Combining Algorithmic Rethinking and AVX-512 Intrinsics for Efficient Simulation of Subcellular Calcium Signaling

Chad Jarvis[1], Glenn Terje Lines[1], Johannes Langguth[1], Kengo Nakajima[2], and Xing Cai[1,3⋆]

[1] Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway
[2] Information Technology Center, The University of Tokyo, Tokyo, Japan
[3] Department of Informatics, University of Oslo, Oslo, Norway

Abstract. Calcium signaling is vital for the contraction of the heart. Physiologically realistic simulation of this subcellular process requires nanometer resolutions and a complicated mathematical model of differential equations. Since the subcellular space is composed of several irregularly-shaped and intricately-connected physiological domains with distinct properties, one particular challenge is to correctly compute the diffusion-induced calcium fluxes between the physiological domains. The common approach is to pre-calculate the effective diffusion coefficients between all pairs of neighboring computational voxels, and store them in large arrays. Such a strategy avoids complicated if-tests when looping through the computational mesh, but suffers from substantial memory overhead. In this paper, we adopt a memory-efficient strategy that uses a small lookup table of diffusion coefficients. The memory footprint and traffic are both drastically reduced, while also avoiding the if-tests. However, the new strategy induces more instructions on the processor level. To offset this potential performance pitfall, we use AVX-512 intrinsics to effectively vectorize the code. Performance measurements on a Knights Landing processor and a quad-socket Skylake server show a clear performance advantage of the manually vectorized implementation that uses lookup tables, over the counterpart using coefficient arrays.

Keywords: Subcellular calcium dynamics · Piecewise constant coefficients · AVX-512 · Xeon Phi Knights Landing · Xeon Skylake.

1 Introduction

The heart needs to be electrically stimulated so that it can contract during every heartbeat. The calcium ion is particularly important for the muscle contraction, and subcellular calcium signaling involves fine-scale physiological details. On the surface of sarcoplasmic reticulum (SR), i.e., each cell's internal calcium storage, there are calcium-sensitive channels called ryanodine receptors (RyRs). They

⋆ Corresponding author: xingca@simula.no.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66

open up when being attached by calcium ions. The typical distance between the cell membrane and the SR surface is 10-20 nm, and this narrow gap is referred to as the cleft space. There are normally between 10 and 100 RyRs inside such a cleft, and together they form a calcium release unit (CRU).

Computer simulations are important for studying subcellular calcium signaling, where the subcellular space is composed of several *physiological domains* (see Fig. 1). These domains, each with distinct properties, are irregularly shaped and elaborately connected. A resulting challenge is to efficiently compute the diffusion-induced, inter-domain calcium fluxes with physiological realism. The main focus of this paper is on utilizing the 512-bit vector length on the Xeon Phi Knights Landing (KNL) and Xeon Skylake processor architectures. In the context of simulating calcium signaling, we will show that manual vectorization needs to be combined with some algorithmic rethinking to fully release the hardware performance potential.

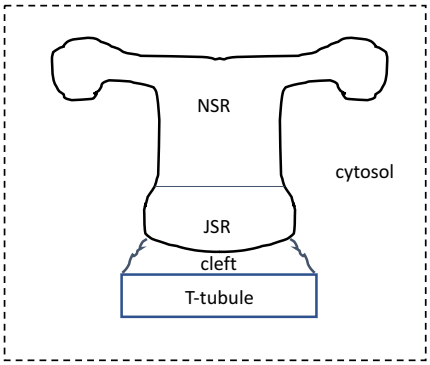

Fig. 1. A 2D schematic of the actual 3D subcellular space when divided into five physiological domains.

2 Computing Diffusion-Induced Calcium Fluxes

The whole 3D spatial domain is covered by a uniform mesh of box-shaped computational voxels. Approximated concentrations of the various calcium species are sought at the center of each voxel (i, j, k). The irregular 3D interior geometries are imbedded into the mesh, such that each voxel belongs *uniquely* to one of the physiological domains. If $u_s(x, y, z, t)$ denotes the concentration of a specific calcium species, then the diffusion-induced increment of u_s over a time step Δt is calculated by a second-order finite volume/difference method as

