Towards simulation of subcellular calcium dynamics at nanometre resolution

Jun Chai¹, Johan Hake², Nan Wu¹,², Mei Wen¹, Xing Cai²,³, Glenn T Lines²,³,⁴, Jing Yang¹, Huayou Su¹,²,³, Chunyuan Zhang¹ and Xiangke Liao¹

Abstract
Numerical simulation of subcellular Ca²⁺ dynamics with a resolution down to one nanometre can be an important tool for discovering the physiological cause of many heart diseases. The requirement of enormous computational power, however, has made such simulations prohibitive so far. By using up to 12,288 Intel Xeon Phi 31S1P coprocessors on the new hybrid cluster Tianhe-2, which is the new number one supercomputer of the world, we have achieved 1.27 Pflop/s in double precision, which brings us much closer to the nanometre resolution. This is the result of efficiently using the hardware on different levels: (1) a single Xeon Phi (2) a single compute node that consists of a host and three coprocessors, and (3) a huge number of interconnected nodes. To overcome the challenge of programming Intel’s new many-integrated core (MIC) architecture, we have adopted techniques such as vectorization, hierarchical data blocking, register data reuse, off-loading computations to the coprocessors, and pipelining computations with intra-/inter-node communications.

Keywords
numerical simulation, subcellular Ca²⁺ dynamics, nano-resolution, hybrid cluster, Tianhe-2, Intel Xeon Phi.

1. Introduction
The main function of the heart is to supply the body with blood. At the cellular level, heart contraction is triggered by an increase in the intracellular calcium (Ca²⁺) concentration. In each heart cell, Ca²⁺ is released from over 10,000 Ca²⁺ release units (CRUs). The release is simultaneous and stable in healthy heart cells. In failing cells, however, Ca²⁺ release is dysynchronous, slower and less reliable, with the possibility of triggering off-beat Ca²⁺ releases that under unfortunate circumstances can be arrhythmogenic and potentially lethal.

A truly multi-scale model of Ca²⁺ release, which can give a physiologically accurate description of both healthy and pathological Ca²⁺ releases, has remained elusive. Current research activities either cut back on the details or use a very small spatial domain. One of the main obstacles is the enormous computational requirement. For instance, to resolve the nanometre scale of the CRUs within a single sarcomere, which occupies a 3D volume of 10 × 10 × 2 μm³, would require 2 × 10¹¹ voxels, each 1 μm³ in volume. The sheer number of voxels often comes together with a huge number of required time steps. For example, a total number of floating-point operations on the order of 10¹⁹ may be needed to simulate one sarcomere over 1 ms, thus making nanometre-resolution simulations of subcellular Ca²⁺ dynamics extremely challenging.

We thus initiate a collaborative research effort in this paper that aims to solve this computational challenge, facilitated by a massive-scale cluster of Xeon Phi coprocessors. Apart from shedding light on the realism of existing mathematical models of Ca²⁺ channel dynamics and subcellular Ca²⁺ diffusion, the main focus of this paper will be on meeting the new programming and performance challenges associated with Intel’s many-integrated core (MIC) architecture.

2. Problem description
A single Ca²⁺ release event is controlled by a stochastic process at the CRUs, each being defined by the juxtaposition between two geometrical structures: the t-tubule and the sarcoplasmic reticulum (SR); see Figure 1. During a normal heartbeat, Ca²⁺ enters the cell through the L-type

¹ Computer College, National University of Defense Technology, China
² Simula Research Laboratory, Norway
³ Department of Informatics, University of Oslo, Norway
⁴ Center for Cardiological Innovation, Oslo University Hospital, Norway

Corresponding author:
Jun Chai, Computer College, National University of Defense Technology, Changsha, Hunan 410073, China.
Email: chajun200306@nudt.edu.cn
Ca\(^{2+}\) channels positioned at the t-tubule, and then diffuses to the ryanodine receptor (RyR), where Ca\(^{2+}\) triggers further release from the SR. The t-tubules are invaginations of the outer membrane and are predominantly positioned at the so-called Z-lines, thus pinpointing the CRUs there. Stochastic Ca\(^{2+}\) release can also occur between heartbeats. However, such Ca\(^{2+}\) releases lack the synchronized triggering of Ca\(^{2+}\) from the L-type channel, and are thus much less frequent and also not spatially coordinated. These spontaneous Ca\(^{2+}\) releases, called Ca\(^{2+}\) sparks, play an important role in setting the overall Ca\(^{2+}\) sensitivity of the cell by regulating the intracellular Ca\(^{2+}\) level between beats (Bers, 2001). Individual Ca\(^{2+}\) sparks can under special conditions evolve into whole cell Ca\(^{2+}\) waves, which traverse the entire cell and flood the intracellular space with Ca\(^{2+}\). These off-beat whole cell Ca\(^{2+}\) releases can be arrhythmogenic and potentially lethal (Cheng et al., 1996, Pogwizd et al., 1998).

