A comparison of multithreading, vectorization, and GPU computing for the acceleration of cardiac electrophysiology models

Chiheb Sakka, Amina Guermouche, Olivier Aumage, Emmanuelle Saillard, Mark Potse, Yves Coudière, Denis Barthou

A comparison of multithreading, vectorization, and GPU computing for the acceleration of cardiac electrophysiology models

Chiheb Sakka[1], Amina Guermouche[1,4], Olivier Aumage[1,5], Emmanuelle Saillard[1,5], Mark Potse[2,3,4], Yves Coudière[2,3,4], and Denis Barthou[1,5]

[1] STORM Research Team, Inria centre at the University of Bordeaux, Talence, France
[2] CARMEN Research Team, Inria centre at the University of Bordeaux, Talence, France
[3] IHU Liryc, fondation Bordeaux Université, Pessac, France
[4] Univ Bordeaux, IMB, UMR 5251, Talence, France
[5] Univ Bordeaux, Bordeaux INP, UMR 5800, Talence, France

Abstract

Realistic simulation of cardiac electrophysiology requires both high resolution and computationally expensive models of membrane dynamics. Optimization of membrane models can therefore have a large impact on time, hardware, and energy usage. We tested both CPU-based and GPU-based optimization techniques for a human heart model with Ten Tusscher-Panfilov 2006 dynamics. Compared to a multithreaded code running on 64 CPU cores, the tested NVIDIA Tesla P100 GPU proved about 3 times faster. Effective use of the CPU's SIMD capabilities allowed a similar performance gain. GPU performance was bounded by the data transfer rate between GPU and main memory. Optimal SIMD use required explicit vectorization and an adapted data structure. We conclude that on mixed CPU-GPU systems the best results are obtained by optimizing both CPU and GPU code and using a runtime system that balances CPU and GPU load.

1. Introduction

In a monodomain reaction-diffusion model of cardiac electrophysiology the integration of the membrane state and the computation of transmembrane ionic currents at each model node is the most expensive part of the simulation. Acceleration of these tasks is interesting especially for large models with millions of nodes or elements on which membrane models must be integrated. Even though this task is embarrassingly parallel (it requires no communication between model nodes) a straightforward parallelization using OpenMP, MPI, CUDA, or OpenCL usually performs far below the expected performance of the CPU or GPU. In this study we tested a variety of acceleration techniques, ranging from compiler options to code transformations, using the Ten Tusscher-Panfilov model [1], a commonly used and moderately complex model for the human ventricular myocyte, and identified data bandwidth as the main bottleneck both on CPU and on GPU.

Table 1. Models used for the tests.

Model	Δx (μm)	Δt depolarization	nr. of nodes
S	200	0.0500	10,040,000
M	100	0.0125	80,160,000

2. Methods

The baseline code for this work was a simplified version of Propag-5 [2], able to run large monodomain reaction-diffusion simulations with hybrid MPI-OpenMP parallelism. To determine the impact of optimizing the shared memory parts, in particular the membrane model, on this code, we tested, with different approaches, the performance of a model of the whole human ventricles with model sizes of 10^7 and $8 \cdot 10^7$ nodes (table 1), using either 1) up to 64 compute cores on two compute nodes each equipped with two 16-core Intel Xeon E5-2683 v4 "Broadwell" CPUs with a clock frequency of 2.1 GHz, an NVIDIA Tesla P100 GPU, and 256 GB memory and equipped with AVX2 extensions, or 2) up to 72 compute cores on two compute nodes each equipped with two 18-core Intel Xeon Gold 6240 "Cascade Lake" CPUs with a clock frequency of 2.6 GHz and 192 GB memory and equipped with AVX512 extensions. In each test we integrated a monodomain reaction-diffusion equation with an explicit Euler method for 50 ms simulated time.

Tests with model S were run on one or two machine nodes. Tests with model M required two nodes.