$$\begin{aligned}\Delta t \Bigg(& \frac{\sigma_s(x_{i+\frac{1}{2}}, y_j, z_k)(u_{s,i+1,j,k} - u_{s,i,j,k}) + \sigma_s(x_{i-\frac{1}{2}}, y_j, z_k)(u_{s,i-1,j,k} - u_{s,i,j,k})}{h^2} \\ & + \frac{\sigma_s(x_i, y_{j+\frac{1}{2}}, z_k)(u_{s,i,j+1,k} - u_{s,i,j,k}) + \sigma_s(x_i, y_{j-\frac{1}{2}}, z_k)(u_{s,i,j-1,k} - u_{s,i,j,k})}{h^2} \\ & + \frac{\sigma_s(x_i, y_j, z_{k+\frac{1}{2}})(u_{s,i,j,k+1} - u_{s,i,j,k}) + \sigma_s(x_i, y_j, z_{k-\frac{1}{2}})(u_{s,i,j,k-1} - u_{s,i,j,k})}{h^2} \Bigg).\end{aligned} \tag{1}$$

To ensure accuracy, the diffusion coefficient needs to be evaluated at the boundary between two neighboring voxels. For example, $\sigma_s(x_{i+\frac{1}{2}}, y_j, z_k)$ denotes the *effective diffusion coefficient* between voxels (i, j, k) and $(i+1, j, k)$. If the two voxels are inside the same physiological domain d, then $\sigma_s(x_{i+\frac{1}{2}}, y_j, z_k)$ naturally

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66

attains the constant diffusion value for species u_s in domain d, denoted by σ_s^d. Care is needed when voxel (i,j,k) is inside domain d whereas voxel $(i+1,j,k)$ is inside another domain e. In such a case, the two voxels are on the border between two physiological domains. To faithfully represent the physiology, we adopt the following formula for evaluating the effective diffusion coefficient:

$$\sigma_s(x_{i+\frac{1}{2}}, y_j, z_k) = \begin{cases} \sigma_s^d & \text{if voxels } (i,j,k) \text{ \& } (i+1,j,k) \text{ both in domain } d; \\ \frac{\sigma_s^d + \sigma_s^e}{2} & \text{if voxel } (i,j,k) \text{ in domain } d, \text{ voxel } (i+1,j,k) \text{ in } e, \\ & \text{and the two domains allow flux in-between;} \\ 0 & \text{if voxels } (i,j,k) \text{ \& } (i+1,j,k) \text{ in two domains} \\ & \text{that do not allow flux in-between.} \end{cases} \tag{2}$$

3 Implementation

The diffusion-related computation, of the form (1), is the most time-consuming part of a 3D subcellular simulation. Since the effective diffusion coefficient between a pair of neighboring voxels may invoke a complex formula of the form (2), care must be taken to ensure the computational efficiency.

3.1 The Coefficient-Array Approach

A commonly used approach is to *pre-calculate* all the effective diffusion coefficients once and for all. This requires three logically 3D arrays to be prepared: `alpha_x`, `alpha_y` and `alpha_z`. For example, `alpha_x` contains values of $\frac{\Delta t}{h^2} \cdot \sigma_s(x_{i+\frac{1}{2}}, y_j, z_k)$, whereas the arrays `alpha_y` and `alpha_z` correspond to the y and z-directions. The diffusion computation can be implemented as below.

```
const int x_offset = ny*nz; const int y_offset = nz;
int xi, yi, zi, pos; double u_value;

for (xi=1; xi<nx-1; xi++) {
  for (yi=1; yi<ny-1; yi++) {
    #pragma ivdep
    for (zi=1; zi<nz-1; zi++) {
      pos = xi*x_offset + yi*y_offset + zi;
      u_value = u[pos];
      // x-direction contribution
      du[pos] += alpha_x[pos]*(u[pos+x_offset]-u_value)
                +alpha_x[pos-x_offset]*(u[pos-x_offset]-u_value);
      // y-direction contribution
      du[pos] += alpha_y[pos]*(u[pos+y_offset]-u_value)
                +alpha_y[pos-y_offset]*(u[pos-y_offset]-u_value);
      // z-direction contribution
      du[pos] += alpha_z[pos]*(u[pos+1]-u_value)
                +alpha_z[pos-1]*(u[pos-1]-u_value);
    }
  }
}
```

Listing 1. The coefficient-array approach to implementing the diffusion computation.

