2016 IEEE 22nd International Conference on Parallel and Distributed Systems

Enabling Tissue-scale Cardiac Simulations using Heterogeneous Computing on Tianhe-2

Johannes Langguth*, Qiang Lan‡§, Namit Gaur*, Xing Cai*†, Mei Wen‡§, and Chun-yuan Zhang‡§
*Simula Research Laboratory, 1325 Lysaker, Norway
Email: {langguth,ngaur,xingca}@simula.no
†Department of Informatics, University of Oslo, 0316 Oslo, Norway
‡College of Computer, National University of Defense Technology, Changsha 410000, China
Email: {lanqiang,meiwen,cyzhang}@nudt.edu.cn
§National Key Laboratory of Parallel and Distributed Processing, Changsha 410000, China

***Abstract*—We develop a simulator for 3D tissue of the human cardiac ventricle with a physiologically realistic cell model and deploy it on the supercomputer Tianhe-2.**

In order to attain the full performance of the heterogeneous CPU-Xeon Phi design, we use carefully optimized codes for both devices and combine them to obtain suitable load balancing.

Using a large number of nodes, we are able to perform tissue-scale simulations of the electrical activity and calcium handling in millions of cells, at a level of detail that tracks the states of trillions of ryanodine receptors. We can thus simulate arrythmogenic spiral waves and other complex arrhythmogenic patterns which arise from calcium handling deficiencies in human cardiac ventricle tissue.

Due to extensive code tuning and parallelization via OpenMP, MPI, and SCIF/COI, large scale simulations of 10 heartbeats can be performed in a matter of hours. Test results indicate excellent scalability, thus paving the way for detailed whole-heart simulations in future generations of leadership class supercomputers.

I. Introduction

One of the most important subjects in computational cardiology is the study of arrhythmias that occur at the tissue and organ scale. For physiological fidelity, it is important to adopt sophisticated cell models of electrophysiology and calcium handling, because these models incorporate the discrete nature of subcellular stochastic calcium release processes [1], [2], [3], [4]. Realistic simulations of calcium handling and action potential formation are essential for understanding the causative and preventive mechanisms of arrhythmogenesis, which originates from the local nanoscopic level of channel and dyadic dysfunction and develops into the subcellular and cellular levels of membrane potential abnormalities. These can be in the form of delayed afterdepolarizations, early afterdepolarizations [5], and cardiac alternans [1], [6], [7].

Due to the large computational power already required for simulating one single cell, simulations at the tissue or organ level have only recently started to become computationally feasible. A typical human heart has around 2×10^9 cells [8]. Each cardiac cell has about 10^6 ryanodine receptors (RyRs) that are distributed over roughly 10^4 calcium release units, also called dyads. Each dyad also has a number of L-type channels operating stochastically in response to membrane potentials and local calcium concentrations [9].

Thus, at the tissue level, when simulating 10^6 cells with 10^6 RyRs each for a few heartbeats requiring 10^5 time steps, the need for supercomputing is obvious. Simulations at the organ level are even more resource-hungry. An MPI+OpenMP implementation was presented in [10], where we found that the principal bottleneck is the CPU performance. Luckily, due to the massive parallelism involved, tissue-level simulations are well suited for making use of hardware accelerators. In [11], the authors use GPUs to accelerate simulations at the cell level.

Our goal is to enable massive simulations on *Tianhe-2* [12], which is currently the second fastest supercomputer according to the TOP500 ranking [13]. Each node is equipped with two Intel Xeon CPUs, in addition to three Xeon-Phi coprocessors. We intend to use both resources, even though such a heterogeneous computation introduces another level of programming complexity.

To effectively use a system such as *Tianhe-2* requires distributing the computation on four different levels of parallelism: between the compute nodes, between the CPU and Phi compute devices on the same node, between the cores on the same device, and between the SIMD lanes in every core since each core is equipped with SIMD vector units that allow it to process multiple calculations at the same time.

II. Target Hardware

The *Tianhe-2* supercomputer consists of 16,000 nodes, and thus a total of 48,000 Xeon Phis of model 31S1P (i.e. Knigths Corner architecture) each with 57 cores running at 1.1 GHz, and 32,000 Intel Xeon E5-2692v2 Ivy Bridge CPUs, each of which has 12 cores running at 2.2 GHz. The Intel Xeon Phi coprocessor is a hardware accelerator based on the manycore design principle. Unlike traditional multicore CPUs that feature a small number of powerful cores, it is composed of a large number of relatively simple cores with wide vector units that are connected to each other and to the device memory via a ring bus. Each core features 64 KB of level 1 cache and 512 KB of level 2 cache. All caches maintain coherency among the cores. This design makes the device ill-suited for sequential programs, but it can be very powerful for highly parallel workloads. In comparison, the Xeon CPUs have 64 KB of level 1 cache and 256 KB of level 2 cache. A core can

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.112

run two hardware threads using hyperthreading, although this feature is currently disabled on the *Tianhe-2* nodes. Unlike the Xeon Phi, each CPU also has 30 MB of L3 cache which is shared among its 12 cores.

The theoretical peak double-precision floating-point capability of a Xeon Phi is derived from its 512-bit vector length. Each core can thus perform 8 double-precision fused multiply-add operations per clock cycle, which yields a total performance of 16 times the number of cores times the clock frequency. For the 31S1P model, this implies a peak performance of 1001 GFlop/s [14]. However, we cannot expect to attain a real-life performance close to the theoretical peak. Therefore, it is worthwhile to include the CPUs in the computation even though they only provide at most 211 GFlop/s each. In total, the *Tianhe-2* system has a peak performance of 54.9 PFlop/s and a Linpack benchmark performance of 33.86 PFlop/s [13].

For inter-node communication, the system uses a custom interconnect called TH Express-2, which has a fat tree topology. While it is theoretically possible to place MPI processes directly on the Phis, doing so provided low performance when communicating between Phis that belong to different nodes. Instead, we only use a single MPI process per node and use the Intel SCIF/COI interface for communication between the Phis and their CPU hosts, in a manner similar to [15].

In this mode, one of the Phi cores is reserved by the operating system and thus not available for our computations, leaving us 56 cores per device. Each core can run up to four hardware threads concurrently, but a given thread is never run in two consecutive clock cycles. This means that effectively 112 threads can run in parallel at the clock frequency of 1.1 GHz. When using 224 threads, i.e. four per core, each thread effectively runs for half the available processing time, which has an effect that is similar to hyperthreading. The 31S1P model is equipped with 8 GB of DDR5 device memory, featuring a theoretical memory bandwidth of 320 GB/s [14]. However, even under ideal circumstances at most half of that is available for applications [16].

III. Mathematical Modeling

A. Physiologically Detailed Cell Modeling

Our multiscale model of stochastic calcium handling in a ventricular myocyte is similar to [2], combined with the O'Hara-Rudy (ORd) model [17] of electrophysiological current formulation for a healthy human cardiac ventricular action potential.

