Modeling of Parachute Dynamics with GPU Enhanced Continuum Fabric Model and Front Tracking Method

A Dissertation Presented

by

Qiangqiang Shi

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

December 2014

Stony Brook University

The Graduate School

Qiangqiang Shi

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation.

Xiaolin Li - Dissertation Advisor
Professor, Department of Applied Mathematics and Statistics

James Glimm - Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

Xiangmin Jiao - Member
Associate Professor, Department of Applied Mathematics and Statistics

Foluso Ladeinde - Outside Member
Associate Professor, Department of Mechanical Engineering

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Modeling of Parachute Dynamics with GPU Enhanced Continuum Fabric Model and Front Tracking Method

by

Qiangqiang Shi

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2014

An advanced mesoscale spring-mass model is used to mimic fabric surface motion. The fabric surface is represented by a high-quality triangular surface mesh. Both the tensile stiffness and the angular stiffness of each spring are determined by the material's Young's modulus and Poisson ratio, as well as the geometrical characteristics of the surface mesh. The spring-mass system is a nonlinear Ordinary Differential Equation (ODE) system solved by fourth order Runge-Kutta method. The model is shown to be numerically convergent under the constraint that the summation of points masses is constant. Through coupling with an incompressible fluid solver and the front tracking method, the spring-mass model is applied to the simulation of the dynamic phenomenon of parachute inflation. Complex validation simulations conclude the effort via drag force comparisons with experiments.

Three applications of Graphics Processing Unit (GPU)-based algorithms for high performance computation of mathematical models were reported. Using one GPU device in the solving of the spring-mass system, we have achieved 6× speedup. In the second set of simulations, the system of one-dimensional gas dynamics equations is solved by the Weighted Essentially Non-Oscillatory (WENO) scheme; the GPU code is 7-20× faster than the pure CPU code. In the last case, a GPU enhanced numerical algorithm for American option pricing under the generalized hyperbolic distribution is studied. We have achieved 2× speedup for pricing single option and 400× speedup for multiple options.

Key Words: elastic membrane, spring model, parachute inflation, front tracking

To my Parents

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to my adviser, Professor Xiaolin Li, for suggesting this important and exciting thesis topic and for his advice, support, and guidance toward my Ph.D. degree. He taught me not only the way to do scientific research, but also the way to become a professional scientist. He is my adviser and a good friend.

I would like to thank Professors James Glimm, Xiangmin Jiao, Roman Samulyak who provided encouragement and valuable technical knowledge. I would also like to thank Professor Foluso Ladeinde for being on my dissertation committee. I would like to acknowledge the support of Department of Applied Mathematics and Statistics and its staff.

I would like to thank all my friends during my years of study as a graduate student at Stony Brook for their friendship and encouragement. I would like to mention Dr. JoungDong Kim, Dr. Yan Li, Dr. Yijing Hu, Yiyang Yang, Zheng Gao, Xiaolei Chen, Saurabh Joglekar, and Xiuzhu Ang.

Throughout my academic career, the constant support and unconditional love of my parents have always motivated me to strive forward. My dissertation is dedicated to them. Above all, I thank God for his love, grace, wisdom, favor, faithfulness, and protection.

Chapter 1

Introduction

1.1 Overview and Motivation

The analysis of the deformations and shape forming of canopy-like fabric structures such as parachutes, is nowadays an important scientific and technological topic, due to the wide range of applications of these structures. The mechanics of such structures can be approached in three ways: instrumentation and experimental studies, which is a development of the old craft methods of subjective evaluation; statistical correlations of test data, or the more recent methods of neural networks, fuzzy logic, and evolutionary algorithms, as described by Majumdar and Abhijit [59]; and applied mechanics modeling. The thinness and lack of bending stiffness of fabrics introduces a separation in scales of local and global curvature, contact interactions, and two-sided fluid-structure interactions. Due to these complexities, parachute modeling and simulation has traditionally been empirical in nature. However, advance of parachute in the 21st century requires a quantitative engineering approach to design instead of the age-old qualitative craft methods based on experience and trial-and-error. Realistic and accurate analysis requires sophisticated techniques in fluid-structure interaction (FSI) including computational fluid dynamics (CFD) and computational structure dynamics (CSD).

1.2 Parachute Inflation

In contrast to terminal descent, parachute inflation has a relatively short time duration, typically a few seconds. However, this short period regime is paramount to the effectiveness of the deceleration system and modeling it accurately with physics-based tools is difficult. Any malfunction, such as inversion, barber's pole, or jumper-in-tow, happens in this short stage could have serious effect on the fate of the personnel and cargo delivery.

Parachute inflation is a complex aeroelastic phenomenon that involves complex aerodynamics and elastic structures. The fact that the flow field and the canopy's geometric shape are interactive makes the inflation process a very difficult event to model [65]. Researchers have studied parachute inflation with different methods including empirical analysis [66], semi-numerical simulation [67], and through experiments [67, 68]. During the inflation sequence, the canopy not only experiences extremely increase in internal volume and large loading, but also involves geometric and material nonlinearities which make it a highly challenging event to model from a dynamic structures perspective [15]. Parachute inflation consists of a sequence of dynamically animated stages. These stages have been summarized in [102, 20]. For example, during the inflation of a circular parachute, the canopy starts as a vertical tube with an open lower end. With very little drag, air quickly rushes into the canopy tube due to the fast descending velocity of the system. Due to the limited volume inside the canopy, the pressure at the apex point increases dramatically leading to a large pressure difference between the internal and the external sides of the canopy. Meanwhile, the continued accumulation of air inside the canopy (inflation) results in the expansion both vertically and horizontally. The duration of the inflation process depends on the orientation of the parachute to the oncoming flow, altitude, and flight speed and ends when a sufficient amount of high pressure air fills the canopy. The aerodynamic forces that act in opposite directions along the parachute determine its steady-state shape and the final shape of the recirculating region (air bubble) inside [66]. In addition, the presence of the intense flow separation outside the canopy, the strong turbulence near the edge of parachute canopy and the narrow space inside the canopy before it is fully inflated result in computational challenges from the CFD perspective [101].

Simulation of parachute inflation *via* computational method has attracted attention of scientists at Laboratories and academic alike. In the using of the finite element method for the fluid and structure dynamics, Stein, Benney, *et al.*had made important contributions [82, 81, 78, 83, 79, 80]. Tezduyar *et al.*[92, 91, 86, 85, 93] had successfully addressed the computational challenges in handling the geometric complexities and the contact between parachutes in a cluster by applying the Deforming-Spatial-Domain/Stabilized Space-Time (DSD/SST) method. Using the immersed boundary method to study the semi-opened parachute in both two and three dimensions, Kim and Peskin *et al.*[49, 50], performed simulations with small Reynold number (about

300). Yu and Min [102] studied the transient aerodynamic characteristics of the parachute opening process. Karagiozis used the large-eddy simulation to study parachute in Mach 2 supersonic flow [45]. Purvis [70, 71] used springs to represent the structures of forebody, suspension lines, canopy, etc. In these papers, the authors used cylindrical coordinate with the center line as the axis. An algorithm called PURL to couple the structure dynamics (PRESTO) and fluid mechanics (CURL) was developed by Strickland *et al.*[84]. In this algorithm mass is added to each structure node based on the diagonally added mass matrix and a pseudo is computed from the fluid code which is the sum of the actual pressure and the pressure associated with the diagonally added mass. Tutt and Taylor [97, 95] used an Eulerian-Lagrangian penalty coupling algorithm and multi-material ALE capabilities with LS-DYNA to replicate the inflation of small round canopies in a water tunnel.

1.3 Fabric Dynamics

Simulation of fabric dynamics through computational method has applications in both computer graphics and engineering. Scientific applications include modeling of cell skin and soft tissues. The textile and fashion industry invites computer tools that can realistically generate the shape of a cloth dressing. Many authors have contributed to the modeling of cloth and fabric surface. A continuous model for deformable objects was proposed by Terzopoulos and Fleischer [90, 88, 89]. Using the Tchebychev net cloth model Aono *et al.*[2, 3] simulated a sheet of woven cloth composites. Due to its intuitiveness and simplicity particle method gained popularity in 1990's and

2000's. A particle-based model capable of being tuned to reproduce the static draping behavior of specific kinds of woven cloth was presented by Breen *et al.*in [8, 9]. Eberhardt *et al.*[26, 25] extended the model and introduced techniques to model measured force data exactly and thus cloth-specific properties. They also extended the particle system to model air resistance. Choi and Ko [12] also used particle spring model, on which our model is base on. On the physics based modeling, Platt and Barr[40] showed how to use mathematical constraint methods based on physics and on optimization theory to create controlled, realistic animation of physically-based flexible models. Carignan *et al.*[11] discussed the use of physics-based models for animating clothes on synthetic actors in motion. Provot [69] described a physics-based model for animating cloth objects, derived from elastically deformable models, and improved order to take into account the non-elastic properties of woven fabrics. Volino *et al.*[99] presented an efficient set of techniques that simulates any kind of deformable surface in various mechanical situations. Hsiao and Chen [39] used the spring model to draw the cloth pattern. They found that cloth shows different appearance with different values of the spring constant. Aileni *et al.*[1] applied the mass-spring model for a three dimensional simulation of apparel products using virtual mannequins. Their model supports different types of human bodies created in the virtual environment. Ji [41] used the mass-spring model to describe the dynamic draping behavior of the selected five types of fabric materials including woven and knitted fabrics. In his paper, the material properties are measured through the Kawabata Evaluation System (KES) [46].

We follow the general idea of the particle and spring mass method for the modeling and simulation of the cloth stretching and draping. Previous studies of Kim et al. [48] and Li et al. [57] discuss the application of the mesoscale model in effects to mimic the dynamic motion of a fabric surface while coupled with an incompressible fluid solver for parachute inflating and descending. The numerical simulations demonstrated good agreement with the experimental results both qualitatively and quantitatively, but questions remain on the validity of the model and its relation to the continuum model of the fabric surface as an elastic membrane. Recently, Delingette [19] proposed a revision of the spring-mass model that includes the angular deformation energy where the force at each triangle vertex includes spring forces through both its adjacent and opposite sides. Delingette demonstrated that with this modification, the force in the spring model is indeed related to the strain and stress of the elastic membrane. This work will demonstrate that the difference between the spring-mass model used in our previous studies and Delingette model can be bridged by a re-interpretation of the spring constant and by adding additional forces contributed by the opposite sides of the vertex. We also demonstrate that the spring-mass model is numerically convergent under the constraints that the total mass is conserved and that both the tensile stiffness and the angular stiffness of the spring conform with the material's Young's modulus and Poisson ratio. In addition, an algorithm to compute the von-Mises stress of a fabric surface under strain was also included. The objective of this work is employ the Delingette-modified spring-mass model in the simulation of parachute inflation.

1.4 Front Tracking Methods

Front Tracking is a numerical algorithm which assigns special degrees of freedom to a surface, moving dynamically through a background grid. This method, when coupled with underlying PDE solvers, can provide high resolution to the geometry and physical variables across the interface. Front Tracking is a Lagrangian interface method; it can deliver solutions superior to Eulerian methods, including the level set and the volume of fluid methods [21].

1.5 Dissertation Organization

In Chapter 2, we will introduce the mathematical models which include Navier-Stokes equations, three spring-mass models, and an algorithm to compute the von-Mises stress of a fabric surface under strain. Chapter 3 presents the numerical framework and the implementations of the above models. We will discuss some of the major extensions we have made to the *FronTier* functionalities and the coupling of the spring models with the incompressible fluid solver in *FronTier*. In Chapter 4, we will introduce the application of the Graphics Processing Unit (GPU) in the simulations. In Chapter 5, several benchmark test cases on the modeling of fabric surface through the front tracking method have been presented. We will give the verification of numerical convergence as well as the verification of Young's modulus and Poisson ratio of the spring system. In addition, the computation of the stress was discussed and the numerical solutions have been compared with the available experimental data for validation. Chapter 6 will summarize this work and discuss some of the on-going work and amendment to our current computational method for the parachute system.

Chapter 2

Mathematical Models

2.1 Navier-Stokes Equations

For personnel and cargo parachute, the speed of the surrounding flow is much smaller than the sound speed, therefore the use of incompressible Navier-Stokes equation is appropriate.

$$\rho \frac{D\mathbf{u}}{Dt} + \nabla p = \mu \nabla^2 \mathbf{u} + \rho \mathbf{g}, \tag{2.1}$$

where $\frac{D}{Dt} = \dfrac{\partial}{\partial t} + \mathbf{u} \cdot \nabla$ is the material derivative of the fluid. The incompressibility is given by Eq. (2.2)

$$\nabla \cdot \mathbf{u} = 0. \tag{2.2}$$

For the parachute system, this equation is solved through the projection method with special treatment at the canopy surface.

