PAPER *Special Section on Parallel and Distributed Computing and Networking*

Simulating Cardiac Electrophysiology in the Era of GPU-Cluster Computing

Jun CHAI[†a)], **Mei WEN**[†], **Nan WU**[†], **Dafei HUANG**[†], **Jing YANG**[†], **Xing CAI**[††*], **Chunyuan ZHANG**[†], ***Nonmembers, and*** **Qianming YANG**[†], ***Student Member***

SUMMARY This paper presents a study of the applicability of clusters of GPUs to high-resolution 3D simulations of cardiac electrophysiology. By experimenting with representative cardiac cell models and ODE solvers, in association with solving the monodomain equation, we quantitatively analyze the obtainable computational capacity of GPU clusters. It is found that for a 501×501×101 3D mesh, which entails a 0.1mm spatial resolution, a 128-GPU cluster only needs a few minutes to carry out a 100,000-time-step cardiac excitation simulation that involves a four-variable cell model. Even higher spatial and temporal resolutions are achievable for such simplified mathematical models. On the other hand, our experiments also show that a dramatically larger cluster of GPUs is needed to handle a very detailed cardiac cell model.

key words: *GPU cluster, simulation of cardiac electrophysiology, high resolution, monodomain equation, cardiac cell model*

1. Introduction

Simulating the cardiac bioelectric activity at the tissue level can be a demanding computational task, due to a spatial resolution requirement down to 0.1 mm and a temporal resolution requirement down to 0.01 ms (or even below), see e.g. [7]. At the same time, sophisticated cardiac cell models can involve up to one hundred state variables, thus requiring robust and computationally intensive solvers for the related systems of ordinary differential equations (ODEs). If quick turnover time (say, a few hours) is desirable for simulating an entire heart cycle with the above computational details, use of very large parallel computers is mandatory. In [8], near-realtime monodomain simulation of human cardiac electrophysiology (within one minute) was achieved, leading to a scalable parallelization on up to 16,384 CPU cores.

With the relatively recent advent of general-purpose GPUs, which can deliver massive computational power, many subjects in computational science have started to adopt this hardware technology. Computational cardiology is no exception for GPU computing. The monodomain equation and ODEs of cardiac tissues were solved on a single GPU in [10], running 20 times faster than a quad-core CPU for 2D cardiac simulations. Bartocci et al. [2] showed that through careful and model-specific optimizations, 2D/3D simulations involving realistic and detailed cardiac cell models now can be performed in times that are close to real time using one GPU. Previous studies have also investigated for cardiac simulations using a small number of GPUs. The cardiac monodomain simulations in [12] demonstrated the speedup of a 3D code with four GPUs by a factor of 1.6 compared with 32 CPU cores. Vigmond et al. [14] showed that solving a set of non-linear ODEs with the cardiac monodomain model on four GPUs could be sped up by a factor in the range of 9–17 compared with four CPU cores. In [9], the cardiac bidomain model was ported to a four-GPU platform, where effective architecture-specific optimizations of the underlying explicit numerical methods, together with fine grained parallelization strategies, gave a drastic improvement (a factor of 2460) of a 3D simulation, compared with using a single CPU core. In [6], by using a state-of-the-art model of rabbit ventricles, the benefits and scalability of cardiac bidomain simulations running on 6–20 GPUs were demonstrated. The most recent work in [1] parallelized a 2D monodomain model using a hybrid CPU-GPU approach, and showed that the hybrid implementation (using 64 CPU cores and 16 GPUs) was nearly 7 times faster than the CPU-only implementation (using 64 CPU cores) when running on an 8-node cluster.

To the best of our knowledge, however, there exists no quantitative study of the performance of GPU clusters with more than 20 GPUs, when applied to cardiac electrophysiology. This paper thus aims to investigate this topic by looking at several relevant issues about programming and performance tuning. The purpose is to provide the readers with a realistic expectation of achievable cardiac dynamics simulations on large clusters with more than 100 GPUs.

In this paper, we start with explaining the mathematical model and numerical strategies. Then our parallel implementation is described with details of CUDA programming. In the subsequent numerical experiments, we analyze the performance of three types of parallel simulations that have run on up to a $2001 \times 2001 \times 401$ spatial mesh and used up to 128 GPUs.

2. Mathematical Model

The propagation of bioelectric activity in the heart can be mathematically described by either the monodomain equa-

Manuscript received December 26, 2012.
Manuscript revised June 7, 2013.

[†]The authors are with School of Computer, National University of Defense Technology, 410073, Changsha, P. R. China.
[††]The author is with the Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, NO-0316 Oslo, Norway.
[*]Presently, with Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway.
a) E-mail: chaijun200306@nudt.edu.cn

DOI: 10.1587/transinf.E96.D.2587

tion or the bidomain equations. As argued in [4], these two mathematical models can produce almost identical propagation sequences, proivded that no current is injected into the extracellular space. We therefore adopt for this paper the monodomain model as the underlying partial differential equation (PDE).

More specifically, the monodomain equation is a PDE of the following form:

$$\frac{\partial v}{\partial t} = \nabla \cdot (\sigma \nabla v) + f(v, \mathbf{s}, t), \tag{1}$$

where v denotes the transmembrane potential, which is the primary unknown of the monodomain model. The conductivity tensor is denoted by σ. Moreover, the known f function is related to the total ionic current, which depends on v and $\mathbf{s}$, with the latter denoting a set of state variables.