In each cell, we assume that 10,000 calcium release units or dyads are arranged as an internal $100 \times 10 \times 10$ grid. Each dyad consists of five calcium compartments: (1) myoplasm, (2) submembrane space, (3) network sarcoplasmic reticulum, (4) junctional sarcoplasmic reticulum, and (5) dyadic space. Of particular importance, however, is the dyadic space that contains 15 L-type calcium channels and 100 RyRs that operate stochastically. More specifically, each RyR can be in one of four states, denoted as C1, C2, C3, and O1, at any given time. Figure 1 shows the possible transitions between them, which occur stochastically with probabilities that are related to the local calcium concentrations. The number of open RyRs, i.e. those having state O1, is of principal interest due to the effect of calcium influx on a cell's interior voltage.

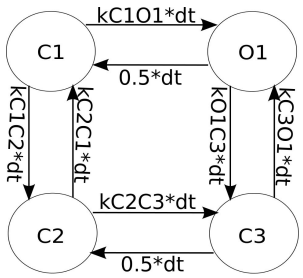

Fig. 1. The eight possible transitions between four states of a RyR, where the labels on the arrows indicate the probabilities of the transitions.

A slab of cardiac tissue is modeled as a 3D uniform grid of cells. To simulate the tissue-scale electrical activity, we use the following partial differential equation, which is commonly called the monodomain model:

$$\frac{\partial V_m}{\partial t} = \frac{-I_{\text{ion}}}{C_m} + D_x \frac{\partial^2 V_m}{\partial x^2} + D_y \frac{\partial^2 V_m}{\partial y^2} + D_z \frac{\partial^2 V_m}{\partial z^2}, \quad (1)$$

where V_m is the membrane potential, I_{ion} is the ionic current provided by the underlying multiscale cell model of calcium handling, $C_m = 1\mu F\text{cm}^{-2}$ is the membrane capacitance of the cell, $D_x = D_y = D_z = 0.2$ mm^2/ms are the voltage diffusion coefficients in three spatial directions.

B. Numerical Strategy

The overall computational procedure is carried out as a time loop, where the work of each time step consists of first looping over all the cells, and then stepping forward the monodomain equation (1). The computational work per cell is in the form of another loop that goes through all the dyads one by one, plus the subsequent inter-dyad diffusion computations. This dyad-level loop contains the most time-consuming computations, due to the large number of dyads per cell.

During each time step, in every dyad, we sample twice from a binomial distribution for each of the four states in Figure 1, i.e. one sampling for each of the two possible transitions from a state. We thus obtain the number of RyRs that changed state in the current time step. The RyRs for which no transition happened remain in the original state. We add the RyRs that transitioned from the neighboring states to obtain the final number of RyRs in each of the four states. L-type calcium channels are treated in a similar fashion. Compared with performing Bernoulli trials, our approach based on binomial distribution requires considerably fewer random values, which are relatively expensive to generate. While binomial distributions can be approximated using the normal and Poisson distributions, which are e.g. used in [1], we opt against sacrificing accuracy and instead design an optimized implementation.

Explicit time integration is used to solve all the differential equations. The involved diffusion terms are discretized with

centered finite differences. For the monodomain equation (1), an operator-splitting approach [18] is used. This means that the diffusion terms are treated separately from the I_{ion} term, where the latter is computed by summing up the total ionic currents obtained from solving the ordinary differential equations in the ORd model.

Thus, the basic structure of the main computation which is performed for every dyad in every cell consists of the probability computation and stochastical opening of both L-type and RyR channels, the computation of the resulting calcium concentration, and finally the voltage diffusion among the dyads. Due to the large amount of random numbers involved, producing them is another significant part of the computation. Due to the large number of dyads per cell, computations and voltage diffusion for the cells do not consume significant compute time.

IV. Implementation

A. Tissue-Level Parallelization

The entire computational problem allows decompositions at multiple levels. Among these, the top level is the cardiac ventricle tissue that is represented as a 3D Cartesian grid of cardiac cells. Each compute node is assigned a cuboid subdomain containing an equal number of cells. Cell voltage values V_m are exchanged along the faces of the subdomains during every time step, via MPI messages [19] between the host CPUs using a standard halo exchange approach. We do not enforce a further geometric partitioning within each cuboid, because this induces unnecessary difficulties in balancing the load among the CPU host and the Phis. Instead, each compute device is assigned a number of cells that is commensurate with its relative speed. This however necessitates exchanging all voltage values V_m within a node. The transfer is performed using the Intel SCIF/COI interface, which has been shown to be an effective way of implementing CPU-Phi communication [15]. Thus, improved flexibility with respect to load balancing easily outweighs the cost of exchanging all voltage values within a node. Figure 2 shows an example of such an intra-node decomposition of a subdomain.

In addition, the limited memory (8 GB) and the large number of concurrent threads (224) of the Xeon Phi mandate further constraints. Due to the large number of dyads, each cell requires approximately 3MB of memory, which limits the number of cells on each Phi to about 2500. This in turn implies that each thread computes only a small number of cells (up to 12) per time step, which makes the computation quite sensitive to unbalanced loads. Therefore, each Phi should be assigned a number of cells which is a multiple of 224 for optimum load balance within the device.

During preliminary experiments, we found that the relative computational speed between the host CPU (two sockets) and one Phi is about $1.5:1$. Together with the above restrictions, we found the optimal subdomain size to be $20\times20\times20$, with $8\times224=1792$ cells assigned to each Phi and the remaining $8000-3\times1792=2624$ cells assigned to the host CPU, which thus processes 32.8% of the cells. Even though we do not always use them in our experiments, proportionally smaller or larger subdomains obeying the same ratios are also possible. Furthermore, we note that performance may vary slightly between the computations, which makes it impossible to have a strictly optimal load balance.

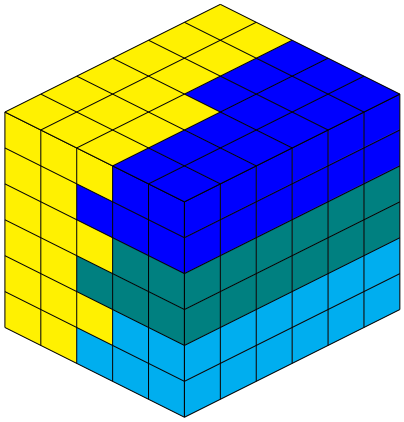

Fig. 2. Example of the intra-node decomposition of a subdomain. Cells allocated to the Phis are shown in blue, teal, and cyan, while the CPU cells are shown in yellow.

B. Cell-Level Parallelization

As stated above, we perform a straightforward and static assignment of cells to the devices, which means that all computations performed for a cell remain on a single core. Alternatively, we can split the computations for the $10,000$ calcium release units over the cores of a device. Doing so would alleviate the load balancing inflexibility. However, due to the fact that parts of the cell computations have to be performed sequentially, this would lead to significant idling of cores during the computation.

