33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

Agent Based Modeling of Blood Coagulation System: Implementation Using a GPU Based High Speed Framework

Wenan Chen, Kevin Ward, Qi Li, Vojislav Kecman, Kayvan Najarian and Nathan Menke

Abstract—**The coagulation and fibrinolytic systems are complex, inter-connected biological systems with major physiological roles. The complex, nonlinear multi-point relationships between the molecular and cellular constituents of two systems render a comprehensive and simultaneous study of the system at the microscopic and macroscopic level a significant challenge. We have created an Agent Based Modeling and Simulation (ABMS) approach for simulating these complex interactions. As the scale of agents increase, the time complexity and cost of the resulting simulations presents a significant challenge. As such, in this paper, we also present a high-speed framework for the coagulation simulation utilizing the computing power of graphics processing units (GPU). For comparison, we also implemented the simulations in NetLogo, Repast, and a direct C version. As our experiments demonstrate, the computational speed of the GPU implementation of the million-level scale of agents is over 10 times faster versus the C version, over 100 times faster versus the Repast version and over 300 times faster versus the NetLogo simulation.**

I. Introduction

THE coagulation system (CS) is a complex, inter-connected biological system with major physiological and pathological roles. The CS may be viewed as a complex adaptive system, in which individual components are linked through multiple feedback and feedforward loops [1]. The non-linear relationships between the numerous coagulation factors and the interplay among the elements of the CS render the study of this biology at a molecular and cellular level nearly impossible [2].

The study presented in this paper applies an agent based modeling simulation (ABMS) to the analysis of the CS due to the potential ability to quantitatively analyze individual components of each system at every point of simulation. ABMS is a dynamic modeling and simulation tool that allows the study of complex non-linear networked systems. In particular, ABMS represents a non-reductionist approach to studying the biologic process, while retaining the information at an individual level [3]. The complexity of the CS has stymied experimental efforts to gain a system level understanding of the coagulation cascade and its subnetwork components. ABMS may readily provide elucidation of the pathophysiology of diseases related to the coagulation system. A model such as the one provided in the paper may prove informative regarding individual disease processes such as genetic and acquired disorders of coagulation. Unfortunately, ABMS is a computationally expensive process due to the large number of elements involved in such simulations (on the order of 10^6). Simulations take days to weeks to run on a desktop computer. Agent based modeling may be performed on existing frameworks. Two commonly used platforms for agent-based simulations, NetLogo and Repast, allow customized programming of user-specified agents interacting within a defined framework [4] [5]. NetLogo and Repast both provide platforms for defining the system and its interactions. However, both programming platforms face the challenge of decreased efficiency when presented with the millions of agents and interactions observed at the biological systems level.

Recently, graphic processing unit (GPU) has evolved into an alternative platform for high speed general purpose computing, and in particular biomedical computing, due to its massive parallel architecture and supporting library. Typically a GPU consists of hundreds of processor cores that operate together; thus allowing higher throughput on parallel tasks than a CPU. Lysenko & D'Souza demonstrate the GPU power using the low level GPGPU [6]. Richmond et al. illustrated approximately 80 times speedup through the use of the NVIDIA Compute Unified Device Architecture (CUDA) based general agent based model framework [7]. Such studies motivated our current GPU-based coagulation simulation.

This study presents an agent based modeling framework for blood coagulation, and implements this model based on the GPU via the CUDA library [8]. Besides, the speed gains are compared with three other implementations: NetLogo and Repast platforms, as well as a C program designed to run on a single processor. As far as we know, this is the first report of GPU implementation of an ABMS model for blood coagulation.

II. Methodology

A. Blood coagulation model

The ABMS in this paper uses a two dimensional particle system whereby particles move freely and interact on a

Wenan Chen is with the Department of Biostatistics, Virginia Commonwealth University (VCU), USA and VCU Reanimation Engineering Science Center (VCURES) (corresponding author, e-mail: chenw6@vcu.edu).

Kevin Ward is with the Department of Emergency Medicine, Virginia Commonwealth University and VCURES (e-mail: krward@vcu.edu).

Qi Li is with the Department of Computer Science, Virginia Commonwealth University (e-mail: liq@mymail.vcu.edu).

Vojislav Kecman is with the Department of Computer Science, Virginia Commonwealth University and VCURES (e-mail: vkecman@vcu.edu).

Kayvan Najarian is with the Department of Computer Science, Virginia Commonwealth University and VCURES (e-mail: knajarian@vcu.edu).