Computational models have been extensively used to better understand the complex dynamics of Ca\(^{2+}\) signalling and Ca\(^{2+}\) wave generation; see Izu et al. (2013) for a review. However, a fundamental challenge has been the hugely different length scales involved. The cleft between the t-tubule and the SR is in the range of 10 nm (Franzini-Armstrong et al. 1999; Hayashi et al., 2009) and the channel mouth of a single RyR is as small as 1 nm (Serysheva et al. 2005). The cellular scale stretches from 10 to 100 \(\mu\)m. The state-of-the-art models of intracellular Ca\(^{2+}\) wave generation compromise details for whole-cell Ca\(^{2+}\) dynamics (Nivala et al. 2012). Instead of resolving the Ca\(^{2+}\) gradients inside each CRU, they use a lumped Ca\(^{2+}\) concentration. With the help of supercomputing on cutting-edge hardware, we aim to resolve the details of a single CRU at the nanometre scale, together with sources for individual RyRs, and combine these with inter-CRU Ca\(^{2+}\) diffusion to study the possibilities for Ca\(^{2+}\) waves to occur.

2.1. Mathematical model

Subcellular Ca\(^{2+}\) dynamics are modelled by a coupled system of nonlinear partial differential equations (PDEs) of the reaction–diffusion type. Our mathematical model for this paper consists of five reaction–diffusion equations and two ordinary differential equations. The choice of reaction–diffusion PDEs is similar to a previously used model (Hake et al., 2012):

\[
\begin{align*}
\frac{\partial c}{\partial t} &= D_{c}^{\text{cyt}} \nabla^2 c + R_{\text{SR}}(c, c^{\text{ATP}}) - \sum R_i(c, c^{B_i}) \\
\frac{\partial c^{\text{ATP}}}{\partial t} &= D_{c}^{\text{cyt}} \nabla^2 c^{\text{ATP}} - \frac{R_{\text{SR}}(c, c^{\text{ATP}})}{\gamma} - R_{\text{CSQN}}(c^{\text{SR}}, c^{\text{CSQN}}) \\
\frac{\partial R_{\text{RyR}}}{\partial t} &= D_{X}^{\text{ATP}} \nabla^2 c^{\text{ATP}} + R_{\text{ATP}}(c, c^{\text{ATP}}) \\
\frac{\partial R_{\text{CMDN}}}{\partial t} &= D_{X}^{\text{CMDN}} \nabla^2 c^{\text{CMDN}} + R_{\text{CMDN}}(c, c^{\text{CMDN}}) \\
\frac{\partial R_{\text{CSQN}}}{\partial t} &= D_{X}^{\text{CSQN}} \nabla^2 c^{\text{CSQN}} + R_{\text{CSQN}}(c^{\text{CSQN}}) \\
\end{align*}
\]

where the seven primary unknowns are the Ca\(^{2+}\) concentrations inside the cytosol and the SR (\(c\) and \(c^{\text{ATP}}\)), and four additional Ca\(^{2+}\) buffer concentrations (\(c^{\text{ATP}}, c^{\text{CMDN}}, c^{\text{CSQN}},\) and \(c^{\text{CSQN}}\)), together with the Ca\(^{2+}\) concentration \(c^{\text{CSQN}}\) that only lives inside regions of calsequestrin (CSQN).

The D constants in the five diffusion \(\nabla^2\)-terms represent the respective diffusion properties. The \(\gamma\) constant in the second equation denotes the SR volume fraction. The different \(R(\cdot, \cdot)\) functions represent the reaction terms, which can all be expressed by the following mathematical formula:

\[R_i(c, c^{B_i}) = k_{\text{on}}^{c} c^{B_i} (c^{B_i} - c^{B_i}) - k_{\text{off}}^{c} c^{B_i}\]

with \(k_{\text{on}}^{c}, B_{\text{tot}}^{B_i}\) and \(k_{\text{off}}^{c}\) being known constants taken from literature. The only exception is \(R_{\text{SR}}(c, c^{\text{ATP}})\), which consists of two components,

\[R_{\text{SR}}(c, c^{\text{ATP}}) = R_{\text{RyR}}(c, c^{\text{ATP}}) - R_{\text{serca}}(c, c^{\text{ATP}})\]

The first term represents the release current through the RyRs and is given as
where \( P_o \) is a binary variable which is zero in voxels where there is no channel. It is also zero for voxels containing a closed RyR and one only if the channel is open, and \( k \) is the maximum conductance of the channel. The second term of \( R_{SR} \) represents the SERCA pump and is of the following form:

\[
R_{	ext{serca}}(c, c^{sr}) = \mu \left( t \right) \frac{c^{sr} \cdot c}{c^{esr} \cdot c^{sr} + c}.
\]

where \( a_1, a_2, a_3, a_4, a_5 \) are prescribed constants taken from Tran et al. (2009) and Hake et al. (2012).

In the first reaction–diffusion equation that models \( c \), the subscript \( i \) in the summation term covers ATP, CMDN, Fluor, and TRPN.

The RyRs are distributed inside the CSQN regions, each having a \( z \)-thickness of 30 \( \mu \)m and being distributed along the Z-line. Figure 2 shows an example of RyR and CSQN distributions in the plane of the Z-line. Values of \( c^{CSQN} \) evolve inside the CSQN regions and are governed by the \( R_{	ext{CSQN}}(c^{esr}, c^{CSQN}) \) function. For each RyR, \( P_o \) is governed stochastically using a reparameterized four-state Markov model based on Stern et al. (1999). The Markov model is solved by a Monte Carlo method. For each RyR, a random number \( R \) is generated at each time step and compared with the total escape rate \( \gamma \) of the present state. If \( R < 1 - e^{-\Delta t \gamma} \), a transition occurs. By choosing a fully explicit numerical strategy (see below) the small value of \( \Delta t \) lets us approximate \( 1 - e^{-\Delta t \gamma} \) with \( \Delta t \gamma \).