2.2 Spring-Mass Models

Using a spring-mass system to model a fabric surface has been explored by computer scientists and applied mathematicians. This spring system provides good model for the simulation of thin surfaces such as skin, soft tissue, paper and textile. It has also become a natural choice for the modeling of leaves and parachute canopy.

The non-fluid material in the parachute simulation is called the structure component whose motion is governed by the Newton's second law subject to certain internal constraints. Both the canopy surface and the string chords (or the risers) which connect the canopy and the payload are flexible structures and they too, are continuum systems. In many literature, the motion of the structure is described by the quasi-ordinary equation

$$\rho_i \frac{d^2\mathbf{x_i}}{dt^2} = \mathbf{f_i} - \nu \frac{d\mathbf{x_i}}{dt} + \nabla \cdot \sigma_{\mathbf{i}}, \tag{2.3}$$

where at position $\mathbf{x_i}$, ρ_i is the density, $\mathbf{f_i}$ is the external force density (for example, due to gravity and fluid pressure), $\sigma_{\mathbf{i}}$ is the stress tensor, and ν is the damping coefficient.

In general, all derivatives in Eq. (2.3) should be considered as partial derivatives. However, very few have attempted to solve Eq. (2.3) exactly. Like the fluid, discretization of Eq. (2.3) is also needed. We followed the work by Choi and Ko [12] and applied to the triangular mesh from the front tracking library. Although the basic idea is similar, there are several marked differences in our application.

2.2.1 Representation of Fabric Structure

In this work, the fabric surface is represented by a high-quality triangular surface mesh which is generated with no small or large angles. Fig. 2.1 illustrates the spring model on the triangulated mesh. In the model we used herein, we seek to approximate Eq. (2.3) through physical intuition, that is, we approximate each discretized element as a mass point while the stress tensor $\sigma_{\mathbf{i}}$ is approximated by a set of springs connecting to the neighboring points. The spring system has only the restoring force against stretching and compression, therefore it may not exactly describe the structure system, especially when the structure's stress tensor may include restoring force against bending and twisting. Since the structure involved in the parachute study contains only fabric surface and string chord, we believe such approximation is good enough and can capture the most important properties of the structure dynamics in the parachute system. The details of such system will be described in the following subsections.

2.2.2 Two Simplified Spring-Mass Models

When no external driving force is applied, the fabric surface is a conservative system whose total energy (kinetic energy and potential energy) is a constant. Assuming each mesh point represents a point mass m in the spring system with position vector $\mathbf{x}_i$, the kinetic energy of the point mass i is $T_i = \frac{1}{2}m|\dot{\mathbf{x}}_i|^2$, where $\dot{\mathbf{x}}_i$ is the time derivative, or velocity vector of the point mass i.

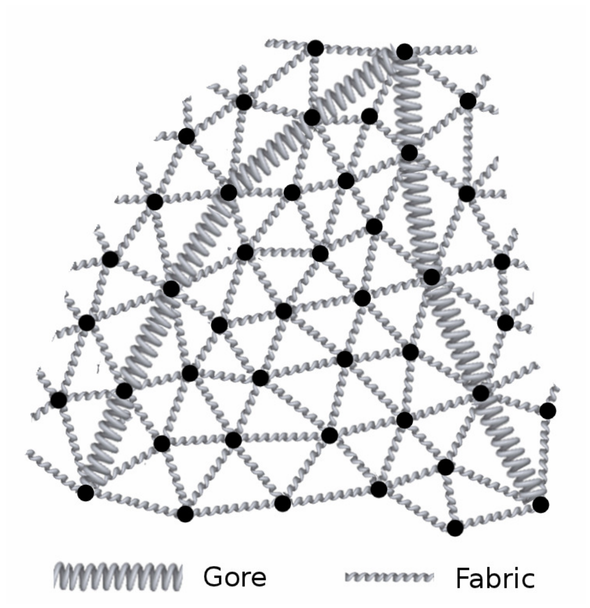

Figure 2.1: The spring model on a triangulated mesh. Each vertex point in the mesh represents a mass point with point mass m. Each edge of a triangle has a tensile stiffness. With the equilibrium lengths set during the initialization, the changing length of each side exerts a tensile spring force on the two neighboring vertices in opposite directions. Each angle of a triangle has an angular stiffness which is set during the initialization. An additional tensile force is generated when the the angle is changed. Gore boundaries are modeled by springs with higher tensile stiffness.

The first simplified spring-mass system, which we refer to as Model-I, has the potential energy between two point masses $\mathbf{x}_i$ and $\mathbf{x}_j$ in the form of

$$V_{ij} = \frac{k}{2} \left|(\mathbf{x}_i - \mathbf{x}_j) - (\mathbf{x}_{i0} - \mathbf{x}_{j0})\right|^2, \tag{2.4}$$

where k is the spring constant and $\mathbf{x}_{i0}$ is the equilibrium position of mass point

i. A spring system with potential energy Eq. (2.4) has pure oscillatory motion and its eigen frequencies have an upper bound $\sqrt{2Mk/m}$, where M is the maximum number of neighbors a mass point can have. This upper bound of eigen frequencies plays an important role in the analysis of numerical stability and accuracy for schemes to solve the system.

It is easy to analyze the upper bound of eigen frequencies of Model-I. However, Model-I contains strong bending force and is not suitable for fabric modeling since a fabric surface is considered as a membrane which is an idealized two dimensional manifold for which forces needed to bend it are negligible. For a realistic spring system to model the fabric surface, we have to assume that the spring force between two neighboring mass points is only proportional to the displacement from their equilibrium distance, the potential energy due to the relative displacement between two neighboring point mass $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by

$$V_{ij} = \frac{k}{2}(|\mathbf{x}_i - \mathbf{x}_j| - l_{ij}^0)^2, \tag{2.5}$$

where $l_{ij}^0 = |\mathbf{x}_{i0} - \mathbf{x}_{j0}|$ is the equilibrium length between the two point masses. We refer to this system model as Model-II.

The Lagrangians of the two systems are, therefore,

$$L = T - V = \sum_{i=1}^{N} \frac{1}{2} m |\dot{\mathbf{x}}_i|^2 - \frac{1}{4} \sum_{i=1}^{N} \sum_{j=1}^{N} k \left| (\mathbf{x}_i - \mathbf{x}_j) - (\mathbf{x}_{i0} - \mathbf{x}_{j0}) \right|^2 \eta_{ij} \tag{2.6}$$

for Model-I and

$$L = T - V = \sum_{i=1}^{N} \frac{1}{2} m |\dot{\mathbf{x}}_i|^2 - \frac{1}{4} \sum_{i=1}^{N} \sum_{j=1}^{N} k(|\mathbf{x}_i - \mathbf{x}_j| - l_{ij}^0)^2 \eta_{ij} \tag{2.7}$$

for Model-II, where η_{ij} is the adjacency coefficient between mass point i and j, and $\eta_{ij} = 1$ if mass points i and j are immediate neighbors, $\eta_{ij} = 0$ if $i = j$ or mass points i and j are not direct neighbors.

Applying the Lagrangian equation

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) = \frac{\partial L}{\partial q_i}$$

to each mass point at $\mathbf{x}_i$ where $q = (\mathbf{x}, t)$, we have

$$m \frac{d\dot{\mathbf{x}}_i}{dt} = m \frac{d^2 \mathbf{x}_i}{dt^2} = -\sum_{j=1}^{N} \eta_{ij} k \left((\mathbf{x}_i - \mathbf{x}_j) - (\mathbf{x}_{i0} - \mathbf{x}_{j0}) \right), \tag{2.8}$$

for Model-I and

$$m \frac{d\dot{\mathbf{x}}_i}{dt} = m \frac{d^2 \mathbf{x}_i}{dt^2} = -\sum_{j=1}^{N} \eta_{ij} k \left(\mathbf{x}_i - \mathbf{x}_j - l_{ij}^0 \mathbf{e}_{ij} \right), \tag{2.9}$$

for Model-II, where $\mathbf{e}_{ij} = \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$ is the unit vector from $\mathbf{x}_i$ to $\mathbf{x}_j$.

The difference between Model-I and the fabric model (Model-II) is that the latter has no bending energy, thus no restoring force in the direction normal to the surface.

2.2.3 An Advanced Spring-Mass Model

Model-II serves as a simplified mesoscale model which assumes the force required to bend the surface is negligible and the force to stretch the surface is proportional to the displacement from the equilibrium distance between adjacent mass points. Although this model can simulate the dynamic motion of a fabric surface and exert correct tension and wrinkling of the surface, there is a lack of proof on the relation between this model and the continuum model for an elastic membrane. Van Gelder argued [30] that the simple spring-mass model cannot be related to the continuum model for the linear elastic membrane and therefore not suitable to represent the fabric surface. However, recently, Delingette [19] proposed a revision of the spring-mass model that includes the angular deformation energy where the force at each triangle vertex includes spring forces through both its adjacent and opposite sides. The work of Delingette [19] details the modeling efforts of a nonlinear membrane using a spring-mass model with a triangulated mesh. This model is favored because of its relation to the elastic membrane model in continuum mechanics.

Illustrative features of Delingette's model are given in Fig. 2.2. The energy of membrane $W(T_{\mathbf{X}_0})$ that is required to deform a single triangle $T_{\mathbf{X}_0}$ with vertices $\{\mathbf{X}_{10}, \mathbf{X}_{20}, \mathbf{X}_{30}\}$ into its deformed position $T_{\mathbf{X}}$ with vertices $\{\mathbf{X}_{1,}, \mathbf{X}_2, \mathbf{X}_3\}$ consists of two parts,

- The energy of three tensile springs that prevent edges from stretching.
- The energy of three angular springs that prevent any change of vertex angles.

For a triangle in equilibrium $T_{\mathbf{X}_0}$, the initial states are given by area $A_{\mathbf{X}_0}$, angles α_i, and lengths l_i^0 $(i \in 1, 2, 3)$ in equilibrium while $A_{\mathbf{X}}$, β_i and l_i denote the area, angles, and lengths of the deformed triangle $T_{\mathbf{X}}$ respectively.

The edge elongation can be written as $dl_i = l_i - l_i^0$. The potential energy is given [19] as

$$W(T_{\mathbf{X}_0}) = \sum_{i=1}^{3} \frac{1}{2} k_i^{T_{\mathbf{X}_0}} (dl_i)^2 + \sum_{\substack{i=1 \\ j=(i+1)\ mod\ 3 \\ k=(i+2)\ mod\ 3}}^{3} \gamma_i^{T_{\mathbf{X}_0}} dl_j dl_k$$

where

$$k_i^{T_{\mathbf{X}_0}} = \frac{(l_i^0)^2 (2 \cot^2 \alpha_i (\lambda + \mu) + \mu)}{8 A_{\mathbf{X}_0}}$$

is the tensile stiffness and

$$\gamma_i^{T_{\mathbf{X}_0}} = \frac{l_j^0 l_k^0 (2 \cot \alpha_j \cot \alpha_k (\lambda + \mu) - \mu)}{8 A_{\mathbf{X}_0}} \tag{2.10}$$

is the angular stiffness, where $j = (i+1)\ mod\ 3$ and $k = (i+2)\ mod\ 3$. λ and μ are the Lamé coefficients of the material. These coefficients are simply related to the two physically meaningful parameters defined in planar elasticity for a membrane, that is, Young's modulus E and the Poisson ratio ν [33]:

$$\lambda = \frac{E\nu}{1 - \nu^2} \quad \text{and} \quad \mu = \frac{E(1 - \nu)}{1 - \nu^2}.$$

Young's modulus quantifies the stiffness of the material, whereas the Poisson ratio characterizes the material compressibility.