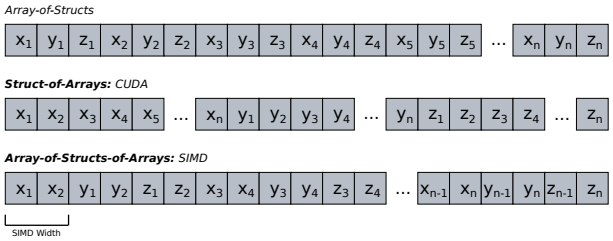

Figure 1. Different data layout representation, in Array-of-Structs-of-Arrays (AoSoA), each component is stored in smaller contiguous chunks corresponding to the size of the vector extension. In SoA, the large contiguous chunks of memory for each variable have benefit in terms of contiguous memory accesses for CUDA kernels

2.1. OpenMP and Vectorization and memory access optimization

Modern processors can execute single-instruction multiple-data (SIMD) instructions, also called vector instructions, allowing to perform up to 8 floating-point operations at once. However, compilers have difficulty recognizing opportunities for vectorization and may be unable to perform the data reorderings that are necessary to use them efficiently. Therefore we enforced vectorization using a portable high-performance C++ SIMD library called MIPP [3], for SSE, AVX and AVX-512 instructions.

Data layout is particularly important for the status variables of the membrane model. In the baseline code these were organized as an array with all variables of a single node stored contiguously. To take advantage of SIMD we used an AoSoA (Array of Structures of Arrays) order, in which a single variable is contiguous for a small group of model nodes, such as to constitute a "data vector."

2.2. GPU CUDA

Graphics processing units (GPUs) have developed into mainstream accelerators for computing. They have parallelization levels that range into the thousands, but their computing capabilities are limited and automated compilation for GPUs often leads to speedups that are orders of magnitude smaller than the number of compute cores in the GPU. Several cardiac simulation codes have already been adapted for use with GPUs [4–10].

A preliminary analysis showed that compute time in our code was small compared to the time needed to transfer data between the CPU and GPU. This pertains to the transmembrane potential, the ionic current, stimulation and diffusion current, and depolarization time of cells. The membrane status variables, a much larger amount of data, stayed on the GPU during the entire simulation. We used CUDA streams to overlap computations with transfers.

3. Results

Figure 2 shows the speedup obtained by different parallel algorithms for model S on the Broadwell architecture: cumulated time for membrane model computation and total program execution. The speedup is measured with respect to the execution time of the multithreaded original code compiled with gcc. It shows the relative increase of the vectorised version's performance using the MIPP wrapper, compiled with icpc version 19.0.4.243 with options `-O3 -inline-forceinline`. The vectorized version used AVX2 extensions, the most advanced available on the machine. The CUDA version shows that concurrently executing copies and kernels using up to 4 streams improves the performance.

Figure 3 shows the speedup obtained by different parallel algorithms for model M on the Broadwell architecture. CUDA-enabled versions use concurrency up to 64 streams due to the size of this model. Using the same compilation options, it shows approximately the same relative performance increase for the vectorised version as for the CUDA version. On the Cascade Lake platform, we could test different SIMD instruction sets (SSE, AVX2 and AVX-512). Figure 4 shows that an optimised data layout increases the performance of the vectorised version: SSE extensions allow to use vectors of 2 double elements (theoretical speedup can reach at maximum 2), but allowed a three-fold performance increase. The AVX2 instructions further increased performance, but AVX512 had no further impact.

In order to understand this behavior, we studied the effect of cache and memory bandwidth on the performance of `tp06` using a roofline model [11]. A roofline model visualizes how kernel performance is affected by the memory bandwidth of a specific architecture. When the performance is close to the peak performance of the architecture, then the kernel is compute-bound and is not affected by the memory bandwidth. In contrast, when the kernel performance is far from the peak performance, then it most likely limited by some cache level bandwidth or by the memory bandwidth. Figure 6 shows the behavior of `tp06` on the Cascade Lake architecture. The figure shows that the vectorization improves the performance compared to the baseline implementation. Moreover, it shows that there is no performance difference between AVX2 and AVX512. This shows that the performance is limited by the level-3 cache and the memory bandwidth, which explains the results of Figure 4.