The dynamic relationship between v and $\mathbf{s}$ is further described by a system of ODEs, which give a mathematical description of the bioelectric properties of cardiac cell tissues. For this paper, we will adopt two cell models, one developed by Bueno-Orovio et al. [3] and the other by Grandi et al. [5]. The first cell model has four state variables (including v), whereas the latter has 39 state variables (including v). For simplicity, we will refer to these two cell models, respectively, as the Bueno model and the Grandi model throughout the following text. The rationale behind choosing these two cell models is that they span a reasonable spectrum of mathematical (and numerical) complexity, with the Bueno model being the simplest model for describing human ventricular action potentials, whereas the Grandi model is very detailed.

3. Numerical Strategies

A standard splitting technique is applied to the monodmain Eq. (1), with the purpose of separating the numerical treatment of the diffusion term $\nabla \cdot (\sigma \nabla v)$ and the reaction term $f(v, \mathbf{s}, t)$. The latter is incorporated into the system of ODEs. Finite differences and forward-Euler temporal discretization are applied to the purely diffusive PDE arised from splitting. More specifically, during each time-step, the following formula is used to first compute an intermediate v solution:

$$\begin{aligned}\tilde{v}^{\ell}_{i,j,k} = v^{\ell}_{i,j,k} &+ \frac{\sigma_x \Delta t}{\Delta x^2}\left(v^{\ell}_{i-1,j,k} - 2v^{\ell}_{i,j,k} + v^{\ell}_{i+1,j,k}\right)\\ &+ \frac{\sigma_y \Delta t}{\Delta y^2}\left(v^{\ell}_{i,j-1,k} - 2v^{\ell}_{i,j,k} + v^{\ell}_{i,j+1,k}\right)\\ &+ \frac{\sigma_z \Delta t}{\Delta z^2}\left(v^{\ell}_{i,j,k-1} - 2v^{\ell}_{i,j,k} + v^{\ell}_{i,j,k+1}\right),\end{aligned} \tag{2}$$

which is then used as the initial v condition for solving the system of ODEs from $t = \ell\Delta t$ until $t = (\ell+1)\Delta t$. Note, in the above formula, the superscript ℓ is the index for the temporal direction, whereas the subscript (i, j, k) contains the spatial mesh point indices. Moreover, we have also assumed in the above formula that the conductivity tensor σ has only three nonzero constant diagnoal values.

When solving the involved system of ODEs, we have adopted two strategies. One is the very simple forward-Euler method, the other is a generalized 2nd-order extension of the Rush–Larsen method [11]. The generalized Rush–Larsen solver [13], denoted GRL2, is both more stable and accurate than the forward-Euler solver. However, with respect to the computational cost, the GRL2 solver is $3m$ times more expensive than the simple forward-Euler solver, where m denotes the number of state variables involved.

```
/*initial conditions*/
Input U_PREV and other state variable arrays;

/*time-stepping computing*/
t = 0;
While(t <T)
  /*task 1: compute the PDE part*/
  For each k,j,i, do
    U[k][j][i] = 7-point stencil applied to U_PREV[k][j][i];
  End for

  /*task 2: compute the ODE part*/
  For each k,j,i, do
    Call ODE model solver to further update U[k][j][i] and other
    state variables
  End for

  /*task 3: enforce boundarycondition*/
  ...

  /* pointer switch between U and U_PREV*/
  ...

  t += dt;
End while
```

Fig. 1 An algorithmic overview of the PDE-ODE numerical scheme.

The main computational work of this PDE-ODE numerical scheme is a time-stepping procedure, which is the while-loop shown in Fig. 1. As the data structure, there are totally 5 or 40 three-dimensional arrays, of which the key arrays are U and U_PREV, representing the v solution on two consecutive time levels. There are three tasks per time-step. First, a seven-point stencil is applied to array U_PREV as the PDE part. Second, either forward-Euler or GRL2 works as the ODE solver, to update all the state variable arrays (including array U). At last, the surrounding ghost-point layer of array U is updated for the enforcement of zero-flux boundary conditions.

4. Parallel Implementation

There is massive parallelism in the PDE-ODE numerical scheme, suiting very well for parallel computing. Data exchange is only needed between the nearest neighbor pairs, in connection with solving the PDE part.

4.1 Using Multiple GPUs

The entire 3D spatial domain is divided into box-shaped subdomains, matching the available GPUs. MPI is used