Through the application of Rayleigh-Ritz analysis the fabric surface, rep-

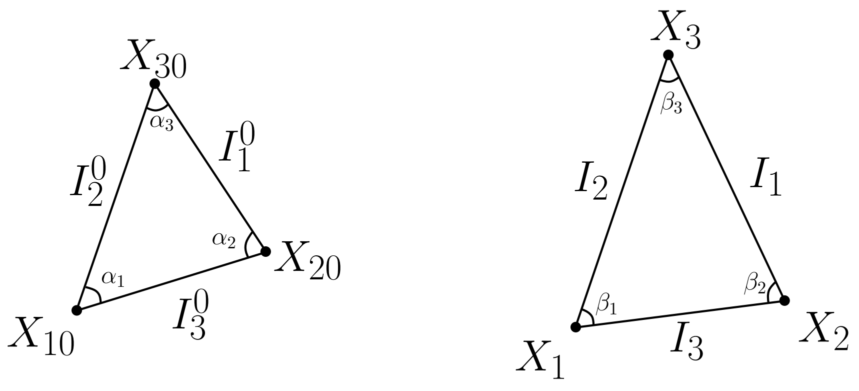

Figure 2.2: (a) Rest triangle $T_{\mathbf{X}_0}$ whose vertices are $\mathbf{X}_{i0}$. (b) Deformed triangle $T_{\mathbf{X}}$ whose vertices are $\mathbf{X}_i$.

resented by the triangular mesh, should evolve by minimizing its membrane energy. Therefore, along the opposite derivative of that energy with respect to the nodes of the system, that is, the deformed positions $\mathbf{X}_i$:

$$\begin{aligned}\mathbf{F}_i(T_{\mathbf{X}_0}) &= -\frac{\partial W(T_{\mathbf{X}_0})}{\partial \mathbf{X}_i} \\ &= \sum_{j\neq i} k_j^{T_{\mathbf{X}_0}}(dl_j)\frac{\mathbf{X}_k - \mathbf{X}_i}{l_j} + \sum_{j\neq i}(\gamma_k^{T_{\mathbf{X}_0}} dl_i + \gamma_i^{T_{\mathbf{X}_0}} dl_k)\frac{\mathbf{X}_k - \mathbf{X}_i}{l_j} \quad (2.11)\end{aligned}$$

The membrane deformation energy of the whole triangulation is the sum of the energy of each triangle. Thus, we obtain the force at each vertex point as Eq. (2.12).

$$\mathbf{F}_i = \sum_{j=1}^{N} \eta_{ij}\mathbf{F}_{ij} \quad (2.12)$$

As illustrated by Fig. 2.3, if $\mathbf{X}_i$ and $\mathbf{X}_j$ are shared by two triangles T_1 and T_2, and the other vertices of triangles T_1 and T_2 as denoted as $\mathbf{X}_m$ and

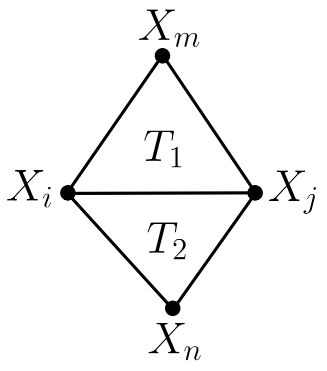

Figure 2.3: Triangles T_1 and T_2 share $\mathbf{X}_i$ and $\mathbf{X}_j$, the other vertices of triangles T_1 and T_2 are $\mathbf{X}_m$ and $\mathbf{X}_n$ respectively.

$\mathbf{X}_n$ respectively, we have

$$\begin{aligned}\mathbf{F}_{ij} &= ((k_{ij}^{T_1} + k_{ij}^{T_2})dl_{ij} + (\gamma_i^{T_1} dl_{im} + \gamma_j^{T_1} dl_{jm} + \gamma_i^{T_2} dl_{in} + \gamma_j^{T_2} dl_{jn}))\mathbf{e}_{ij} \\ &= \tilde{k}_{ij} dl_{ij} \mathbf{e}_{ij} + \tilde{\gamma}_{ij} dl_{ij} \mathbf{e}_{ij} \qquad (2.13)\end{aligned}$$

where $\tilde{k}_{ij} = k_{ij}^{T_1} + k_{ij}^{T_2}$, $\tilde{\gamma}_{ij} = (\gamma_i^{T_1} dl_{im} + \gamma_j^{T_1} dl_{jm} + \gamma_i^{T_2} dl_{in} + \gamma_j^{T_2} dl_{jn})/dl_{ij}$ and $\mathbf{e}_{ij}$ is the unit vector from $\mathbf{X}_i$ to $\mathbf{X}_j$.

If the second term in Eq. (2.13) is neglected, Delingette's model is same as Model-II except that the spring constant varies if the corresponding initial triangles deviate from isosceles triangles. Numerical evidence suggests that both the variation of $\tilde{k}_{ij}$ and the modification from the second term (due to angular stiffness) are significant in the simulations.

2.3 Stress Analysis

The stress distribution of the parachute canopy fabric and gores during the process of inflation is valuable information prior to the real-world test and evaluation process. The natural stresses are normal stresses directed parallel to the triangle sides and the natural stress of the triangle in the spring-mass model is due to the stretching of each triangle side. Let τ_1, τ_2, τ_3 be the natural stress on side $1, 2, 3$ of a triangle, we have

$$\tau_i = k(l_i - l_i^0), \qquad i = 1, 2, 3, \tag{2.14}$$

where k is the spring constant, l_i^0 is the equilibrium length of side i and l_i is the stretched length of the side i. This natural stress can be converted [33] into the stress in Cartesian coordinates on the plane of the triangle. The Cartesian stress is a 2×2 tensor in the plane of the triangle

$$\sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix}. \tag{2.15}$$

The conversion from natural stress to Cartesian stress is accomplished through a mapping matrix, that is

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = \begin{pmatrix} c_1^2 & s_1^2 & s_1c_1 \\ c_2^2 & s_2^2 & s_2c_2 \\ c_3^2 & s_3^2 & s_3c_3 \end{pmatrix}^{-1} \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} \tag{2.16}$$

where $c_i = \cos\theta_i = dx_i/l_i$ and $s_i = \sin\theta_i = dy_i/l_i$ are the trigonometric functions of the angle of each side with respect to the $x-$axis. The stresses in each of the two principal directions are obtained via diagonalization of the stress tensor, that is

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} = T^{-1} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} T, \tag{2.17}$$

where σ_1 and σ_2 are the solutions of the characteristic equation

$$\begin{vmatrix} \sigma_{xx} - \sigma_{1,2} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} - \sigma_{1,2} \end{vmatrix} = 0. \tag{2.18}$$

The von-Mises stress is commonly used to evaluate safety factors of material and is given by Eq. (2.19). So called, safety factors, are defined as the ratio of the significant strength of the material to the von-Mises stress observed under the application of interest. When this factor is observed to be a critical value (that corresponds to a high stresses application of the fabric surface or gores) the parachute construction or material may need to be modified or a more appropriate model used for that particular application.

$$\sigma_{vm} = \sqrt{\sigma_1^2 + \sigma_2^2 - \sigma_1\sigma_2} \tag{2.19}$$

Chapter 3

Numerical Methods and the Implementations

The computational procedures for the parachute system are demonstrated by the flow-chart in Fig. 3.1. The numerical method we have used in this paper for the simulation of parachute system contains several components. The data structures and many functionalities are based on the *FronTier* library developed for the front tracking method. The parachute canopy is modeled by the spring-mass system on a homogeneously triangulated mesh while the string chords connecting the payload. A finite damping coefficient is added to both the canopy and the riser spring chains in order to dissipate the kinetic energy of oscillatory motion. We solve the Navier-Stokes equations using the projection scheme [13, 10] and couple the fluid equation with the canopy surface through the impulse method. We briefly discuss the components of the computational framework in the following sections.

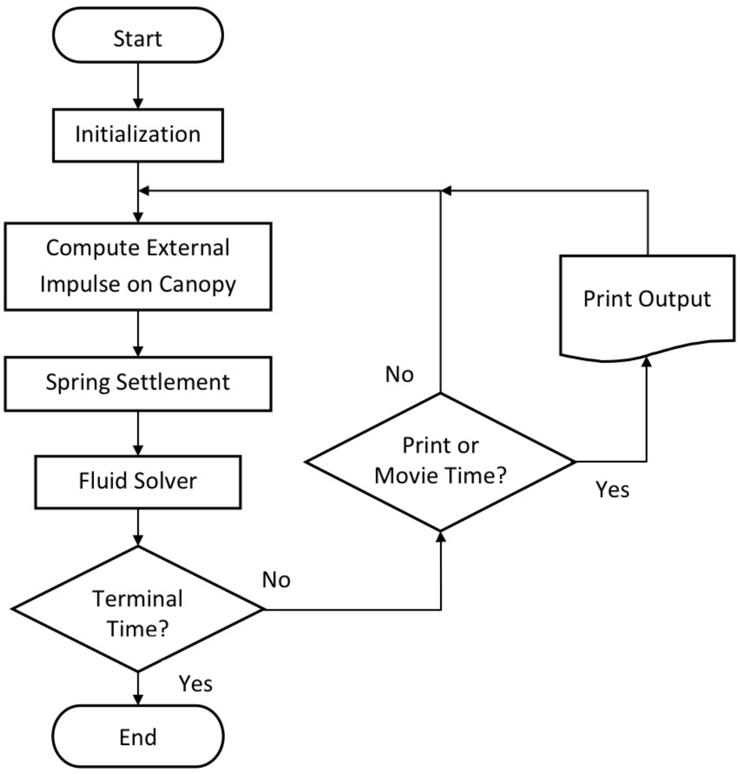

Figure 3.1: The flow chart of the computation for the parachute system using *FronTier* as the base library. The spring system is a new module built on the *FronTier* data structure and functions. The system is coupled with a high order incompressible solver for the Navier-Stokes equation.

3.1 Numerical Methods for the Fluid Dynamics

Most of the components in the numerical model are formally in at least the second order, but due to its geometrical complexity, the overall system cannot achieve global second order due to splitting computation over each system.

We improved the accuracy of the fluid solver by adapting the staggered grid algorithm in the projection step [87, 61, 55, 56].

3.1.1 Projection Method

Projection method is an effective means of numerically solving time-dependent incompressible fluid flow problems. The advantage of the projection method is that the computation of velocity and the pressure fields are decoupled. Some approximation to the momentum Eq. (2.1) is used to determine the velocity u or a provisional velocity, and then an elliptic equation is solved that enforces the divergence constraint Eq. (2.2) and determines the pressure. In some variations, the viscous term in Eq. (2.1) is advanced in a separate step from the advection terms. The original projection method is that the velocity field is forced to satisfy a discrete divergence constraint at the end of each time step. Projection methods which enforce a discrete divergence constraint, or exact projection methods, have often been replaced with approximate projection methods. Approximate projection methods are used because of observed weak instabilities in exact methods and the desire to use more complicated or adaptive finite difference meshes on which exact projections are difficult or mathematically impossible to implement. Additionally, as with all fractional step methods, a crucial issue is how boundary conditions are determined for some or all of the intermediate variables.

Projection method pioneered by Chorin [13, 14] for numerically integrating Eqs. (2.1) and (2.2) is based on the observation that the left-hand side of equation Eq. (2.1) is a Hodge decomposition. Hence an equivalent projection formulation is given by

$$u_t = P[-(u \cdot \nabla)u + \nu \nabla^2 u], \tag{3.1}$$

where P is the operator which projects a vector field onto the space of divergence-free vector fields with appropriate boundary conditions. In the 1980s, several papers appeared in which second-order accurate versions of the projection method were proposed. Those of Goda [38], Bell *et al.*[5], Kim and Moin [47], and Van Kan [98] are motivated by the second-order, time-discrete semi-implicit forms of Eqs. (2.1) and (2.2),

$$\frac{u^{n+1} - u^n}{\Delta t} + \nabla p^{n+1/2} = -[(u \cdot \nabla)u]^{n+1/2} + \frac{\nu}{2}\nabla^2(u^{n+1} + u^n) \tag{3.2}$$

$$\nabla \cdot u^{n+1} = 0, \tag{3.3}$$

where $[(u \cdot \nabla)u]^{n+1/2}$ represents a second-order approximation to the convective derivative term at time level $t^{n+1/2}$ which is usually computed explicitly. Spatially discretized versions of the coupled Eqs. (3.2) and (3.3) are cumbersome to solve directly. Therefore, a fractional step procedure can be used to approximate the solution of the coupled system by first solving an analog to Eq. (3.2) for an intermediate quantity u^*, and then projecting this quantity onto the space of divergence-free fields to yield u^{n+1}. In general this procedure is given by

Step 1: Solve for the intermediate field u^*

$$\frac{u^* - u^n}{\Delta t} + \nabla q = -[(u \cdot \nabla)u]^{n+1/2} + \frac{\nu}{2}\nabla^2(u^* + u^n), \tag{3.4}$$

$$B(u^*) = 0, \tag{3.5}$$

where q represents an approximation to $p^{n+1/2}$ and $B(u^*)$ a boundary condition for u^* which must be specified as part of the method.

Step 2: Perform the projection

$$u^* = u^{n+1} + \Delta t \nabla \phi^{n+1} \tag{3.6}$$

$$\nabla \cdot u^{n+1} = 0, \tag{3.7}$$

using boundary conditions consistent with $B(u^*) = 0$ and $u^{n+1}|_{\partial\Omega} = u_b^{n+1}$.

Step 3: Update the pressure

$$p^{n+1/2} = q + L(\phi^{n+1}), \tag{3.8}$$

where the function L represents the dependence of $p^{n+1/2}$ on ϕ^{n+1}. Once the time step is completed, the predicted velocity u^* is discarded, not to be used again at that or later time steps. There are three choices that need to be made in the design of such a method. They are the pressure approximation q, the boundary condition $B(u^*)$, and the function $L(\phi^{n+1})$ in the pressure-update equation. An important issue is that the boundary condition for u^* must be consistent with Eq. (3.6) although at the time the boundary conditions are applied the function ϕ^{n+1} is not yet known and hence must be approximated.