4. Discussion

On our test platforms, both vectorization and the use of GPUs allowed for a roughly three-fold acceleration of the membrane model compared to multithreaded execution on

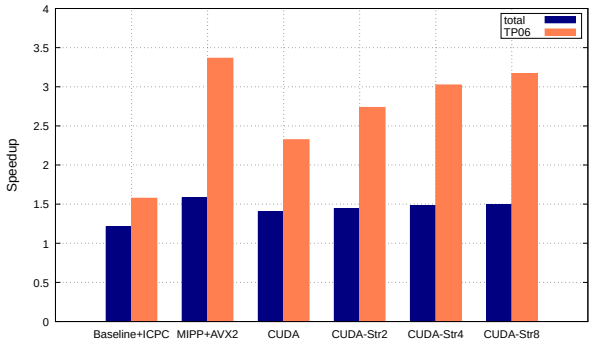

Figure 2. Speedup for model S on 1 node (32 threads) on Broadwell architecture. The graph shows the impact of improving membrane model performance on the membrane model alone ("TP06") and on the whole program ("total"). Str2. . . Str8 stands for 2 to 8 CUDA streams.

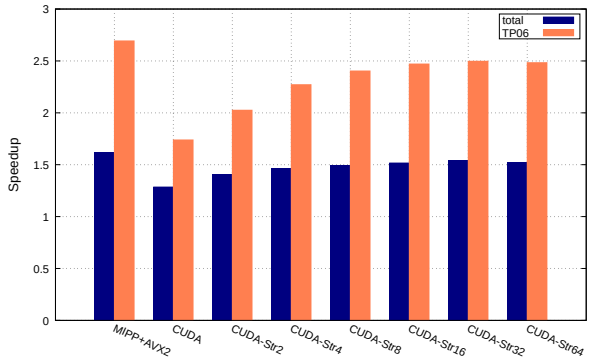

Figure 3. Comparison of the speedup for model M on 2 MPI nodes (64 OpenMP threads in total) on Broadwell architecture.

the CPUs alone with no special care to leverage the SIMD capabilities of the compute cores. Since this probably constitutes the current state of many cardiac simulation codes, this is a useful investigation into two very different avenues for acceleration.

The outcome of our study indicates that vectorization and GPU coding allow for significant acceleration, and that the optimal solution would be to perform both and divide the computations between the CPU and the GPU.

A second important outcome is that memory transfers are the bottleneck for GPU performance on our tested systems. A practical consequence is that investing in more CUDA cores is useless if it is not combined with a higher data transfer rate between the GPU and the main memory.

The outcome of a comparison such as ours obviously depends on the capabilities of the CPUs and GPUs involved. We used CPU types that are currently mainstream in powerful workstations. With cheaper CPUs or more powerful GPUs the outcome would likely have been different.

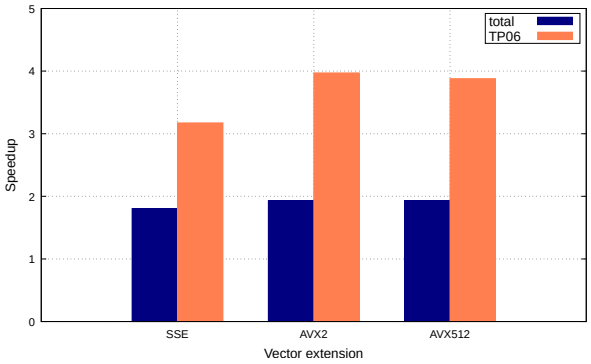

Figure 4. Comparison of the speedup of vectorization according to Vector Extensions for model S on 1 node (36 threads) on Cascade Lake architecture.

