APPROVAL SHEET

Title of Thesis: An MPI-CUDA Implementation of a Model for Calcium Induced Calcium Release in a Three-Dimensional Heart Cell on a Hybrid CPU/GPU Cluster

Name of Candidate: Xuan Huang
Doctor of Philosophy, 2015

Thesis and Abstract Approved: _____
Dr. Matthias K. Gobbert
Professor
Department of Mathematics and Statistics

Date Approved: _____

CURRICULUM VITAE

Name: Xuan Huang

Collegiate Institutions Attended:

- University of Maryland, Baltimore County, 2011–2015,
  Ph.D. Applied Mathematics May 2015.
- University of Toledo, 2010–2011.
- Fudan University, Shanghai, China, 2004–2008,
  B.S. Mathematics May 2008.

Research Experience:

- Research Assistant, High Performance Computing Facility, UMBC, 2012–2015, `www.umbc.edu/hpcf`
- Research Assistant, REU Site: Interdisciplinary Program in High Performance Computing, UMBC, Summers 2013 and 2014, `www.umbc.edu/hpcreu`
- Visiting Scholar, University of Kassel, Kassel, Germany, January 2013
- Affiliated Student, Center for Interdisciplinary Research and Consulting, UMBC, 2011–2012, `www.umbc.edu/circ`
- Teaching Assistant, Department of Mathematics and Statistics, UMBC, Fall 2011 and Spring 2015

Professional Publications:

- X. Huang, M. K. Gobbert, B. E. Peercy, S. Kopecz, P. Birken, and A. Meister. Order Investigation of Scalable Memory-Efficient Finite Volume Methods for Parabolic Advection-Diffusion-Reaction Equations with Point Sources. In Preparation.

- X. Huang and M. K. Gobbert. Long-time Simulation of Calcium Induced Calcium Release in a Heart Cell using Finite Element Method on a Hybrid CPU/GPU Node. High Performance Computing Symposium 2015.

- N. Mistry, J. Ramsey, B. Wiley, J. Yanchuck, X. Huang and M. K. Gobbert. Throughput Studies on an InfiniBand Interconnect via All-to-All Communications. High Performance Computing Symposium 2015.

- J. Schäfer, X. Huang, S. Kopecz, P. Birken, M. Gobbert, and A. Meister. A Memory-Efficient Finite Volume Method for Advection-Diffusion-Reaction Systems with Non-Smooth Sources. Numerical Methods for Partial Differential Equations.

- X. Huang, S. Khuvis, S. Askarian, M. K. Gobbert, and B. E. Peercy. Coupled PDEs with Initial Solution from Data in COMSOL 4. COMSOL Conference 2013.

- N. K. Neerchal, J. G. Morel, X. Huang, and A. Moluh, (2014). A Stepwise Algorithm for Generalized Linear Mixed Models. Proceedings of SAS Global Forum 2014.

- X. Huang and M. K. Gobbert. Parallel Performance Studies for a Three-Species Application Problem on the Cluster maya. Technical Report number HPCF–2015–8, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2015

- O. Adenikinju, J. Gilyard, J. Massey, T. Stitt, J. Graf, X. Huang, S. Khuvis, M. K. Gobbert, Y. Wang, and M. Olano. Real Time Global Illumination Solutions to the Radiosity Algorithm using Hybrid CPU/GPU Nodes.

Technical Report number HPCF–2014–15, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

- A. Cunningham, G. Payton, J. SlettebakU, J. Wolfson-PouU, J. Graf, X. Huang, S. Khuvis, M. K. Gobbert, T. Salter, and D. J. Mountain. Pushing the Limits of the Maya Cluster. Technical Report number HPCF–2014–14, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

- X. Huang and M. K. Gobbert. Parallel Performance Studies for a Three-Species Application Problem on maya 2013. Technical Report number HPCF–2014–8, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

- N. Mistry, J. Ramsey, B. Wiley, J. Yanchuck, X. Huang, A. Raim, M. K. Gobbert, N. K. Neerchal, and P. J. Farabaugh. Clustering of Multidimensional Data Sets with Applications to Spatial Distributions of Ribosomal Proteins. Technical Report number HPCF–2013–10, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2013.

Presentations:

- GPU Technology Conference 2015, Poster presentation, Long-time Simulation of Advection-Diffusion-Reaction System using FEM and FVM on Hybrid CPU/GPU Nodes.

- SIAM Annual Meeting 2014, Contributed talk, Simulation of Calcium Waves in a Heart Cell on Modern Parallel Architectures.

- DELMAR Numerics Day 2014, Challenges and Opportunities in Long-Time Simulations of PDEs on Modern Parallel Computing Platforms.

- COMSOL Conference 2013 Boston, User Presentation, Coupled PDEs with Initial Solution from Data in COMSOL 4.
- SIAM Conference on Computational Science and Engineering 2013, Contributed talk, Efficient Time-Stepping for Parabolic Reaction-Diffusion Equations.
- Graduate Research Conference at UMBC, Poster presentation, A two-dimensional model for calcium flow in a heart cell.
- Differential Equations Seminar, University of Maryland, Baltimore County, A Memory-Efficient Finite Volume Method for Advection-Diffusion-Reaction Systems.

ABSTRACT

Title of Thesis: An MPI-CUDA Implementation of a Model for Calcium Induced Calcium Release in a Three-Dimensional Heart Cell on a Hybrid CPU/GPU Cluster

Xuan Huang, Doctor of Philosophy, 2015

Thesis directed by: Dr. Matthias K. Gobbert, Professor
Department of Mathematics and Statistics
University of Maryland, Baltimore County

A model for Calcium Induced Calcium Release (CICR) in a heart cell describes a physiological process where calcium is able to activate calcium release from the sarcoplasmic reticulum into the cytosol, which is crucial for excitation-contraction coupling in the cardiac muscle. It is modeled by a system of coupled, non-linear, time-dependent advection-diffusion-reaction equations that can be solved by a method of lines approach. The finite element method only solves the equation system without advection, while the finite volume method can also capture advection. Through numerical convergence studies we show that the finite volume method has the same convergence order as the finite element method when there is no advection. We also discuss appropriate discretizations of the advection term for different source terms. We present parallel performance studies for two parallel implementations, one using MPI and running on CPU only nodes, the other using CUDA and MPI together and running on hybrid CPU/GPU nodes. We first establish strong and weak scalability of the implementation using MPI. Then with an extended implementation using CUDA and MPI, we show how to combine several hybrid CPU/GPU nodes successfully in a multi-node distributed-memory compute cluster with high performance interconnect. We present results for a combination of different spatial discretizations and different linear solvers, all showing good speedup and outperforming the CPU only implementation.

# AN MPI-CUDA IMPLEMENTATION OF A MODEL FOR CALCIUM INDUCED CALCIUM RELEASE IN A THREE-DIMENSIONAL HEART CELL ON A HYBRID CPU/GPU CLUSTER

by

Xuan Huang

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

To my parents and my wife

# ACKNOWLEDGMENTS

I dedicate my dissertation work to my dear advisor Dr. Gobbert, for his encouragement, advice, patience, and trust. His guidance not only facilitated my research, but will continue to be the lighthouse in my life's journey. I also want to thank my parents and my wife who have supported me throughout the process. My family is my source of support and encouragement during the challenges of pursuing higher education and study abroad. I am grateful to faculties in the department of mathematics and statistics, especially Dr. Hoffman, who helped me getting the first financial support and introduced me to Dr. Gobbert, and also Dr. Neerchal, who taught me how to turn potential into ability. My thanks must also go to my friends, especially Samuel Khuvis, Jonathan Graf and Zana Coulibaly whose friendship and knowledge have supported and enlightened me over the years. A special thanks to Paul Schou: without his help I could not have finished my dissertation work on time. I wish to thank my committee members who were more than generous with their expertise and precious time.

I acknowledge financial support as Research Assistant in the High Performance Computing Facility (HPCF) and as Teaching Assistant in the Department of Mathematics and Statistics, UMBC.

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258 and CNS–1228778) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Overview

Advection-diffusion-reaction equations occur in a wide variety of applications, for instance fluid flow, heat transfer, spread of pollutants, and transport-chemistry problems. We consider a very general framework of this problem as a testbed to investigate the choices of numerical methods in the face of Dirac delta distributions among the source terms. We also demonstrate the ability of memory-efficient parallel implementations of these methods to solve the problem on extremely fine meshes efficiently using a cluster with state-of-the-art CPU nodes and cutting-edge hybrid CPU/GPU nodes. Our consideration of this problem is inspired by the need to simulate Calcium Induced Calcium Release (CICR) in a heart cell [7, 9, 26]. CICR describes a physiological process within a cell where calcium is able to activate calcium release from the sarcoplasmic reticulum into the cytosol, which is crucial for excitation-contraction coupling in the cardiac muscle.

The general model we consider is a system of coupled, non-linear, time-dependent advection-diffusion-reaction equations

$$u_t^{(i)} - \nabla \cdot \left( D^{(i)} \nabla u^{(i)} \right) + \beta^{(i)} \cdot \left( \nabla u^{(i)} \right) = q^{(i)}, \quad i = 1, \ldots, n_s, \tag{1.1.1}$$

with functions $u^{(i)} = u^{(i)}(\mathbf{x}, t)$, $i = 1, \ldots, n_s$, of space $\mathbf{x} \in \Omega \subset \mathbb{R}^3$ and time $0 \leq t \leq t_{\text{fin}}$ representing the concentrations $u^{(i)}$ of the $n_s$ species. The diagonal diffusivity matrices $D^{(i)} = \operatorname{diag}\left(D_{11}^{(i)}, D_{22}^{(i)}, D_{33}^{(i)}\right) \in \mathbb{R}^{3\times 3}$ consist of positive entries and are assumed to dominate the advection velocity vectors $\beta^{(i)} \in \mathbb{R}^3$, so that numerical methods for parabolic problems are always justified. The right-hand side $q^{(i)}$ is written in a way

that distinguishes the different dependencies and effects as

$$q^{(i)}(u^{(1)},\ldots,u^{(n_s)},\mathbf{x},t) = s^{(i)}(u^{(i)},\mathbf{x},t) + r^{(i)}(u^{(1)},\ldots,u^{(n_s)}) + f^{(i)}(\mathbf{x},t). \qquad (1.1.2)$$

For the calcium species $i = 1$, the terms $s^{(1)}(u^{(1)},\mathbf{x},t)$ in (1.1.2) contains many thousands of point sources modeled by Dirac delta distributions on a large lattice throughout the cell. This crucial feature of the model is responsible for many of the challenges: The numerical method used as spatial discretization will not be convergent to as high an order as conventional, since the source functions are not sufficiently smooth. However the point sources are the crucial driver of the physiological effects, and the domain of the cell needs to be discretized with a very fine mesh to accommodate the large number of point sources. For application problems involving chemical reactions, the system in (1.1.2) conserves total mass, so the numerical method should have this property, as well.

In (1.1.2), the reaction terms $r^{(i)} = r^{(i)}(u^{(1)},\ldots,u^{(n_s)})$ are, in general, non-linear autonomous functions of all species and couple the reaction equations in the system (1.1.1). In (1.1.2), the term $q^{(i)}$ also includes the function $f^{(i)} = f^{(i)}(\mathbf{x},t)$, so that the scalar linear test problem $u_t - \nabla\cdot(D\,\nabla u) + \beta\cdot\nabla u = f(x,t)$ is incorporated in the formulation. This combined formulation of the problems allows one to switch the code from one problem to the other by turning on and off terms and is useful in testing correctness of the code and convergence of the numerical methods in the testbed. These terms will be discussed in detail in Section 2.1.

Gobbert [7] (see also [9] for details) gives a matrix-free Newton-Krylov method for the simulation of calcium flow in a heart cell. The underlying model of calcium flow is given by a system of three coupled diffusion-reaction equations, in which the occurring source terms can be divided into linear and non-linear parts, as well as point sources. Due to the shape of a heart cell, a rectangular domain with a structured grid is a natural choice. The method is based on a finite element discretization and

implemented in a matrix-free manner. The convergence of the finite element method in the presence of the measure valued source terms which occur in the calcium model was rigorously shown in [27].

In [26], a finite volume method for advection-diffusion-reaction systems with smooth and non-smooth sources was introduced. The finite volume method is designed for transport problems, since it satisfies mass conservation in the discretized equations. Moreover, since we intend to consider a more general class of equation systems of advection-diffusion-reaction (ADR) type, the finite volume method becomes a natural choice as opposed to finite element methods which would need additional stabilization terms [24]. However, it is not clear whether a higher order discretization of the advection term is necessary at the cost of more MPI communication among processes.

The main contributions of this work are:

(i) We first extend the physiological example in [26] by adding the effect of advection in the advection-diffusion-reaction system (1.1.1). We demonstrate this effect in three-dimensional long-time simulation of the CICR model. Different advection speeds of the calcium waves are observed, which in turn shows our numerical methods produce physiologically sensible results.

(ii) In the absence of a rigorous convergence theory for the finite volume method for problems involving non-smooth sources, we demonstrate convergence of the method numerically and compare the results to those obtained in [7] by the finite element method. We show convergence for scalar test problems with choices between smooth versus non-smooth source terms and first-order versus second-order discretization for advection term, in both two and three dimensional mesh spacing. These results are compared to simulations using the finite element method in situations where there is no advection.

(iii) Next we discuss parallel implementations that can greatly speed up the com-

putation. In the future, our existing CICR model needs to be extended to incorporate one or two additional species and then should be solved up to a larger final time to be better comparable to laboratory experiments that might take a final time of one minute or more. Thus, our numerical methods and their parallel implementation need to demonstrate the potential to scale up. A conventional performance study in [26] demonstrates good strong scalability by generally halving computing time when using twice as many resources. However, the fixed problem size in that study does not reflect the effectiveness in solving larger problems. We introduce here weak scalability studies for a scalar test problem as well as CICR simulation with advection. Our code demonstrates the ability to handle larger problems with the same efficiency.

(iv) This work is particularly dedicated to using GPUs (graphics processing units) in conjunction with CPUs as tools to enable the faster simulations necessitated by the application. Specifically, we show how to combine several compute nodes in a multi-node distributed-memory compute cluster with high performance interconnect successfully, and we study the performance using several possible combinations of the CPUs and GPUs of the hybrid CPU/GPU nodes.

To proceed, we first explain the hardware and software in the computational environment in detail in Section 1.2, before discussing the motivation for using GPUs in detail in Section 1.3. Section 1.4 points to related work, and Section 1.5 provides an outline of the remainder of this thesis.

## 1.2 Computational Environment

The UMBC High Performance Computing Facility (HPCF) is the community-based, interdisciplinary core facility for scientific computing and research on parallel algorithms at UMBC. Started in 2008 by more than 20 researchers from ten academic departments and research centers from all three colleges, it is supported by faculty

contributions, federal grants, and the UMBC administration. The facility is open to UMBC researchers at no charge. Researchers can contribute funding for long-term priority access. System administration is provided by the UMBC Division of Information Technology, and users have access to consulting support provided by dedicated full-time graduate assistants. See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

Released in Summer 2014, the machine in HPCF is the 240-node distributed-memory cluster maya. The newest components of the cluster are the 72 nodes in maya 2013 with two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs and 64 GB memory that include 19 hybrid nodes with two high-end NVIDIA K20 GPUs designed for scientific computing and 19 hybrid nodes with two cutting-edge 60-core Intel Phi 5110P accelerators. These new nodes are connected along with the 84 nodes in maya 2009 with two quad-core 2.6 GHz Intel Nehalem X5550 CPUs and 24 GB memory by a high-speed quad-data rate (QDR) InfiniBand network for research on parallel algorithms. The remaining 84 nodes in maya 2010 with two quad-core 2.8 GHz Intel Nehalem X5560 CPUs and 24 GB memory are designed for fastest number crunching and connected by a dual-data rate (DDR) InfiniBand network. All nodes are connected via InfiniBand to a central storage of more than 750 TB.

The computational studies in Section 4.1, Section 4.3, and Section 4.4 are obtained using the maya 2013 portion of the cluster. The studies in Section 2.3, Section 3.2.3, Section 3.4, and Section 4.2 are obtained using the maya 2009 portion of the cluster. Each node of maya 2013 contains two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs and 64 GB memory. Figure 1.2.1 shows the architecture of one node containing two CPUs. Each core of each CPU has dedicated 32 kB of L1 and 256 kB of L2 cache. All cores of each CPU share 20 MB of L3 cache. The 64 GB of the node's memory is the combination of eight 8 GB DIMMs, four of which are connected to each CPU with

dedicated memory channels. The two CPUs of a node are connected to each other by two QPI (quick path interconnect) links. Nodes are connected by a quad-data rate InfiniBand interconnect. The InfiniBand is very efficient for data communication between nodes, a detailed throughput study of the InfiniBand interconnect can be found in [21] and [22]. Also, a detailed test of the cluster using both benchmark and application code can be found in [6]. Efficient use of these nodes is demonstrated in

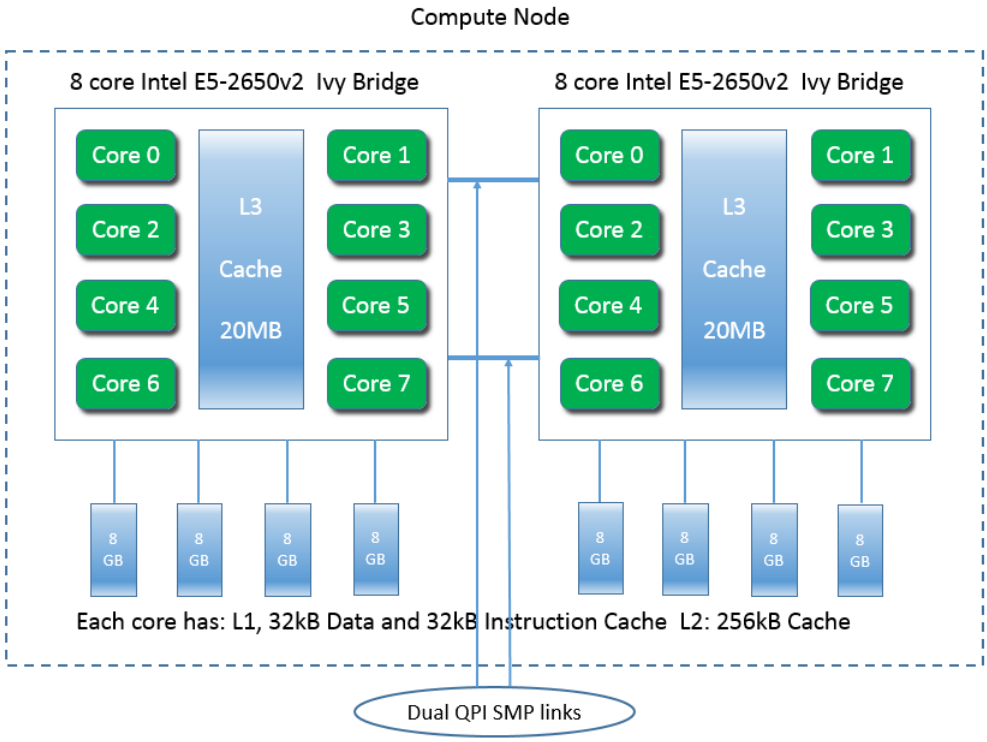



Figure 1.2.1: Schematics of node with two Intel E5-2650v2 Ivy Bridge CPUs.

strong and weak scalability studies in Chapter 4.

There are 19 hybrid nodes with GPUs, one of them is a user node, the rest are compute nodes, see Figure 1.2.2. In addition to two eight-core 2.6 GHz Intel E5-2650v2 Ivy Bridge CPUs, each CPU is connected to one high-end NVIDIA K20 GPU via PCIe interface.

The NVIDIA K20 is a powerful general purpose graphics processing unit (GPGPU)

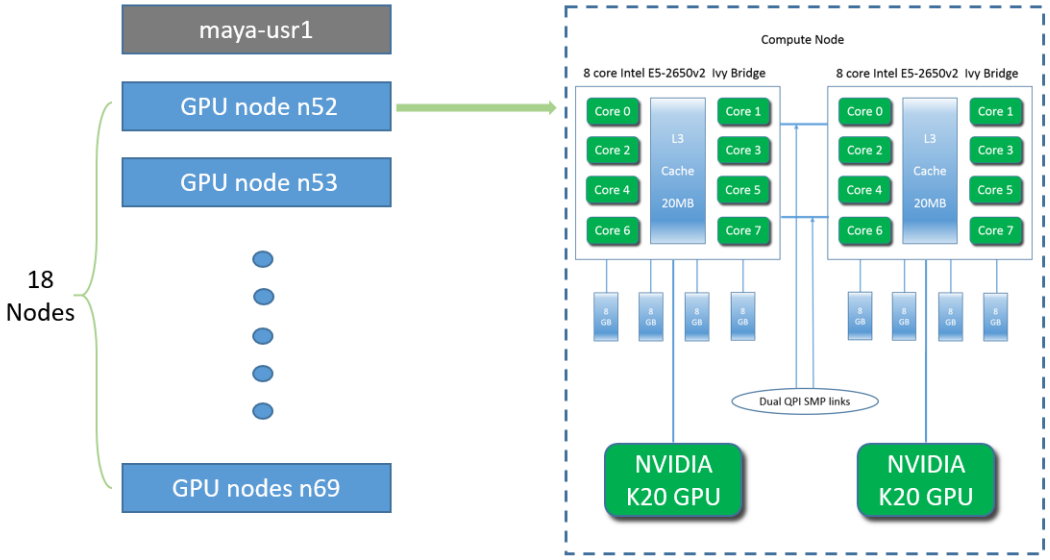



Figure 1.2.2: Schematics of hybrid nodes with GPUs.

with 2,496 computational cores designed for efficient double-precision calculation. GPU accelerated computing has become popular in recent years due to the GPU's ability to achieve high performance in computationally intensive portions of code beyond a general purpose CPU. The NVIDIA K20 GPU has 5 GB of memory. A schematic of the NVIDIA K20 GPU is shown in Figure 1.2.3. Performance of an implementation using the hybrid nodes is presented in Chapter 4, and is compared to CPU only results.

The cluster maya runs the Linux operating system, specifically RedHat EL6. The bash shell is the default. SLURM is used as job scheduler. We use the 64 bit Intel C compiler with version number 15.0. We use two MPI libraries for best performance: In the parallel implementation using MPI described in Section 3.3.1 we use the 64 bit Intel MPI compiler with version number 4.1.3/049. In the parallel implementation using CUDA and MPI described in Section 3.3.2 we use the 64 bit MVAPICH2 with version number 2.0b. To program the NVIDIA GPUs, we use

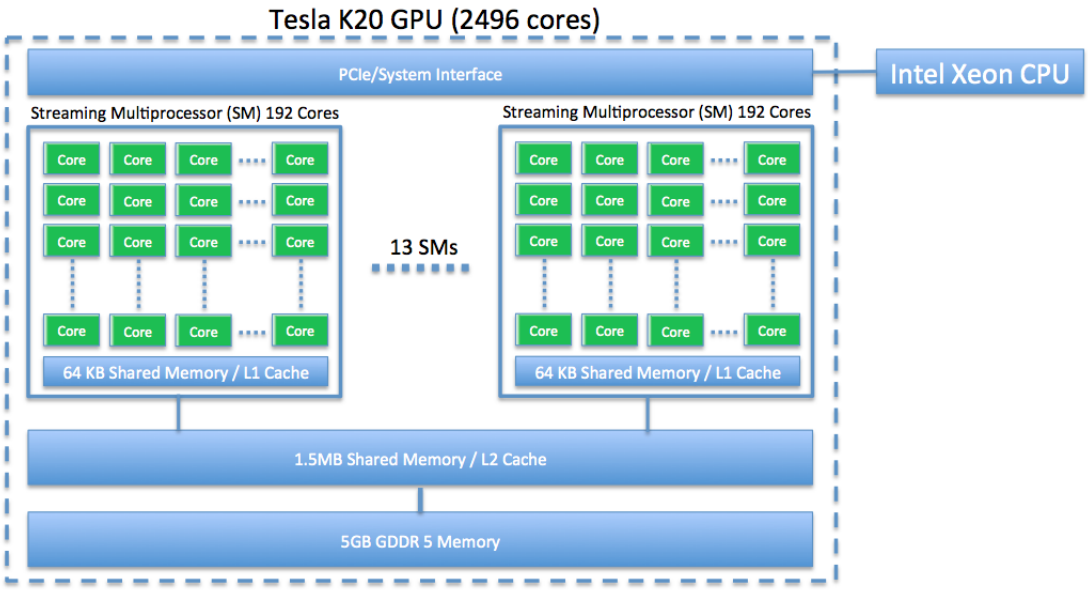



Figure 1.2.3: Schematics of NVIDIA Tesla K20 GPU.

the latest CUDA compilers with version 6.5. CUDA, which stands for Compute Unified Device Architecture, is developed by NVIDIA to enable the programming of their GPUs. It is a parallel computing platform and programming model invented by NVIDIA. An excellent general introduction to CUDA can be found at `http://www.nvidia.com/object/cuda_home_new.html`. A large number of training materials for programming with CUDA including presentations and sample codes can be found at `https://developer.nvidia.com/cuda-education`.

A free on-line course on the introduction to parallel programming using CUDA can be found at `https://www.udacity.com/course/cs344`.

## 1.3 Motivation of using GPUs as Accelerators

GPUs have been developed to perform data-parallel computation using multiple cores. Using GPUs to perform computations that traditionally have been done on CPUs is referred to as General-Purpose Computation on Graphics Processing Unit (GPGPU). The motivations behind our approach of using GPUs as accelerators are

multifold. First of all, the models that we have are becoming more and more complicated. We need to solve the complicated problem with finer meshes, larger (cell) domain, longer simulation time. We also need to prepare for more species which means more PDEs coupled. Furthermore, we might run into situations where thousands of simulations are required for certain studies [3]. The degree of freedom (DOF) as well as computational burden will increase substantially as the model increases complexity. And the implementation using MPI that runs on multi-core CPU cluster will reach a limit. This implementation has good scalability provided you have access to large number of compute nodes with cutting-edge CPUs. However, it is not always easy to have access to a large number of nodes for a reasonably long time. On the other hand, general-purpose computation on GPUs is becoming more and more popular. A GPU is well suited to accomplish single instruction multiple data (SIMD) parallelism. While the CPUs are optimized for low latency, the GPUs are optimized for high throughput. A typical NVIDIA K20 GPU in our cluster has a peak double precision floating point performance of 1.17 Tflops. We have tested the potential of GPU programming with our cluster and were able to speed up a linear solver in the radiosity algorithm: details can be found in [1].

Our problem is suitable for GPU computation for the following reasons: The program is computationally intensive, heavy computation can be done on the GPU with few data transfers. The program is also massively parallel with similar tasks performed repeatedly on different data. Also, from the cost effective aspect, a high-end GPU card is much cheaper to acquire than an up-to-date CPU node, while providing comparable or more throughput. Hence, offloading to accelerators such as GPUs is considered a natural choice.

## 1.4 Related Work

This work is not the first time GPUs have been used to solve similar problems, since general purpose computing with GPUs is getting more and more popular. A problem of reaction-diffusion type is solved in [19], where the authors compare different time-integration methods with a 2D model using one GPU. A Jacobian-Free Newton-Krylov (JFNK) method with GPU acceleration is discussed in [23], where the problem size is limited by using one GPU. A Jacobian-Free Newton-Krylov solver for a one-dimensional particle-in-cell method using hybrid CPU–GPU implementation is introduced in [4], where the JFNK solver is kept on the CPU in double precision (DP), while the particle mover is implemented on a GPU in single precision. A matrix-free Rosenbrock-Krylov method applied to the shallow water equations using CUDA acceleration is presented in [33]. The authors use a Krylov subspace method to solve the linear system on one GPU. The matrix-free implementation saves memory usage, but the authors acknowledge that the size of one GPU memory limits the problem size that can be solved. A CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows is introduced in [31]. Speedups against a single core AMD or Intel processor are observed using one or more GPUs. However, the multi-GPU desktop platform implementation can only scale within the GPUs connected to one CPU, and cannot benefit from multiple nodes.

Our method solves a system of PDEs on a three-dimensional domain with more than 25 million degrees of freedom on one hybrid CPU/GPU node using one GPU, with potential to solve larger problem on multiple nodes. Calculations on both CPUs and GPUs are in double precision to meet the requirements of numerical tolerances. We use performance on one state-of-the-art node with 16 cores in two CPUs as baseline for speedup computation, rather than just serial performance on one core. Our implementation with MPI/CUDA is scalable to multiple nodes with multiple GPUs.

## 1.5 Outline

Chapter 2 is devoted to a detailed explanation of the calcium induced calcium release (CICR) model. Special attention will be given to the probabilistic term that models the superposition of calcium injection into the cell at calcium release units (CRUs). A detailed discussion of two scalar test problems reduced from the application problem follows. These test problems are used in numerical convergence studies in Section 3.4. Simulations of the application problem with advection are presented as part of Chapter 2.

Chapter 3 will explain the numerical methods used to solve the problem. Using either the finite element method (FEM) or the finite volume method (FVM), we can take advantage of the constant diffusion coefficients by solving the problem over a regular spatial mesh. Using a matrix free approach, we can eliminate the storage requirements of the system matrix, which makes computations feasible even with one GPU for reasonably fine mesh. The parallel implementation is then derived by using an implicit time discretization. One implementation uses MPI and runs on CPU only nodes, the other uses CUDA and MPI that runs on hybrid nodes. Two levels of parallelism in the second implementation will be discussed. One level with MPI distributes work among CPU nodes and processes, manages data communication among processes and launches GPU tasks. The second level of parallelism is within each GPU card, where thousands of cores perform calculations on its own chunk of data. Algorithms are designed to keep the data structure inherited from the MPI implementation, which are also optimized to take advantage of the GPU's capability of large throughput, while avoiding the cost of data communication between CPU and GPU as much as possible. This implementation carefully utilizes the structure of data to allow finer numerical meshes and to scale up with more computer nodes. A full convergence study is presented to demonstrate the reliability of the numerical

methods.

In Chapter 4, it will be demonstrated that the first implementation using CPU nodes is efficient and can maintain good scalability. Then the performance of the parallel MPI-CUDA implementation will be presented, to illustrate how effective the MPI-CUDA parallel method is in terms of improving performance. It can be noted from the results that the parallel method using a hybrid node delivers much faster performance than with a single CPU node. The MPI-CUDA approach is also scalable on multiply hybrid nodes.

Chapter 5 summarizes our conclusions.

# CHAPTER 2

# MODEL

This chapter starts with detailed description of the Calcium Induced Calcium Release (CICR) model in Section 2.1, followed by the scalar test cases in Section 2.2. Then, Section 2.3 demonstrates the CICR simulations with advection.

## 2.1 Calcium Induced Calcium Release with Advection

Calcium Induced Calcium Release (CICR) describes a physiological process where calcium is able to activate calcium release into the cytosol, which is crucial for excitation-contraction coupling in the cardiac muscle. This CICR model described in (1.1.1)–(1.1.2) was originally introduced in [14,16], extended in [15], and its numerics discussed in [7,9,26,27]. The problem can be modeled by the system of time-dependent advection-reaction-diffusion equations (1.1.1)–(1.1.2) coupled by non-linear reaction terms. We consider the rectangular spatial domain

$$\Omega = (-6.4\ \mu\text{m}, 6.4\ \mu\text{m}) \times (-6.4\ \mu\text{m}, 6.4\ \mu\text{m}) \times (-32.0\ \mu\text{m}, 32.0\ \mu\text{m}) \subset \mathbb{R}^3$$

which captures the essential size and elongated shape of a heart cell. This model consists of $n_s = 3$ equations corresponding to calcium ($i = 1$), an endogenous calcium buffer ($i = 2$), and a fluorescent indicator dye ($i = 3$). The model uses no-flux boundary conditions

$$\mathbf{n} \cdot \left(D^{(i)} \nabla u^{(i)}\right) = 0 \quad \text{for } x \in \partial\Omega,\ 0 < t \leq t_{\text{fin}} \tag{2.1.1}$$

and provides initial conditions

$$u^{(i)}(\mathbf{x}, 0) = u_{\text{ini}}^{(i)}(\mathbf{x}) \quad \text{for } x \in \Omega,\ t = 0. \tag{2.1.2}$$

The values of the initial concentrations are listed in Table 2.1.1. The values of $u_{\text{ini}}^{(i)}$ for $i = 2, 3$ are calculated, such that the reaction terms $r^{(i)}$ on the right-hand side 1.1.2

cancel for the calcium concentration at basal level of $u_{\text{ini}}^{(i)} = 0.1\ \mu\text{M}$.

We describe the terms on the right hand side (1.1.2) as follows:

In the present model, the term $s^{(i)}(u^{(i)}, \mathbf{x}, t)$ applies only to the calcium species $i = 1$, which is implemented using the Kronecker delta function $\delta_{i1}$ in the definition

$$s^{(i)}(u^{(i)}, \mathbf{x}, t) = \left(J_{\text{SR}}(u^{(1)}, \mathbf{x}, t) - J_{\text{pump}}(u^{(1)}) + J_{\text{leak}}\right) \delta_{i1}, \quad i = 1, \ldots, n_s. \tag{2.1.3}$$

The key term $J_{\text{SR}}$ houses the stochastic aspect of the model, since it models how calcium is released from the sarcoplasmic reticulum (SR) into the cytosol. The calcium release units (CRUs) which represent release sites on the sarcoplasmic reticulum are arranged discretely on a three-dimensional lattice. Each CRU has a probability of opening depending on the concentration of calcium present at that site. This process takes the form

$$J_{\text{SR}}(u^{(1)}, \mathbf{x}, t) = \sum_{\hat{\mathbf{x}} \in \Omega_s} g\, S_{\hat{\mathbf{x}}}(u^{(1)}, t)\, \delta(\mathbf{x} - \hat{\mathbf{x}}). \tag{2.1.4}$$

The equation models the superposition of calcium injection into the cell at CRUs, which are modeled as point sources at all $\hat{\mathbf{x}}$ in the set of CRU locations $\Omega_s$. The Dirac delta distribution $\delta(\mathbf{x} - \hat{\mathbf{x}})$ together with the constant flux density $g$ models a point source at a CRU located at $\hat{\mathbf{x}} \in \Omega_s$. $S_{\hat{\mathbf{x}}}$ is an indicator function, its value is either 1 or 0 indicating whether the CRU at $\hat{\mathbf{x}}$ is open or closed. The value of $S_{\hat{\mathbf{x}}}$ is determined by comparing a uniform random number to the value of the probability distribution

$$J_{\text{prob}}(u^{(1)}) = \frac{P_{\max}\,(u^{(1)})^{n_{\text{prob}}}}{(K_{\text{prob}})^{n_{\text{prob}}} + (u^{(1)})^{n_{\text{prob}}}}. \tag{2.1.5}$$

If the value of the probability distribution is higher than the random number, then the CRU switches on by setting $S_{\hat{\mathbf{x}}} = 1$, otherwise it remains closed by $S_{\hat{\mathbf{x}}} = 0$. When the CRU is open, it stays open for 5 ms, then it remains closed for 100 ms. In our physiological simulations, we study the self-initiation of calcium waves without

stimulation, therefore, the comparison of $J_{\text{prob}}$ to the uniform random number is the only mechanism available in the model to start a calcium wave.