Nathan Menke is with the Department of Emergency Medicine, Virginia Commonwealth University and VCURES (e-mail: nbmenke@aol.com).

discrete spatial grid. In this specific model, we define the agents of the system as the reactants, enzymes, and products of the coagulation system. These include XI, XIa, XII, II, Antithrombin III (AT), AT-H complex, totally 62 different types. The modeling diagram is illustrated in Figure 1. The spatial grid is a two dimensional grid where the agent's location is identified by its x and y coordinates. Each coordinate pair (x, y) delineates a unique location. The spatial grid is in the shape of a 112 x 112 square allowing the agents to interact and bounce off the edge of the grid. Grid locations in this model are designated as either empty or occupied by one or more substrates, enzymes, or reaction products. The agents are allowed to move randomly with equal probability on each direction (8 directions if not on the border). The joining and breaking are governed by a rule table with probabilities. The joining parameter determines the extent of interaction between adjacent agents. The breaking parameter defines the extent of disruption of agents that have joined. This model sets the probability of joining and breaking based on experimentally determined kinetic constants. The agents are allowed to interact with all their neighbors, but meaningful interactions are limited to those in a rule table. The neighborhood of each agent in this model is defined as all agents located in the same grid location. After each time step, the agents can move in a random manner to an adjacent grid location.

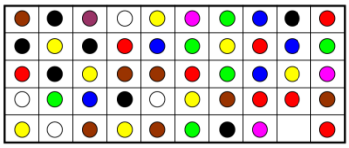

Figure 1. Agents in a grid with different colors as different types of agents.

ABMS modeling requires the assignment of the probability of conversion associated with each chemical reaction. Reductionist in vitro experimental techniques have allowed a detailed understanding of the individual chemical reactions involved in the process of coagulation. The information obtained by studying the individual reactions is used as the basis for the rules governing the updating of the ABMS at each time step. We assigned a probability of conversion value related to the kinetics of the reactions. The initial configuration is random; each substrate and enzyme is assigned a predefined number of agents based on their desired initial concentration.

The initial configuration of ABMS contained a pre-defined number of agents for the initial reactants, which is proportional to the known concentration of agent under normal physiologic conditions. These agents were randomly distributed on the 2-D grid at time t=0. At each time step t, the agents act independently with the two designed actions: move one unit randomly and perform reaction if possible. After the reaction, some new types of agent may be generated and some agent may die. One iteration cycle (time step) consists of performing the two actions for all existing agents in the grid. As the simulation goes, the counts of each type of agents will be updated. The termination condition of the simulation is determined by the formation of a virtual clot.

B. *GPU design overview*

The idea is to let each execution core in GPU work on a grid location. Since execution core cannot allocate the GPU memory by itself while executing the kernel code, all the data structures must be allocated and copied to the GPU before the GPU kernel execution. A major challenge of the simulation involves GPU memory management due to constant generation of new agents and destruction of old agents. Although there is an upper bound of the number of agents running simultaneously during simulation, the designed data structures must have the flexibility to integrate newly generated agents. In our solution we create a pre-allocated memory block capable of accommodating the largest possible number of living agents the simulation may reach. The memory space of dead agents will be collected for new born agents. In our simulation, a preset constant defines the maximal number of living agents which is set to about 2 million to allow million-level agents in the simulation. Thus, both the data structures for agents and the grid allow for the change of the number of agents during the simulation.

The actions of each agent consist of two steps. The first step is moving action. The second is a composite of verifying adequate conditions for the specified equation (rule from the rule table) followed by execution upon satisfactory verification. In the simulation, these two steps are executed separately in a batch mode. Specifically, in each cycle, all the agents perform the move action first. Then the reaction steps of all agents are executed.

C. *Agent data structure*

Each agent stores its coordinates, the agent type, its state and the storage position in the grid environment as shown in the following schematic diagram:

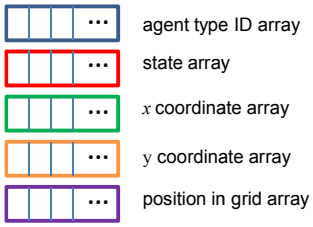

Figure 2. Data structure agentMatrix using struct of arrays.

The agentMatrix uses the struct of arrays instead of array of structs for coalesced memory accesses [7]. The agent IDs are the index in the array that the agents occupy; thus, in the simulation, new born agents may reuse the agent ID used by dead agents before. There is also a position array pos which is used to access the grid data structure quickly for the dynamic update of grid explained later.

D. *Grid data structure*

The grid structure provides quick access of agents on a certain location. After the agent's move, the system needs to

reflect the location change into the grid data structure. This is done as follows. At the beginning of the simulation, the agents' coordinates are scanned from the agentMatrix and the grid is built. The data structure of the grid is then updated during the simulation. In order to quickly look up the locations of all agents in the same grid cell, the grid data structure is organized based on grid cell coordinates and agent types. The grid data structure is described in Figure 3:

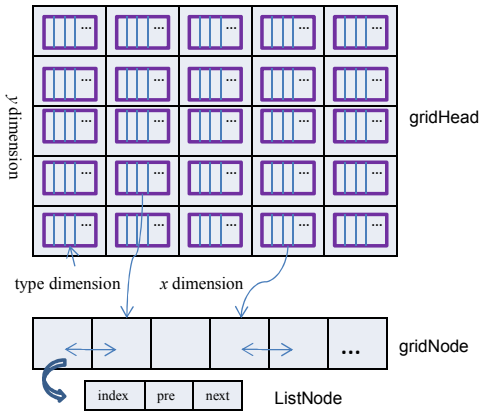

Figure 3. Grid data structure.

