16], [24], [23], [39], [38]. The aforementioned successful simulation of 1200 deformable drops [40] used a boundary integral formulation. But, that implementation was sequential and required 120 CPU hours to complete. Overall, limited work exists in parallelizing such methods. One exception is the parallel Stokes solver of Thien, et al. [21]; their calculations, however, were performed using a suboptimal $\mathcal{O}(N^2)$ algorithm and not a fast multipole method.

II. FORMULATION AND ALGORITHMS FOR PARTICULATE FLOWS

Notation. Before we proceed with describing the kernels of MOBO, let us introduce some notation. (We use MATLAB's notation for linear algebra operations.) We use upper-case letters for matrices and lower-case letters for vectors. We use upper-case bold-face letters for discretizations integral and differential operators, and lower-case bold-face letters to denote vectors and points in \mathbb{R}^3 .

Mathematical formulation. The equations governing the dynamics of particulate flows in the free space \mathbb{R}^3 are the Stokes equation for the plasma and a differential equation for

Symbol	Definition
m	Number of points used to discretize the RBC surface
q	Spherical harmonics expansion order
n	Total number of Red Blood Cells
p	Number of processors
\mathbf{v}	Velocity
\mathbf{F}	Discrete Fourier Transform operator
\mathbf{P}_k	k^{th} -order Discrete Legendre transform
$\mathbf{S}, \mathbf{S}^{-1}$	Forward/inverse spherical harmonics transform

TABLE I: Index of frequently used symbols and operators.

the forces on the membrane of the RBCs. All of the RBCs are assumed to be filled with a Stokesian fluid that is the same as the surrounding fluid (henceforth “fluid”).²

Specifically, we have

$$-\Delta \mathbf{v} + \nabla p = 0 \quad \text{in } \mathbb{R}^3 \setminus \cup \gamma_k, \quad (2)$$

where \mathbf{v} is the velocity of the fluid, p is the pressure, and γ_k is the interface between the k^{th} RBC and the surrounding fluid. This equation is subject to the constraints

$$[[\mathbf{Tn}]] = \mathbf{f} \quad \text{on } \cup_k \gamma_k, \quad \text{div } \mathbf{v} = 0, \quad [[\mathbf{v}]] = 0 \quad \text{in } \mathbb{R}^3, \quad (3)$$

where $[[\cdot]]$ denotes a “jump”, a difference between the inside and outside of the RBC membrane, \mathbf{Tn} is the fluid force on the membrane and \mathbf{f} is the force related to the elasticity of the cell membrane (i.e., the cell membrane’s resistance to bending and stretching). The first equation states the balance of forces: the bending and tension forces at the membrane must equal the viscous forces exerted by the fluid. The second equation states the incompressibility of the fluid. The third equation merely implies that a point on the RBC’s membrane must move with the same velocity as the surrounding fluid (non-slip condition). Using the last equation, the evolution of a point on the membrane of a RBC reads as

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}), \quad \text{for } \mathbf{x} \in \gamma_k, \quad \text{and all } k, \quad (4)$$

where \mathbf{x} denotes a point on γ_k . As mentioned in the introduction, the velocity $\mathbf{v}(\mathbf{x})$ can be decomposed into three components:

$$\mathbf{v}(\mathbf{x}) = \mathbf{v}_{local}(\mathbf{x}) + \mathbf{v}_{global}(\mathbf{x}) + \mathbf{v}_{background}(\mathbf{x}) \quad (5)$$

The first term, \mathbf{v}_{local} , at a point \mathbf{x} on γ_k (the membrane of the k^{th} RBC) depends only on the shape of γ_k . The second term depends on the shapes of all of the RBCs in the simulation and requires an N-body calculation. The third term, $\mathbf{v}_{background}$ is known analytically and depends on the numerical experiment (e.g., imposed shear flow). The precise expressions for \mathbf{v}_{global} are quite lengthy and due to space limitations, are omitted. See

²The more general (and accurate) case in which the inside and outside fluid are different does not introduce any complications either mathematically or algorithmically. See [25].

[34] for further details. The global velocity is easy to state:

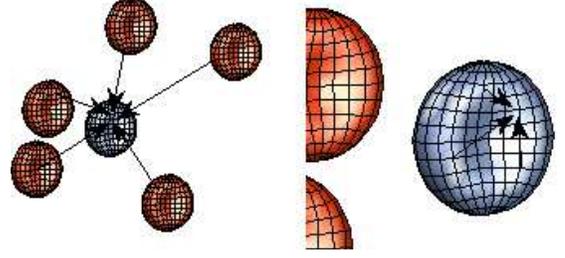


Fig. 2: AN EXAMPLE OF GLOBAL AND LOCAL INTERACTIONS. The interaction (\mathbf{v}_{global}) between points on the surface of the blue membrane and points on the surfaces of the other red membranes is **global** (left figure). The interaction (\mathbf{v}_{local}) between points on the surface of the same (blue) membrane is an example of **local** interactions (right figure).

$$\mathbf{v}_{global}(\mathbf{x}_k) = \sum_i \frac{1}{\rho_{ki}} \left(\mathbf{d}_i + \frac{(\mathbf{r}_{ki} \cdot \mathbf{d}_i) \mathbf{r}_{ki}}{\rho_{ki}^2} \right), \quad (6)$$

where $\mathbf{r}_{ki} = \mathbf{x}_k - \mathbf{x}_i$, $\rho_{ki} = |\mathbf{r}_{ki}|$, \mathbf{d}_i is the given density at point \mathbf{x}_i , $\mathbf{r}_{ki} \cdot \mathbf{d}_i$ denotes the geometric dot product between these two vectors, and $|\cdot|$ denotes the vector norm in \mathbb{R}^3 .

Representation of the surface. Let $U = \{(\theta, \phi) : \theta \in (0, \pi), \phi \in (0, 2\pi)\}$. Then, we denote a parametrization of the surface by $\mathbf{x} : U \rightarrow \mathbb{R}^3$. θ parametrizes the latitude and ϕ parametrizes the longitude. In this way, \mathbf{x} can be represented in the spherical harmonics basis with spectral accuracy [34]. This spectral representation enables fast and accurate computation of high-order derivatives of \mathbf{x} with respect to θ and ϕ . Such derivatives are required for the calculation of \mathbf{v}_{local} . (For example, \mathbf{v}_{local} depends on the surface Laplacian of the mean curvature of the membrane, the normal vector, and other geometric quantities.)

