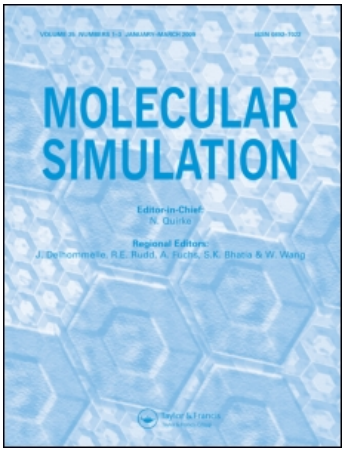

Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

Modeling Heterogeneous Mesoscopic Fluids in Irregular Geometries using Shared Memory Systems

Krzysztof Boryczko[a]; Witold Dzwinel[a]; David A. Yuen[b]
[a] Institute of Computer Science, AGH University of Technology, Kraków, Poland [b] Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN, USA

To cite this Article Boryczko, Krzysztof , Dzwinel, Witold and Yuen, David A.(2005) 'Modeling Heterogeneous Mesoscopic Fluids in Irregular Geometries using Shared Memory Systems', Molecular Simulation, 31: 1, 45 — 56
To link to this Article: DOI: 10.1080/08927020412331299341
URL: http://dx.doi.org/10.1080/08927020412331299341

Molecular Simulation, Vol. 31 (1), 15 January 2005, pp. 45–56

Taylor & Francis
Taylor & Francis Group

Modeling Heterogeneous Mesoscopic Fluids in Irregular Geometries using Shared Memory Systems

KRZYSZTOF BORYCZKO[a,*], WITOLD DZWINEL[a] and DAVID A. YUEN[b]

[a]*Institute of Computer Science, AGH University of Technology, Mickiewicza 30, 30-059 Kraków, Poland;* [b]*Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN 55455, USA*

(Received August 2004; In final form August 2004)

We discuss the use of current shared-memory systems for discrete-particle modeling of heterogeneous mesoscopic complex fluids in irregular geometries. This has been demonstrated by way of mesoscopic blood flow in bifurcating capillary vessels. The plasma is represented by fluid particles, while the other blood constituents are made of "solid" particles interacting with harmonic forces. The particle code was tested on 4 and 8 processors of SGI/Origin 3800 (R14000/500), IBM Regatta (Power4/1300), SGI Altix 3000 (Itanium®2/1300) systems and 2-processor AMD Opteron 240 motherboard. The tests were performed for the same system employing two million fluid and "solid" particles. We show that irregular boundary conditions and heterogeneity of the particle fluid inhibit efficient implementation of the model on superscalar processors. We improve the efficiency almost threefold by reducing the effect of computational imbalance using a simple load-balancing scheme. Additionally, in employing MPI on shared memory machines, we have constructed a simple middleware library to simplify parallelization. The efficiency of the particle code depends critically on the memory latency. Therefore, the latest architectures with the fastest CPU-memory interface, such as AMD Opteron and Power4, represent the most promising platforms for modeling the complex mesoscopic systems with fluid particles. As an example of application of small, shared-memory clusters in solving very complex problems we demonstrate the results of modeling red blood cells clotting in blood flow in capillary vessels due to fibrin aggregation.

Keywords: Parallel algorithm; Load-balancing; Fluid particle model; Blood flow dynamics; Shared memory systems

INTRODUCTION

Mesoscopic modeling of the physical and chemical processes can be carried out by using the discrete-particle paradigm. Attacking this sort of problems by using non-equilibrium molecular dynamics (NEMD) approach [1] requires multi-billion particle ensembles and high-performance computing tools, such as distributed memory systems consisting of hundreds and thousands of processors [2]. As shown in Ref. [3], in the mesoscale the fast modes of particle motion can be eliminated or averaged out in favor of a coarse-grain representation—fluid particles. This level of granularity allows for modeling mesoscopic phenomena by using a smaller ensemble of fluid particles instead of a large number of discrete atoms.

In Ref. [4], we presented the results from parallel computations of the fluid particle model (FPM) and the results of its implementation on 2 multiprocessor platforms of different architecture: distributed memory machine IBM SP with Power3+/375 and non-uniform memory access (NUMA) SGI/Origin 3800 with R14000/500. The speed-up of 13 and 26 on 32 processors of IBM SP and SGI/Origin 3800, respectively, was reported for 16 million fluid particles in 3D. It is considerably lower than for optimized distributed memory NEMD codes [2]. The reasons are that our code was not tuned for a specific computer architecture and the memory and communication loads required per FPM particle is much greater than for MD atom. We showed that the effect of small cache dictates the program efficiency on both machines, but communication overhead is distinctly lower on a virtually shared memory SGI/Origin than on a distributed memory IBM SP.

The results discussed in Ref. [4] come from using the version of fluid particle code, which was

*Corresponding author. E-mail: boryczko@uci.agh.edu.pl

ISSN 0892-7022 print/ISSN 1029-0435 online © 2005 Taylor & Francis Ltd
DOI: 10.1080/08927020412331299341