No-flow boundary conditions are applied to the entire boundary of the 3D computational domain. Moreover, no-flow conditions are also enforced around each CSQN region (except the RyRs), as well as along each dyadic cleft, creating a confined region for the released \( \text{Ca}^{2+} \) (see Figure 1).

### 2.2. Numerical method

The 3D solution domain is uniformly discretized into a 3D mesh of voxels of size \((\Delta x, \Delta y, \Delta z)\). Numerical solutions are sought at the centre of each voxel, and central finite differences are adopted to discretize the diffusion terms, giving rise to a standard seven-point computational stencil. The diffusion and reaction terms are numerically treated separately, both using a forward Euler discretization in the temporal direction. Although the time step size \( \Delta t \) of this fully explicit numerical strategy has to be chosen to be very small to ensure numerical stability, the computational work per time step is dramatically cheaper than a fully implicit scheme that would have required solving a coupled system of nonlinear algebraic equations per time step. Another advantage of the fully explicit scheme is its good parallel scalability, because only nearest-neighbour communication is needed.

### 3. Target architecture

In the search for the tremendous computing power required, we have resorted to using a cutting-edge supercomputer, named Tianhe-2, developed by the National University of Defense Technology in China. Tianhe-2 has 16,000 compute nodes, each with two Intel Xeon IvyBridge processors and three Xeon Phi coprocessors, and thus a combined total of 3,120,000 computing cores, while also being the largest installation of Intel Xeon

---

**Figure 2.** Typical RyR distribution (left) and CSQN distribution (right) for a healthy myocyte. The distributions were generated using a modified version of the simple growth algorithm presented in Baddeley et al. (2009).
calls from the host CPU to invoke the coprocessors via kernel functions. The increased complexity of programming, requiring the extra coding effort for using the coprocessors, but implies for careful consideration of both programming and performance optimization. While MPI is the obvious choice for inter-node communication, one layer of halo voxels wraps around the subdomain mesh. In total, 12 3D arrays of dimension \((N_x^{\text{subd}} + 2) \times (N_y^{\text{subd}} + 2) \times (N_z^{\text{subd}} + 2)\) are allocated per subdomain for storing single-time-step values of the seven Ca\textsuperscript{2+} concentration fields, \(c, c^{\text{ATP}}, c^{\text{CMDN}}, c^{\text{Ecm}}, c^{\text{EcmPy}}, c^{\text{CSQN}}\). Note that the first five fields require one additional array each for numerically computing their respective diffusion terms. Each 3D array has its shortest stride in the \(x\)-direction, and the largest stride in the \(z\)-direction.

The entire simulation is done through time-stepping, with the following computational tasks per time step:

- **Diffusion computation**: five Ca\textsuperscript{2+} concentration fields, \(c, c^{\text{ATP}}, c^{\text{CMDN}}, c^{\text{Ecm}}, c^{\text{EcmPy}}, c^{\text{CSQN}}\), are updated at each voxel, by applying a seven-point computational stencil to values from the previous time step. For example, the diffusion computation for field \(c\) is

\[
c^{c+1}_{i,j,k} = c^{c}_{i,j,k} + \frac{\Delta t}{\Delta x^2} \left( c^{c}_{i-1,j,k} - 2c^{c}_{i,j,k} + c^{c}_{i+1,j,k} \right) + \frac{\Delta t}{\Delta y^2} \left( c^{c}_{i,j-1,k} - 2c^{c}_{i,j,k} + c^{c}_{i,j+1,k} \right) + \frac{\Delta t}{\Delta z^2} \left( c^{c}_{i,j,k-1} - 2c^{c}_{i,j,k} + c^{c}_{i,j,k+1} \right)
\]

where the superscript denotes the time level and the subscripts refer to 3D voxel indices. The constant \(\frac{\Delta t}{\Delta x^2}\) takes the value of \((D_{\text{Ca}}^{\text{ATP}}\Delta t) / \Delta x^2\), and constants \(\frac{\Delta t}{\Delta y^2}\) and \(\frac{\Delta t}{\Delta z^2}\) are defined similarly.

- **Reaction computation**: the five newly calculated fields, plus also the \(c^{\text{EcmPy}}\) field, are further updated at each voxel using all their respective reaction terms, except for the \(R_{\text{CSQN}}\) term.

- **CSQN computation**: inside all CSQN regions, which only occupy a very small portion of the entire 3D solution domain, \(c^{\text{Ecm}}\) and \(c^{\text{CSQN}}\) are further updated using the \(R_{\text{CSQN}}\) reaction term.

- **Ca\textsuperscript{2+} channel handling**: each RyR is then subject to a stochastic Markov-chain computation deciding
whether Ca\(^{2+}\) release happens or not. If yes, the values of \(c\) and \(c^{\text{sr}}\) are re-calculated at the corresponding voxel.

The computations per subdomain are divided between the host CPU and the three coprocessors, such that each coprocessor is responsible for a 3D subpiece of the subdomain. During each time step, using the offload mode, the host CPU invokes the three coprocessors to each compute within their respective subpieces. In addition to computing the remaining 3D volume exterior to the three subpieces, the host CPU is also responsible for all the necessary intra- and inter-node data movement. Details are to be presented in Sections 4.3 and 4.4.