A. Overall algorithm and the main kernels

Due to space limitations, we describe a first-order explicit Euler time-marching scheme. (That is explicit on the RBC membrane mechanics. The Stokes equations for the plasma and the interior of the cells are solved *always implicitly*.) The actual time-stepping scheme used in these simulations is a multistep, semi-implicit scheme and is described in detail in [34].³

Given n RBCs, each one being represented by its surface γ_k a set of points on this surface \mathbf{x} , the algorithm proceeds as follows:

- 1) Compute $\mathbf{v}_{local}(\mathbf{x}, \gamma_k)$, for all $\mathbf{x} \in \gamma_k$, $k = 1, \dots, n$.
- 2) Compute \mathbf{v}_{global} using FMM;
- 3) Evaluate $\mathbf{v}_{background}$ analytically;
- 4) Update the position using $\mathbf{x}_{new} = \mathbf{x} + \Delta t(\mathbf{v}_{local}(\mathbf{x}) + \mathbf{v}_{global}(\mathbf{x}) + \mathbf{v}_{background}(\mathbf{x}))$,

where Δt is the time-step size.

³The linear solves in the semi-implicit are done with a Krylov method, which requires a matrix-vector operation. The latter has been implemented using the computational kernels described in this section.

For the semi-implicit scheme described in [34], the first step is embarrassingly parallel across k (RBCs). For each cell, the complexity of computing \mathbf{v}_{local} for all \mathbf{x} on its surface is (q^6) . The second step involves a communication-heavy, all-to-all, N-body calculation with the Stokes kernel. The complexity of the Stokes kernel was analyzed in [17]. We briefly summarize the main components of FMM in the next subsection.

B. Global interactions: FMM kernel

The FMM consists of two main steps, the octree construction phase and the evaluation phase. The evaluation phase consists of the computation of the far-field approximated interactions and the exact near-field direct interaction. Specifically, the evaluation phase involves the following substeps:

- 1) a bottom-up (post-order) tree-traversal to compute the multipole-moments approximation;
- 2) an arbitrary-order traversal to translate multipole-moments to local approximations;
- 3) a top-down (pre-order) tree traversal to accumulate all far-field interactions to the target points, the so-called “VXW-list”-calculation in FMM jargon; and
- 4) a near-neighbor exchange at the leafs to compute the near-range interactions, the so-called “U-list”-calculation in FMM jargon.

In parallelizing FMM, the main challenges are the tree construction, and the communication involved in step (2), which in principle is local, but involves “fat” neighbors that can cause communication imbalance for highly nonuniform trees. One can prove, under assumptions on the distribution of RBCs, that the complexity of the tree-construction phase is $\mathcal{O}(\frac{n}{p} \log \frac{n}{p}) + \mathcal{O}(p \log^2 p)$ and of the evaluation phase is $\mathcal{O}(\frac{n}{p}) + \mathcal{O}(\sqrt{p})$, where p is the number of MPI processes.

In [17], we proposed a novel tree-construction algorithm, a novel hypercube-based reduction for the V-list calculation, and a hybrid GPU-MPI implementation for the U-list and V-list calculations. However, that approach left the multi-core CPUs that were driving the GPU spinning idle. In , we introduced a set of optimizations that can further accelerate the CPU computations (OpenMP acceleration and SSE vectorization).

In section III, we report results from a different multithreading strategy. We employ a hybrid OpenMP-MPI-CUDA scheme in which we compute the dense interactions in parallel with the far-field interactions. The CPU sockets are responsible for the far-field computations (V-list) and the GPUs are responsible for the direct interactions (U-list). In addition, we have introduced several optimizations that are specific to our Kernel Independent FMM [36].

We use Streaming SIMD Extensions (SSE) technology, available in a number of modern CPUs, to speed up the floating point computations. In a nutshell, using SSE allows to perform basic arithmetic operations on small vectors of floating point numbers. Specifically, vectors can consist of either four single-precision floating point numbers or two double-precision floating point numbers (in any case vectors are 128 bit long). Arithmetic

operations on different vector entries are performed by the CPU in parallel, thus speeding up the computation (in an ideal scenario) by a factor of four or two, depending on the precision. We use SSE to accelerate the particle-to-particle interactions (specifically, the evaluation of the Stokes kernel). We replicate the data associated with each target point (e.g., x -coordinate) into four entries of an SSE vector, and load another SSE vector with x -coordinates of four different source points. Then we apply SSE vector subtraction to evaluate 4 differences in parallel, then we square these four differences in parallel and so forth. Eventually we obtain four potentials in an SSE register and sum them up. We use this approach for the source-to upward equivalent densities. This work is explained in further detail in [7].

Finally, we use point-to-point interactions without precomputation in many parts of the algorithm for improve float-to-memory access rations and improved overall performance.

a) *Partitioning of Red Blood Cells*: Partitioning RBCs among MPI processes is done using the FMM underlying data structure. This is necessary because after a few time steps the partitioning of RBCs does not match the optimal partitioning for the FMM and this results to excessive communication. The parallel FMM code that we are using [35], requires particles to be Morton-sorted not only locally on each MPI task but also across MPI tasks. That is, in order to apply FMM for computations, we first have to redistribute (“scatter”) the points between tasks so that points become Morton-sorted, then evaluate the potentials and finally scatter the potentials back to the original layout of the points. When using multiple (say, tens of thousands) MPI tasks, the cost of these two scatters can become prohibitive, unless special measures are taken. We periodically re-distribute the RBCs between MPI tasks, so that the overall distribution of points is “close” to being Morton sorted, and thus, the scatters are relatively inexpensive. Specifically, we use the partitioning of the space between MPI tasks produced by the last call to FMM. For each RBC, we determine preliminary “target MPI task” for each point of the RBC. Then we decide the final MPI task for the RBC by the voting procedure. For the actual data transfer we employ an `MPI_Alltoallv()` call.