Thus, on each core we use the SIMD vector units to process four (CPU) or eight (Xeon Phi) calcium release units in parallel within the cell. This vectorization is key to obtaining efficient simulation code. Due to the complexity of the simulation, we employ mixed vectorization, employing both automatic vectorization via the Intel icc compiler and manual vectorization through the Intel intrinsics library. To do so, it is necessary to split the original dyad-loop into sections of two different types. Sections of the *arithmetic* type contain expensive computations, but have a trivial control flow. Modern compilers are capable of automatically vectorizing such loops perfectly. The second type on the other hand is characterized by a complex control flow, preventing the compiler from vectorizing the loop on its own. These sections will be labeled as the *conditional* type.

In our simulations, *arithmetic* sections involve determining the probabilities for the opening of L-type channels or RyR state transitions, computing the calcium concentrations, and performing the diffusion among the dyads. They have a simple control flow and can be vectorized automatically, although the inter-dyad diffusion section is memory bound and contains

few arithmetic instructions, which limits its scalability. On the other hand, the *conditional* sections concern sampling from binomial distributions in order to determine the number of state transitions, since this requires *while* loops or equivalent structures to be efficient. Compilers are unable to vectorize these on their own.

The main disadvantage of this technique lies in the reduced data locality when one large computational loop is split into several smaller loops. Instead of performing the entire computation for one element, one section is executed for all elements which implies that intermediate results (such as probabilities for RyR state transitions in our code) must be written to memory and retrieved later, which puts additional pressure on the memory bandwidth. Avoiding the resulting memory traffic provides an incentive to manually vectorize small *arithmetic* sections as well and merge them with *conditional* sections in order to overcome the need for additional memory operations. Doing so provided small but noticeable performance gains.

C. Vectorized Binomial Sampling

We vectorize the *conditional* sections manually using the AVX-512 instructions[1] of the Xeon Phi [20]. The key to vectorizing the code despite the conditional statements lies in the powerful `mask` instructions. They allow the SIMD units to apply vector instructions only on some elements of the vector, which are determined by a bit mask. The AVX vector instructions of the Sandy Bridge CPU lack these instructions. Therefore, in the CPU code binomial sampling is not vectorized.

The AVX-512 vectorization is shown in Figure 3. It performs the sampling by implicitly computing the cumulative distribution functions for 8 dyads simultaneously, taking as input the vectors `N, P` and `RANDVAL` which contain the numbers of channels in the respective state, the transition probabilities, and a random number for the sampling for each dyad respectively. The results of the samplings *K*, i.e. the number of RyRs that change state[2], are computed by reducing the entries of `RANDVAL` in iteration i by the probability that a given binomial distribution returns the value i. If a given entry of `RANDVAL` becomes negative, its corresponding value of `MASK` will be set to 0. In that case its value in `K` will remain fixed and become the sampling output.

To save time, vector versions of the coefficients `PKNK` and `P1P` are computed in the initialization. Since the maximum value in `N` will never be above 100, we can compute the binomial coefficients prior to the actual simulation and store them in `BC_table`. While this is very helpful in the scalar case on the CPU, the fact that memory bandwidth per core is limited, while computational power is plenty on the Phi negated the performance advantage of this technique.

[1]Technically, the Xeon Phi only possesses a subset of the upcoming AVX-512 instruction set called Intel Initial Many Core Instructions (Intel IMCI).

[2]We use exactly the same binomial sampling technique for both the RyRs and L type channels. For the sake of readability, the sampling is described here for RyRs only.

```
function VectorizedBinomial
Input: Vectors N, P, RANDVAL
Output: Vector K
   Initialize K = 0
   Initialize 1P = Vector_subtract(1,P);
   Initialize PKNK = Vector_power(1P,N);
   Initialize P1P = Vector_divide(P,1P);
   for (int i = 0; i < max(N); i++) {
       BC = Vector_gather(BC_table,N,K);
       SUB = Vector_multiply(BC,PKNK);
       RANDVAL = Vector_subtract(RANDVAL,SUB);
       PKNK = Vector_multiply(P1P,PKNK);
       MASK = Vector_mask_compare(RANDVAL > 0);
       K = Vector_mask_add(K,1,MASK); }
```

Fig. 3. The vectorized version of the binomial sampling computation. It can be derived from a scalar implementation via the use of MASK to increase the output value for some vector elements, instead of using a `while` loop. Vector intrinsics are renamed for clarity.

The main problem with this approach is that the computation runs until the last sampling is finished. This means that the speedup depends on the ratio between the sampling results of the 8 dyads within one vector. In case they all happen to be identical, we can obtain a perfect eightfold speedup, but in practice this will rarely be the case. In addition, testing if the `MASK` value is 0, i.e. all samplings have completed to allow early termination is expensive, especially on the Xeon Phi. We settled for a compromise where we perform this test after every 8 iterations of the for loop, and terminate the loop if all 8 samplings are finished.

This means that the lower the higest value in `K`, the less time the sampling takes, which is desirable since most transition probabilities and thus sampling results are low. In addition, if an entry of `N` is 0, the corresponding value of `K` will always be 0. In that case, we can immediately set the result and store the entry of `RANDVAL` for later use. Furthermore, as shown in Figure 1, transitions from the O1 state to the C1 state, and those from the C3 state to the C2 state have a constant probability, which means that for these cases the cumulative distribution function can be precomputed. Doing so further cuts down the number of actual sampling function evaluations.

Note that sampling from a binomial distribution is indeed not a method that is very amenable to vectorization, and a straightforward implementation that stochastically determines the number of open channels using a large number of Bernoulli experiments would vectorize better due to the `compare` intrinsics available on the Xeon Phi. However, the amount of random numbers required for doing so renders this approach prohibitively expensive. This is particularly problematic on the Xeon Phi where random number generation is disproportionately more expensive.

D. Random Number Generation

In order to generate the random numbers required by the binomial samplings, we use the standard function `vdRngUniform` provided by the Intel MKL library [21] at the beginning of every time step. Performance of this function

on the Xeon Phi is quite low compared to the CPU, being almost 40 times slower on a single core. In addition, unlike on the CPU, random number generation does not scale perfectly on the Xeon Phi when increasing the number of threads, making it one of the main performance bottlenecks on the accelerator.

Since `vdRngUniform` is a predefined library function, we can perform no further optimizations on it. However, as mentioned above, we mitigate its impact by skipping over all state transitions where the number of channels in the starting state is 0, thereby conserving random numbers. To do so, we keep a count of the random numbers used in the current time step. At the beginning of the subsequent time step, only the random numbers that were actually consumed by state transitions must be replaced. Due to the 8 possible state transitions for the RyR channels and 2 additional transitions for the L-type channels, up to 10 random numbers per dyad can be used in every time step. The above technique cuts the average number approximately in half. Note that even though the running times of the *conditional* sections are non-deterministic, the large number of binomial samplings per cell (up to $800,000$ for the standard subdomain size on the Phi) on each core in each time step makes it extremely unlikely to encounter load imbalance for that reason.

