Optimizing Parallel Performance of the Cell Based Blood Flow Simulation Software HemoCell

Victor Azizi Tarksalooyeh[1(✉)], Gábor Závodszky[1], and Alfons G. Hoekstra[1,2]

[1] Computational Science Lab, Institute of Informatics, University of Amsterdam, Amsterdam, The Netherlands
v.w.azizitarksalooyeh@uva.nl

[2] ITMO University, Saint-Petersburg, Russian Federation

Abstract. Large scale cell based blood flow simulations are expensive, both in time and resource requirements. HemoCell can perform such simulations on high performance computing resources by dividing the simulation domain into multiple blocks. This division has a performance impact caused by the necessary communication between the blocks. In this paper we implement an efficient algorithm for computing the mechanical model for HemoCell together with an improved communication structure. The result is an up to 4 times performance increase for blood flow simulations performed with HemoCell.

Keywords: Blood flow simulation · High performance computing · Computational optimization

1 Introduction

Blood flow simulation remains an area of active research. Many interesting properties have been identified with the help of simulations [4,6,9–11,13]. There is an increasing interest in blood flow simulations in which the blood cells (red blood cells, platelets, white blood cells) are fully resolved [3,8,15,16]. These simulations can be used to understand and find underlying mechanics of complex behaviour of blood flows including but not limited to platelet margination [9], the formation of the cell free layer [6], the Fåhræus–Lindqvist effect [2], the behaviour in microfluidic devices or the behaviour around micromedical implants [1,5]. Simulations that model blood as a pure fluid flow are not able to recover these intricate properties of blood flow.

One of the challenges of suspension simulation codes is to parallelize them such that interesting systems with sufficient number of cells (>1000 cells) can be simulated for an extended duration (>0.1 s) in a reasonable time span (<5 days). Only a few open source solutions exist for suspension simulations that need to implement a complex mechanical model for the simulated cells, HemoCell [16] and Palabos-LAMMPS [14] are examples of available open-source codes that can

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11538, pp. 537–547, 2019.
https://doi.org/10.1007/978-3-030-22744-9_42

be used to simulate blood flow. Other codes exists but are not (yet) available as open-source.

HemoCell is a software package that is developed at the University of Amsterdam that is able to simulate blood flow at high shear rates (>1000 s^{-1}) and with a high number of cells (>1000 cells). In this paper we present HemoCell an highly efficient parallel code for blood flow suspension simulations.

HemoCell is built on top of Palabos [7] and offers support for complex suspension simulations. Palabos is a general purpose lattice Boltzmann solver with high performance computing capabilities. We will shortly introduce HemoCell and its underlying models, followed by a discussion of challenges and solutions for efficient parallel simulations. These include boundary communication of processors for the suspension part, efficiently storing relevant information while avoiding global communication, and efficiently computing the complex material model associated with the cells within HemoCell. Next, we discuss the theoretical and practical implications of the methods we used to implement the suspension simulation software within HemoCell and provide performance measurements.

1.1 HemoCell

HemoCell [16] is an open source parallel code for simulating blood flows with fully resolved cells that is built as a library on top of Palabos [7]. Palabos is a versatile library which can be used to solve pure fluid flow problems with the lattice Boltzmann method (LBM). Palabos offers relevant multi-processing abilities. HemoCell implements the cell mechanics simulations and their coupling to the fluid using the immersed boundary method (IBM), see also Fig. 1.

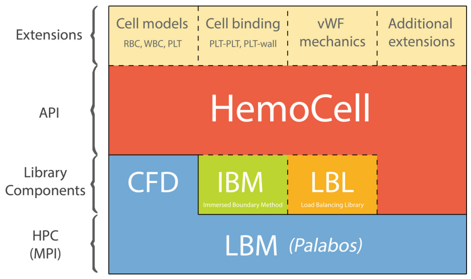


Fig. 1. Overview of the Palabos and HemoCell libraries

HemoCell uses data parallelism to distribute the workload over many cores. With the help of PalaBos HemoCell can divide the flow domain into multiple rectangular blocks of each which represents a processor These domains are called atomic blocks (AB). ABs are abstracted away from the user through the use of functionals, which can be used to perform operations on a domain without knowing about the underlying distributed structure. Furthermore, each simulation can have multiple fields, which span the whole domain and represent a specific part of the simulation. In Hemocell two fields are used, a fluid field and a cell field. Palabos takes care of the boundary communication between processors for the fluid field. HemoCell takes care of the cell field and of the communication between the two fields as required by the immersed boundary method [12].