Aiming at resource-efficient utilization of Tianhe-2, whose main computing power draws from the new MIC architecture, we have prioritized our coding effort in three aspects: single-coprocessor utilization, single-node utilization, and multi-node efficiency.

### 4.2. Single-coprocessor utilization

Let us denote by \(N_c^{\text{coproc}} \times N_y^{\text{coproc}} \times N_z^{\text{coproc}}\) the 3D voxel block that the host CPU assigns to each coprocessor. Twelve 3D arrays of dimension \((N_c^{\text{coproc}} + 2) \times (N_y^{\text{coproc}} + 2) \times (N_z^{\text{coproc}} + 2)\) are thus allocated on the coprocessor, and the fastest-changing index direction coincides with \(x\). The computations typically traverse triply nested for-loops, for which the three loop indices go from 1 to \(N_c^{\text{coproc}}\), \(N_y^{\text{coproc}}\), and \(N_z^{\text{coproc}}\), respectively. OpenMP pragma omp for collapse(2) is used to parallelize each triple loop nest. Here, we remark that only 56 (out of 57) coprocessor cores can participate in the computations when the offload mode is chosen. The last core is not accessible in offload mode, because it is reserved for running the coprocessor’s micro operating system. In connection with the Monte Carlo method for solving the Markov model related to each RyR (see Section 2.1), we have made sure that the seed for each OpenMP thread’s random number generator is uniquely determined by the computer node ID, the local coprocessor ID, and OpenMP thread ID.

The two main computational tasks, diffusion and reaction, can be most conveniently implemented as ten separate triply nested for-loops. That is, the first five loops handle the diffusion computation for, respectively, \(c\), \(c^{\text{sr}}\), \(c^{\text{ATP}}\), \(c^{\text{CMDN}}\), and \(c^{\text{Fluo}}\). The next five loops handle the reaction computation for, respectively, \(c\), \(c^{\text{ATP}}\), \(c^{\text{CMDN}}\), \(c^{\text{Fluo}}\), and \(c^{\text{TRPN}}\). Note that the \(c\) field is also jointly updated during each of the latter five loops.

The above straightforward implementation, using ten separate loop nests, is far from optimal. The reason is poor data reuse in the caches. We have therefore adopted two loop transformation techniques: fusion and hierarchical blocking, which give an improved code outline as follows:

```c
#pragma omp for collapse(2) private(j1,j2,j3)
for (k=1; k<=Nz_coproc; k++)
for (j1=1; j1<=Ny_coproc; j1++)
for (j2=1; j2<=Nx_coproc; j2++)
/* diffusion for c, one voxel */
for (j1=1; j1<block_size1; j1++)
for (j2=1; j2<block_size2; j2++)
/* diffusion for c^{sr}, one voxel */
for (j2=1; j2<block_size2; j2++)
/* reaction for c^{sr} & c, one voxel */
for (j2=1; j2<block_size2; j2++)
/* diffusion for c^{ATP}, one voxel */
for (j2=1; j2<block_size2; j2++)
/* reaction for c^{ATP} & c, one voxel */
for (j2=1; j2<block_size2; j2++)
/* diffusion for c^{CMDN}, one voxel */
for (j2=1; j2<block_size2; j2++)
/* reaction for c^{CMDN} & c, one voxel */
for (j2=1; j2<block_size2; j2++)
/* diffusion for c^{Fluo}, one voxel */
for (j2=1; j2<block_size2; j2++)
/* reaction for c^{Fluo} & c, one voxel */
for (j2=1; j2<block_size2; j2++)
/* diffusion for c^{TRPN}, one voxel */
for (j2=1; j2<block_size2; j2++)
/* reaction for c^{TRPN} & c, one voxel */
```

Looking at the above code sketch, we can see that the original ten separate loop nests are fused into a single loop nest that now has five levels. Two new loop levels are introduced for the \(y\)-direction, because the \(z\)-direction is reserved for OpenMP parallelization and the \(x\)-direction is reserved for vectorization (discussed later). The first new loop level, which uses the \(j1\) index, arises from loop blocking and aims to improve data reuse in the L2 cache, where the value of block_size1 is calculated by

\[
\text{Size of 12 cache per core} = \frac{11}{\# \text{threads per core}} \times (N_c^{\text{coproc}} + 2) \times 8 \text{ bytes}
\]

The reason for having a factor of 11 in the denominator is that data from all 12 3D arrays, except \(c^{\text{RCSP}}\), will be used in the computations. The other new loop level that uses the \(j2\) index targets the L1 cache. The value of block_size2 can be calculated similarly to in the above formula, using the size of L1 cache instead of L2 cache. Another very important factor is the order in which the different computations are carried out in the \(j1\) loop. This is to ensure that new results from a diffusion computation can immediately participate in a corresponding reaction computation. We also remark that optimization in the form of data tiling in both the \(x\)- and \(y\)-directions, as suggested in Rivera and Tseng (2000), will not work for our case. This is because the \(x\)-direction should not be accessed in a strided fashion for the reason of losing the effect of code vectorization.
Nevertheless, the fused loop nest above still has one major weakness in not using the vectorization capability of the Xeon Phi coprocessor. Although our computations are memory-bandwidth bounded, experiments have shown that code vectorization is an extremely important performance booster. For this purpose, all the for-loops with the \( i \) index should not compute voxel by voxel, but eight by eight voxels instead. This is because Xeon Phi’s 512-bit vector operations can handle eight double-precision values at a time. Although the SIMD pragma and auto-vectorization by compiler can be used, we have chosen to manually insert \texttt{mm512}_xxx intrinsics for better performance. Two associated performance-enhancing techniques are (1) aligning the start of each array row on a 64-byte boundary, and (2) reusing data in the vector registers via explicit programming.