The term $s^{(i)}(u^{(i)}, \mathbf{x}, t)$ also houses the non-linear pump term

$$J_{\text{pump}}(u^{(1)}) = \frac{V_{pump}(u^{(1)})^{n_{pump}}}{(K_{pump})^{n_{pump}} + (u^{(1)})^{n_{pump}}} \tag{2.1.6}$$

and the constant balance term $J_{\text{leak}}$. By design, these terms balance out as $J_{\text{leak}} = J_{\text{pump}}(0.1) \equiv$ constant (see Table 2.1.1) for the calcium concentration at basal level 0.1 $\mu$M.

The reaction terms are

$$r^{(i)}(u^{(1)}, \ldots, u^{(n_s)}) := \begin{cases} \displaystyle\sum_{j=2}^{n_s} R^{(j)}(u^{(1)}, u^{(j)}), & \text{for } i = 1, \\ R^{(i)}(u^{(1)}, u^{(i)}), & \text{for } i = 2, \ldots, n_s, \end{cases} \tag{2.1.7}$$

where the reaction rates are given by

$$R^{(i)} = -k_i^+ u^{(1)} u^{(i)} + k_i^- \left(\overline{u}_i - u^{(i)}\right) \quad \text{for } i = 2, \ldots, n_s, \tag{2.1.8}$$

are modeled as autonomous non-linear functions of the different species and couple the equations in the general system (1.1.1)–(1.1.2). Since the only sources/drains for the species $i = 2, 3$ are the terms that model the binding/unbinding reactions with calcium, the no-flow boundary conditions assure that the total concentration of these species bound and not bound with calcium remains constant, and this constant is denoted by $\bar{u}_i$.

The term $f^{(i)} = f^{(i)}(\mathbf{x}, t)$ is not physiological, but allows for the scalar test problems described in Section 2.2. For the CICR problem, we set $f^{(i)} \equiv 0$ for all $i$.

Similarly, to demonstrate the effect of advection, we extend the model by artificially adding advection to the right in the $z$-direction, i.e., setting $\beta^{(i)} = (0, 0, \beta_3^{(i)})^T$, $i = 1, \ldots, n_s$, with $\beta_3^{(i)} > 0$. The application boundary condition is then modified accordingly to a homogeneous Neumann condition, such that, the species flow in from

Table 2.1.1: Table of parameters for the CICR model.

| Parameter | Description | Values/Units |
|---|---|---|
| $t$ | Time | ms |
| $\mathbf{x}$ | Position | $\mu$m |
| $u^i$ | Concentration | $\mu$M |
| $\Omega$ | Rectangular domain in $\mu$m | $(-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$ |
| $D^{(1)}$ | Calcium diffusion coefficient | $\text{diag}(0.15, 0.15, 0.30)$ $\mu\text{m}^2$/ ms |
| $D^{(2)}$ | Mobile buffer diffusion coefficient | $\text{diag}(0.01, 0.01, 0.02)$ $\mu\text{m}^2$/ ms |
| $D^{(3)}$ | Stationary buffer diffusion coefficient | $\text{diag}(0.00, 0.00, 0.00)$ $\mu\text{m}^2$/ ms |
| $\beta^{(i)}$ | Advection velocity | $\mu$m / ms |
| $u_{\text{ini}}^{(1)}$ | Initial calcium concentration | 0.1 $\mu$M |
| $u_{\text{ini}}^{(2)}$ | Initial mobile buffer concentration | 45.9184 $\mu$M |
| $u_{\text{ini}}^{(3)}$ | Initial stationary buffer concentration | 111.8182 $\mu$M |
| $\Delta x_s$ | CRU spacing in $x$-direction | 0.8 $\mu$m |
| $\Delta y_s$ | CRU spacing in $y$-direction | 0.8 $\mu$m |
| $\Delta z_s$ | CRU spacing in $z$-direction | 0.2 $\mu$m |
| $g$ | Flux density distribution | 110.0 $\mu$M $\mu\text{m}^3$ / ms |
| $\text{P}_{\text{max}}$ | Maximum probability rate | 0.3 / ms |
| $K_{\text{prob}}$ | Probability sensitivity | 0.2 $\mu$M |
| $n_{\text{prob}}$ | Probability Hill coefficient | 4.0 |
| $\Delta t_s$ | CRU time step | 1.0 ms |
| $t_{open}$ | CRU opening time | 5.0 ms |
| $t_{closed}$ | CRU refractory period | 100 ms |
| $k_2^+$ | Forward reaction rate | 0.08 / ($\mu$M ms) |
| $k_2^-$ | Backward reaction rate | 0.09 / ms |
| $\bar{u}_2$ | Total of bound and unbound indicator | 50.0 $\mu$M |
| $k_3^+$ | Forward reaction rate | 0.10 / ($\mu$M ms) |
| $k_3^-$ | Backward reaction rate | 0.10 / ms |
| $\bar{u}_3$ | Total bound and unbound buffer | 123.0 $\mu$M |
| $V_{\text{pump}}$ | Maximum pump strength | 4.0 $\mu$M / ms |
| $K_{\text{pump}}$ | Pump sensitivity | 0.184 $\mu$M |
| $n_{pump}$ | Pump Hill coefficient | 4 |
| $J_{\text{leak}}$ | Leak term | 0.320968365152510 $\mu$M / ms |

the left and leave to the right end of the domain [26]. A complete list of the model's parameter values is given in Table 2.1.1.

## 2.2 Scalar Test Cases

To demonstrate convergence and show weak scalability, we consider two scalar test problems, which are simplifications of the system (1.1.1). Here, we only consider the advection-diffusion-reaction equation, where there is only one species, hence we drop the superscripts. The domain $\Omega$ is chosen to be the same as in the calcium problem, and the diffusion coefficient matrix $D = D^{(1)} = \mathrm{diag}(0.15, 0.15, 0.30)$. The advection velocity is designed as product of a weight $\omega$ and vector $(0.15, 0.15, 0.30)^T$ of the form $\beta = \beta^{(1)} = \omega\,(0.15, 0.15, 0.30)^T$ such that we can control the magnitude of advection by varying the constant $\omega$. For $\omega = 0$, there is no advection, and for $\omega = 1$, diffusion and advection are of the same order of magnitude. To test the effect on accuracy of the spatial discretization, the boundary condition is then modified accordingly to a homogeneous Neumann condition, such that, the species flow in from the left and leave to the right end of the domain [26]. These scalar test problems are used in Sections 3.4 and 4.2.

### 2.2.1 Smooth Source Term

With respect to the right-hand side, we set $r \equiv 0$ and $s \equiv 0$. The initial condition $u_{\mathrm{ini}}$ and right-hand side $f$ are chosen such that the true solution is exactly known as

$$u(x, y, z, t) = \frac{1 + \cos(\lambda_x x)e^{-D_x\lambda_x^2 t}}{2}\frac{1 + \cos(\lambda_y y)e^{-D_y\lambda_y^2 t}}{2}\frac{1 + \cos(\lambda_z z)e^{-D_z\lambda_z^2 t}}{2} \tag{2.2.1}$$

with $\lambda_x = \lambda_y = \pi/6.4$ and $\lambda_z = \pi/32$.

### 2.2.2 Non-smooth Source Term

To show the method's convergence even in the presence of point sources, we consider setting $f \equiv 0$, $r \equiv 0$, and $J_{\mathrm{pump}} \equiv J_{\mathrm{leak}} \equiv 0$. The intention is to simplify the calcium problem by modeling a single CRU in the center of the domain, which opens at time $t = 1$ and remains open afterwards. Therefore, we set $\Omega_s = \{(0, 0, 0)\}$,

$s \equiv s^{(1)} = g\, S_{\hat{\mathbf{x}}}(u,t)\, \delta(\mathbf{x} - \hat{\mathbf{x}})$ with $\hat{\mathbf{x}} \in \Omega_s$ and $g$ from Table 2.1.1, and control manually $S_{\hat{\mathbf{x}}}(u,t) = 0$ for $t < 1$ and $S_{\hat{\mathbf{x}}}(u,t) = 1$ for $t \geq 1$. The true solution is not available for this test problem.

The simplified two-dimensional test problems are set up the same as the three-dimensional cases except that the $y$-dimension is dropped from the domain and model parameters.

## 2.3 CICR Simulations with Advection

The study in this section is part of [13].

We solve the model of CICR from Section 2.1 given by the system of coupled, time-dependent advection-reaction-diffusion equations (1.1.1)–(1.1.2), where the calcium injection is modeled by (2.1.4) with a constant uniform CRU flux density $g$. The parameters are given in Table 2.1.1. This section analyzes the influence of increasing advection in (1.1.1). The initial conditions are as specified in Section 2.1. All studies use the same seed to the random number sequence to allow for a physiological comparison of the simulations.

We first take the case where there are no advection effects, as in [7] and [5], to show three ways to visualize the simulation results. In this case, we have several waves self initiate and propagate throughout the cell, in both orientations of the $z$-direction. The first plotting method is called a CRU Plot, shown in Figure 2.3.1. The long-time simulations of the CICR model go up to the large final time ($t_{\text{fin}}$ = 1,000 ms, as noted in Section 2.1). The plots in this figure show which CRUs are open at each time step during the simulation. We see that at $t = 100$ a few CRUs are open, the wave mostly spreads along $x$ and $y$ dimensions at this point. Later on we see that the CRUs have begun to open on both sides of the cell and spread across it. During our simulation of 1,000 ms, several waves have been generated and run across the

cell, with similar speed on both ways of the $z$-direction. The second plotting method is called an Isosurface Plot, shown in Figure 2.3.2. The plots in the figure take the same time steps as in the CRU plots, but they show calcium concentration instead of open CRUs. The Isosurface Plots give us a 3-dimensional representation of how the calcium diffuses through the cell based on the concentration of calcium species $u_1$. The different shades of blue, red and yellow indicate the level of calcium concentration throughout the cell as the simulation, while red indicates higher concentration, blue indicates lower. There is a critical value associated with our plots, $u_{\text{crit}} = 65\mu\text{M}$. Inside the plotted region, the concentration is higher than $u_{\text{crit}}$, while outside the plotted area the concentration is lower than $u_{\text{crit}}$. On the boundary of the plotted area inside the cell the concentration is equal to $u_{\text{crit}}$, and the color is blue. However on the boundary of the cell domain, the concentration might be higher than $u_{\text{crit}}$, hence the color is more red. Again, we see that when $t = 100$ in Figure 2.3.2 there is a small amount of calcium in the cell. As time advances, we see that the amount of calcium in the cell increases and diffuses throughout the cell. The third plotting method is called a Confocal Image Plot, shown in Figure 2.3.3. The confocal images are meant to replicate what scientists see in the laboratory experiments using florescent dye to bind to the calcium in the heart cell. The lighter of green shades indicate higher calcium concentrations, while the darker green shades indicate lower concentrations of calcium. When $t = 100$, we see calcium start to diffuse across the cell as shown in Figure 2.3.3.

The key motivation to expand the current model to include advection effects comes from work such as [2] that details several mechanisms for intracellular calcium pathology and [30] that discusses mechanisms for intracellular calcium waves to impact multi-cellular electrical arrhythmias. This paper introduces a simple advection affect with constant advection velocity vectors $\beta^{(i)}$ in (1.1.1) (i) to demonstrate that the

model behaves correctly with advection and (ii) to test the convergence of the numerical method for the advection term in Section 3.4. We study advection in the elongated $z$-direction of the cell by setting $\beta^{(i)} = (0, 0, \beta_3^{(i)})$, $i = 1, \ldots, n_s$, in the general system (1.1.1). We vary the value of the $z$-component $\beta_3^{(i)} = 0.01, 0.03, 0.05, 0.1, 0.2$. Figures 2.3.4, 2.3.5, 2.3.6, 2.3.7, and 2.3.8 show CRU plots for these values of $\beta_3$, analogous to Figure 2.3.1 without advection. These plots show that with larger $\beta_3$ the effect of advection is stronger, as demonstrated by the waves been pushed progressively more in the positive $z$-direction.

To demonstate the effect of advection, Figures 2.3.9 and 2.3.10 show line scan plots of the calcium concentration along the longitudinal axis of the cell. Each line scan plot shows the concentration at fixed values of $x$ and $y$, for $-32 \leq z \leq 32$ on the vertical and $0 \leq t \leq 1{,}000$ on the horizontal axis. The color represents the calcium concentration, with different shades of blue, yellow, and red indicating the level of calcium concentration, where blue indicates a lower concentration and red indicates a higher one.

Figure 2.3.9 shows line scan plots along the line segment for $x = y = 0$. Figure 2.3.9 (a) shows line scan plots when there is no advection. At $t = 0$ ms, we see constant blue, which represents the initial condition. At around $t = 50$ ms, we see lighter blue at some point $P$ close to the center of the line segment, which means the calcium concentration is higher at this point. As $t$ gets larger, we observe that the concentrations at neighboring points of $P$ increase, and then the concentrations at the points further away increase, and so on. These points form two symmetric skew lines, capturing the propagation of the calcium waves along the line segment. We observe several pairs of symmetric skew lines, representing several waves, which we have observed in Figures 2.3.1, 2.3.2, and 2.3.3. The slopes of the lines are the speeds at which the calcium waves travel. In the case with no advection, the calcium wave

travels at the same speed along the positive and negative $z$-direction, hence the skew lines are symmetric about the starting point $P$.

Figures 2.3.9 (b)–(f) show line scan plots with the $z$-component of advection velocity vector $\beta_3^{(i)} = 0.01, 0.03, 0.05, 0.1, 0.2$, respectively. We observe from Figure 2.3.9 (b) that the slopes of the skew lines on the positive side of $P$ are slightly increased, meanwhile the slopes of the skew lines on the negative side of $P$ are slightly decreased, compare to Figure 2.3.9 (a). This means that, as we add the effect of advection on the positive $z$-direction, the calcium wave traveling in the positive $z$-direction is faster, and the wave traveling in the negative $z$-direction is slower. We also observe from Figure 2.3.9 (c) that as we increase advection, the slopes of the skew lines on the positive side of $P$ are further increased, and the concentrations on the skew lines are also higher. In the meantime, the slopes of the skew lines on the negative side of $P$ decreased significantly, and the lines are much shorter. This indicates that the calcium waves cannot propagate to the far negative side of the domain. The CRUs there cannot open due to low calcium concentration. As the $\beta_3^{(i)}$ gets larger and larger, we observe from Figures 2.3.9 (d)–(f) that the slopes of the skew lines on the positive side of $P$ gets larger and larger, indicating the speed at which calcium waves travel along the positive $z$-direction gets faster and faster. The calcium concentration on the skew lines are getting higher and higher as well. We also observe that fewer waves are generated. No lines are observed on the negative side of $P$. These plots match our observations from Figures 2.3.4, 2.3.5, 2.3.6, 2.3.7, and 2.3.8, and they clearly demonstrate the impact of advection on the speed of calcium waves.

Figure 2.3.10 shows line scan plots along the line segment for $x = y = 5.6$, which is near the edge of the domain $\Omega$. In Figures 2.3.10 (a)–(f), We observe similar patterns just like Figures 2.3.9 (a)–(f), respectively, but the calcium concentrations are generally lower due to the position of the line segment in the domain. These plots

confirm that we capture the effect of advection on calcium waves in the whole domain, not just along any single line segment.

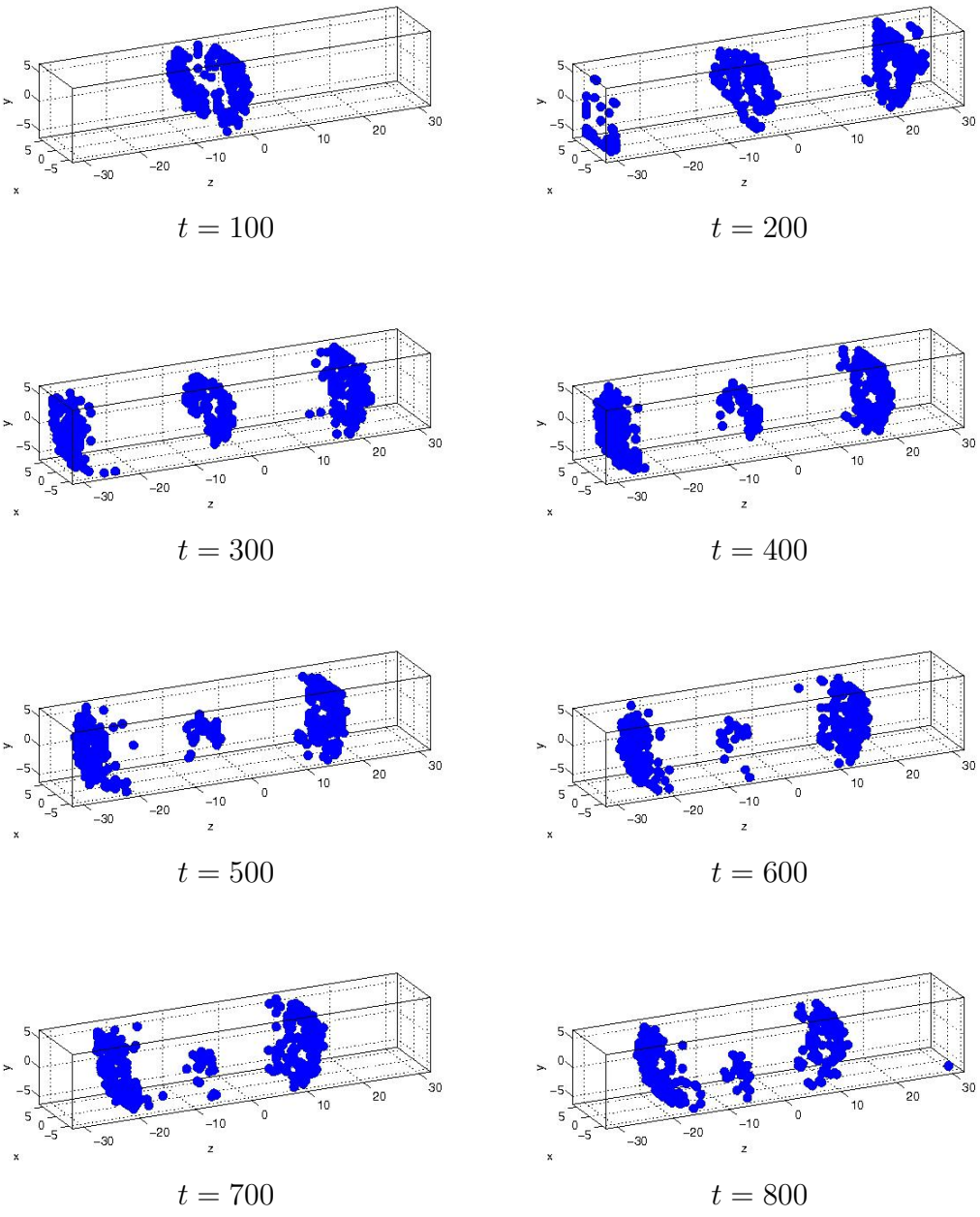
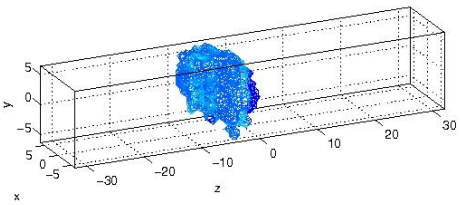
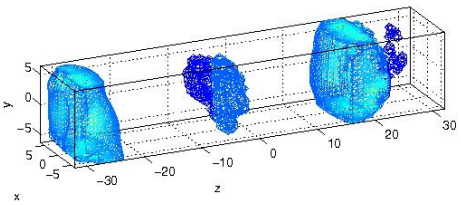


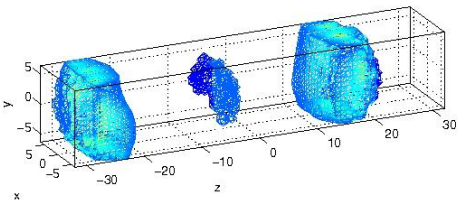

Figure 2.3.1: Open calcium release units throughout the cell using the finite volume method without advection on mesh size $32 \times 32 \times 128$.



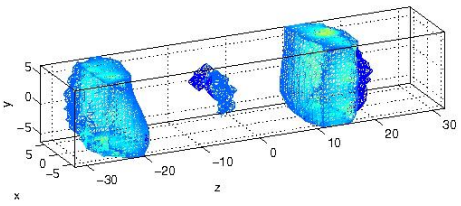

$t = 100$



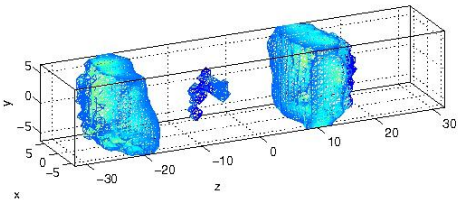

$t = 200$



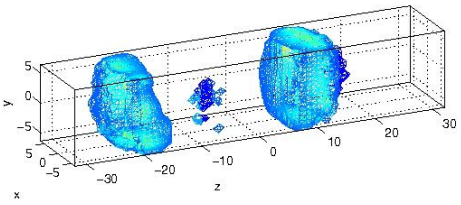

$t = 300$



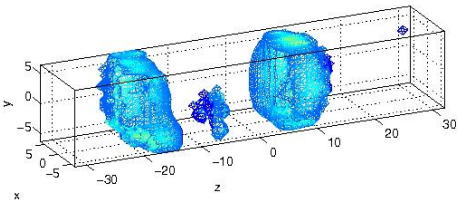

$t = 400$



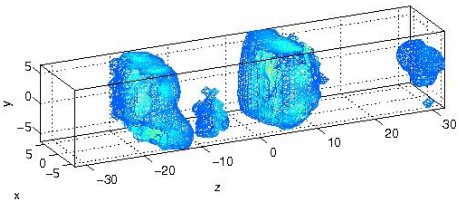

$t = 500$




$t = 600$




$t = 700$




$t = 800$

Figure 2.3.2: Isosurface plots of the calcium concentration using the finite volume method without advection on mesh size $32 \times 32 \times 128$.

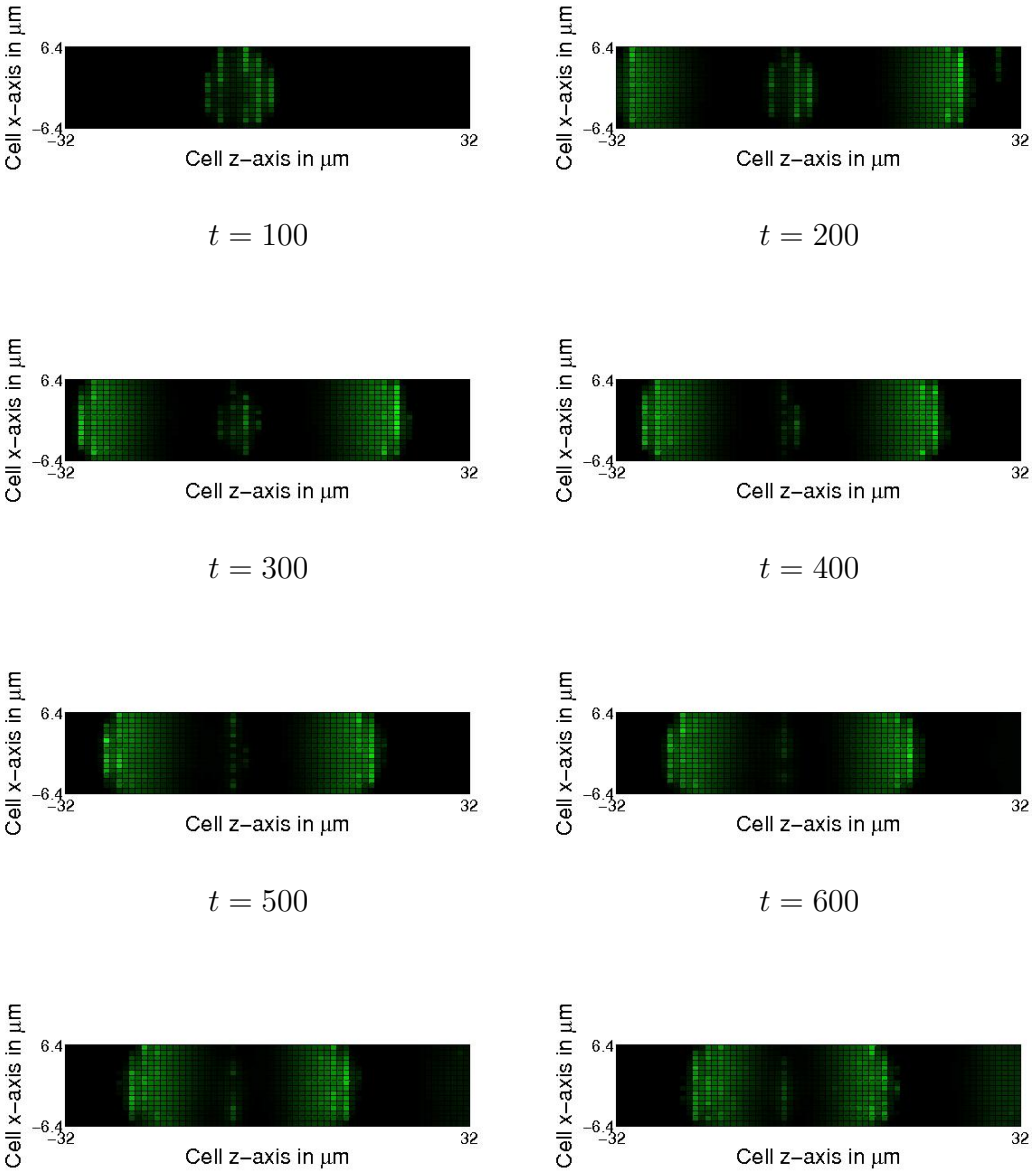
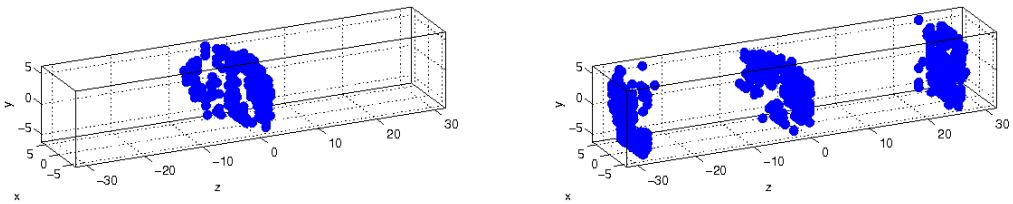


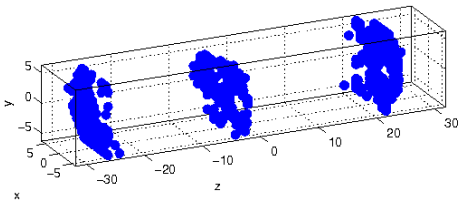

Figure 2.3.3: Confocal image plots of the calcium concentration using the finite volume method without advection on mesh size $32 \times 32 \times 128$.



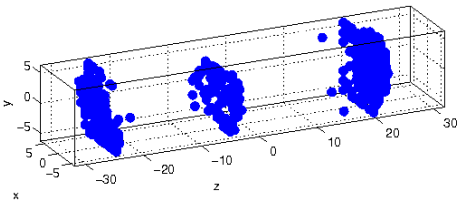
$t = 100$



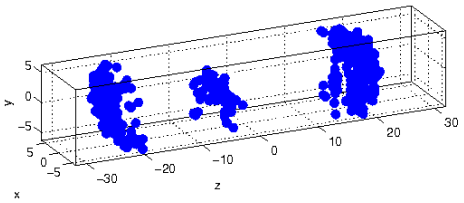
$t = 200$



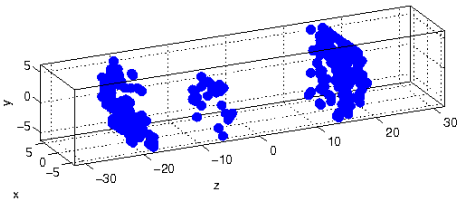
$t = 300$



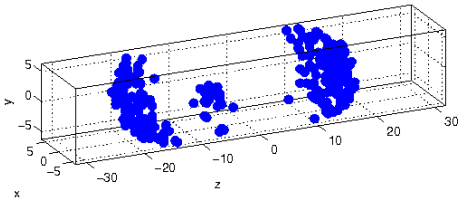
$t = 400$



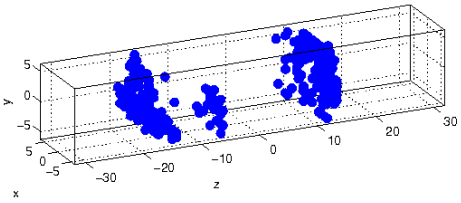
$t = 500$



$t = 600$

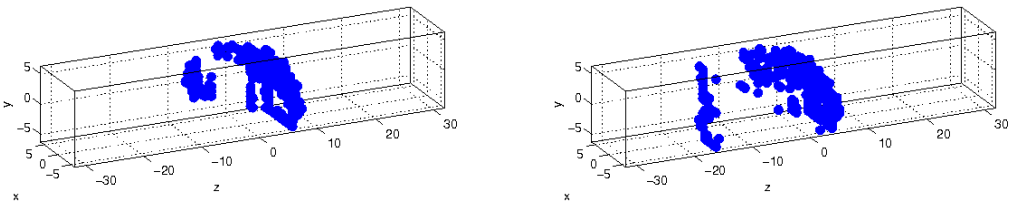


$t = 700$



$t = 800$

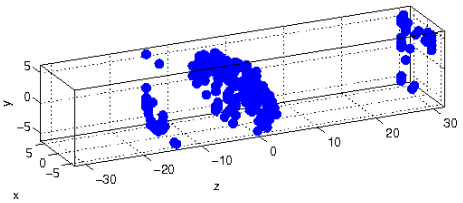
Figure 2.3.4: Open calcium release units throughout the cell using finite volume method with $\beta^{(i)} = (0, 0, 0.01), i = 1, \ldots, n_s$, on mesh size $32 \times 32 \times 128$.



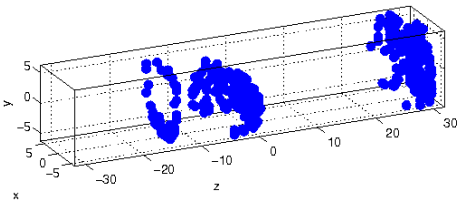
$t = 100$



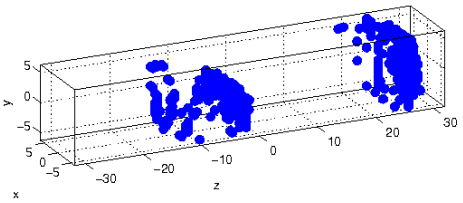
$t = 200$



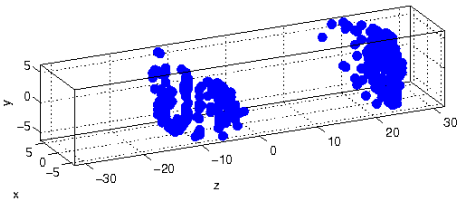
$t = 300$



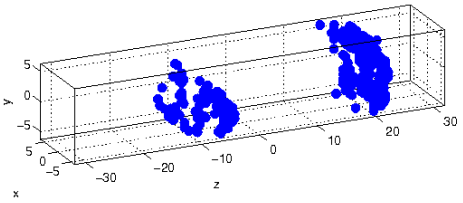
$t = 400$



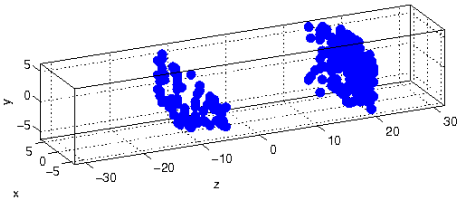
$t = 500$



$t = 600$

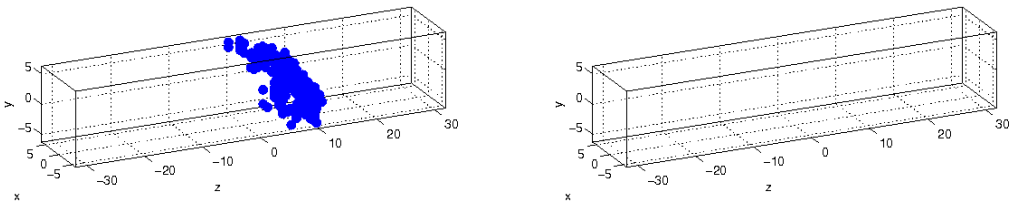


$t = 700$



$t = 800$

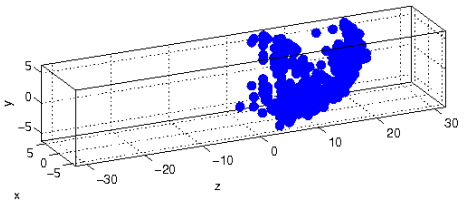
Figure 2.3.5: Open calcium release units throughout the cell using finite volume method with $\beta^{(i)} = (0, 0, 0.03), i = 1, \ldots, n_s$, on mesh size $32 \times 32 \times 128$.