E. Code Optimization

Despite the strong similarities between the x86 CPU und the Xeon Phi, porting high performance code between the two devices is not an easy task. Low sequential performance, long vector units, and higher computational vs. memory performance on the Xeon Phi lead to several challenges that must be addressed when using these devices.

Since the Xeon Phi relies heavily on its 512-bit vector length to attain high computational performance, successful vectorization of the arithmetic sections is paramount to rendering the device competitive. Because all relevant variables use double precision, this provides an eightfold speedup on the Xeon Phi and a fourfold speedup on the CPU. The Intel icc compiler is able perform this vectorization automatically.

For the CPU, AVX vectorization cannot provide a quadruple speedup everywhere, because only two double precision divisions can be performed simultaneously, as discussed in [22]. Due to the fact that a division is almost 20 times as expensive as a multiplication on the CPU, vectorization speedup is dominated by the speedup attained on divisions. The Xeon Phi does not suffer from this problem and can perform eight divisions in parallel on each core, but divisions remain very expensive here. In the code, they are manually replaced by multiplications wherever possible, since the compiler does not perform this optimization automatically.

For the intracellular, inter-dyad diffusion computations, we found that automatic vectorization is also beneficial. An efficient implementation of the stencil computation can avoid using conditional statements, since the instructions to be executed are not data-dependent. However, since these operations are always memory bound, the expected speedup is lower than for the purely arithmetic operations. For the CPU, we obtained better performance by using manually vectorized code. Note that the intercellular diffusion computations do not take a significant amount of time in comparison to the intracellular diffusions, since there are $10,000$ dyads in each cell.

V. Experimental Setup

All experiments are performed on the *Tianhe-2* supercomputer. Codes are compiled with the Intel icc compiler version 14.0.2. For MPI communication the system uses MPICH version 3.1.1. We spawn only one MPI process per node. This process is controlled by the CPU. To launch Phi computation and to communicate between accelerator and host, we use the low-level COI and SCIF interfaces. While these are not particularly user-friendly or easy to program, doing so gives us full control over the communication. The Xeon Phi accelerators are thus used in an offload strategy, even though we do not employ the directive based *offload* mode offered by the Intel compiler. While it would be possible to use MPI at this level, doing so would result in either suboptimal communication or suboptimal load balancing due to the wide parallelism of at least 112 threads per Xeon Phi.

We use OpenMP for shared memory parallelism over all cores, both on the CPUs and on the Xeon Phi, placing 24 and 224 threads per device respectively. On *Tianhe-2*, hyperthreading is disabled for the CPUs. Preliminary tests on a dual Sandy Bridge machine with 16 cores in total showed an improvement in performance due to hyperthreading of 23%. It therefore stands to reason that we would see improved performance due to hyperthreading on the Ivy Bridge CPUs as well.

Note that since both CPU sockets can access shared memory, we treat them as a single device. Threads are allocated using *scatter* affinity, i.e. in a round-robin fashion. While all Xeon Phis are of the early 31S1P model which is designed for a clock frequency of 1.1 GHz, due to stability reasons some Phis are underclocked to 0.8 GHz, which affects the overall results. We do not compensate for this machine specific problem. Since access to the machine is partially restricted, we were limited to using 400 Xeon Phi enabled nodes in our experiments.

For all the experiments, we use a fixed time step of 0.05 ms at both the tissue level and the cell level. For the tissue-scale simulations, we chose a fixed spatial mesh resolution of 0.5 mm to discretize the diffusion terms in (1). In our cardiac simulations we run 10,000 time steps which amounts to simulating one cardiac beat of 500 ms. For the large scaling experiments, this is reduced to 200 ms. Unless otherwise noted, the cells are stimulated at $t = 50$ ms.

To make running times comparable between different simulation durations, we report performance in cell computations per second (CC/s). A single cell computation includes close to 6.4 million floating point operations. However, as it contains a large number of divisions and few multiply-add operations, this number is not comparable to other computations [22].

VI. Performance Experiments

A. Device Performance

We perform a series of experiments to study the performance of our implementation. As a first step, we test the single core performace on both CPU and Xeon Phi. Results are shown in Figure 4. Clearly, the mixed vectorization succeeds in cutting the compute time of the *arithmetic* sections. On the Xeon Phi, Calcium concentration computation sped up by a factor of 8, which is the maximum due to the vector length. L-type probability calculation attained a factor of 6. Even though it is a purely arithmetic section, it contains only few calculations but sizeable memory transfers, thus it is most likely somewhat latency bound. The intracellular diffusion sped up fivefold. Since this part consists mainly of stencil computations, it is ultimately memory bound. Finally, the speed of the RyR probability calculation quadrupled. We found that the principal reason inhibiting further speedup is the use of a single exponential function, which takes up the majority of the compute time there.

For the *conditional* sections, we observe large gains for the L-type channels of about a factor of 6 as the result of our manual vectorization. Here, all binomial distributions have the maximum number of iterations. For the RyR channels, this is not the case. Consequently, vectorization gains are small here at about 20%, making this the most expensive section in the entire cell computation. Furthermore, random value conservation also provides a significant benefit on the Xeon Phi. The large overall gains illustrate the importance of code optimization for the Xeon Phi. Note that the *conditional* sections were not vectorized on the CPU since it lacks the required vector intrinsics.

The second step is to examine the scaling within a single Xeon Phi. We test both weak and strong scaling. Due to the limited memory, we use 8 cells per thread for weak scaling in order to arrive at a maximum size of 8×224 cells for 224 threads. For strong scaling, we also use this number. Due to the excessive running time, we do not test strong scaling for less than 16 threads. Results in Figure 5 are shown as running time over 10,000 time steps. By converting the running times to cell computations per seconds, we obtain almost identical values for both weak and strong scaling. Thus, the differences between them are insignificant. When using physical cores, i.e. up to 56 threads, we observe good scaling with 86% efficiency. A likely reason for the suboptimal scaling is contention for the limited memory bandwidth among the cores. At 112 threads we use all the core clock cycles available on the machine, but efficiency drops to 65%. In addition to further memory contention, at this point two threads share one core, reducing the available cache per thread. When placing 224 threads on the device, two threads share the available core clock cycles. Thus, we do not use additional hardware (except registers) but we observe a noticeable boost in performance. While the efficiency w.r.t. the number of threads is low, efficiency relative to the actual resources used, i.e. core clock cycles is quite high at 88%. Most likely, performance is bound to a large extent by latency, not throughput, and the use of hyperthreading alleviates this problem.