There are two important components in the grid structure: the head pointer of a 3 dimension array gridHead and the storage array gridNode. The first two dimensions of gridHead are the x and y coordinates, the third one is the agent type. In a general grid design, we may view each cell in the grid as a unit and put all the agents in a cell either in a linked list or an array. In this case, verification of the presence of the type of agent required to complete a reaction equation necessitates scanning the entirety (i.e. all agents) of a specific grid cell. The addition of another dimension to the grid, the agent type, allows for this verification through direct access of the gridHead. If there are some agents of this type, the head pointer points to the beginning of the linked list of these agents, called gridNode. gridNode is a set of one dimensional linked lists storing agent IDs of the same type agents on the same grid location. These agents are linked together into a two direction linked list. This design allows for rapid access to an agent of certain coordinates and type on the grid. Once an agent moves, the linked lists will be updated: inserting the agent from the previous list into a new list corresponding to the new location indexed by the new x, y coordinates and the agent type. In addition, the free nodes (including nodes of dead agents) of the storage array gridNode are also linked together into a linked list. Allocation of new ListNode to new agents or putting dead agent nodes into the free nodes' list provides dynamic node allocation to match the constant birth and destruction of agents in the simulation.

E. Agent birth and death

The death of an agent is easier to handle than the birth of an agent. The death of an agent is managed solely through marking its state and removing the agent from the grid. As each reaction step is executed independently on each grid cell, grid removal does require access to the grid as a whole. Each CUDA thread executes only on its own sets of grid cells.

In contrast, agent births are organized in a batch mode. During each simulation cycle, the dead agents leave space for new agents. Execution of a single thread by a batch operation places all newborn agents into spaces previously occupied by dead agents. If the number of newborn agents exceeds the space created by dead agents, newborn agents are placed in a buffered free space until they reach the maximal number of agents. Typically each cycle has several hundred reactions executed, thus handling new born agents does not need GPU parallel processing for speed. The batch operation for new born agents after each cycle reuses the memory of dead agents and therefore achieves dynamic memory allocation during simulation.

F. Evaluation of implementations based on different platforms

In order to compare the efficiency between our CUDA version and others, we also implemented the same simulation in three other platforms: a NetLogo based implementation, a Repast (Simphony) based implementation and a direct C implementation. NetLogo is set without graphic output in order to maximize the speed. The Repast version runs in the batch mode without graphic updates. The C version is similar to the CUDA version except for the parallel kernel execution of agents' moves and reactions in CUDA. The speed is measured as the average time per cycle and all speedups are compared with the NetLogo version.

III. Results

A. Simulation setting

The number of initial agents is 886, 915 and the number of agents remains above 800,000 during all the simulations. Thus the population size is on the order of one million agents in the simulations. To illustrate the advantage of CUDA version on large scale simulations, we also compare the speed when the initial population size is reduced to 10% and 1% of our standard number of agents. These numbers corresponds to the level of 100k, 10k of population size. The CUDA version runs on a workstation with Tesla C1060 GPU, which has 240 cores. Other versions run on a desktop with Intel Xeon 2.27G Hz CPU, 6G RAM.

B. Verification of different implementations

Because of the stochastic nature of agent based systems, the outputs of each simulation are unique. In order to verify the equivalence between the multiple simulation types, we run multiple simulations for each implementation. We compare the iteration cycles before termination (enough coagulation formed) among the four different implementations using ANOVA. A p-value 0.65 supports that there is no significant difference.

C. Speed comparison

From Table I, one can see that, in a million-level simulation, the speedup of the CUDA version is over 200

times versus the NetLogo version, over 100 times versus Repast version. The speed improvement is due to the high speed parallel CUDA execution as well as the special design of the data structures for this application. An interesting trend in the table is that as the number of agents increase, the speedups of both the C version and CUDA version increases. Thus, the designed framework scales much better than the versions based on the Repast or NetLogo. The economy of scale is built into the design of the grid structure as described previously in the methods section. We also observe that the CUDA version is over 10 times versus C implementation in the million-agent level. With the exception of GPU execution, the coding is almost identical; therefore, this speedup can be thought as purely from the high speed advantage of GPU execution versus CPU execution. We also note that the C version is slower than the NetLogo or Repast version when only 1% agents are simulated. In these smaller scale models, the speed benefits are outweighted by the cost of the dynamic updates of the specialized grid data structure. This suggests that the proposed framework is beneficial primarily for large scale agent based modeling.