C. Local interactions: RBC physics kernels

The computation of \mathbf{v}_{local} consists of several kernels. In the following, we discuss nine kernels in which the majority of the computation takes place:

b) *The spherical harmonics transform kernel*: The spherical harmonics transform may be expressed in terms of matrix operations [6]. For a spherical harmonic expansion of order q , there are $2q$ points in the east-west direction and $q + 1$ points on the north-south direction. Let X , Y , and Z denote matrices, each of size $2q \times (q + 1)$, that hold the x -, y -, and z -coordinate components of the grid points, respectively. The points are stored in a “latitude-major order”. Then, the k^{th} order spherical harmonic coefficients of X is given by

$$\hat{X}_k = \mathbf{P}_k \mathbf{W}(\mathbf{F}X)_k^T, \quad k = 0, \dots, 2q, \quad (7)$$

where $\mathbf{F} \in \mathbb{R}^{2q \times 2q}$ denotes the discrete Fourier transform, $\mathbf{W} \in \mathbb{R}^{(q+1) \times (q+1)}$ is a diagonal matrix holding the Gaussian quadrature weights, $\mathbf{P}_k \in \mathbb{R}^{k \times (q+1)}$ is the k^{th} order associated Legendre transform and $\hat{X}_k \in \mathbb{R}^{k \times 1}$ is the vector of spherical harmonic coefficients of the k^{th} order. The same formula is also true for Y and Z . Considering the fact that q is rather small compared to the number of RBCs, it is best to perform both Fourier and Legendre transforms as matrix multiplications [9]. This also motivates the data structure for our implementation.

The inverse of spherical harmonics transform is given by

$$X = \mathbf{F}^T [\mathbf{P}_1^T \hat{X}_1 \dots \mathbf{P}_{2q}^T \hat{X}_{2q}]^T. \quad (8)$$

Hereinafter, we formally denote the forward and inverse spherical transforms by \mathbf{S} and \mathbf{S}^{-1} . When we have n surfaces, the complexity of the forward and inverse spherical harmonics transform is $\mathcal{O}(nq^3)$ and the depth is $\log q$.

In order to accelerate spherical harmonics transform, we represent the transform as a sequence of multiplications of real matrices, that is we use real DFT and not the complex one; we use BLAS or CUDA-BLAS for all the matrix multiplications; and we use column-wise (Fortran-style) storage for all the matrices.

The input data corresponding to different RBCs is packed into a $2q \times (q+1)n$ matrix. Each row of this matrix corresponds to a particular latitude across all RBC. Each column corresponds to a particular longitude on a particular RBC. Columns corresponding to the same RBC are grouped together (columns related to the first RBC are followed by the columns related to the second RBC, etc.)

We start the transform by multiplying the input matrix from the left by the DFT matrix, thus applying DFT to each column independently. Then we transpose the resulting matrix. Note that now each column of the transposed matrix corresponds to a particular frequency and one of the two possible functions (sine or cosine). We then treat each column of the transposed matrix as a $(q+1) \times n$ matrix (stored column-wise). Rows of this matrix correspond to different longitudes across all RBCs, and columns correspond to different RBCs. We then multiply this matrix from the left by an appropriate Legendre transform matrix, as given in Equation (7).

Note that in case of GPU computations (CUDA-BLAS), we perform both matrix multiplications and the transpose on GPU.

All the data is stored in a one dimensional array and depending on the size parameter passed to the BLAS kernels, its content can be interpreted as matrices of different sizes. Using MATLAB's notation, we store the coordinates array for the k^{th} RBC as $C_k = [X(\cdot)^T \ Y(\cdot)^T \ Z(\cdot)^T]$, and for all RBCs as $C = [C_1 \dots C_n]$. Using this structure, data can be streamed to BLAS subroutines for the calculation of spherical harmonics transforms.

c) Pole rotation kernel for weakly-singular quadratures:

Given the surface C_k ($k = 1, \dots, n$) and a target point (x, y, z) on the same surface, there exists a linear transformation $\mathbf{R} \in \mathbb{R}^{m \times m}$ such that the pole for the surface $\bar{C} = \mathbf{R}C$ is located

at (x, y, z) . Note that the transformation \mathbf{R} depends on the parametrization of the surface and the target point, but it is independent of the geometry of the surface. An example of this transformation is given in Figure 3. Let $\mathbf{R}_1, \dots, \mathbf{R}_{q+1}$ be the transformations with the target point as the grid points on the $\phi = 0$ meridian, then the transformation for other points on the θ_k latitude is a permutation of \mathbf{R}_k [12].

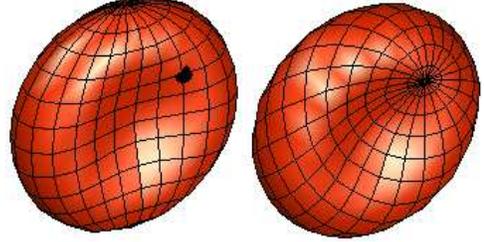


Fig. 3: THE TYPICAL BICONCAVE SHAPE. The plot on the right is the same surface as in the one in the left, but the pole is moved to the point marked by the circle in the left figure.

In our simulation, we need to perform this rotation for all of the points on the surface of a RBC. For our method, the complexity of a single rotation of the pole is $\mathcal{O}(q^4)$ and the memory requirement is $\mathcal{O}(q^5)$. There is another algorithm for the rotation of the pole that is based on the spherical harmonics expansion of the surface [12], and [34]. This algorithm reduces the complexity to $\mathcal{O}(q^5)$ and the memory requirements to $\mathcal{O}(q^4)$. That algorithm has more expensive logic and it is preferable only for small number of RBCs. Table II summarizes the result of our comparison between the two algorithms.

n	Direct (cublas)		Rotation via spherical harmonics	
	$q = 6$	12	6	12
8	21.25	169.17	0.84	29.15
64	23.48	258.38	6.15	225.85
512	57.75	1360.07	47.85	1795.29
1024	98.02	2597.93	96.20	3589.00

TABLE II: The execution time (ms) on a Tesla GPU to move the pole to all points on the surface for different algorithms.

d) The kernel for the weakly-singular integrals: The computation of $\mathbf{v}_{local}(\mathbf{x})$, $\mathbf{x} \in \gamma_k$ can be written as

$$\mathbf{v}_{local}(\mathbf{x}) = \int_{\gamma_k} K(\mathbf{x}, \mathbf{y}) \mathbf{f}(\mathbf{y}) d\mathbf{y},$$

where K is the Green's function for the Stokes equations. This integral has a weak-singularity for $\mathbf{y} = \mathbf{x}$ for all \mathbf{x} in γ_k except the two poles. One method to evaluate the Stokes' integral at a point on the surface is to move the pole to that target point and evaluate the integral for that particular point [34]. In Algorithm 1, we outline the evaluation of Stokes integral. For n surfaces, the work is $\mathcal{O}(nq^6)$ and the depth is $\mathcal{O}(\log q)$.