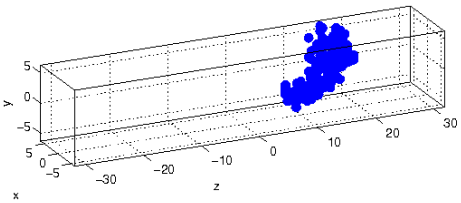
$t = 100$



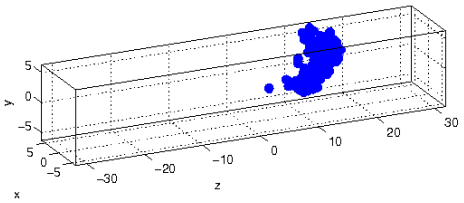
$t = 200$



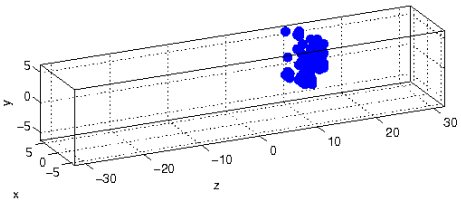
$t = 300$



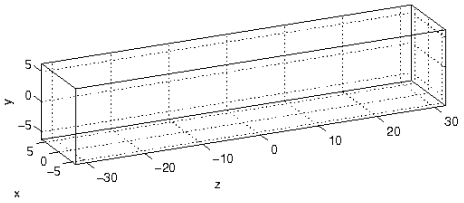
$t = 400$



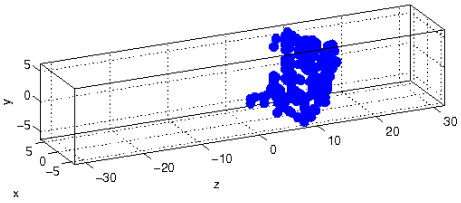
$t = 500$



$t = 600$

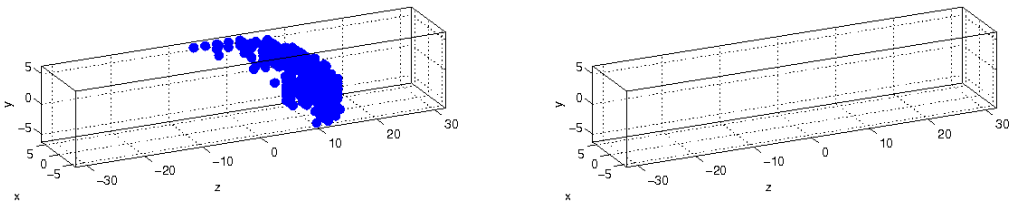


$t = 700$



$t = 800$

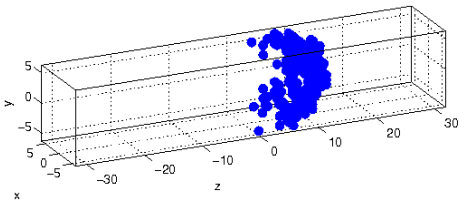
Figure 2.3.6: Open calcium release units throughout the cell using finite volume method with $\beta^{(i)} = (0, 0, 0.05), i = 1, \ldots, n_s$, on mesh size $32 \times 32 \times 128$.



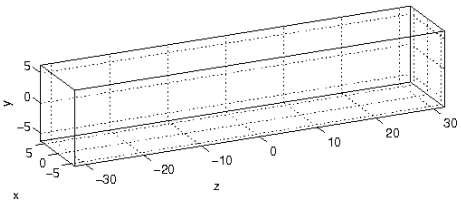
$t = 100$



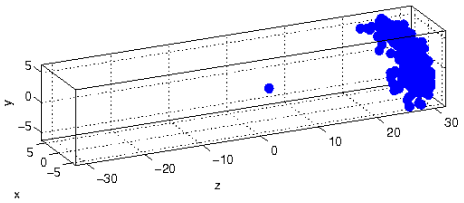
$t = 200$



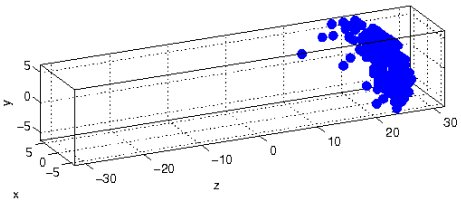
$t = 300$



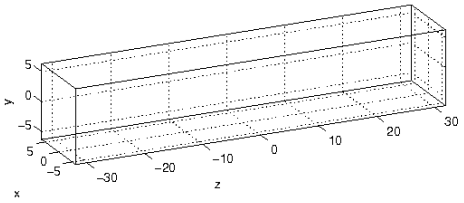
$t = 400$



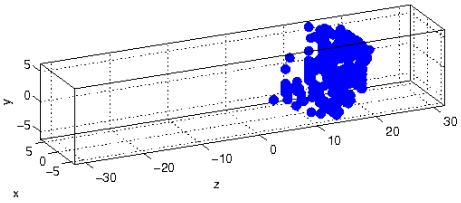
$t = 500$



$t = 600$

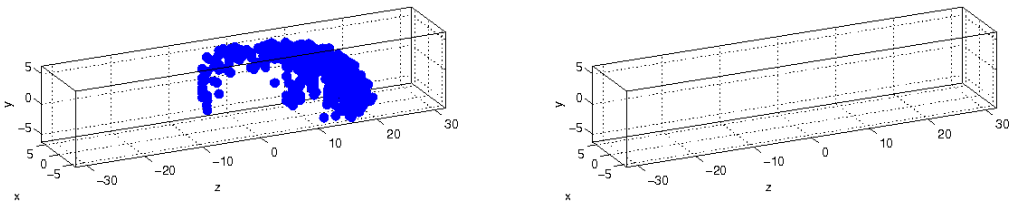


$t = 700$



$t = 800$

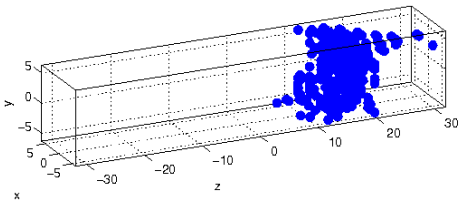
Figure 2.3.7: Open calcium release units throughout the cell using finite volume method with $\beta^{(i)} = (0, 0, 0.1), i = 1, \ldots, n_s$, on mesh size $32 \times 32 \times 128$.



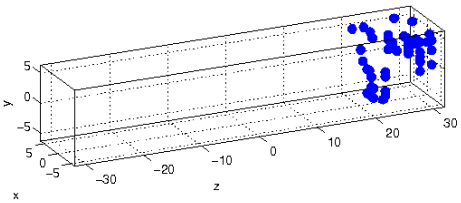
$t = 100$



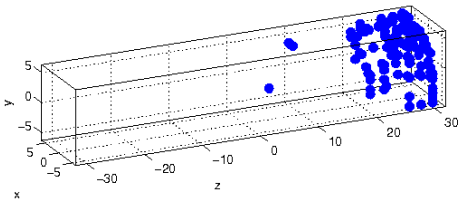
$t = 200$



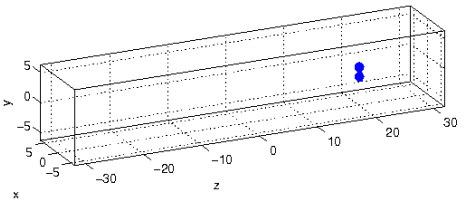
$t = 300$



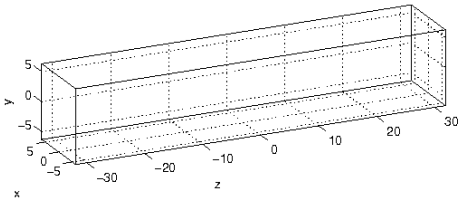
$t = 400$



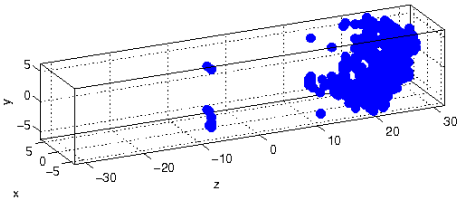
$t = 500$



$t = 600$

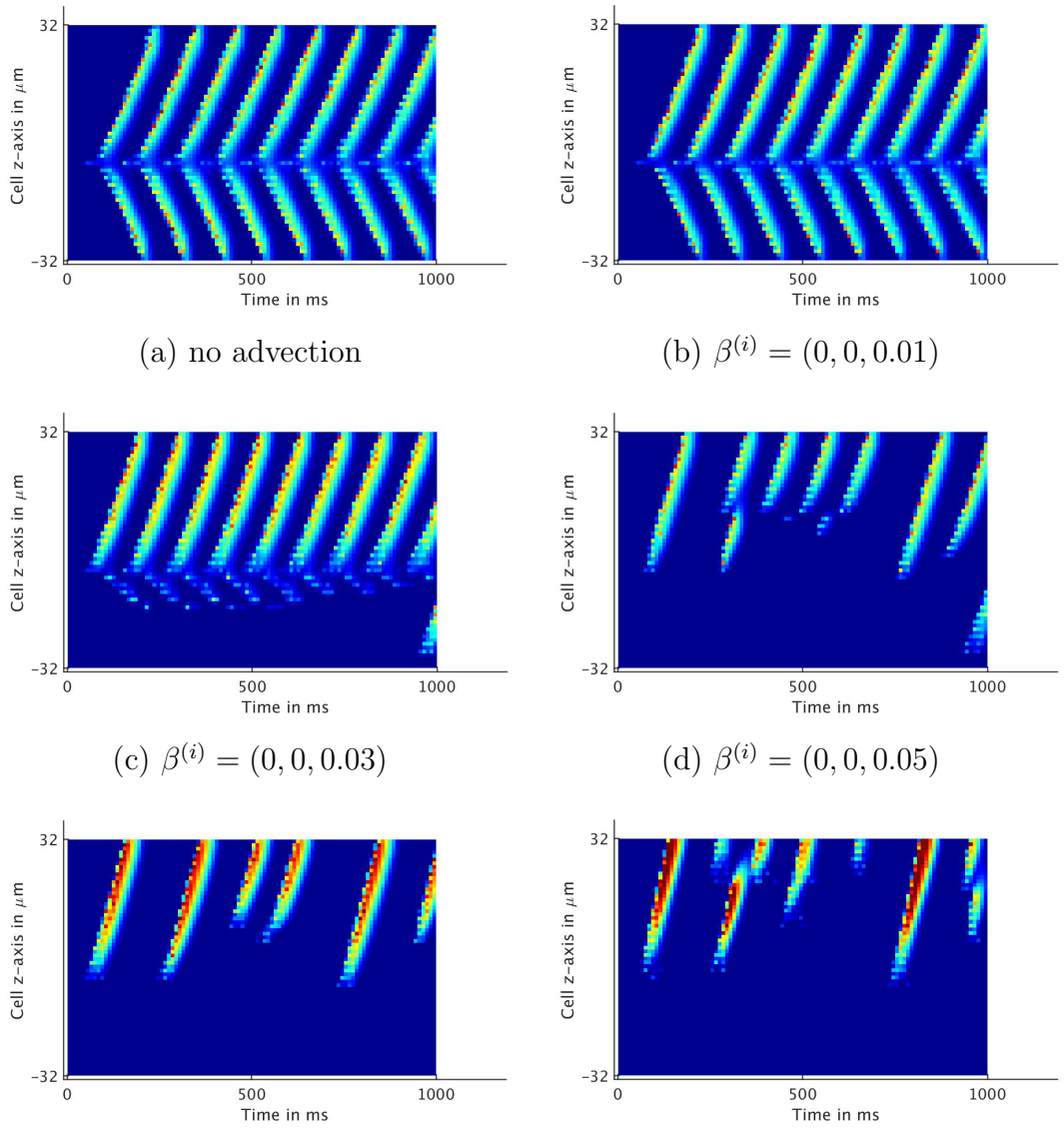


$t = 700$



$t = 800$

Figure 2.3.8: Open calcium release units throughout the cell using finite volume method with $\beta^{(i)} = (0, 0, 0.2), i = 1, \ldots, n_s$, on mesh size $32 \times 32 \times 128$.




(a) no advection

(b) $\beta^{(i)} = (0, 0, 0.01)$

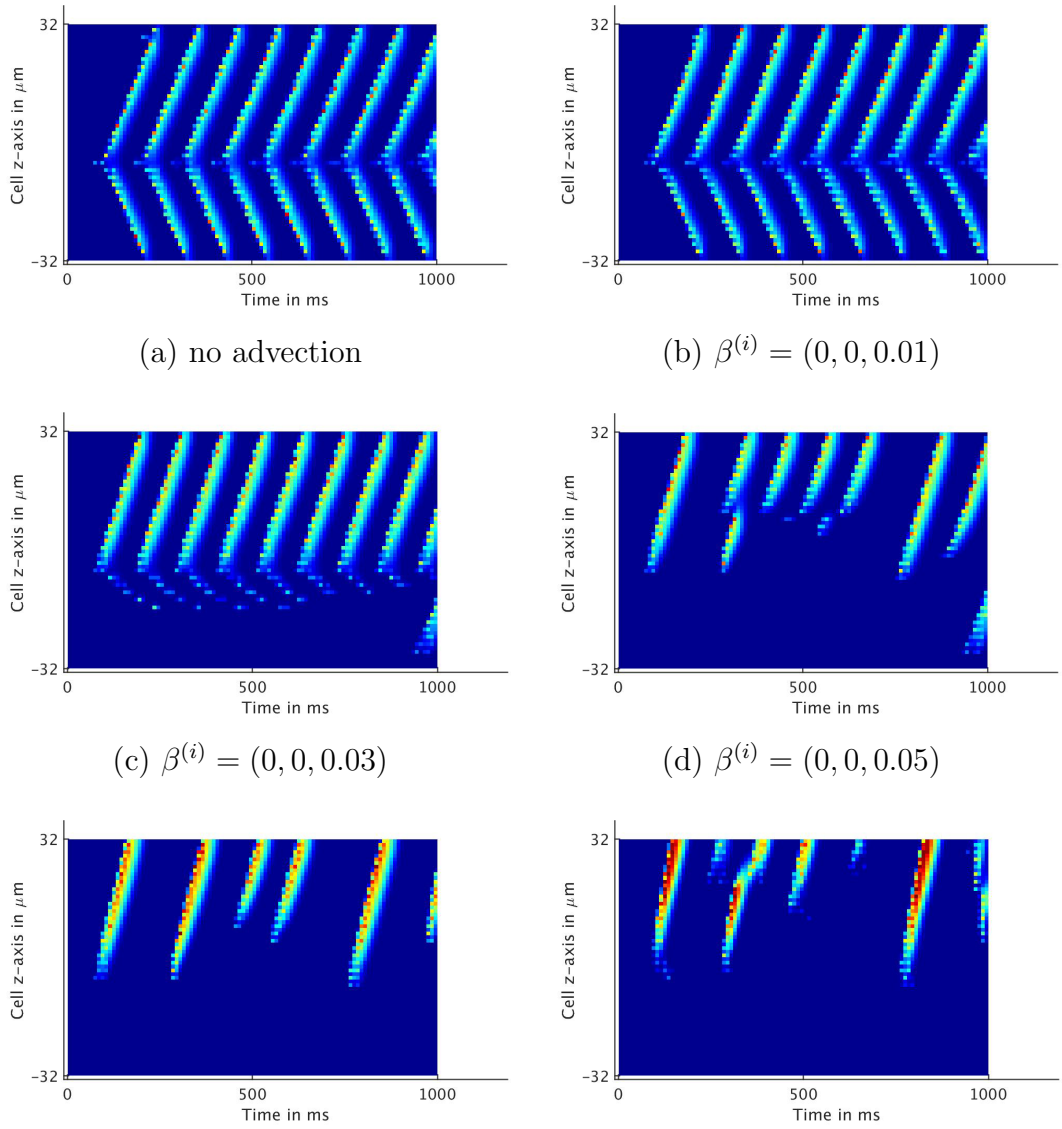
(c) $\beta^{(i)} = (0, 0, 0.03)$

(d) $\beta^{(i)} = (0, 0, 0.05)$

(e) $\beta^{(i)} = (0, 0, 0.1)$

(f) $\beta^{(i)} = (0, 0, 0.2)$

Figure 2.3.9: Time evolution of the longitudinal line scan showing the calcium concentration along the line $x = y = 0$ $\mu$m, $0 \leq t \leq 1{,}000$ ms.




(a) no advection

(b) $\beta^{(i)} = (0, 0, 0.01)$

(c) $\beta^{(i)} = (0, 0, 0.03)$

(d) $\beta^{(i)} = (0, 0, 0.05)$

(e) $\beta^{(i)} = (0, 0, 0.1)$

(f) $\beta^{(i)} = (0, 0, 0.2)$

Figure 2.3.10: Time evolution of the longitudinal line scan showing the calcium concentration along the line $x = y = 5.6$ $\mu$m, $0 \leq t \leq 1{,}000$ ms.

# CHAPTER 3

# NUMERICAL METHOD

In this chapter, we describe the numerical methods for the solution of the system (1.1.1)–(1.1.2). Section 3.1 discusses two spatial discretizations to be used in a method of lines approach, namely the finite element method and the finite volume method. Using both methods, we obtain large stiff ODE systems which need to be solved by an implicit ODE solver. Brief descriptions of the time integration, the non-linear solver, and the linear solvers are given in Section 3.2. Section 3.3 discusses the parallel implementation using MPI and the parallel implementation using CUDA with MPI. Section 3.4 contains results of a thorough convergence study of the finite volume method, which provide insights for the choices of discretization of the advection term under different circumstances.

## 3.1 Spatial Discretization

In Section 3.1.1, we give an outline of the finite element space discretization. In Section 3.1.2 the finite volume space discretization is explained. This also includes a discussion of the treatment of Dirac delta sources.

In order to numerically simulate the calcium spark model, a numerical method must be designed that is very efficient in memory use. The uniform rectangular CRU lattice gives a naturally induced regular numerical mesh. Additionally, the model uses constant diffusion coefficients. Using a finite element method (FEM) or finite volume method (FVM) with these properties (constant coefficients, regular mesh) will allow for system matrices whose components can be computed by analytical formulas. Therefore routines, specifically the matrix-vector product, can be designed without an explicitly stored system matrix. A matrix-free method dramatically reduces the

memory requirements of the method, thereby making computations on very fine mesh feasible.

The convergence of the finite element method in presence of measure valued source terms was rigorously shown in [27], and numerical results agree well with the theoretical predictions [7]. In addition, it is demonstrated in [26] that the FVM can also resolve the discontinuous forcing term in (1.1.1)–(1.1.2), despite the lack of theory on the numerical performance in this case. In fact, the integral definition of the Dirac delta distribution lends itself to a clean discrete formulation of the $J_{SR}$ function. Without considering advection, the finite element method is applied. Since we intend to consider systems with advection, this is done for the more general class of equation systems of advection-diffusion-reaction (ADR) type. For this type of problems, the finite volume method is a natural choice as opposed to the finite element method.

### 3.1.1 Spatial Discretization using the Finite Element Method

The method of lines approach using finite elements is appropriate for the system (1.1.1) without the advection term $\beta^{(i)} \cdot \left(\nabla u^{(i)}\right)$.

This approach takes advantage of the regular shape of the domain $\Omega$ and uses a uniform mesh of 3-D brick elements of size $\Delta x\, \Delta y\, \Delta z$. To derive the finite element discretization, we interpret the PDE for $u^{(i)}(\mathbf{x}, t)$ for $0 < t \leq t_{\text{fin}}$ in a weak sense. Integrating (1.1.1) without the advection term $\beta^{(i)} \cdot \left(\nabla u^{(i)}\right)$ against all test functions $v(\mathbf{x})$, we obtain

$$\int_\Omega v\, \frac{\partial u^{(i)}}{\partial t}\, d\mathbf{x} - \int_\Omega v\, \nabla \cdot \left(D^{(i)} \nabla u^{(i)}\right)\, d\mathbf{x} = \int_\Omega v\, q^{(i)}\, d\mathbf{x} \quad \text{for all } v \in V. \tag{3.1.1}$$

Using Green's theorem, (3.1.1) becomes

$$\int_\Omega v\, \frac{\partial u^{(i)}}{\partial t}\, d\mathbf{x} + \int_\Omega \nabla v \cdot \left(D^{(i)} \nabla u^{(i)}\right)\, d\mathbf{x} - \int_{\partial\Omega} v\ \mathbf{n} \cdot \left(D^{(i)} \nabla u^{(i)}\right)\, dS = \int_\Omega q^{(i)}\, v\, d\mathbf{x}, \tag{3.1.2}$$

where the term $\mathbf{n} \cdot \left(D^{(i)} \nabla u^{(i)}\right) = 0$ due to the no-flow boundary condition (2.1.1).

Hence, we obtain the weak form:

Find $u^{(i)}(\mathbf{x},t) \in V$ for $0 < t \le t_{\text{fin}}$ such that

$$\int_\Omega v\,\frac{\partial u^{(i)}}{\partial t}\,d\mathbf{x} + \int_\Omega \nabla v \cdot \left(D^{(i)}\nabla u^{(i)}\right)\,d\mathbf{x} = \int_\Omega q^{(i)}\,v\,d\mathbf{x} \quad \text{for all } v \in V. \tag{3.1.3}$$

In order to to approximate $u^{(i)}(\mathbf{x},t) \in V$ by a FEM solution $u_h^{(i)}(\mathbf{x},t) \in V_h$, we pick a finite-dimensional subspace $V_h \subset V$. Then we obtain the approximate weak form:

Find $u_h^{(i)}(\mathbf{x},t) \in V_h$ for $0 < t \le t_{\text{fin}}$ such that

$$\int_\Omega v\,\frac{\partial u_h^{(i)}}{\partial t}\,d\mathbf{x} + \int_\Omega \nabla v \cdot \left(D^{(i)}\nabla u_h^{(i)}\right)\,d\mathbf{x} = \int_\Omega q^{(i)}\,v\,d\mathbf{x} \quad \text{for all } v \in V_h. \tag{3.1.4}$$

Since $V_h$ is a finite-dimensional function space, it has a basis of finitely many basis functions $\{\varphi_k(\mathbf{x})\}_{k=1,\dots,N}$. This means concretely for the approximate weak form we have the expansion of the FEM solution $u_h^{(i)}(\mathbf{x},t) = \sum_{\ell=1}^N \mathbf{u}_\ell^{(i)}(t)\,\varphi_\ell(\mathbf{x})$ with coefficient functions $\mathbf{u}_\ell^{(i)}(t)$.

Thus, finding the FEM solution $u_h^{(i)}(\mathbf{x},t) \in V_h$ is equivalent to finding all the coefficient functions $\mathbf{u}_\ell^{(i)}(t)$, $\ell = 1,\dots,N$. Meanwhile, integrating against all test functions $v \in V_h$ is equivalent to considering $v = \varphi_k(\mathbf{x})$ for $k = 1,\dots,N$.

Therefore, the approximate weak form (3.1.4) can be rewritten as:

Find all $\mathbf{u}_\ell^{(i)}(t)$, $\ell = 1,\dots,N$, for $0 < t \le t_{\text{fin}}$ such that

$$\int_\Omega \varphi_k\,\frac{\partial}{\partial t}\left(\sum_{\ell=1}^N \mathbf{u}_\ell^{(i)}\varphi_\ell\right)\,d\mathbf{x} + \int_\Omega \nabla\varphi_k \cdot \left(D^{(i)}\nabla\left(\sum_{\ell=1}^N \mathbf{u}_\ell^{(i)}\varphi_\ell\right)\right)\,d\mathbf{x} = \int_\Omega q^{(i)}\,\varphi_k\,d\mathbf{x} \tag{3.1.5}$$

for all $k = 1,\dots,N$. Reorder terms in the approximate weak form (3.1.5), we obtain:

Find all $\mathbf{u}_\ell^{(i)}(t)$, $\ell = 1,\dots,N$, for $0 < t \le t_{\text{fin}}$ such that

$$\sum_{\ell=1}^N \int_\Omega \varphi_k\,\varphi_\ell\,d\mathbf{x}\,\frac{d\mathbf{u}_\ell^{(i)}}{dt} + \sum_{\ell=1}^N \int_\Omega \nabla\varphi_k \cdot \left(D^{(i)}\nabla\varphi_\ell\right)\,d\mathbf{x}\,\mathbf{u}_\ell^{(i)} = \int_\Omega q^{(i)}\,\varphi_k\,d\mathbf{x} \tag{3.1.6}$$

for all $k = 1,\dots,N$.

Let us define the unknown vector $\mathbf{u}^{(i)}(t) = (\mathbf{u}_\ell^{(i)}(t))$, the (lumped) mass matrices

$$\hat{M}^{(i)} = (\hat{M}_{k\ell}^{(i)}) = \left(\int_\Omega \varphi_k\,\varphi_\ell\,d\mathbf{x}\right), \tag{3.1.7}$$

the stiffness matrices

$$K^{(i)} = (K^{(i)}_{k\ell}) = \left( \int_\Omega \nabla\varphi_k \cdot \left( D^{(i)} \nabla\varphi_\ell \right) \, d\mathbf{x} \right), \tag{3.1.8}$$

and the vector on the right hand side

$$\mathbf{q}^{(i)} = (\mathbf{q}^{(i)}_k) = \left( \int_\Omega q^{(i)} \, \varphi_k \, d\mathbf{x} \right). \tag{3.1.9}$$

Then we can further simplify the approximate weak form to be:

Find $\mathbf{u}^{(i)}(t) \in \mathbb{R}^N$ for $0 < t \le t_{\text{fin}}$ such that

$$\hat{M}^{(i)} \, \frac{d\mathbf{u}^{(i)}}{dt} + K^{(i)} \, \mathbf{u}^{(i)} = \mathbf{q}^{(i)} \tag{3.1.10}$$

In the currently implemented model we have $n_s = 3$ species. Also, let us define

$$y = \begin{bmatrix} \mathbf{u}^{(1)} \\ \mathbf{u}^{(2)} \\ \mathbf{u}^{(3)} \end{bmatrix}, \quad \hat{M} = \begin{bmatrix} \hat{M}^{(1)} & & \\ & \hat{M}^{(2)} & \\ & & \hat{M}^{(3)} \end{bmatrix}, \quad K = \begin{bmatrix} K^{(1)} & & \\ & K^{(2)} & \\ & & K^{(3)} \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} \mathbf{q}^{(1)} \\ \mathbf{q}^{(2)} \\ \mathbf{q}^{(3)} \end{bmatrix}, \tag{3.1.11}$$

to collect the approximate weak forms (3.1.10) for all species $1 \le i \le 3$ into one system of ODEs for $y(t) \in \mathbb{R}^{3N}$. The result is the canonical initial value problem with mass matrix

$$M \, \frac{dy}{dt} = f^{\text{ode}}(t, y), \quad 0 < t \le t_{\text{fin}}, \quad y(0) = y_{\text{ini}}, \tag{3.1.12}$$

with $M := \hat{M}$ and $f^{\text{ode}}(t, y) := -K \, y + \mathbf{q}$.

This approach to discretizing the time-dependent PDE is the finite element method. The solution to the resulting stiff ODE system will be discussed in Section 3.2.

### 3.1.2 Spatial Discretization using the Finite Volume Method

The finite volume method (FVM) is appropriate for problems that contain both advection and diffusion [18].

To derive the finite volume discretization, let $\mathcal{T}_h = \{K_1, \ldots, K_M\}$ be a mesh such that $\overline{\Omega} = \bigcup_{l=1}^{M} \overline{K}_l$. Each $K_l \in \mathcal{T}_h$ is an open subset of $\Omega$, referred to as a cell or control volume. Integrating (1.1.1) over an arbitrary cell $K_l \in \mathcal{T}_h$ and applying the divergence theorem yields

$$\frac{d}{dt}\int_{K_l} u^{(i)}\,d\mathbf{x} - \int_{\partial K_l} (D^{(i)}\nabla u^{(i)} - u^{(i)}\beta^{(i)}) \cdot \mathbf{n}_l\,dS = \int_{K_l} q^{(i)}\,d\mathbf{x}, \tag{3.1.13}$$

where $\partial K_l$ denotes the boundary of $K_l$ and $\mathbf{n}_l$ its outward unit normal vector. This is the equation we are actually trying to solve, since it imposes less regularity on the solution than (1.1.1). In particular, solutions with discontinuities are now admissible. Denoting the volume of $K_l$ by $|K_l|$, the spatial mean value of $u^{(i)}$ over $K_l$ is given by

$$\bar{u}_l^{(i)}(t) := \frac{1}{|K_l|}\int_{K_l} u^{(i)}(\mathbf{x}, t)\,d\mathbf{x}. \tag{3.1.14}$$

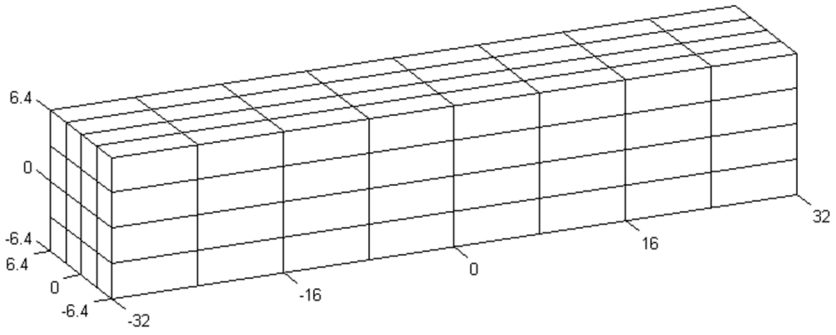
With this notation, (3.1.13) can be rewritten as

$$\frac{d}{dt}\bar{u}_l^{(i)} - \frac{1}{|K_l|}\int_{\partial K_l} (D^{(i)}\nabla u^{(i)} - u^{(i)}\beta^{(i)}) \cdot \mathbf{n}_l\,dS = \frac{1}{|K_l|}\int_{K_l} q^{(i)}\,d\mathbf{x}. \tag{3.1.15}$$

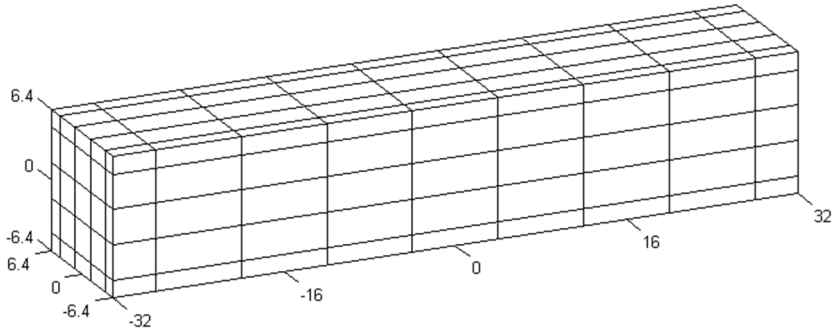
This is a system of ordinary differential equations for the temporal evolution of the mean values $\bar{u}_l^{(i)}$. With this expression, the crucial issues are to compute the boundary fluxes in terms of neighboring mean values, as well as the volume integral on the right hand side.

We now introduce the mesh which is used for this discretization. In [7] a regular mesh $\Omega_h \subset \overline{\Omega}$ with constant mesh spacings $\Delta x$, $\Delta y$, and $\Delta z$ was used for the finite element space discretization. Here we employ the corresponding dual mesh $\mathcal{T}_h$, which is constructed by connecting all centers of mesh cubes of $\Omega_h$ with lines parallel to the coordinate axes and extending these lines in a straight manner to the boundary.

The dual mesh is a rectilinear mesh and each inner node of $\Omega_h$ is the center of a cell of $\mathcal{T}_h$ with volume $\Delta x\,\Delta y\,\Delta z$. Furthermore, the volume of a cell is reduced to $\Delta x\Delta y\Delta z/2$, $\Delta x\Delta y\Delta z/4$, or $\Delta x\Delta y\Delta z/8$, if this cell has a common face, edge or corner, respectively, with the boundary $\partial\Omega$. The exterior views of a simple regular mesh and the corresponding dual mesh are depicted in Figures 3.1.1 (a) and (b), respectively.




(a)




(b)

Figure 3.1.1: (a) Regular mesh of mesh resolution $4 \times 4 \times 8$ and (b) its dual mesh.

By construction the number of nodes of $\Omega_h$ equals the number of cells in $\mathcal{T}_h$. In the following, we assume that $\mathcal{T}_h$ consists of $M = M_x M_y M_z$ control volumes, with $M_x$, $M_y$, and $M_z$ denoting the number of cells in each direction and introduce the

enumeration scheme

$$l = i + (j-1)M_x + (k-1)M_xM_y \tag{3.1.16}$$

for $1 \le i \le M_x$, $1 \le j \le M_y$, and $1 \le k \le M_z$. Thus, a control volume $K_l$ has the neighbors $K_{l-1}$ and $K_{l+1}$ in the $x$-direction, $K_{l-M_x}$ and $K_{l+M_x}$ in the $y$-direction, and $K_{l-M_xM_y}$ and $K_{l+M_xM_y}$ in the $z$-direction.

For a given control volume $K_l = (x_L, x_R) \times (y_L, y_R) \times (z_L, z_R) \in \mathcal{T}_h$ with spacings $\Delta x_l = x_R - x_L, \Delta y_l = y_R - y_L, \Delta z_l = z_R - z_L$, and $\beta = (\beta_1, \beta_2, \beta_3)$, the boundary fluxes from (3.1.13) can be written as

$$\begin{aligned}
&\int_{\partial K_l} (D\nabla u - u\beta) \cdot \mathbf{n}_l \, dS \qquad (3.1.17)\\
&\quad = \int_{z_L}^{z_R}\int_{y_L}^{y_R} (D_{11}\partial_x u - \beta_1 u)|_{x=x_R} - (D_{11}\partial_x u - \beta_1 u)|_{x=x_L} \, dy\, dz\\
&\quad + \int_{z_L}^{z_R}\int_{x_L}^{x_R} (D_{22}\partial_y u - \beta_2 u)|_{y=y_R} - (D_{22}\partial_y u - \beta_2 u)|_{y=y_L} \, dx\, dz\\
&\quad + \int_{y_L}^{y_R}\int_{x_L}^{x_R} (D_{33}\partial_z u - \beta_3 u)|_{z=z_R} - (D_{33}\partial_z u - \beta_3 u)|_{z=z_L} \, dx\, dy.
\end{aligned}$$

Here we have exploited the fact that the faces of $K_l$ are parallel to the planes defined by the coordinate axes, so only one entry of the corresponding normal vectors is non-zero. The superscripts $(i)$ were dropped for readability. Note that the above equation is valid for each of the $n_s$ species of the system.