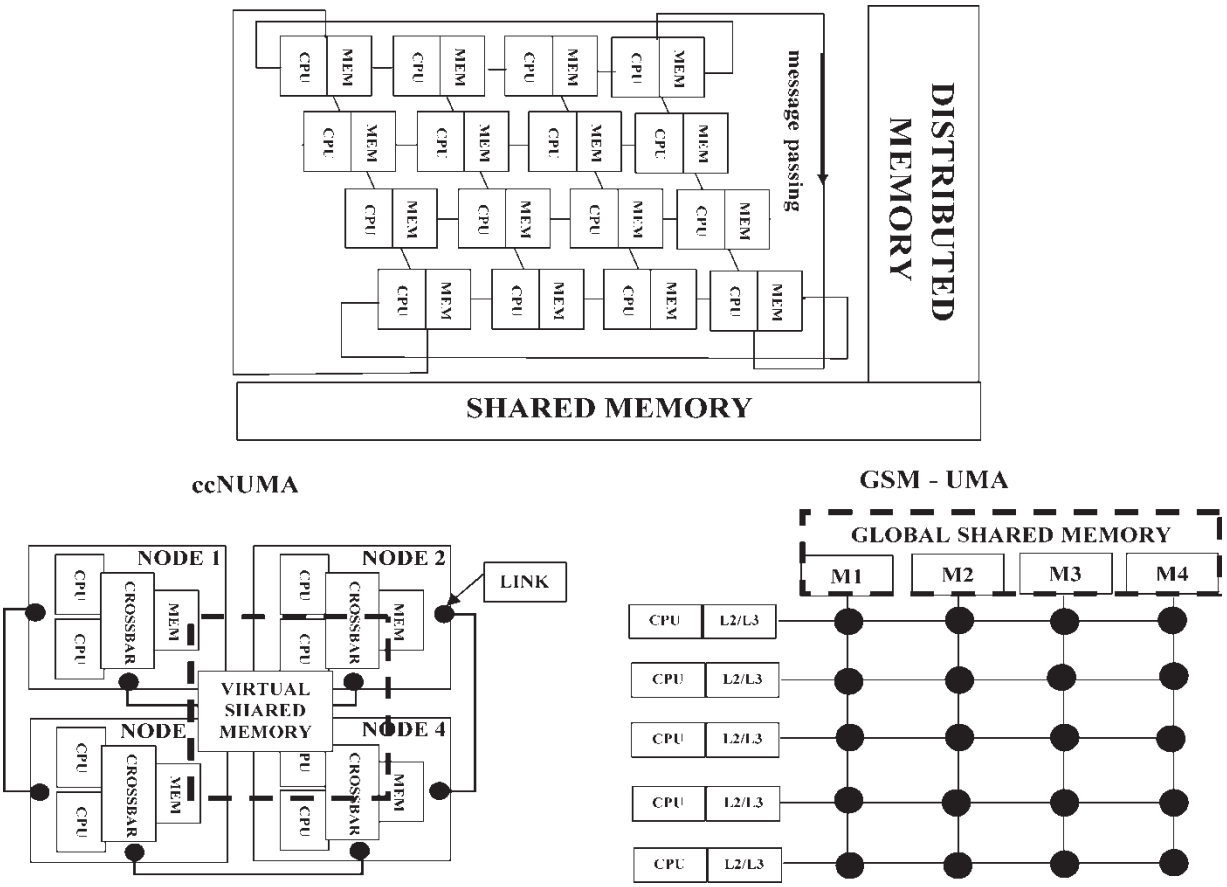

FIGURE 1 Distributed versus shared memory architectures: cache coherent non-uniform memory access (ccNUMA) and global shared memory — uniform memory access machines (GSM-UMA).

originally designed for geometrically regular problems. Implementations of discrete particle methods in simulating flows in irregular geometries, such as red blood cells clotting in bifurcating capillaries, involve using load-balancing schemes. However, due to the load-balancing, the code will require greater communication bandwidth. Consequently, more efficient communication schemes between local memories of distributed memory system (see Fig. 1) have to be implemented to realize message passing paradigm.

Instead of tuning the communication schemes, thus making them more and more platform dependent, we use shared memory systems, such as the virtual (NUMA) and global shared memory (GSM) architectures for which processes communicate using main memory (see Fig. 1). Shared memory is an efficient tool of passing data between processes. One process will create a memory portion, which other processes can access. The attaching procedure must have the appropriate permissions. Once attached, the process can read or write to the segment, as allowed by the permission requested in the attach operation.

This issue is reasonable due to reduction of the problem size of mesoscopic modeling — partially at the expense of more complex functional model of interactions—by a coarse-graining procedure. Smaller number but powerful processors with large memory allows for simplifying sophisticated domain decomposition used in NEMD [2,5]. Consequently, complicated communication algorithms and load-balancing procedures, which are designed for achieving a maximal efficiency for distributed memory systems, can be replaced with simpler schemes.

The paper is organized as follows. First, we introduce and discuss the shared memory platforms and processors we have employed. Second, we describe briefly the main assumptions of the parallel fluid particle algorithm. Then we present the load-balancing procedures and analyze the timings. Finally, we discuss the conclusions.

PRINCIPAL FEATURES OF SELECTED SHARED MEMORY SYSTEMS

A slow memory access time poses a huge bottleneck for achieving high efficiency for problems, such as N-body simulations both on distributed memory and shared memory systems. There exist

many CPU-memory interfaces on shared memory machines involving cache memories of various architectures and sizes, a choice of memory bandwidths, memory latencies and CPUs-memory interconnections. To find the most suitable architecture, we measure the efficiency of fluid particle code running on the following multi-processor architectures:

1. cache coherent non-uniform memory access (ccNUMA) SGI/Origin 3800 with R14000/500 processors supported with two level cache and relatively large L2 cache (8 MB),
2. symmetric multiprocessor (SMP) IBM Regatta with Power4/1300 processors supported with three level cache with much smaller L2 (1.44 MB shared by 2 processors) but large 32 MB L3 cache,
3. GSM SGI Altix 3000 system with Itanium®2/1300 processor with small L2 and L3 caches (256 kB and 3 MB, respectively) but very fast access to the main memory,
4. Opteron 240 motherboard with two AMD Opteron 1.4 GHz processors and small L2 cache (1 MB) but very fast access to the main memory.

The basic features of these systems are shown in Table I.

The IBM Regatta (http://www.utexas.edu/research/) machine has a typical SMP architecture with uniform access time to operational memory (UMA machine). The basic building block of this pSeries 690 Turbo system is a single Power4 chip. Each chip has two cores, or CPUs, that share a common L2 cache. The path to memory through the L3 cache (13.8 GB/s) is also shared by both cores. Each core processor has two floating point units each of which perform a "fused" multiply-add operation per clock cycle. At a clock speed of 1.3 GHz, four floating point operations per cycle can deliver 5.2 Gflop/s from each processor. The next architectural level for Power4 systems is the Multi-Chip Module (MCM). Connecting 4 dies on a multichip module forms an "HPC" "Turbo" 8-way (dual core) SMP that shares all L3 and memory modules associated with the MCM. Chip-to-chip interconnect "controller", interconnects 4 chips to form a distributed crossbar switch. These interconnect fabric controllers allow each Power4 chip to access the L3 cache and memory of the other 3 chips. There exists also the MCM-to-MCM interconnection that allows nodes to be built-up with multiple MCM modules (up to 4). However, in our test we use only a single 8-processor MCM board.

SGI Altix 3000 servers (http://www.sgi.com/servers/altix/) and superclusters featuring the new Intel Itanium®2 processors (1.30 GHz with 3M L3 cache) are Linux-based machines. The Itanium represent the new IA64 technology and EPIC (explicitly parallel instruction computer) architecture. This type of architecture is characterized by large instruction words of 128 bits that contain 3 41-bit instructions and a 5-bit template that aids in steering and decoding the instructions. The two load/store units fetch two instruction words per cycle so six instructions per cycle are dispatched. The Itanium has also in common with these systems in that the scheduling of instructions, unlike in RISC processors, is not carried out dynamically at run time but rather by the compiler. Instead of large cache memories, the L2cache-memory latency is very low for Itanium (see Table I). Moreover, the SGI Altix 3000 family of servers and superclusters brings a new scalability feature to Linux clusters—GSM. Systems with GSM allow for accessing all data in the system's memory directly and efficiently, without having to move data through I/O or networking bottlenecks. GSM requires a sophisticated system memory interconnect, like the SGI NUMAlink and application libraries, that enable shared-memory calls, such as MPT and XPMEM from SGI.

The AMD64 architecture (http://www.tomshardware.com/cpu/20030422/opteron-06.html) extends the 32-bit register of the IA-32 processors to 64-bit Opteron processor (clock speed ranging from 1.4 to 2 GHz). Instead of the usual parallel FSB, the CPU communicates via a HyperTransport interface. The serial interface with a variable bit-rate allows the SledgeHammer to attain a data transfer rate of 3.2 GB/s—in both directions simultaneously. This results in a total bandwidth of 6.4 GB/s per Hypertransport port. The entire data traffic of the Opteron processor runs through the HyperTransport interface and the integrated memory controller. In order to let the neighboring CPU gain direct access to its system memory, the Opteron uses the XBAR switch. For commands and addresses, the XBAR switch has additional 64-bit buses available. The memory latency achieved by the Opteron is 65 ns. The inter-processor latency is under 140 ns.

The SGI/Origin 3800 (http://www.nsc.liu.se/sgi3k/recent.html) is an older system constructed on the basis of MIPS R14000/500 processors. They are interconnected into 4 processors SMP nodes, which form 3-D cubes supernodes. SGI/Origin 3800 is ccNUMA machine with virtual shared memory and with highly optimized distant memory calls. They must be optimized due to the calls to the slow, distant memory (located on distant nodes) are the main source of the overheads on ccNUMA systems. The latency of L2cache-memory is 180 ns for a local memory access and 290–440 ns for distant calls. It is longer than for Regatta, which is only 178 ns, however, communication bandwidth on 32-way node must be shared between 32 processors instead of 4 on an internal node of SGI/Origin machine.

