Eike M. Wülfers*, Zhasur Zhamoliddinov, Olaf Dössel, and Gunnar Seemann

Accelerating mono-domain cardiac electrophysiology simulations using OpenCL

Abstract: Using OpenCL, we developed a cross-platform software to compute electrical excitation conduction in cardiac tissue. OpenCL allowed the software to run parallelized and on different computing devices (e.g., CPUs and GPUs). We used the macroscopic mono-domain model for excitation conduction and an atrial myocyte model by Courtemanche et al. for ionic currents. On a CPU with 12 HyperThreading-enabled Intel Xeon 2.7GHz cores, we achieved a speed-up of simulations by a factor of 1.6 against existing software that uses OpenMPI. On two high-end AMD FirePro D700 GPUs the OpenCL software ran 2.4 times faster than the OpenMPI implementation. The more nodes the discretized simulation domain contained, the higher speed-ups were achieved.

Keywords: computational cardiac electrophysiology; cardiac modelling; parallel computing; GPU computing; OpenCL

DOI: 10.1515/CDBME-2015-0100

1 Introduction

Computational models of electrophysiology of cells and tissue are widely used in cardiac research. Most cardiac tissue simulators, like most other scientific software, perform calculations using the central processing unit (CPU) of a computer. Another computing unit present in most computers is the graphical processing unit (GPU). Originally designed to render 3D scenes to the screen in real-time, GPUs provide multiple pipelines for, originally, bulk parallel geometric calculations with low memory requirements. Recently, GPUs have been used more and more for general purpose, non-graphics, but highly parallel computing. Using parallel computing devices such as GPUs for cardiac mono-domain simulation has been shown to greatly improve simulation times [1].

We implemented the models described in section 2 in a new, simple software to simulate cardiac tissue in a faster way. We employed the mono-domain model, finite differences discretization and OpenCL parallelization. The software was developed in the C programming language, parallelized portions in OpenCL C.

2 Methods

2.1 The mono-domain model

Assuming stationary current fields, we can formulate Poisson’s equation of the electric potential for the intracellular domain of electrically active tissue as follows:

\[ \nabla \cdot (\sigma \nabla V_m) = f_i \]

where \( V_m \) is the transmembrane voltage, \( \sigma \) a conductivity tensor, and \( f_i \) the total intracellular current source density. For tissue consisting of electrically active cells, the latter can be expressed more explicitly. The model expressed in one partial differential equation (PDE) then reads

\[ \nabla \cdot (\sigma \nabla V_m) = \beta \left( C_m \frac{dV_m}{dt} + I_{ion} + I_{stim} \right) \] (1)

with the myocyte surface to volume ratio \( \beta \), the specific membrane capacitance per area \( C_m \), the transmembrane ionic current density \( I_{ion} \), and a stimulus current density \( I_{stim} \) [2].

\( I_{ion} \) was calculated using the atrial myocyte model by Courtemanche, Ramirez, and Nattel (CRN)[3]. The model contains formulations for various ionic currents through the ion channels, exchangers, and generally the cell membrane. \( I_{ion} \) is the total density of these currents. The ionic currents depend on \( V_m \) and various state variables. Cell model state variables are formulated as a system of ordinary differential equations (ODEs) such that

\[ I_{ion} = I_{ion}(V_m, \eta) \text{ and } \frac{d\eta}{dt} = f(V_m, \eta) \] (2)

where \( \eta \) is a vector of \( L \) state variables with initial values \( \eta(t = t_0) = \eta^0 \) [1], and \( f \) a function \( \mathbb{R} \times \mathbb{R}^L \rightarrow \mathbb{R}^L \).

2.2 Finite differences discretization

The finite differences method approximates partial derivatives with differences. For this purpose, the domain is dis-
cretized spatially into regularly distributed nodes. The first order derivative of a variable \(u\) in one dimension \(x\) can then be approximated by a difference such as

\[
\frac{\partial_x u_i}{\partial x} = \frac{u_{i+1} - u_i}{\Delta x}
\]

or similar, where \(\Delta x\) is the distance between the nodes. For \(N\) nodes \(u = (u_0, \ldots, u_N)^T\) the derivative can then be approximated as \(\partial_x u = (\partial_x u_0, \ldots, \partial_x u_N)^T = Mu\) with \(M \in \mathbb{R}^{N \times N}\).