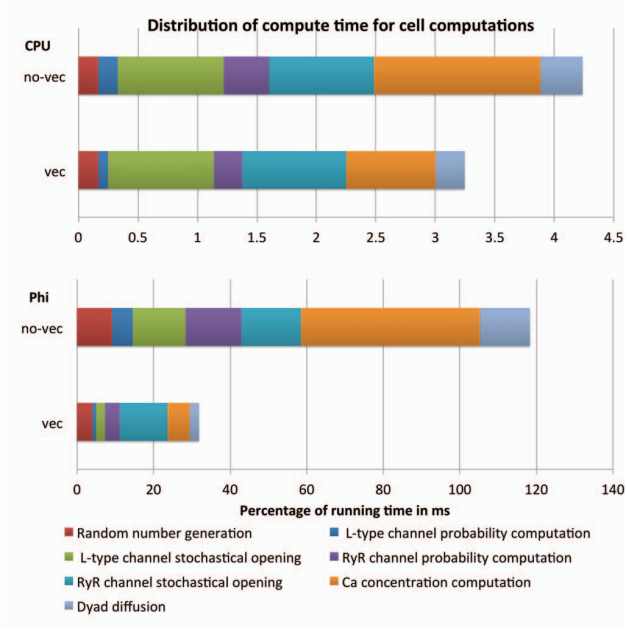

Fig. 4. Performance improvement of the individual functions in the dyad computations due vectorization. For the Xeon Phi, this also includes the conservation of random numbers. Note the change in scale due to the low single-core performance of the Xeon Phi. Performance is reported per time step, based on a measurement over 10,000 time steps.

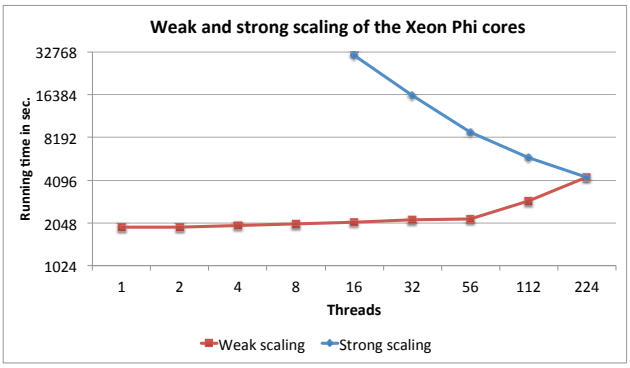

Fig. 5. Weak and strong scaling on a single Xeon Phi, shown as running time over 10,000 time steps for 8 cells per thread in weak and 1,792 cells in strong scaling.

For the Ivy Bridge CPU, we performed the same experiment, running 8 cells per core with weak scaling and 8×24 cells for strong scaling. We observe a very similar scaling behavior, except that hyperthreading is not available here. Thus, the number of cores used always equals the number of threads. Details are shown in Figure 6. Scaling is limited by memory bandwidth saturation, which does not increase linearly with the number of cores when using more than two cores per socket, i.e. four in total. As the computation is mostly

TABLE I
SINGLE AND AGGREGATE PERFORMANCE OF THE DEVICES WITHIN A NODE. CC/S IS THE NUMBER OF CELL COMPUTATIONS PER SECOND.

Device	Cells	Time (in sec.)	CC/s
2 CPUs	2,624	1,706	6,152
1 PHI	1,792	1,859	3,856
2 PHIs	3,584	1,860	7,708
3 PHIs	5,376	1,861	11,555
2 CPUs+3 PHIs	8,000	1,879	17,030

compute bound, the impact of memory bandwidth saturation is comparatively small.

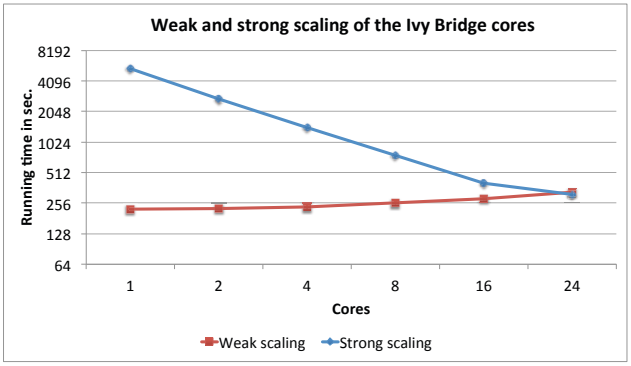

Fig. 6. Weak and strong scaling on the CPU, shown as running time over 10,000 time steps for 8 cells per thread in weak and 192 cells in strong scaling. Threads are distributed evenly over both sockets.

B. Node performance

In our third experiment we measure the performance of the devices in a tissue scale computation. The three Phis are assigned 8×224 cells each and the CPU 2624, i.e. the remainder out of a standard subdomain of $20 \times 20 \times 20$ cells. Table I shows the results. Note that we always allocate a single chunk of cells to both CPUs. Thus, there are no measurements using only one of the Ivy Bridge processors. Clearly, the performance of the Phis within one node is quite stable. However, due to technical reasons, their clock frequency varies between the nodes. Therefore, load balance on some nodes was worse than anticipated, causing the CPUs to idle.

Figure 7 illustrates the situation. While the intra node communication takes an insignificant amount of time, imperfect load balancing causes the CPU to idle at almost 10%. Clearly, this is no difficulty inherent to the code or the Xeon Phi accelerators, but it shows how susceptible the heterogeneous computation is to load balancing issues. In fact, if the CPU showed weaker than expected performance, the loss due to the Phis idling would be even greater. Note that due to the restriction to cuboid subdomains, a perfect load balancing is impossible for most node configurations.

C. Scalability

Based on the performance we achieved on one node, we tested our code on multiple nodes to assess the influence of

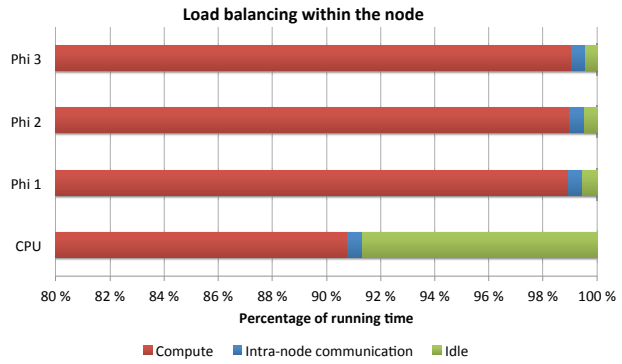

Fig. 7. Load balancing within the node. The communication overhead is insignificant, but the imperfect load balancing is noticeable. (Percentages which are not show consist of compute time for all devices.)

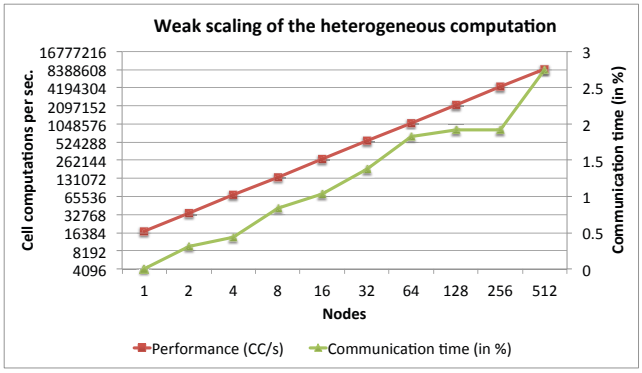

Fig. 8. Weak scaling of the overall simulation. The communication overhead remains below 3%, allowing for near perfect scaling.