In order to approximate the advective flux $\beta_j u(x_{l+1})$ in terms of the mean values $\bar{u}_l$, we introduce a numerical flux function $H = H(\hat{u}_l^+, \hat{u}_{l+1}^-)$. The values $\hat{u}_l^+$ and $\hat{u}_{l+1}^-$ are approximations to the unknown values of $u$ on either side of $x_{l+1}$. Assuming that $\beta_j \ge 0$ as before, either a first-order or second-order accurate upwind flux function can be defined:

- Assuming a constant distribution of $u$ in $K_l$ leads to the first-order discretization

$$H_{a,j}(\hat{u}_l^+, \hat{u}_{l+1}^-) = \beta_j \hat{u}_l^+, \quad j = 1,2,3, \qquad \hat{u}_l^+ = \bar{u}_l. \tag{3.1.18}$$

- Since the advection term is linear, no non-linear discretization is necessary, as would be standard for non-linear fluid mechanics. To obtain the second-order discretization via a linear representation of the solution, we define a slope within the cell by

$$H_{a,j}(\hat{u}_l^+, \hat{u}_{l+1}^-) = \beta_j \hat{u}_l^+, \quad j = 1,2,3, \qquad \hat{u}_l^+ = \bar{u}_l + \frac{\bar{u}_l - \bar{u}_{l-1}}{\mathbf{m}_l - \mathbf{m}_{l-1}}(x_{l+1} - \mathbf{m}_l), \tag{3.1.19}$$

  where $\mathbf{m}_l$ and $\mathbf{m}_{l-1}$ denote the barycenters of the cells belonging to $\bar{u}_l$ and $\bar{u}_{l-1}$, respectively.

To approximate the diffusive flux $D_{jj}\partial_l u(x_{l+1})$, we use a central difference of the neighboring mean values and define the diffusive flux function as

$$H_{d,j}(\bar{u}_l, \bar{u}_{l+1}) = D_{jj}\frac{\bar{u}_{l+1} - \bar{u}_l}{\mathbf{m}_{l+1} - \mathbf{m}_l}. \tag{3.1.20}$$

In this case, the mean values on either side of $x_{l+1}$ are sufficient to obtain a scheme of second order. Now we can approximate the boundary fluxes as

$$\begin{aligned}
&\int_{\partial K_l} (D\nabla u - u\beta)\cdot \mathbf{n}_l \, dS \qquad (3.1.21)\\
&\quad \approx \Delta y_l \Delta z_l \big(H_{d,1}(\bar{u}_l, \bar{u}_{l+1}) - H_{a,1}(\hat{u}_l^{1,+}, \hat{u}_{l+1}^{1,-})\big)\\
&\quad - \Delta y_l \Delta z_l \big(H_{d,1}(\bar{u}_{l-1}, \bar{u}_l) - H_{a,1}(\hat{u}_{l-1}^{1,+}, \hat{u}_l^{1,-})\big)\\
&\quad + \Delta x_l \Delta z_l \big(H_{d,2}(\bar{u}_l, \bar{u}_{l+M_x}) - H_{a,2}(\hat{u}_l^{2,+}, \hat{u}_{l+M_x}^{2,-})\big)\\
&\quad - \Delta x_l \Delta z_l \big(H_{d,2}(\bar{u}_{l-M_x}, \bar{u}_l) - H_{a,2}(\hat{u}_{l-M_x}^{2,+}, \hat{u}_l^{2,-})\big)\\
&\quad + \Delta x_l \Delta y_l \big(H_{d,3}(\bar{u}_l, \bar{u}_{l+M_x M_y}) - H_{a,3}(\hat{u}_l^{3,+}, \hat{u}_{l+M_M M_y}^{3,-})\big)\\
&\quad - \Delta x_l \Delta y_l \big(H_{d,3}(\bar{u}_{l-M_x M_y}, \bar{u}_l) - H_{a,3}(\hat{u}_{l-M_x M_y}^{3,+}, \hat{u}_l^{3,-})\big),
\end{aligned}$$

where the enumeration scheme (3.1.16) was used to describe the location of the input data of the flux functions. The notation $\hat{u}_l^{j,\pm}$ indicates that the approximation to the

value of $u$ lives in $K_l$ and belongs to the face which is between $K_l$ and its neighbor of positive or negative side.

The last step of the discretization is the treatment of the volume integral on the right hand side of (3.1.13). This can be approximated adequately by the midpoint rule

$$\int_{K_l} q(u^{(1)}, \dots, u^{(n_s)}, \mathbf{x}, t)\, d\mathbf{x} \approx |K_l|\, q(\bar{u}_l^{(1)}, \dots, \bar{u}_l^{(n_s)}, \mathbf{m}_l, t), \tag{3.1.22}$$

with $\mathbf{m}_l$ denoting the barycenter of $K_l$. In the special case of a Dirac delta distribution as source term, i.e., $q = \delta(\mathbf{x} - \hat{\mathbf{x}})$, the volume integral can be computed exactly. The Dirac delta distribution is defined by requiring $\delta(\mathbf{x} - \hat{\mathbf{x}}) = 0$ for all $\mathbf{x} \neq \hat{\mathbf{x}}$ and $\int_{\mathbb{R}^3} \psi(\mathbf{x})\delta(\mathbf{x} - \hat{\mathbf{x}})\, d\mathbf{x} = \psi(\hat{\mathbf{x}})$ for any function $\psi \in C_0^\infty(\mathbb{R}^3)$. Thus, we obtain

$$\int_{K_l} q(\mathbf{x})\, d\mathbf{x} = \int_{K_l} \delta(\mathbf{x} - \hat{\mathbf{x}}) \cdot 1\, d\mathbf{x} = \begin{cases} 1, & \hat{\mathbf{x}} \in K_l, \\ 0, & \hat{\mathbf{x}} \notin K_l. \end{cases} \tag{3.1.23}$$

This completes the description of the finite volume discretization and, after division by the local volume $|K_l| = \Delta x_l \Delta y_l \Delta z_l$, (3.1.15) can now be written as

$$\begin{aligned} &\frac{d}{dt}\bar{u}_l - \frac{1}{\Delta x_l}\big(H_{d,1}(\bar{u}_l, \bar{u}_{l+1}) - H_{a,1}(\hat{u}_l^{1,+}, \hat{u}_{l+1}^{1,-})\big) \\ &\quad + \frac{1}{\Delta x_l}\big(H_{d,1}(\bar{u}_{l-1}, \bar{u}_l) - H_{a,1}(\hat{u}_{l-1}^{1,+}, \hat{u}_l^{1,-})\big) \\ &\quad - \frac{1}{\Delta y_l}\big(H_{d,2}(\bar{u}_l, \bar{u}_{l+M_x}) - H_{a,2}(\hat{u}_l^{2,+}, \hat{u}_{l+M_x}^{2,-})\big) \\ &\quad + \frac{1}{\Delta y_l}\big(H_{d,2}(\bar{u}_{l-M_x}, \bar{u}_l) - H_{a,2}(\hat{u}_{l-M_x}^{2,+}, \hat{u}_l^{2,-})\big) \\ &\quad - \frac{1}{\Delta z_l}\big(H_{d,3}(\bar{u}_l, \bar{u}_{l+M_xM_y}) - H_{a,3}(\hat{u}_l^{3,+}, \hat{u}_{l+M_xM_y}^{3,-})\big) \\ &\quad + \frac{1}{\Delta z_l}\big(H_{d,3}(\bar{u}_{l-M_xM_y}, \bar{u}_l) - H_{a,3}(\hat{u}_{l-M_xM_y}^{3,+}, \hat{u}_l^{3,-})\big) \\ &\quad = q(\bar{u}_l^{(1)}, \dots, \bar{u}_l^{(n_s)}, \mathbf{m}_l, t). \end{aligned} \tag{3.1.24}$$

(3.1.25)

Let $\bar{\mathbf{u}}^{(i)} = (\bar{u}_1^{(i)}, \ldots, \bar{u}_M^{(i)})^T, \mathbf{q}^{(i)} = (q_1^{(i)}, \ldots, q_M^{(i)})^T, q_l^{(i)} = q^{(i)}(\bar{u}_l^{(1)}, \ldots, \bar{u}_l^{(n_s)}, \mathbf{m}_l, t)$, (3.1.24) now reads

$$\frac{d}{dt}\bar{\mathbf{u}}^{(i)} = \left(\mathbf{H}_{\text{diff}}^{(i)} - \mathbf{H}_{\text{adv}}^{(i)}\right)\bar{\mathbf{u}}^{(i)} + \mathbf{q}^{(i)}(\bar{\mathbf{u}}^{(1)}, \ldots, \bar{\mathbf{u}}^{(n)}), \tag{3.1.26}$$

where $\mathbf{H}_{\text{diff}}, \mathbf{H}_{\text{adv}} \in \mathbb{R}^{M\times M}$ are the flux matrices, built from terms containing $H_{d,j}$ and $H_{a,j}$ in the system of equations (3.1.24). Finally, collecting all $n_s$ vectors $\bar{\mathbf{u}}^{(i)}$ in $\bar{\mathbf{U}} \in \mathbb{R}^{n_s M}$ the system (3.1.26) can be written as

$$\frac{d}{dt}\bar{\mathbf{U}}(t) = \mathbf{f}^{\text{ode}}(t, \bar{\mathbf{U}}(t)) \tag{3.1.27}$$

with $\mathbf{f}^{\text{ode}} = (\mathbf{f}^{(1)}, \ldots, \mathbf{f}^{(n)})^T \in \mathbb{R}^{n_s M}$ and components

$$\mathbf{f}^{(i)} = \mathbf{A}^{(i)}\bar{\mathbf{u}}^{(i)} + \mathbf{q}^{(i)}(\bar{\mathbf{u}}^{(1)}, \ldots, \bar{\mathbf{u}}^{(n_s)}), \tag{3.1.28}$$

with

$$\mathbf{A}^{(i)} = \mathbf{H}_{\text{diff}}^{(i)} - \mathbf{H}_{\text{adv}}^{(i)}. \tag{3.1.29}$$

## 3.2 Other Numerical Components

Both spatial discretization approaches yield large stiff ODE systems (3.1.12) and (3.1.27), respectively. This section gives a brief description of the time integration, the non-linear solver, and the linear solver.

### 3.2.1 Time Integration and Matrix-Free Implementation

The spatial discretization of the application problem (3.1.27) with $n_s = 3$ species using the finite volume method with $M = M_x M_y M_z$ control volumes results in a system of non-linear ordinary differential equations (ODEs) with $n_{eq} = n_s\, M$ degrees of freedom (DOF). A method of lines discretization of advection-diffusion-reaction equations with second-order spatial derivatives results necessarily in a stiff ODE system, since the time step size restrictions due to the CFL condition are considered too severe to allow for explicit time-stepping methods. To reach the very large final times

demanded to simulate laboratory time scales of the CICR application, we need to be able to take fairly large time steps whenever possible. This necessitates the use of a sophisticated ODE solver such as the family of numerical differentiation formulas (NDF$k$) with variable order $1 \leq k \leq 5$ and adaptively chosen time step size [28, 29]. This method is also used for a method of lines discretization using finite elements in [7]. We use relative and absolute tolerances of $10^{-6}$ and $10^{-8}$, respectively, for the error estimator of the NDF$k$ method. In studies for the CICR problem, the time step sizes vary widely, with fine step sizes on the order of $10^{-5}$ ms immediately after CRUs open or close. Since the parabolic system is smooth away from these times, the time steps increase steadily up to the order of $10^{-2}$ ms, while the error controller ensures that the total error incurred from the time-stepping remains bounded by the selected tolerances. This high variation in step size allows the solver to reach the desired final time of 1,000 ms in under 90,000 time steps for the finest mesh. The average method order observed is 3, showing that we are profiting significantly from the variable order method.

We use the numerical differentiation formulas (NDF$k$) with variable order $1 \leq k \leq 5$ and adaptively chosen time step size, see [28, 29] for details. This implicit method demands the solution of a non-linear system in each time step. For its solution, a matrix-free method is applied, which means that results of the Jacobian-vector products needed in the Krylov subspace method are provided directly without storing the Jacobian. The purpose of this approach is to save memory and hence to allow for computations on very fine meshes. In addition, the usage of the exact Jacobian should lead to quadratic convergence of the Newton method. Furthermore, a matrix-free implementation of diffusive and advective flux matrices is implemented to take advantage of the structured mesh.

### 3.2.2 The Non-linear Solver

The implicit ODE method needed for a stiff problem demands the solution of a non-linear system $\mathbf{F}^{newt}(\bar{\mathbf{U}}) = 0$ of $n_{eq}$ equations at every time step. The Newton method is used with Jacobian $\mathbf{J}^{newt}(\bar{\mathbf{U}}) = \nabla_{\bar{\mathbf{U}}}\mathbf{F}^{newt}(\bar{\mathbf{U}})$. This method profits from the low-order spatial discretization on a uniform mesh used, because we are able to compute analytically all the matrices in (3.1.24), as originally demonstrated in [9] for the method of lines using finite elements. The purpose of this approach is to save memory and hence allow for computations on very fine meshes. In addition, the usage of the exact Jacobian should lead to quadratic convergence of the Newton method. The iteration is stopped if $\|\bar{\mathbf{U}}_{\text{new}}\| < \varepsilon^{\text{newt}}\|\bar{\mathbf{U}}_{\text{new}} - \bar{\mathbf{U}}_{\text{old}}\|$. We use the tolerance of $\varepsilon^{\text{newt}} = 10^{-4}$ and maximum number of Newton iterations of 4. Since the matrix-vector products in Krylov subspace methods [25] used as linear solvers below are implemented in matrix-free form, this Jacobian is automatically evaluated at the current Newton iteration without any additional cost.

### 3.2.3 Krylov Subspace Methods

The comparison of Krylov subspace methods below is part of the work that supports our choice of using BiCGSTAB as linear solver in [26] and [13].

At each Newton iteration, we need to solve a linear system with a non-symmetric system matrix for the $n_{eq}$ unknowns. Numerical experiments demonstrate that the biconjugate gradient stabilized method (BiCGSTAB) is preferable over GMRES as well as to QMR, the latter one being used with the finite element method in [7]. We stop the iteration within the linear solver if the residual $\mathbf{r}$ satisfies the condition $\|\mathbf{r}\|_2 < \varepsilon^{lin}\|\mathbf{b}\|_2$ with $\mathbf{b}$ denoting the right-hand side of the linear system and a given tolerance $\varepsilon^{lin}$. We use a tolerance of $\varepsilon^{lin} = 10^{-6}$.

The essential part of the method described in Section 3.2 (or [7]) is the solution

of the linear systems within the Newton method. Here we investigate the performance of different Krylov subspace methods simulating the calcium flow in heart cells, introduced in Section 2.1.

In [7], QMR is used to solve the linear systems, and thus, the availability of the transpose of the Jacobian is essential. This is not an issue in [7], since the sub-blocks of the corresponding Jacobian are symmetric. In the situation of Section 3.1.2, this becomes an important issue, since the advective and diffusive flux matrices are not symmetric. Hence, the coding of the matrix-free transpose is as hard to implement as the original matrix.

In [15], another method for the solution of the calcium model is described. Therein also Newton's method is used to solve the non-linear systems and a preconditioned version of GMRES was chosen to solve the linear systems.

Table 3.2.1 and Table 3.2.2 show comparisons of different Krylov subspace methods used in long time simulations of the CICR model. See Table 2.1.1 for all parameters of the simulations. The data in Table 3.2.1 comes from simulations using the finite element method of [7], Table 3.2.2 stems from finite volume simulations. Both tables show the wall clock times of the simulations, as well as the average number of iterations per linear solve, i.e., the total sum of linear iterations divided by the number of linear systems, for various meshes. Both tables show that BiCGSTAB is faster than the variants of GMRES and clearly preferable over QMR. This also shows that the extra effort of an implementation of the transpose of the Jacobian is not necessary, since BiCGSTAB and GMRES obtain better results without requiring matrix-vector products with the transpose.

Table 3.2.1: Comparison of linear solvers in the finite element method.

| (a) Wall clock time | | | | | |
|---|---|---|---|---|---|
| | BiCGSTAB | GMRES(5) | GMRES(10) | GMRES(20) | QMR |
| $16 \times 16 \times 64$ | 00:00:24 | 00:00:25 | 00:00:24 | 00:00:23 | 00:00:32 |
| $32 \times 32 \times 128$ | 00:13:00 | 00:15:39 | 00:15:29 | 00:15:18 | 00:19:17 |
| $64 \times 64 \times 256$ | 01:18:41 | 01:35:14 | 01:30:15 | 01:30:26 | 01:59:30 |
| $128 \times 128 \times 512$ | 74:18:23 | 90:49:36 | 90:57:59 | 88:15:39 | 113:50:44 |
| (b) Average iteration count per linear solve | | | | | |
| | BiCGSTAB | GMRES(5) | GMRES(10) | GMRES(20) | QMR |
| $16 \times 16 \times 64$ | 3.4835 | 6.1124 | 5.527 | 5.401 | 5.231 |
| $32 \times 32 \times 128$ | 2.0602 | 3.4426 | 3.3792 | 3.3792 | 3.1082 |
| $64 \times 64 \times 256$ | 2.3252 | 4.0419 | 3.8977 | 3.8927 | 3.6152 |
| $128 \times 128 \times 512$ | 2.6836 | 4.7638 | 4.7638 | 4.6338 | 4.2298 |

Table 3.2.2: Comparison of linear solvers in the finite volume method.

| (a) Wall clock time | | | | | |
|---|---|---|---|---|---|
| | BiCGSTAB | GMRES(5) | GMRES(10) | GMRES(20) | QMR |
| $16 \times 16 \times 64$ | 00:01:19 | 00:01:35 | 00:01:35 | 00:01:34 | 00:01:53 |
| $32 \times 32 \times 128$ | 00:06:57 | 00:08:38 | 00:08:37 | 00:08:38 | 00:10:24 |
| $64 \times 64 \times 256$ | 01:01:44 | 01:15:27 | 01:15:23 | 01:15:15 | 01:34:42 |
| $128 \times 128 \times 512$ | 40:01:06 | 90:50:26 | 90:49:09 | 88:12:30 | 113:51:38 |
| (b) Average iteration count per linear solve | | | | | |
| | BiCGSTAB | GMRES(5) | GMRES(10) | GMRES(20) | QMR |
| $16 \times 16 \times 64$ | 1.0585 | 1.8064 | 1.8017 | 1.8017 | 1.8589 |
| $32 \times 32 \times 128$ | 1.0964 | 1.7944 | 1.7944 | 1.7944 | 1.8475 |
| $64 \times 64 \times 256$ | 1.2272 | 2.0169 | 2.0158 | 2.0159 | 2.0669 |
| $128 \times 128 \times 512$ | 1.5131 | 4.7638 | 4.7638 | 4.6338 | 4.2298 |

## 3.3 Parallel Implementation

This section introduces two parallel implementations. The first implementation uses MPI for parallel communications and runs on CPU only nodes. The second implementation uses CUDA with MPI and runs on hybrid CPU/GPU nodes.

### 3.3.1 Parallel Implementation with MPI

Parallel computing breaks a problem into smaller ones, which are then solved concurrently. The implementation using MPI has two key advantages: (i) It enables the simulations on fine meshes and (ii) it speeds up the computations sufficiently to enable simulations up to large final times within a reasonable amount of wall clock time.

The code is written in C with MPI commands for the parallel communications for maximum portability. We split the domain $\Omega$ into non-overlapping sub-domains, with one on each of the $p$ parallel processes, by cutting in the (long) $z$ dimension of $\Omega$. As a result, the $(N_z + 1)$ mesh points are block-distributed to the $p$ parallel processes. This choice makes the $x$-$y$ planes of nodes whose values need to be exchanged between neighboring processes as small as possible, i.e., it is the optimal decomposition in a graph partitioning sense. Our code is capable of using any number of parallel processors within a limit determined by the mesh size. Since each process must have at least one mesh point in the $z$-direction, the number of processes $p$ must not be larger than the number of $(N_z + 1)$ mesh points. In other words, combinations of $p$ and $N_z$ with $p > (N_z + 1)$ are not feasible.

The communications between neighboring processes occur in each matrix-vector product needed in the Krylov subspace method. They are implemented by non-blocking `MPI_Isend` and `MPI_Irecv` commands. These have proven to be faster than blocking communication commands and as fast as any other MPI point-to-

point communication commands available. `MPI_Allreduce` commands are needed for all norm computations as well as for various diagnostic quantities such as minimum and maximum values of the solutions. The wall clock times are measured using `MPI_Barrier` and `MPI_Wtime` in sequence at the beginning and end of the run. We use the `genrand()` function from [20], with different seeds on each parallel process, to generate sequences of uniformly distributed pseudo-random numbers.

Table 3.3.1 summarizes several key parameters of the MPI implementation solved with the finite element method. The first three columns show the spatial mesh resolution of $N_x \times N_y \times N_z$, the number of mesh points $N = (N_x + 1)(N_y + 1)(N_z + 1)$, and their associated numbers of unknowns $n_s N$ for the $n_s$ species that need to be computed at every time step, commonly referred to as degrees of freedom (DOF). The following column lists the number of time steps taken by the ODE solver, which are significant and which increase with finer resolutions. The final two columns list the memory usage in GB, both predicted by counting variables in the implementation and by observation provided in a memory log file produced from the performance run. We notice that even the finest resolution fits comfortably in the memory of one NVIDIA K20 GPU. The same degrees of freedom (DOF) applies to the finite volume method, and also time steps using the finite volume method are comparable.

Results using this implementation are reported in Section 4.1 and Section 4.2.

### 3.3.2 Parallel Implementation using CUDA and MPI

This section is an extension of parallel implementation using CUDA and MPI in [11] and [8].

In the CUDA programming language, the CPU and the system's memory are referred to as host, and the GPU and its memory are referred to as device. Figure 3.3.1 explains how threads are grouped into blocks, and blocks grouped into grids. Threads

Table 3.3.1: Sizing study listing the mesh resolution $N_x \times N_y \times N_z$, the number of mesh points $N = (N_x+1)(N_y+1)(N_z+1)$, the number of degrees of freedom (DOF $= n_s N$) for the CICR problem with $n_s = 3$ species, the number of time steps taken by the ODE solver, and the predicted and observed memory usage in MB for a one-process run.

| $N_x \times N_y \times N_z$ | $N$ | DOF = 3 N | number of time steps | memory usage (GB) | |
|---|---|---|---|---|---|
| | | | | predicted | observed |
| $32 \times 32 \times 128$ | 140,481 | 421,443 | 58,416 | 0.05 | 0.08 |
| $64 \times 64 \times 256$ | 1,085,825 | 3,257,475 | 73,123 | 0.41 | 0.48 |
| $128 \times 128 \times 512$ | 8,536,833 | 25,610,499 | 89,088 | 3.24 | 3.68 |

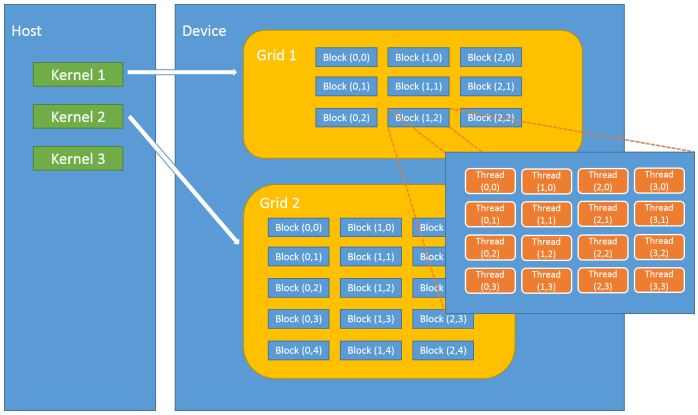



Figure 3.3.1: Schematic of blocks and threads.

unite into thread blocks – one-, two-, or three-dimensional grids of threads that interact with each other via shared memory and synchpoints. A program (kernel) is executed over a grid of thread blocks. One grid is executed at a time. Each block can also be one-, two-, or three-dimensional in form. This gives much flexibility in launching kernels with different structure of the data. However, there are still limitations such as the maximum dimension size of a thread block is $(1024, 1024, 64)$, and the maximum number of threads per block is 1024 for the K20 GPU we have.

The parallel implementation in this section is an extension of the one described in

Section 3.3.1, which uses MPI for parallel communications. Therefore, it inherited the main structure of the C program. However, to enable efficient calculations on GPU, almost all calculations have been redesigned to take advantage of GPU parallelism. While inputs are still managed by the host, C structs are shared by host and device. Large arrays are allocated directly on the device memory before numerical iterations, hence to prevent frequent communications between host and device. Functions are written in CUDA format from the start, to keep consistency and allow for future improvement. Since the host handles MPI communication and output to files, data communications between host and device occur before and after these events. The CUDA program has several levels, a detailed discussion is as follows:

The uppermost level is where computational resources are managed. First, MPI processes are setup in `main.cu`. After detecting the number of CUDA capable devices (NVIDIA GPUs) on each node, the `main.cu` function then sets the CUDA device for each MPI process. The idea behind this setup is to allow each MPI process to have access to a unique GPU device. If more than one MPI process is accessing the same GPU, kernels will queue up and the performance will be degraded. In our cluster, each GPU enabled node has two GPUs, each connected to a CPU socket via PCI bus, as shown in Figure 1.2.3. This means the best approach is to request 2 processes from each node and have each MPI process utilize a unique GPU. We can also run the program in serial, with a single MPI process and one GPU. The 5 GB GPU memory can hold the memory allocated for all large arrays for our current finest mesh $128 \times 128 \times 512$. After the computational resources are correctly allocated, the `main.cu` program launches the program that does the actual computation. Lastly, the `main.cu` program records the total memory used for each CPU node.

At the next level, the program selects the right solver based on a set of parameters. The code is a package that can solve many different problems with one-dimensional,

two-dimensional and three-dimensional domains. Including parabolic PDE systems, the classic Poisson problem, the power method, etc. In the case of the CICR model, `main.cu` call the function `run_parabolic.cu`, and the other model problems can be programmed in the same fashion. Within `problem_par_3d.cu`, we first read parameters from one input file, set up C structs, allow them to be shared by both C and CUDA code, and write arrays associated to the structs. These steps are achieved by working with `inputs.cu` and `struct_define.h`. Parameters in Table 2.1.1 are read in and set up in these steps. A crucial arrangement in the code is to put C struct type definitions in one header file called `struct_define.h`. After the definition of various struct data types, we need to declare each struct with `extern`. Just as we declare all other functions with `extern "C"`, this allows the mixture of C and CUDA to work properly. This header file is then included by every other `.cu` file. However, to be able to compile correctly, we also need to put regular struct declarations at the beginning of `main.cu`. After the initialization of structs and parameters, the function `run_parabolic.cu` then allocates memory for the solution for the PDE system on the CPU memory.

At level three the ODE solver is called. The solver is based on numerical differentiation formulas (NDF$k$) with variable order $1 \leq k \leq 5$ and adaptively chosen time step size. As described in Section 3.2, at each time step a nonlinear system is solved via a matrix-free Newton method. The matrix-vector multiplication function therein has many GPU kernels that can readily take data already existing on GPU memory, hence be able to reduce the cost of transferring data between CPU and GPU memories. The function `cublasDdot` is used for dot product with double precision. There are other choices for reduce operation like `atomicAdd` function for double precision numbers, but it is not natively supported by NVIDIA and is relatively slow compare to cublas. While running with multiple MPI processes, data transfers between CPU and GPU

memories are inevitable. Firstly, data has to be transferred back to the CPU memory for output. Secondly, data needs to be transferred to CPU memory before MPI communication, and transferred back to GPU memory for calculation. But it is expected that the second level of parallelism on GPU will outperform CPU. The most crucial technique to obtain speedup is to hide the time used for MPI communications behind actual computation time on the GPUs. This is done through non-blocking MPI communications. We split GPU kernel in the matrix-vector multiplication function into two separate kernels, one does not require computation on MPI communicated data, the other does. Since non-blocking MPI communication functions such as `MPI_Isend` and `MPI_Irecv` return immediately, we can continue to execute the first kernel while the MPI communication takes place. We put a `MPI_Waitall` only before the launch of the second kernel, making sure the MPI communication is finished before accessing these data.

Finally, it is vitally important to design kernels that can run with different mesh sizes in Table 3.3.1, and also have room for finer mesh. The most crucial design in kernels are the choice of block and grid sizes. As mentioned before, each block and grid can have one, two or three-dimensions. This gives much flexibility in launching kernels with different structure of the data. In the application problem here, the number of threads in one block is determined by the mesh on $x$-direction $N_x$, as `dim3 threads(Nx, 1)`. This allows $N_x$ to be as large as the limit of 1024 threads. Moreover the number of blocks in one grid is determined by the mesh on $y$- and $z$-directions $N_y$ and $l\_N_z$, as `dim3 blocks(Ny, l_Nz)`. $l$ means local to the MPI process. `dim3` can be used to define arrays of up to three-dimensions. In this setup, all utility functions and kernels that need to access large arrays on GPU can be designed to use the same block thread structure, making it much easier to program the actual kernels.

### 3.4 Convergence Studies

The convergence studies in this section is part of [13].

This section presents numerical studies of convergence order of the spatial discretization for the two scalar test problems of Section 2.2. If the true solution is denoted by $u$ and its numerical approximation by $u_h$, then classical results for the spatial error in the $L^2$-norm have the form

$$\|u(\mathbf{x},t) - u_h(\mathbf{x},t)\|_{L^2(\Omega)} \leq C\, h^q, \quad \text{as } h \to 0, \tag{3.4.1}$$

for all $0 < t \leq t_{\text{fin}}$, where the constant $C$ is independent of the mesh size $h$. The number $q$ is the convergence order of the spatial discretization. Here, the $L^2(\Omega)$-norm is defined as

$$\|v\|_{L^2(\Omega)} = \left( \int_\Omega v^2 \, d\mathbf{x} \right)^{1/2}.$$

For the finite element method, the classical theory for linear elements specifies $q = 2$, which does not depend on space dimension, see, e.g., [32]. The classical theory requires the source terms to be in the function space $L^2(\Omega)$, which is not true for source terms involving point sources modeled by the Dirac delta distribution. For the finite element method, heuristic arguments and the computational results of [7] indicate that $q = 0.5$ in three spatial dimensions and $q = 1.0$ in two spatial dimensions. Motivated by these computational results, rigorous analysis in [27, Theorem 5.1] establishes that (3.4.1) holds with $q = 2 - d/2 - \varepsilon$ in dimensions $d = 2, 3$, which confirms the computational results.

For the finite volume method, we are not aware of any rigorous theory for problems involving non-smooth source terms. The purpose of this section is to analyze the convergence order of the finite volume method numerically, analogous to [7], and additionally (i) analyze the impact of increasing advection on the convergence order and (ii) compare the first- and second-order discretization of the advection term in

(3.1.18) and (3.1.19), respectively.

The results in this section are collected in 10 tables, with Tables 3.4.1 through 3.4.4 containing results for problems on the three-dimensional domain $\Omega \subset \mathbb{R}^3$, Tables 3.4.5 through 3.4.8 results for problems on the two-dimensional domain $\Omega \subset \mathbb{R}^2$, and Tables 3.4.9 and 3.4.10 providing summaries of the results. Specifically, we start with Tables 3.4.1 and 3.4.2 considering the same method as in [26], using the second-order discretization of the advection term (3.1.19). Then, Tables 3.4.3 and 3.4.4 provide the comparison to using the first-order discretization of the advection term (3.1.18). The two pairs of Tables 3.4.5 and 3.4.6 and Tables 3.4.7 and 3.4.8 repeat the comparison of second- and first-order advection discretizations for the problems in two dimensions.

Each of the Tables 3.4.1 through 3.4.8 contains five subtables. The entries in each subtable report the $L^2$-norm of the error $\|u(\cdot,t) - u_h(\cdot,t)\|_{L^2(\Omega)}$ and in parentheses a numerical estimate of the convergence order $q$ from (3.4.1), for four progressively finer meshes. Given numerical solutions on two meshes with mesh widths $h$ and $2h$, the order $q$ can be estimated using the formula

$$q^{est} = \log_2 \left( \frac{\|u(\cdot,t) - u_{2h}(\cdot,t)\|_{L^2(\Omega)}}{\|u(\cdot,t) - u_h(\cdot,t)\|_{L^2(\Omega)}} \right). \tag{3.4.2}$$

For problems, whose true solution $u$ is not available, it is customary to use the numerical solution on the finest available mesh as a reference solution in place of $u$ in the norm of the errors. This procedure is necessary for the non-smooth test problem. For consistency, we apply this procedure also for the smooth test problem. Hence, all tables in this section show errors against a reference solution. However, for the smooth problem, we also obtained the analogous results of $\|u(\cdot,t) - u_h(\cdot,t)\|_{L^2(\Omega)}$ and $q^{est}$ using the true solution and confirmed that the procedure with the reference solution worked correctly in this case.

First, let us discuss results in three spatial dimensions. For instance, Table 3.4.1 reports error and convergence order against reference solution using second-order dis-

cretization of the advection term. The first subtable 3.4.1 (a) reports the results for the finite element method. Since this method is not suitable for problems with advection, it is applied to the test problems without the advection term. The convergence order results for the finite element method agree with those in [7] and are included here to allow for a ready comparison. Recall from Section 2.2 and Table 2.1.1 that the scalar test equations use the calcium diffusion coefficient matrix $D = D^{(1)} = \text{diag}(0.15, 0.15, 0.30)$. The advection velocity is the product of the weight $\omega$ and vector $(0.15, 0.15, 0.30)^T$ of the form $\beta = \beta^{(1)} = \omega\,(0.15, 0.15, 0.30)^T$ such that we can control the magnitude of advection by varying the constant $\omega$. For $\omega = 0$, there is no advection, and for $\omega = 1$, diffusion and advection are on the same order of magnitude. Subtables 3.4.1 (b)–(e) contain results for the finite volume method with second-order discretization of advection term described in (3.1.19), with $\omega = 0, 0.01, 0.1, 1$. We observe that both FEM and FVM with second-order advection discretization have a numerical convergence order of $q = 2$. Table 3.4.2 contains five subtables for the test problem with non-smooth source term. Recall from Section 2.2 that the single CRU in the center of the domain opens at time $t = 1$ and remains open afterwards. We observe a numerical convergence order of $q = 0.5$ for both FEM and FVM with second-order advection discretization. Table 3.4.3 and Table 3.4.4 repeat the tests above but use the first-order advection discretization as described in (3.1.18). Table 3.4.3 show that as we increase the weight of advection $\omega$, the numerical convergence order dropped from 2 to 1.5, indicating the convergence rate is lower when using first-order advection discretization for the test problem with smooth source term. From Table 3.4.4 we observe the numerical convergence order of $q = 0.5$ for both FEM and FVM with first-order advection discretization.