In the first step of the method, if q is a good approximation to $p^{n+1/2}$, the field u^* may not differ significantly from the fluid velocity and thus a reasonable choice for the boundary condition $B(u^*) = 0$ may be $(u^* - u_b)|_{\partial\Omega} = 0$. On the other hand, one may not be interested in computing the pressure at every time step and would like to choose $q = 0$ and obviate the third step in the method. In this case u^* may differ significantly from the fluid velocity, requiring the boundary condition $B(u^*)$ to include a nontrivial approximation of $\nabla\phi^{n+1}$ in Eq. (3.6). Regarding the third step of the method, substituting Eq. (3.6) into Eq. (3.4), eliminating u^*, and comparing with Eq. (3.2) yield a formula for the pressure-update

$$p^{n+1/2} = q + \phi^{n+1} - \frac{\nu\Delta t}{2}\nabla^2\phi^{n+1}. \tag{3.9}$$

The last term of this equation plays an important role in computing the correct pressure gradient and allows the pressure to retain second-order accuracy up to the boundary. Without this term, the pressure gradient may have zeroth-order accuracy at the boundary even if the pressure itself is high-order accurate.

3.1.2 Fluid Solver for the Advection Term

The calculation of advection of the velocity field is based on the fifth order finite difference WENO schemes by Jiang and Shu[42] with a general framework for the design of smoothness indicators and nonlinear weights. A key component of the WENO scheme is the linear combination or reconstruction of lower order fluxes to obtain a higher order approximation. The WENO schemes use the idea of adaptive stencils to automatically achieve high order accuracy and non-oscillatory property near discontinuities. In the system case, WENO scheme is based on local characteristic decomposition and flux splitting to avoid spurious oscillation. The time discretization of WENO schemes is implemented by a class of high order TVD Runge-Kutta methods. Assuming $\mathcal{L}(\mathbf{u})$ is a discretization of the spatial advection operator, the third-order TVD Runge-Kutta is

$$\begin{aligned}
\mathbf{u}^{(1)} &= \mathbf{u}^n + \Delta t \cdot \mathcal{L}(\mathbf{u}^n) \\
\mathbf{u}^{(2)} &= \frac{3}{4}\mathbf{u}^n + \frac{1}{4}\mathbf{u}^{(1)} + \frac{1}{4}\Delta t \cdot \mathcal{L}(\mathbf{u}^{(1)}) \\
\mathbf{u}^{n+1} &= \frac{1}{3}\mathbf{u}^n + \frac{2}{3}\mathbf{u}^{(2)} + \frac{2}{3}\Delta t \cdot \mathcal{L}(\mathbf{u}^{(2)}).
\end{aligned}$$

The advection equation is a scalar equation but has both linear and nonlinear flux functions (flux of Burgers equation). Using the flux version of the WENO scheme, this can be computed robustly. This is followed by a second order Crank-Nicolson solver for the diffusion (viscous) term. The fluid pressure is a derived variable from the elimination of velocity divergence at each time step. Only pressure gradient or pressure difference (across canopy) is used to calculate the force which the fluid interacts with the structure (canopy).

3.2 Numerical Methods for the Structure Dynamics

The resulting equation for the spring mass model is an ODE system. To accurately and efficiently solve this system, it is important to understand the characteristic motion of the mass points. In particular, we need to understand the eigen frequencies of the oscillatory modes and estimate the upper bound of the eigen frequencies. The choice of numerical scheme and the criterion for choosing time step will affect the stability and accuracy of the solution. Previous analysis proved that the spring-mass system without external force is a conservative system and the motion of point mass in its tangential direction to the surface (or string on 2D) is purely oscillatory, and that there exists an upper bound for the eigen frequency of the oscillation

$$\omega_{\max} \leq \sqrt{\frac{2Mk}{m}}, \tag{3.10}$$

where k is the spring constant, m is the point mass, and M is the maximum number of neighbors a vertex point in the spring mesh can have. Using the fourth order Runge-Kutta method, we showed that the energy amplification factor per time step is $\xi^2 \sim 1 - \frac{1}{72}(\omega_{\max}\Delta t)^6$, where Δt is the time step. Therefore to ensure stability and accuracy, $\omega_{\max}\Delta t < 0.1$ is chosen for simulations discussed herein. The modification of the spring model due to Delingette adds variation to the spring constant, but the general principle of the upper bound is still valid except that the maximum spring constant in Eq. (3.10) should be substituted in for the determination of the upper bounds. This claim is numerically verified in all the simulations.

3.3 The Implementations in the *FronTier* Library

To correctly model the parachute system, an accurate coupling between the Navier-Stokes equation and the structure dynamics must be carefully considered near the canopy surface. The method we designed for the simulations in this paper uses the superposition of impulse on every mass point. Each mass point in the spring system acts as an elastic boundary point and exerts an impulse to the fluid in its normal direction. Our algorithm ensures that the action and reaction between the spring mass point and the fluid solver are equal in magnitude and opposite in directions, a requirement of Newton's third law.

3.3.1 The Extensions of the *FronTier* Library

Front tracking method treats a hyper-surface as a topologically linked set of marker points. The front tracking library contains data structure and functionalities to optimize and resolve the hyper-surface mesh with topological consistency. This method has been used for the simulations of fluid interface instabilities [24, 23, 32, 31], diesel jet droplet formation [7], and plasma pellet injection process [73, 74]. In these problems, the hyper-surface is used to model the interior discontinuities of materials and such manifold surface may undergo complicated changes in geometry and topology. The modeling of fabric surface is simpler in topological handling due to the fact that a fabric surface cannot bifurcate. However the fabric system has certain constraints and poses new challenges to the front tracking data structure and some associated functionalities.

Saving of the Initial Conditions

Since there is always a finite elasticity of a fabric material, to approximate the fabric surface as a highly stiffened spring system is not only convenient, but also realistic. However, the calculating of the force on each spring always need the equilibrium lengths of the related springs and the initial sizes of related angles. This requirement prompts us to add an equilibrium state of the mesh and treat the marker points of the hyper-surface as a set of spring vertices. This equilibrium lengths of the triangles' sides and the initial size of the triangles' angles are computed after the mesh optimization and stored in memory throughout the computation.

Collision Handling

Front tracking method also relies on the index of the side to check the topological consistency of the interface. For fluid interface instability problems, collision and contact of surfaces are resolved by merging or bifurcation. However, fabric surface can neither merge nor bifurcate, therefore we need to have functions which can carefully deal with the repulsive contact of structure surfaces, especially when the fabric surface is folded. Parachute collision/-contact is handled through the standard *FronTier* library with major revisions such as functions to handle the non-manifold surface and three dimensional curves (not as boundary of a surface, such as the spring chords). Fig. 3.2 shows the difference of collision handling between the fluid-fluid interface and the fabric-fabric surface.

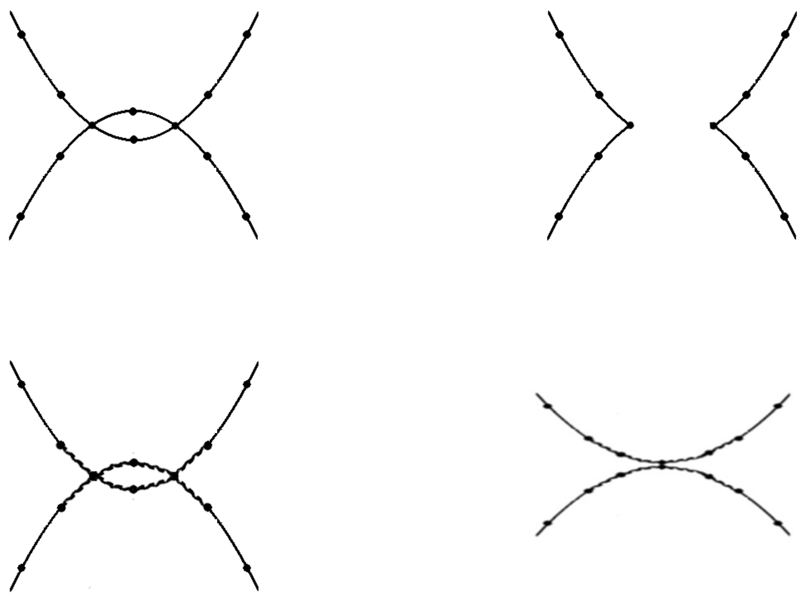

Figure 3.2: The difference between the collision handling for fluid interface and the fabric surface. The upper two plots show the topological bifurcation of fluid interface. The lower two plots show the repulsion of the fabric collision.

Global Indexing of Surface Mesh

Global indexing is a new feature that added to the computational framework. The original *FronTier* [22] had to deal with frequent surface mesh optimization and topological reconstructions. This makes the parallelization based on the matching of global index very difficult. As a result, the original *FronT ier* library relied on floating point matching for parallel communication. The floating point matching is not completely reliable therefore more complicated algorithms were implemented as reinforcement. However for fabric-like surface, especially when a spring-mass model is used, the inter-connectivity and proximity of the interface marker points should not be changed. Therefore, global indexing is ideal for the parallel communication of the surface information as employed in this work.

Local Index Coating Algorithm

Front tracking method has been mostly used for the study of fluid interface problems such as the Rayleigh-Taylor instability [36, 34], Richtmyer-Meshkov instability [29], and the jet problem [37, 35]. In these problems, the fluid interface is topologically a manifold, that is, the two sides of each surface are the boundaries of separate subdomains. Many front tracking functions are based on this assumption. However, in the parachute system, the canopy surface has an open boundary. In general, a space point with a finite distance away from the canopy surface cannot distinguish to which side of the canopy it belongs. But we may still assign the side to which a point belongs if the point is sufficiently close to the canopy surface.

The local side information of a space point close to canopy surface plays an important role in the calculation of pressure difference, and thus the drag force of the air to the canopy. The pressure in one side at the canopy surface is not continuous and therefore should be interpolated and computed using the value from its own side. This is realized by "painting" the grid cells using the so-called locally mesh coating algorithm.

Assuming the domain in which the canopy surface is immersed is indexed by l, the local index coating algorithm follows three steps:

(1). For any grid point P with a distance $d \leq h$ away from the surface, where h is the grid spacing, find the nearest point on the interface (*FronTier* is well equipped with these geometry functions) P_s. Using the sign of the scalar product $\overline{PP_s} \cdot \mathbf{n_s}$, we can determine the side of the point. Reassign domain index to $l-1$ if the point P is on the negative side, otherwise reassign the index to $l+1$.

(2). Reassign any grid point adjacent to a point indexed $l-1$ to the same; reassign any grid point adjacent to a point indexed $l+1$ to the same. Repeat this assignment for three sweeps. The resulting landscape of the grid indices will look like Fig. 3.3.

(3). The interpolation of side-sensitive variables will use grid points of the same locally coated index.

Modularized Functionalities

Modularization is emphasized in our code development. The parachute module is an independent application program. This new module consists of four components:

(1). the initialization module,

(2). the ODE module for the spring system,

(3). the PDE module for fluid dynamics,

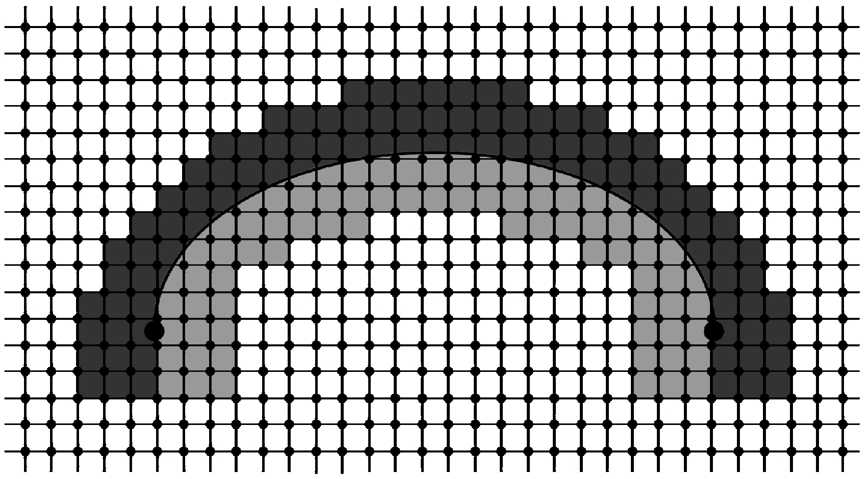

Figure 3.3: The parachute canopy is an open surface and cannot separate the space into subdomains. But we can still coat different indices for mesh cells close to the surface using the local geometrical information. The light and dark shaded polygons represent the sets of mesh cells on the positive and negative sides of the canopy respectively. An interpolation is carried out on vertices of the same color.

(4). the *FronTier* library for the interface geometry handling.