TABLE I Basic parameters of SGI Origin 3800, IBM Regatta, SGI Altix systems and Opteron 240 motherboard

	SGI/Origin 3800	*MIPS R14000 500 MHz*	*IBM Regatta pSeries 690 Turbo*	*Power4 1300 MHz*	*SGI Altix 3000 (Linux cluster)*	*Itanium®2 1300 MHz*	*Opteron 240 motherboard 1400 MHz*
No. of processors	128	1	48	1	48	1	2
Nodes	32	–	4 (MCMs)	–	–	–	–
Memory per node	8 GB (4 proc)	–	8 GB (8 proc.)	–	2 GB (1 proc.)	–	4 GB
No. on-chip proc.	–	1	–	2 (cores)	–	1	1
Architecture	ccNUMA	RISC 64	SMP	RISC 64	GSM (Global Shared Memory Cluster)	EPIC	AMD64
Cache on chip	–	L1 (32/32 kB) L2 (8 MB)	–	L1 (32/64 kB) L2 (1.44 MB) L3 (32 MB) L3 is off-chip L2, L3 shared by 2 cores	–	L1 (32/32) L2 (256 kB) L3 (3 MB)	L1 (64/64) L2 (1 MB)
CPU-Memory bandwidth	–	3.2 GB/s	–	13,86 GB/s (L3-Main Memory) shared by 2 cores	–	6.4 GB/s	6.4 GB/s
Memory latency	180 ns (close) 290–440 ns (distant) closest memory shared by only 4 processors	–	178 ns processor-memory communication bandwidth shared by 32 processors	–	50 ns SGI NUMAlink interconnect	–	65 ns Interprocessor latency 140 ns

TABLE II CPU speed and throughput on SGI Origin 3800, IBM Regatta, SGI Altix systems and Opteron 240 motherboard (http://www.specbench.org/cpu2000/results/)

	SGI/Origin 3800 (throughput)	*MIPS R14000/500 500 MHz (speed)*	*IBM Regatta pSeries 690 Turbo (throughput)*	*Power4 1300 MHz (speed)*	*SGI Altix 3000 (throughput)*	*Itanium® 2 1300 MHz (speed)*	*Opteron 240 motherboard 1400 MHz (speed)*
No. of processors	128	1	32	1	32	1	2
Peak performance (R_{peak})	128 Gflop/s	1 Gflop/s	83.2 Gflop/s	2,6 Gflop/s per core (5.2 Gflop/s per chip)	166,4 Gflop/s	5,2 Gflop/s	2.8 Gflop/s (per 1 processor)
LINPACK $n = 100$	n/a	427 Mflop/s	n/a	1070 Mflop/s (average)	n/a	3800 Mflop/s (best, R_{max})	2100 Mflop/s
SPEC CFP2000	532–570	436–463	251–260	1202–1266	541	1770–1780	1012
SPECCINT2000	582–605	410–427	232–249	804–839	311	875	933

Basic parameters for the four systems are presented in Table I.

The sustained performance of SPEC CPU2000 per single processor for the IBM Regatta system is 1.8 (for multitask tests—throughput) to 2.8 (for single processor and single task tests—speed) faster than for a single R14000/500 of SGI/Origin 3800 (www.specbench.org). The first number means that for intensive communication and multitasking a 32-way turbo node (2 processors on chip) has too slow a memory bandwidth, especially for benchmarks with highly localized data structures, for which NUMA architecture is better suited. However, for a 16-way IBM pSeries 690 HPC (16 processors, 1 processor on chip) with greater memory bandwidth the throughput is higher and is 2.6 times greater per processor performance than the throughput for SGI/Origin. We note that this ratio is smaller than the ratio between the frequencies of the 2 processors (2.8). Despite the same frequency, due to more efficient architectural issues (fast shared global memory) SGI Altix is almost 2 times faster than IBM Regatta (in throughput) while a single Itanium®2 is 50% faster than Power4 (in speed). The benchmarks from Table II show that the AMD Opteron is slower both than Power4 and Itanium®2. However, having 3 times shorter memory latency than the Power4 and almost the same as Itanium®2 but 4 times larger cache, the Opteron processor can beat both processors for problems with irregular geometry, which require a high memory load.

PARTICLE MODEL OF BLOOD FLOW IN CAPILLARY VESSELS

The equations of macroscopic fluid dynamics for blood [6–8] describe the motion and energetics of homogeneous fluid over a lengthscale over 1 mm. Conversely, mesoscopic flows of heterogeneous fluid are modeled in spatial scales of 10–200 μm [9–11].

As we have shown in Refs. [10,11], the mesoscopic blood system can be composed of plasma represented by fluid particles carrying other blood constituents. The capillary walls and blood components, i.e. fibrins and red blood cells, are made of "solid" particles. In Fig. 2, we show the particle system representing capillary wall.

We consider here a three-dimensional system, which consists of M both fluid and "solid" particles. The particles are represented by their centers of mass, which posses several attributes, as mass, position, translational and angular velocities and type. The particles of various types are distributed within the simulation domain according to a given scenario. Particles interact with each other by different forces depending on their physical characteristics.

Fluid particles interact with forces consisting of a sum of conservative force $\mathbf{F}^{C}$, two dissipative components $\mathbf{F}^{T}$ and $\mathbf{F}^{R}$ and the Brownian force $\tilde{\mathbf{F}}$ [12], that is

$$\mathbf{F}_{ij} = \mathbf{F}^{C}_{ij} + \mathbf{F}^{T}_{ij} + \mathbf{F}^{R}_{ij} + \tilde{\mathbf{F}}_{ij}. \quad (1)$$

The conservative and dissipative forces are the functions of distance between particles r_{ij} and their relative translational and angular velocities. The Brownian force is of a stochastic nature, which is defined by the symmetric, antisymmeric and trace diagonal random matrices with independent Wiener increments [12]. The forces are short-ranged, i.e. there exists a cut-off radius r_{cut}, which defines the range of FPM particle interactions. For $r_{ij} > r_{cut}$, $\mathbf{F}_{ij} = 0$.

Fluid particles interact with "solid" particles representing wall, red blood cells and fibrins with dissipative particle dynamics (DPD) force [13]. The "solid" particles making up the various system components such as, capillary walls, red blood cell and fibrin, interact with neighboring particles of the same type with harmonic forces (for details see Ref. [11]) to mimic elastic properties of the wall and red blood cells. The particles from isolated

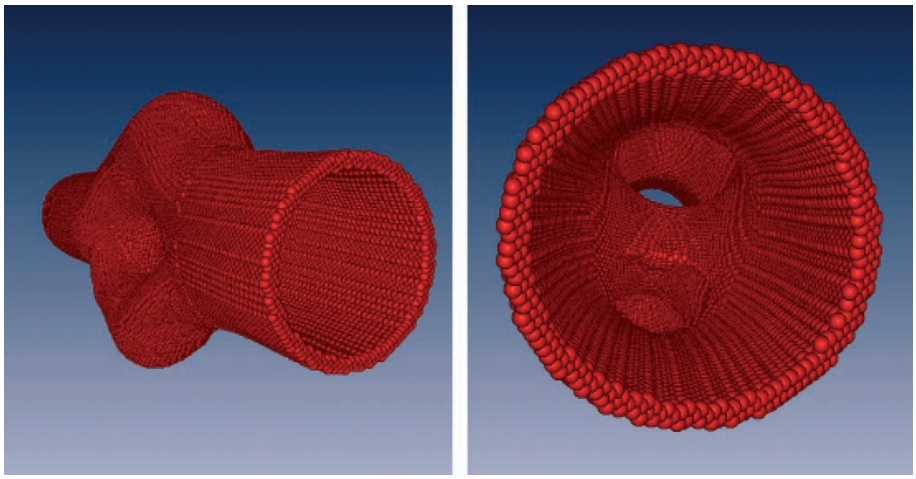

FIGURE 2 Bifurcating capillary vessels. The particles represent endothelial cells. They interact with one another via harmonic force to mimic the physics of elastic properties of the walls.

