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

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

[1]STORM Team, Inria Bordeaux Sud-Ouest, Talence, France
[2]CARMEN Team, Inria Bordeaux Sud-Ouest, Talence, France
[3]IHU Liryc, fondation Bordeaux Université, Pessac, France
[4]Univ Bordeaux, IMB, UMR 5251, Talence, France

Introduction 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.

Methods Our baseline code was parallelized with MPI and OpenMP. Optimization for CPUs with single-instruction multiple-data (SIMD) capabilities was performed using MIPP library functions and an adapted storage order for the membrane status variables. Optimization for GPUs was achieved using multiple CUDA streams to overlap computation time with memory transfers. Only the membrane model was optimized. Comparisons were made on a monodomain model of the human ventricles with 100 µm resolution and 80 million model nodes, four Intel CPUs (either Xeon E5-2683 v4 "Broadwell" or Xeon Gold 6240 "Cascade Lake"), and two NVIDIA Tesla P100 GPUs.

Results Compared to a multithreaded code running on 64 CPU cores, the P100 GPU ran the membrane model about 3 times faster (1.5 times faster for the whole code). GPU performance was bounded by the data transfer rate between GPU and main memory.

Effective use of the CPU's SIMD capabilities allowed a similar performance gain. Optimal SIMD use required explicit vectorization and an adapted data structure. The 512-bit AVX512 instruction set did not perform faster than the 256-bit AVX2 instruction set.

Discussion In our test case both the use of a GPU and effective SIMD usage were 3 times faster than a naïve multithreaded code. We conclude that on mixed CPU-GPU systems the best results could be obtained by optimizing both CPU and GPU code while using a runtime system that balances CPU and GPU load empirically.