The coefficient-array implementation is fully justified if the effective diffusion coefficients indeed vary everywhere in space. For the example of five physiological

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66

domains, however, there are only $5 \times 5 = 25$ combinations of the effective diffusion constant between any pair of neighboring voxels. An enormous memory footprint overhead is thus associated with the three arrays `alpha_x`, `alpha_y` and `alpha_z`, needed per calcium species. Moreover, a considerable amount of memory traffic arises from (repeatedly) reading these coefficient arrays.

3.2 The Lookup-Table Approach

A memory-friendly approach is thus to pre-calculate for each calcium species a small lookup table (e.g. named `coef_table`) of length `num_domains*num_domains`. It stores the different combinations of the effective coefficient constant. A logically 3D array of `char` values, named `dm_ids`, is assumed to store the physiological domain type info for all the computational voxels. Listing 2 shows this"lookup-table" approach to implementing the diffusion computation.

```
const int x_offset = ny*nz; const int y_offset = nz;
int xi, yi, zi, pos; double u_value, *coef;
char di_m, di_p;

for (xi=1; xi<nx-1; xi++) {
  for (yi=1; yi<ny-1; yi++) {
    #pragma ivdep
    for (zi=1; zi<nz-1; zi++) {
      pos = xi*x_offset + yi*y_offset + zi;
      u_value = u[pos];
      // Focusing on the corresponding row in the lookup table
      coef = coef_table + (dm_ids[pos]*num_domains);
      // x-direction contribution
      di_m = dm_ids[pos-x_offset]; di_p = dm_ids[pos+x_offset];
      du[pos] += coef[di_p]*(u[pos+x_offset]-u_value)
                +coef[di_m]*(u[pos-x_offset]-u_value);
      // y-direction contribution
      di_m = dm_ids[pos-y_offset]; di_p = dm_ids[pos+y_offset];
      du[pos] += coef[di_p]*(u[pos+y_offset]-u_value)
                +coef[di_m]*(u[pos-y_offset]-u_value);
      // z-direction contribution
      di_m = dm_ids[pos-1]; di_p = dm_ids[pos+1];
      du[pos] += coef[di_p]*(u[pos+1]-u_value)
                +coef[di_m]*(u[pos-1]-u_value);
    }
  }
}
```

Listing 2. The lookup-table approach to implementing the diffusion computation.

Code vectorization is needed on processers with SIMD capability to get good performance. On the Xeon Phi Knights Landing and Xeon Skylake architectures, manual vectorization through AVX-512 intrinsics can be essential. Due to space limits, we cannot show the detailed code using AVX-512 intrinsics. It suffices to say that the `mask` variants of AVX-512 intrinsics allow a much more elegant manual vectorization than the previous generations of AVX intrinsics.

4 Performance Measurement and Comparison

Two hardware testbeds have been used. The first testbed is one node on the Oakforest-PACS system [8], i.e., a 68-core Xeon Phi KNL processor of model

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66

7250 that has in total 272 hardware threads. The other testbed is a 4-socket server with Xeon Skylake Gold 16-core CPUs of model 6142, i.e., in total 64 CPU cores and 128 hardware threads. (Due to a suboptimal configuration, however, only four of the six memory channels are occupied per CPU.) The cores of both systems are capable of two 512-bit wide FMA operations per clock cycle. The C compilers used on the two systems are ICC v18 on the KNL node and GCC v8.2 on the Skylake server.

Table 1 lists the time measurements obtained on the two hardware testbeds. Specifically, we have employed two real-world simulations of subcellular calcium signaling, one using a small-scale global mesh of $168 \times 168 \times 168$ voxels, the other using a medium-scale $672 \times 672 \times 168$ mesh. The geometries are based on images obtained from confocal microscopy of rat ventricular myocytes, see [6] for details. In total four implementations (parallelized with OpenMP) have been tested. Two of them correspond to the "plain" coefficient-array and lookup-table approaches, i.e., Listings 1 and 2. These are denoted as `CA_auto` and `LUT_auto` because partial vectorization is automatically enabled by the compilers. In comparison, `CA_man` and `LUT_man` denote the implementations that explicitly use AVX-512 intrinsics.

Table 1. Time measurements of four implementations for the diffusion computation.