constituents (e.g. wall and red blood cell particles, particles from different red blood cells) interact with each other via FPM force with a strongly repulsive conservative component. In Fig. 3, we display the heterogeneous particle system, which produces a clot of red blood cells entangled by fibrinogen, one of the most important ingredients in blood clotting. Therapies to remove clots both through enzymatic and mechanical approaches require consideration of the biochemistry and structure of blood clots in conjunction with local transport phenomena [14].

As shown in Refs. [10,11], accurate 3D simulations of blood flowing in capillary vessels require a large ensemble of particles exceeding two million. For calculating interparticle forces we have used standard approach described in Ref. [15]. As shown in Fig. 4, the box is divided onto cubic cells of the edge size equal to cut-off radius. The force on a given particle includes contribution from all the particles that are closer than the value of cut-off radius and which are located within the cell containing the given particle or within the adjacent cell. Then the forces are computed by using an O(M) order *link-list* scheme [15].

The temporal evolution of the entire particle system is described by the Newtonian equations of motion. Because of the random Brownian forces, the equations of motion are now stochastic differential equations (SDE), which are computationally more difficult than ordinary differential equations. Numerical integration of SDE by using classical Verlet scheme [15] generates large numerical errors and artifacts, e.g. resulting in unacceptable temperature drift with simulation time [16]. Therefore, very tiny timesteps must be used to obtain a reasonable approximation to the thermodynamical quantities. In contrast to DPD [13], fluid-particle model the angular velocities and torques for each fluid particle have to be computed. More complex functional dependences of forces formula produce additional overhead. We note that the forces and torques in FPM depend not only on the particle positions and its translational velocity but also on the angular velocities. The predictor–corrector numerical schemes must be used twice per timestep.

In parallelizing the code, we have employed the domain-decomposition procedure [17,18]. The total volume of the computational periodic box is divided into P overlapping subsystems of equal volume (Fig. 5). Each subsystem is assigned to a single processor in a P processors array. The processors follow an identical predetermined sequence of

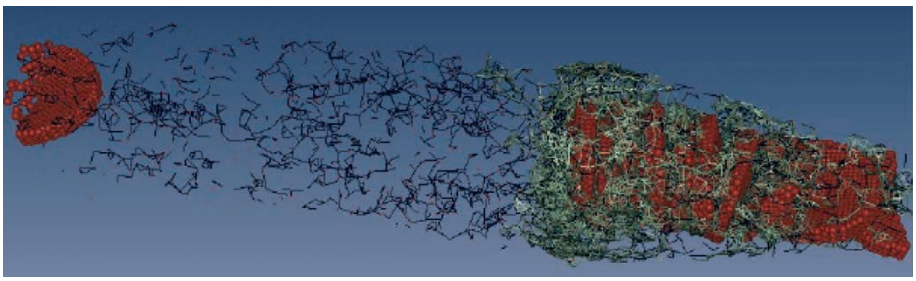

FIGURE 3 This snapshot shows the formation of a clot consisting of the red blood cells (red plates) and fibrins close to the capillary neck placed on the right hand side of the figure. The fibrinogen, which is responsible for clot formation concentrates mainly in the stagnation points of the flow, producing fibrins entangling with the red blood cells. The fluid and wall particles are not shown in this figure.

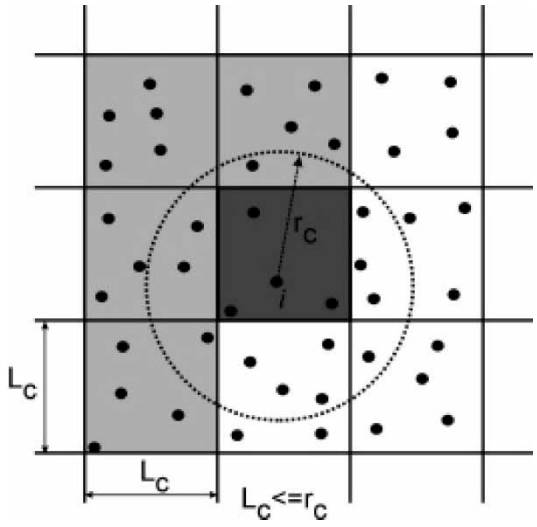

FIGURE 4 The linked-cell algorithm in 2D. In 3D the number of neighboring cells is greater but the idea is the same.

operations in calculating the forces associated with the particles within assigned domain. From cells, which are situated on the boundaries between processor domains, we copy all the particle attributes to the neighboring processor. The number of particles located in the boundary cells defines the communication overhead. Slicing the box along the z-axis simplifies greatly the routing of messages. It allows also sending them in unimpeded fashion. Each processor sends the message only along one direction to its closest neighbor. This domain decomposition algorithm also matches the data parallel programming paradigm used on shared memory systems. The memory blocks containing the arrays and other data structures from neighboring domains can be also placed close to each other in the main memory address space. The memory cells with the physical attributes of the particles located close to the domain borders, can be defined as being shared with read and write permissions.

In the case of the red blood cells flowing in a capillary (as shown in Figs. 3 and 5), equal box sections will contain different number of particles. Thus the slowest processor dealing with the largest number of particles will dictate the code efficiency. As shown in Fig. 5b, the load-balancing in this case is accomplished by shifting the boundaries of processor domains, while keeping the same number of particles in each section of the computational box [19–21]. From the point of data-parallel shared memory programming, this will correspond to changing write and read permissions for particles moving from one domain to the other.

We show the flowchart of the load-balancing in Fig. 6. The cell size (see Fig. 4) defines the smallest spatial step in which one can shift the boundaries between the processor domains. The CPU times $t_{\mathrm{li}}$from each process l (where $l = 1, \ldots, \mathrm{P}$) are collected during subsequent timesteps of simulation and averages t_l are computed. The differences $\Delta t_l = \mathbf{t} - t_l$ where $\mathbf{t} = 1/\mathrm{P} \times \sum_{i=1,\mathrm{P}} t_1$ govern the degree of load imbalance. Because of this problem, we must set up the new borders between the processor domains.