Other factors can also affect the OpenMP performance on the Xeon Phi coprocessor. We have always spawned four OpenMP threads per coprocessor core, and adopted thread binding, first-touch, and explicit data prefetch via pragmas and compiler options.

4.3. Single-node utilization

We have mentioned that the offload mode is chosen to avoid slow MPI communication that directly involves coprocessors. Another advantage is the flexibility in assigning computational work to the host CPU. While the sub-pieces assigned to each coprocessor is always a 3D block of voxels, this does not have to be the case for the host CPU. For example, the host CPU can be assigned to handle a thin or thick layer of voxels wrapping around the 3D block of a coprocessor.

Compared with the typical hardware configuration where each compute node has one coprocessor (e.g., the Stampede machine), Tianhe-2’s choice of three coprocessors per node requires a couple of extra considerations. First, how shall the three coprocessors be placed within each subdomain? Experiments indicate that it is best to place the three coprocessors side by side in the \( y \)-direction. The second consideration concerns the needed data exchanges between the coprocessors, which lie side by side. Although direct data movement between two intra-node coprocessors is possible to program, we have, for efficiency, used the host CPU to relay the data. To summarize, the host CPU has three tasks per time step:

1. Invoke computations in the three coprocessors.
2. Carry out its own computation, if any, in the remaining region of the subdomain.
3. Invoke host-MIC and MIC-host data exchanges.

It is important to notice that each of the above tasks involves sub-tasks. For example, the computational region of the host CPU consists of up to six parts. Thus, it pays off to dedicate a small number of OpenMP threads to the non-computational sub-tasks, while the remaining OpenMP threads carry out the computation.

4.4. Multi-node efficiency

In the case of multiple nodes working together through MPI communication, each host CPU has the additional task of exchanging halo data with up to six neighbouring subdomains. Therefore, in order to hide the inter-node communication overhead, we have adopted a pipelined execution of the different sub-tasks of the host CPU. A similar pipelining approach can be found in Shimokawabe et al. (2011) and Wen et al. (2012).

Figure 3 illustrates a situation where there are six neighbouring subdomains: left, right (\( x \)-direction), front, back (\( y \)-direction), top, bottom (\( z \)-direction). To help relieve the burden on the host CPU, computations along the ‘front’ and ‘back’ subdomain boundaries are done by MIC\(_0\) and MIC\(_2\), respectively, which upon completion immediately pass them to the host CPU to initiate non-blocking MPI communication with the ‘front’ and ‘back’ neighbours. These sub-tasks are intermingled with the other sub-tasks of the host CPU, that is, computing and communicating on the other
four subdomain boundaries, one by one. Moreover, the host CPU also pipelines the various intra-node data exchanges.

5. Performance studies

All the following numerical experiments used double precision, and the Intel compiler icc v13.0.0 was used.

5.1. Single-coprocessor performance

Figure 4 shows the performance obtained on a single Xeon Phi 31S1P coprocessor, using 224 OpenMP threads (divided among 56 cores). The Gflop/s rates are calculated on the basis that each voxel needs 150 floating-point operations per step time, which is counted by the PAPI tool. Two problem sizes were tested, one with $62 \times 24 \times 112$ voxels, the other with $142 \times 400 \times 112$ voxels. The first problem size was chosen such that the required amount of memory (16 MB) can be satisfied by aggregating all the L2 caches, whereas the second problem size (640 MB) greatly exceeds the L2 capacity. The worst performance in Figure 4 was associated with using ten separate triply nested loops, with neither thread binding, nor loop blocking or vectorization, whereas the best performance was the result of using all the optimizations mentioned in Section 4.2.

Figure 4. The impact of different optimizations on single-coprocessor performance. TLB denotes translation lookaside buffer.

Notice that the best single-coprocessor performance was 118 Gflop/s for the ‘beyond-L2’ test problem. Although it only reaches 11.8% of the theoretical peak double-precision capability of the coprocessor, a couple of comments are in order here. From the angle of obtained memory bandwidth, 118 Gflop/s translates to

$$\frac{118 \text{ Gflop/s} \times (6\text{ writes} + 16\text{ reads}) \times 8 \text{ bytes}}{150 \text{ flop}} = 138 \text{ GB/s}$$

This is because there are precisely six writes per voxel of, respectively, $c$, $c^{\text{Fluo}}$, $c^{\text{ATP}}$, $c^{\text{CMDN}}$, $c^{\text{TRPN}}$, and $c^{\text{IB}}$ to the coprocessor’s DRAM. The number of memory reads includes one for $c^{\text{TRPN}}$ and three for each of the five diffusion terms, because the L1 and L2 caches can support data reuse along the x- and y-axes, but there is no data reuse in the z-direction. (Notice that $N_{\text{coproc}} = 112$, so each of the 224 OpenMP threads is only assigned to compute over a half $x_1y_1$-plane.)