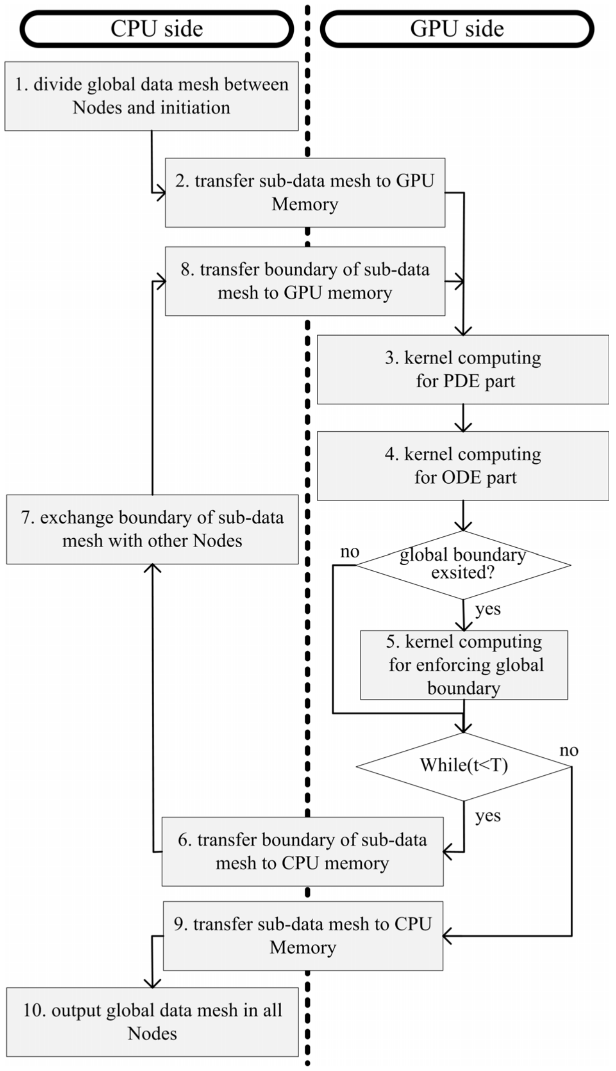

CPU side
GPU side
1. divide global data mesh between Nodes and initiation
2. transfer sub-data mesh to GPU Memory
8. transfer boundary of sub-data mesh to GPU memory
3. kernel computing for PDE part
4. kernel computing for ODE part
7. exchange boundary of sub-data mesh with other Nodes
global boundary exsited?
no
yes
5. kernel computing for enforcing global boundary
While(t<T)
no
yes
6. transfer boundary of sub-data mesh to CPU memory
9. transfer sub-data mesh to CPU Memory
10. output global data mesh in all Nodes

Fig. 2 Flowchart of cardiac simulations on a GPU-enhanced cluster.

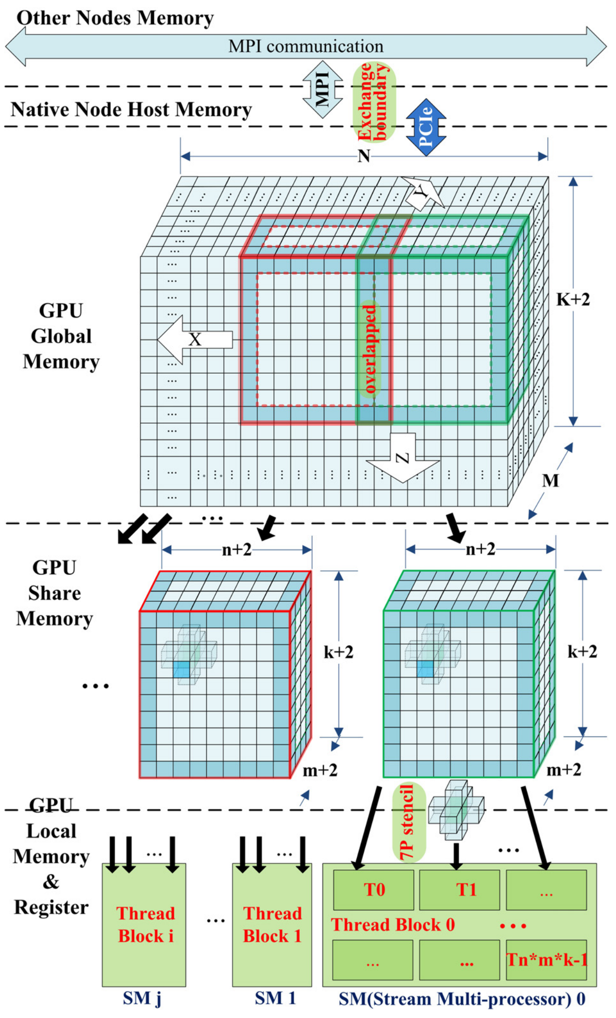


Fig. 3 Thread block granularity and use of GPU's memory hierarchy.

to enable the needed inter-subdomain data exchange. For the current hardware architectures, the GPUs have to go through their host CPUs for each MPI communication. Nevertheless, comparing with only using CPUs for the computations, the performance can be greatly enhanced on a GPU-enhanced cluster, by offloading the computations to the GPUs. In this host-slave parallel implementation of a GPU-enhanced cluster, the CPUs are responsible for all MPI communications, whereas GPUs do the computations using CUDA threads, as shown in Fig. 2.

The detailed flowchart of our cardiac simulations on a GPU-enhanced cluster is shown in Fig. 2. Each host CPU first initializes the data structure for its subdomain. Then, all the 3D arrays are transferred to the GPU side, which is responsible for all the computations. At the end of each time-step, MPI communication between pairs of the nearest neighbors is invoked by the host CPUs. This incurs the overhead of GPU-CPU-GPU data copy, which only touches the outermost boundary layer of each subdomain array U.

In addition to the above-mentioned overhead of shuffling boundary data between CPUs and GPUs, another limiting factor is that the GPUs usually have a smaller memory size than that on the host CPUs.

4.2 Computational Kernel on GPU

One detail of GPU programming for our case is about mapping the PDE-ODE computation to the CUDA threads. Data processing thread granularity and on-chip data locality have a major impact on a GPU kernel's performance. Figure 3 shows the organization of thread and data for the GPU computation kernel. Supplying a basic GPU implementation without extensive performance optimization, our purpose is to reveal a baseline performance to cardiac researchers, who may not be skilled at GPU programming.

First, at the top level, we partition a subdomain mesh of U that is assigned to a GPU into several sub-blocks of size $(n + 2) \times (m + 2) \times (k + 2)$, as shown in Fig. 3. Each such sub-block has an inner $n \times m \times k$ region plus a surrounding boundary layer. Due to no need of communication, the other state variable arrays need only to be divided into $n \times m \times k$ sub-blocks, i.e., without overlapping boundary layers. Then, on the second level, we assign each thread block of the GPU to perform computations on one sub-block, making use of the massive parallelism of multiple Stream Multi-processors (SMs) that constitute a GPU. Memory access locality should be carefully considered. Using the global memory of a GPU is costly, whose access latency is around 400 $\sim$ 600 cycles, while those of the on-chip shared memory and registers are less than 10 cycles. As shown at the bottom of Fig. 3, we elaborately distribute data to the global/shared/register memory respectively. Data processed by a thread block are loaded into the shared and local memory, on which the threads operate. In our 7-point stencil computations, each point value of U is used seven times. The seven global memory references can thus be replaced by seven shared memory references.