Discrete particle model code was written in Fortran95. Our programs run on shared memory platforms and the small number of required processors. The efficient use of SPMD (single program multiple data) data-parallel programming paradigm with Pthreads, OpenMP or high performance Fortran (HPF) [22] for particle codes is not trivial. For unbalanced systems it causes poor performance [23–25]. The problems with thread scheduling appear for many nested branch instructions. Moreover, using message passing interface (MPI) library gives a better portability to the code. However, we understand that when designing a message passing algorithm, the writer assumes a distributed memory model. Replication of data is necessary. Genuine shared memory programming

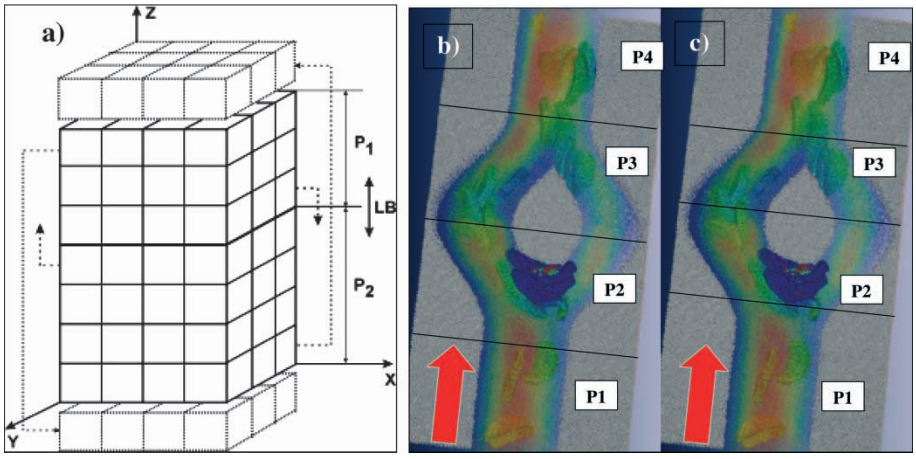

FIGURE 5 (a) The schematics of the geometrical decomposition of the computational box. (b,c) The decomposition of the computational boxes for red blood cells flow in a curved capillary without (b) and with (a) load-balancing.

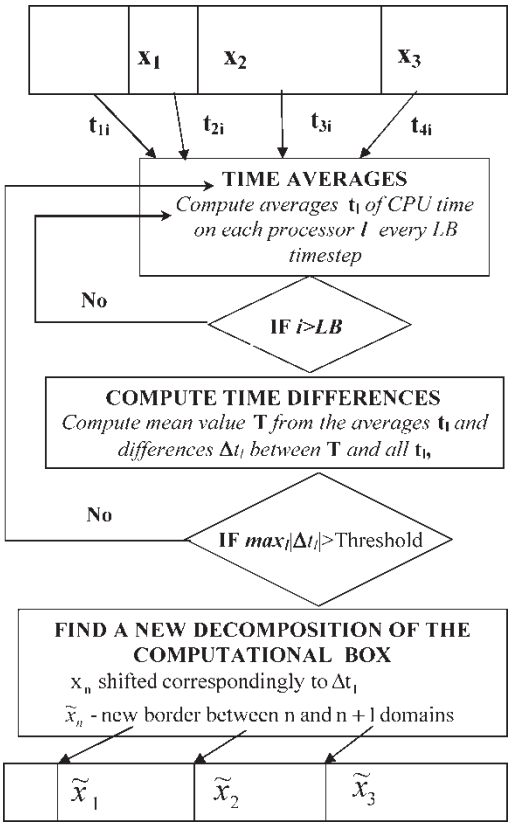

FIGURE 6 The flowchart of the load-balancing scheme. The mean execution CPU times t_l in a given number of timesteps LB are calculated for processes corresponding to the computational domains delineated by $x_1 - x_3$ boundary. The differences between average execution times for all the processes **t** and the mean times for each process t_l are used to set up new $\tilde{x}_1 - \tilde{x}_3$ boundaries between the domains.

is totally different. It makes very different assumptions on the breakdown and distribution of data. Moreover, MPI library involves using too many subroutines with many parameters. Therefore, we have created an easy-to-use middleware library. As shown in Fig. 7, it uses the MPI procedures but simplifies the implementation of interprocessor communication and allows for instantaneous monitoring of parallel computations.

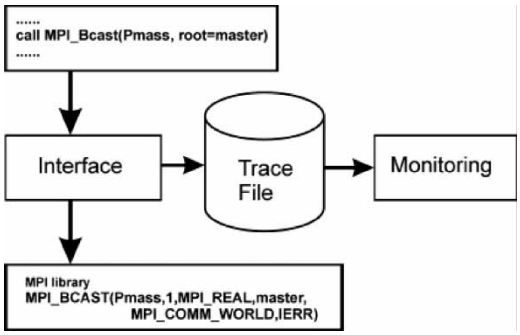

FIGURE 7 The communication interface library devised as an interface between original MPI library and the code. It simplifies greatly the calling of the message-passing procedures and creates a trace file of log type for monitoring the distributed processes.

TIMINGS AND TESTS ON SHARED MEMORY SYSTEMS

We have performed 4 and 8 processor simulations on the four systems of the same model of blood flow along bifurcating capillary vessel for $M = 2 \times 10^6$ particles (see Fig. 5b,c). We have activated an aggressive −O3 option for the compilers. Examples of the timings from these simulations are displayed in Figs. 8–10. The average sphere of a given cut-off radius includes 15 fluid particles.

In Figs. 8–10, we present the timings for the code running on 4 and 8 processors of the SGI/Origin 3800, IBM Regatta and SGI Altix 3000 computers. The plots from the left hand side deal with the case in which a computational box was decomposed uniformly on each processor (see Fig. 5a). We demonstrate that, the CPU time needed for completion the tasks on separate processors can differ even 4 times. The process of the longest execution time decides about the execution time of the entire code. To eliminate this deleterious effect, a load-balancing scheme must be implemented.

As shown in the plots on the right hand side of Figs. 8–10 the efficiency of the code with active load-balancing procedure increases twice over the average. Comparing the timings for various frequencies of invoking load-balancing scheme (for every 50th to 200th timestep) we cannot observe any difference on SGI Origin. This means there is a small communication overhead resulting from the load-balancing. It is somewhat greater for IBM Regatta than for SGI/Origin 3800, due to faster memory access to the data residing on the local memory of the NUMA system. We can observe large fluctuations of the code running on 4 processors of SGI Altix system (Fig. 10). Because they are not detected on 8 processors, we can conclude that this may be the effect of small cache of Itanium®2 (only 256 kB). For irregular data structure involved by particle codes frequent cache misses drastically reduce its performance comparing to Power4 and AMD Opteron processors. When the processor speeds are very different and depend non-linearly on the memory load, load-balancing becomes very difficult.

The execution time for sequential FPM code on SGI/Origin 3800 (L2 8MB cache size) increases substantially after about 60 timesteps (Fig. 11). This happens when the particles cross the cells boundaries and the attributes (position, velocities, forces and torques) of particles residing in the same cells (see Fig. 4) are located in distant memory addresses. This overhead is small for IBM Regatta (see Fig. 11), which is equipped with L3 32 MB cache (shared by 2 cores). It is completely invisible for AMD Opteron with adequately large secondary cache size and very fast access to the main memory (see Table I). Therefore, we have found that the Opteron reaches