Table I. SPEED COMPARISON OF DIFFERENT IMPLEMENTATIONS

# of Agents		NetLogo	Repast	C	CUDA
886,915	Time/cycle	4740 ms	2310 ms	276 ms	20.1 ms
(100%)	Speedup	1	~2	~17	~233
88,693	Time/cycle	210 ms	142 ms	43 ms	7.0 ms
(10%)	Speedup	1	~1.5	~4.8	~30
8,872	Time/cycle	15 ms	11 ms	18.8 ms	6.1 ms
(1%)	Speedup	1	~1.3	~0.8	~2.5

* Abbreviations: h - hour, m – minute ms - millisecond.

IV. DISCUSSION

The speedup in mega scale is critical for large scale agent based simulations, as seen in biological modeling. In order to determine whether we have fully explored the GPU power, we check our speed gain with other related work, although the overall speed gains are highly application dependent. In [6], the speedup of GPU based simulation versus Repast is about 125 times. Our model demonstrated a speedup of about 100 times. In [9], a speedup of approximately 70 times is reported versus a Java version. These results are generally consistent with our comparison results between Repast and CUDA. Richmond et al. compared the CUDA implementation versus its original version of a general framework FLAME used for agent based simulation. The reported speedup is around 80 times [7]. It seems to be due to the fact that the optimized GPU version may provide higher speed gains in more complex agent based system, i.e., the more complex the system is, the more probable a higher speedup using CUDA. This can be seen from the table in Richmond et al.'s paper, for the population size of 131072, GPU takes 5163 ms while CPU takes 463,310 ms for one iteration cycle. In our case, the time of each cycle in mega level is less than 300ms in the C version. A recent study from Intel shows that the speedup of tested applications between GPU and CPU are up to 14 times [10]. This is consistent with our comparison between CUDA and C version. The speed of the CUDA version (about 20ms/cycle) offers the potential for a smooth dynamic display of agents on the grid (more than 24 updates per second). Such updates are the topic for future efforts that will bring computational modeling closer to clinical application. The emerging multi-GPU technologies may also help reduce the computational modeling time furthermore (NVIDIA CUDA 4.0 offers new features to simplify multi-GPU programming).

V. CONCLUSION

The paper presents an agent based modeling and simulation approach for simulating complex interactions involved in blood coagulation system. The paper also presents a high-speed framework for the coagulation simulation that utilizes the computing power of GPUs. Results indicate that our complex models of human CS can be implemented using GPU based computing with significant performance gains. The GPU based coagulation modeling makes it possible for future applications including discovery of new mediators, understanding of proximal and distal effects of interactions between systems, discovery of new diagnostic and therapeutic options. Future directions include optimization for speed improvements. Increasing the number of agents by a factor of ten is another goal of the modeling project. It is anticipated that at this number (10^7) of agents, CUDA will be crucial for creating biological meaningful simulations that run in a reasonable amount of time.

REFERENCES

[1] Monroe, D.M. and M. Hoffman, What Does It Take to Make the Perfect Clot? *Arterioscler. Thromb. Vasc. Biol.,* 26(1). 41-48, 2006.

[2] Zhu, D., Mathematical modeling of blood coagulation cascade: kinetics of intrinsic and extrinsic pathways in normal and deficient conditions. *Blood Coagul Fibrinolysis*, 18(7). 637-46, 2007.

[3] Bonabeau, E., Agent-based modeling: methods and techniques for simulating human systems. *Proc Natl Acad Sci*, 99 Suppl 3. 7280-7. 2002.

[4] Wilensky, U. NetLogo. *http://ccl.northwestern.edu /netlogo/.* Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. 1999

[5] Repast. Repast Organization for Architecture and Design, "Repast," Available at http://repast.sourceforge.net/. 2008.

[6] Lysenko, M., D'Souza, R. M. "A framework for mega-scale agent-based model simulations on the GPU," *Journal of Artificial Societies and Social Simulation.* (JASSS). 2008.

[7] Richmond, P., Coakley, S., Romano, M. D. A high performance agent based modeling framework on graphics card hardware with CUDA. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems.* 2009.

[8] CUDA. http://www.nvidia.com/object/cuda_home_new.html.

[9] Strippgen, D., Nagel, K. "Using common graphics hardware for multi-agent traffic simulation with CUDA," *International Conference On Simulation Tools And Techniques For Communications, Networks And Systems & Workshops.* 2009.

[10] Lee, V. W., Kim, C., Chhugani, J., Deisher, M. Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R.and Dubey P. "Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu." *In ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, pp. 451–460, New York, NY, USA. ACM. 2010.