Notice that in Table 3.4.2 and Table 3.4.4 the actual observed errors have a magnitude of $10^{+1}$. This is because the calcium release unit at the center of the domain is

modeled by Dirac delta distribution, which is a highly non-smooth source term. Note that the apparently large values for the error must be viewed in the context of the large size of the domain $\Omega$, which is $(12.8)(12.8)(64) = 10,\, 485.76$. In the meantime, convergence studies in [7] have shown that the $L^2$-norms have the magnitude from $10^{-1}$ to $10^{-3}$ and converge quadratically on the domain $\Omega$ with a small area centered about the calcium release unit removed. This confirms that the non-smooth source term is the reason of observed convergence order $q = 0.5$.

Now, let us move on to results in two spatial dimensions. Table 3.4.5 and Table 3.4.6 show that with second-order discretization of the advection term, the numerical convergence order $q = 2$ for finite volume method with smooth source term, and $q = 1$ with non-smooth source term. Table 3.4.7 and Table 3.4.8 show that with first-order discretization of advection term, $q$ drops below 2 while increasing the weight of advection with smooth source term, and $q = 1$ with non-smooth source term. This shows that the convergence order is dependent on the space dimensions while dealing with highly non-smooth source terms such as point sources here.

Table 3.4.1: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with smooth source term in 3-D using a second-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 2.4322e–01 | 2.1725e–01 | 1.9453e–01 |
| $32 \times 32 \times 128$ | 6.0327e–02 (2.0114) | 5.3869e–02 (2.0119) | 4.8182e–02 (2.0134) |
| $64 \times 64 \times 256$ | 1.4488e–02 (2.0579) | 1.2954e–02 (2.0561) | 1.1598e–02 (2.0546) |
| $128 \times 128 \times 512$ | 3.0194e–03 (2.2626) | 2.7163e–03 (2.2536) | 2.4465e–03 (2.2450) |
| (b) 2nd order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.1596e–01 | 4.0106e–01 | 3.8575e–01 |
| $32 \times 32 \times 128$ | 1.0682e–01 (1.9612) | 1.0245e–01 (1.9689) | 9.8285e–02 (1.9726) |
| $64 \times 64 \times 256$ | 2.6685e–02 (2.0011) | 2.5573e–02 (2.0022) | 2.4520e–02 (2.0030) |
| $128 \times 128 \times 512$ | 6.9077e–03 (1.9498) | 6.6055e–03 (1.9529) | 6.3208e–03 (1.9558) |
| (c) 2nd order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.1588e–01 | 4.0096e–01 | 3.8564e–01 |
| $32 \times 32 \times 128$ | 1.0683e–01 (1.9608) | 1.0246e–01 (1.9684) | 9.8294e–02 (1.9721) |
| $64 \times 64 \times 256$ | 2.6687e–02 (2.0011) | 2.5574e–02 (2.0023) | 2.4520e–02 (2.0031) |
| $128 \times 128 \times 512$ | 6.9076e–03 (1.9499) | 6.6050e–03 (1.9530) | 6.3202e–03 (1.9559) |
| (d) 2nd order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.1551e–01 | 4.0076e–01 | 3.8570e–01 |
| $32 \times 32 \times 128$ | 1.0699e–01 (1.9574) | 1.0273e–01 (1.9639) | 9.8654e–02 (1.9670) |
| $64 \times 64 \times 256$ | 2.6726e–02 (2.0011) | 2.5638e–02 (2.0025) | 2.4601e–02 (2.0037) |
| $128 \times 128 \times 512$ | 6.9112e–03 (1.9512) | 6.6163e–03 (1.9542) | 6.3338e–03 (1.9576) |
| (e) 2nd order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.3888e–01 | 4.5043e–01 | 4.6647e–01 |
| $32 \times 32 \times 128$ | 1.1749e–01 (1.9013) | 1.2257e–01 (1.8777) | 1.2900e–01 (1.8544) |
| $64 \times 64 \times 256$ | 2.9760e–02 (1.9811) | 3.1218e–02 (1.9732) | 3.3070e–02 (1.9638) |
| $128 \times 128 \times 512$ | 7.4323e–03 (2.0015) | 7.5946e–03 (2.0393) | 7.8554e–03 (2.0738) |

Table 3.4.2: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with non-smooth source term in 3-D using a second-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.4249e+01 | 5.0676e+01 | 5.0706e+01 |
| $32 \times 32 \times 128$ | 3.6689e+01 (0.5643) | 3.7651e+01 (0.4286) | 3.7806e+01 (0.4235) |
| $64 \times 64 \times 256$ | 2.6378e+01 (0.4760) | 2.6396e+01 (0.5124) | 2.6400e+01 (0.5181) |
| $128 \times 128 \times 512$ | 1.6083e+01 (0.7138) | 1.6083e+01 (0.7147) | 1.6084e+01 (0.7149) |
| (b) 2nd order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 7.6451e+01 | 8.5493e+01 | 8.8933e+01 |
| $32 \times 32 \times 128$ | 6.4231e+01 (0.2513) | 6.5344e+01 (0.3877) | 6.5578e+01 (0.4395) |
| $64 \times 64 \times 256$ | 4.6063e+01 (0.4797) | 4.6135e+01 (0.5022) | 4.6152e+01 (0.5068) |
| $128 \times 128 \times 512$ | 3.0898e+01 (0.5761) | 3.0903e+01 (0.5781) | 3.0905e+01 (0.5786) |
| (c) 2nd order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 7.6263e+01 | 8.5145e+01 | 8.8515e+01 |
| $32 \times 32 \times 128$ | 6.4070e+01 (0.2513) | 6.5172e+01 (0.3857) | 6.5405e+01 (0.4365) |
| $64 \times 64 \times 256$ | 4.5999e+01 (0.4781) | 4.6070e+01 (0.5004) | 4.6087e+01 (0.5050) |
| $128 \times 128 \times 512$ | 3.0875e+01 (0.5751) | 3.0880e+01 (0.5772) | 3.0881e+01 (0.5776) |
| (d) 2nd order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 7.4665e+01 | 8.2230e+01 | 8.5029e+01 |
| $32 \times 32 \times 128$ | 6.2670e+01 (0.2527) | 6.3683e+01 (0.3687) | 6.3902e+01 (0.4121) |
| $64 \times 64 \times 256$ | 4.5425e+01 (0.4643) | 4.5495e+01 (0.4852) | 4.5512e+01 (0.4896) |
| $128 \times 128 \times 512$ | 3.0667e+01 (0.5668) | 3.0672e+01 (0.5688) | 3.0673e+01 (0.5693) |
| (e) 2nd order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 6.5034e+01 | 6.6909e+01 | 6.7530e+01 |
| $32 \times 32 \times 128$ | 5.2889e+01 (0.2982) | 5.3328e+01 (0.3273) | 5.3437e+01 (0.3377) |
| $64 \times 64 \times 256$ | 4.0584e+01 (0.3821) | 4.0636e+01 (0.3921) | 4.0648e+01 (0.3947) |
| $128 \times 128 \times 512$ | 2.8747e+01 (0.4975) | 2.8751e+01 (0.4991) | 2.8752e+01 (0.4995) |

Table 3.4.3: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with smooth source term in 3-D using a first-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 2.4322e–01 | 2.1725e–01 | 1.9453e–01 |
| $32 \times 32 \times 128$ | 6.0327e–02 (2.0114) | 5.3869e–02 (2.0119) | 4.8182e–02 (2.0134) |
| $64 \times 64 \times 256$ | 1.4488e–02 (2.0579) | 1.2954e–02 (2.0561) | 1.1598e–02 (2.0546) |
| $128 \times 128 \times 512$ | 3.0194e–03 (2.2626) | 2.7163e–03 (2.2536) | 2.4465e–03 (2.2450) |
| (b) 1st order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.1596e–01 | 4.0106-e-01 | 3.8575e–01 |
| $32 \times 32 \times 128$ | 1.0682e–01 (1.9612) | 1.0245e–01 (1.9689) | 9.8285e–02 (1.9726) |
| $64 \times 64 \times 256$ | 2.6685e–02 (2.0011) | 2.5573e–02 (2.0022) | 2.4520e–02 (2.0030) |
| $128 \times 128 \times 512$ | 6.9077e–03 (1.9498) | 6.6055e–03 (1.9529) | 6.3208e–03 (1.9558) |
| (c) 1st order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.1399e–01 | 3.9820e–01 | 3.8205e–01 |
| $32 \times 32 \times 128$ | 1.0598e—01 (1.9658) | 1.0122e–01 (1.9761) | 9.6683e–02 (1.9824) |
| $64 \times 64 \times 256$ | 2.6350e–02 (2.0079) | 2.5085e–02 (2.0126) | 2.3890e–02 (2.0168) |
| $128 \times 128 \times 512$ | 6.8263e–03 (1.9486) | 6.4894e–03 (1.9507) | 6.1736e–03 (1.9522) |
| (d) 1st order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 3.9932e–01 | 3.7901e–01 | 3.5979e–01 |
| $32 \times 32 \times 128$ | 1.0109e–01 (1.9819) | 9.5928e–02 (1.9822) | 9.2084e–02 (1.9661) |
| $64 \times 64 \times 256$ | 2.5389e–02 (1.9934) | 2.5096e–02 (1.9345) | 2.5552e–02 (1.8495) |
| $128 \times 128 \times 512$ | 6.9876e–03 (1.8613) | 7.2888e–03 (1.7837) | 7.8148e–03 (1.7091) |
| (e) 1st order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.4935e–01 | 7.1330e–01 | 8.7562e–01 |
| $32 \times 32 \times 128$ | 2.4039e–01 (1.1923) | 3.3733e–01 (1.0804) | 4.2579e–01 (1.0401) |
| $64 \times 64 \times 256$ | 1.0429e–01 (1.2047) | 1.4883e–01 (1.1805) | 1.8866e–01 (1.1743) |
| $128 \times 128 \times 512$ | 3.5657e–02 (1.5484) | 5.0768e–02 (1.5517) | 6.4249e–02 (1.5541) |

Table 3.4.4: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with non-smooth source term in 3-D using a first-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.4249e+01 | 5.0676e+01 | 5.0706e+01 |
| $32 \times 32 \times 128$ | 3.6689e+01 (0.5643) | 3.7651e+01 (0.4286) | 3.7806e+01 (0.4235) |
| $64 \times 64 \times 256$ | 2.6378e+01 (0.4760) | 2.6396e+01 (0.5124) | 2.6400e+01 (0.5181) |
| $128 \times 128 \times 512$ | 1.6083e+01 (0.7138) | 1.6083e+01 (0.7147) | 1.6084e+01 (0.7149) |
| (b) 1st order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 7.6451e+01 | 8.5493e+01 | 8.8933e+01 |
| $32 \times 32 \times 128$ | 6.4231e+01 (0.2513) | 6.5344e+01 (0.3877) | 6.5578e+01 (0.4395) |
| $64 \times 64 \times 256$ | 4.6063e+01 (0.4797) | 4.6135e+01 (0.5022) | 4.6152e+01 (0.5068) |
| $128 \times 128 \times 512$ | 3.0898e+01 (0.5761) | 3.0903e+01 (0.5781) | 3.0905e+01 (0.5786) |
| (c) 1st order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 7.6299e+01 | 8.5218e+01 | 8.8596e+01 |
| $32 \times 32 \times 128$ | 6.4101e+01 (0.2513) | 6.5198e+01 (0.3863) | 6.5426e+01 (0.4374) |
| $64 \times 64 \times 256$ | 4.6010e+01 (0.4784) | 4.6080e+01 (0.5007) | 4.6097e+01 (0.5052) |
| $128 \times 128 \times 512$ | 3.0881e+01 (0.5752) | 3.0886e+01 (0.5772) | 3.0887e+01 (0.5777) |
| (d) 1st order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 7.4977e+01 | 8.2850e+01 | 8.5714e+01 |
| $32 \times 32 \times 128$ | 6.2953e+01 (0.2522) | 6.3918e+01 (0.3743) | 6.4105e+01 (0.4191) |
| $64 \times 64 \times 256$ | 4.5538e+01 (0.4672) | 4.5596e+01 (0.4873) | 4.5609e+01 (0.4911) |
| $128 \times 128 \times 512$ | 3.0725e+01 (0.5676) | 3.0729e+01 (0.5693) | 3.0730e+01 (0.5697) |
| (e) 1st order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 6.5199e+01 | 6.7564e+01 | 6.8367e+01 |
| $32 \times 32 \times 128$ | 5.4115e+01 (0.2688) | 5.4522e+01 (0.3094) | 5.4708e+01 (0.3216) |
| $64 \times 64 \times 256$ | 4.1512e+01 (0.3825) | 4.1587e+01 (0.3907) | 4.1646e+01 (0.3936) |
| $128 \times 128 \times 512$ | 2.9290e+01 (0.5031) | 2.9304e+01 (0.5050) | 2.9316e+01 (0.5065) |

Table 3.4.5: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with smooth source term in 2-D using a second-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.5341e–01 | 1.4291e–01 | 1.3300e–01 |
| $32 \times 32 \times 128$ | 3.7920e–02 (2.0164) | 3.5316e–02 (2.0167) | 3.2857e–02 (2.0172) |
| $64 \times 64 \times 256$ | 8.9647e–03 (2.0806) | 8.3605e–03 (2.0787) | 7.7914e–03 (2.0762) |
| $128 \times 128 \times 512$ | 1.8359e–03 (2.2878) | 1.7270e–03 (2.2753) | 1.6268e–03 (2.2598) |
| (b) 2nd order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.3268e–01 | 1.2941e–01 | 1.2582e–01 |
| $32 \times 32 \times 128$ | 3.4128e–02 (1.9589) | 3.3101e–02 (1.9671) | 3.2096e–02 (1.9709) |
| $64 \times 64 \times 256$ | 8.5443e–03 (1.9979) | 8.2794e–03 (1.9993) | 8.0214e–03 (2.0005) |
| $128 \times 128 \times 512$ | 2.2398e–03 (1.9316) | 2.1654e–03 (1.9349) | 2.0924e–03 (1.9387) |
| (c) 2nd order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.3264e–01 | 1.2937e–01 | 1.2576e–01 |
| $32 \times 32 \times 128$ | 3.4126e–02 (1.9586) | 3.3097e–02 (1.9667) | 3.2088e–02 (1.9706) |
| $64 \times 64 \times 256$ | 8.5434e–03 (1.9980) | 8.2774e–03 (1.9994) | 8.0183e–03 (2.0007) |
| $128 \times 128 \times 512$ | 2.2395e–03 (1.9317) | 2.1649e–03 (1.9349) | 2.0916e–03 (1.9387) |
| (d) 2nd order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.3246e–01 | 1.2917e–01 | 1.2559e–01 |
| $32 \times 32 \times 128$ | 3.4144e–02 (1.9558) | 3.3128e–02 (1.9632) | 3.2125e–02 (1.9670) |
| $64 \times 64 \times 256$ | 8.5449e–03 (1.9985) | 8.2783e–03 (2.0006) | 8.0182e–03 (2.0023) |
| $128 \times 128 \times 512$ | 2.2385e–03 (1.9326) | 2.1635e–03 (1.9360) | 2.0900e–03 (1.9398) |
| (e) 2nd order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.4105e–01 | 1.4700e–01 | 1.5385e–01 |
| $32 \times 32 \times 128$ | 3.7675e–02 (1.9046) | 3.9556e–02 (1.8938) | 4.1453e–02 (1.8920) |
| $64 \times 64 \times 256$ | 9.4614e–03 (1.9935) | 9.8093e–03 (2.0117) | 1.0087e–02 (2.0390) |
| $128 \times 128 \times 512$ | 2.3737e–03 (1.9949) | 2.3841e–03 (2.0407) | 2.4068e–03 (2.0673) |

Table 3.4.6: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with non-smooth source term in 2-D using a second-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 6.4814e+01 | 5.6294e+01 | 5.4132e+01 |
| $32 \times 32 \times 128$ | 3.2145e+01 (1.0117) | 3.1586e+01 (0.8337) | 3.1433e+01 (0.7842) |
| $64 \times 64 \times 256$ | 1.9717e+01 (0.7051) | 1.9524e+01 (0.6941) | 1.9461e+01 (0.6917) |
| $128 \times 128 \times 512$ | 9.4765e+00 (1.0570) | 9.4550e+00 (1.0461) | 9.4477e+00 (1.0425) |
| (b) 2nd order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.2421e+01 | 6.5340e+01 | 7.1232e+01 |
| $32 \times 32 \times 128$ | 3.7185e+01 (0.4954) | 3.9076e+01 (0.7417) | 3.9606e+01 (0.8468) |
| $64 \times 64 \times 256$ | 1.9847e+01 (0.9058) | 2.0017e+01 (0.9650) | 2.0073e+01 (0.9805) |
| $128 \times 128 \times 512$ | 9.6699e+00 (1.0373) | 9.6869e+00 (1.0471) | 9.6926e+00 (1.0503) |
| (c) 2nd order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.2229e+01 | 6.4951e+01 | 7.0739e+01 |
| $32 \times 32 \times 128$ | 3.7048e+01 (0.4955) | 3.8925e+01 (0.7387) | 3.9452e+01 (0.8424) |
| $64 \times 64 \times 256$ | 1.9808e+01 (0.9033) | 1.9978e+01 (0.9623) | 2.0033e+01 (0.9777) |
| $128 \times 128 \times 512$ | 9.6608e+00 (1.0358) | 9.6779e+00 (1.0456) | 9.6835e+00 (1.0488) |
| (d) 2nd order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.0582e+01 | 6.1664e+01 | 6.6600e+01 |
| $32 \times 32 \times 128$ | 3.5859e+01 (0.4963) | 3.7610e+01 (0.7133) | 3.8115e+01 (0.8052) |
| $64 \times 64 \times 256$ | 1.9462e+01 (0.8817) | 1.9630e+01 (0.9381) | 1.9685e+01 (0.9533) |
| $128 \times 128 \times 512$ | 9.5802e+00 (1.0225) | 9.5973e+00 (1.0324) | 9.6029e+00 (1.0355) |
| (e) 2nd order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.0331e+01 | 4.3925e+01 | 4.5555e+01 |
| $32 \times 32 \times 128$ | 2.7520e+01 (0.5514) | 2.8498e+01 (0.6242) | 2.8846e+01 (0.6592) |
| $64 \times 64 \times 256$ | 1.6563e+01 (0.7326) | 1.6710e+01 (0.7702) | 1.6760e+01 (0.7834) |
| $128 \times 128 \times 512$ | 8.8417e+00 (0.9055) | 8.8579e+00 (0.9156) | 8.8632e+00 (0.9191) |

Table 3.4.7: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with smooth source term in 2-D using a first-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.5341e–01 | 1.4291e–01 | 1.3300e–01 |
| $32 \times 32 \times 128$ | 3.7920e–02 (2.0164) | 3.5316e–02 (2.0167) | 3.2857e–02 (2.0172) |
| $64 \times 64 \times 256$ | 8.9647e–03 (2.0806) | 8.3605e–03 (2.0787) | 7.7914e–03 (2.0762) |
| $128 \times 128 \times 512$ | 1.8359e–03 (2.2878) | 1.7270e–03 (2.2753) | 1.6268e–03 (2.2598) |
| (b) 1st order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.3268e–01 | 1.2941e–01 | 1.2582e–01 |
| $32 \times 32 \times 128$ | 3.4128e–02 (1.9589) | 3.3101e–02 (1.9671) | 3.2096e–02 (1.9709) |
| $64 \times 64 \times 256$ | 8.5443e–03 (1.9979) | 8.2794e–03 (1.9993) | 8.0214e–03 (2.0005) |
| $128 \times 128 \times 512$ | 2.2398e–03 (1.9316) | 2.1654e–03 (1.9349) | 2.0924e–03 (1.9387) |
| (c) 1st order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.3216e–01 | 1.2864e–01 | 1.2481e–01 |
| $32 \times 32 \times 128$ | 3.3906e–02 (1.9626) | 3.2769e–02 (1.9729) | 3.1656e–02 (1.9792) |
| $64 \times 64 \times 256$ | 8.4566e–03 (2.0034) | 8.1492e–03 (2.0076) | 7.8502e–03 (2.0117) |
| $128 \times 128 \times 512$ | 2.2192e–03 (1.9300) | 2.1353e–03 (1.9322) | 2.0539e–03 (1.9344) |
| (d) 1st order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.2828e–01 | 1.2350e–01 | 1.1877e–01 |
| $32 \times 32 \times 128$ | 3.2633e–02 (1.9749) | 3.1378e–02 (1.9767) | 3.0445e–02 (1.9638) |
| $64 \times 64 \times 256$ | 8.2180e–03 (1.9895) | 8.1770e–03 (1.9401) | 8.3483e–03 (1.8667) |
| $128 \times 128 \times 512$ | 2.2688e–03 (1.8569) | 2.3632e–03 (1.7908) | 2.5289e–03 (1.7229) |
| (e) 1st order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 1.6987e–01 | 2.1957e–01 | 2.7156e–01 |
| $32 \times 32 \times 128$ | 7.1933e–02 (1.2397) | 1.0169e–01 (1.1105) | 1.3014e–01 (1.0612) |
| $64 \times 64 \times 256$ | 3.0802e–02 (1.2236) | 4.4549e–02 (1.1908) | 5.7368e–02 (1.1817) |
| $128 \times 128 \times 512$ | 1.0502e–02 (1.5523) | 1.5167e–02 (1.5544) | 1.9505e–02 (1.5564) |

Table 3.4.8: $L^2$-error against reference solution (and estimated convergence order $q^{est}$) for scalar test problem with non-smooth source term in 2-D using a first-order advection discretization in the finite volume method.

| (a) FEM with $\omega = 0$ | | | |
|---|---|---|---|
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 6.4814e+01 | 5.6294e+01 | 5.4132e+01 |
| $32 \times 32 \times 128$ | 3.2145e+01 (1.0117) | 3.1586e+01 (0.8337) | 3.1433e+01 (0.7842) |
| $64 \times 64 \times 256$ | 1.9717e+01 (0.7051) | 1.9524e+01 (0.6941) | 1.9461e+01 (0.6917) |
| $128 \times 128 \times 512$ | 9.4765e+00 (1.0570) | 9.4550e+00 (1.0461) | 9.4477e+00 (1.0425) |
| (b) 1st order FVM with $\omega = 0$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.2421e+01 | 6.5340e+01 | 7.1232e+01 |
| $32 \times 32 \times 128$ | 3.7185e+01 (0.4954) | 3.9076e+01 (0.7417) | 3.9606e+01 (0.8468) |
| $64 \times 64 \times 256$ | 1.9847e+01 (0.9058) | 2.0017e+01 (0.9650) | 2.0073e+01 (0.9805) |
| $128 \times 128 \times 512$ | 9.6699e+00 (1.0373) | 9.6869e+00 (1.0471) | 9.6926e+00 (1.0503) |
| (c) 1st order FVM with $\omega = 0.01$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.2269e+01 | 6.5038e+01 | 7.0839e+01 |
| $32 \times 32 \times 128$ | 3.7075e+01 (0.4955) | 3.8942e+01 (0.7400) | 3.9460e+01 (0.8441) |
| $64 \times 64 \times 256$ | 1.9811e+01 (0.9041) | 1.9979e+01 (0.9628) | 2.0032e+01 (0.9781) |
| $128 \times 128 \times 512$ | 9.6619e+00 (1.0360) | 9.6785e+00 (1.0456) | 9.6839e+00 (1.0487) |
| (d) 1st order FVM with $\omega = 0.1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 5.0941e+01 | 6.2423e+01 | 6.7459e+01 |
| $32 \times 32 \times 128$ | 3.6108e+01 (0.4965) | 3.7769e+01 (0.7249) | 3.8200e+01 (0.8204) |
| $64 \times 64 \times 256$ | 1.9500e+01 (0.8888) | 1.9641e+01 (0.9433) | 1.9682e+01 (0.9567) |
| $128 \times 128 \times 512$ | 9.5911e+00 (1.0237) | 9.6047e+00 (1.0321) | 9.6088e+00 (1.0344) |
| (e) 1st order FVM with $\omega = 1$ | | | |
| | $t = 2$ | $t = 3$ | $t = 4$ |
| $16 \times 16 \times 64$ | 4.0855e+01 | 4.5125e+01 | 4.7064e+01 |
| $32 \times 32 \times 128$ | 2.8669e+01 (0.5110) | 2.9562e+01 (0.6102) | 3.0137e+01 (0.6431) |
| $64 \times 64 \times 256$ | 1.6990e+01 (0.7548) | 1.7220e+01 (0.7797) | 1.7454e+01 (0.7880) |
| $128 \times 128 \times 512$ | 8.9959e+00 (0.9173) | 9.0551e+00 (0.9273) | 9.1155e+00 (0.9371) |

Tables 3.4.9 and 3.4.10 summarizes the observed convergence orders for smooth and non-smooth source terms, respectively, with both 1st order and 2nd order advection discretization. The convergence orders are averages from the previous tables, as noted in each column of Tables 3.4.9 and 3.4.10, of convergence rate at $t = 2, 3, 4$ on the finest mesh. First, we notice that using finite volume method without advection, the convergence orders are consistent with those using finite element method, in 2-D and 3-D. Second, we notice that for the problem with smooth source term in Table 3.4.9, the convergence order drops when increasing the weight of advection, if using FVM with first-order discretization. This is clearly expected and is overcome by using the second-order accurate advection discretization, which gives the optimal convergence order of 2 in all cases in Table 3.4.9. Third, in Table 3.4.10 for the problem with non-smooth source term, the spatial dimension determines the convergence orders of $q = 1$ for 2-D and $q = 0.5$ for 3-D, independent of discretization order and strength of advection. Therefore, we gain no benefit by using second-order advection discretization in the presence of non-smooth sources terms such as point source.

Our recommendations based on observations above: (i) For smooth source terms, use FVM with 2nd order advection discretization, since it remains most accurate despite increasingly dominant advection. (ii) For non-smooth source terms, use FVM with 1st order advection discretization, since 2nd order advection discretization has no advantage and 1st order advection discretization require less MPI communication.

Table 3.4.9: Observed convergence orders for test problem with smooth source term.

| Smooth source | 1st order FVM | | 2nd order FVM | |
|---|---|---|---|---|
| | 2-D Table 3.4.7 | 3-D Table 3.4.5 | 2-D Table 3.4.3 | 3-D Table 3.4.1 |
| FEM with ($\omega = 0$) | 2.27 | 2.25 | 2.27 | 2.25 |
| FVM with ($\omega = 0$) | 1.94 | 1.95 | 1.94 | 1.95 |
| FVM with $\omega = 0.01$ | 1.93 | 1.95 | 1.94 | 1.95 |
| FVM with $\omega = 0.1$ | 1.79 | 1.78 | 1.94 | 1.95 |
| FVM with $\omega = 1$ | 1.55 | 1.55 | 2.03 | 2.04 |

Table 3.4.10: Observed convergence orders for test problem with non-smooth source term.

| Non-smooth source | 1st order FVM | | 2nd order FVM | |
|---|---|---|---|---|
| | 2-D Table 3.4.8 | 3-D Table 3.4.6 | 2-D Table 3.4.4 | 3-D Table 3.4.2 |
| FEM with ($\omega = 0$) | 1.05 | 0.71 | 1.05 | 0.71 |
| FVM with ($\omega = 0$) | 1.04 | 0.58 | 1.04 | 0.58 |
| FVM with $\omega = 0.01$ | 1.04 | 0.58 | 1.04 | 0.58 |
| FVM with $\omega = 0.1$ | 1.03 | 0.57 | 1.03 | 0.57 |
| FVM with $\omega = 1$ | 0.93 | 0.50 | 0.91 | 0.50 |

# CHAPTER 4

# PARALLEL PERFORMANCE RESULTS

This chapter discusses the parallel performance for the solution of the CICR problem using parallel implementations introduced in Section 3.3.1 and Section 3.3.2. We first demonstrate strong and weak scalability of the parallel implementation using MPI in Section 4.1 and Section 4.2, respectively. Then we show the performance of the parallel implementation using the hybrid CPU/GPU cluster. While Section 4.3 discusses single-node GPU performance, Section 4.4 elaborates multi-node GPU performance.

## 4.1 Strong Scalability

This section describes the parallel performance studies for the solution of the CICR problem on the 2013 portion of maya.

### 4.1.1 Strong Scalability with FEM

Table 4.1.1 collects the results of the performance study by number of nodes and processes per node on maya 2013. The CICR problem described in Section 2.1 is solved with the finite element method described in Section 3.1.1, with BiCGSTAB as linear solver. The table summarizes the observed wall clock time (total time to execute the code) in HH:MM:SS (hours:minutes:seconds) format. In situations where wall clock times are not obtained, ET indicates "excessive time required" (more than 5 days), N/A indicates that the case is not feasible due to $p > (N_z + 1)$, where $N_z + 1$ is the number of finite volume cells on the $z$-direction for spatial mesh resolution of $N_x \times N_y \times N_z$.

We first discuss Table 4.1.1 (c) with mesh resolution of $64 \times 64 \times 256$ in detail as

example, since this subtable has all data possible. Reading along the first column of this subtable, we observe that by doubling the number of processes from 1 to 2 we approximately halve the runtime from each column to the next. We observe the same improvement from 2 to 4 processes. We also observe that by doubling the number of processes from 4 to 8 and from 8 to 16 there are still significant improvement in runtime, although not the halving we observed previously.

Table 4.1.1 (d) reports the observed wall clock time in HH:MM:SS for the highest mesh resolution $128 \times 128 \times 512$ which results in a system of over 25 million equations to be solved at every time step. Wall clock times are given for all possible combination of numbers of nodes and processes per node (that are powers of 2), that is, for 1, 2, 4, 8, 16, 32, and 64 nodes and 1, 2, 4, 8, and 16 processes per node. We observe good scalability while increasing the number of nodes or increasing the number of processes per node.

Table 4.1.2 collects the results of the performance study by number of processes. Each row lists the results for one problem size. Each column corresponds to the number of parallel processes $p$ used in the run. Mesh 1 represents $16 \times 16 \times 64$, Mesh 2 represents $32 \times 32 \times 128$, Mesh 3 represents $64 \times 64 \times 256$, Mesh 4 represents $128 \times 128 \times 512$. ET indicates "excessive time required" (more than 5 days), N/A indicates that the case is not feasible due to $p > (N_z+1)$. For the $128 \times 128 \times 512$ mesh, we use the modified definition $S_p = 4T_4(N)/T_p(N)$. Data is based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$, where not all of the 16 cores of one node are utilized. This table is intended to demonstrate strong scalability on maya 2013, which is also one key motivation for parallel computing: The run times for a problem of a given, fixed size can potentially be dramatically reduced by spreading the work across a group of parallel processes. More precisely, the ideal behavior of code for a fixed problem size using $p$ parallel processes is that it be $p$ times as fast. If $T_p(N)$

denotes the wall clock time for a problem of a fixed size parameterized by $N$ using $p$ processes, then the quantity $S_p = T_1(N)/T_p(N)$ measures the speedup of the code from 1 to $p$ processes, whose optimal value is $S_p = p$; for the finest resolution, where data are only available starting with $p = 4$, this definition is extended by the formula $S_p = 4T_4(N)/T_p(N)$. The efficiency $E_p = S_p/p$ characterizes in relative terms how close a run with $p$ parallel processes is to this optimal value, for which $E_p = 1$.

Table 4.1.2 (b) shows the speedup observed. The speedup $S_p$ is increasing significantly as we increase the number of processes. However, the ratio over the optimal value of speedup $p$ seems to decrease as we increase the number of processes. We also observe that the speedup is better for larger problems. Table 4.1.2 (c) shows the observed efficiency $E_p$. The primary decrease of efficiency is between $p = 8$ and $p = 16$, similar to studies in [17] but not as severe. This suggests the bottleneck of CPU memory channels we observed in [17] may still be affecting the scalability on the CICR problem. The fundamental reason for the speedup and efficiency to trail off is that simply too little work is performed on each process. Due to the one-dimensional split in the $z$-direction into as many sub-domains as parallel processes $p$, eventually only one or two $x$-$y$-planes of data are located on each process. This is not enough calculation work to justify the cost of communicating between the processes. In effect, this leads to a recommendation how many nodes to use for a particular $N_x \times N_y \times N_z$ mesh, with more nodes being justifiable for larger meshes.

The customary graphical representations of speedup and efficiency are presented in Figures 4.1.1 (a) and (b), respectively. Figure 4.1.1 (a) shows the speedup pattern as we observed in Table 4.1.2 (b) but more intuitively. The efficiency plotted in Figure 4.1.1 (b) is directly derived from the speedup, but the plot is still useful because it details interesting features for small values of $p$ that are hard to discern in the speedup plot. Here, we notice the consistency of most results for small $p$.

Table 4.1.1: Performance study of the CICR problem solved with finite element method on maya 2013 by number of nodes and processes per node. ET indicates "excessive time required" (more than 5 days), N/A indicates that the case is not feasible due to $p > (N_z + 1)$.

| (a) Mesh resolution $N_x \times N_y \times N_z = 16 \times 16 \times 64$, DOF = 56,355 | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | 00:03:56 | 00:02:09 | 00:01:05 | 00:00:42 | 00:00:25 | 00:00:18 | 00:00:25 |
| 2 processes per node | 00:02:10 | 00:01:06 | 00:00:43 | 00:00:23 | 00:00:17 | 00:00:16 | N/A |
| 4 processes per node | 00:01:10 | 00:00:41 | 00:00:27 | 00:00:19 | 00:00:16 | N/A | N/A |
| 8 processes per node | 00:00:47 | 00:00:26 | 00:00:19 | 00:00:15 | N/A | N/A | N/A |
| 16 processes per node | 00:00:23 | 00:00:18 | 00:00:14 | N/A | N/A | N/A | N/A |
| (b) Mesh resolution $N_x \times N_y \times N_z = 32 \times 32 \times 128$, DOF = 421,443 | | | | | | | |
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | 04:12:42 | 02:05:35 | 01:03:55 | 00:33:26 | 00:18:04 | 00:10:36 | 00:07:01 |
| 2 processes per node | 02:11:30 | 01:06:22 | 00:34:21 | 00:18:24 | 00:10:44 | 00:06:38 | 00:05:03 |
| 4 processes per node | 01:12:39 | 00:36:50 | 00:19:32 | 00:11:09 | 00:06:50 | 00:04:49 | N/A |
| 8 processes per node | 00:40:49 | 00:20:40 | 00:11:16 | 00:06:49 | 00:04:46 | N/A | N/A |
| 16 processes per node | 00:20:28 | 00:10:58 | 00:06:40 | 00:04:39 | N/A | N/A | N/A |
| (c) Mesh resolution $N_x \times N_y \times N_z = 64 \times 64 \times 256$, DOF = 3,257,475 | | | | | | | |
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | 29:39:29 | 14:47:48 | 07:33:10 | 03:44:59 | 01:55:11 | 01:00:27 | 00:34:46 |
| 2 processes per node | 15:33:52 | 07:51:21 | 03:56:31 | 01:58:21 | 01:02:16 | 00:34:14 | 00:21:01 |
| 4 processes per node | 08:48:36 | 04:24:14 | 02:09:58 | 01:07:38 | 00:36:20 | 00:21:30 | 00:14:45 |
| 8 processes per node | 05:17:38 | 02:35:59 | 01:15:55 | 00:39:56 | 00:22:46 | 00:14:32 | N/A |
| 16 processes per node | 02:36:56 | 01:15:20 | 00:40:07 | 00:22:48 | 00:14:49 | N/A | N/A |
| (d) Mesh resolution $N_x \times N_y \times N_z = 128 \times 128 \times 512$, DOF = 25,610,499 | | | | | | | |
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | ET | ET | ET | 60:03:40 | 30:23:30 | 15:21:16 | 08:04:43 |
| 2 processes per node | ET | ET | 62:58:07 | 31:45:39 | 16:18:36 | 08:26:48 | 04:37:13 |
| 4 processes per node | ET | 70:17:37 | 35:51:17 | 18:17:27 | 09:22:47 | 04:59:08 | 02:54:31 |
| 8 processes per node | 82:31:01 | 41:48:36 | 21:29:59 | 11:10:15 | 05:47:37 | 03:11:25 | 01:59:34 |
| 16 processes per node | 42:07:19 | 21:29:09 | 11:14:39 | 05:51:53 | 03:14:21 | 02:05:49 | N/A |

Table 4.1.2: Performance study of the CICR problem solved with finite element method on maya 2013 by number of processes. Data based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$.