In our current implementation, each CUDA thread processes one mesh point, including its 7-ponit stencil computation and ODE solver. In total, 256 threads constitute a thread block, running on a SM of the GPU. This setting of thread granularity is to make as many thread blocks as possible to feed the 448 SPs in 14 SMs on a Tesla M2050 GPU.

5. Numerical Experiments and Discussion

Similar to [7], we have also chosen a cuboid as the 3D geometry of the cardiac tissues. Zero-flux boundary conditions are assumed. The size of our spatial domain is $5 \times 5 \times 1$cm. The value of σ_x is chosen as 3.2×10^{-4} cm^2/ms, and $\sigma_y = \sigma_z$ has value 1.3×10^{-4} cm^2/ms. We have mostly experimented with three spatial meshes: 501×501×101, 1001×1001×201 and 2001 × 2001 × 401, which have 0.1mm, 0.05mm and 0.025mm as the spatial resolution, respectively.

5.1 Simulation Setup and Results

As the hardware platform for all of our numerical experiments, we have used one portion of the Tianhe-1A Hunan Solution supercomputer [15], which is currently ranked as the world's No. 30 supercomputer according to the TOP500 list published in November 2012 [16]. This supercomputer consists of 2048 nodes connected via a customized THNI network. Each node is equipped with two six-core IntelX5670 (Nehalem-EP) 2.93 GHz CPUs and one Tesla M2050 GPU, together with 24GB DDR3 main memory and 3GB GPU global memory. Each GPU is attached to a dedicated PCI Express 2.0 bus (16 lanes, 16GB/s of bi-directional bandwidth). All the nodes are connected to a fat-tree interconnect with custom network interface and switch chips, where each channel has 160 Gbps bi-direction bandwidth. We used MPICH2 version 1.3.2 for inter-node communication, and CUDA v3.2 for GPU programming.

We have run three types of cardiac simulations, which are denoted by BF (Bueno cell model with forward Euler ODE solver), BG (Bueno cell model with GRL2 solver) and GG (Grandi cell model with GRL2 solver). It is remarked that the simple forward Euler solver often cannot produce stable numerical solutions for the Grandi cell model, therefore not used in connection with this advanced model. First, we tried a small mesh size on a single GPU and a single CPU core for obtaining a reference performance. Then we ran larger mesh sizes using multiple GPUs. Figure 4 shows a 2D cross-section of the 3D solution from BF, obtained at t=690ms.

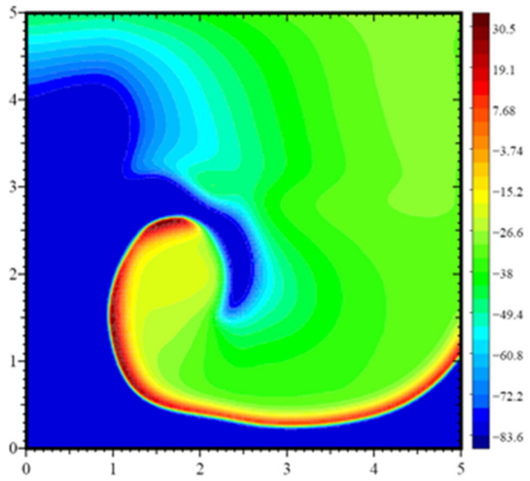


Fig. 4 2D cross-section of a 3D solution involving the Bueno cell model.

Table 1 Hardware counter statistics per mesh point and time step of a CPU implementation.

	#DP FLOPs	#MEM OPs	#L1 cache miss	#L2 cache miss	#L3 cache miss
BF	235	234	0.967	0.107	0.061
BG	2956	2354	1.020	0.125	0.075
GG	202068	172347	8.159	2.097	1.246

5.2 Single GPU v.s. Single CPU Core

We ran the three simulations on a single GPU and a single CPU core, respectively. Because of the limited size of GPU's global memory, we set the data mesh to 126×126×26 for these small runs. The number of time steps was fixed at 100.

For the purpose of understanding the performance differences between BF, BG and GG, we used PAPI [17] to get the hardware counter statistics on CPU. The measurements are shown in Table 1. On one hand, due to the complex cell model with 39 states and the GRL2 solver, GG needs 202068 floating-point operations per mesh point at every time step, about one thousand times more than BF, which has the simplest combination of cell model and ODE solver. On the other hand, with the same ODE model, the computation efforts of BF and BG differ because the GRL2 solver is drastically more expensive. Despite a large amount of floating-point and load/store operations, the numbers of L1/L2 data cache misses are relatively low because of good data locality, which is proper for achieving high perfor-

Table 2 Statistics of memory usage by a single GPU (126 × 126 × 26 mesh, 100 time steps).

	Amount of memory allocated				# memory accesses		
	registers per thread	shared memory per block	local memory per thread	constant memory per kernel	shared memory per warp	local memory per warp	global memory per kernel
BF	62	13232 bytes	32 bytes	44 bytes	415296	256032	1708108
BG	66	13232 bytes	160 bytes	56 bytes	415296	2820384	1708685
GG	77	13232 bytes	1632 bytes	124 bytes	437472	594472032	35640718

Table 3 Performance comparison between a single CPU core and a single GPU (126 × 126 × 26 mesh, 100 time steps).

	CPU		GPU		GPU/CPU
	Time	GFLOPS	Time	GFLOPS	speedup
BF	9.4s	1.02	0.4s	23.05	23
BG	113.7s	1.07	2.3s	54.28	51
GG	5610.0s	1.46	258.3s	31.68	22

mance.