Thus, the left hand side of eq. 1 was approximated as

\[
\nabla \cdot (\sigma \nabla V_m) = Ax
\]

where \(x\) is a vector of spatially discrete values of \(V_m\) and \(A\) is the finite differences system matrix. We assumed no-flux boundary conditions.

### 2.3 Time-stepping solution of differential equations

Solving eq. 1 for \(\frac{\partial V_m}{\partial t}\) yields

\[
\frac{\partial V_m}{\partial t} = \frac{1}{C_m} \left( \nabla \cdot (\sigma \nabla V_m) - I_{ion} - I_{stim} \right).
\]

With \(A\) and \(x\) as introduced in section 2.2, and using forward Euler integration, we get the discretized formulation

\[
x^{t+\Delta t} = x^t + \frac{1}{C_m} \left( A x^t - i \right) \Delta t.
\]

Here, \(i\) is the vector of the spatially discretized current densities \(I_{ion} + I_{stim}\), as \(x\) is to \(V_m\). \(I_{ion}\) are calculated by one CRN model instance per node.

Generally, ODEs of the cell model states (eq. 2) were integrated using the explicit forward Euler method. Thus, for a state variable \(\eta\) it was

\[
\eta^{t+\Delta t} = \eta^t + f(V_m, \eta) \Delta t
\]

where \(\Delta t\) is the time step and \(f = \frac{dx}{dt}\) is given by the cell model. However, when \(f\) was of the form

\[
f(V_m, \eta) = (\eta_\infty(V_m) - \eta) / \tau_\eta(V_m)
\]

with the \(V_m\)-dependent steady-state value \(\eta_\infty\) and rate variable \(\tau_\eta\), we used the Rush-Larsen method instead. It was then

\[
\eta^{t+\Delta t} = \eta_\infty + (\eta^t - \eta_\infty) \exp \left( -\frac{\Delta t}{\tau_\eta} \right).
\]

The Rush-Larsen method is also explicit but more stable than the forward Euler method [2].

### 2.4 Error estimation

To quantify the accuracy of the heavily optimized OpenCL software, we compared the activation times of the node furthest from the stimulation area. A node was considered activated once \(V_m > 0\) V.

Similarly to [1], we estimated the overall simulation error with a relative root-mean-square (RRMS) measure

\[
e = 100 \sqrt{\sum_t \frac{\left| x^t_{\text{ref}} - x^t \right|^2}{\left| x^t_{\text{ref}} \right|^2} (\sqrt{\sum_t \left| x^t_{\text{ref}} \right|^2})^{-1}}
\]

where \(x^t_{\text{ref}}\) are the vectors of nodal \(V_m\) values at time \(t\) for a reference solution and \(x^t\) the corresponding vectors for the solution whose error is to be estimated.

### 3 Implementation

#### 3.1 Data structures and distribution

We used 64bit, IEEE-754 double precision floating point numbers as the default data type for all real values. Natural numbers (e.g., indices) were stored in \(m\) bit integer variables, where \(m\) was either 32 or 64 depending on the smallest native integer data type of all devices in the so-called OpenCL context. In our case, the context always included all devices that were used in the simulation. Temporal values were also represented as natural numbers, in nanoseconds.

Vectors were implemented as continuous OpenCL buffers storing floating point values. A buffer in OpenCL is made completely accessible on a device in the context, when referenced as parameter to a so-called kernel (i.e., a function that can be run in parallel on the device). A sub-buffer is a buffer object that points to a block of memory inside another buffer, thus eliminating the need for the driver to copy the full buffer for a kernel invocation on the sub-buffer. For every vector buffer, we defined one sub-buffer per device, splitting it into distinct slices of equal size.