3.3.2 The Fluid-Canopy Coupling

For incompressible Navier-Stokes equation, it is the boundary condition that controls the dynamics of the solution. For the parachute problem, the boundary condition consists of two parts, the external boundary and the two interior sides of the canopy surface. In our computation, we have three different types of external boundary conditions, the preset Dirichlet boundary, the flow-through boundary, and the periodic boundary. The periodic boundary does not need special treatment and always follows the order of the numerical scheme. The Dirichlet boundary for the hyperbolic and parabolic (advection-diffusion) part of the numerical scheme is also relatively easy to specify. Normally the preset Dirichlet boundary is on the upwind side while the flow-through boundary is on the downwind side. The only approximation we made is to assume the downwind side of flow-through boundary is a constant extrapolation. This is needed for the parabolic equation. To compute the projection, we have used Neumann boundary for the preset Dirichlet boundary and the constant pressure (or ϕ) for the flow-through boundary. Since only the pressure gradient is physically significant, without losing generality, we set the ϕ at the flow-through boundary to zero. The interaction between fluid and the canopy is the most crucial part of the algorithm for the parachute simulation. We noted that the immersed boundary method by Kim and Peskin [49, 50] can only carry payload up to a few grams. This is not in the realistic range of parachute payload. The T-10 personnel parachute and the G-11 cargo parachute both carry payload ranging from hundreds to thousands of kilograms. Aside from the fact that Kim and Peskin's simulations are in fluid with small Reynold number, we also think that a more accurate modeling between the air flow and the canopy surface should be considered.

The system described by Eq. (2.8) conserves the total energy. However in dynamic simulation of the fabric surface, the total energy may increase and the system can be excited due to stretching and compression by external forces. The external force not only adds to the acceleration of the macro-scale motion of the fabric surface, it also displaces the mass points in the tangential directions and incites internal energy for the spring system. The restoring force between each pair of neighboring mass points of the spring system serves as the self-adjustment to satisfy the fabric constraint and Young's modulus. However, when the internal energy of the spring system is too high, the system may be dominated by the random meso-scale motion of the mass points. Therefore adding a damping force will help to stabilize the system. When there is an external velocity field $\mathbf{v}^e$, we define the external impulse as $\mathbf{I}_i^e = m\mathbf{v}_i^e$, where $\mathbf{v}_i^e$ is the external driving velocity at point $\mathbf{x}_i$. At any given time, we can solve the equations of the spring system and obtain the internal impulse $\mathbf{I}_i^s$. Since the spring force is a function of the relative position of the point mass with respect to its neighbors, we can use the superposition principle and add to the total impulse

$$\mathbf{I}_i = \mathbf{I}_i^e + \mathbf{I}_i^s. \tag{3.11}$$

Our method is to apply damping to the internal impulse only.

Physically, the canopy experiences three forces, the gravitational force due to the weight of the fabric, the lift force due to the pressure difference between the two sides of the canopy, and the internal force, which in our model is the spring restoring force and the friction force (to prevent the spring system becoming over-excited). The gravitational force of the payload is propagated through the spring system from the string chord to the boundary of the canopy, and then spread to each mass point of the canopy through the elastic sides of simplices. Although the interaction between the fluid and canopy has the participation of both the external and the internal forces, for each mass point in the spring system, we can still divide the impulse into three components, the gravitational impulse, the fluid impulse due the pressure difference between the two sides of the canopy, and the internal impulse due to its neighboring mass points in the spring system, that is

$$\mathbf{I}_i^c = \mathbf{I}_{gi}^c + \mathbf{I}_{pi}^c + \mathbf{I}_{si}^c \tag{3.12}$$

Our current model has not considered the fluid interaction with the mass point of the string chord and the payload. Therefore for these mass points, the impulse is

$$\mathbf{I}_i^s = \mathbf{I}_{gi}^s + \mathbf{I}_{si}^s. \tag{3.13}$$

In our method, the external impulse (due to gravity and pressure) is time integrated for each mass point, that is

$$\mathbf{I}_{gi} = \int_0^t m\mathbf{g}dt \tag{3.14}$$

for both canopy and string chord mass points and

$$\mathbf{I}_{pi} = \int_0^t \sigma(p^- - p^+)\mathbf{n}dt \tag{3.15}$$

for canopy points only, where p^- and p^+ are the pressure on lower and upper sides of the canopy, σ is the mass density of canopy per unit area, and $\mathbf{n}$ is the unit normal vector pointing from lower to upper side of the canopy. We would like to emphasize that the current calculation of fluid impulse has considered only the pressure in the normal direction, we have followed Kalro and Tezduyar [44] for using this simplification. A more accurate fluid interaction should include the velocity shear near the canopy surface and the stress to the surface. We will put this as future improvement in the new papers.

The internal impulse for both canopy and string chord mass points are solved by the damping spring system. The impulse due to payload force is propagated through the string chords to the edge points of the canopy surface.

The interaction between the canopy and fluid is through the normal velocity component of each mass point on the canopy. At every time step, the fluid exerts an impulse to the mass points, but this part of the impulse is balanced by the gravitational impulse and the restoring force of the spring. The normal component of the superposition of the three forces feeds back to the fluid in the following step. The result is that the momentum exchange between the canopy and the fluid is equal in magnitude and opposite in directions, a requirement by Newton's third law.

To prevent the spring system getting into over-excited state, we add a damping force to the system. Therefore, the complete system of equations is the following

$$\begin{aligned} \frac{d\mathbf{v}_i}{dt} &= -\frac{1}{m}\sum_{j=1}^{N}\eta_{ij}k\left(|\mathbf{x}_i-\mathbf{x}_j|-l_{ij}^0\right)\mathbf{e}_{ij}+\mathbf{f}_i^e-\kappa\mathbf{v}_i^s, \\ \frac{d\mathbf{x}_i}{dt} &= \mathbf{v}_i, \end{aligned}$$

where $\mathbf{f}_i^e$ is the external force, κ is the damping coefficient and $\mathbf{v}_i^s$ is the velocity component due to the spring impulse $\mathbf{v}_i^s = \mathbf{I}_i^s/m$.

We have studied two ways to implement the reacting impulse which the canopy exerts on the fluid (or vice versa). The first method is through the immersed boundary method which treats the reaction of the canopy as the normal force approximated by the smoothed delta function. The second method is to use the canopy as an internal moving boundary with a normal velocity (after spring settlement) of the canopy as the boundary value.

In the first method, we compute the increment of the impulse at each point on the canopy. We have followed Peskin's delta function method, that is, let

$$\mathbf{f}(\mathbf{x}, t) = \int \mathbf{F}(s, t)\delta(\mathbf{x} - \mathbf{X}(s, t))ds. \tag{3.16}$$

The difference between our method and Kim and Peskin's method lies in the calculation of $\mathbf{F}$. Instead of computing the tension through the derivative with respect to the arc length, we use the impulse of the mass point as a result of the superposition of three forces from the spring system, that is

$$\mathbf{F}(\mathbf{x}_i, i) = d(\mathbf{I}_g + \mathbf{I}_p + \mathbf{I}_s)/dt \tag{3.17}$$

Eq. (3.17) is more physically realistic, especially because $\mathbf{I}_s$ is solved from the spring equations. In the canopy spring system, we have observed that the tension is high at the top of the canopy where the curvature is almost zero.

The numerical implementation of Eq. (3.17) is straight forward after we have obtained the solution from the ODEs of the spring system. The surface force is the product of normal component of the acceleration and mass density at the canopy surface

$$\mathbf{F}(s, t) = \rho_c(\mathbf{a} \cdot \mathbf{n})\mathbf{n}, \tag{3.18}$$

where $\mathbf{n}$ is the unit normal vector on the canopy surface, and ρ_c is the mass density of canopy per unit area. In the second method we treat the canopy as a moving boundary rather than an immersed interface. Instead of using the normal component of the acceleration as a singular force, we solve a Dirichlet boundary problem on each side of the canopy, we use the normal velocity computed from the ODEs of the spring system. Since the acceleration is the derivative of velocity in a time step, the two methods are physically consistent but with different truncation errors.

3.3.3 Modeling of Parachute Gores

Most parachute canopies are made in patches called gores. Parachute gores are stitched by the reinforcement cables. The reinforcement cables are important structures which can have substantial impact on the parachute's interaction with the surrounding fluid. The gore structures in the parachute system can also affect the stability of the parachute motion. To accurately model the aerodynamic motion of a parachute, a mathematical model which reveals the geometry as well as the material strength of the canopy surface and the gore stitches must be employed. In the present model, the reinforcement cables are treated as curves embedded in the canopy surface. Young's modulus for the fabric surface and the reinforcement cable differ by assigning different spring constants to the surface mesh and to the gore curves. The insertion of reinforced gore boundaries as stiffened interior curves in the canopy surface mesh is demonstrated by Fig. 2.1. Fig. 3.4 demonstrates that when fully inflated, the spring system reveals both the patches and the indentation of the reinforcement cables.

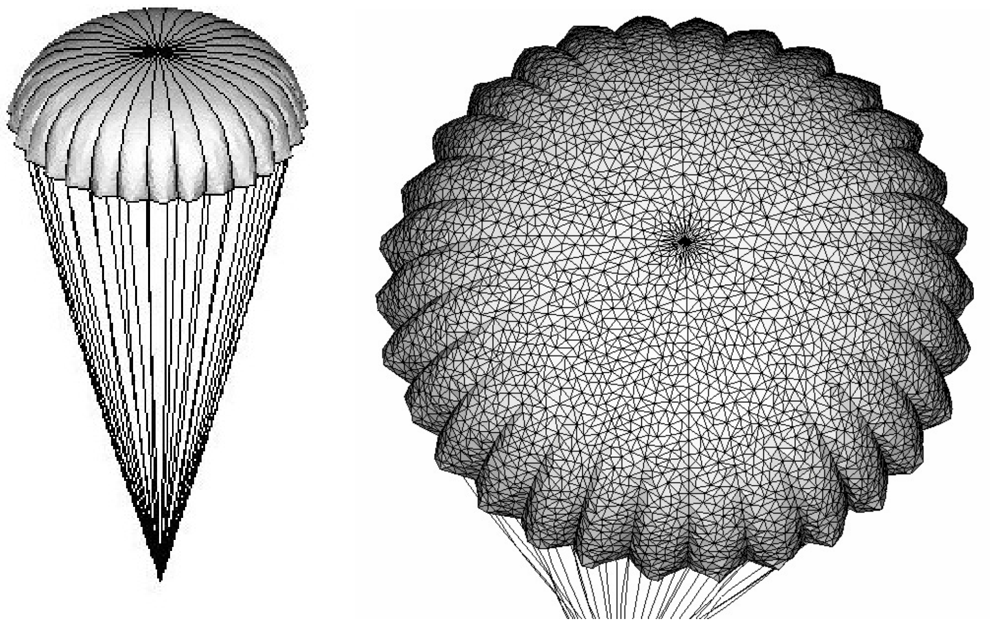
Figure 3.4: The reinforcement cables for the gores, are modeled using the same spring system, but with different spring constant along the gore seams or curved edges. The left plot shows fully opened canopy with expanded gores. After the inflation, the gore structure is clearly revealed. The right plot shows the surface mesh and the details of the gore boundary curves.

3.3.4 Modeling of Canopy Porosity

It has long been known that permeability is an important material property that influences canopy performance; it is paramount to accurately capturing fabric dynamics, total drag, and drift distance. A finite permeability will make the parachute much more stable while still maintain the balance of the payload. The permeability of the parachute material often plays a vital role in the parachute design. A state-of-art tuning from an impervious material to a highly permeable fabric can make a parachute from a wandering sloth into a plummeting stabilizer. We have considered three different implementations of porosity for the parachute including Tezduyar and Sathe [94], Kim and Peskin [49], and Tutt[96]. We have chosen to implement the two latter methodologies because they share a similar frame work of computational mesh and are compatible with the front tracking environment employed.

For Kim and Peskin's method, porosity is considered as leaking pores in the immersed elastic boundary. Let β be the density of pores and there are βds pores in the interval $(s, s + ds)$ along the surface. If each pore has an aerodynamic conductance equal to γ, the flux through the pore is $\gamma(p_1 - p_2)$ where p_1 and p_2 are the pressures on the two sides of the boundary, then the flux through the interval $(s, s+ds)$ of the boundary is given by $\beta\gamma(p_1 - p_2)ds$.

Tutt's algorithm is closest to the front tracking implementation. Tutt used the Ergun equation to describe the magnitude of porous flow velocity at a given pressure difference based on two coefficients in the following equation:

$$\frac{\Delta P}{L} = \frac{\mu}{K_1} \cdot v_f + \frac{\rho}{K_2} \cdot v_f^2 = a \cdot v_f + b \cdot v_f^2$$

where:

$$K_1 = \frac{\varepsilon^3 \cdot D^2}{150 \cdot (1-\varepsilon)^2}$$

$$K_2 = \frac{\varepsilon^3 \cdot D}{1.75 \cdot (1-\varepsilon)}$$

are referred to as the viscous and inertial factors respectively. D is defined as the characteristic length, ε is the porosity and is equal to the ratio of the void and total volume. v_f, μ, and ρ are fluid velocity, viscosity, and density respectively. In the *FronTier*-airfoil code used here, the pressure difference is computed after the velocity projection into the divergence-free space. We can then use the pressure difference on two sides of the canopy to compute v_f and add it to the advection solver as flux from the immersed elastic surface.