Compared with the reported copy rate of the STREAM benchmark at around 150 GB/s (Williams et al., 2012), our achieved memory bandwidth of 138 GB/s has reached more than 90% of the realistic memory bandwidth limit.

5.2. Single-node performance

A strong scaling study was done within a single node, which used one, two, and three coprocessors. The host CPU was configured to do no computation of its own, but was responsible for the relay of data exchanges between the coprocessors. The global mesh size was fixed at $142 \times 1200 \times 112$ voxels and we measured the time usage of 1000 time steps, as reported in Table 2.

<table>
<thead>
<tr>
<th># MICs</th>
<th>Mesh per MIC</th>
<th>Time usage</th>
<th>Gflop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$142 \times 1200 \times 112$</td>
<td>25.86 s</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>$142 \times 600 \times 112$</td>
<td>12.68 s</td>
<td>226</td>
</tr>
<tr>
<td>3</td>
<td>$142 \times 400 \times 112$</td>
<td>8.78 s</td>
<td>326</td>
</tr>
</tbody>
</table>

5.3. Weak scalability

Two weak scaling studies were done using up to 4096 nodes of Tianhe-2, that is, 12,288 coprocessors in total. The achieved Flop/s rates are shown in Figure 5. The first weak scaling study followed the above single-node performance
study, in that the host CPUs did no computations of their own. Each node was assigned a fixed number of \(142 \times 1200 \times 112\) voxels, and the global mesh grew only in the \(z\)-direction; in other words, each subdomain had two MPI neighbours. Noticeably, this ‘1D-extension’ test achieved 1.27 Pflop/s on 4096 nodes, which translates to an average of 310 Gflop/s per node. This means a 95\% parallel efficiency, if we compare with the single-node performance of 326 Gflop/s, shown in Table 2. It also means that the pipelining approach to hiding the communication overhead (see Section 4.4) has taken effect.

The second weak scaling study was slightly more challenging, where the global mesh grew on the \(xy\)-plane when the number of subdomains increased. That is, each subdomain had four MPI neighbours. In addition, the host CPUs were assigned to compute four layers of voxels each on the ‘left’ and ‘right’ subdomain boundaries. The fixed shape of each subdomain was \(150 \times 1206 \times 112\). This ‘2D-extension’ test achieved considerably lower performance. We believe that this inferior weak scalability was mainly due to the unstable fat-tree interconnect of TH-2, which was constantly under adjustment and development. This made it very difficult to find an efficient 2D mapping of the MPI processes, especially when 2048 and 4096 nodes were used. More specifically, 131 and 259 Tflop/s were obtained using 512 and 1024 nodes, whereas the performance using 2048 and 4096 nodes was 486 and 863 Tflop/s.

5.4. Strong scalability
As a more realistic test, we investigated the strong scalability by using a fixed global mesh of \(3456 \times 3456 \times 672\) voxels. This global mesh is equivalent to discretizing a single sarcomere at a 3 nm resolution. Figure 6 shows a satisfactory speedup from 72 nodes to 1152 nodes. Notice that 2D mesh-partitioning was adopted in this strong-scaling test, so it confirmed that hiding the communication overhead via pipelining is at least effective on relatively small node counts.

6. Simulation results
At the time of writing, although our parallel code is capable of running simulations of one sarcomere at a 1 nm resolution, the multi-user sharing mode of Tianhe-2 unfortunately prevents us from using thousands of nodes for many undisturbed hours. Instead, in this section we will show results from simulations with a 3 nm resolution, which were quickly done on Tianhe-2.

The top left plot in Figure 7 shows the cytosolic Ca\(^{2+}\) distribution at \(t = 7\) ms. We can clearly see local concentration elevations in the vicinity of open RyRs. The plot below shows the Ca\(^{2+}\) concentration in the SR, where we see a corresponding local depletion around open RyRs. The right column of the figure shows close-ups around one of the CRUs. In the top right plot, the colour scale is changed to highlight the fact that steep gradients exist within the CRUs on the cytosolic side as well. In previous 3D studies of Ca\(^{2+}\) waves, individual RyRs were not resolved within a CRU (Izu et al. 2006; Soeller et al. 2009; Nivala et al., 2012). With our high resolution we are able to resolve these gradients.

In the corresponding close-up for SR, the individual open RyRs can be seen by the highly localized depletions in concentration. Around each open RyR, the concentration on either side reaches a local equilibrium within around 1 ms, meaning that the out-flux of SR is not limited by the conductance of the channel, but rather by the diffusion, as is evidenced by the steep Ca\(^{2+}\) gradients in SR.

The bottom two plots show the concentration of Ca\(^{2+}\) -bound Fluo, as a way of simulating how Ca\(^{2+}\) ions bind to Ca\(^{2+}\) -sensitive dye Fluo4, which then emits light. These two plots should be compared with the top row. There is naturally a good correlation between the two, but a striking feature is that the Fluo images are much smoother, and indeed much closer to what is experimentally observed. Notice that the Ca\(^{2+}\) activity in the upper part of the close-up plot of \(c\) is completely masked in the Fluo imaging.