The sparse quadratic finite differences system matrix was stored in the ELL format. The ELL format represents a matrix \(A \in \mathbb{R}^{N \times N}\) with at most \(M\) non-zero elements per row as a tuple \((K, V)\). Every row of \(K \in \mathbb{N}^{N \times M}\) contains the column indices of all non-zero entries from \(A\) in that row, filled up with zeros if the row contains less than \(M\) non-zeros. \(V \in \mathbb{R}^{N \times M}\) contains the non-zero values of each row, also filled up with zeros.

Dense matrices \(K\) and \(V\) were stored row after row in one continuous buffer in host memory each. For each ma-
Matrix-vector multiplication of the form \( b = A x \) where \( A \in \mathbb{R}^{N \times N} \) and \( x, b \in \mathbb{R}^{N} \) was implemented in an OpenCL kernel for a row \( i \in [1, N] \) as

\[
b_i = \sum_{j=1}^{M} V_{ij} x_k \text{ where } k = K_{ij}
\]

with \( M, V \) and \( K \) as defined above. These kernels were launched in parallel with \( N \) instances. Since the values in \( K_{ij} \) were not restricted to device-local indices, \( x \) had to be readable completely from every device, while only the device-local sub-buffers were needed from \( b, V, \) and \( K \). Each thread could thus set \( i \) to its so-called “global ID”, which was its thread ID on the device.

3.3 Cell model ODE kernels

All ODEs for one cell were integrated within one OpenCL kernel with the methods described in section 2.3. Since the cells were independent of each other for this step, these kernels were run with \( N \) parallel instances as well.

State variables from all cell models were stored in a global vector. The slice relevant to a cell was completely copied to private memory at the beginning of a kernel and restored at the end, coalescing the global memory access.

To improve calculation speed, any intermediate values that depended on \( V_m \) and not on any state variable were tabularized beforehand for a range of \(-0.2 \leq V_m \leq 0.2 \) with 4000 steps. This especially applied to Rush-Larsen based calculations (eq. 3) since \( \Delta t \) was constant in this implementation and the whole exponential term could be tabularized.

4 Results

4.1 Simulation setups

Our simulation setups were inspired by a previously published N-version benchmark of electrophysiological simulation software [4]. The simulation setups and parameters were the same for the most part. However, we used the CRN atrial cell model with the initial values from [3]. Our variations of the geometry are detailed in table 1. Variations a, b, and c corresponded directly to the ones used in the benchmark. Geometries d and e were elongated versions of geometry c.

Table 2 shows the parallelization variants we compared. CTL and MPI used our existing software acCELLerate [5]. CLCPU and CLGPU used the newly written software on the CPU and both GPUs, respectively, using the methods and implementations described above. We compared the accuracy and speed against acCELLerate as the latter was itself benchmarked in [4]. Both programs used the same finite differences system matrix. Tabularization, as described in section 3.3, was also done in both programs. For all setups, we simulated 200 ms at a time step of 0.01 ms.

All tests were carried out on a 2013 Apple MacPro with an Intel Xeon E5-2697v2 CPU and two AMD FirePro D700 GPUs. The CPU had 12 physical cores of 2.7 GHz, but provided 24 OpenCL compute units due to HyperThreading. For each GPU, the OpenCL driver reported 32 compute units with 150 MHz.

4.2 Accuracy

We compared CLCPU and CLGPU against CTL on geometries a-c. For the most coarse geometry (a), activation occurred at 38.45 ms for CTL and at 38.54 ms for both, CLCPU and CLGPU (0.09 ms or 0.23% later). For geometry b, activation occurred at 2796 ms for CTL and at 2797 ms for CLCPU and CLGPU (0.01 ms or 0.04% later). In geometry c, activation occurred at 26.66 ms for CTL and again 0.01 ms (i.e., one time step) or 0.04% later.
Table 3: RRMS error, comparing CTL (as reference) and CLGPU. Summation for t from 0 ms in steps of 1 ms. AT is the time of activation of the last node.

<table>
<thead>
<tr>
<th>Geo</th>
<th>t = 15 ms</th>
<th>t &lt; AT</th>
<th>t &lt; 200 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1.4841</td>
<td>1.7861</td>
<td>1.6179</td>
</tr>
<tr>
<td>b</td>
<td>0.1610</td>
<td>0.1335</td>
<td>0.1152</td>
</tr>
<tr>
<td>c</td>
<td>0.0204</td>
<td>0.0159</td>
<td>0.0135</td>
</tr>
</tbody>
</table>

Table 4: Runtime of simulations in seconds. In parentheses the speed-up (ratio) against CTL. Average of 10 simulations.