Global computational mesh: $168 \times 168 \times 168$, time steps: 16000								
Code version	`CA_auto`		`LUT_auto`		`CA_man`		`LUT_man`	
Testbed	KNL	Skylake	KNL	Skylake	KNL	Skylake	KNL	Skylake
Serial performance	2006.7	963.0	3368.6	846.1	1675.7	861.2	1301.7	463.8
2 OpenMP threads	1146.2	459.4	1767.7	415.6	1003.6	415.3	714.7	210.2
4 OpenMP threads	580.2	187.0	891.7	178.8	510.3	176.2	360.3	89.9
8 OpenMP threads	295.7	101.9	450.4	94.0	258.9	95.6	185.8	48.7
16 OpenMP threads	155.4	58.4	230.0	53.1	136.1	55.7	97.4	28.3
32 OpenMP threads	89.6	43.2	113.0	30.8	79.2	43.2	50.5	21.4
64 OpenMP threads	66.5	40.9	60.1	21.4	62.8	40.9	30.4	18.9
128 OpenMP threads	50.2	45.9	46.2	23.0	54.8	46.8	27.0	22.9
256 OpenMP threads	52.4	N/A	44.0	N/A	54.4	N/A	27.6	N/A
Global computational mesh: $672 \times 672 \times 168$, time steps: 1000								
Code version	`CA_auto`		`LUT_auto`		`CA_man`		`LUT_man`	
Testbed	KNL	Skylake	KNL	Skylake	KNL	Skylake	KNL	Skylake
Serial performance	1905.5	885.7	3381.9	851.1	1606.2	779.2	1284.6	472.7
2 OpenMP threads	1096.9	455.5	1757.9	438.4	989.7	405.6	695.8	252.2
4 OpenMP threads	551.6	236.0	880.8	222.7	498.8	212.2	347.7	130.7
8 OpenMP threads	277.9	134.7	442.2	118.6	252.8	118.4	175.8	68.2
16 OpenMP threads	141.7	69.4	221.3	62.0	128.5	64.7	89.0	36.9
32 OpenMP threads	80.6	47.8	106.9	33.2	72.9	47.8	43.7	26.1
64 OpenMP threads	60.3	58.2	54.4	36.6	57.9	56.6	23.7	30.5
128 OpenMP threads	43.8	67.2	39.3	49.1	49.8	64.3	21.8	42.9
256 OpenMP threads	49.4	N/A	35.7	N/A	46.5	N/A	22.4	N/A

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66

In almost all cases manual vectorization (`CA_man` or `LUT_man`) gives better performance than compiler auto-vectorization (`CA_auto` or `LUT_auto`). On the KNL node, the `CA_auto` version performs better than the `LUT_auto` version using up to 32 OpenMP threads. This seems to suggest that ICC does a better job with auto vectorization for `CA_auto`. On the Skylake server `LUT_auto` always outperforms `CA_auto`. This is likely due to the much smaller memory footprint of `LUT_auto`. Comparing the KNL node with the Skylake server, it is clear that a single Skylake core is much more powerful than a single KNL core. However, the performance advantage of the Skylake server decreases with an increasing number of OpenMP threads used. Figure 2 plots all the time usages related to the medium-scale simulation ($672 \times 672 \times 168$ voxels).

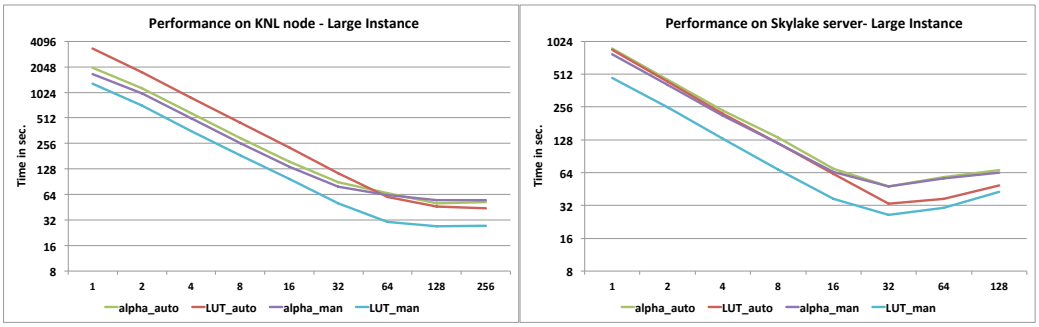

Fig. 2. Comparing the time usage of the four implementations on diffusion computation for the medium-scale simulation ($672 \times 672 \times 168$ voxels).