MPI communication and load balance on scalability. Figure 8 shows the weak scalability from 1 to 512 nodes. We achieve near-perfect scalabilty due to low MPI communication. In addition, load balancing between the nodes has only a minor effect on performance in this experiment.

Each run uses the standard subdomains of $20 \times 20 \times 20$ cells arranged in a cuboid manner. For the 400 node run which simulates a flat slab of tissue containing $400 \times 400 \times 20$ cells, the communication overhead was slightly lower at 1.76% due to the 2D nature of the communication there. Note that due to machine limitations, communication time for the 512 node run was measured using only CPU computation and performance we extrapolated from the previous measurements.

Based on the rate of increase, we estimate that when using all 16,000 nodes, communication will still amount to less than 6% of the total running time, making this code well suited for extreme scale simulations.

VII. CARDIAC SIMULATIONS

We report on a series of cardiac tissue simulations at subcellular, cellular and tissue scales with healthy and unhealthy cells. In unhealthy cells, the RyRs are more sensitive to openings by intracellular calcium (Ca_i) compared to healthy

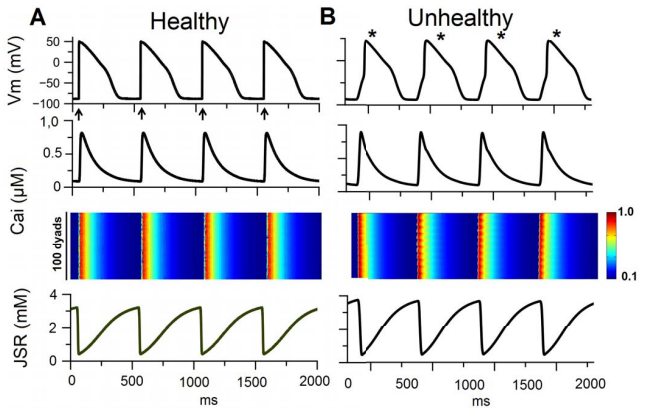

Fig. 9. Action Potential (AP) and calcium dynamics in a healthy and unhealthy cardiac cell. A.) Membrane voltage (V_m), Intracellular calcium concentration (Ca_i) and JSR calcium concentration in a healthy cell. The last 4 paced beats of the healthy cell are indicated by arrows. B.) In an unhealthy cell. Basic cycle length (BCL) = 500 ms. For the healthy tissue, K_m of RyR transition rate from C1 state to O1 state is 15 μM. For the unhealthy tissue it is 3 μM.

cells. This simulates pathological conditions in which RyR openings are sensitized to Ca_i due to genetic mutations of RyR [23], heart failure, and other cardiac myopathies [24].

A. Cell Simulations

In the first simulation (Figure 9), we run the tissue simulator with a dimension of $1\times1\times1$ thus simulating a single cell. The cell is paced to a steady-state followed by a pause. Under unhealthy conditions, triggered (non-pacing driven) action potentials (APs) (indicated by asterisks) occur after a pause. Triggered APs are known to be arrhythmogenic [25]. The most likely mechanism of triggered APs is spontaneous intracellular calcium release. This hypothesis is supported by the simulations where it is noted that spontaneous calcium release precedes triggered APs. In the healthy case, triggerd APs do not occur after a pause. We use a multi-scale cell model that utilizes realistic three-dimensional calcium diffusion between calcium release units. Each unit has several RyRs and L-type Ca channels operating stochastically. Based on these results, we explore the implications of cellular arrhythmogenesis at the tissue scale.

B. 2D Tissue Simulations

At the single cell level we observe that triggered APs occur in unhealthy cells (Figure 9). Triggered APs have been shown to be arrhythmogenic in tissue lab experiments. To determine how arrhythmias may arise at the tissue level we run a 2D (400×400 cells) tissue simulations. In Figure 10, a region of 100×100 cells in the lower left part of the tissue consists of unhealthy cells. We observe that initially triggered APs arise in this region and then propagate along the normal tissue. After some time there is a failure of repolarization in this region. The propagating wave-front meets regions of heterogeneous refractoriness and a complex pattern of conduction block, anterograde conduction and retrograde conduction follows

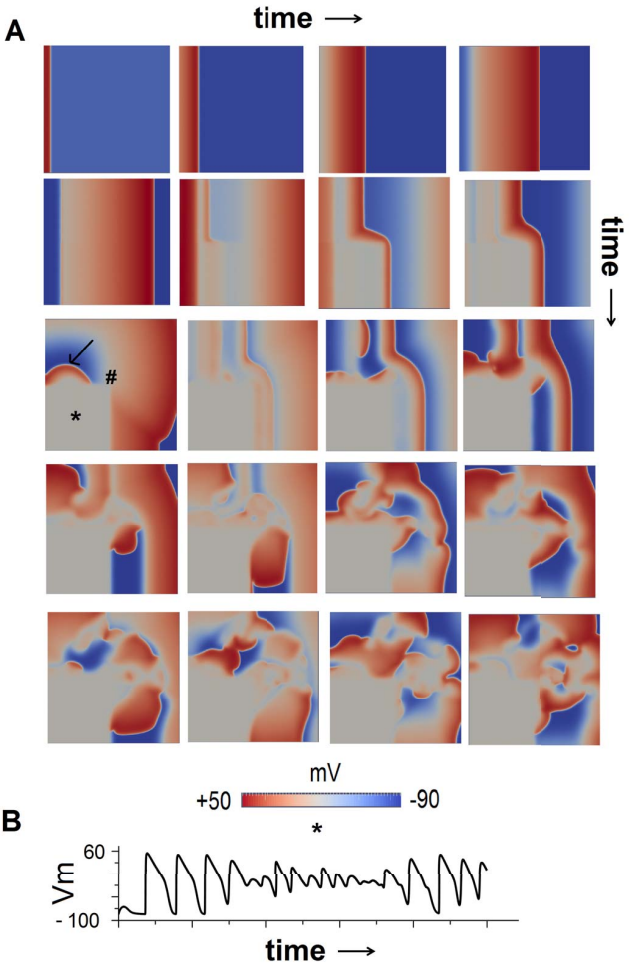

Fig. 10. Fibrillation pattern in an unhealthy tissue. The tissue consists of 400×400 cells. The lower-left part of the tissue consists of unhealthy 100×100 cells. Triggered APs arising from this region propagate into the healthy zone where we can observe complex patterns of wave-front break ups and fibrillation. The corresponding membrane potential (V_m) at the tissue center is shown at the bottom.

resulting in wave-break ups (initially indicated by arrow in panel A) and fibrillation. The corresponding V_m of the cell at the tissue center (indicated by # in panel A) shows pattern commonly observed in ECG of Torsades de Point (TdP) arrhythmias (panel B).