Table 3 compares the performance between a single GPU and CPU core, where GFLOPS denotes 10^9 double-precision floating-point operations per second. We can see that using a GPU is much faster than using a CPU core, achieving about 20× ∼ 50× speedup. BG gets the largest speedup, and the reason is discussed below.

Comparing with CPU, the advantage of GPU comes from thousands of threads running on hundreds of cores, where the threads can utilize zero-cost context switching to hide the memory access latency. Hence, the GPU compiler allocates private on-chip registers for each thread. When there are insufficient registers, the GPU compiler will use the local memory in addition. However, GPU's local memory is off-chip, as expensive as accessing the global memory, which requires 400 ∼ 600 clock cycles of memory latency. The achievable performance is thus limited by the memory latency. We have summarized in Table 2 the GPU's memory usage reported by CUDA Profiler. The complex Grandi cell model, which has 39 state variables, requires using more registers, local memory and constant memory, as is evident by comparing BG and GG in Table 2. Consequently, the numbers of accesses to the local and global memory by GG are dramatically larger. This explains why the obtained performance of GG (in terms of GFLOPS) is lower than BG. The reason for the inferior performance of BF is due to a relatively low computation intensity.

Comparing BG with GG, their performance difference on a single CPU core is opposite to that on a single GPU. This is due to the architectural differences. When there are not enough registers on CPU, data is held in the caches, which are much faster to access than the off-chip local memory of GPU. As shown in Table 1, hardware and software data prefetch on CPU keeps the cache miss ratios very low. Therefore, a higher computation intensity makes GG more advantageous than BG on the CPU architecture.

5.3 Performance on Multiple GPUs

Now, we will investigate the strong-scaling property of multi-GPU simulations that ran on three spatial meshes: 501×501×101, 1001×1001×201 and 2001×2001×401. The detailed time usages are shown in Tables 4-6, which include the intra-node CPU-GPU data copy overhead, GPU kernel overhead and inter-node MPI communication overhead.

We can see in Table 4 that all the time costs decrease almost linearly with the increasing number of GPUs used. For a fixed mesh size, when more GPUs are used, the speedup achieved by GG is the best because the overheads have the least impact.

Table 4 Time usage (in seconds) of BF, BG and GG (501 × 501 × 101 mesh, 100 time steps).

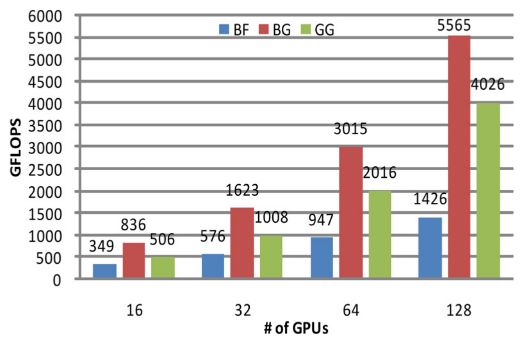
	# GPUs			
	16	32	64	128
BF-total	1.709	1.034	0.629	0.418
copy	0.648	0.338	0.185	0.087
compute	0.838	0.433	0.227	0.110
comm	0.223	0.263	0.217	0.221
BG-total	8.952	4.611	2.482	1.345
copy	0.665	0.336	0.180	0.087
compute	8.058	3.955	2.062	0.973
comm	0.229	0.320	0.240	0.284
GG-total	1011.594	507.593	253.722	127.088
copy	0.644	0.346	0.184	0.087
compute	1010.717	506.842	253.255	126.685
comm	0.233	0.405	0.283	0.316
File I/O	0.076	0.044	0.034	0.021

Table 5 Time usage (in seconds) of BF and BG (1001×1001×201 mesh, 100 time steps).

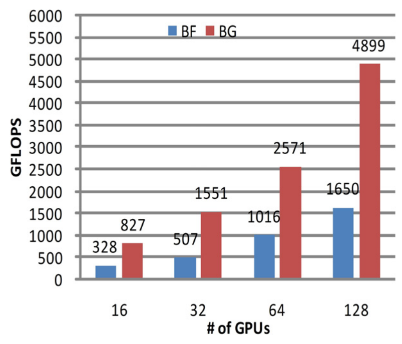
	# GPUs			
	16	32	64	128
BF-total	14.32	9.27	4.63	2.85
copy	4.83	2.44	1.24	0.60
compute	6.48	3.32	2.05	1.01
comm	3.02	3.51	1.34	1.23
BG-total	71.50	38.12	22.99	12.07
copy	4.85	2.44	1.27	0.62
compute	63.52	32.64	20.40	10.16
comm	3.14	3.04	1.32	1.29

Table 6 Time usage (in seconds) of BF and BG (2001×2001×401 mesh, 100 time steps).