Figure 8 shows the evolution of a Ca\(^{2+}\) wave, by focusing on a \(3 \times 3 \times 2\) \(\mu m^3\) segment of the sarcomere. The first column shows an initial increase in Ca\(^{2+}\) due to the spontaneous activity at two CRUs. Then, at \(t = 8\) ms, the activity has...
Figure 7. \( \text{xy}-\text{snapshots of a simulation at 7 ms. The left column shows solutions in the full extent of the} \ \text{xy-domain, while the right column shows close-ups around a CRU. Notice the use of } \mu\text{M and mM}\) as units for \(c\) and \(c_{\text{sr}}\), respectively.
spread, especially from the leftmost cluster. With increased Ca\(^{2+}\) concentration there comes an increase in the probability of triggering neighbouring CRUs, and this is the mechanism behind Ca\(^{2+}\) wave propagation. In the last column, at \(t = 24\) ms, all of the CRUs in the domain have been activated at least once. Notice the low concentration in the lower right corner, where there was an early independent activation. The incoming wave from the left did not re-trigger this CRU due to inactivation of the RyRs. Instead, the Ca\(^{2+}\) is restored towards baseline values by the activity of the SERCA pump.

The second row in Figure 8 shows \(c^{\text{SR}}\). As in Figure 7 there are local depletions around the open RyRs. Furthermore, at \(t = 24\) ms, we see that Ca\(^{2+}\) is drained from remote locations as well, for example the top left corner where there are no CRUs (as evident from the CSQN distribution in the last row). Also notice that the SERCA pump together with intra-SR Ca\(^{2+}\) diffusion has partially restored Ca\(^{2+}\) levels around the CRU in the bottom left plot. The last row shows the concentration of Ca\(^{2+}\) bound to CSQN, which is located in the SR, around the RyRs; see Figure 1. The presence of these buffers contributes to the current through the RyRs as they act as a local reserve for calcium during release. Initially, they are fully loaded (dark red colour in first row), and gradually become depleted as the SR is drained.

**Figure 8.** Evolution of a Ca\(^{2+}\) wave. In the last row, the dark blue colour signifies empty space, that is, no CSQN buffers there.
Even though we did not run simulations with a 1 nm resolution, we have presented a model of intracellular Ca\(^{2+}\) wave generation and propagation at an unprecedented resolution. In future studies, we will include several sarcomeres so we can study whole-cell Ca\(^{2+}\) wave generation. Furthermore, during heart failure, the Ca\(^{2+}\) handling system is remodelled and the propensity for Ca\(^{2+}\) waves increases. Recent research suggests that the CRUs are smaller, they contain fewer RyRs, but the density of the CRUs is higher, keeping the overall number of RyRs at the same level as in healthy cells or higher (Wu et al. 2012). With our model we can easily capture such a structural remodelling and in future studies we will look at Ca\(^{2+}\) wave generation in failing cells.

7. Conclusions

One may argue that the extra programming complexity due to using the offload mode will become obsolete when Intel’s next generation of MIC has a much improved MIC–MIC communication speed. However, our aim is to use 10,000 nodes or more for the computing power required for simulating subcellular Ca\(^{2+}\) dynamics at nanometre resolution. There is a big difference between having 30,000 (or 40,000) and 10,000 MPI processes. The latter is, at the moment, only possible with the offload mode. Our future work in this aspect will rather be to improve the quality of pipelining, so that the overhead associated with various data transfers can be masked better.

Regarding our achieved single-coprocessor performance, where the obtained memory bandwidth has reached more than 90% of a realistic limit, the lesson learned is that one has to consider the numerical characteristics of the underlying computations. The key is, not surprisingly, to maximize data reuse throughout the entire local memory hierarchy on a coprocessor. Our findings in Section 4.2 should be applicable to similar problems that involve a coupled system of differential equations.

Are we now ready to do simulations of subcellular Ca\(^{2+}\) dynamics at nanometre resolution? For studying a single sarcomere, the required global mesh needs 2 \(\times\) \(10^{11}\) voxels, and it can be estimated that the fully explicit numerical scheme needs about \(1.5 \times 10^6\) time steps to simulate 1 ms. If each voxel does 150 floating-point operations per time step, the total number of floating-point operations required is

\[
1.5 \times 10^6 \times 2 \times 10^{11} \times 150 = 4.5 \times 10^{19}
\]

If we use 10,000 nodes on Tianhe-2, where each node contributes 150 Gflop/s realistic performance, then 30,000 s (8.33 hours) of wall-time is needed. However, several simulations that each last about 50 ms will be needed to study, for example, the propagation speed of the Ca\(^{2+}\) wave with sufficient statistical reliability. A resolution of 1 nm is thus still not within reach. On the other hand, if a spatial resolution of 3 nm is chosen, then the total amount of required computational effort can be reduced by a factor of \(3^3 = 243\). This is because the number of voxels is reduced by a factor of \(3^3 = 27\), whereas the number of time steps is reduced by a factor of \(3^2 = 9\). In other words, a spatial resolution of 3 nm is within reach.

Another possibility for time-saving can arise from improving the numerical strategy. Notice that the currently used very strict requirement of time step is dictated by the strongest diffusion coefficient among the five reaction–diffusion equations. The chosen small value of \(\Delta t\) is due to numerical stability, not accuracy. The other four diffusion coefficients can actually allow larger time steps. Also, since the diffusion and reaction terms are numerically treated separately (see Section 2.2) the reaction terms can use much larger time steps. Part of our future work will thus try a multi-level time-stepping strategy. More specifically, we will find a largest possible \(\Delta T\), such that the accuracy and stability of the reaction terms can be ensured. This value of \(\Delta T\) will be used as the overall time step size. When treating each of the five diffusion terms, inner time steps of size \(\Delta t\) will be adopted to proceed one large time step \(\Delta T\). The maximally allowed value of \(\Delta t\) will depend on the diffusion coefficient of each of the five reaction–diffusion equations. Improved computational efficiency is mainly due to the greatly reduced frequency of updating all the five reaction parts, whereas four of the five diffusion parts are also updated less frequently.