To understand the subcellular, cellular and ionic mechanisms of tissue-arrhythmogenesis, we observe the important ionic currents, calcium concentrations and fluxes in cells in the region of unhealthy tissue (indicated by * in Figure 10A). After an initial triggered AP, the calcium release does not terminate and there is long-lasting calcium release flux. The flux activates the Na-Ca exchange current (I_{NaCa}) which provides the depolarizing charge at the plateau potentials that balances other repolarizing currents, so that there is failure of repolarization. Note that the depolarizing charge carrier for this kind of early-afterdepolarization (EAD) is provided by I_{NaCa} and not by L-type Ca current I_{CaL} or non-

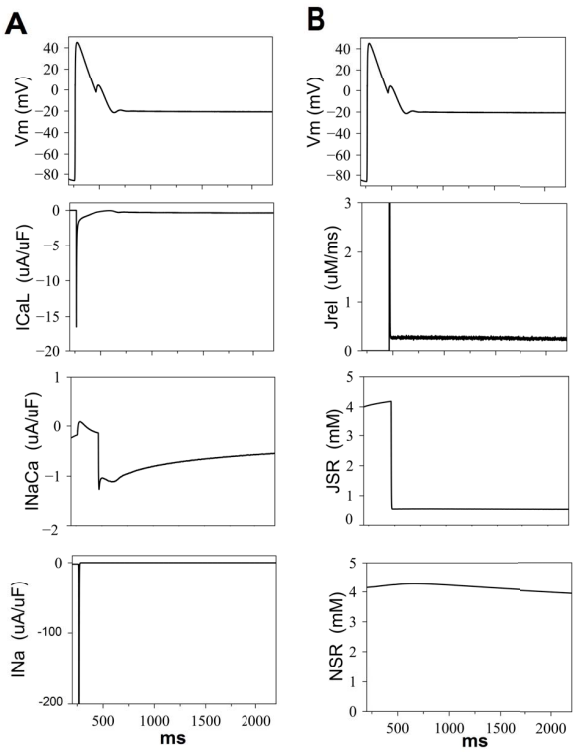

Fig. 11. Cellular mechanism of arrhythmogenesis in an unhealthy tissue. Membrane potential, important ionic currents (A) calcium release flux and calcium concentrations in junctional sarcoplasmic reticulum (JSR) and network sarcoplasmic reticulum (NSR)(B).

equilibrium reactivation of Na-current I_{Na}. The membrane potential approaches a quasi-steady value at plateau potentials providing current sources for adjoining activatable tissue as well as region of electrical refractoriness leading to wave-blocks. Such complex tissue-heterogeneities leads to formation of fibrillation patterns (Figure 10 A). Note that a significant region of the tissue must show synchronized aberrant calcium release which then leads to synchronized cellular triggered APs to overcome the source-sink mismatch at the level of the tissue. Sporadic random occurrences of aberrant calcium release and triggered APs will most likely be averaged out to provide negligible effects to wave propagation. With our choice of parameters such synchronized dysfunctional calcium release and triggered APs can indeed occur at the cellular and tissue level.

C. 3D Tissue Simulations

In order to verify these results obtained in the 2D tissue, we carried out similar simulations in a 3D tissue of size $400\times400\times20$ unhealthy cells (Figure 12). After a normal wave-front, a part of the tissue is simulated to recover prematurely from refractoriness of calcium release. This results in triggered APs in this tissue region. The triggered APs propagate and interact with normal excitation-wave fronts to form complex patterns of arrhythmias similar to the 2D case,

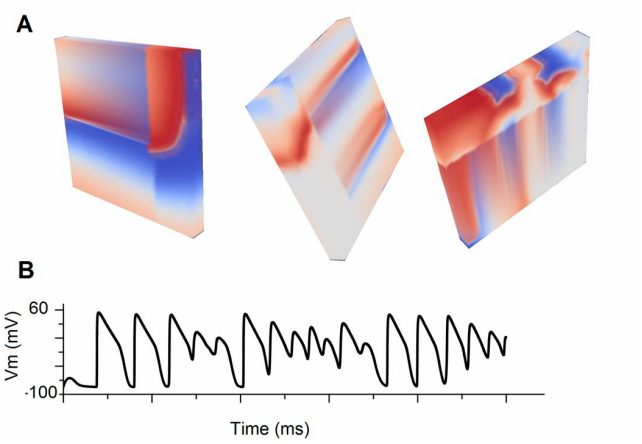

Fig. 12. A. Fibrillation patterns in a 3D unhealthy tissue. B. Corresponding membrane potential (Vm) at tissue center.

as evidenced by the corresponding membrane potential at the tissue-center.

We have seen that complex patterns of wave-break ups and fibrillations occur in tissue consisting of unhealthy cells. These arrhythmic patterns emerge from dysfunctions at the single-receptor level of stochastically operating RyRs. Due to the current machine limitations, this experiment represents the largest practical sample. Running at organ scale would require simulating almost three orders of magnitude more cells, which is beyond the capabilities of today's computers.

VIII. Conclusion

The study of arrhythmias that occur at the tissue and organ scale remains a prominent topic in computational medicine. Large scale simulations using detailed cell models can help cardiologists in the understanding of arrhythmogenesis due to deficiencies in intracellular calcium handling.

We have shown how such a simulation can be implemented to run on modern hardware accelerators, allowing for extreme scale simulations on modern supercomputers. While the attained performance with highly optimized code on a single Xeon Phi accelerator is somewhat lower than that of two Ivy Bridge CPUs, and much lower than the device capabilities would suggest, the result is in line with many other studies that outline the difficulties in unlocking the full potential of the accelerator, e.g. [26], [27]. Furthermore, the available memory per device remains a principal limitation on the size of the simulation. While it would theoretically be possible to swap date between host and accelerator memory, the 64 GB available to the host CPUs imply that this would only roughly double the simulation size since three time 8 GB from the Xeon Phis plus a corresponding 12 GB for the CPU already add up to an additional 36 GB to be stored on the host.

However, future supercomputers such as NERSC Cori and other machines in the CORAL initiative [28] will use the new Knight's Landing (KNL) generation of Xeon Phi, which has a much larger memory capacity. Cori will be equipped with

$96 + 16$ GB of memory per device. With 9,300 such devices, this is not enough to simulate a whole heart. However, the upcoming Aurora supercomputer with more than 50,000 nodes will be sufficient. Thus, running our simulations at organ scale will be within reach in the near future.

While performing a large number of smaller code optimizations, we found that in addition to algorithmic improvements such as using binomial distributions and saving random numbers, the largest gains come from judicious use of both manual and automatic vectorization. Furthermore, correct use of nontrivial OpenMP parallelization plays a major role in obtaining high performance for complex manycore codes. These results on effective Xeon Phi performance optimization also apply to many other scientific codes. In addition, our results suggest that due to the complexity of the code, performance is partially latency bound here, which implies that hyperthreading provides extra performance at no additional cost, which is usually not the case in simpler compute kernels.

For the heterogeneous computation, we selected a hierarchical domain decomposition which creates one subdomain for each node. The subdomains are further divided among the devices within the nodes. From a programming point of view, in addition to the benefits of more flexible load balancing, this has the advantage that the MPI communication is completely separate from the intra-node computation, making it easy to adapt the code to other supercomputers with different node structures.