The cells consist of vertices which are connected through links that make up a boundary to represent a cell in the fluid. A RBC in HemoCell has 1280 vertices. A complex mechanical model is used to calculate forces [16]. This mechanical model requires that a cell is present on both processors whenever it is crossing a boundary. This results in the two main bottlenecks and thus challenges for HemoCell.

1. The material model of the cells needs to be calculated efficiently.
2. Dividing the cell field into multiple processors is complex because the material model requires duplication of cells over boundaries.

2 Calculating the Mechanical Model of a Cell

The cells within HemoCell are implemented as vertices and connections that form a triangulated mesh. These cells compute the forces acting on its vertices through a mechanical model [16]. Figure 2 shows a mesh used to represent a red blood cell.

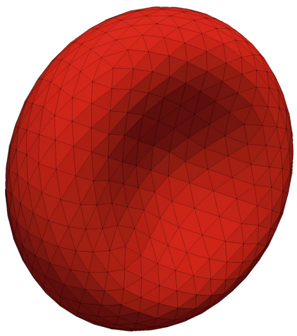


Fig. 2. Mesh representing red blood cell in HemoCell

Závodszky et al. [16] model the forces acting on the vertices of a cell as follows:

$$F_{\text{total}} = F_{\text{link}} + F_{\text{bend}} + F_{\text{volume}} + F_{\text{area}} + F_{\text{visc}} \tag{1}$$

Below we list all five forces and explain in detail what information is needed to compute them.

1. The link force F_{link} acts along the edges and between neighbouring points. The force on a single vertex i (F^i_{link}) can be described as follows:

$$F^i_{\text{link}} = \sum_{n=1}^{m} C_{\text{link}} \frac{E_{i,i_n} - |i^x_n - i^x|}{E_{i,i_n}} \tag{2}$$

   Where E_{i,i_n} is the equilibrium length between two vertices, i^x is the location of vertex i, m is the number of direct neighbours of vertex i and i_n is the n'th direct neighbour of vertex i. C_link consists of all the constant terms that do not change during a simulation as explained by Závodszky et al. [16].
2. The bending force F_{bend} uses patches which are defined as a plane that goes through the average location of all direct neighbours of vertex i. The normal direction of this plane is defined as the average normal of all neighbouring triangles that include vertex i. The distance along the normal direction of this plane towards vertex i is used to calculate the bending force on vertex i, a negative term is added to the neighbours of i to make the force zero-sum.

$$F^i_{\text{bend}} = C_{\text{bend}} \left(E^{\text{patch}}_i - \left(\frac{\sum_{n=1}^{m} i^x_n}{m} - i^x \right) \cdot \left(\frac{\sum_{n=1}^{m} \mathbf{normal}\left(t^n_i\right)}{L} \right) \right) - \sum_{n=1}^{m} \frac{1}{N^m_i} F^{i_n}_{\text{bend}} \tag{3}$$

   Where E^{patch}_i is the equilibrium distance between the patch and the vertex i along the patch normal. t^n_i is the n'th triangle that is a direct neighbour of vertex i. **normal**() returns the normal pointing outward from a triangle. L is the length of the summation of the normal vectors of all the triangles that are part of the patch, thus this division results in a unit vector along the average normal direction. The dot product results in a length term along the patch normal. N^m_i is the number of direct vertex neighbours of i_n. C again of all the constant terms that do not change during a simulation.
3. The area force F_{area} acts on all the triangles that are part of the mesh. Therefore the force on a single vertex is a sum over all neighbouring triangles:

$$F^i_{\text{area}} = \sum_{n=1}^{m} \mathbf{C}_{\text{area}} \left(\frac{E^{\text{area}}_{i^n_t} - \mathbf{area}\left(i^n_t\right)}{E^{\text{area}}_{i^n_t}} \right) \left(i^x - \mathbf{middle}\left(i^n_t\right)\right) \tag{4}$$