After the multiplications by the rotation matrices \mathbf{R} , the most costly kernel in our simulation is the Stokes evaluation

Algorithm 1 Evaluation of singular Stokes integral. $\mathcal{O}(q^6)$
 D is the input density and U is the evaluated potential.

```

for  $k = 0$  to  $q + 1$  do
  for  $l = 0$  to  $2q$  do
     $\mathbf{R} \leftarrow$  Permute  $\mathbf{R}_k$  for the target  $\phi_l$ 
     $\tilde{C} = \mathbf{R}C, \tilde{D} = \mathbf{R}D,$ 
    Evaluate  $U_{kl}$  (by direct Stokes kernel)
  end for
end for

```

kernel. For the CPU code, this kernel is accelerated using SSE instructions.

e) *The kernel for surface differentiation:* The differentiation with respect to the ϕ parameter is a straightforward calculation using the DFT. But, differentiation with respect to θ needs extra care. With an abuse of notation, let $\mathbf{H}_k = d\mathbf{P}_k/d\theta$ and $\mathbf{W}_k = d^2\mathbf{P}_k/d\theta^2$. Then, to differentiate a function $G \in \mathbb{R}^{2q \times (q+1)}$ on the surface with respect to the parameter θ we have

- (i) $\hat{G} = \mathbf{S}G,$
- (ii) $dG/d\theta = \mathbf{F}^T[\mathbf{H}_1^T \hat{G}_1 \dots \mathbf{H}_{2q}^T \hat{G}_{2q}]^T,$
- (iii) $d^2G/d\theta^2 = \mathbf{F}^T[\mathbf{W}_1^T \hat{G}_1 \dots \mathbf{W}_{2q}^T \hat{G}_{2q}]^T.$

The complexity of these steps is the same as spherical harmonics transform and is $\mathcal{O}(q^3)$. With different matrices, the same kernel is used to evaluate the inverse spherical harmonics and all the derivatives.

f) *Other computation kernels:* We have implemented several other kernels that are required for computations on the surfaces, for instance, the computation of geometric properties of the RBC. These kernels include the geometric cross ($\mathbf{a} \times \mathbf{b}$), and dot ($\mathbf{a} \cdot \mathbf{b}$, $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$) products, matrix transpose, and scaling of vectors. They have been implemented on top of our specific data format for multiple RBCs and have been optimized on both CPUs and GPUs.

g) *The kernel for the FMM correction:* The FMM algorithm indiscriminately calculates all the pairwise interactions between the source and target points. When we use FMM to compute \mathbf{v}_{global} , we also compute the interaction between the points that belong to the same RBC. But, these interactions need to be evaluated as *local* interactions. Therefore, we need calculate and subtract these erroneous terms from \mathbf{v}_{global} . The direct Stokes kernel, is used to evaluate the correction.

h) *The kernel for surface reparametrization:* In a typical simulation, the RBCs go through great distortion and the quality of the grid on their surface diminishes very fast. In Figure 4, we give an example of such distortion. In [34], we proposed a *tangential correction* algorithm to compensate for the distortions and maintain the grid quality. The reparametrization of the surface, involves calculation of the normal vector to the surface, mapping the surface to the spherical harmonics domain, filtering the high frequencies, and restricting the correction to the tangential direction on the surface (to keep the shape of the RBC intact). The complexity of this kernel is $\mathcal{O}(nq^3)$.

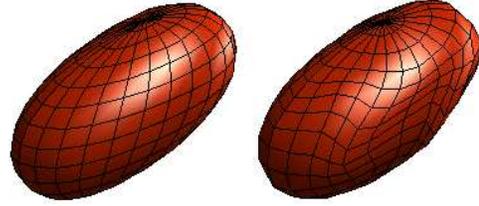


Fig. 4: REPARAMETERIZATION OF RBC. *The comparison between the quality of the grid for two simulation with and without the reparametrization, i.e, the redistribution of points on surface of an RBC in order to improve the numerical stability of the time-stepping scheme. Without such reparametrization, the distribution of points on the surface becomes distorted as we march in time and the accuracy of the simulation is quickly lost.*

III. SCALABILITY RESULTS

In this section, we describe the results from numerical experiments we conducted to investigate the performance characteristics of MOBO and the parallel scalability of our implementation across different architectures. Below, we summarize the different aspects of our numerical tests.

Platforms and architectures: The large scale weak and strong scalability results have been obtained on the *Jaguar PF* platform at the National Center for Computational Sciences (UT/ORNL). Jaguar is a Cray XT5 system with 224,256 cores (2.6 GHz hex-core AMD Opteron, 2GB/core) and a 3D-torus topology. Jaguar is ranked first in the top-500 list of supercomputers (www.top500.org) as of July of 2010. The GPU scalability results have been obtained on TeraGrid's *Lincoln* at the National Center for Supercomputing Applications (UIUC/NSF), a Dell cluster with NVIDIA Tesla S1070 accelerators, 1536 cores (Intel Harpertown/2.33 Ghz dual-socket quad-core 2GB/core), 384 GPUs (4GB/GPU), and InfiniBand (SDR) interconnect. The results on Fermi were obtained on a single node AMD machine at ORNL. The Nehalem tests were performed in an in-house 8-node cluster, with 16 sockets and one NVIDIA T10P-based GPU per socket. In all of the experiments on Jaguar, we use one MPI process per socket and six threads per socket. Both \mathbf{v}_{local} and \mathbf{v}_{global} calculations have been multithreaded using OpenMP. Also, in all of our GPU experiments, we use one MPI process per socket.

Implementations and libraries: The code is written in C++ and the accelerator modules in CUDA. We use the PETSC [3] for profiling and certain parts of communication, and the DENDRO [26] package for the tree construction and repartitioning of the RBCs. The \mathbf{v}_{global} module was implemented using our Kernel Independent Fast Multipole Method [17]. All of the kernels required for the calculation of \mathbf{v}_{local} were implemented from scratch. We used the native CRAY libsci and MKL BLAS libraries on the Jaguar and the Intel boxes respectively.

