Advances in Engineering Software xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research paper

An optimization approach for agent-based computational models of biological development

Pablo Gonzalez-de-Aledo*,[a], Andrey Vladimirov[d], Marco Manca[b], Jerry Baugh[c], Ryo Asai[d], Marcus Kaiser[e,f], Roman Bauer[f,e]

[a] *Software Performance Optimization Group, Imperial College London, London, United Kingdom*
[b] *CERN Openlab, IT Department, CERN, Geneva 1211, Switzerland*
[c] *Intel Corporation, Santa Clara, CA 95052, United States*
[d] *Colfax International, 750 Palomar Ave, Sunnyvale, CA 94085, United States*
[e] *Interdisciplinary Computing and Complex BioSystems Research Group, School of Computing, Newcastle University, Newcastle upon Tyne, United Kingdom*
[f] *Institute of Neuroscience, Newcastle University, Newcastle upon Tyne, United Kingdom*

ARTICLE INFO

Keywords:
Optimizations
Parallel computing
Vectorization
Coprocessor
Performance
Agent-based models
Biological
Simulation

ABSTRACT

Current research in the field of computational biology often involves simulations on high-performance computer clusters. It is crucial that the code of such simulations is efficient and correctly reflects the model specifications.

In this paper, we present an optimization strategy for agent-based simulations of biological dynamics using Intel Xeon Phi coprocessors, demonstrated by a prize-winning entry of the “Intel Modern Code Developer Challenge” competition. These optimizations allow simulating various biological mechanisms, in particular the simulation of millions of cells, their proliferation, movements and interactions in 3D space. Overall, our results demonstrate a powerful approach to implement and conduct very detailed and large-scale computational simulations for biological research. We also highlight the main difficulties faced when developing such optimizations, in particular the assessment of the simulation accuracy, the dependencies between different optimization techniques and counter-intuitive effects in the speed of the optimized solution. The overall speedup of 595 × shows a good parallel scalability.

1. Introduction

With the recent improvements in computing performance, it has become possible to conduct very detailed and large-scale computational simulations for biological research (e.g. [1–7]). However, the efficient use of computing resources remains a major topic in computational biology.

Agent-based models are a powerful computational approach for research on many topics [8]. These models often involve large numbers of interacting agents, and so are usually very demanding from a computational resource point of view. Along these lines, a number of studies have used high-performance computing for agent-based computer simulations. For example, Deissenberg et al. model the European economy by incorporating millions of agents [9]. In biological simulations, agent-based models usually are multi-scale, including interactions between intracellular, extracellular and cell behavioral dynamics in space. The question of how to implement models for the efficient simulation of such computation-intense biological problems is an important research topic in computational biology (e.g. [10–12]), and the application of modern code development approaches has big potentials to advance this field in various biological scenarios. Along those lines, we here focus on an examplary scenario to maximize the efficacy of multi-core simulations relevant to developmental biology.

In particular, we address general optimization techniques for spatial, agent-based simulations in developmental biology. These simulations comprise millions of cells, the interaction among these, as well as their movements in 3D space, and a computational load that changes during simulation. Our study involves the application of parallel coprocessors in high-performance supercomputing. The optimization of code for such hardware is in its infancy, and recent studies demonstrate impressive improvements for scientific simulations [13,14]. However, it remains underinvestigated how biological agent-based simulations that incorporate multiple interacting scales can benefit from such hardware.

Although various programming interfaces and operating systems ease the transition from sequential to multi-threaded parallel code,

* Corresponding author.
E-mail addresses: pgonzal5@ic.uc.uk, pabloga@teisa.unican.es (P. Gonzalez-de-Aledo), andrey@colfax-intl.com (A. Vladimirov), marco.manca@cern.ch (M. Manca), jerry.r.baugh@intel.com (J. Baugh), ryo@colfax-intl.com (R. Asai), marcus.kaiser@ncl.ac.uk (M. Kaiser), roman.bauer@ncl.ac.uk (R. Bauer).

https://doi.org/10.1016/j.advengsoft.2018.03.010
Received 16 July 2017; Received in revised form 2 March 2018; Accepted 20 March 2018

fully-automated parallelization is very difficult to archive due to the lack of adaptation of compilers and profilers for the underlying hardware structure and the data that feeds the program at hand. Therefore, there is still a huge variation in performance due to different programming styles across functionally equivalent versions of the same code.

In order to explore and assess the performance of current optimization techniques for parallelized scientific software, Intel(R) organized in 2015 the Intel Modern Code Developer Challenge. The primary goal of this challenge is to expose students and researchers to the field of parallel computation with the Xeon Phi platform and the Intel compiler, by teaching modern parallelization techniques that not only increase the performance of a given code on a well-known and established platform, but also keep the code portable for future generations of the same platform. Not less important, the challenge is based on a common language and encourages the discussion of new programming techniques for parallel computation. Moreover, it serves as a showcase for an automated and semi-automated parallelization strategy, using the Intel Parallel compiler.