   Where **area**() calculates the area of a triangle, $E^{\text{area}}_{i^n_t}$ is the equilibrium value for the area of triangle i^n_t. **middle**() calculates the average of the three triangle vertices of triangle i^n_t. $\mathbf{C}_{\text{area}}()$ is a function that takes the area ratio as input and outputs a force coefficient.
4. The volume force F_{volume} results from the total volume of the cell, thus information about all vertices is needed. The force is distributed over the vertices proportional to the area of the direct neighbouring triangles of that vertex.

$$F^i_{\text{volume}} = \frac{\mathbf{volume}(\text{cell}^i) - E^{\text{volume}}_{\mathbf{cell}^i}}{E^{\text{volume}}_{\text{cell}^i}} \sum_{n=1}^{m} C_{\text{volume}} \frac{\mathbf{area}\left(t^n_i\right)}{E^{\text{area}}_{t^n_i}} \mathbf{normal}\left(t^n_i\right) \tag{5}$$

Where **volume** () calculates the volume of a complete cell, this function needs every vertex of the cell as input. $E^{\text{volume}}_{\text{cell}^i}$ is the equilibrium volume of cell^i and **normal**() is the normal direction of triangle t^n_i.

5. The viscous force F_{visc} limits the relative velocity of neighbouring vertices connected with an edge.

$$F^i_{\text{visc}} = \sum_{n=1}^{m} C_{\text{visc}} \cdot \left(\left(v^i - v^i_n\right) \cdot \left(\frac{i^x_n - i^x}{|i^x_n - i^x|} \right) \right) \cdot \left(\frac{i^x_n - i^x}{|i^x_n - i^x|} \right) \tag{6}$$

Where v^v and v^i_n are the velocity of vertex i and i_n respectively. $\sum_{n=1}^{m}$ sums over all direct vertex neighbours of i.

2.1 Implementation of the Mechanical Model

The formulas for calculating force on each independent vertex are explained above. Between the calculation of the separate forces there are some overlaps, for example the calculation of the area of a triangle is used for both the volume and area forces Eqs. 4 and 5. This leaves room for optimization within implementing the calculations. In Fig. 3 a pseudo code of the implementation is shown. In this implementation we have tried to calculate each necessary value only once. Furthermore, we try to minimize the number of loops. Most notably in the first loop which calculates F_{area} all the necessary calculations for F_{volume} are stored for the second loop. In addition F_{link} and F_{visc} are calculated in the same loop as well.

```
for triangle in cell.triangles:
  volume += volume_from_triangle(triangle);
  normal,area,center = triangle_properties(triangle)
  area_force = ((area - eq(area))/eq(area)) * C_area
  for vertex in triangle:
    vertex.force += (center-vertex)*area_force
volume_force = ((volume - eq(volume))/eq(volume)) * C_volume
for triangle in cell.triangles:
  triangle_volume_force = triangle_volume_force_formula()
  for vertex in triangle:
    vertex.force += triangle_volume_force
for vertex in cell:
  for neighbour in vertex:
    middle += neighbour
  vertex.force +=bending_force_formula(middle, vertex)
for edge in cell:
  vertex.force += link_force_formula(edge)
  vertex.force += visc_force_formula(edge)
```

Fig. 3. Pseudocode explaining how we optimized the calculation of the mechanical model within HemoCell.

3 Implementation of the Cell Field Communication Structure

When the cell field is divided up into multiple atomic blocks it becomes necessary to implement a communication structure. For a regular fluid field this simply constitutes to communicating the values of the fluid cells in the boundary layer to their corresponding neighbours. However it is not so simple for the cell field. The number of vertices in a communication boundary can change over time and therefore the communication size is not static but dynamic. Furthermore at every communication step it has to be determined which vertices are present within a communication boundary and which vertices are not.

Cells need information from all their vertices to calculate the mechanical forces. Almost all forces ($F_{\text{area}}, F_{\text{link}}, F_{\text{bend}}, F_{\text{visc}}$) that act on the vertices only need information from their direct neighbours to be calculated. However the volume force F_{volume} needs information of all the vertices of the cell to be calculated. Therefore whenever a single vertex of a cell is present in an atomic block, the boundaries must include every other vertex of the corresponding cell as well. This means that the size of the boundary must be larger than the largest possible diameter of a cell. Figure 4 shows that a larger boundary size means that the number of neighbours and thus the communication will increase if the atomic blocks get too small.