Single node Experiments: To assess the performance of our code on a single node, we performed various tests for the \mathbf{v}_{local} and \mathbf{v}_{global} calculations. The results are reported in Figures 5 for the \mathbf{v}_{local} evaluation, and 6 for the \mathbf{v}_{global} evaluation. Overall, we observe little difference between CPUs and GPUs

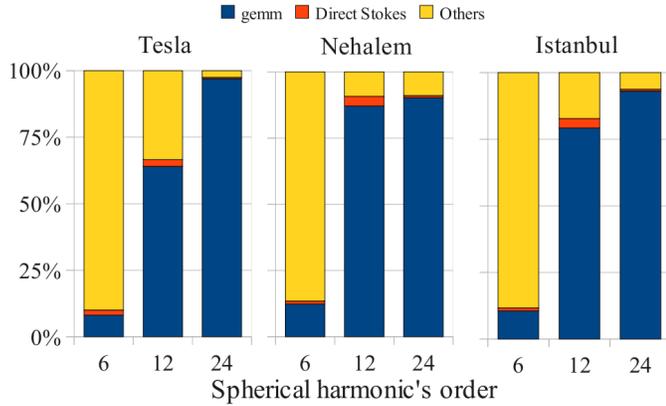
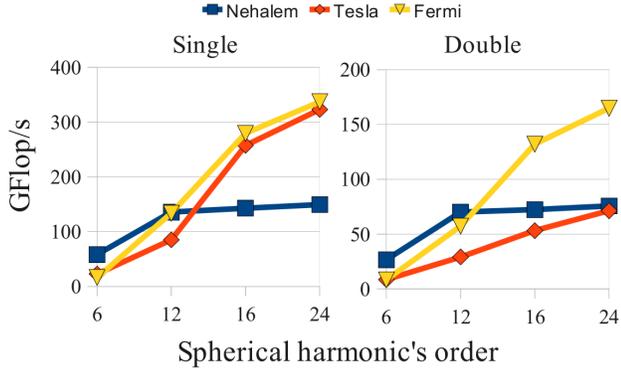


Fig. 5: SINGLE NODE FOR LOCAL INTERACTIONS. The first figure shows the sustained single and double precision FLOPS per second of the local kernel, on a Nehalem with eight OpenMP threads, a Tesla, and a Fermi. The second figure is the work share of the major components of the local kernel. For this figure, we used 8 OpenMP threads on Nehalem and 12 OpenMP threads on Istanbul.

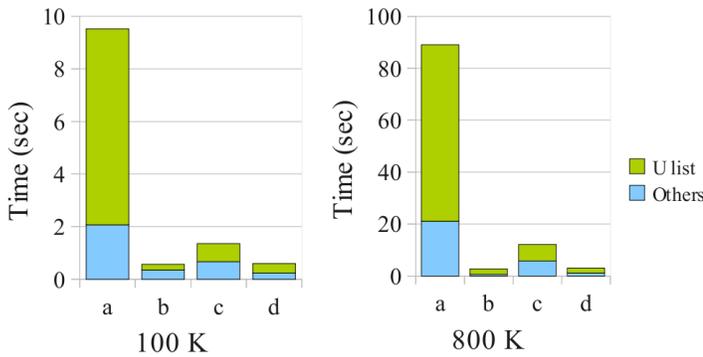


Fig. 6: SINGLE NODE RESULTS FOR THE GLOBAL INTERACTIONS. In this figure we present results for the FMM code for various hybrid architecture setups: (a) One thread CPU only without SSE, (b) GPU only for both direct and near evaluations, (c) Four threads on CPU with SSE, (d) Four threads on the CPU with SSE for the V-list and asynchronous evaluation of U-list on the GPU. On both Istanbul and Nehalem architectures we observe 1.2 ($m=6$)–1.7 ($m=12$) GFlops/s per core, for the overall FMM evaluation phase.

although for higher resolutions, GPUs seem to outperform the

x86 architectures. Recall that \mathbf{v}_{global} and \mathbf{v}_{local} utilize both CPU and GPUs. For example, the performance of \mathbf{v}_{local} on a dual socket, dual GPU node exceeds 800 GFlops/s for $m = 12$. From Figure 6, we observe that the GPU accelerated version of FMM is roughly three times faster than the CPU-only thus, delivering a combined 60–70 GFlops/s per node for \mathbf{v}_{global} . The only data transfers between host and device is for the FMM evaluation in which the host collects the information from all RBCs and then invokes FMM. This is somewhat suboptimal. We are working on having both GPU and CPU versions for all phases of the FMM.

MPI, strong scalability tests on Jaguar: The results are reported in Figure 7. The problem size is 300,000 RBCs with 84 points per RBC, which corresponds to 100,000,000 unknowns. The strong scalability results demonstrate excellent speed up resulting in an unprecedented five seconds per time-step on 24,576 cores.

A. Jaguar

p	48	384	3072	24576
Time (sec)	899.8	116.7	16.7	4.9
Efficiency	1.00	0.96	0.84	0.35

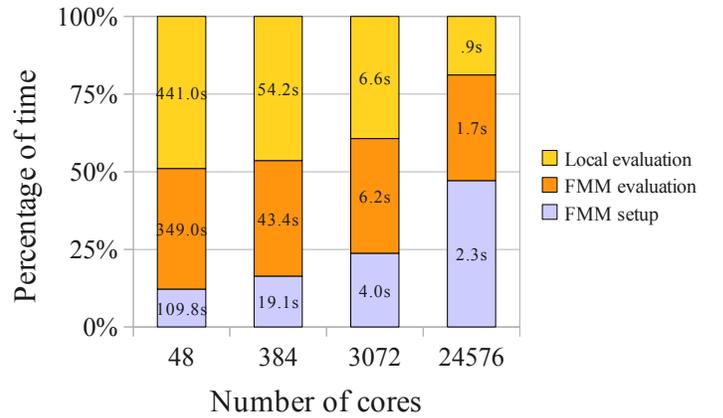


Fig. 7: STRONG SCALINGS ON JAGUAR PF. The strong scalability result for 262,144 vesicles, and total number of 22M grid points. There are 6 cores (and 6 OpenMP threads) per MPI process. The finest level of the octree is nine and the coarsest is three.

We get excellent speed up and we require less than 10 seconds per time step for 300,000 RBCs. The efficiency, of course, is reduced for the largest processor count as the memory traffic dominates the computations.