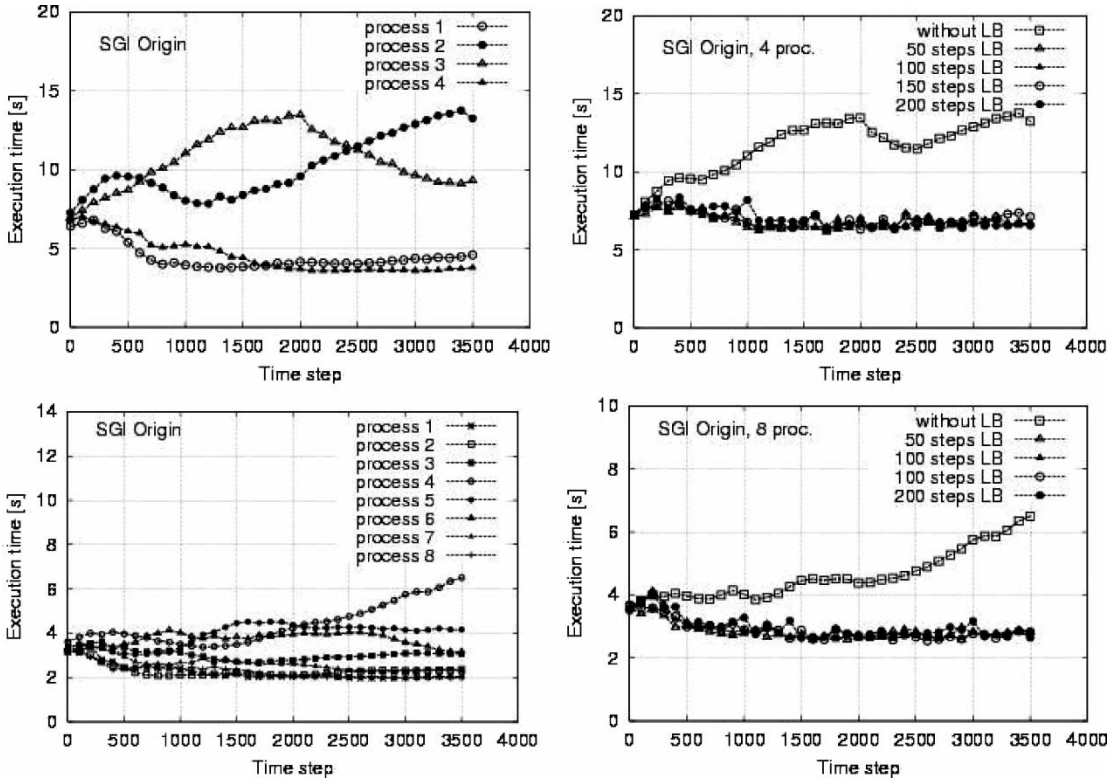

FIGURE 8 The timings for SGI/Origin 3800 machine with and without load-balancing.

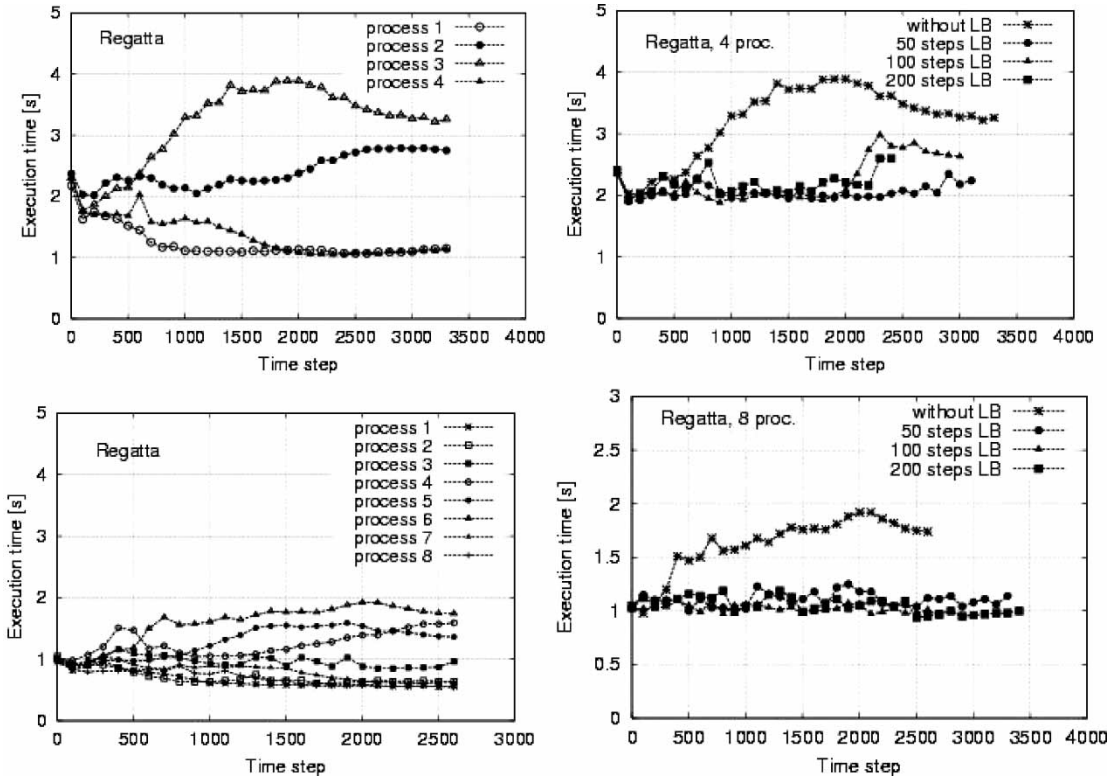

FIGURE 9 The timings for IBM Regatta machine with and without load-balancing.

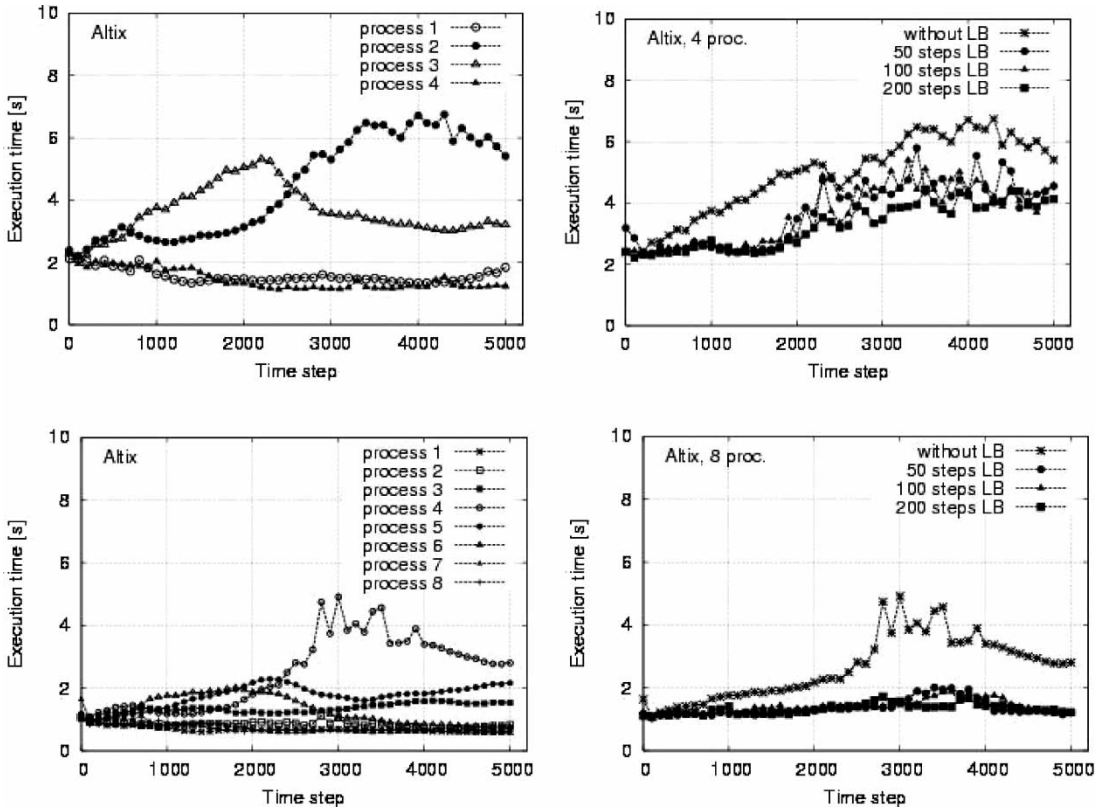

FIGURE 10 The timings for SGI Altix 3000 machine with and without load-balancing.

the fastest speed for a sequential FPM code. It is almost 2 times faster than one core of the Power4 processor and Itanium®2.