3.3.5 Modeling of Parachute Clusters

Parachute clusters have been used in a wide range of applications throughout their history, and the process utilized for their development and design has significantly improved over the years. Compared with excessively large single canopies, parachute cluster systems have a number of benefits in many applications. These benefits include the ability to rig and manufacture the system, backup protection, and excellent stability characteristics. Of course, there are some undesirable features of parachute clusters, which include the difficulty in obtaining a time-sequenced opening of all canopies together. Most cluster applications described in the literature are National Aeronautics and Space Administration (NASA) systems, including the Apollo recovery systems and the space shuttle solid rocket booster recovery systems [62, 16, 53, 28, 77, 54]. However, parachute clusters are employed for many other applications, including numerous military applications. Military systems that use clusters include extraction systems and low-velocity airdrop systems for cargo, which consistently deliver payloads as heavy as 60,000 pounds. Based on the single parachute modeling, we implemented structures and functions for parachute clusters. Fig. 3.5 demonstrates a cluster consisting of three G11 parachutes.

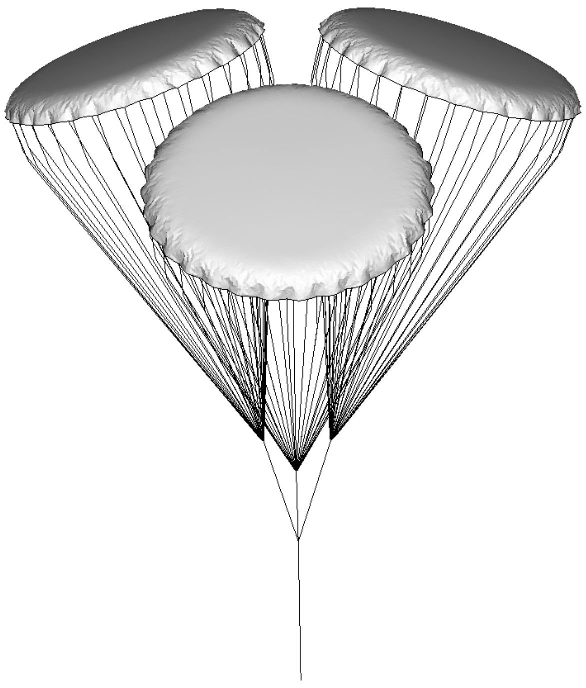
Figure 3.5: A cluster of three G11 parachutes.

Chapter 4

Application of GPU

4.1 Introduction to GPU Computing

General Purpose Graphics Processing Units (GPGPU) computing [52] is to use the GPUs together with CPUs to accelerate a general-purpose scientific and engineering application. Heterogeneous computing can offer dramatically enhanced application performance by offloading computation-intensive portions of the programming code to the GPU units, executing the remainder of the code still on the CPU. Joint CPU/GPU applications constitute a powerful combination because CPUs consist of a few cores optimized for serial processing, while GPUs on the other hand, consist of thousands of smaller, more efficient cores are designed for massive parallel calculations. Therefore, running the serial portions of the code on the CPU and intensive parallel portions of the code on the GPU improves the performance of the applications dramatically. Fig. 4.1 demonstrates the structure of a hybrid GPU/CPU code. From the viewpoint of a user, applications simply run remarkably faster. To fully exploit the acceleration effect of GPUs, the property of a specific problem should be thoroughly considered and an optimized execution resources allocation strategy should be developed.

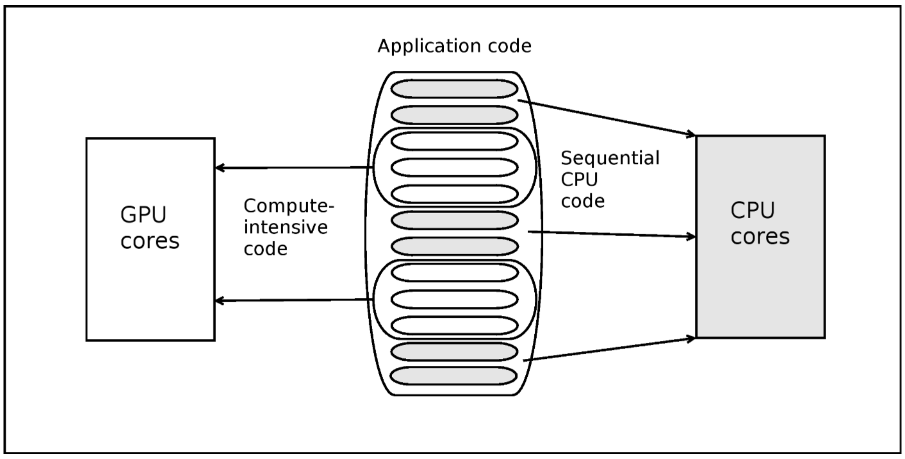

Figure 4.1: GPU accelerated code structure

4.2 Experiment Platform

Both the CPU and the GPU studies in this work were implemented on a dell precision T7600 Workstation with dual Intel Xeon E5-2687W CPUs and dual NVIDIA Quadro 6000 graphics cards. The Intel Xeon E5-2687W CPU is the latest multi-threaded multi-core Intel-Architecture processor. It offers eight cores on the same die running at 3.10 GHz. The Intel Xeon E5-2687W processor cores feature an out-of-order super-scalar micro-architecture, with newly added 2-way hyper-threading. In addition to scalar units, it also has 4-wide SIMD units that support a wide range of SIMD instructions. Each has a separate 32KB L1 cache for both instructions and data and a 256 KB unified L2 data cache. All eight cores share an 20 MB L3 data cache. The

Intel Xeon E5-2687W processor also features an on-die memory controller that connects to four channels of DDR memory. Each Quadro 6000 graphics card consists of 14 streaming multiprocessors (SMs) running at 1.15 GHz that share a single 768 KB L2 cache and 6 GB global memory on the device. Each SM consists of 32 streaming processors (SPs), a 48 KB shared memory and 32768 32-bit registers. Fedrora 18 with kernel 3.9.2-200, CUDA Toolkit 5.0 and GCC 4.7.2 were used in the computations. Table 4.1 shows the hardware structure of the computer on which the experiments run. Fig. 4.2 demonstrates the architecture of multi-GPU devices.

Hardware	CPU	Dual Eight Core XEON E5-2687W, 3.1GHz 64GB DDR3 32KB x 16 L1 Cache, 256KB x 16 L2 Cache 20MB x 2 L3 Cache
	GPU	Dual Quadro 6000 with 14 multiprocessor 448 cores, 1.15Hz 6GB global memory, 64 KB constant memory 48KB shared memory 32768 registers per multiprocessor
Software	OS	Fedora 18 with kernel 3.9.2-200.fc18.x86_64
	Compiler	gcc version 4.7.2
	CUDA	CUDA Toolkit 5.0

Table 4.1: A Dell Precision T7600 Workstation with dual NVIDIA Quadro graphics cards was used to set up the test environment.

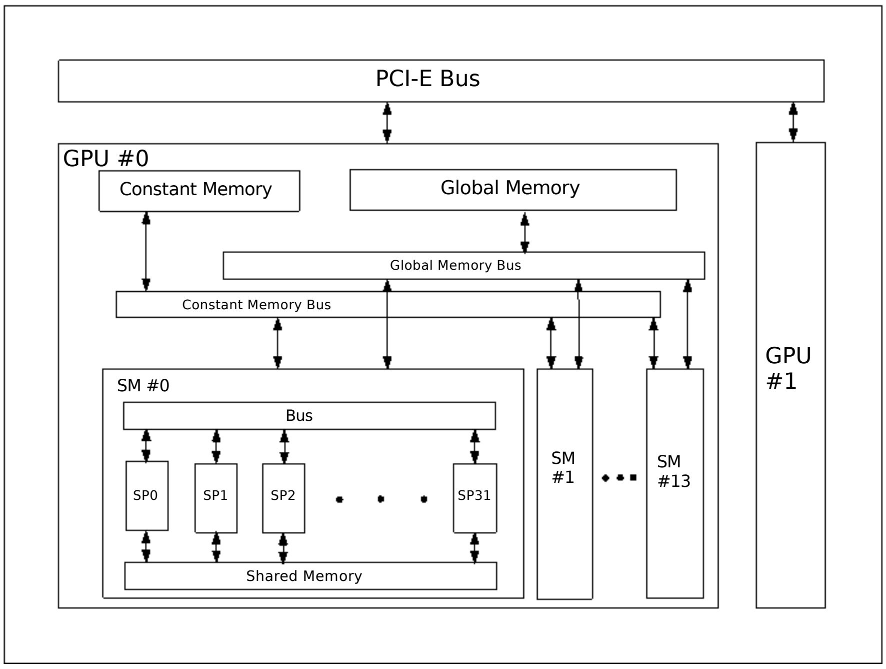

Figure 4.2: Architecture of multi-GPU devices. Each GPU hardware consists of memory (global, constant, shared) and 14 SMs. Each SM consists of 32 SPs and can run 1536 threads simultaneously.

4.3 GPU Investigation

4.3.1 Optimized Execution Resource Allocation

Hardware Limitation

In a heterogeneous computing code, the most computation intensive and parallelizable part will be executed on GPU, but the serial part will remain on CPU. GPU computation can be activated by invoking a kernel function which will launch a grid containing hundreds of thousands of threads. Thread is the most fundamental execution unit and threads in same grid execute the same kernel function. In the computation process, threads are organized into blocks and the grid is made up of thread blocks. The number and dimension of blocks in the grid and the number and dimension of threads in the block can be specified according to the requirement of specific problem.

There are some hardware limitations on the dimension of grids and blocks, as well as the number of threads per block. In our computing platform, which is Quadro 6000 with compute capability 2.0, the corresponding limitations are:

- Maximum number of threads per block is 1024
- Maximum dimension size of a thread block is $1024 \times 1024 \times 64$
- Maximum dimension size of a grid is $65535 \times 65535 \times 65535$

Listing 4.1 demonstrates a calling of the kernel function of 1D gas dynamic model, *flux_kernel*, on the host. When this kernel function is launched, a one dimensional grid with size $(total\_mesh\_size + block\_size[0] - 1)/block\_size[0]$ is generated and each block is one dimensional with size $block\_size[0]$. This thread hierarchy setting satisfies the hardware limitation of the current computing platform.

```
dim3 block_size(128, 1, 1);
dim3 grid_size((total_mesh_size + block_size[0] - 1) / block_size
    [0], 1, 1);
int shared_mem_size_Bytes = 11 * (block_size[0] + 2 *
    ghost_point_size) * sizeof (double);
flux_kernel <<<grid_size, block_size, shared_mem_size_Bytes>>>(
    gamma, lambda, maxeig[0], maxeig[1], maxeig[4], extend_size,
    ghost_size , dev_u , dev_flux);
```

Listing 4.1: The host calls the kernel function of 1D gas dynamic.

Dynamic Resource Allocation

To obtain better parallalization effect, the intuitive idea is to run as many threads as possible concurrently. The number of active threads is limited by available execution resources which is dynamically allocated block-by-block in each streaming multiprocessor. So, efficient thread management is important for GPU acceleration. The corresponding resource limitations on our computing platform are:

- Maximum blocks per streaming multiprocessor is 8.
- Maximum registers per streaming multiprocessor is 32768 *(unit 32 bit).*
- Maximum shared memory per streaming multiprocessor is 49152 bytes.
- Maximum warps per streaming multiprocessor is 48, which is equivalent to maximum 1536 threads per streaming multiprocessor.

Block size determines the number of active threads and occupancy of computing resources in each streaming multiprocessor(SM). Inside a block, warp, which is made up of 32 threads, is the basic unit for resources allocation.

Based on the register limitation, the maximum number of warps per SM is

$$number\ of\ warps\ per\ SM = \left\lfloor \frac{total\ registers}{registers\ per\ thread \times threads\ per\ warp} \right\rfloor,$$

which is equivalent to the maximum number of threads per SM

$$total\ active\ threads = number\ of\ warps\ per\ SM \times threads\ per\ warp.$$

Then, the maximum number of blocks derived from register limitation is

$$\left\lfloor \frac{total\ active\ threads}{block\ size} \right\rfloor.$$

Considering the shared memory limit, the maximum number of threads per SM is

$$\left\lfloor \frac{total\ shared\ memory}{shared\ memory\ per\ block.} \right\rfloor$$

In sum, for any fixed block size, the maximum number of blocks per SM can be calculated as

$$number\ of\ blocks\ per\ SM = \min\left(\left\lfloor \frac{total\ active\ threads}{block\ size} \right\rfloor, \right.$$

$$\left. \frac{total\ shared\ memory}{shared\ memory\ per\ block}, limit\ of\ blocks\ per\ SM. \right)$$

Theoretical analysis based on the actual execution resources is given in section 4.4 in detail which is consistent with the numerical tests results.