The Intel Modern Code Developer Challenge took place in October 2015 and comprised the optimization of code for simulating the formation of biological tissue in the early stages of brain development. This code allows the simulation of millions of neural progenitor cells that interact with each other biochemically in 3D space. In particular, it involves a number of fundamental processes during the formation of the brain; namely cell proliferation, migration, and secretion as well as detection of diffusable substances and their concentration gradients. Understanding how these key mechanisms of brain tissue development play out, by taking into account genetic factors in a spatially and temporally dependent way, is crucial for the identification of the causes and potential treatments for neurodevelopmental diseases, such as epilepsy, autism and schizophrenia [15–17]. This code has been developed in the context of a collaboration between CERN openlab and Newcastle university, called the BioDynaMo project [18]. The challenge was accepted by over 17.000 students representing more than 130 universities across 19 countries. The criterion for evaluating the entries was based on the optimized execution time as well as the correctness of the final implementation. The former was measured in the same cluster that the students used to test their optimizations, which is described in Section 3. The latter was ensured by the inclusion of two functions in the code that check the final energy of the cellular clusters as well as by manual inspection by three expert judges from Intel.

As part of the facilities offered to the students, Colfax provided remote access to Intel Xeon processor and Xeon Phi coprocessor-based clusters (whose architectural details are described in Section 3). Students were provided free copies of the Intel Parallel Studio XE Cluster Edition and over 20 h of instructional material (that was used by over 1.000 participants).

In this work, we present the results obtained from one of the prize-winning submissions of the challenge (2nd place), aiming at the optimization of the aforementioned neuroscientific code. Importantly, this particular simulation example at hand comprises processes relevant for many problems in computational biology, because they involve intracellular processes as well as intercellular communication via physical mechanisms. Moreover, the code yields remarkable performance also on architectures other than used in the Intel Modern Code Developer Challenge.

From a computational perspective, one distinguishing feature of developmental models is the dynamic nature of the computational load: the developing brain comprises only a small number of cells at the beginning, but subsequently the system size increases exponentially. Hence, the allocated computing resources meet temporally changing requirements during simulation.

Overall, the performance and correctness of optimized code are paramount factors for the explanatory power and scientific practicality of computer simulations of biological dynamics. The optimized code described in this manuscript, as well as the code of the first and third winning entries are provided as supplementary material.[1] Due to the fact that there are many possible interleaved ways of optimizing this non-trivial code, a detailed comparison between the three versions is out of the scope of this work. The third winner of the competition also provides an explanation of the optimization techniques that he employed in [19], and a substantial overlap exists between the techniques described in his solution and the ones described in this manuscript.

The main contributions of this paper can be summarized as:

- We present a highly parallel implementation of a computer simulation that involves millions of agents interacting in a 3D environment.
- We study the simulation of biological development using various multi-core platforms.
- We explain a general approach to transform sequential code of biological dynamics to run on modern, highly parallel architectures such as the Intel Knights Landing, Broadwell, Sandy-Bridge and AMD Opteron.
- We present the techniques that enabled us to obtain almost 600 × speed-up over the mentioned platforms and simulations.
- The manuscript exemplifies the innovative use of computational strategies and numerical algorithms for large-scale biological problems.

2. Initial software architecture

The initial architecture of the code can be seen in Fig. 1. The code can be partitioned into two phases. Initially, a single precursor cell is placed in the middle of a 3D space.

2.1. Proliferation phase

In the first phase of the simulation, cells move randomly and divide until the final number of cells is reached. After each cell division, the daughter cells each adopt one of two possible cell types, hence giving rise to cell differentiation. This simulation is performed using the functions *produceSubstances* and *cellMovementAndDuplication*. In the function *produceSubtances*, one of the two substances (a or b) is produced depending on the cell type (+ 1 or − 1). Hence, each cell secretes only one of two possible substances, which is determined by their cell type. Cells of type + 1 secrete substance a, and cells of type − 1 secrete substance b. The dynamics of these two substance concentrations are described by the following partial differential equations:

$$\frac{\partial a}{\partial t} = p - \mu a + D_a \frac{\partial^2 a}{\partial^2 x} \tag{1}$$

$$\frac{\partial b}{\partial t} = p - \mu b + D_b \frac{\partial^2 b}{\partial^2 x}, \tag{2}$$

with the basal production constant $p = 0.1$. The prespecified diffusion constant D and decay constant μ are identical for both substances.

In the functions *runDiffusionStep* and *runDecayStep*, the diffusion and decay of the two cellularly secreted substances are numerically simulated. These functions read and write into the *Conc* 3-dimensional matrix, which stores the concentrations of both substances within a spatial grid.

At the end of the iterative step in the proliferation phase, the function *cellMovementAndDuplication* updates the arrays *posAll, pathTraveled, typesAll*. The array *pathTraveled* includes the overall length of the path that each cell travels. If this value exceeds a given threshold parameter T_{path}, the cell divides and the value is reset.

[1] Also available at https://www.github.com/pablo-aledo/intel-modern-code-challenge.