MPI, weak scalability tests on Jaguar: The results are reported in Figure 8. The problem size (number of RBCs) per core is kept fixed to 8000 RBCs, again with 84 points per RBC for the line-distribution on the Poiseuille flow. We can observe the the calculation of \mathbf{v}_{local} remains almost constant, whereas the cost of the tree-setup and \mathbf{v}_{global} increase. This is due to several reasons. As we increase the problem size, the tree gets deeper and the cost per core increases. Also

p	24576	98304	196608
Time (sec)	228.3	258	304.9
Efficiency	1.00	0.88	0.75

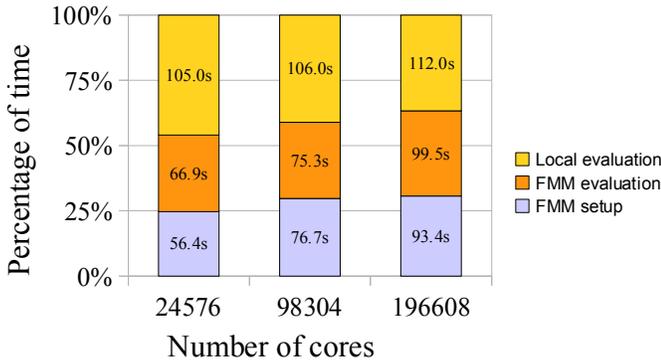


Fig. 8: WEAK SCALINGS ON JAGUAR PF. *The weak scalability of the simulation to 196,608 cores. We have chosen a line distribution of 8,000 RBCs with 84 points per RBC. We use one MPI process per socket and all of the six OpenMP threads in each MPI process. The finest to coarsest octree levels range from 24 to 4. In the largest, simulation, there are 200 million red blood cells and 90 billion unknowns. These results represent the average timings of four explicit Euler time steps.*

for such non-uniform trees it is difficult to load balance for all phases of FMM. The solution to these scaling problems is to employ the hypercube-like tree-broadcast algorithm we developed in [17] for all of the phases of FMM. (Currently it is used only in the post-order communication phase of the evaluation phase.) Finally, the setup is not multithreaded; we are currently working on this, and we expect a factor of four on more speed-ups in this calculation. Despite, the need for further optimizations, we achieve good utilization of the resources: the v_{local} phase sustains over 18 GFlops/s per core (single precision) and the v_{global} phase sustains over 1.2 GFlops/per core (double precision). Overall, the code exceeds 0.7 Petaflops of sustained performance.

GPU weak scalability results for FMM on Lincoln. We report these results in Figure 9. We only report results for the uniform distribution using 1M points per GPU. We use one socket per MPI process and one GPU per socket. In this experiment we use one core per socket. The results on GPUs are excellent on up to 256 processes/GPUs. We get over a 25X per core consistently and we were able to evaluate a 256-million particle sum in 2.3 seconds for a total of approximately 8 TFlops/s.

Red Blood Cell distributions and background flow: We test a line-like distribution of cells (Figure 10) on a Poiseuille background flow.⁴ The results of having a line of cells exposed to such a Poiseuille flow are easy to informally “verify” visually. Also, such a flow results in a highly non-uniform distributions

⁴More precisely, this is a “pseudo-Poiseuille flow”, since we do not impose confinement boundary conditions around the cells. Rather, we impose a free-space velocity that corresponds to a Poiseuille flow. Roughly speaking, such background flow corresponds to an unperturbed laminar flow in a blood vessel.

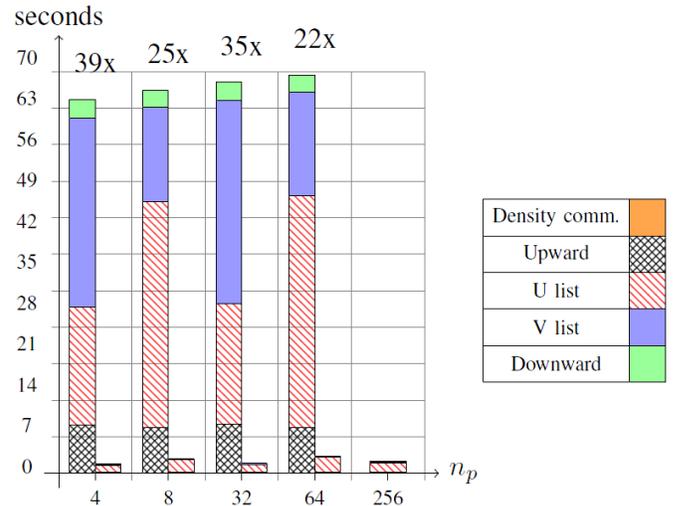


Fig. 9: GPU WEAK SCALING. *Here we compare CPU-only with GPU/CPU configuration on up to 256 processes. For the largest run the total evaluation on 256 million points takes 2.2 secs. Throughout the computation, we maintain a 25X speed-up over the single-core CPU version with only one thread per socket. When multithreading and vectorization is enabled, the differences become less pronounced as we can see in Figure 6. For the GPU runs, we use a shallower tree by allowing a higher number of points per box. In this way, we favor dense interactions over far-field computations. The former has a favorable computation/memory communication ratio and performs favorably on a GPU. In this examples, we used roughly 400 points per box for the GPU runs, and 100 points per box for the CPU runs. Both numbers were optimized for their respective architectures. We were able to maintain a 1.8-3 secs / evaluation for the GPU-based implementation. (This figure is reproduced from [17].)*

of points as the simulation horizon increases.

IV. CONCLUSIONS

We have presented MOBO, a framework that enables large-scale direct simulations of blood microcirculation. MOBO exposes and exploits concurrency at all stages of a complex multiphysics, multiscale problem and uses several parallel programming paradigms. We showed that we can efficiently scale the different parts of the method and we observe good scalability across different architectures.

For the computation of v_{local} , on a single node, we get roughly near peak performance for GEMM on both CPUs and GPUs. Our algorithmic choices were targeted to an extensive use of GEMM routines—without compromising overall work optimality. We are able to deliver spectral accuracy while using only a small number of degrees of freedom per RBC (e.g., compare to the 1000s of degrees of freedom per RBC for Lattice Boltzmann methods). For the global interaction, we have achieved 1 GFlop/s per CPU core for the overall FMM, which is quite remarkable given the complexity of the algorithm.