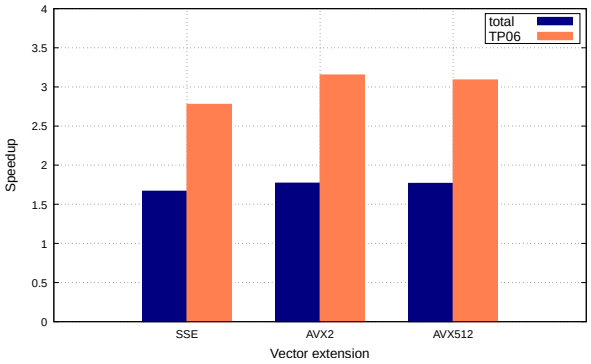

Figure 5. Comparison of the speedup of vectorization according to Vector Extensions for model M on 2 MPI nodes (72 threads) on Cascade Lake architecture.

Speedups reported in the literature indeed vary widely. For example, Sato et al. [4] reported a factor 30 speedup with an NVIDIA Geforce 9800 GX2 GPU, but this was in comparison to a two-core CPU. Given that we used 32 times as many cores for our baseline code, our speedup was actually 3 times better than theirs, if we assume that both CPUs and GPUs improved by the same amount in the 13 years that passed since their study was published. In other words: in the study by Sato et al. a GPU appeared to be worth 60 CPU cores, and in ours 192 CPU cores.

In a more recent study by Neic et al. [6], a single GPU appeared to be worth about 15 CPU cores. This implementation was quite different from ours, because also the partial differential equations were solved on the GPUs.

Although the cited codes and hardware are very different from each other, it is clear that the performance of a GPU core is, in practice, for large-scale cardiac electrophysiological models, one or two orders of magnitude lower than that of a CPU core.

We tested either vectorization or GPU optimization. Optimization of code with such fixed choices requires man-