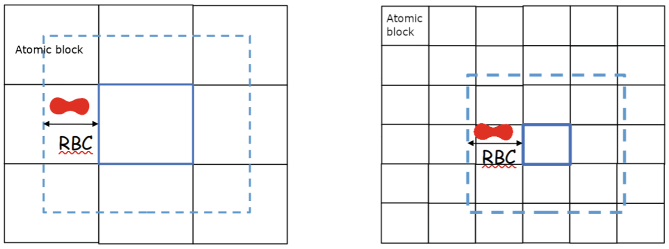


Fig. 4. Visualization of the boundary size needed for the cell field.

There is a simple way to implement this boundary, namely by communication of vertices in the boundary. We will use this communication pattern as the base upon which we propose improvements, see Fig. 5. In the naïve implementation firstly all neighbours are determined that overlap with the boundary of the atomic block. Within HemoCell a RBCs (the largest cell) can stretch up to 12 µm. Thus all neighbours within a 12 µm range send the vertices corresponding to the overlap they have with the boundary. This method has two drawbacks: First a lot of unnecessary data is communicated and second when the boundary size is larger than an atomic block the number of neighbours with which communication is necessary grows, usually in the form of $(2N+1)^3-1$ Where N is the number of neighbours in a single direction. So going from $N=1$ to $N=2$ creates $124-26=98$ extra neighbours.

```
neighbours = block.neighbours(12) for neighbour in neighbours:
  send_particles = block.findparticles(intersect(block, neighbour)
  MPI_Isend(neighbour,send_particles.size())
  MPI_Isend(neighbour,send_particles)
While (MPI_WaitAny(neighbours)):
  MPI_Recv(neighbour, size)
  MPI_Recv(neighbour, recv_particles, size)
  block.add_particles(recv_particles)
```

```
neighbours = block.neighbours(1)
  #Same communication pattern as top code block
  #But with a boundary of size one
Neighbours = block.neighbours(12) requested_cells =
block.findlocalcellIds() for neighbour in neighbours:
  MPI_Isend(neighbour,requested_cells)
for neighbour in neighbours:
  MPI_Probe(neighbour) #Get any neighbour
  MPI_Recv(neighbour, requested_cells)
  send_particles =
block.findParticlesFromCells(requested_cells)
  MPI_Isend(neighbour, send_particles)
for (neighbour in neighbours):
  MPI_Probe()          #Get any neighbour
  MPI_Irecv(neighbour,recv_buffer)
for (neighbour in neighbours):
  MPI_WaitAny(receive) #Wait for any receive
  block.addParticles(recv_buffer)
MPI_WaitAll(sends)
```

Fig. 5. The top block shows in pseudocode a naïve implementation of the boundary communication. The bottom block shows our optimized implementation of the boundary communication algorithm.

We implemented an improved and consequently faster method to communicate vertices in boundaries. The main idea is to only communicate vertices of cells that are needed. For this an extra communication step needs to be implemented. In this extra communication step an atomic block sends a list with all the IDs of the cells that need to be communicated to its neighbours. In the next communication step only these vertices are communicated. It is not possible to get rid of the inefficient boundary communication entirely as vertices very close to the domain are needed for non-local force calculations (e.g. inter cellular forces). However, this is much more efficient if only a very small boundary needs to be communicated.