The neighboring particles should also be closer one another in the computer memory. Therefore, to avoid frequent cache misses the particles are renumbered every some period of time. Consequently, the particles in the same cell are classified with consecutive numbers. However, the gap between particle numbers still exists for the particles from neighboring cells used for computing the forces. This is due to the incoherence between the linear addressing of memory and indexing of 3D particle system. Increasing the number of processors for the same number of particles, we have narrowed the gap between memory addresses of neighboring cells. However, we can observe the super-linear speed-up resulting from a more efficient use of cache memory for a decreasing number of particles per processor. The maximum speed-ups obtained by

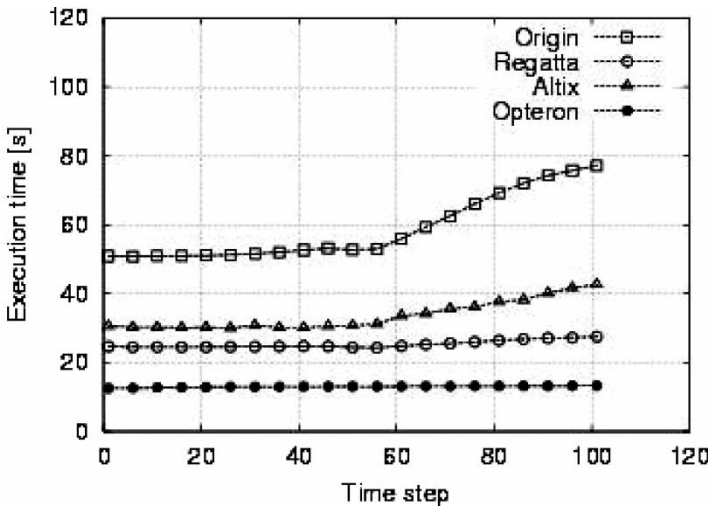

FIGURE 11 Execution time for sequential FPM code (about 2.5×10^5 fluid particles) running on SGI/Origin 3800 and IBM Regatta processors.

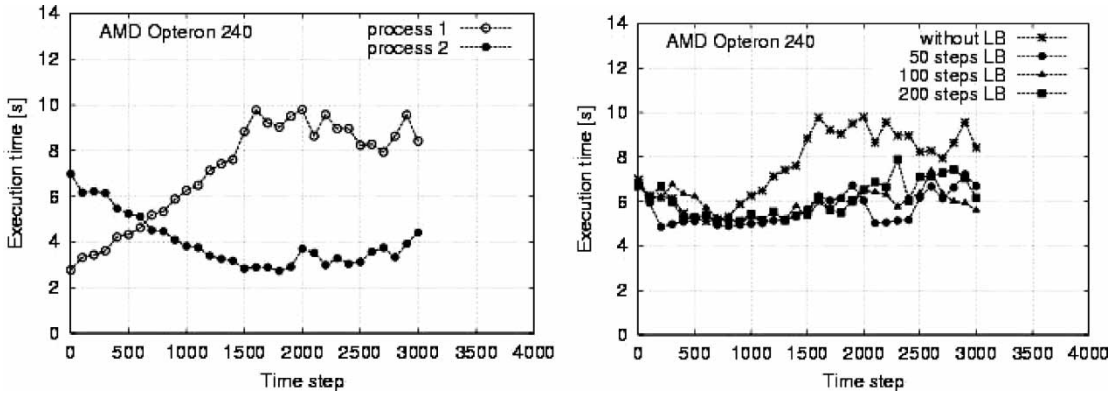

FIGURE 12 The timings for a 2-processor Opteron 240 motherboard with and without load-balancing.

increasing the number of processors from 4 to 8 (see Fig. 5) are 2.72, 2.42 and 2.27 for SGI/Origin 3800, IBM Regatta and SGI Altix, respectively.

In Figs. 8–12, we can see the positive feedback of large and shared L3 cache of Power4 IBM Regatta system and short memory latency and relatively large secondary cache of AMD Opteron on diminishing the effect of frequent cache misses. In sum, a single Power4 processor of IBM Regatta is 2.6 times faster in average than R14000/500 SGI/Origin processor. This result is in a good agreement with SPEC benchmarks reported in Table II. The SGI Altix is nearly 2 times faster than the IBM Regatta in SPEC throughput and 40% in SPEC processor speed (Table II). Nevertheless, we find that its measured performance for our discrete particle model is even worse off than for the IBM. Comparing the timings measured on four Itanium®2 processors (Fig. 10) and 2-processor Opteron 240 motherboard (Fig. 12), the SGI Altix is also slower per processor than AMD Opteron, which ranks the slowest in SPECFPM benchmarks from the three (see Table II). As shown in Fig. 11, the AMD Opteron is faster than the Power4 processor by a factor of two on a sequential code. However, for more processors, the IBM Regatta system uses more efficiently its large cache and appears to be the fastest.

In our opinion, discrete particles codes with irregular data structures and indirect addressing require processors with the shortest CPU-memory access, which could be reached by using large multilevel caches and/or short main memory latencies. Considering the timings of AMD Opteron from Figs. 11 and 12 and its low price/performance, we find more convincing this second issue. However, the cache size cannot be too small or too slow, as it is in Itanium®2 case. Many types of interactions involving many *if* statements almost inhibit the efficient use of powerful features of superscalar EPIC processors, such as VLIW (very long instruction word).

CONCLUDING REMARKS

Highly optimized parallel codes, running on 100 or more processors, rarely meet the common needs of the scientific community because of lack of availability and the queuing systems They are used for benchmarking high-performance systems or they represent special purpose solvers boosting up high-performance computing. This is not a general rule and many high-performance computing centers make their resources accessible through GRID infrastructure (see e.g. Refs. [26,27]). However, these services always will have very restricted access, and be very expensive and difficult to adapt for most client specified problems. Therefore, we suppose that the core of GRID computational services will be based on large amount of more universal solvers run on middle-ranged shared memory SMP machines and supported with user friendly interface and visualization facilities [28].

The problems involving multi-million particle ensembles, found in modeling mesoscopic phenomena, were considered only recently as the typical problems. Rapid increase of computational power of modern processors and growing popularity of coarse-grained discrete particle methods, such as DPD, FPM smoothed particle hydrodynamics and lattice Boltzmann gas, allow for the modeling of complex problems by using smaller shared-memory systems.

The main technical problem connected with efficient use of shared memory machines in modeling of large ensembles of discrete-particles lies in highly irregular data structure they require. We show that the computational efficiency can be considerably improved by using load-balancing algorithms. However, load-balancing can solve only the problems with irregular boundary conditions and domain decomposition. Many particle types and interaction forces involve using nesting *if* statements and/or indirect addressing. This strongly inhibits an efficient use of new architectural issues of modern processors, which

exploit explicit parallelism involving regularity of data structures and data independence.