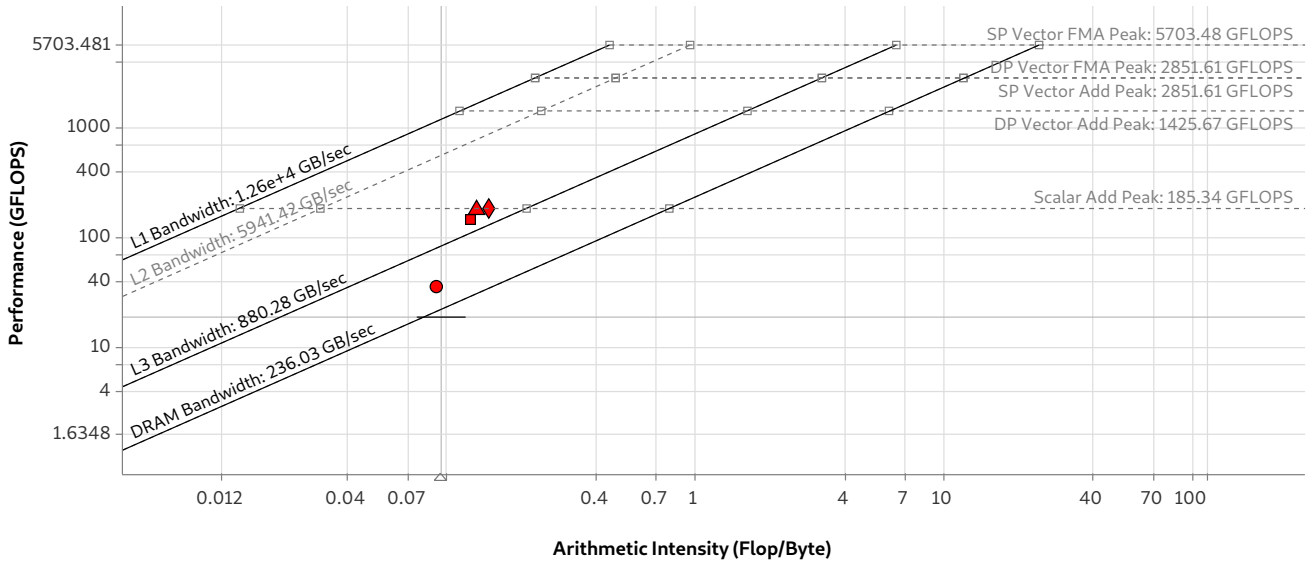

Figure 6. Roofline model comparison for the baseline tp06 loop (represented as circle) and vectorised algorithms (represented by a square (SSE), triangle (AVX2) and diamond (AVX512)) using Intel Advisor for model S on 1 node (36 threads) on Cascade Lake architecture.

ual adaptation to each specific platform and depends on the performance of its CPUs, GPUs, the data transfer rates between them, and on the balance between computational load and memory requirements of the simulation. An alternative for such manual tuning would be to use a runtime scheduler [12], which can find the optimal partitioning of the workload at runtime.

Acknowledgments

This work was supported by the French National Research Agency, grant references ANR-18-CE46-0010 (EXACARD) and ANR-10-IAHU04-LIRYC.

References

[1] Ten Tusscher KHWJ, Panfilov AV. Alternans and spiral breakup in a human ventricular tissue model. Am J Physiol Heart Circ Physiol 2006;291:H1088–H1100.

[2] Krause D, Potse M, Dickopf T, Krause R, Auricchio A, Prinzen FW. Hybrid parallelization of a large-scale heart model. In Keller R, Kramer D, Weiss JP (eds.), Facing the Multicore-Challenge II, volume 7174 of Lecture Notes in Computer Science. Berlin: Springer, 2012; 120–132.

[3] Cassagne A, Aumage O, Barthou D, Leroux C, Jégo C. MIPP: A portable C++ SIMD wrapper and its use for error correction coding in 5G standard. In Proceedings of the 2018 4th Workshop on Programming Models for SIMD/Vector Processing. Vienna, Austria: ACM, 2018; 2.

[4] Sato D, Xie Y, Weiss JN, Qu Z, Garfinkel A, Sanderson AR. Acceleration of cardiac tissue simulation with graphic processing units. Med Biol Eng Comput 2009;47:1011–1015.

[5] Shen W, Wei D, Xu W, Zhu X, Yuan S. Parallelized computation for computer simulation of electrocardiograms based on whole-heart models using personal computers with multi-core CPU and GPGPU. Comp Meth Prog Biomed 2010;100:87–96.

[6] Neic A, Liebmann M, Hoetzl E, Mitchell L, Vigmond EJ, Haase G, Plank G. Accelerating cardiac bidomain simulations using graphics processing units. IEEE Trans Biomed Eng 2012;59:2281–2290.

[7] Mena A, Ferrero JM, Matas JFR. GPU accelerated solver for nonlinear reaction-diffusion systems. Application to the electrophysiology problem. Comput Phys Comm 2015; 196:280–289.

[8] Kudryashova N, Tsvelaya V, Angaldze K, Panfilov A. Virtual cardiac monolayers for electrical wave propagation. Scientific Reports 2017;7:7887.

[9] Vandersickel N, de Boer TP, Vos M, Panfilov AV. Perpetuation of torsade de pointes in heterogeneous hearts: competing foci or re-entry? J Physiol 2016;594:6865–6878.

[10] Hustad KG. Solving the monodomain model efficiently on GPUs. Master's thesis, University of Oslo, Sep. 2019.

[11] Williams S, Waterman A, Patterson D. Roofline: An insightful visual performance model for multicore architectures. Commun ACM apr 2009;52(4):65–76.

[12] Augonnet C, Thibault S, Namyst R, Wacrenier PA. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. CCPE Special Issue Euro Par 2009 February 2011;23:187–198.

Address for correspondence:

Chiheb Sakka, Storm team, Inria Bordeaux – Sud-Ouest
chiheb.sakka@inria.fr