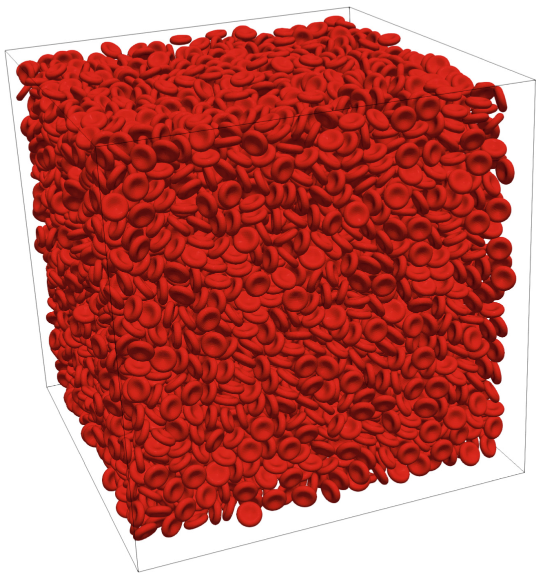
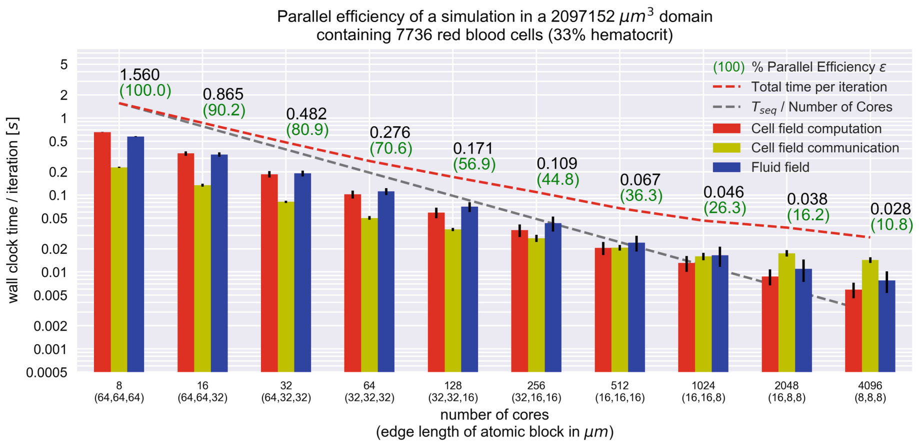


Fig. 6. The domain with which the simulations are performed with a differing number of processors

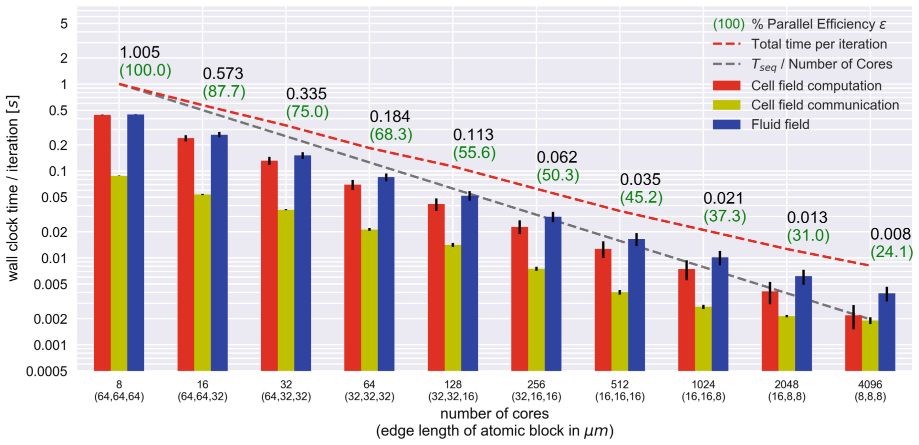
3.1 Comparison Between Naïve and Optimized Implementation of the Boundary Communication Algorithm

To test the performance gain of our optimized boundary communication algorithm we have set up a simulation which is executed both with the naïve and the optimized implementation. The simulation consists of a cubic 128 μm^3 volume that is periodic in all directions. Within this volume 7736 red blood cells are present. Figure 6 shows the simulated domain. An external body force is applied to drive the cell suspension inside the volume. The volume is simulated for 0.1 s and statistics are collected over the whole duration. The results are plotted in Fig. 7.

The results show a significant improvement of HemoCell in two ways. Firstly, the base performance has improved by $\approx$36%, this can be deducted from the difference in wall clock time per iteration in Fig. 7 for 8 cores. Secondly, the strong scaling (dividing the same domain into more smaller atomic blocks) properties are better. In the worst case (512 μm^3 per atomic block) the edges of an atomic block are only 8 μm long. This means that the boundary of each block overlaps with 124 neighbours. In this case we see a performance improvement of $\approx$4 times over the naïve version. Over the whole range we see that our improved communication performs better.



a)



b)

Fig. 7. Statistics for each of the simulations. The fluid part is handled by Palabos. The dotted line shows perfect linear scaling. (a) shows the statistics for the naïve implementation of the communication. (b) shows the statistics for our improved implementation.