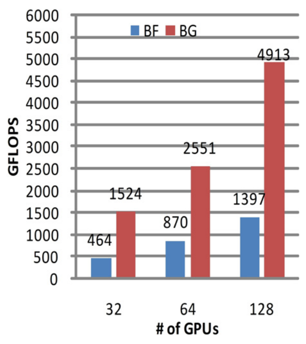
	# GPUs		
	32	64	128
BF-total	81.53	43.50	27.08
copy	19.30	9.92	5.60
compute	25.83	15.86	7.51
comm	36.40	17.72	13.97
BG-total	312.30	186.54	96.87
copy	19.20	9.79	5.59
compute	254.14	158.55	73.95
comm	38.96	18.21	17.33



(a) Mesh Size=501 × 501 × 101

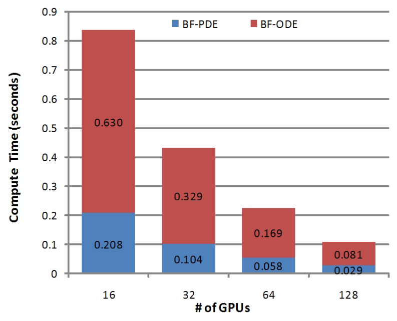


(b) Mesh Size=1001 × 1001 × 201

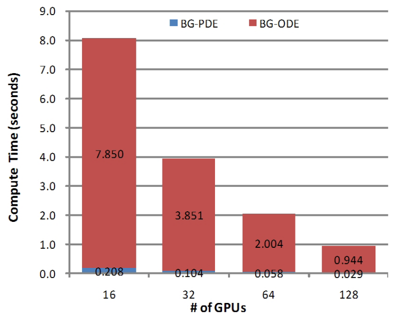


(c) Mesh Size=2001 × 2001 × 401

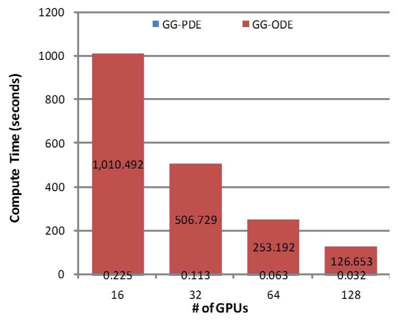
Fig. 5 Strong scaling characteristics (in terms of GFLOPS) of BF, BG and GG.



(a) BF



(b) BG



(c) GG

Fig. 6 Breakdown of the computation time of BF, BG and GG into the PDE and ODE parts, with mesh size=501 × 501 × 101 and 100 time steps.

According to Table 4, BF spends only 0.418 second to finish one hundred time steps using 128 GPUs. To simulate an entire heart cycle, supposing 10^5 time steps are needed (using $\Delta t = 0.01$ms), 418 seconds (seven minutes) are sufficient. On the other hand, GG spends 127 seconds for executing 100 time steps on 128 GPUs, which can be translated to 35 hours for a 10^5-time-step simulation of an entire heart cycle. The scalability of GG for using more GPUs will, however, be better than that of BF.

For practical studies, it is important to regularly store the computed results to files. If an entire simulation takes 10^5 time steps, i.e., Δt=0.01ms, it might be necessary to save to the files every 100th time step. We have therefore listed, as the bottom row of Table 4, the time spent on storing the v solution for one time step. It should be noted that this I/O cost does not depend on the particular numerical scheme, and is negligible in comparison with the total cost of 100 time steps. Moreover, the cost of I/O decreases as more GPUs are used, because the amount of subdomain data that each GPU stores to file decreases. In the Tianhe-1A system [15], several I/O management nodes and hundreds of I/O storage nodes are connected with all the compute nodes via a high-speed interconnect network with low latency and high bandwidth, leading to parallel and fast file I/O operations.

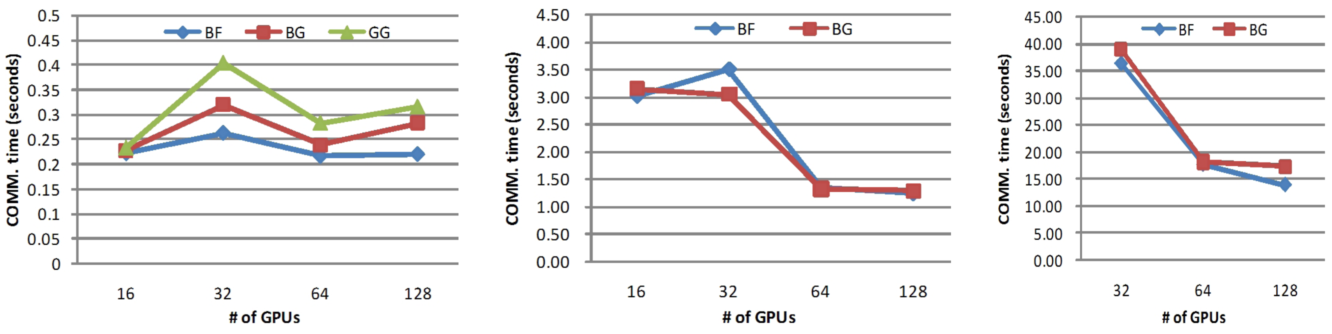
For Table 5, the spatial mesh was enlarged 8 times in comparison with Table 4. As expected, the property of strong scaling prevails in Table 5, as is also confirmed by Table 6 that used a global mesh of 2001 × 2001 × 401. The reason of not showing measurements of GG in Tables 5 and 6 is simply because running 100 time steps on these two huge meshes took too much time for GG. Also, for the 2001 × 2001 × 401 mesh, the aggregate global memory of 16 GPUs was not large enough for running BF or BG, thus no measurements for 16 GPUs in Table 6.

We have also translated the time measurements of Tables 4-6 into GFLOPS rates in Fig. 5. It can be seen that all the simulations got increased rates of GFLOPS as more GPUs were used. We can also see that BG achieved the highest GFLOPS rates among the three numerical schemes, corresponding to the same observation in Table 3. In particular, BG achieved 5565 GFLOPS using 128 GPUs, translating to an average of 43 GFLOPS per GPU, which is more than 8% of the theoretical peak performance of the M2050 GPU (515 GFLOPS).

To further dissect the computation time usage, we have plotted in Fig. 6 a breakdown into two parts: PDE time usage and ODE time usage. This dissection is associated with running 100 time steps on the 501 × 501 × 101 mesh. On average, the time used by the ODE solver amounts to 75%, 97% and almost 100% of the total computation time, associated with BF, BG and GG, respectively.