4.4 GPU Application to Gas Dynamics

In recent years high-order numerical methods have been widely used to effectively resolve complex flow features such as turbulent or vertical flows [27].

High-order shock-capturing schemes such as the Essentially Non-Oscillatory (ENO) and Weighted ENO (WENO) [58, 43] schemes not only make the computational fluid dynamics (CFD) solvers get rid of extremely fine mesh for complex flows, but also perfectly eliminate the oscillations near discontinuities. However, such high-order schemes are complicated and much more time consuming than traditional schemes. GPUs, although originally developed for computer graphics, now become a popular tool for scientific computations, especially in the CFD area [64, 103] because it extremely enhanced the time efficiency of the CFD computations. In this part, we focus on the use of GPUs for solving one dimensional fluid dynamics problems with WENO scheme.

4.4.1 Euler Equations of the Gas Dynamics

The one dimensional governing equations of the compressible in-viscid gases are the Euler equations of gas dynamics, shown as Eq. (4.1) .

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x = 0 \tag{4.1}$$

where

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (E+p)u \end{pmatrix}$$

$$E = \rho(e + \frac{u^2}{2}), p = \rho e(\gamma - 1), \gamma = 1.4$$

where ρ, u, P, e, and γ denote the density, velocity, pressure, internal energy per unit mass and ratio of specific heats, respectively. The Jacobian matrix of the Euler equations is defined as Eq. (4.2)

$$A = \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \tag{4.2}$$

The eigenvalues of the Jacobian matrix A are Eq. (4.3)

$$\lambda_1 = u - c, \quad \lambda_2 = u, \quad \lambda_3 = u + c \tag{4.3}$$

where $c = \sqrt{\frac{\gamma p}{\rho}}$ is the sound speed. and the corresponding right eigenvectors are Eq. (4.4)

$$\mathbf{r}_1 = \begin{pmatrix} 1 \\ u - a \\ H - ua \end{pmatrix}, \quad \mathbf{r}_2 = \begin{pmatrix} 1 \\ u \\ \frac{u^2}{2} \end{pmatrix}, \quad \mathbf{r}_3 = \begin{pmatrix} 1 \\ u + a \\ H + ua \end{pmatrix} \tag{4.4}$$

where H is the total specific enthalpy, which is related to the specific enthalpy h and other variables, namely

$$H = \frac{(E+p)}{\rho} = \frac{1}{2}u^2 + h, \quad h = e + \frac{P}{\rho} \tag{4.5}$$

We denote the matrix whose columns are eigenvectors in Eq. (4.4) by

$$\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) \tag{4.6}$$

and denote $L = R^{-1}$.

4.4.2 GPU implementation

The Euler equations of gas dynamics are nonlinear hyperbolic equations and its Jacobian matrix has three different eigenvalues which represent the wave speeds. Let $\{I_i\}$ be a partition of the computation domain R, where $I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is the i-th cell. The evaluation of the numerical flux of the WENO scheme consists the following steps:

1. At the fixed grid point $x_{j+\frac{1}{2}}$, compute average state $U^m_{j+\frac{1}{2}}$ by the simple mean $U^m_{j+\frac{1}{2}} = \frac{1}{2}(U_j + U_{j+1})$.

2. Compute the eigenvalues $\lambda_{i,j+\frac{1}{2}}$ $(i = 1, 2, 3)$, the matrix R, and L in terms of $U^m_{j+\frac{1}{2}}$.

3. Do the local characteristic decomposition to get primitive variables.

4. Compute both the positive flux and negative flux.

5. Update U_j.

GPUs parallelization is desirable since the floating point operation of the above procedures is highly intensive and the size of the partition can be very large. We interchangeably use the term ”host” to refer to the CPU, and the term ”device” to refer to the GPUs. Listing 4.2 shows part of the host-side code for the stepping and Listing 4.3 shows the part of the device side code for the flux calculating.

For each time step, the host fulfills three steps:

1. Copy the fluid states of all points to the device.

2. Call the GPU kernel function to calculate the flux on all points and wait.

3. Copy all points' flux back from the device.

When the GPU kernel function was called, the device was triggered. Each thread on the device acts according to its identification number. An activated thread will fetch corresponding fluid state data and perform the flux calculation according to the WENO scheme.

```
// for each time step
{
    // copy fluid variables from host to device
    ... ...
    // call kernel to compute flux
    flux_kernel <<<grid_size, block_size, shared_mem_size_Bytes
   >>>(gamma, lambda, maxeig[0], maxeig[1], maxeig[4],
   extend_size, ghost_size, dev_u, dev_flux);
    // copy the results back from device to host
    ... ...
}
```

Listing 4.2: Partial host-side code of the WENO scheme implementation.

```
__global__ void flux_kernel(...) {
    ... ...
    // use shared memory for acceleration
    extern __shared__ double uf[];
    // get thread global and local id
    int gindex = threadIdx.x + blockIdx.x * blockDim.x +
ghost_size;
    int lindex = threadIdx.x + ghost_size;
    int block_extend_size = blockDim.x + 2 * ghost_size - 1;

    if (gindex < extend_size - ghost_size + 1) {
        // 1) load fluid variables from global memory to shared
memory
        // 1.1) centeral cells for of block
        for (int i = 0; i < 6; i++) {
            uf[lindex + i * block_extend_size] = dev_u[i][gindex];
        }
        // 1.2) ghost cells of the block
        if (threadIdx.x < ghost_size) {
            for (int i = 0; i < 6; i++) {
                uf[threadIdx.x + i * block_extend_size] = dev_u[i
][gindex - ghost_size];
            }
        }
        if (lindex > blockDim.x || gindex + 2 * ghost_size - 1 >
extend_size) {
            for (int i = 0; i < 6; i++) {
                uf[lindex + ghost_size - 1 + i * block_extend_size
] = dev_u[i][gindex + ghost_size - 1];
            }
        }
        // 2) synchronize all threads
        __syncthreads();
        // 3) compute average state
        for (int i = 0; i < 5; i++) {
            u_mid[i] = 0.5 * (uf[lindex + i * block_extend_size] + uf[lindex - 1 + i * block_extend_size]);
        }
        ... ...
        // 4) compute matrix R, and L in terms of the average state
        ... ...
        // 5) do the local characteristic decomposition to get primitive
        // variables, matmvec(y,A,x) returns y = Ax
        for (int i = 0; i < 6; ++i) {
            for (int j = 0; j < 5; ++j)
                u[j] = uf[lindex - ghost_size + i + j * block_extend_size];
            matmvec(sten_u[i], RL, u);
            for (int j = 0; j < 5; ++j)
                u[j] = uf[lindex - ghost_size + i + (6 + j) * block_extend_size];
            matmvec(sten_f[i], RL, u);
        }
        ... ...
        // 6) compute both the positive flux and negative flux
        for (int j = 0; j < 5; ++j) {
            f_tmp[j] = weno5_scal(gfluxp[j]);
            f_tmp[j] += weno5_scal(gfluxm[j]);
        }
        matmvec(ff, RL, f_tmp);
        ... ...
```

```
    }
}
```

Listing 4.3: Partial device-side code of the WENO scheme implementation.

4.4.3 Optimization

Shared memory usage

The computation on different stencils can be performed simultaneously by GPUs which dramatically enhanced the time efficiency. Fifth order finite difference WENO scheme is used on structured mesh in these simulations. Each stencil needs to read the information of six nearby points which is still very time consuming. The shared memory of the GPUs give a better solution to this problem. Because it is on-chip, shared memory latency is roughly 100x lower than uncached global memory latency. Shared memory is allocated per thread block, so all threads in the same block have access to the same shared memory. Threads can access data in the shared memory loaded from the global memory by other threads within the same block. Using this capability, the code reads the information of each point to the shared memory by corresponding thread once, then for stencils need the information of this point can fetch it directly from the shared memory. This strategy dramatically enhanced the performance of the computation. Figure. 4.3 demonstrates the difference between cases without and with shared memory usage.

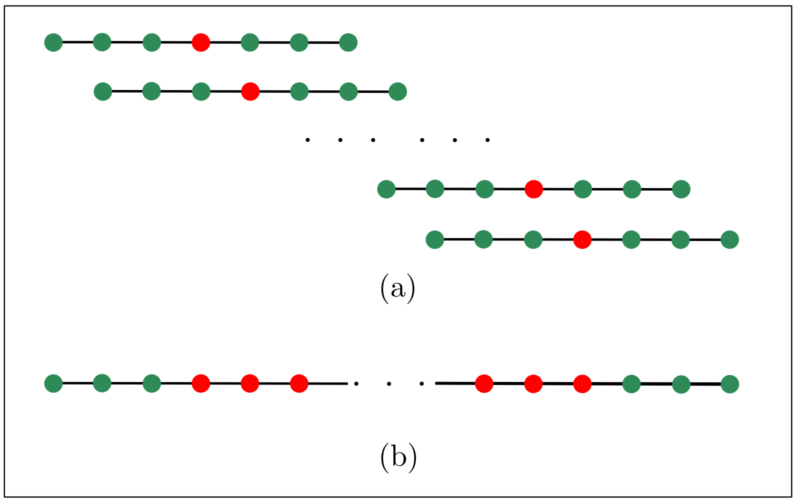

Figure 4.3: Fifth order WENO scheme stencils. Each point represents a computational node. Red points are updated by the threads while the green points are only used as data source. Each thread updates one red point only. (a) Without shared memory usage, each thread reads seven points' information. (b) With shared memory, each thread reads only one point's information. In the testing case, the number of threads in one block is 512. Without shared memory usage, each block will fetch 3584 (512*7) points' information; with shared memory, only 518 (512+6) points' information is necessary.

Block size

In section 4.3, we discussed the optimization of block size based on the hardware limitation and dynamic resources allocation theoretically. Here we give a detailed analysis based on the actual execution resources of this application. To find the optimized thread management strategy, we calculate the thread occupancy at different block sizes. This algorithm requires 63 registers per thread. Then, the maximum number of active threads per SM is

$$\left\lfloor \frac{32768}{63 \times 32} \right\rfloor \times 32 = 512,$$

and this implies the theoretical maximum thread occupancy is

$$\frac{512}{1536} = \frac{1}{3}.$$

In Table 4.2, given specific block size, the number of blocks per SM can be limited by one or several of three major factors: register limitation (544 case), shared memory limitation (64 case) and block per SM limitation (32 case). In addition, different block sizes lead to different effective threads per device which determines the scale of parallalization. As we can see, in the case of block size 128, 256 and 512, the number of effective threads is larger and they can handle more threads at the same time. So, in this specific problem, 128, 256 and 512 are optimized block size. However, even in these most optimized cases, only 33.3% of SM threads are occupied. To further improve the efficiency of the problem, a direct method is to improve the threads occupancy by eliminating number of registers consumed per thread and corresponding shared memory per block.

To verify above analysis, seven groups of test cases with block sizes of 32, 64, 128, 192, 256, 384, and 512, respectively are carried out. Each group has eight different mesh sizes of 1024, 2048, 3072, 4096, 5120, 6144, 7168, and 8192. Figure. 4.4 shows the results. In table 4.2 we conclude that the effective threads per device is 3584 when the block size is 32. In the the right

Block size		32	64	128	192	256	384	512	544
Shared memory/block (bytes)		3344	6160	11792	17424	23056	34320	45584	48400
Number of blocks limited by	register	16	8	4	2	2	1	1	0
	shared memory	14	7	4	2	2	1	1	1
	max blocks/SM	8	8	8	8	8	8	8	8
	effective blocks	8	7	4	2	2	1	1	0
Effective threads/SM		256	448	512	384	512	384	512	0
Effective threads/device		3584	6272	7168	5376	7168	5376	7168	0
SM threads occupancy (%)		16.7	29.2	33.3	25.0	33.3	25	33.3	0

Table 4.2: Execution Resource analysis. In this particular application, each thread needs 63 registers and each block needs $(block\ size{+}2{*}ghost\ size){*}(number\ of\ shared\ doubles\ per\ thread){*}sizeof(double)$ shared memory where $ghost\ size = 3$, and $number\ of\ shared\ doubles\ per\ thread = 11$ are decided by the algorithm and $sizeof(double) = 8$ is decided by the compiler. Given these parameters and the total size of registers and shared memory per SM has, we can get the maximum number of blocks limited by registers and shared memory.

upper subplot (subplot (2)) of Fig. 4.4, we can see two jumps on the line, the first jump happened when the mesh size increased from 3072 to 4096 and the second one from 7168 to 8192. The first one happened because the effective threads number 3584 is between 3072 and 4096. When the mesh size is 3072, all the threads can perform simultaneously; however, when the mesh size is 4096, two rounds are needed. In the first round, only 3584 threads can perform simultaneously and the rest threads have to wait until the first round ends. Similarly, since $7168 = 3584 * 2 < 8192$, two and three rounds needed when mesh size is 7168 and 8192 respectively. In the same way, we can explain the jumps in other subplots of Fig. 4.4. In sum, from the consistency of the analysis in the table 4.2 and the results in Fig. 4.4, we can conclude that the method used to optimize the block size is reliable and using optimized block size to perform calculation is less likely to suffer performance jumps.