5 Related Work and Concluding Remarks

There exist many mathematical models of calcium signaling, see [4] for a review. The model adopted by the present paper differs from most of the existing work in that the individual RyRs are accurately resolved with realistic positions and geometries. Moreover, the shape and location of the various physiological domains are reproduced from medical imaging data. This represents a major improvement of the modeling strategy used in [1], where the different domains are simplified to co-exist "on top of each other" throughout the subcellular space. The computational capability of the KNL architecture has been studied in previous publications, such as [2,7,5], using both real-world simulators and well-known benchmarks. However, we are not aware of a detailed study of applying the new AVX-512 intrinsics on the KNL architecture. Regarding the Xeon Skylake server architecture, the existing work such as [3] does not seem to have specifically studied the applicability and performance of AVX-512 intrinsics either.

The work presented in this paper is only a first step towards physiologically realistic simulations of subcellular calcium signaling. Such simulations will even-

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66

tually require using a large number of KNL nodes or high-end Skylake server nodes. We have shown that manual code vectorization through AVX-512 intrinsics is necessary for ensuring good single-node performance. Lookup tables are known to be notoriously difficult to vectorize for a performance advantage. However, with AVX-512 we have seen a clear benefit of the manually vectorized lookup-table implementation, compared with a plain code intended for compiler auto-vectorization.

We want to stress that the strategy of using a lookup table is not restricted to simulations of subcellular calcium signaling. It is applicable to any diffusion-similar calculation where the diffusion coefficient is patch-wise constant. As long as the number of different "patch" types is small, all the different combinations of inter-patch diffusion coefficients can be pre-calculated in a small lookup table. The AVX-512 intrinsics, with the `mask` variants, allow a much more elegant vectorization of this optimization strategy than the previous AVX intrinsics, while giving an additional performance boost due to SIMD.

Acknowledgements: The work is partially supported by the Research Council of Norway (grant No. 251186/F20) and a JHPCN grant (No. JHPCN-jh180024) from Japan.

References

1. Chai, J., Hake, J., Wu, N., Wen, M., Cai, X., Lines, G.T., Yang, J., Su, H., Zhang, C., Liao, X.: Towards simulation of subcellular calcium dynamics at nanometre resolution. International Journal of High Performance Computing Applications **29**(1), 51–63 (2015)
2. Doerfler, D., Deslippe, J., Williams, S., Oliker, L., Cook, B., Kurth, T., Lobet, M., Malas, T., Vay, J.L., Vincenti, H.: Applying the roofline performance model to the Intel Xeon Phi Knights Landing processor. In: Taufer, M., Mohr, B., Kunkel, J. (eds.) ISC High Performance Workshops 2016. Lecture Notes in Computer Science, vol. 9945, pp. 339–353. Springer (2016)
3. Hammond, S., Vaughan, C., Hughes, C.: Evaluating the Intel Skylake Xeon processor for HPC workloads. In: 2018 International Conference on High Performance Computing & Simulation (HPCS). pp. 342–349. IEEE (2018)
4. Izu, L.T., Xie, Y., Sato, D., Bányász, T., Chen-Izu, Y.: Ca^{2+} waves in the heart. J. Mol. Cell Cardiol. **58**, 118–124 (2013)
5. Kang, J.H., Kwon, O.K., Jeong, J., Lim, K., Ryu, H.: Performance evaluation of scientific applications on Intel Xeon Phi Knights Landing clusters. In: 2018 International Conference on High Performance Computing & Simulation (HPCS). pp. 338–341. IEEE (2018)
6. Kolstad, T.R., van den Brink, J., MacQuaide, N., Lunde, P.K., Frisk, M., Aronsen, J.M., Norden, E.S., Cataliotti, A., Sjaastad, I., Sejersted, O.M., Edwards, A.G., Lines, G.T., Louch, W.E.: Ryanodine receptor dispersion disrupts Ca^{2+} release in failing cardiac myocytes. eLife (2018). https://doi.org/10.7554/eLife.39427
7. Langguth, J., Jarvis, C., Cai, X.: Porting tissue-scale cardiac simulations to the Knights Landing platform. In: Kunkel, J., Yokota, R., Taufer, M., Shalf, J. (eds.) ISC High Performance Workshops 2017. Lecture Notes in Computer Science, vol. 10524, pp. 376–388. Springer (2017)
8. Homepage of Oakforest-PACS at JCAHPC. http://jcahpc.jp/eng/ofp_intro.html

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22750-0_66