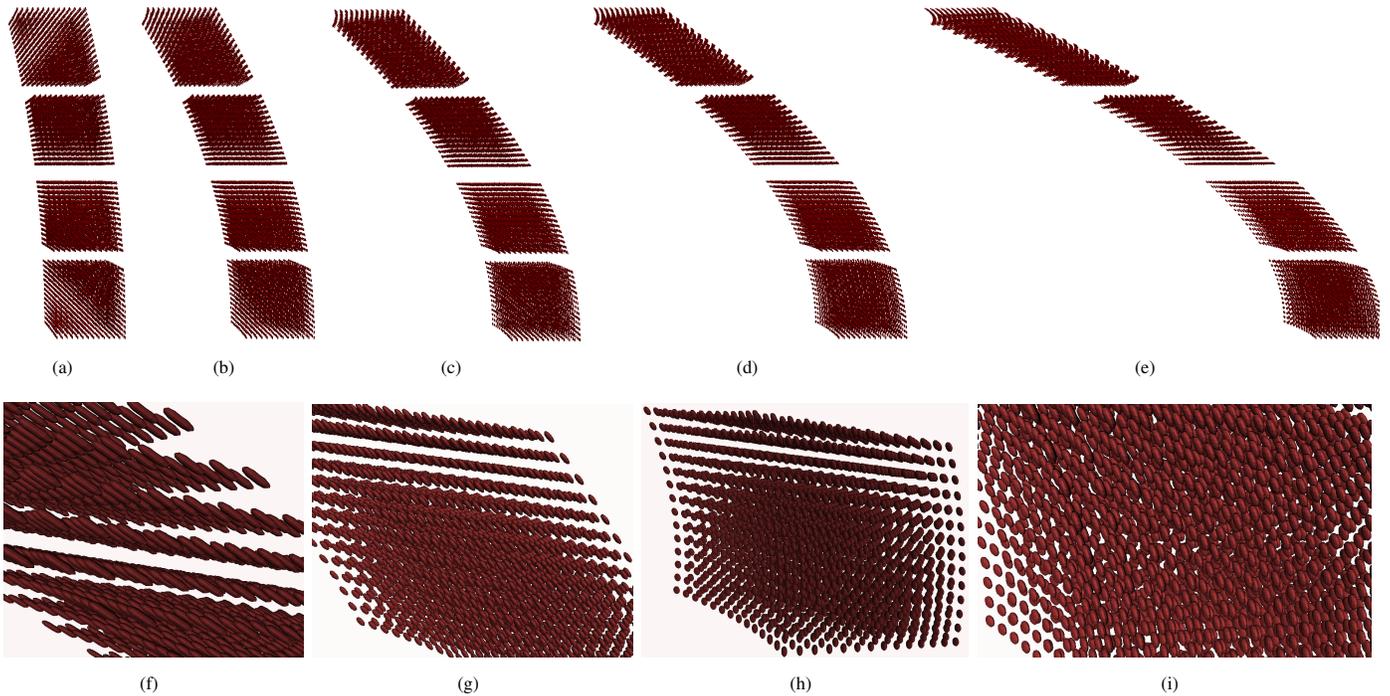


Fig. 10: SIMULATION RESULTS FOR 40,000 RBCs. In this figure, we present results from a 40,000-RBC simulation with 84 points per RBC for a total 15,000,000 unknowns. In the top row (a–e), we can observe the alignment of the cells with the background Poiseuille flow as we advance in time. We can verify the need for non-uniform solvers. Every time step of this simulation, requires a Stokes solve at the extraordinarily complicated domain defined by the exterior and interior of the RBCs. In addition, the interfacial forces at each RBC are computed by inverting an integro-differential operator that involves fourth order derivatives. In the bottom row, we zoom in on different regions of the flow snapshot (e). We can observe the different deformations of the cells in different regions of the flow. For example, (f) and (g) depict cells from the upper tip (e) in which the shear rate is higher and the cells experience larger deformations. Subfigures (h) and (i) depict cells from the bottom left of (e), which is near the center of the Poiseuille flow and thus, the cells experience smaller deformations. The visualization was performed on a single workstation using ParaView (www.paraview.org).

In our largest calculation on 196,608 cores, we achieved 0.7 petaflops for the multiphysics/multiscale problem of direct numerical simulation of blood flow. Let us emphasize that these results represent the worst case scenario with respect performance, as we use a very small number of points per cell. If we use an $m = 12$ spherical harmonics approximation for the RBCs the percentage of time spent in the v_{local} part of the calculation will further increase.

Taken together, MOBO opens the way for blood flow simulations of unprecedented fidelity. It enables the simulation of microfluidic and nanofluidic devices. For example, the 2D version of MOBO, has already resulted in significant scientific discoveries [15].

ACKNOWLEDGMENTS.

This work was supported by the U.S. National Science Foundation grants CNS-0929947, OCI-0749285 and OCI-0749334. Computing resources on the TeraGrid systems were provided under the grants ASC070050N and MCA04N026. We would like to thank the NCCS and TeraGrid support staff, from whom we have received significant assistance.

REFERENCES

- [1] G. Agresar, J.J. Linderman, G. Tryggvason, and K. G. Powell. An adaptive, Cartesian, front-tracking method for the motion, deformation and adhesion of circulating cells. *Journal Of Computational Physics*, 143(2):346–380, 1998.
- [2] P. Bagchi. Mesoscale simulation of blood flow in small vessels. *Biophysical Journal*, 92(6):1858–1877, 2007.
- [3] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc home page, 2001. <http://www.mcs.anl.gov/petsc>.
- [4] Becker, L. E. and Shelley, M. J. Instability of elastic filaments in shear flow yields first-normal-stress differences. *Physical Review Letters*, 8719(19), 2001.
- [5] George Biros, Lexing Ying, and Denis Zorin. A fast solver for the Stokes equations with distributed forces in complex geometries. *Journal of Computational Physics*, 193(1):317–348, 2003.
- [6] John P. Boyd. *Chebyshev and Fourier Spectral Methods*. 1999.
- [7] Aparna Chandramowlishwaran, Samuel Williams, Leonid Oliker, Ilya Lashuk, George Biros, and Richard Vuduc. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *Proceedings of IPDPS*, Atlanta, GA, 2010. IEEE Computer Society.
- [8] J.R. Clausen, D.A. Reasor Jr, and C.K. Aidun. Parallel performance of a lattice-Boltzmann/finite element cellular blood flow solver on the IBM Blue Gene/P architecture. *Computer Physics Communications*, 2010.
- [9] John B. Drake, Pat Worley, and Eduardo D’Azevedo. Algorithm 888: Spherical harmonic transform algorithms. *ACM Trans. Math. Softw.*, 35(3):1–23, 2008.
- [10] W. Dzwinel, K. Boryczko, and D. A. Yuen. A discrete-particle model of blood dynamics in capillary vessels. *Journal Of Colloid And Interface Science*, 258(1):163–173, 2003.