5.4 Discussion of Communication Cost

The overhead incurred by the inter-subdomain MPI com-



(a) Mesh Size=501 × 501 × 101 (b) Mesh Size=1001 × 1001 × 201 (c) Mesh Size=2001 × 2001 × 401

Fig. 7 Accumulated MPI communication overhead over 100 time steps.

Table 7 The maximum amount of MPI communication (in MBytes) per GPU at every time step.

Mesh size	# GPUs			
	16	32	64	128
501 × 501 × 101	2.4	1.28	0.74	0.41
1001 × 1001 × 201	16.4	8.4	4.5	2.4
2001 × 2001 × 401	128	64.8	33.6	18.8

munication is worth a closer look. To this purpose, we have plotted in Fig. 7 the time spent on the MPI communication by 100 time steps. It should be noted that the MPI overhead should be independent of the numerical scheme, because the frequency and amount of MPI data exchange are the same for BF, BG and GG. Therefore, the differences between the three numerical schemes in Fig. 7 are due to noises in the time measurement and also because of the dynamic status of network contention. Nevertheless, there are some interesting observations.

First, for a given number of GPUs, the MPI overhead increases with the increased global mesh size. This follows naturally from the fact that the amount of MPI data exchange increases, as illustrated by Table 7. The reason of not seeing an increasing factor of 4 between two consecutive global mesh sizes in Table 7 is because different partitioning strategies were adopted. Take the case of 16 GPUs, we partitioned the $501 \times 501 \times 101$ mesh into a $[1, 4, 4]$ decomposition, whereas the $1001 \times 1001 \times 201$ mesh used a $[1, 2, 8]$ decomposition and the largest $2001 \times 2001 \times 401$ mesh used a $[1, 1, 16]$ decomposition. The varying decompositions were motivated by the wish to maintain an appropriate size of the thread blocks, while satisfying an old version of the CUDA driver that only allows one thread block per GPU in the z-direction.

Second, for a fixed global mesh size, the MPI overhead fluctuates as the number of GPUs increases. This is because, although the amount of MPI data exchange per GPU decreases as more and more GPUs are in use, the factor of communication latency may still be influential, especially for the smallest $501 \times 501 \times 101$ mesh. The influence of latency decreases as the mesh size becomes larger. Moreover, the topological structure of the Tianhe-1A interconnect network is a hierarchical fat tree [15]. At the bottom level, 16 nodes are connected with each other through the switch board in each rack. This means that communication within 16 GPUs has a higher bandwidth than that involving 32 GPUs or more.

5.5 Predicting Time Usage

Considering the observed linear trend of total time usage with respect to the mesh size, while assuming good scalability, we can derive a simplistic formula for roughly predicting the time cost of cardiac simulations on a GPU enhanced cluster as follows:

$$T_e = \frac{M_S \times F_a \times N_t}{P \times R \times N_G}, \tag{3}$$

where T_e denotes the expected total time usage. M_s is the mesh size and F_a is the average floating-point operations per point at every time step, which can be measured by a performance tool such as PAPI (running a small simulation on CPU). N_t denotes the number of time steps. P is the theoretical peak performance of one GPU. R is the realistically achievable fraction of the peak, which can be estimated such as between 5% and 8% from our experiments. N_G is the number of GPUs used in the simulation.

6. Conclusion

In this paper, we have looked at multi-GPU simulations of cardiac electrophysiology, from the angle of achievable performance and scalability on up to 128 GPUs. By choosing two representative cell models and two representative ODE solvers, we believe that our investigation can provide cardiac researchers with a realistic estimate of the actual capability of GPU-enhanced clusters for such computations. Painstaking and model-specific GPU code optimizations have thus not been our focus in this paper. We also believe that, for sophisticated cell models that involve many state variables, scalability can be achieved on even larger numbers of GPUs, giving the hope of efficiently carrying out demanding simulations of cardiac electrophysiology.

Acknowledgements

The assistance from Dr. Glenn T. Lines of Simula Research

Laboratory is gratefully acknowledged. The authors also gratefully acknowledge the support from National Natural Science Foundation of China under NSFC Nos. 61033008, 61103080 and 61272145, Research Fund for the Doctoral Program of Higher Education of China under SRFDP No. 20104307110002, Hunan Provincial Innovation Foundation for Postgraduate under No.CX2010B028, Fund of Innovation in Graduate School of NUDT under No.B100603. We also thank the National Supercomputing Center in Changsha for providing technical asistance. Last but not least, the authors thank the financial support from the Research Council of Norway, through FRINATEK project No. 214113.

References

[1] B.G. de Barros, R.S. Oliveira, W. Meira Jr., M. Lobosco, and R.W. dos Santos, "Simulations of complex and microscopic models of cardiac electrophysiology powered by Multi-GPU platforms," Comput Math Methods Med, vol.2012, published online: Nov. 2012, doi: 10.1155/2012/824569.

[2] E. Bartocci, E.M. Cherry, J. Glimm, R. Grosu, S.A. Smolka, and F.H. Fenton, "Toward realtime simulation of cardiac dynamics," Proc. CMSB'11, pp.103–112, New York, USA, 2011.

[3] A. Bueno-Orovio, E.M. Cherry, and F.H. Fenton, "Minimal model for human ventricular action potentials in tissue," J. Theoretical Biology, vol.253, no.3, pp.544–560, Aug. 2008.

[4] R.H. Clayton, O. Bernus, E.M. Cherry, H. Dierckx, F.H. Fenton, L. Mirabella, A.V. Panfilov, F.B. Sachse, G. Seemann, and H. Zhang, "Models of cardiac tissue electrophysiology: Progress, challenges and open questions," Progress in Biophysics and Molecular Biology, vol.104, pp.22–48, Jan. 2011.