(a) Wall clock time $T_P$ in HH:MM:SS

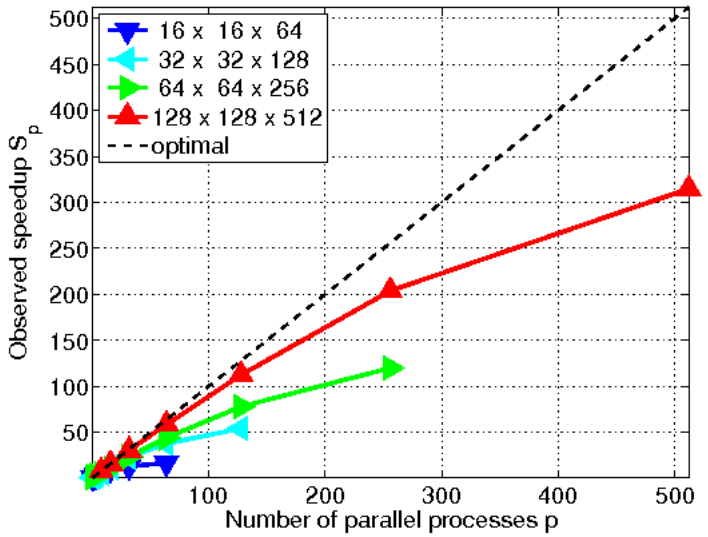
| Mesh | $p=1$ | $p=2$ | $p=4$ | $p=8$ | $p=16$ | $p=32$ | $p=64$ | $p=128$ | $p=256$ | $p=512$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 00:03:56 | 00:02:10 | 00:01:10 | 00:00:47 | 00:00:23 | 00:00:18 | 00:00:14 | N/A | N/A | N/A |
| 2 | 04:12:42 | 02:11:30 | 01:12:39 | 00:40:49 | 00:20:28 | 00:10:58 | 00:06:40 | 00:04:39 | N/A | N/A |
| 3 | 29:39:29 | 15:33:52 | 08:48:36 | 05:17:38 | 02:36:56 | 01:15:20 | 00:40:07 | 00:22:48 | 00:14:49 | N/A |
| 4 | ET | ET | ET | 82:31:00 | 42:07:19 | 21:29:09 | 11:14:39 | 05:51:53 | 03:14:21 | 02:05:49 |

(b) Observed speedup $S_P$

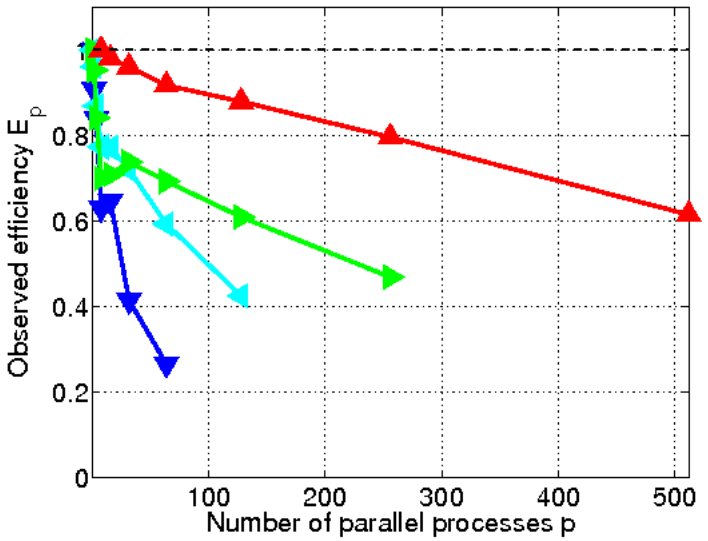
| Mesh | $p=1$ | $p=2$ | $p=4$ | $p=8$ | $p=16$ | $p=32$ | $p=64$ | $p=128$ | $p=256$ | $p=512$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.00 | 1.82 | 3.36 | 5.04 | 10.34 | 13.27 | 16.99 | N/A | N/A | N/A |
| 2 | 1.00 | 1.92 | 3.48 | 6.19 | 12.34 | 23.05 | 37.93 | 54.39 | N/A | N/A |
| 3 | 1.00 | 1.91 | 3.37 | 5.60 | 11.34 | 23.62 | 44.35 | 78.03 | 120.07 | N/A |
| 4 | ET | ET | ET | 8.00 | 15.67 | 30.72 | 58.71 | 112.56 | 203.81 | 314.82 |

(c) Observed efficiency $E_p$

| Mesh | $p=1$ | $p=2$ | $p=4$ | $p=8$ | $p=16$ | $p=32$ | $p=64$ | $p=128$ | $p=256$ | $p=512$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.00 | 0.91 | 0.84 | 0.63 | 0.65 | 0.41 | 0.27 | N/A | N/A | N/A |
| 2 | 1.00 | 0.96 | 0.87 | 0.77 | 0.77 | 0.72 | 0.59 | 0.42 | N/A | N/A |
| 3 | 1.00 | 0.95 | 0.84 | 0.70 | 0.71 | 0.74 | 0.69 | 0.61 | 0.47 | N/A |
| 4 | ET | ET | ET | 1.00 | 0.98 | 0.96 | 0.92 | 0.88 | 0.80 | 0.61 |




(a) Observed speedup $S_p$




(b) Observed efficiency $E_p$

Figure 4.1.1: Performance study of the CICR problem solved with finite element method on maya 2013 by number of processes. Data based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$.

### 4.1.2 Strong Scalability with FVM

This section is part of performance studies in [12] and [10].

Table 4.1.3 collects the results of the performance study by number of nodes and processes per node on maya 2013. The table summarizes the observed wall clock time (total time to execute the code) in HH:MM:SS (hours:minutes:seconds) format. In situations where wall clock times are not obtained, ET indicates "excessive time required" (more than 5 days), N/A indicates that the case is not feasible due to $p > (N_z + 1)$, where $N_z + 1$ is the number of finite volume cells on the $z$-direction for spatial mesh resolution of $N_x \times N_y \times N_z$.

We first discuss Table 4.1.3 (c) with mesh resolution of $64 \times 64 \times 256$ in detail as example, since this subtable has all data possible. Reading along the first column of this subtable, we observe that by doubling the number of processes from 1 to 2 we approximately halve the runtime from each column to the next. We observe the same improvement from 2 to 4 processes. We also observe that by doubling the number of processes from 4 to 8 and from 8 to 16 there are still significant improvement in runtime, although not the halving we observed previously. This is better than in the study in [17], where only small improvements in runtime are observed by doubling the number of processes from 8 to 16. This indicates our application problem is not heavily memory bound as the test problem in [17].

Table 4.1.3 (d) reports the observed wall clock time in HH:MM:SS for the highest mesh resolution $128 \times 128 \times 512$ which results in a system of over 25 million equations to be solved at every time step. Wall clock times are given for all possible combinations of numbers of nodes and processes per node (that are powers of 2), that is, for 1, 2, 4, 8, 16, 32, and 64 nodes and 1, 2, 4, 8, and 16 processes per node. We observe good scalability while increasing the number of nodes or increasing the number of processes per node. Moreover, we observe that the serial run takes more than 5 days, while

the run using either 32 or 64 nodes on maya 2013 can take less than 2 hours. These results demonstrate the power of parallel computing, since jobs require excessive time in serial can be achieved within hours using parallel computing.

Table 4.1.4 collects the results of the performance study by number of processes. Each row lists the results for one problem size. Each column corresponds to the number of parallel processes $p$ used in the run. Mesh 1 represents $16 \times 16 \times 64$, Mesh 2 represents $32 \times 32 \times 128$, Mesh 3 represents $64 \times 64 \times 256$, Mesh 4 represents $128 \times 128 \times 512$. ET indicates "excessive time required" (more than 5 days), N/A indicates that the case is not feasible due to $p > (N_z + 1)$. For the $128 \times 128 \times 512$ mesh, we use the modified definition $S_p = 4T_4(N)/T_p(N)$.

Data is based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$, where not all of the 16 cores of one node are utilized. This table is intended to demonstrate strong scalability on maya 2013, which is also one key motivation for parallel computing: The run times for a problem of a given, fixed size can be potentially dramatically reduced by spreading the work across a group of parallel processes. More precisely, the ideal behavior of code for a fixed problem size using $p$ parallel processes is that it be $p$ times as fast. If $T_p(N)$ denotes the wall clock time for a problem of a fixed size parameterized by $N$ using $p$ processes, then the quantity $S_p = T_1(N)/T_p(N)$ measures the speedup of the code from 1 to $p$ processes, whose optimal value is $S_p = p$; for the finest resolution, where data are only available starting with $p = 4$, this definition is extended by the formula $S_p = 4T_4(N)/T_p(N)$. The efficiency $E_p = S_p/p$ characterizes in relative terms how close a run with $p$ parallel processes is to this optimal value, for which $E_p = 1$.

Table 4.1.4 (b) shows the speedup observed. The speedup $S_p$ is increasing significantly as we increase the number of processes. However, the ratio over the optimal value of speedup $p$ seems to decrease as we increase the number of processes. We

also observe that the speedup is better for larger problems. Table 4.1.4 (c) shows the observed efficiency $E_p$. The primary decrease of efficiency is between $p = 8$ and $p = 16$, similar to studies in [17] but not as severe. This suggests the bottleneck of CPU memory channels we observed in [17] may still be affecting the scalability on the CICR problem. The fundamental reason for the speedup and efficiency to trail off is that simply too little work is performed on each process. Due to the one-dimensional split in the $z$-direction into as many sub-domains as parallel processes $p$, eventually only one or two $x$-$y$-planes of data are located on each process. There are not enough calculation work to justify the cost of communicating between the processes. In effect, this leads to a recommendation how many nodes to use for a particular $N_x \times N_y \times N_z$ mesh, with more nodes being justifiable for larger meshes.

The customary graphical representations of speedup and efficiency are presented in Figures 4.1.2 (a) and (b), respectively. Figure 4.1.2 (a) shows the speedup pattern as we observed in Table 4.1.4 (b) but more intuitively. The efficiency plotted in Figure 4.1.2 (b) is directly derived from the speedup, but the plot is still useful because it details interesting features for small values of $p$ that are hard to discern in the speedup plot. Here, we notice the consistency of most results for small $p$.

Notice that the performance of our implementation of the finite volume method is much faster than the implementation using finite element method in Section 4.1.1. This is partially due to the fact that our implementation of the finite volume method is less computationally intensive. To calculate the mean value of the solution on a control volume, we need fluxes to and from neighboring control volumes, which is a total of 7 control volumes. However to calculate the solution at one mesh point with FEM, we need to consider all neighboring 27 points including itself. Thus the FVM has less computation in matrix vector multiplication. This is to say, our implementation using FVM is faster. In general, we do not conclude that FVM is faster than FEM. We also

notice that the implementation of the finite volume method does not scale as efficiently as the finite element method. This is also due to the fact that our implementation of the finite volume method is less computationally intensive. When there are too many MPI processes and each MPI process do not have enough calculation work to justify the cost of communicating between the processes, the scalability tends to be bad. This applies to both implementations, but much severe in the case of FVM. Due to this limitation of MPI implementation, we cannot further speedup the calculation by adding more nodes. However, if we can greatly speedup the calculation on each MPI process using GPU offloading, then we can speedup the calculation significantly with much fewer number of nodes.

Table 4.1.3: Performance study of the CICR problem solved with first order FVM on maya 2013 by number of nodes and processes per node. ET indicates "excessive time required" (more than 5 days), N/A indicates that the case is not feasible due to $p > (N_z + 1)$.

| (a) Mesh resolution $N_x \times N_y \times N_z = 16 \times 16 \times 64$, DOF = 56,355 | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | 00:12:38 | 00:05:55 | 00:03:23 | 00:02:04 | 00:01:31 | 00:01:20 | 00:01:21 |
| 2 processes per node | 00:07:23 | 00:03:13 | 00:02:00 | 00:01:27 | 00:01:25 | 00:01:19 | N/A |
| 4 processes per node | 00:04:41 | 00:02:03 | 00:01:34 | 00:01:26 | 00:01:37 | N/A | N/A |
| 8 processes per node | 00:03:19 | 00:01:32 | 00:01:32 | 00:01:46 | N/A | N/A | N/A |
| 16 processes per node | 00:02:11 | 00:01:34 | 00:02:43 | N/A | N/A | N/A | N/A |
| (b) Mesh resolution $N_x \times N_y \times N_z = 32 \times 32 \times 128$, DOF = 421,443 | | | | | | | |
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | 02:22:19 | 01:10:46 | 00:36:09 | 00:19:17 | 00:10:54 | 00:06:37 | 00:04:56 |
| 2 processes per node | 01:11:43 | 00:35:48 | 00:19:13 | 00:10:45 | 00:06:47 | 00:04:48 | 00:04:50 |
| 4 processes per node | 00:37:52 | 00:19:27 | 00:11:07 | 00:06:59 | 00:05:18 | 00:04:41 | N/A |
| 8 processes per node | 00:21:35 | 00:11:28 | 00:07:24 | 00:05:41 | 00:05:36 | N/A | N/A |
| 16 processes per node | 00:12:58 | 00:07:24 | 00:06:30 | 00:07:26 | N/A | N/A | N/A |
| (c) Mesh resolution $N_x \times N_y \times N_z = 64 \times 64 \times 256$, DOF = 3,257,47 5 | | | | | | | |
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | 25:02:01 | 12:25:01 | 06:10:14 | 03:07:33 | 01:37:56 | 00:53:17 | 00:31:46 |
| 2 processes per node | 12:25:07 | 06:11:38 | 03:08:20 | 01:37:41 | 00:52:27 | 00:30:36 | 00:20:05 |
| 4 processes per node | 06:32:39 | 03:16:02 | 01:41:55 | 00:55:03 | 00:31:50 | 00:21:03 | 00:16:10 |
| 8 processes per node | 03:52:48 | 01:53:44 | 01:00:20 | 00:34:24 | 00:22:30 | 00:18:33 | N/A |
| 16 processes per node | 02:25:55 | 01:10:26 | 00:39:04 | 00:25:46 | 00:21:29 | N/A | N/A |
| (d) Mesh resolution $N_x \times N_y \times N_z = 128 \times 128 \times 512$, DOF = 25,610 ,499 | | | | | | | |
| | 1 node | 2 nodes | 4 nodes | 8 nodes | 16 nodes | 32 nodes | 64 nodes |
| 1 process per node | ET | ET | 69:15:37 | 34:51:02 | 17:31:44 | 08:59:06 | 04:49:17 |
| 2 processes per node | ET | 69:46:29 | 35:16:03 | 17:45:06 | 09:00:47 | 04:47:14 | 02:43:04 |
| 4 processes per node | 72:31:51 | 36:34:34 | 18:36:29 | 09:32:04 | 05:01:44 | 02:50:34 | 01:46:47 |
| 8 processes per node | 42:01:27 | 26:23:03 | 11:03:41 | 05:46:44 | 03:09:23 | 01:56:47 | 01:23:57 |
| 16 processes per node | 26:53:37 | 13:56:38 | 07:21:17 | 03:54:47 | 02:17:48 | 01:40:35 | N/A |

Table 4.1.4: Performance study of the CICR problem solved with first order FVM on maya 2013 by number of processes. Data based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$.

(a) Wall clock time $T_P$ in HH:MM:SS

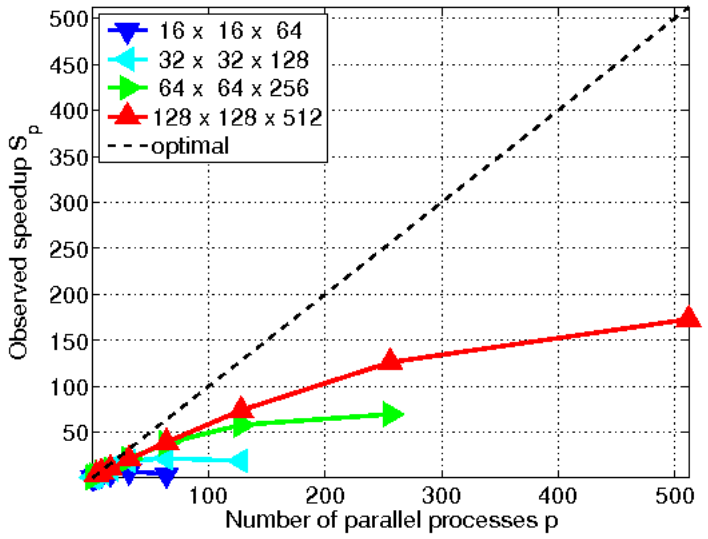
| Mesh | $p=1$ | $p=2$ | $p=4$ | $p=8$ | $p=16$ | $p=32$ | $p=64$ | $p=128$ | $p=256$ | $p=512$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 00:12:38 | 00:07:23 | 00:04:41 | 00:03:19 | 00:02:11 | 00:01:34 | 00:02:43 | N/A | N/A | N/A |
| 2 | 02:22:19 | 01:11:43 | 00:37:52 | 00:21:35 | 00:12:58 | 00:07:24 | 00:06:30 | 00:07:26 | N/A | N/A |
| 3 | 25:02:01 | 12:25:07 | 06:32:39 | 03:52:48 | 02:25:55 | 01:10:26 | 00:39:04 | 00:25:46 | 00:21:29 | N/A |
| 4 | ET | ET | 72:31:51 | 42:01:27 | 26:53:37 | 13:56:38 | 07:21:17 | 03:54:47 | 02:17:48 | 01:40:35 |

(b) Observed speedup $S_P$

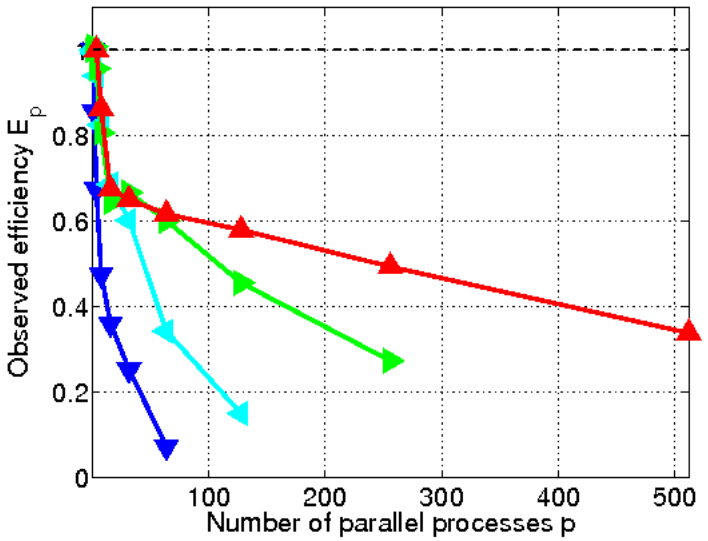
| Mesh | $p=1$ | $p=2$ | $p=4$ | $p=8$ | $p=16$ | $p=32$ | $p=64$ | $p=128$ | $p=256$ | $p=512$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.00 | 1.71 | 2.70 | 3.81 | 5.77 | 8.09 | 4.65 | N/A | N/A | N/A |
| 2 | 1.00 | 1.98 | 3.76 | 6.60 | 10.97 | 19.25 | 21.92 | 19.16 | N/A | N/A |
| 3 | 1.00 | 2.02 | 3.83 | 6.45 | 10.29 | 21.33 | 38.45 | 58.28 | 69.91 | N/A |
| 4 | ET | ET | 4.00 | 6.90 | 10.79 | 20.81 | 39.45 | 74.14 | 126.32 | 173.06 |

(c) Observed efficiency $E_p$

| Mesh | $p=1$ | $p=2$ | $p=4$ | $p=8$ | $p=16$ | $p=32$ | $p=64$ | $p=128$ | $p=256$ | $p=512$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.00 | 0.86 | 0.67 | 0.48 | 0.36 | 0.25 | 0.07 | N/A | N/A | N/A |
| 2 | 1.00 | 0.99 | 0.94 | 0.82 | 0.69 | 0.60 | 0.34 | 0.15 | N/A | N/A |
| 3 | 1.00 | 1.01 | 0.96 | 0.81 | 0.64 | 0.67 | 0.60 | 0.46 | 0.27 | N/A |
| 4 | ET | ET | 1.00 | 0.86 | 0.67 | 0.65 | 0.62 | 0.58 | 0.49 | 0.34 |




(a) Observed speedup $S_p$




(b) Observed efficiency $E_p$

Figure 4.1.2: Performance study of the CICR problem solved with first order FVM on maya 2013 by number of processes. Data based on 16 processes per node, except for the cases $p = 1, 2, 4, 8$.

## 4.2 Weak Scalability

This section is part of [13] that discusses the weak scalability study on the CICR problem on the cluster tara, which is now maya 2009 portion of the cluster maya.

Parallel computing using MPI offers one key advantage: For efficient implementations of appropriate algorithms, problems can be solved significantly faster by pooling the processing power of several compute nodes. We have observed in [26] that the code demonstrates good strong scalability. That is, for a fixed problem size, as we double the number of processes, the wall clock time required for the simulation is nearly halved. However we also observe that the speedup and efficiency dropped significantly when we use a large number of nodes. One reason for this is that increasing the number of processes increases the cost of MPI communication among them. Another reason is that by splitting the fixed problem across more processes, each process has less computations to perform. The ratio of wall clock time on computation over wall clock time on MPI communication is smaller. Hence we lose the strong scalability as we reach certain large number of nodes. Moreover, the strong scalability study failed to demonstrate how efficient the implementation is as we increase problem size. Another key advantage of parallel computing is, problems with larger scale can be solved within comparable time by pooling the processing power and memory of more compute nodes. A weak scalability study is designed to show this, the basic idea is to fix the workload per node while doubling the number of nodes. Since in Section 3.4 we have demonstrated first-order finite volume method is preferable for CICR problem, we will use that for the remaining studies of this chapter.

### 4.2.1 Weak Scalability Study Design

In order to maintain the workload for each node, while doubling the number of nodes, we first double the domain size along the $z$-direction. Then, for each resolution, we preserve the mesh resolution in $x$- and $y$-directions, and double the mesh resolution in $z$-direction.

We test the scalar test problem with smooth source term in Section 2.2 first, since it is much easier to control workload per time step for this basic problem. Then we move on to the three-species application such as the CICR problem where the actual physiology is more complicated. In addition to doubling the domain and mesh on the $z$-direction, we also double the distance between the CRUs. This ensures that the total number of CRUs does not change. This increased space between CRUs is not physiologically realistic, but is necessary to fix the sequence of opening and closing CRUs manually for all runs, allowing each run to have a stable and comparable workload per node.

Table 4.2.1 and Table 4.2.2 outline calculated number of degrees of freedom and estimated total memory usage for the scalar test problem and the CICR problem, respectively. As shown in Tables 4.2.1 (a) and 4.2.2 (a), the number of degrees of freedom depends on the mesh resolution, ranging from tens of thousands all the way to 819 million for the CICR problem and 2 billion for the linear test problem. It also depends on the number of nodes used, since we attempt to fix the workload per node. Because each node has 8 parallel processes, we define $N_z = (4N_x)(p/8)$. When $p = 8$, only one node is used, and $N_z = 4N_x$ is the mesh setting we normally use to solve the CICR problem in Section 2.1. When the number of nodes doubles, the mesh resolution in $z$-direction doubles as well. For mesh resolution $N_x \times N_y \times N_z$, the number of control volumes $M = M_x M_y M_z = (N_x + 1)(N_y + 1)(N_z + 1)$. Hence the number of degrees of freedom $DOF = n_s M = n_s(N_x + 1)(N_y + 1)(N_z + 1)$. Our implementation requires

using 17 large arrays of length $n_s M$, therefore the memory estimation equation is given by: Total memory $= 17 n_s M 8/(1024^3)$ GB. Here, 8 represents 8 bytes of memory for one double precision number. Tables 4.2.1 (b) and 4.2.2 (b) show the estimated memory usage in total based on the degrees of freedom. It demonstrates that the workload per node is expected to grow with the number of nodes used, and we will compare them to the observed memory usage in Table 4.2.3 and Table 4.2.4. The reason the finest mesh in Table 4.2.2 (b) is not as fine as in Table 4.2.1 (b) is because it would exceed the 24 GB memory per node.

Table 4.2.1: Calculated degrees of freedom and estimated total memory for weak scalability study of the scalar test problem with smooth source term. The mesh size in $z$-direction $N_z = (4N_x)(p/8)$ doubles as the number of processes doubles.

| (a) Total number of degrees of freedom for the scalar test problem | | | | | | |
|---|---|---|---|---|---|---|
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 18,785 | 37,570 | 75,140 | 150,280 | 300,560 | 601,120 |
| $32 \times 32$ | 140,481 | 280,962 | 561,924 | 1,123,848 | 2,247,696 | 4,495,392 |
| $64 \times 64$ | 1,085,825 | 2,171,650 | 4,343,300 | 8,686,600 | 17,373,200 | 34,746,400 |
| $128 \times 128$ | 8,536,833 | 17,073,666 | 34,147,332 | 68294664 | 136,589,328 | 273,178,656 |
| $256 \times 256$ | 67,700,225 | 135,400,450 | 270,800,900 | 541,601,800 | 1,083,203,600 | 2,166,407,200 |
| (b) Estimated memory usage based on the degrees of freedom (GB) | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.00 | 0.00 | 0.01 | 0.02 | 0.04 | 0.08 |
| $32 \times 32$ | 0.02 | 0.04 | 0.07 | 0.14 | 0.28 | 0.57 |
| $64 \times 64$ | 0.14 | 0.28 | 0.55 | 1.10 | 2.20 | 4.40 |
| $128 \times 128$ | 1.08 | 2.16 | 4.33 | 8.65 | 17.30 | 34.60 |
| $256 \times 256$ | 8.57 | 17.15 | 34.30 | 68.60 | 137.20 | 274.40 |

Table 4.2.2: Calculated degrees of freedom and estimated total memory for weak scalability study of the CICR problem. The mesh size in $z$-direction $N_z = (4N_x)(p/8)$ doubles as the number of processes doubles.

| (a) Total number of degrees of freedom for the calcium problem | | | | | | |
|---|---|---|---|---|---|---|
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 56,355 | 112,710 | 225,420 | 45,0840 | 901,680 | 1,803,360 |
| $32 \times 32$ | 421,443 | 842,886 | 1,685,772 | 3,371,544 | 6,743,088 | 13,486,176 |
| $64 \times 64$ | 3,257,475 | 6,514,950 | 13,029,900 | 26,059,800 | 52,119,600 | 104,239,200 |
| $128 \times 128$ | 25,610,499 | 51,220,998 | 102,441,996 | 204,883,992 | 409,767,984 | 819,535,968 |
| (b) Estimated memory usage based on the degrees of freedom (GB) | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.01 | 0.01 | 0.03 | 0.06 | 0.11 | 0.23 |
| $32 \times 32$ | 0.05 | 0.11 | 0.21 | 0.43 | 0.85 | 1.71 |
| $64 \times 64$ | 0.41 | 0.83 | 1.65 | 3.30 | 6.60 | 13.20 |
| $128 \times 128$ | 3.24 | 6.49 | 12.98 | 25.95 | 51.90 | 103.80 |

### 4.2.2 Weak Scalability Study Results

We present two weak scalability studies to estimate the parallel performance of the matrix-free method. Table 4.2.3 is based on the scalar test problem with smooth source term from Section 2.2 to final time $t_{\text{fin}} = 1{,}000$. Table 4.2.3 (a) reports the observed wall clock time in HH:MM:SS for each simulation. We observe that the wall clock times only increase slightly as the problem sizes increase in each row. This demonstrates weak scalability of our implementation. Table 4.2.3 (b) shows the number of time steps for the linear problem does not vary too much, even though we increase mesh resolution as well as the number of nodes. Table 4.2.3 (c) shows calculated wall clock time per time step based on data from the previous two subtables. In this case we see the same slight increase in each row as Table 4.2.3 (a). But this subtable is designed to give more insight when wall clock time and time steps differ. We also include observed memory usage per node as a relative indicator for the workload per node in Table 4.2.3 (d). In the mean time, Table 4.2.3 (e) reports the observed total memory usage. The numbers are very close to estimated numbers in Table 4.2.1 (b). These subtables demonstate we have indeed increased the workload both in terms of calculation and memory consumption. And if we increase the computing resources in the same pace, then we can expect the same wall clock time.

Table 4.2.4 shows a weak performance study for the CICR problem from Section 2.1 to final time $t_{\text{fin}} = 100$. It also has 5 subtables like Table 4.2.3. We want to bring about the same idea of doubling work load per time step while double the number of nodes used. One additional setup to ensure we get the same physiological outcome is to use an input file that specifies the open CRUs at each time step, forcing every simulation to reproduce the same CRU sequence. The input file is originally generated by a typical simulation as in Section 2.3. From Table 4.2.4 (a), we observe that the wall clock times in each row decrease slightly. Also, the number of time

steps also decrease slightly as we increase the number of nodes used in Table 4.2.4 (b). These are different from what we observe from Table 4.2.3, where we see weak scalability but wall clock times increase slightly. This is due to the fact that we have the same number of CRUs as a typical CICR simulation, but since we increase the distances between CRUs on the $z$-direction, the effect of diffusion on that direction is less and the ODE problem is smoother. In Table 4.2.4 (c) We observe that the wall clock time per time step did not vary too much and memory usage per node stays the same in Table 4.2.4 (d). Hence, we conclude the implementation demonstrates weak scalability, also for the CICR problem.

Table 4.2.3: Performance study for the scalar test problem with smooth source term, solved with first-order finite volume method to $t_{\text{fin}} = 1{,}000$ ms. The mesh size in $z$-direction $N_z = (4N_x)(p/8)$ doubles as the number of processes doubles.

| (a) Wall clock time $T_p$ in HH:MM:SS | | | | | | |
|---|---|---|---|---|---|---|
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 00:00:21 | 00:00:20 | 00:00:21 | 00:00:22 | 00:00:25 | 00:00:29 |
| $32 \times 32$ | 00:00:39 | 00:00:40 | 00:00:39 | 00:00:41 | 00:00:45 | 00:00:55 |
| $64 \times 64$ | 00:04:02 | 00:04:07 | 00:04:16 | 00:04:31 | 00:04:58 | 00:05:45 |
| $128 \times 128$ | 00:45:34 | 00:46:11 | 00:46:56 | 00:49:11 | 00:52:52 | 00:59:21 |
| $256 \times 256$ | 10:13:48 | 10:17:59 | 10:28:33 | 10:41:11 | 11:09:21 | 12:09:12 |
| (b) Time steps | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 2015 | 2015 | 2015 | 2015 | 2015 | 2015 |
| $32 \times 32$ | 2018 | 2018 | 2018 | 2018 | 2018 | 2018 |
| $64 \times 64$ | 2020 | 2021 | 2020 | 2020 | 2020 | 2020 |
| $128 \times 128$ | 2019 | 2019 | 2019 | 2019 | 2019 | 2019 |
| $256 \times 256$ | 2014 | 2014 | 2014 | 2014 | 2014 | 2014 |
| (c) Wall clock time per time step in seconds | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| $32 \times 32$ | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.03 |
| $64 \times 64$ | 0.12 | 0.12 | 0.13 | 0.13 | 0.15 | 0.17 |
| $128 \times 128$ | 1.35 | 1.37 | 1.39 | 1.46 | 1.57 | 1.76 |
| $256 \times 256$ | 18.29 | 18.41 | 18.73 | 19.10 | 19.94 | 21.72 |
| (d) Observed memory usage per node (GB) | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.10 | 0.12 | 0.12 | 0.13 | 0.10 | 0.10 |
| $32 \times 32$ | 0.12 | 0.14 | 0.14 | 0.15 | 0.12 | 0.11 |
| $64 \times 64$ | 0.26 | 0.27 | 0.27 | 0.28 | 0.25 | 0.25 |
| $128 \times 128$ | 1.34 | 1.34 | 1.34 | 1.35 | 1.31 | 1.31 |
| $256 \times 256$ | 9.73 | 9.73 | 9.73 | 9.73 | 9.70 | 9.70 |
| (e) Observed total memory usage (GB) | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.10 | 0.24 | 0.49 | 1.04 | 1.61 | 3.05 |
| $32 \times 32$ | 0.12 | 0.27 | 0.55 | 1.18 | 1.90 | 3.61 |
| $64 \times 64$ | 0.26 | 0.55 | 1.10 | 2.26 | 4.05 | 7.91 |
| $128 \times 128$ | 1.34 | 2.68 | 5.36 | 10.76 | 21.04 | 41.86 |
| $256 \times 256$ | 9.73 | 19.47 | 38.91 | 77.85 | 155.22 | 310.25 |

Table 4.2.4: Performance study for the calcium problem, solved with first-order finite volume method to $t_{\text{fin}} = 100$ ms. The mesh size in $z$-direction $N_z = (4N_x)(p/8)$ doubles as the number of processes doubles.

| (a) Wall clock time $T_p$ in HH:MM:SS | | | | | | |
|---|---|---|---|---|---|---|
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 00:00:15 | 00:00:14 | 00:00:13 | 00:00:13 | 00:00:14 | 00:00:14 |
| $32 \times 32$ | 00:02:26 | 00:02:21 | 00:02:14 | 00:02:06 | 00:02:01 | 00:01:56 |
| $64 \times 64$ | 00:27:18 | 00:26:35 | 00:25:08 | 00:23:50 | 00:22:54 | 00:21:35 |
| $128 \times 128$ | 05:03:11 | 04:50:14 | 04:38:24 | 04:34:29 | 04:20:49 | 04:17:25 |
| (b) Time steps | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 2558 | 2401 | 2254 | 2126 | 1999 | 1893 |
| $32 \times 32$ | 3277 | 3158 | 2960 | 2738 | 2589 | 2424 |
| $64 \times 64$ | 4089 | 3913 | 3694 | 3458 | 3227 | 3030 |
| $128 \times 128$ | 4955 | 4724 | 4422 | 4153 | 3908 | 3679 |
| (c) Wall clock time per time step in seconds | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| $32 \times 32$ | 0.04 | 0.04 | 0.05 | 0.05 | 0.05 | 0.05 |
| $64 \times 64$ | 0.40 | 0.41 | 0.41 | 0.41 | 0.43 | 0.43 |
| $128 \times 128$ | 3.67 | 3.69 | 3.78 | 3.97 | 4.00 | 4.20 |
| (d) Observed memory usage per node (GB) | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.11 | 0.13 | 0.13 | 0.13 | 0.11 | 0.10 |
| $32 \times 32$ | 0.16 | 0.18 | 0.18 | 0.19 | 0.16 | 0.15 |
| $64 \times 64$ | 0.57 | 0.58 | 0.58 | 0.59 | 0.56 | 0.55 |
| $128 \times 128$ | 3.74 | 3.75 | 3.75 | 3.76 | 3.73 | 3.72 |
| (e) Total memory usage (GB) | | | | | | |
| $N_x \times N_y$ | $p = 8$ | $p = 16$ | $p = 32$ | $p = 64$ | $p = 128$ | $p = 256$ |
| $16 \times 16$ | 0.11 | 0.25 | 0.50 | 1.07 | 1.71 | 3.23 |
| $32 \times 32$ | 0.16 | 0.35 | 0.71 | 1.49 | 2.54 | 4.89 |
| $64 \times 64$ | 0.57 | 1.16 | 2.32 | 4.71 | 8.96 | 17.72 |
| $128 \times 128$ | 3.74 | 7.50 | 14.99 | 30.06 | 59.63 | 119.06 |

## 4.3 Single-Node GPU Performance

This section is part of [11].