We show that the Power4 processor with off-chip and a slow but large L3 cache can be more efficient than Itanium®2 with a modern EPIC architecture with fast access to the main memory. Moreover, the AMD Opteron with 65 ns memory latency and 1 MB L2 cache size can be 2 times faster than Power4 with 178 ns memory latency and 16 MB L3 cache and more than 2 times faster than Itanium®2 with 50 ns memory latency and slow 3 MB L3 cache. We conclude that mesoscopic particle codes based on DPD and fluid particle dynamics involving many types of particles should be executed preferably on the processor architectures with memories of the shortest latencies and reinforced with a sufficiently large secondary cache.

We have also demonstrated that, by employing simple parallel algorithms and communication schemes, we can easily adapt the discrete particle code for modeling irregular and heterogeneous problems, such as blood clotting in microscopic vessels of various shapes. We can run the same code without any changes on various shared memory servers. Flexibility, portability and efficiency of the particle model can be important in achieving of various GRID services, which will speed up mesoscopic modeling.

Acknowledgements

Support for this work has come from the Polish Committee for Scientific Research (KBN) project 4T11F02022 and the Complex Fluids Program of DOE.

References

[1] Boon, J.P. and Yip, S. (1991) *Molecular Hydrodynamics* (McGraw Hill, Dover).
[2] Vashishta, P., Kalia, R.K., Kodiyalam, S., Lidorikis, E., Nakano, A., Walsh, P., Bachlechner, M.E., Campbell, T.J., Ogata, S. and Shimojo, F. (2002) "Multimillion atom simulations of nanosystems on parallel computers", In: Ukawa, A., ed., *Proceedings of the International Symposium on Computational Science and Engineering 2002* (Japan Society for the Promotion of Science, Tokyo, Japan).
[3] Flekkoy, E.G. and Coveney, P.V. (1999) "From molecular dynamics to dissipative particle dynamics", *Phys. Rev. Lett.* **83**, 1775–1778.
[4] Boryczko, K., Dzwinel, W. and Yuen, D.A. (2002) "Parallel implementation of the fluid particle model for simulating complex fluids in the meso-scale", *Concurr. Comput. Pract. Exp.* **14**, 1–25.
[5] Nakano, A., Bachlechner, M.E., Kalia, R.K., Lidorikis, E., Vashishta, P., Voyiadjis, G.Z., Campbell, T.J., Ogata, S. and Shimojo, F. (2001) "Multiscale simulation of nanosystems", *IEEE Comput. Sci. Eng.* **3**(4), 56–66.
[6] Berger, S.A. (1992) In: Cheer, A.Y. and Van Dam, C.P., eds, *Flow in Large Blood Vessels, Fluid Dynamics in Biology,* Contemporary Mathematical Series (Am. Math. Soc., Providence), pp 479–518.
[7] Maury, B. (1999) "Direct simulations of 2D fluid-particle flows in biperiodic domains", *J. Comput. Phys.* **156**, 325–351.
[8] Quarteroni, A., Veneziani, A. and Zunino, P. (2002) "Mathematical and numerical modelling of solute dynamics in blood flow and arterial walls", *SIAM J. Numer. Anal.* **39/5**, 1488–1511.
[9] Dzwinel, W., Yuen, D.A. and Boryczko, K. (2002) "Mesoscopic dynamics of colloids simulated with dissipative particle dynamics and fluid particle methods", *J. Mol. Model.* **8**, 33–45.
[10] Dzwinel, W., Boryczko, K. and Yuen, D.A. (2003) "A discrete-particle model of blood dynamics in capillary vessels", *J. Colloid Int. Sci.* **258/1**, 163–173.
[11] Boryczko, K., Dzwinel, W. and Yuen, D.A. (2003) "Dynamical clustering of red blood cells in capillary vessels", *J. Mol. Model.* **9**, 16–33.
[12] Espanol, P. (1998) "Fluid particle model", *Phys. Rev. E* **57**, 2930–2948.
[13] Hoogerbrugge, P.J. and Koelman, J.M.V.A. (1992) "Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics", *Europhys. Lett.* **19**(3), 155–160.
[14] Diamond, S.L. (1999) "Engineering design of optimal strategies for blood clot dissolution", *Annu. Rev. Biomed. Eng.* **01**, 427–461.
[15] Hockney, R. and Eastwood, J.W. (1981) *Computer Simulation using Particles* (Mc Graw-Hill, New York).
[16] Vattulinen, I., Kartunen, M., Bsold, B. and Polson, J.M. (2002) "Integration schemes for dissipative particle dynamics simulations: from softly interacting systems towards models", *J. Chem. Phys.* **116**, 3967–3979.
[17] Fox, G. (1987) Domain Decomposition in Distributed and Shared Memory Environments. I: A Uniform Decomposition and Performance Analysis for the NCUBE and JPL Mark IIIfp Hypercubes. Supercomputing, 1st International Conference, Athens, Greece, June 8–12, 1987, Proceedings. *Lecture Notes in Computer Science* 297, pp. 1042–1073.
[18] Foster, I. (1995) *Designing and Building Parallel Programs* (Addison-Wesley Publishing Inc., Reading), p 377.
[19] Boryczko, K., Kitowski and Moscinski, J. (1994) *Load-balancing procedure for distributed shortrange Molecular Dynamics*. Lecture Notes in Computer Science 879, pp. 100–109.
[20] Hayashi, R. and Horiguchi, S. (2000) "Relationships between efficiency of dynamic load-balancing and particle concentration for parallel molecular dynamics simulation", *Proc. High Perform. Comput. Asia*, 976–983.
[21] Deng, Y., Peierls, R.F. and Rivera, C. (2000) "An adaptive load-balancing method for parallel molecular dynamics simulatons", *J. Comput. Phys.* **161**, 250–263.
[22] Briguglio, S., Di Martino, B. and Vlad, G. (2001) Workload decomposition for particle simulation applications on hierarchical distributed-shared memory parallel systems with integration of HPF and OpenMP. ICS 2001. *Proceedings of the 2001 International Conference on Supercomputing*, June 16–21, 2001, Sorrento, Napoli, Italy. ACM, 464.
[23] Boryczko, K., Kitowski, J. and Moscinski (1996) Efficiency comparison of data parallel programming and message passing, In: Liddell, H., Colbrook, A., Hertzberger, B. and Sloot, P., Eds, *Proc. Int. Conf. High Performance Computing and Networking*, Brussels, April 15–19, 1996, Lecture Notes in Computer Science **1067**, pp. 228–234.
[24] Thornley, J. (1998) *Molecular Dynamics Simulation on Shared-Memory Multiprocessors* (*ASCI Progress Report: October 1998*). ASCI Site Visit, Caltech, October 8–9.
[25] Thornley, J., Maria, Hui, Hao, Li, Tahir, Cagin and Goddard, W.A. III (1999) *Molecular Dynamics Simulation on Shared-Memory Multiprocessors with Lightweight Multithreading*. Advanced Simulation Technologies Conference (ASTC'99), High-Performance Computing Symposium (HPC'99), April 12, 1999.
[26] Foster, I. and Kesselman, C. (1998) *The Grid: Blueprint for a new Computing Infrastructure* (Morgan Kaufmann Publishers Inc., San Francisco).
[27] Berman, F., Fox, G., Hey, A.J. and Hey, A. (2003) *Grid Computing. Making the Global Infrastructure a Reality* (John Wiley and Sons Inc., New York).
[28] Garbow, Z.A., Yuen, D.A., Erlebacher, G., Bollig, E.F. and Kadlec, B.J. (2003) "Remote visualization and cluster analysis of 3-D geophysical data over the internet using off-screen rendering", *Visual Geosci.* **8**.