Funding

This work was supported by the National Natural Science Foundation of China, under the NSFC (grant numbers 61033008 and 61272145). Support from the Innovation Program of NUDT’s Graduate School (grant numbers B100603, B120605, and CJ11-06-01) and the FRINATEK program of the Research Council of Norway (project number 214113) is also acknowledged.

Acknowledgements

The authors gratefully acknowledge the assistance from the Tianhe-2 team of NUDT and the National Supercomputing Center in Changsha. NOTUR in Norway is also acknowledged because some small-scale initial numerical studies were carried out on the computing facilities of NOTUR.

References


**Author biographies**

**Jun Chai** is currently working towards a PhD at the Computer College at the National University of Defense Technology, China. His research interests include parallel programming, high-performance scientific computing, and computer architecture. He received BS and MS degrees in Computer Science and Technology from the National University of Defense Technology in 2007 and 2009, respectively. He is a student member of the IEEE and ACM.

**Johan Hake** received a PhD from the Department of Informatics at the University of Oslo in 2009. He is currently employed as a postdoc under the supervision of Dr Joakim Sundnes. Prior to his current position he stayed two years at University of California, San Diego (UCSD) in the Cardiac Mechanics Research Group under the supervision of Professor Andrew McCulloch. His main interest is to develop and use computational tools which help in understanding biological processes. He is a driven computer scientist with more than 10 years of experience in using and developing computational tools to understand biological phenomena in neuro and cardiac physiology, and a maintainer of several computational software tools, including being one of the main developers of FEniCS, a numerical tool for solving PDEs; GAMer, a tool to generate high-quality finite element meshes; and SubCell, a framework for modelling arbitrary reaction diffusion equations on reconstructed geometries. He is also the main developer of Gotran, a modelling tool for general ODEs, and GOSS, a solver for vectorized ODEs.

**Nan Wu** is currently an associate professor at the Computer College at the National University of Defense Technology,
China, and was a visiting researcher at Simula from August 2012 to August 2013. He received his BS, MS, and PhD in Computer Science and Technology from the National University of Defense Technology in 2003, 2005, and 2008, respectively. His research interests include computer architecture, parallel programming, and computer networks.

Mei Wen is currently an associate professor at the Computer College at the National University of Defense Technology, China, and was a visiting researcher at Simula from February 2011 to February 2012. She received her BS, MS, and PhD in Computer Science and Technology from the National University of Defense Technology in 1995, 1999 and 2006, respectively. Her research interests include computer architecture, parallel programming, and scientific computing.

Xing Cai received a PhD in Scientific Computing from the Department of Informatics at the University of Oslo in 1998. In 1999, he was appointed to the position of associate professor at the University of Oslo, and was promoted to full professorship in 2008. He joined Simula at its very beginning in 2001, taking an 80% leave from his university position. His research interests include parallel programming and high-performance scientific computing on multicore CPUs and GPUs, numerical methods for solving PDEs, and generic PDE software. He has participated in numerous PDE-related software projects, most noticeably as the principal developer of the Parallel Toolbox within Diffpack, which is now a commercial product marketed by InuTech, Germany.

Glenn Terje Lines received his MSc and PhD from the Department of Informatics at the University of Oslo, Norway, in 1994 and 1999, respectively. He joined Simula as a postdoc in 2001, where he is currently an associate professor. He holds a part-time position at the Department of Informatics, University of Oslo. He is responsible for the course ‘Mathematical Models in Medicine’ in addition to supervising Masters and PhD students. His research interests include numerical methods for PDEs and computer simulation of cardiac electrophysiology.

Jing Yang is currently working towards an MS degree at the Computer College at the National University of Defense Technology, China. She received a BS degree in Computer Science and Technology from the National University of Defense Technology in 2011. Her research interests include parallel programming and computer architecture.

Huayou Su is currently working towards a PhD at the Computer College at the National University of Defense Technology, China, and was a joint PhD student at Simula from August 2012 to August 2013. He received his BS and MS degrees in Computer Science and Technology from the National University of Defense Technology, China, in 2008 and 2010, respectively. His research interests include parallel programming, high-performance scientific computing, and computer architecture.

Chunyuan Zhang is a professor at the Computer College at the National University of Defense Technology, China. He received his BS, MS, and PhD in Computer Science and Technology from the National University of Defense Technology, China, in 1985, 1990, and 1996, respectively. His research interests include computer architecture, parallel programming, low power design, embedded systems, media processing, and scientific computing. He is a member of the IEEE.

Xiangke Liao is currently a professor and the Dean of the Computer College at the National University of Defense Technology, China. He received a BS degree in Computer Science from Tsinghua University, China, in 1985, and an MS degree in Computer Science from the National University of Defense Technology in 1988. His research interests include parallel and distributed computing, high-performance computer systems, operating systems, and networked embedded systems.