[5] E. Grandi, F.S. Pasqualini, and D.M. Bers, "A novel computational model of the human ventricular action potential and Ca transient," J. Molecular and Cellular Cardiology, vol.48, no.1, pp.112–121, Jan. 2010.

[6] A. Neic, M. Liebmann, E. Hoetzl, L. Mitchell, E.J. Vigmond, G. Haase, and G. Plank, "Accelerating cardiac bidomain simulations using graphics processing units," IEEE Trans. Biomed. Eng., vol.59, no.8, pp.2281–2290, Aug. 2012.

[7] S.A. Niederer, E. Kerfoot, A.P. Benson, M.O. Bernabeu, O. Bernus, C. Bradley, E.M. Cherry, R. Clayton, F.H. Fenton, A. Garny, E. Heidenreich, S. Land, M. Maleckar, P. Pathmanathan, G. Plank, J.F. Rodríguez, I. Roy, F.B. Sachse, G. Seemann, O. Skavhaug, and N.P. Smith, "Verification of cardiac tissue electrophysiology simulators using an *N*-version benchmark," Phil. Trans. R. Soc. A, vol.369, no.1954, pp.4331–4351, Nov. 2011.

[8] S. Niederer, L. Mitchell, N. Smith, and G. Plank, "Simulating human cardiac electrophysiology on clinical time-scales," Frontiers in Physiology, published online: April 2011, doi:10.3389/fphys.2011.00014.

[9] V.K. Nimmagadda, A. Akoglu, S. Hariri, and T. Moukabary, "Cardiac simulation on multi-GPU platform," J. Supercomput, vol.59, no.3, pp.1360–1378, March 2012.

[10] B.M. Rocha, F.O. Campos, R.M. Amorim, G. Plank, R.W. dos Santos, M. Liebmann, and G.xs Haase, "Accelerating cardiac excitation spread simulations using graphics processing units," Concurrency and Computation: Practice and Experience, vol.23, no.7, pp.708–720, May 2011.

[11] S. Rush and H. Larsen, "A practical algorithm for solving dynamic membrane equations," IEEE Trans. Biomed. Eng., vol.25, pp.389–392, July 1978.

[12] D. Sato, Y. Xie, J.N. Weiss, Z. Qu, A. Garfinkel, and A.R. Sanderson, "Acceleration of cardiac tissue simulation with graphic processing units," Medical and Biological Engineering and Computing, vol.47, no.9, pp.1011–1015, Sept. 2009.

[13] J. Sundnes, R. Artebrant, O. Skavhaug, and A. Tveito, "A second order algorithm for solving dynamic cell membrane equations," IEEE Trans. Biomed. Eng., vol.56, no.10, pp.2546–2548, Oct. 2009.

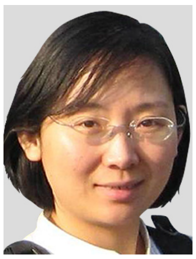
[14] E.J. Vigmond, P.M. Boyle, L. Leon, and G. Plank, "Near-real-time simulations of bioelectric activity in small mammalian hearts using graphical processing units," Proc. IEEE Engineering in Medicine and Biology Society 2009, pp.3290–3293, Sept. 2009.

[15] X.J. Yang, X.K. Liao, K. Lu, Q.F. Hu, J.Q. Song, and J.S. Su, "The TianHe-1A supercomputer: Its hardware and software," J. Computer Science and Technology, vol.26, no.3, pp.344–351, May 2011.

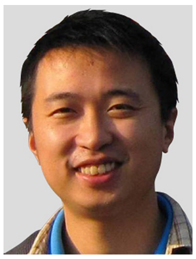
[16] Top500, "November 2012," Top500 Supercomputer Sites, http://www.top500.org/lists/2012/11, accessed Dec. 25. 2012.

[17] PAPI 5.0.1, "Current PAPI Software and Patches," The University of Tennessee, http://icl.cs.utk.edu/papi/software/index.html, accessed Dec. 25. 2012.

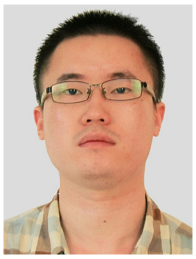
Jun Chai was born in 1985. He is a Ph.D. candidate in School of Computer at National University of Defence Technology. His research interests include parallel programming, high-performance scientific computing, and computer architecture.



Mei Wen was born in 1975. She is an associate professor in School of Computer at National University of Defence Technology. Her research interests include computer architecture, parallel programming and scientific computing.



Nan Wu was born in 1980. He is an associate professor in School of Computer at National University of Defence Technology. His research interests include computer architecture, parallel programming and computer network.



Dafei Huang was born in 1987. He is a Ph.D. candidate in School of Computer at National University of Defence Technology. His research interests include parallel programming and computer architecture.

Jing Yang was born in 1989. She is a M.Sc. candidate in School of Computer at National University of Defence Technology. Her research interests include parallel programming and computer architecture.

Xing Cai was born in 1968. He is a professor in Department of Informatics at University of Oslo and Simula Research Laboratory. His research interests include parallel programming, high-performance scientific computing, numerical methods and generic PDE software.

Chunyuan Zhang was born in 1964. He is a professor in School of Computer at National University of Defence Technology. His research interests include computer architecture, parallel programming, low power design, embedded systems, media processing and scientific computing.

Qianming Yang was born in 1984. He is a Ph.D. candidate in School of Computer at National University of Defence Technology. His research interests include computer architecture, parallel programming, advanced memory design and embedded system.