4 Conclusions

Improving the performance of fully resolved blood flow simulations allows us to perform simulations up to 4 times faster. For a simulation of 1 s a total number of 10 million timesteps is required. This means that the improved version of HemoCell only needs one day to complete this simulation with ABs of 512 μm^3, whereas the naïve version would need four days.

We have shown that it is possible to merge the calculation the forces of the mechanical model in such a way that there is less computation than when all the forces are computed separately. This is achieved by re-using intermediate values and combining loops where possible.

By improving the communication structure better strong scaling results are achieved for HemoCell. Furthermore, the base performance with large ABs is improved as well.

Acknowledgments. This work was supported by the European Union Horizon 2020 research and innovation programme under grant agreement no. 675451, the CompBioMed project and grant agreement no. 671564, the ComPat project.

References

1. Augsburger, L., Reymond, P., Rufenacht, D., Stergiopulos, N.: Intracranial stents being modeled as a porous medium: flow simulation in stented cerebral aneurysms. Ann. Biomed. Eng. **39**(2), 850–863 (2011)
2. Bagchi, P.: Mesoscale simulation of blood flow in small vessels. Biophys. J. **92**(6), 1858–1877 (2007)
3. Bernaschi, M., Melchionna, S., Succi, S., Fyta, M., Kaxiras, E., Sircar, J.: MUPHY: a parallel MUlti PHYsics/scale code for high performance bio-fluidic simulations. Comput. Phys. Commun. **180**(9), 1495–1502 (2009)
4. Czaja, B., Závodszky, G., Azizi Tarksalooyeh, V., Hoekstra, A.: Cell-resolved blood flow simulations of saccular aneurysms: effects of pulsatility and aspect ratio. J. Roy. Soc. Interface **15**(146), 20180485 (2018)
5. Farb, A., Burke, A.P., Kolodgie, F.D., Virmani, R.: Pathological mechanisms of fatal late coronary stent thrombosis in humans. Circulation **108**(14), 1701–1706 (2003)
6. Fedosov, D.A., Caswell, B., Popel, A.S., Karniadakis, G.E.: Blood flow and cell-free layer in microvessels. Microcirculation **17**(8), 615–628 (2010)
7. Latt, J.: Palabos, parallel lattice Boltzmann solver (2009). https://palabos.org
8. Moeendarbary, E., Ng, T.Y., Zangeneh, M.: Dissipative particle dynamics: introduction, methodology and complex fluid applications - a review. Int. J. Appl. Mech. **01**(04), 737–763 (2009)
9. Mountrakis, L., Lorenz, E., Hoekstra, A.G.: Where do the platelets go? A simulation study of fully resolved blood flow through aneurysmal vessels. Interface Focus **3**(2), 20120089 (2013)
10. Mountrakis, L., Lorenz, E., Hoekstra, A.G.: Scaling of shear-induced diffusion and clustering in a blood-like suspension. EPL (Europhys. Lett.) **114**(1), 14002 (2016)
11. Ouared, R., Chopard, B.: Lattice Boltzmann simulations of blood flow: non-Newtonian rheology and clotting processes. J. Stat. Phys. **121**(1–2), 209–221 (2005)
12. Peskin, C.S.: The immersed boundary method. Acta Numerica **11**, 479–517 (2002)
13. Skorczewski, T., Erickson, L., Fogelson, A.L.: Platelet motion near a vessel wall or thrombus surface in two-dimensional whole blood simulations. Biophys. J. **104**(8), 1764–1772 (2013)
14. Tan, J., Sinno, T.R., Diamond, S.L.: A parallel fluid-solid coupling model using LAMMPS and Palabos based on the immersed boundary method. J. Comput. Sci. **25**, 89–100 (2018)

15. Ye, T., Phan-Thien, N., Lim, C.T.: Particle-based simulations of red blood cells-a review. J. Biomech. **49**(11), 2255–2266 (2016)
16. Závodszky, G., van Rooij, B., Azizi, V., Hoekstra, A.: Cellular level in-silico modeling of blood rheology with an improved material model for red blood cells. Front. Physiol. **8**, 563 (2017)