Single-node GPU performance is interesting because people do not always get a cluster of GPUs to work with. Moreover, even a single GPU-CPU framework can provide advantages that multiple CPUs on their own do not offer due to specialization in each chip. As an example, we will show single-node performance of the CICR problem using the finite element method with BiCGSTAB as linear solver in this section. The next section contains results on more numerical methods and how they scale up on multiple nodes.

Table 4.3.1 summarizes the wall clock times for the CICR problem solved with the finite element method using $p = 1, 2$, and 16 MPI processes on a CPU node with two eight-core CPUs. All runs fit into the memory of the node, but some runs would take longer than the maximum time of 5 days allowed for a job on the system. For the cases, where the run with $p = 1$ process is possible, the parallel scalability is excellent to $p = 2$ processes, and using all 16 cores available on the node is clearly the fastest run in each case.

Table 4.3.2 summarizes the wall clock times for the CICR problem solved with the finite element method using a hybrid CPU/GPU node. For mesh resolution $32 \times 32 \times 128$, the wall clock time on one CPU core and one GPU is more than 5 times faster than a serial run on a CPU, but slower than using all 16 cores on a CPU node. For this coarse resolution, the time on one node with two MPI processes and two GPUs does not improve performance. This is due to time spent on data transfer between CPU and GPU memory dominating over time spent on calculation.

For mesh resolution $64 \times 64 \times 256$, the wall clock time on one CPU core and one GPU is more than 15 times faster than a serial run on a CPU. For this resolution, the wall clock time on one node with two MPI processes and two GPUs is 1.8 times

faster than using all 16 cores on a CPU node.

The amount of time needed for calculation increased rapidly due to increased mesh size. For the fine mesh resolution $128 \times 128 \times 512$, the serial run on a CPU is not available due to excessive time requirement. The wall clock time on one node with two MPI processes and two GPUs is around 3 times faster than using all 16 cores on a CPU node.

Table 4.3.1: Wall clock time for CICR problem solved with FEM using $p$ MPI processes on a CPU node. E.T. indicates excessive time requirement (more than 5 days).

| $N_x \times N_y \times N_z$ | $p = 1$ | $p = 2$ | $p = 16$ |
|---|---|---|---|
| $32 \times 32 \times 128$ | 04:12:42 | 02:11:30 | 00:20:28 |
| $64 \times 64 \times 256$ | 29:39:29 | 15:33:52 | 02:36:56 |
| $128 \times 128 \times 512$ | E.T. | E.T. | 42:07:19 |

Table 4.3.2: CICR problem solved with FEM on a hybrid node using $p$ MPI processes and one GPU per MPI process. Each MPI process launches kernels on a unique GPU. (a) Wall clock time, (b) speedup over $p = 16$ MPI processes on a sixteen-core CPU node.

| (a) Wall clock time | | |
|---|---|---|
| $N_x \times N_y \times N_z$ | $p = 1$ | $p = 2$ |
| | 1 GPU | 2 GPUs |
| $32 \times 32 \times 128$ | 00:42:33 | 00:43:32 |
| $64 \times 64 \times 256$ | 01:58:19 | 01:25:32 |
| $128 \times 128 \times 512$ | 25:09:06 | 13:46:41 |
| (b) Speedup over $p = 16$ run on CPU Node | | |
| $N_x \times N_y \times N_z$ | $p = 1$ | $p = 2$ |
| | 1 GPU | 2 GPUs |
| $32 \times 32 \times 128$ | 0.48 | 0.47 |
| $64 \times 64 \times 256$ | 1.33 | 1.83 |
| $128 \times 128 \times 512$ | 1.67 | 3.06 |

## 4.4 Multi-Node GPU Performance

Part of the results in this section is presented in a poster at the NVIDIA GPU Conference 2015.

This section reports wall clock time in HH:MM:SS and speedup for CICR problem described in Section 2.1 on hybrid CPU/GPU nodes. Speedup is calculated using CPU only runtime on one 16-core node as baseline for each mesh size. Tables of data are based on simulations using the CUDA with MPI implementation described in Section 3.3.2. We present now four sets of results, namely for both the finite element and finite volume methods and using both the BiCGSTAB and QMR methods as linear solvers.

Table 4.4.1 reports wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite element method and BiCGSTAB as linear solver. Since we have 19 hybrid CPU/GPU nodes on our cluster, we report wall clock time using 1, 2, 4, 8, and 16 nodes, as powers of 2. As explained in Section 3.3.2, in the parallel implementation using CUDA with MPI, each MPI process that runs on one CPU core have access to a unique GPU. This is to avoid kernels from different MPI processes queue up in the same GPU, rendering reduced performance. Thus, for a fixed number of nodes, we report performance using 16 CPU cores per node, using 1 CPU core and 1 GPU per node, and using 2 CPU cores and 2 GPUs per node, for mesh resolution $32 \times 32 \times 128$, $64 \times 64 \times 256$, and $128 \times 128 \times 512$. Numbers in parenthesis represent speedups against CPU only runtime on one 16-core node for the corresponding CPU/GPU nodes combination, and corresponding mesh resolution. The following is a summary of our observations:

(i) The second column shows performance on mesh resolution $32 \times 32 \times 128$. We gain no benefit using CPU/GPU hybrid nodes on this mesh. The speedup numbers

in parenthesis are all below 1, indicating the wall clock times using hybrid nodes are slower than using one 16-core CPU node. Since it is a very coarse mesh, the time saved by using GPUs is not significant compare to the extra time incurred by having data transfers between CPU and GPU memories.

(ii) The third column shows performance on mesh resolution $64 \times 64 \times 256$. We observe that the speedup numbers in parenthesis are greater than 1, indicating the wall clock times using hybrid nodes are faster than using one 16-core CPU node. We also observe that using 1 or 2 hybrid CPU/GPU nodes, the wall clock times are faster than using the same number of CPU nodes with 16 cores per node. But using 4 hybrid nodes or more, the performance is worse than using the same number of CPU only nodes. These results confirm that we are starting to take advantage of using GPUs for heavy computation, but the performance does not scale up well enough for this mesh.

(iii) The last column shows performance on mesh resolution $128 \times 128 \times 512$. We observe that the speedup numbers in parenthesis are greater than 1, indicating the wall clock times using hybrid nodes are faster than using one 16-core CPU node. We also observe that the speedup numbers increase significantly as we increase the number of nodes, which shows the performance scales up.

(iv) From the last column we observe that for a fixed number of nodes, the performance gets better as we increase the number of GPUs per node. For instance, using 1 node with 1 GPU, the wall clock time is 25:09:06, roughly 1.67 times faster than using one 16-core CPU node; using 1 node with 2 GPUs, the wall clock time is 13:46:41, which is significantly faster than using 1 node with 1 GPU. This is also true for using 2, 4, 8, and 16 nodes.

(v) From the last column we also observe that as we increase the number of hybrid nodes, we can further improve the performance. Take the case of using 2 GPUs per

node as example, we observe that the wall clock time reduces from 13:46:41 on one hybrid node to 02:28:00 on 16 hybrid nodes.

(vi) It is interesting to notice from the last column that, the performance using 2 GPUs per node is faster than the case where twice the number of CPU only nodes are used. For instance, using 2 hybrid nodes with 2 GPUs the wall clock time is 07:47:28, faster than using 4 nodes with 16-cores, where it took 11:14:39. The same applies to larger number of nodes.

Figure 4.4.1 gives a more intuitive view of the wall clock time by number of nodes for mesh resolution $128 \times 128 \times 512$. We observe that if the same number of nodes are used, 2 GPUs per node gives better performance than 1 GPU per node, which again is better than CPU only performance with all 16-cores on each node. We also observe that better performance than CPU only simulations can be achieved by using half number of hybrid nodes. For instance, using 1 node with 2 GPU is faster than using 2 CPU only nodes, using 2 nodes with 2 GPUs per node is faster than using 4 CPU only nodes, and so on.

Figure 4.4.2 gives a more intuitive view of the speedup against performance on one 16-core CPU. The CPU only speedup on one node is 1, since it is the baseline. We observe that if same number of nodes are used, the speedup increases significantly as we increase the number of GPUs used. Figure 4.4.3 demonstrates speedup line for each implementation. The CPU only implementation has the best scalability, since there are no data transfer between CPU and GPU memory. The implementation using one GPU per node has better speedup, and the implementation using two GPUs per node has the highest speedup. However, the scalability for the implementation using two GPUs per node is slightly reduced. This is due to the fact that each GPU has less computation to carry out as we increase the number of GPUs used, the time saved by using GPUs is reduced. The cost of data transfer between CPU and GPU memory

Table 4.4.1: Wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite element method and BiCGSTAB as linear solver.

| nodes (GPU/node) | $32 \times 32 \times 128$ | $64 \times 64 \times 256$ | $128 \times 128 \times 512$ |
|---|---|---|---|
| 1 node (16 cores) | 00:20:28 | 02:36:56 | 42:07:19 |
| 1 node (1 GPU) | 00:42:33 (0.48) | 01:58:19 (1.33) | 25:09:06 (1.67) |
| 1 node (2 GPUs) | 00:43:42 (0.47) | 01:25:32 (1.83) | 13:46:41 (3.06) |
| 2 nodes (16 cores) | 00:10:58 | 01:15:20 | 21:29:09 |
| 2 nodes (1 GPU) | 00:36:24 (0.56) | 01:13:29 (2.13) | 13:25:47 (3.14) |
| 2 nodes (2 GPUs) | 00:40:21 (0.50) | 01:02:11 (2.52) | 07:47:28 (5.41) |
| 4 nodes (16 cores) | 00:06:40 | 00:40:07 | 11:14:39 |
| 4 nodes (1 GPU) | 00:29:20 (0.69) | 00:49:29 (3.17) | 07:19:21 (5.75) |
| 4 nodes (2 GPUs) | 00:39:35 (0.51) | 00:51:35 (3.04) | 04:41:47 (8.97) |
| 8 nodes (16 cores) | 00:04:39 | 00:22:48 | 05:51:53 |
| 8 nodes (1 GPU) | 00:30:47 (0.66) | 00:39:42 (3.95) | 04:13:37 (9.96) |
| 8 nodes (2 GPUs) | 00:40:28 (0.50) | 00:47:25 (3.30) | 03:10:48 (13.25) |
| 16 nodes (16 cores) | N/A | 00:14:49 | 03:14:21 |
| 16 nodes (1 GPU) | 00:30:02 (0.68) | 00:36:22 (4.31) | 02:40:37 (15.73) |
| 16 nodes (2 GPUs) | 00:42:03 (0.49) | 00:47:26 (3.30) | 02:28:00 (17.08) |

should be reduced as well, since there are less data to be transferred to and from each GPU. However this reduction of cost is not significant compare to the reduction of saved time. We expect to observe better scalability for more computationally intensive problem.

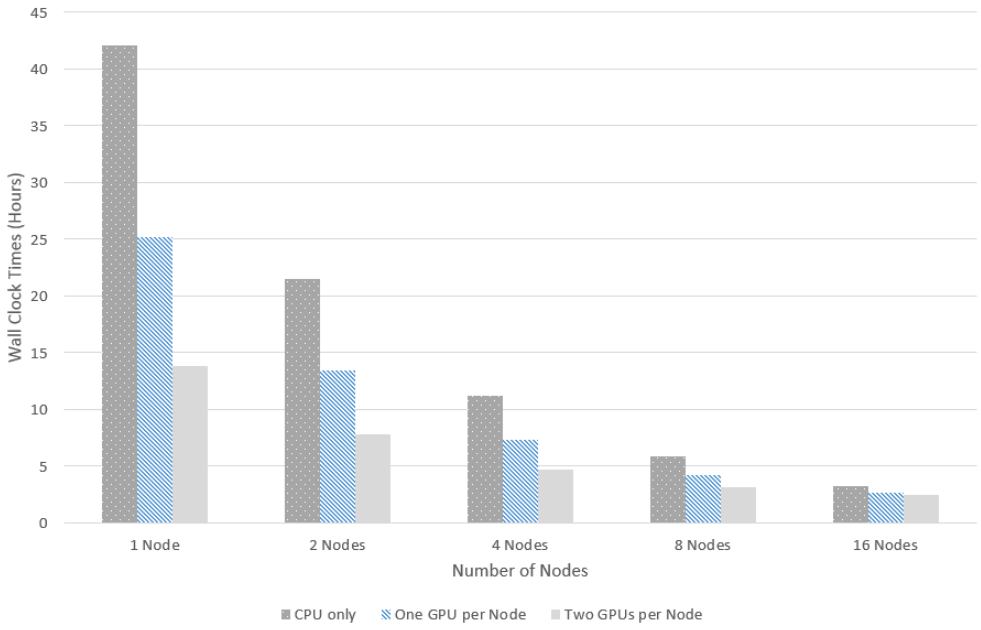



Figure 4.4.1: Wall clock time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite element method with BiCGSTAB as linear solver, for mesh resolution $128 \times 128 \times 512$.

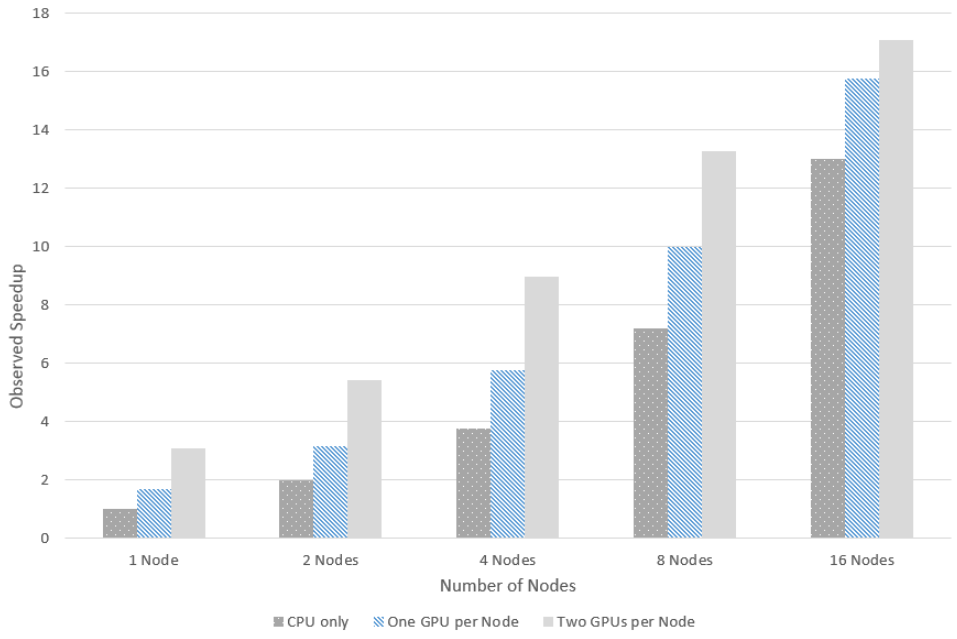



Figure 4.4.2: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite element method with BiCGSTAB as linear solver. for mesh resolution $128 \times 128 \times 512$.

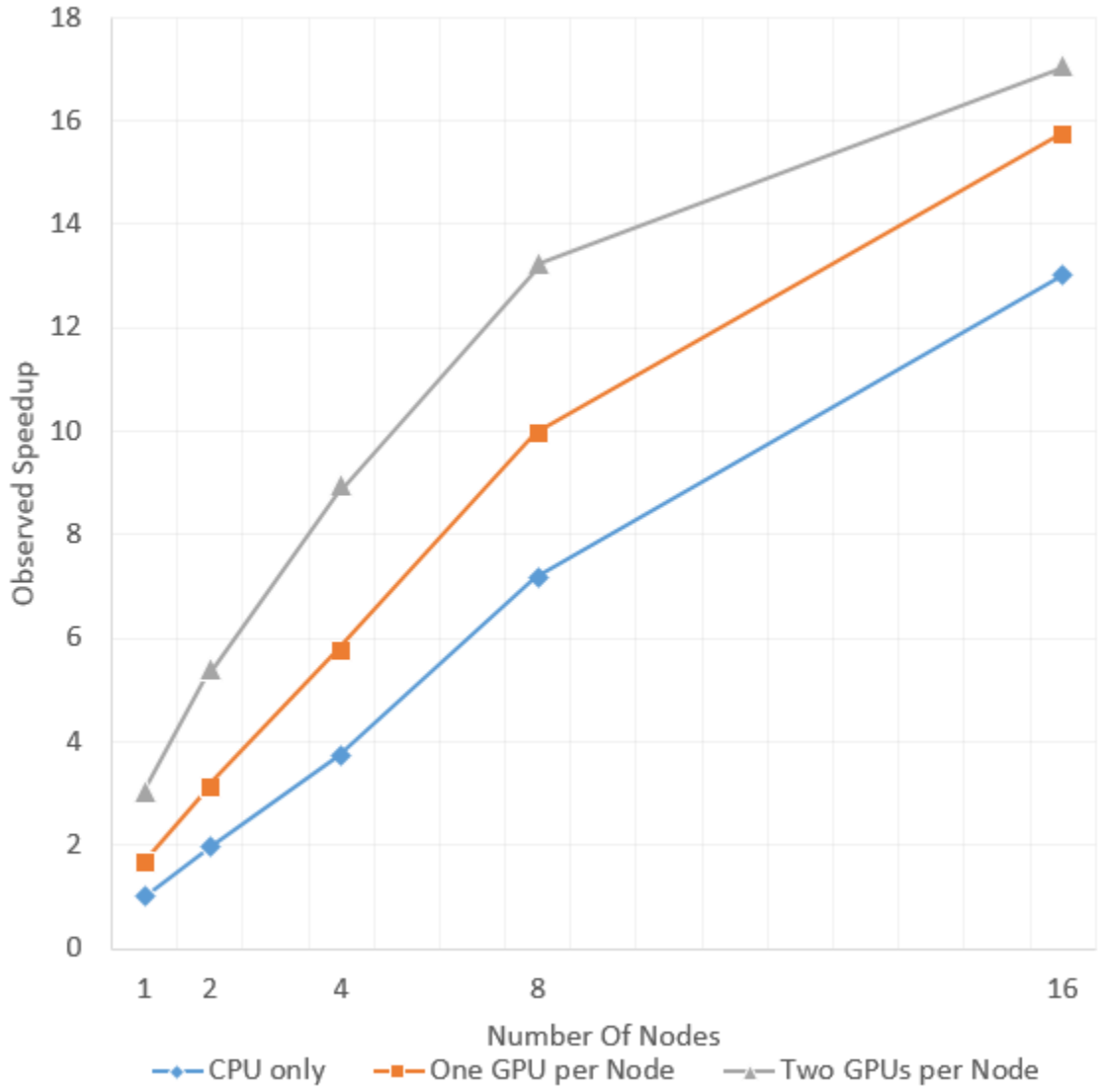



Figure 4.4.3: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite element method with BiCGSTAB as linear solver. for mesh resolution $128 \times 128 \times 512$.

Table 4.4.2 reports wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite volume method and BiCGSTAB as linear solver. Table 4.4.2 has the same format as in Table 4.4.1, where we report wall clock time using 1, 2, 4, 8, and 16 nodes, as powers of 2. As explained in Section 3.3.2, in the parallel implementation using CUDA with MPI, each MPI process that runs on one CPU core have access to a unique GPU. This is to avoid kernels from different MPI processes queue up in the same GPU, rendering reduced performance. Thus, for a fixed number of nodes, we report performance using 16 CPU cores per node, using 1 CPU core and 1 GPU per node, and using 2 CPU cores and 2 GPUs per node, for mesh resolution $32 \times 32 \times 128$, $64 \times 64 \times 256$, and $128 \times 128 \times 512$. Numbers in parenthesis represent speedups against CPU only runtime on one 16-core node for the corresponding CPU/GPU nodes combination, and corresponding mesh resolution. The following is a summary of our observations:

(i) The second column shows performance on mesh resolution $32 \times 32 \times 128$. We gain no benefit using CPU/GPU hybrid nodes on this mesh. The speedup numbers in parenthesis are all below 1, indicating the wall clock times using hybrid nodes are slower than using one 16-core CPU node. Since it is a very coarse mesh, the time saved by using GPUs is not significant compare to the extra time incurred by having data transfers between CPU and GPU memories.

(ii) The third column shows performance on mesh resolution $64 \times 64 \times 256$. We observe that the speedup numbers in parenthesis are greater than 1, indicating the wall clock times using hybrid nodes are faster than using one 16-core CPU node. We also observe that using 1 or 2 hybrid CPU/GPU nodes, the wall clock times are faster than using the same number of CPU nodes with 16 cores per node. But using 4 hybrid nodes or more, the performance is worse than using the same number of CPU

only nodes. These results confirm that we are starting to take advantage of using GPUs for heavy computation, but the performance does not scale up well enough for this mesh.

(iii) The last column shows performance on mesh resolution $128 \times 128 \times 512$. We observe that the speedup numbers in parenthesis are greater than 1, indicating the wall clock times using hybrid nodes are faster than using one 16-core CPU node. We also observe that the speedup numbers increase significantly as we increase the number of nodes, which shows the performance scales up.

(iv) From the last column we observe that for a fixed number of nodes, the performance gets better as we increase the number of GPUs per node. For instance, using 1 node with 1 GPU, the wall clock time is 16:22:51, roughly 1.64 times faster than using one 16-core CPU node; using 1 node with 2 GPUs, the wall clock time is 08:58:16, which is significantly faster than using 1 node with 1 GPU. This is also true for using 2, 4, 8, and 16 nodes.

(v) From the last column we also observe that as we increase the number of hybrid nodes, we can further improve the performance. Take the case of using 2 GPUs per node as example, we observe that the wall clock time reduces from 08:58:16 on one hybrid node to 01:48:19 on 16 hybrid nodes.

(vi) It is interesting to notice from the last column that, the performance using 2 GPUs per node is faster than the case where twice the number of CPU only nodes are used. For instance, using 2 hybrid nodes with 2 GPUs the wall clock time is 05:03:27, faster than using 4 nodes with 16-cores, where it took 07:21:17. The same applies to larger number of nodes.

Comparing Table 4.4.2 and Table 4.4.1 we observe the same pattern of speedup against CPU-only performance. It indicates our approach of hybrid CUDA/MPI implementation is appropriate for both the finite element and the finite volume method.

Table 4.4.2: Wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite volume method and BiCGSTAB as linear solver.

| nodes (GPU/node) | $32 \times 32 \times 128$ | $64 \times 64 \times 256$ | $128 \times 128 \times 512$ |
|---|---|---|---|
| 1 node (16 cores) | 00:12:58 | 02:25:55 | 26:53:37 |
| 1 node (1 GPU) | 00:31:09 (0.39) | 02:00:32 (1.19) | 16:22:51 (1.64) |
| 1 node (2 GPUs) | 00:30:33 (0.39) | 01:30:03 (1.60) | 08:58:16 (3.00) |
| 2 nodes (16 cores) | 00:07:24 | 01:10:26 | 13:56:38 |
| 2 nodes (1 GPU) | 00:23:46 (0.51) | 01:15:14 (1.91) | 08:39:43 (3.10) |
| 2 nodes (2 GPUs) | 00:28:08 (0.42) | 01:05:04 (2.21) | 05:03:27 (5.32) |
| 4 nodes (16 cores) | 00:06:30 | 00:39:04 | 07:21:17 |
| 4 nodes (1 GPU) | 00:20:46 (0.58) | 00:52:09 (2.76) | 04:45:01 (5.66) |
| 4 nodes (2 GPUs) | 00:28:02 (0.43) | 00:55:10 (2.61) | 03:05:28 (8.70) |
| 8 nodes (16 cores) | 00:07:26 | 00:25:46 | 03:54:47 |
| 8 nodes (1 GPU) | 00:20:16 (0.59) | 00:43:12 (3.33) | 02:49:08 (9.54) |
| 8 nodes (2 GPUs) | 00:28:03 (0.43) | 00:51:44 (2.78) | 02:11:12 (12.30) |
| 16 nodes (16 cores) | N/A | 00:21:29 | 02:17:48 |
| 16 nodes (1 GPU) | 00:20:09 (0.59) | 00:39:35 (3.64) | 01:56:17 (13.88) |
| 16 nodes (2 GPUs) | 00:28:46 (0.42) | 00:51:41 (2.79) | 01:48:19 (14.90) |

Figure 4.4.4 gives a more intuitive view of the wall clock time by number of nodes for mesh resolution $128 \times 128 \times 512$. We observe that if the same number of nodes are used, 2 GPUs per node gives better performance than 1 GPU per node, which again is better than CPU only performance with all 16-cores on each node. We also observe that better performance than CPU only simulations can be achieved by using half number of hybrid nodes. For instance, using 1 node with 2 GPU is faster than using 2 CPU only nodes, using 2 nodes with 2 GPUs per node is faster than using 4 CPU only nodes, and so on. This can be seen easier from Figure 4.4.5.

Figure 4.4.5 gives a more intuitive view of the speedup against performance on one 16-core CPU. The CPU only speedup on one node is 1, since it is the baseline.

We observe that if same number of nodes are used, the speedup increases significantly as we increase the number of GPUs used. Figure 4.4.6 demonstrates speedup line for each implementation. We observe the same pattern as in Figure 4.4.3. The CPU only implementation has the best scalability, since there are no data transfer between CPU and GPU memory. The implementation using one GPU per node has better speedup, and the implementation using two GPUs per node has the highest speedup. However, the scalability for the implementation using two GPUs per node is slightly reduced.

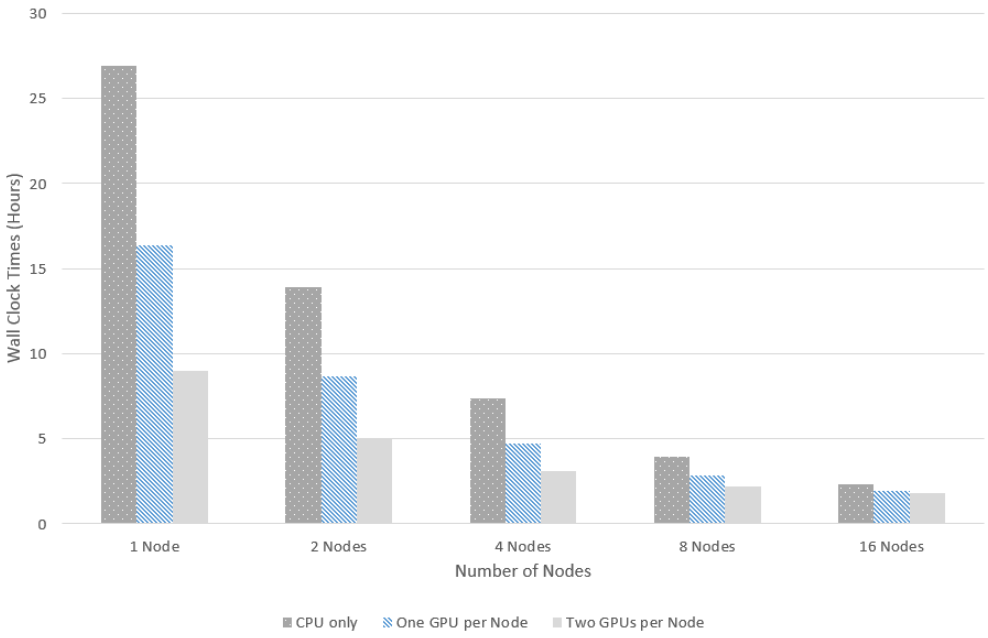



Figure 4.4.4: Wall clock time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite volume method with BiCGSTAB as linear solver, for mesh resolution $128 \times 128 \times 512$.

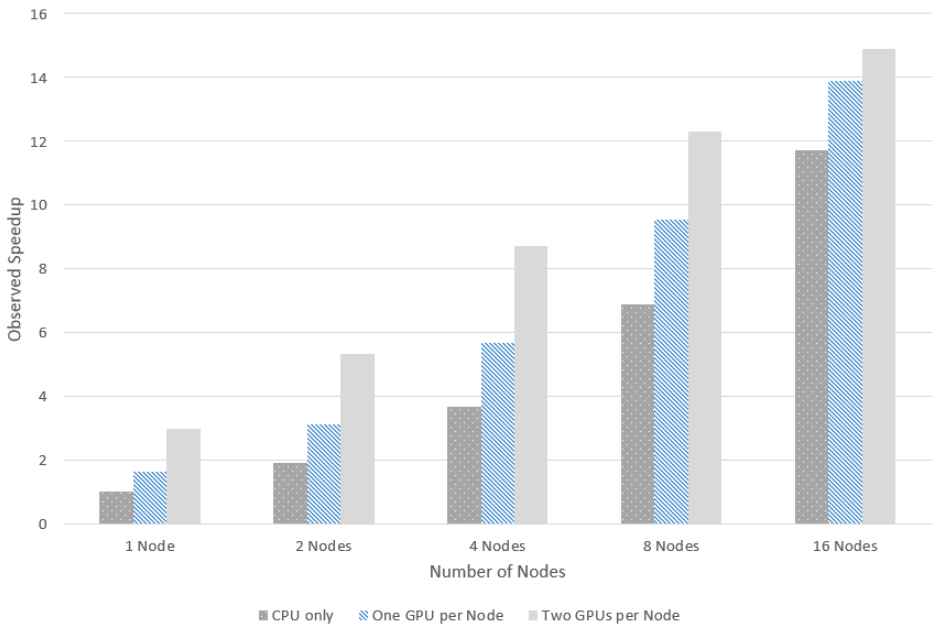



Figure 4.4.5: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite volume method with BiCGSTAB as linear solver. for mesh resolution $128 \times 128 \times 512$.

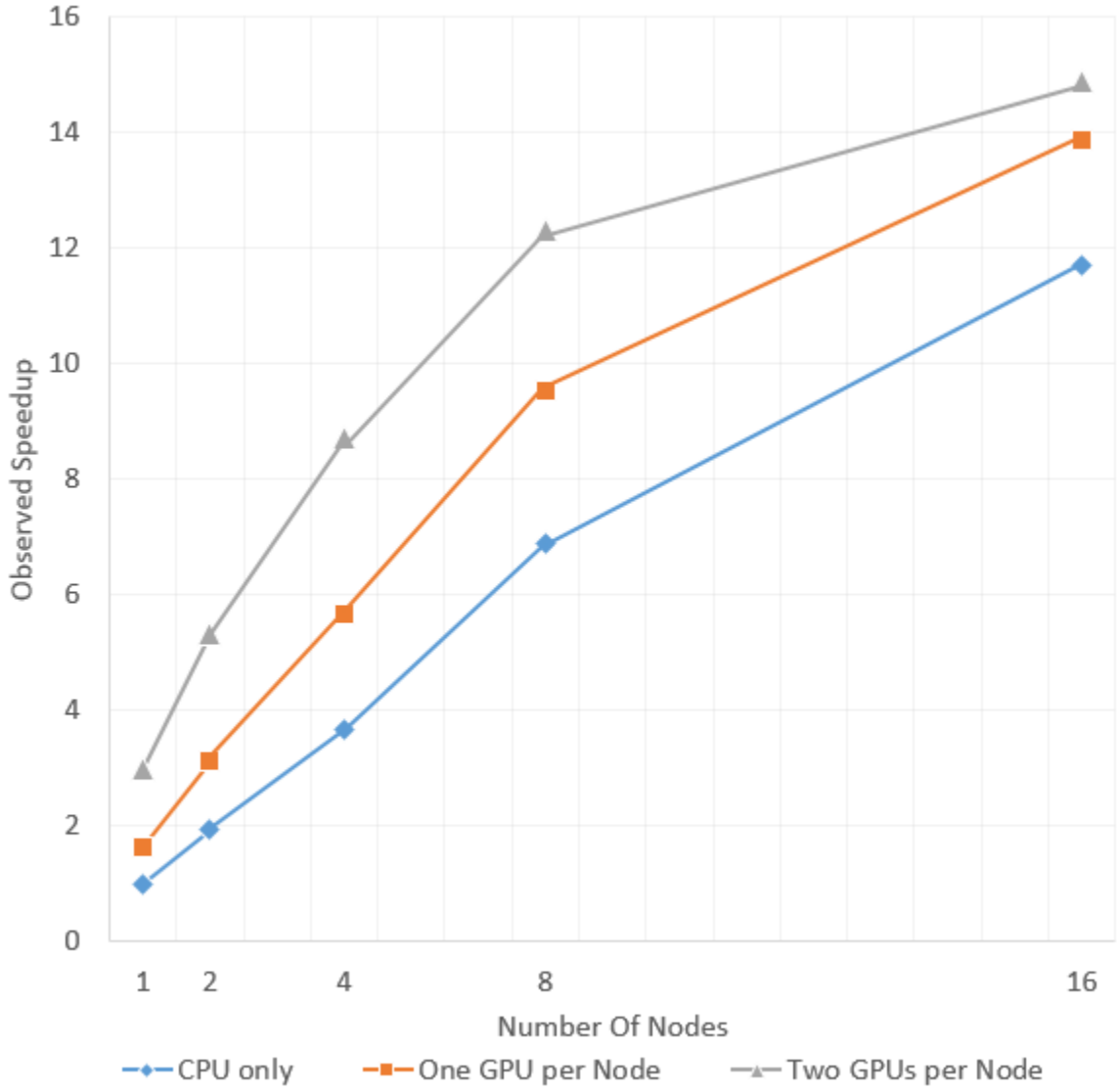



Figure 4.4.6: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite volume method with BiCGSTAB as linear solver. for mesh resolution $128 \times 128 \times 512$.