In the course of this work, we performed preliminary experiments using GPUs as accelerators, which resulted in a performance superior to that of the CPU with only moderate optimizations. Thus, our future work will study the attainable performance on Pascal GPUs and Knight's Landing Xeon Phis.

REFERENCES

[1] J. G. Restrepo, J. N. Weiss, and A. Karma, "Calsequestrin-mediated mechanism for cellular calcium transient alternans," *Biophysical Journal*, vol. 95, no. 8, pp. 3767–3789, 2008.

[2] N. Gaur and Y. Rudy, "Multiscale modeling of calcium cycling in cardiac ventricular myocyte: macroscopic consequences of microscopic dyadic function," *Biophysical Journal*, vol. 100, no. 12, pp. 2904–2912, 2011.

[3] M. Nivala, E. de Lange, R. Rovetti, and Z. Qu, "Computational modeling and numerical methods for spatiotemporal calcium cycling in ventricular myocytes," *Frontiers in Physiology*, vol. 3, p. 114, 2012.

[4] G. S. Williams, A. C. Chikando, H.-T. M. Tuan, E. A. Sobie, W. Lederer, and M. S. Jafri, "Dynamics of calcium sparks and calcium leak in the heart," *Biophysical Journal*, vol. 101, no. 6, pp. 1287–1296, 2011.

[5] Z. Song, C. Y. Ko, M. Nivala, J. N. Weiss, and Z. Qu, "Calcium-voltage coupling in the genesis of early and delayed afterdepolarizations in cardiac myocytes," *Biophysical Journal*, vol. 108, no. 8, pp. 1908–1921, 2015.

[6] M. Nivala and Z. Qu, "Calcium alternans in a couplon network model of ventricular myocytes: role of sarcoplasmic reticulum load," *American Journal of Physiology – Heart and Circulatory Physiology*, vol. 303, no. 3, pp. H341–H352, 2012.

[7] M. Nivala, Z. Song, J. N. Weiss, and Z. Qu, "T-tubule disruption promotes calcium alternans in failing ventricular myocytes: Mechanistic insights from computational modeling," *Journal of Molecular and Cellular Cardiology*, vol. 79, pp. 32–41, 2015.

[8] C. Adler and U. Costabel, "Cell number in human heart in atrophy, hypertrophy, and under the influence of cytostatics," *Recent Advances in Studies on Cardiac Structure and Metabolism*, vol. 6, pp. 343–355, 1975.

[9] H. Cheng, W. Lederer, and M. B. Cannell, "Calcium sparks: elementary events underlying excitation-contraction coupling in heart muscle," *Science*, vol. 262, no. 5134, pp. 740–744, 1993.

[10] Q. Lan, N. Gaur, J. Langguth, and X. Cai, "Towards detailed tissue-scale 3d simulations of electrical activity and calcium handling in the human cardiac ventricle," in *Algorithms and Architectures for Parallel Processing - 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015. Proceedings, Part III*, ser. Lecture Notes in Computer Science, G. Wang, A. Y. Zomaya, G. M. Pérez, and K. Li, Eds., vol. 9530. Springer, 2015, pp. 79–92.

[11] Z. Song, C. Y. Ko, M. Nivala, J. N. Weiss, and Z. Qu, "Calcium-voltage coupling in the genesis of early and delayed afterdepolarizations in cardiac myocytes," *Biophysical journal*, vol. 108, no. 8, pp. 1908–1921, 2015.

[12] "Tianhe-2 (Milky Way-2) Supercomputer," http://www.tianhe2.org.

[13] "Top500 Supercomputing Sites," http://www.top500.org.

[14] "Intel Xeon Phi coprocessor peak theoretical maximums," http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-theoretical-maximums.html.

[15] X. Dong, M. Wen, J. Chai, X. Cai, M. Zhao, and C. Zhang, "Communication-hiding programming for clusters with multi-coprocessor nodes," *Concurrency and Computation: Practice and Experience*, vol. 27, pp. 4172–4185, 05 2015. [Online]. Available: http://dx.doi.org/10.1002/cpe.3507

[16] J. Fang, H. Sips, L. Zhang, C. Xu, Y. Che, and A. L. Varbanescu, "Test-driving Intel Xeon Phi," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '14. ACM, 2014, pp. 137–148. [Online]. Available: http://doi.acm.org/10.1145/2568088.2576799

[17] T. O'Hara, L. Virág, A. Varró, and Y. Rudy, "Simulation of the undiseased human cardiac ventricular action potential: model formulation and experimental validation," *PLoS Computational Biology*, vol. 7, no. 5, p. e1002061, 2011.

[18] Z. Qu and A. Garfinkel, "An advanced algorithm for solving partial differential equation in cardiac conduction," *IEEE Transactions on Biomedical Engineering*, vol. 46, no. 9, pp. 1166–1168, 1999.

[19] MPICH, "High-performance portable MPI," https://www.mpich.org.

[20] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming*, 1st ed. Waltham, MA, USA: Morgan Kaufmann Publishers Inc., 2013.

[21] "Intel Math Kernel Library – Documentation," https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation, 2015.

[22] A. Vladimirov, "Arithmetics on Intel's Sandy Bridge and Westmere CPUs: not all FLOPs are created equal," Colfax International, Tech. Rep., 2012.

[23] N. Liu, B. Colombi, M. Memmi, S. Zissimopoulos, N. Rizzi, S. Negri, M. Imbriani, C. Napolitano, F. A. Lai, and S. G. Priori, "Arrhythmogenesis in catecholaminergic polymorphic ventricular tachycardia insights from a RyR2 R4496C knock-in mouse model," *Circulation Research*, vol. 99, no. 3, pp. 292–298, 2006.

[24] M. Yano, T. Yamamoto, Y. Ikeda, and M. Matsuzaki, "Mechanisms of disease: ryanodine receptor defects in heart failure and fatal arrhythmia," *Nature Clinical Practice Cardiovascular Medicine*, vol. 3, no. 1, pp. 43–52, 2006.

[25] B. F. Hoffman and M. R. Rosen, "Cellular mechanisms for cardiac arrhythmias." *Circulation research*, vol. 49, no. 1, pp. 1–15, 1981.

[26] G. Crimi, F. Mantovani, M. Pivanti, S. Schifano, and R. Tripiccione, "Early experience on porting and running a lattice boltzmann code on the xeon-phi co-processor," *Procedia Computer Science*, vol. 18, pp. 551 – 560, 2013, 2013 International Conference on Computational Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913003621

[27] I. E. Venetis, G. Goumas, M. Geveler, and D. Ribbrock, "Porting FEASTFLOW to the intel xeon phi: Lessons learned," Partnership for Advanced Computing in Europe (PRACE), Tech. Rep., 2014.

[28] R. Brueckner, "A closer look at Intel's Coral supercomputers coming to Argonne," http://insidehpc.com/2015/04/intel-build-coral-supercomputers-argonne-200-procurement/, 2015.