<table>
<thead>
<tr>
<th>Geo</th>
<th>CTL</th>
<th>MPI</th>
<th>CLCPU</th>
<th>CLGPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>24.2</td>
<td>4.3 (5.61)</td>
<td>9.6 (2.53)</td>
<td>21.9 (1.10)</td>
</tr>
<tr>
<td>b</td>
<td>324.3</td>
<td>34.6 (9.37)</td>
<td>24.6 (13.19)</td>
<td>26.8 (12.11)</td>
</tr>
<tr>
<td>c</td>
<td>2570.2</td>
<td>289.0 (8.89)</td>
<td>187.6 (13.7)</td>
<td>743.8 (3.46)</td>
</tr>
<tr>
<td>d</td>
<td>6366.9</td>
<td>684.8 (9.30)</td>
<td>431.1 (14.76)</td>
<td>354.6 (17.80)</td>
</tr>
<tr>
<td>e</td>
<td>12732.2</td>
<td>1352.6 (9.41)</td>
<td>827.5 (15.39)</td>
<td>558.9 (22.78)</td>
</tr>
</tbody>
</table>

Table 3 shows the RRMS error (eq. 4) of CLGPU against CTL for a single time step with an active excitation front (t = 15 ms), for the time span until activation, and for t from 0 ms to 200 ms, the latter two in steps of 1 ms. Since the differences between CLCPU and CLGPU were extremely small, the shown results compare only CTL with CLGPU.

4.3 Speed

Table 4 shows overall simulation runtimes and speed-up against CTL for the different setups. For all cases but one that are larger than geometry a, the OpenCL version improved on the OpenMPI parallelization. In the one exceptional case (c), the OpenCL implementation was noticeably slower on the GPU while no similar behavior was seen on the CPU. In this one case, the simulation was in fact faster on one GPU than on two.

With the exception of geometry c, the larger the number of nodes, the higher speed-up factors were achieved by the OpenCL parallelizations. This behavior was especially visible for the GPU case, improving the speed-up factor from 12.11 to 22.78 where the same implementation on the CPU improves from 13.19 to 15.39.

5 Discussion

We implemented and evaluated an OpenCL-based mono-domain simulation software for cardiac tissue. Simulation speeds improved compared to our OpenMPI-based implementation. However, we did not see drastic speed-ups on GPUs as described in [1], even though the GPUs we used had a higher core speed.

We achieved higher speed-ups with more nodes on the GPUs. This can be attributed to the fact that the improvements of parallel integration of the many differential equations outweighs the synchronization overhead (that grows by only one (8 byte) value per node). Setup c however showed that the behavior is not monotonic but that there are cases where the distribution across two GPUs is not beneficial.

While accuracy was high for most geometries, there was noticeably slower excitation propagation in the coarse geometry a. While the delay of the activation time of the furthest node was relatively small (0.23%), it was nine times the time step. Together with the steep wave front this led to comparatively large relative errors throughout the simulations when comparing corresponding time steps. Comparing the activation times with those of geometries b and c however, it can be seen that the error caused by the coarse spatial resolution is much larger.

6 Outlook

The developed implementation only uses one cell model for the whole tissue domain. Using multiple cell models would enable simulation of heterogeneous tissue.

OpenCL provides some more features that we have not used, such as heterogeneous computing, using CPU and GPU in parallel. A further speed-up seems possible since only the vector of transmembrane voltages (i.e., 8 MiB for one million nodes) needs to be synchronized at every time step, and data transfer rates between the devices are upwards of 200 GiB/s.

Author's Statement

Conflict of interest: Authors state no conflict of interest.

Material and Methods: Informed consent: Informed consent has been obtained from all individuals included in this study. Ethical approval: The research related to human use has been complied with all the relevant national regulations, institutional policies and in accordance the tenets of the Helsinki Declaration, and has been approved by the authors’ institutional review board or equivalent committee.
References