- [11] A. L. Fogelson and R. D. Guy. Platelet-wall interactions in continuum models of platelet thrombosis: formulation and numerical solution. *Mathematical Medicine And Biology-A Journal Of the IMA*, 21(4):293–334, 2004.
- [12] Z. Gimbutas and L. Greengard. Short note: A fast and stable method for rotating spherical harmonic expansions. *J. Comput. Phys.*, 228(16):5621–5627, 2009.
- [13] E. Givberg and K. Yelick. Distributed immersed boundary simulation in titanium. *SIAM Journal On Scientific Computing*, 28(4):1361–1378, 2006.
- [14] R. Glowinski, T.W. Pan, T.I. Hesla, D.D. Joseph, and J. P eriaux. A fictitious domain approach to the direct numerical simulation of incompressible viscous flow past moving rigid bodies: Application to particulate flow. *Journal of Computational Physics*, 169:363–426, 2001.
- [15] Badr Kaoui, George Biros, and Chaouqi Misbah. Why do red blood cells have asymmetric shapes even in a symmetric flow? *Physical Review Letters*, 103(18):188101, 2009.
- [16] Michael R. King and Daniel A. Hammer. Multiparticle adhesive dynamics: Hydrodynamic recruitment of rolling leukocytes. *PNAS*, 98(26):14919–14924, 2001.
- [17] I. Lashuk, A. Chandramowlishwaran, T-A. Nguyen H. Langston, R. Sampath, A. Shringarpure, R. Vuduc, D. Zorin L. Ying, and G. Biros. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *SC '09: Proceedings of the 2009 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2009. IEEE Press.
- [18] Y. L. Liu and W. K. Liu. Rheology of red blood cell aggregation by computer simulation. *Journal Of Computational Physics*, 220(1):139–154, 2006.
- [19] D. M. McQueen and C. S. Peskin. Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart. *Journal Of Supercomputing*, 11(3):213–236, 1997.
- [20] RR Nourgaliev, TN Dinh, TG Theofanous, and D. Joseph. The lattice Boltzmann equation method: theoretical interpretation, numerics and implications. *International Journal of Multiphase Flow*, 29(1):117–169, 2003.
- [21] Nhan Phan-Thien, Ka Yan Lee, and David Tullock. Large scale simulation of suspensions with PVM. In *Proceedings of SC97*, The SCxy Conference series, San Jose, CA, November 1997. ACM/IEEE.
- [22] I. V. Pivkin, P. D. Richardson, and G. Karniadakis. Blood flow velocity effects and role of activation delay time on growth and form of platelet thrombi. *Proceedings Of The National Academy Of Sciences Of The United States Of America*, 103(46):17164–17169, 2006.
- [23] C. Pozrikidis. Numerical simulation of the flow-induced deformation of red blood cells. *Annals of Biomedical Engineering*, 31(10):1194–1205, 2003.
- [24] Costas Pozrikidis. Interfacial dynamics for Stokes flow. *Journal of Computational Physics*, 169:250–301, 2001.
- [25] Abtin Rahimian, Shravan K. Veerapaneni, and George Biros. A fast algorithm for simulation of locally inextensible vesicles suspended in an arbitrary 2d domain. *Journal of Computational Physics*, 2010. in press.
- [26] Rahul Sampath, Santi S. Adavani, Hari Sundar, Ilya Lashuk, and George Biros. DENDRO home page, 2008.
- [27] Rahul S. Sampath, Santi S. Adavani, Hari Sundar, Ilya Lashuk, and George Biros. Dendro: parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.
- [28] K. M. Singh and J. J. R. Williams. A parallel fictitious domain multigrid preconditioner for the solution of Poisson’s equation in complex geometries. *Computer Methods In Applied Mechanics And Engineering*, 194(45-47):4845–4860, 2005.
- [29] C. H. Sun and L. L. Munn. Particulate nature of blood determines macroscopic rheology: A 2-D lattice Boltzmann analysis. *Biophysical Journal*, 88(3):1635–1645, 2005.
- [30] Hari Sundar, Rahul Sampath, Christos Davatzikos, and George Biros. Low-constant parallel algorithms for finite element simulations using linear octrees. In *Proceedings of SC2007*, The SCxy Conference series in high performance networking and computing, Reno, Nevada, 2007. ACM/IEEE.
- [31] T. E. Tezduyar and A. Sameh. Parallel finite element computations in fluid mechanics. *Computer Methods In Applied Mechanics And Engineering*, 195(13-16):1872–1884, 2006.
- [32] G. Tryggvason, B. Bunner, A. Esmaeeli, and N. Al-Rawahi. Computations of multiphase flows. In *Advances In Applied Mechanics, Vol 39*, volume 39 of *Advances In Applied Mechanics*, pages 81–120. Academic Press Inc, San Diego, 2003.
- [33] Shravan K. Veerapaneni, Denis Gueyffier, George Biros, and Denis Zorin. A numerical method for simulating the dynamics of 3d axisymmetric vesicles suspended in viscous flows. *Journal of Computational Physics*, 228(19):7233–7249, 2009.
- [34] Shravan K. Veerapaneni, Abtin Rahimian, George Biros, and Denis Zorin. A fast algorithm for simulating vesicle flows in three dimensions. 2010. Submitted for publication.
- [35] Lexing Ying, George Biros, Harper Langston, and Denis Zorin. KIFMM3D: The kernel-independent fast multipole (FMM) 3D code. GPL license.
- [36] Lexing Ying, George Biros, Denis Zorin, and Harper Langston. A new parallel kernel-independent fast multiple algorithm. In *Proceedings of SC03*, The SCxy Conference series, Phoenix, Arizona, November 2003. ACM/IEEE.
- [37] L. T. Zhang, G. J. Wagner, and W. K. Liu. A parallelized meshfree method with boundary enrichment for large-scale CFD. *Journal Of Computational Physics*, 176(2):483–506, 2002.
- [38] H. Zhao, A.H.G. Isfahani, L.N. Olson, and J.B. Freund. A Spectral Boundary Integral Method for Flowing Blood Cells. *Journal of Computational Physics*, 2010.
- [39] Alexander Z. Zinchenko and Robert H. Davis. An efficient algorithm for hydrodynamical interaction of many deformable drops. *Journal of Computational Physics*, 157:539–587, 200.
- [40] A.Z. Zinchenko and R. H. Davis. Large-scale simulations of concentrated emulsion flows. *Philosophical Transactions Of The Royal Society Of London Series A-Mathematical Physical And Engineering Sciences*, 361(1806):813–845, 2003.