Table 4.4.3 and Table 4.4.4 discussed below are the same studies presented in Table 4.4.1 and Table 4.4.2, except that the linear solver used is QMR. As observed in Table 3.2.1 and Table 3.2.2, the observed average number of iterations in linear solver for QMR is higher than BiCGSTAB. This indicates the program is more computationally intensive than using BiCGSTAB, thus we expect to see higher speedup for the implementation using CUDA and MPI.

Table 4.4.3 reports wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite element method and QMR as linear solver. The first column shows performance on mesh resolution $32 \times 32 \times 128$. We observe that using the parallel implementation using MPI on one 16-core CPU the simulation runs for about 31 minutes. As in Table 4.4.1, 1 node (1 GPU) means one MPI process running on one CPU core with access to one GPU, 1 node (2 GPUs) means two MPI processes, each accessing one GPU on the node. It is the same with simulations on multiple hybrid CPU/GPU nodes. Here we observe that using one CPU core with one GPU the simulation runs for 40 minutes. The performance of adding a GPU is not faster than using all 16 cores on the node. We also observe that using more nodes with one or two GPUs per node we can get better performance than the 16-core CPU. This is different with Table 4.4.1, where we cannot get better performance than a 16-core CPU on this coarse mesh. It confirms that the program is more computationally intensive. However, the speedup is very low that we cannot beat multiple CPU nodes. The second column shows performance on mesh resolution $64 \times 64 \times 256$. We observe that using one 16-core node the simulation runs for about 4 hours. Using one CPU core and one GPU, the simulation runs for about 2 hours 38 minutes, faster than 16-core CPU result. Reading along this column, we observe the wall clock time been further reduced using more nodes and two GPUs per node. Also, we observe that

the speedup numbers are larger than those in Table 4.4.1. The third column shows performance on mesh resolution $128 \times 128 \times 512$. We observe that using one 16-core node the simulation runs for more than 72 hours. We observe that using one CPU core and one GPU, the simulation runs for around 37 hours. Using one node with two GPUs, the simulation runs for 19 hours and 29 minutes. This is approximately 3.7 times faster than one 16-core CPU, higher than 3 times faster in Table 4.4.1. Reading along the column we observe the wall clock time decreases significantly using more nodes, all the way to 3 hours on 16 nodes. The speedup using 16 nodes and 2 GPUs per node is 23.5, much higher than 17.08 in Table 4.4.1.

Table 4.4.3: Wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite element method and QMR as linear solver.

| nodes (GPU/node) | $32 \times 32 \times 128$ | $64 \times 64 \times 256$ | $128 \times 128 \times 512$ |
|---|---|---|---|
| 1 node (16 cores) | 00:30:42 | 03:56:54 | 72:05:43 |
| 1 node (1 GPU) | 00:40:35 (0.76) | 02:37:42 (1.50) | 37:06:58 (1.94) |
| 1 node (2 GPUs) | 00:25:32 (1.20) | 01:35:14 (2.49) | 19:29:32 (3.70) |
| 2 node (1 GPU) | 00:29:36 (1.04) | 01:34:00 (2.52) | 19:31:06 (3.69) |
| 2 node (2 GPUs) | 00:20:14 (1.52) | 01:04:15 (3.69) | 10:41:27 (6.74) |
| 4 node (1 GPU) | 00:21:23 (1.43) | 01:00:17 (3.93) | 10:28:33 (6.88) |
| 4 node (2 GPUs) | 00:18:27 (1.66) | 00:53:36 (4.42) | 06:07:49 (11.76) |
| 8 node (1 GPU) | 00:19:23 (1.58) | 00:48:20 (4.90) | 05:53:10 (12.25) |
| 8 node (2 GPUs) | 00:18:29 (1.66) | 00:51:34 (4.59) | 03:57:46 (18.19) |
| 16 node (1 GPU) | 00:19:59 (1.53) | 00:46:58 (5.04) | 03:42:36 (19.43) |
| 16 node (2 GPUs) | 00:20:14 (1.52) | 00:56:26 (4.20) | 03:04:03 (23.50) |

Figure 4.4.7 gives a more intuitive view of the wall clock time by number of nodes for mesh resolution $128 \times 128 \times 512$. We observe that if the same number of nodes are used, 2 GPUs per node gives better performance than 1 GPU per node, which again is better than CPU only performance with all 16-cores on each node. We also

observe that better performance than CPU only simulations can be achieved by using half number of hybrid nodes. For instance, using 1 node with 2 GPU is faster than using 2 CPU only nodes, using 2 nodes with 2 GPUs per node is faster than using 4 CPU only nodes, and so on. This can be seen easier from Figure 4.4.8.

Figure 4.4.8 gives a more intuitive view of the speedup against performance on one 16-core CPU. The CPU only speedup on one node is 1, since it is the baseline. We observe that if same number of nodes are used, the speedup increases significantly as we increase the number of GPUs used. Figure 4.4.9 demonstrates speedup line for each implementation. We observe the same pattern as in Figure 4.4.3. The CPU only implementation has the best scalability, since there are no data transfer between CPU and GPU memory. The implementation using one GPU per node has better speedup, and the implementation using two GPUs per node has the highest speedup. However, the scalability for the implementation using two GPUs per node is slightly reduced. Compare to Figure 4.4.3, we gain much higher speedup using 16 hybrid CPU/GPU nodes, due to the more computationally intensive program. This indicates that better scalability can be expected for more computationally intensive program. It is preferable to use more nodes on larger problems.

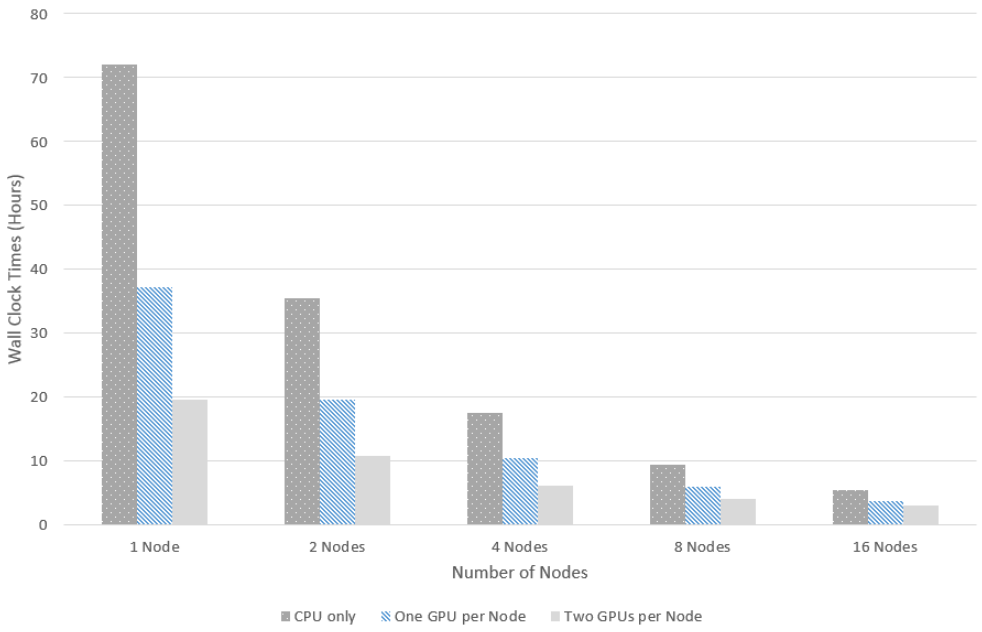



Figure 4.4.7: Wall clock time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite element method with QMR as linear solver, for mesh resolution $128 \times 128 \times 512$.

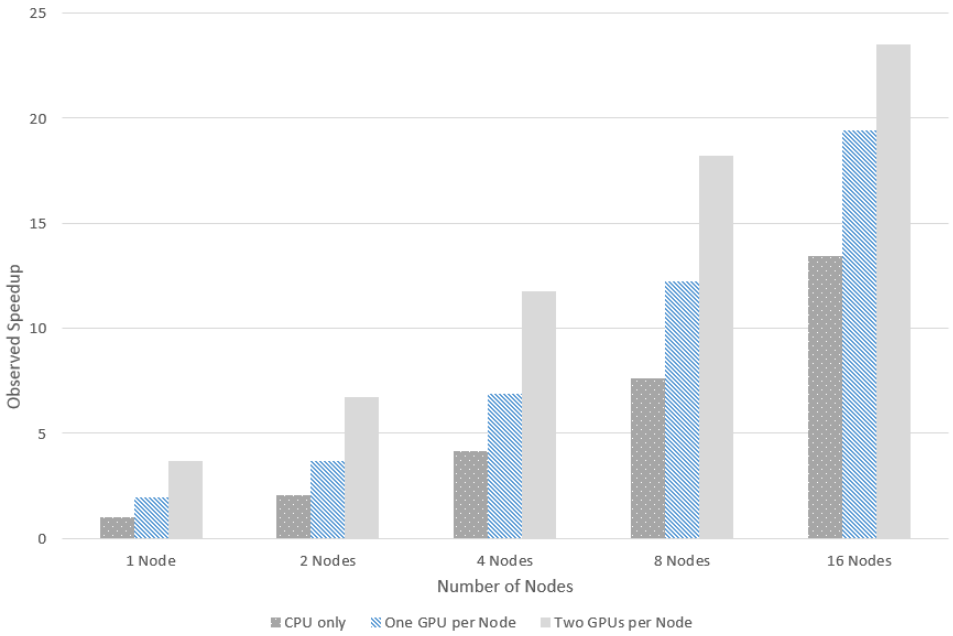



Figure 4.4.8: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite element method with QMR as linear solver. for mesh resolution $128 \times 128 \times 512$.

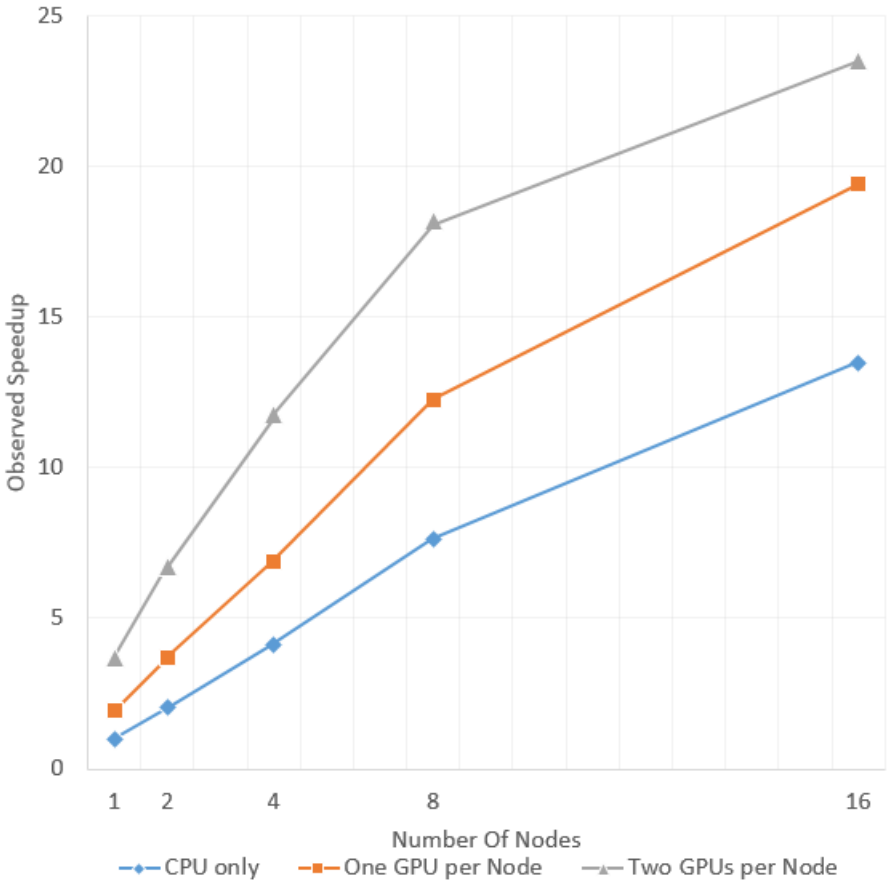



Figure 4.4.9: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite element method with QMR as linear solver. for mesh resolution $128 \times 128 \times 512$.

Table 4.4.4 reports wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite volume method and QMR as linear solver. The first column shows performance on mesh resolution $32 \times 32 \times 128$. We observe that using the parallel implementation using MPI on one 16-core CPU the simulation runs for about 18 minutes. As in Table 4.4.1, 1 node (1 GPU) means one MPI process running on one CPU core with access to one GPU, 1 node (2 GPUs) means two MPI processes, each access one GPU on the node. It is the same with simulations on multiple hybrid CPU/GPU nodes. Here we observe that using one CPU core with one GPU the simulation runs for 26 minutes. The performance of adding a GPU is not faster than using all 16 cores on the node. We also observe that using more nodes with one or two GPUs per node we can get better performance than the 16-core CPU. This is different with Table 4.4.2, where we cannot get better performance than the 16-core CPU on this coarse mesh. It also confirms that the program is more computationally intensive. However, the speedup is very low that we cannot beat multiple CPU nodes. The second column shows performance on mesh resolution $64 \times 64 \times 256$. We observe that using one 16-core node the simulation runs for about 3 hours 41 minutes. Using one CPU core and one GPU, the simulation runs for about 2 hours 29 minutes, faster than the 16-core CPU result. Reading along the column we observe using more nodes and two GPUs per node we can further reduce the wall clock time. Also, we observe that the speedup numbers are larger than those in Table 4.4.2. The third column shows performance on mesh resolution $128 \times 128 \times 512$. We observe that using one 16-core node the simulation runs for more than 40 hours. We observe that using one CPU core and one GPU, the simulation runs for around 22 hours. Using one node with two GPUs, the simulation runs for 11 hours and 50 minutes. This is approximately 3.45 times faster than one 16-core CPU, higher than 3 times faster in Table 4.4.1. Reading along

the column we observe the wall clock times decrease significantly using more nodes, all the way to 2 hours 17 minutes on 16 nodes. The speedup using 16 nodes and 2 GPUs per node is 17.83, much higher than 14.9 in Table 4.4.1.

Table 4.4.4: Wall clock time in HH:MM:SS for CICR problem on hybrid CPU/GPU nodes and speedup against CPU only runtime on one 16-core node, for finite volume method and QMR as linear solver.

| nodes (GPU/node) | $32 \times 32 \times 128$ | $64 \times 64 \times 256$ | $128 \times 128 \times 512$ |
|---|---|---|---|
| 1 node (16 cores) | 00:18:24 | 03:41:39 | 40:51:02 |
| 1 node (1 GPU) | 00:26:02 (0.71) | 02:29:48 (1.48) | 22:02:11 (1.85) |
| 1 node (2 GPUs) | 00:16:48 (1.10) | 01:34:21 (2.35) | 11:50:11 (3.45) |
| 2 node (1 GPU) | 00:17:38 (1.04) | 01:32:18 (2.40) | 11:46:06 (3.47) |
| 2 node (2 GPUs) | 00:13:48 (1.33) | 01:08:00 (3.26) | 06:34:43 (6.21) |
| 4 node (1 GPU) | 00:14:15 (1.29) | 01:03:01 (3.52) | 06:23:56 (6.38) |
| 4 node (2 GPUs) | 00:13:20 (1.37) | 01:00:34 (3.66) | 03:54:06 (10.47) |
| 8 node (1 GPU) | 00:13:34 (1.36) | 00:55:21 (4.00) | 03:44:44 (10.91) |
| 8 node (2 GPUs) | 00:15:47 (1.17) | 01:04:09 (3.46) | 02:43:41 (14.97) |
| 16 node (1 GPU) | 00:15:50 (1.16) | 00:55:55 (3.96) | 02:32:39 (16.06) |
| 16 node (2 GPUs) | 00:20:48 (0.88) | 01:10:58 (3.12) | 02:17:29 (17.83) |

Figure 4.4.10 gives a more intuitive view of the wall clock time by number of nodes for mesh resolution $128 \times 128 \times 512$. We observe that if the same number of nodes are used, 2 GPUs per node gives better performance than 1 GPU per node, which again is better than CPU only performance with all 16-cores on each node. We also observe that better performance than CPU only simulations can be achieved by using half number of hybrid nodes. For instance, using 1 node with 2 GPU is faster than using 2 CPU only nodes, using 2 nodes with 2 GPUs per node is faster than using 4 CPU only nodes, and so on. This can be seen easier from Figure 4.4.11.

Figure 4.4.11 gives a more intuitive view of the speedup against performance on one 16-core CPU. The CPU only speedup on one node is 1, since it is the baseline.

We observe that if same number of nodes are used, the speedup increase significantly as we increase the number of GPUs used. Figure 4.4.12 demonstrates speedup line for each implementation. We observe the same pattern as in Figure 4.4.6. The CPU only implementation has the best scalability, since there are no data transfer between CPU and GPU memory. The implementation using one GPU per node has better speedup, and the implementation using two GPUs per node has the highest speedup. However, the scalability for the implementation using two GPUs per node is slightly reduced. Compare to Figure 4.4.6, we gain much higher speedup using 16 hybrid CPU/GPU nodes, due to the more computationally intensive program. This indicates that better scalability can be expected for more computationally intensive program. It is preferable to use more nodes on larger problems.

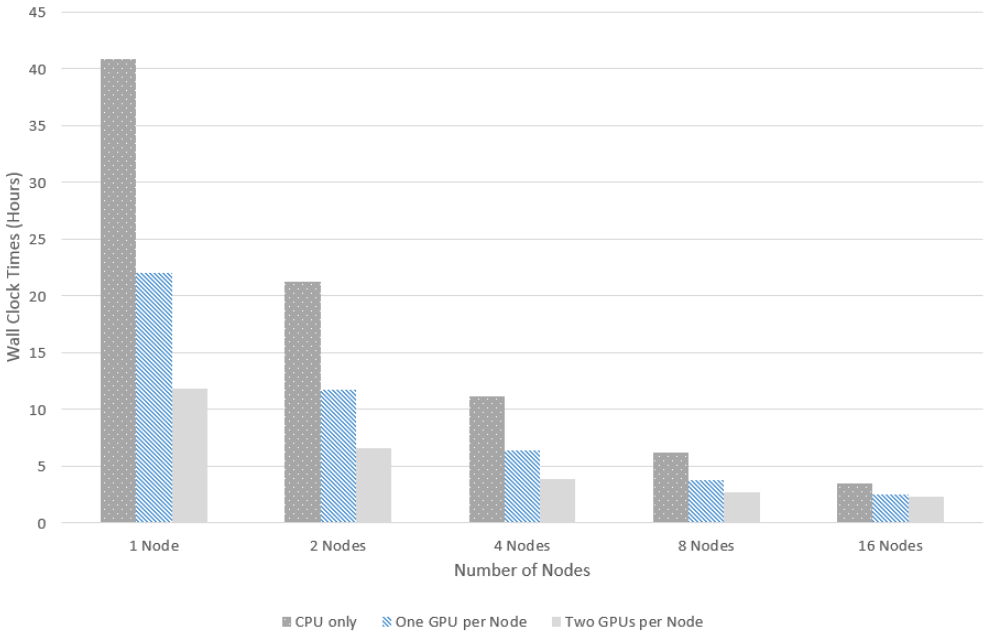



Figure 4.4.10: Wall clock time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite volume method with QMR as linear solver, for mesh resolution $128 \times 128 \times 512$.

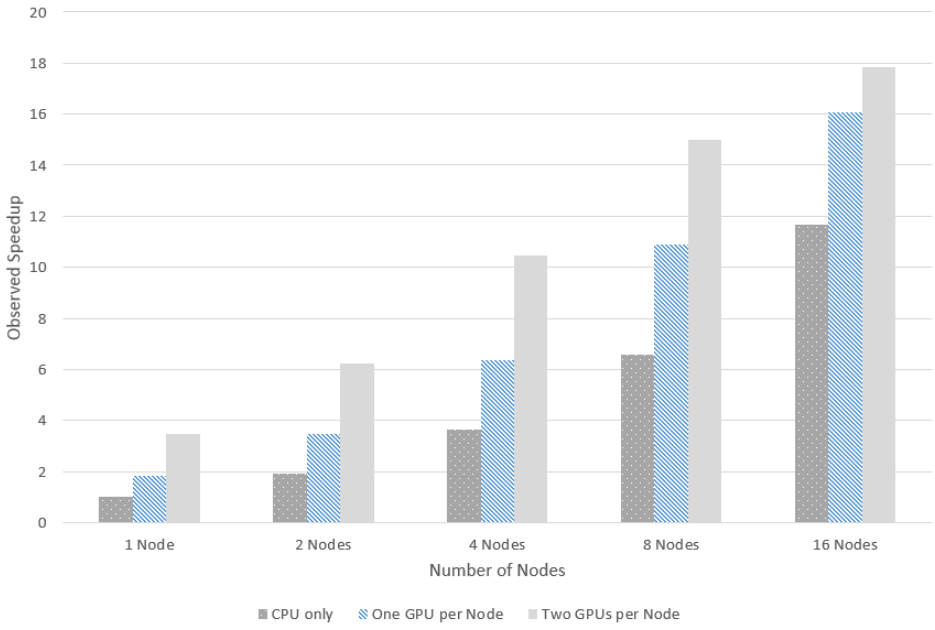



Figure 4.4.11: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite volume method with QMR as linear solver. for mesh resolution $128 \times 128 \times 512$.

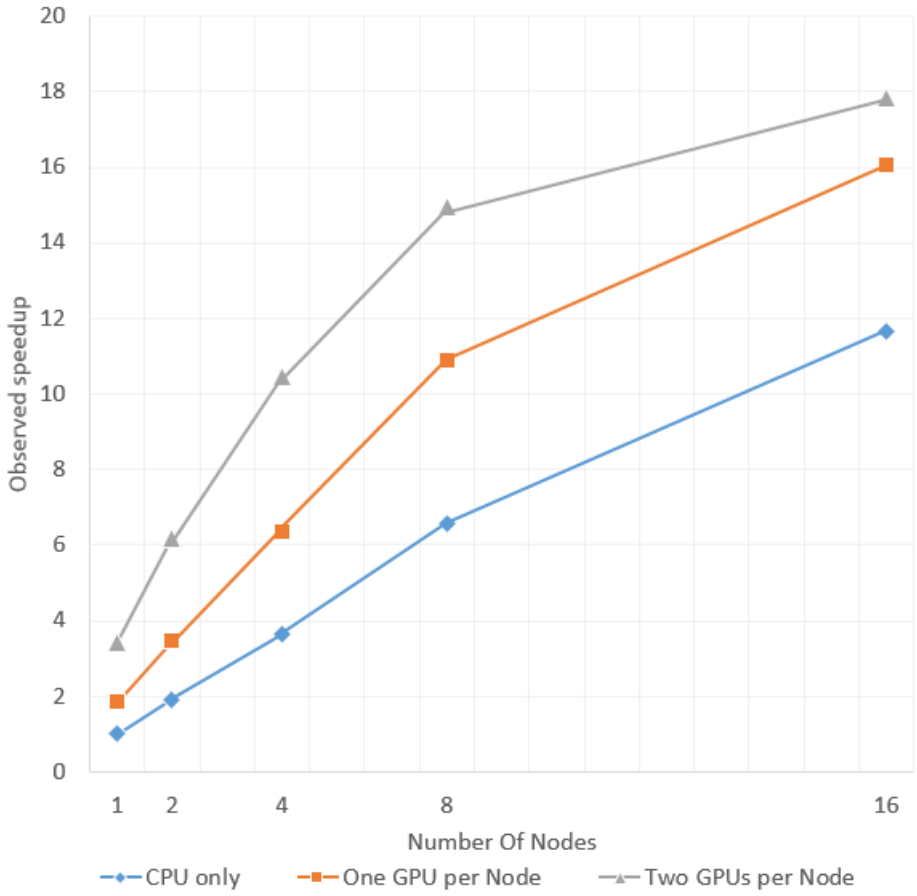



Figure 4.4.12: Observed speedup over one 16-core CPU node reference time by number of nodes using CPU only node, one GPU per node and two GPU per node, for CICR problem solved with finite volume method with QMR as linear solver. for mesh resolution $128 \times 128 \times 512$.

# CHAPTER 5

# CONCLUSIONS

We capture the effect of advection in three-dimensional long-time simulation of the CICR model. Different advection speeds of the calcium waves are observed, which in turn show our numerical methods are producing physiologically plausible results.

We demonstrate convergence of the finite volume method numerically and compare the results to those obtained by the finite element method from [7]. We show convergence for scalar test problems with choices between smooth versus non-smooth source terms and first-order versus second-order discretization of the advection term, in both two- and three-dimensional mesh spacing. If there is no advection, the convergence rates are the same as for the finite element method. For the CICR problem, where the source term is highly non-smooth, we recommend using the first-order discretization of the advection term.

Through a series of parallel performance studies, we show the strong and weak scalability of the parallel implementation using MPI. This indicates we can solve the CICR problem faster and on finer meshes, provided that we have access to a large number of CPU nodes.

With a parallel implementation using CUDA and MPI, we show how to combine several hybrid CPU/GPU nodes in a multi-node distributed-memory compute cluster with high performance interconnect successfully. The results demonstrate that using CUDA and MPI on one hybrid node with two CPUs and two GPUs, the CICR problem can be solved much faster than using all 16 cores of two eight-core CPUs on a CPU node. The data transfer between CPU and GPU memory is inevitable, but performance can be improved by splitting kernels and using non-blocking MPI communications. Moreover, we show the scalability of the implementation, where the

wall clock time can be further reduced by using multiple GPU nodes. It is noticeable that we can outperform the CPU-only implementation by using half as many hybrid CPU/GPU nodes.

# BIBLIOGRAPHY

[1] Oluwapelumi Adenikinju, Julian Gilyard, Joshua Massey, Thomas Stitt, Jonathan Graf, Xuan Huang, Samuel Khuvis, Matthias K. Gobbert, Yu Wang, and Marc Olano. Real time global illumination solutions to the radiosity algorithm using hybrid CPU/GPU nodes. Technical Report HPCF–2014–15, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

[2] Lynda M. Blayney and F. Anthony Lai. Ryanodine receptor-mediated arrhythmias and sudden cardiac death. *Pharmacology & Therapeutics*, vol. 123, no. 2, pp. 151–177, 2009.

[3] Matthew W. Brewster. The Influence of Stochastic Parameters on Calcium Waves in a Heart Cell. Senior thesis, Department of Mathematics and Statistics, University of Maryland, Baltimore County, 2014.

[4] Guangye Chen, Luis Chacón, and Daniel C Barnes. An efficient mixed-precision, hybrid CPU-GPU implementation of a nonlinearly implicit one-dimensional particle-in-cell algorithm. *Journal of Computational Physics*, vol. 231, no. 16, pp. 5374–5388, 2012.

[5] Zana A. Coulibaly, Bradford E. Peercy, and Matthias K. Gobbert. Insight into spontaneous recurrent calcium waves in a 3-D cardiac cell based on analysis of a 1-D deterministic model. *Int. J. Comp. Math.*, vol. 92, no. 3, pp. 591–607, 2015.

[6] Adam Cunningham, Gerald Payton, Jack Slettebak, Jordi Wolfson-Pou, Jonathan Graf, Xuan Huang, Samuel Khuvis, Matthias K. Gobbert, Thomas Salter, and David J. Mountain. Pushing the limits of the maya cluster. Technical

Report HPCF–2014–14, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

[7] Matthias K. Gobbert. Long-time simulations on high resolution meshes to model calcium waves in a heart cell. *SIAM J. Sci. Comput.*, vol. 30, no. 6, pp. 2922–2947, 2008.

[8] Jonathan Graf, Xuan Huang, and Matthias K. Gobbert. Performance of linear and non-linear time-dependent partial differential equations on a hybrid CPU/GPU node. Submitted ICCS (2015).

[9] Alexander L. Hanhart, Matthias K. Gobbert, and Leighton T. Izu. A memory-efficient finite element method for systems of reaction-diffusion equations with non-smooth forcing. *J. Comput. Appl. Math.*, vol. 169, no. 2, pp. 431–458, 2004.

[10] Xuan Huang and Matthias K. Gobbert. Parallel performance studies for a three-species application problem on maya 2013. Technical Report HPCF–2014–8, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

[11] Xuan Huang and Matthias K. Gobbert. Long-time simulation of calcium induced calcium release in a heart cell using finite element method on a hybrid CPU/GPU node, 2015. 23rd High Performance Computing Symposium (HPC 2015).

[12] Xuan Huang and Matthias K. Gobbert. Parallel performance studies for a three-species application problem on the cluster maya. Technical Report HPCF–2015–8, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2015.

[13] Xuan Huang, Matthias K. Gobbert, Bradford E. Peercy, Stefan Kopecz, Philipp Birken, and Andreas Meister. Order investigation of scalable memory-efficient

finite volume methods for parabolic advection-diffusion-reaction equations with point sources. In preparation (2015).

[14] Leighton T. Izu, Joseph R. H. Mauban, C. William Balke, and W. Gil Wier. Large currents generate cardiac $Ca^{2+}$ sparks. *Biophys. J.*, vol. 80, pp. 88–102, 2001.

[15] Leighton T. Izu, Shawn A. Means, John N. Shadid, Ye Chen-Izu, and C. William Balke. Interplay of ryanodine receptor distribution and calcium dynamics. *Biophys. J.*, vol. 91, pp. 95–112, 2006.

[16] Leighton T. Izu, W. Gil Wier, and C. William Balke. Evolution of cardiac calcium waves from stochastic calcium sparks. *Biophys. J.*, vol. 80, pp. 103–120, 2001.

[17] Samuel Khuvis and Matthias K. Gobbert. Parallel performance studies for an elliptic test problem on maya 2013. Technical Report HPCF–2014–6, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2014.

[18] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*, vol. 31 of *Cambridge Texts in Applied Mathematics*. Cambridge University Press, 2002.

[19] Christopher D. Marcotte and Roman O. Grigoriev. Implementation of PDE models of cardiac dynamics on GPUs using OpenCL. *ArXiv e-prints*, 2013.

[20] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.

[21] Nil Mistry, Jordan Ramsey, Benjamin Wiley, Jackie Yanchuck, Xuan Huang, and Matthias K. Gobbert. Throughput studies on an InfiniBand interconnect via All-

to-All communications, 2015. 23rd High Performance Computing Symposium (HPC 2015).

[22] Nil Mistry, Jordan Ramsey, Benjamin Wiley, Jackie Yanchuck, Xuan Huang, Matthias K. Gobbert, Christopher Mineo, and David Mountain. Contention of communications in switched networks with applications to parallel sorting. Technical Report HPCF–2013–13, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2013.

[23] Ravindra Pethiyagoda, Scott W. McCue, Timothy J. Moroney, and Julian M. Back. Jacobian-free Newton–Krylov methods with GPU acceleration for computing nonlinear ship wave patterns. *Journal of Computational Physics*, vol. 269, pp. 297–313, 2014.

[24] Hans-Görg Roos, Martin Stynes, and Lutz Tobiska. *Robust Numerical Methods for Singularly Perturbed Differential Equations*, vol. 24 of *Springer Series in Computational Mathematics*. Springer-Verlag, 2nd edition, 2008.

[25] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, second edition, 2003.

[26] Jonas Schäfer, Xuan Huang, Stefan Kopecz, Philipp Birken, Matthias K. Gobbert, and Andreas Meister. A memory-efficient finite volume method for advection-diffusion-reaction systems with non-smooth sources. *Numer. Methods Partial Differential Equations*, vol. 31, no. 1, pp. 143–167, 2015.

[27] Thomas I. Seidman, Matthias K. Gobbert, David W. Trott, and Martin Kružík. Finite element approximation for time-dependent diffusion with measure-valued source. *Numer. Math.*, vol. 122, no. 4, pp. 709–723, 2012.

[28] Lawrence F. Shampine. *Numerical Solution of Ordinary Differential Equations*. Chapman & Hall, 1994.

[29] Lawrence F. Shampine and Mark W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, vol. 18, no. 1, pp. 1–22, 1997.

[30] Yohannes Shiferaw, Gary L. Aistrup, and J. Andrew Wasserstrom. Intracellular Ca2+ waves, afterdepolarizations, and triggered arrhythmias. *Cardiovascular Research*, vol. 95, no. 3, pp. 265–8, 2012.

[31] Julien C. Thibault and Inanc Senocak. "CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows". *47th AIAA Aerospace Sciences Meeting*, 2009.

[32] Vidar Thomée. *Galerkin Finite Element Methods for Parabolic Problems*, vol. 25 of *Springer Series in Computational Mathematics*. Springer-Verlag, second edition, 2006.

[33] Paul Tranquilli, Ross Glandon, and Adrian Sandu. CUDA acceleration of a matrix-free Rosenbrock-K method applied to the shallow water equations. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '13, pp. 5:1–5:6, New York, NY, USA, 2013. ACM.