4.5 GPU Application to the Spring Model

Fig. 4.5 is the complete flow chart of the algorithm. Upon testing, we identified the panel marked by red color in Fig. 4.5 is the most time consuming section.

Solving the spring model by 4-th order Runge-Kutta method consists the following steps:

1. Find the positions of all the neighbors of each vertex.
2. Calculate the force on each vertex using Delingette method .
3. Calculate the acceleration of each vertex.
4. Update the location of the vertex.
5. Go to step (2) if this is not the 4-th Runge-Kutta step, end this time step, otherwise.

The computation of the spring model is time consuming because the number of the vertices is large. However, at each time step, all vertices are

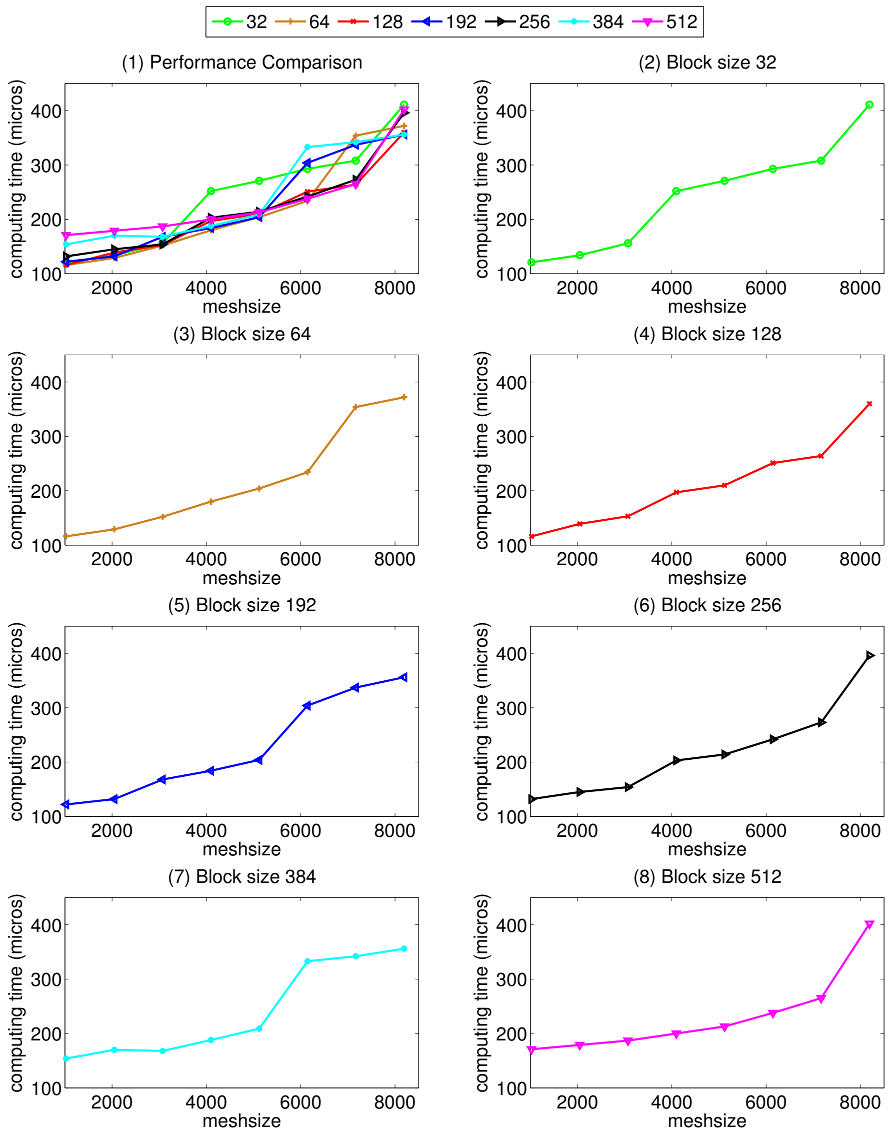

Figure 4.4: GPU computing performance on different block sizes and mesh sizes.

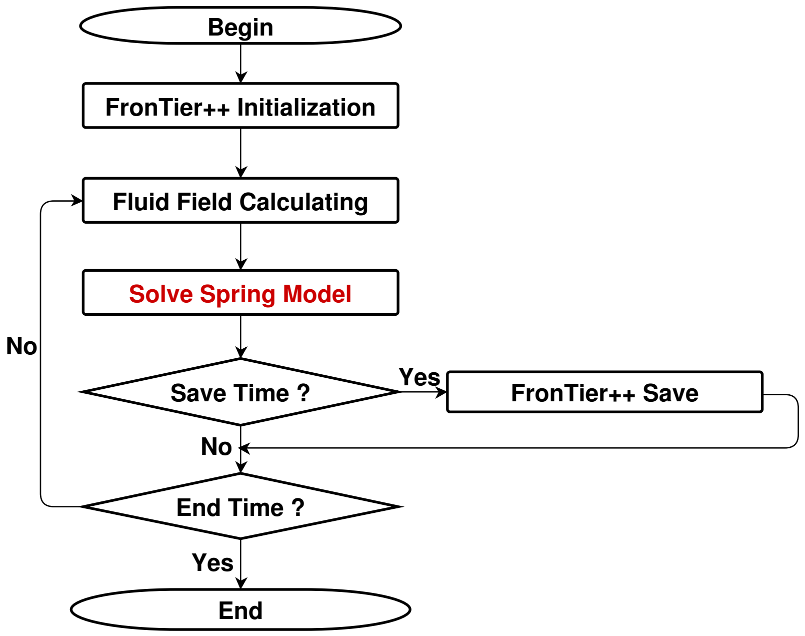

Figure 4.5: Flow chart of the complete algorithm. The computation of the spring model, which is marked by red color, is the most time consuming section. This part is calculated in parallel by the GPU device with multiple threads in order to improve the computational efficiency

independent. To accelerate the calculation of this part, we then shift it to the GPU cores for massively parallel processing.

For each time step, the host performs the following steps:

1. Copy the position and velocity of each point to the device.
2. Call the first GPU kernel function to duplicate current position of each point.
3. Call the second GPU kernel function to calculate acceleration of each point .

4. Call the third GPU kernel function to fulfill the Runge-Kutta step and goto step (3) if this is not the 4-th Runge-Kutta step.

5. Copy the position and velocity of each point back from the device.

The difference between this case and the gas dynamics case is that the mesh here is unstructured mesh while the mesh used in the gas dynamics problem is structured mesh. Therefore, we can not use the shared memory of GPUs to further enhance the performance of the computation. Listing 4.4 shows part of the host-side code for each step.

```
// for each time step
{
    // copy the position and velocity of all points from host to device
    ... ...
    // duplicate current position of each point
    pos_to_old <<<grid_size, block_size>>>(dev_x_old, dev_x_pos, dev_v_old, dev_v_pos, size, dim);
    // calculate acceleration of each point
    comp_spring_accel <<<grid_size, block_size>>>(dev_sv, dev_accel, size,dim);
    // update each point's location by fourth order Rung-Kutta method
    for (n = 0; n < loop; ++n) {
        RK_1 <<<grid_size, block_size>>>(dev_x_new, dev_v_new, dev_x_pos, dev_v_pos, dev_x_old, dev_v_old, dev_accel, dt, size, dim);
        comp_spring_accel <<<grid_size, block_size>>>(dev_sv, dev_accel, size, dim);

        RK_2 <<<grid_size, block_size>>>(dev_x_new, dev_v_new,
    dev_x_pos, dev_v_pos, dev_x_old, dev_v_old, dev_accel, dt,
    size, dim);
        comp_spring_accel <<<grid_size, block_size>>>(dev_sv,
    dev_accel, size, dim);

        RK_3 <<<grid_size, block_size>>>( dev_x_new, dev_v_new,
    dev_x_pos, dev_v_pos, dev_x_old, dev_v_old, dev_accel, dt,
    size, dim);
        comp_spring_accel <<<grid_size, block_size>>>(dev_sv,
    dev_accel, size, dim);

        RK_4 <<<grid_size, block_size>>>(dev_x_new, dev_v_new,
    dev_x_pos, dev_v_pos, dev_x_old, dev_v_old, dev_accel, dt,
    size, dim);
        if (n != n_tan - 1) {
            comp_spring_accel <<<grid_size, block_size>>>(dev_sv,
    dev_accel, size, dim);

            pos_to_old <<<grid_size, block_size>>>(dev_x_old,
    dev_x_pos, dev_v_old, dev_v_pos, size, dim);
        }
    }
    // copy the results back from device to host
    ... ...
}
```

Listing 4.4: Partial host-side code of the Spring-Mass model implementation.

4.6 GPU Application to American Option Pricing

4.6.1 Partial-integro differential equation for American options

Substantial volume of exchange-traded options are American options and there have been many attempts to solve such option pricing problems which have no closed form of analytical solutions. Most pioneer works [6, 63, 18] assume that the log-return of the underlying asset follows normal distribution and this assumption fails to capture skewness and asymmetry properties indicated by empirical observations [51]. A possible solution is to use the exponential Lévy model to simulate the price of the underlying asset which represents the extreme returns as discontinuities in the prices.

We build the model based on the generalized hyperbolic distributions introduced by Barndorff-Nielsen [4] which contain five parameters and encompass many special cases such as the hyperbolic distribution and the normal inverse Gaussian distribution etc. The Lévy measure which characterizes the exponential Lévy model based on generalized hyperbolic distribution has the form

$$g(x) = \frac{e^{\beta x}}{|x|}\left(\int_0^\infty \frac{\exp(-\sqrt{2y+\alpha^2}|x|)}{\pi^2 y[J^2_{|\lambda|}(\delta\sqrt{2y}) + Y^2_{|\lambda|}(\delta\sqrt{2y})]}dy + 1_{\lambda>0}\lambda e^{-\alpha|x|}\right)$$

where $\lambda \in \mathbb{R}$ demonstrates certain sub-classes; $\alpha > 0$ determines the shape; $0 \le |\beta| < \alpha$ represents the skewness; $\delta > 0$ is the scaling parameter. The fifth parameter $\mu \in \mathbb{R}$ which is not shown in the formula indicates the location. Due to these five parameters, the distribution is relatively flexible and can better characterize real market behaviors; but this will lead to more complicated form and multiple integrations which greatly increase calculation complexity and calls the need for GPU.

Based on Cont, Rama and Tankov [17], assume $f(\tau, x) = V(t, S)$, $\tau = T - t$ and $x = \ln \frac{S}{S_0}$, the partial integro differential equation for European option is

$$f_\tau(\tau, x) - (r - \frac{\sigma^2}{2}) f_x(\tau, x) - \frac{\sigma^2}{2} f_{xx}(\tau, x) + r f(\tau, x)$$

$$- \int g(y) \left[f(\tau, x + y) - f(\tau, x) - (e^y - 1) f_x(\tau, x) \right] dy = 0,$$

where t is time, S is underlying asset price, r is the interest rate, σ^2 is the volatility of the underlying asset and $V(t, S)$ is the option price at time t with underlying asset price S. Then the American put option pricing can be realized by coupling appropriate free boundary condition.

4.6.2 GPU implementation

Assume $f_k^n = f(\tau_n, x_k)$, then the discretization of the PIDE can be represented as

$$\frac{f_k^{n+1} - f_k^n}{\triangle t} = \left(r - \frac{\sigma^2 + \sigma_\epsilon^2}{2} - \int_{|y| > \epsilon} g(y)(e^y - 1) dy \right) \frac{f_{k+1}^n - f_{k-1}^n}{2 \triangle x} +$$

$$\frac{\sigma^2 + \sigma_\epsilon^2}{2} \frac{f_{k+1}^{n+1} - 2 f_k^{n+1} + f_{k-1}^{n+1}}{(\triangle x)^2} - r f_k^n + \int_{|y| > \epsilon} g(y) \left[f(\tau_n, x_k + y) - f_k^n \right] dy$$

where $\sigma_\epsilon^2 = \int_{-\epsilon}^{\epsilon} y^2 g(y)\, dy$ [100]. The flow chart in Fig. 5.18 (left) illustrates the major steps of the algorithm. After testing, the calculation of Lévy measure $\{g(k\triangle x)\},\ k = 1, \cdots, N$ related terms is identified as quite intensive in the pricing of single option and can be moved to GPU. The corresponding flow chart describes the process can be found in Fig. 5.18 (right). Listing 4.5 shows part of the host-side code for the stepping of pricing single option.

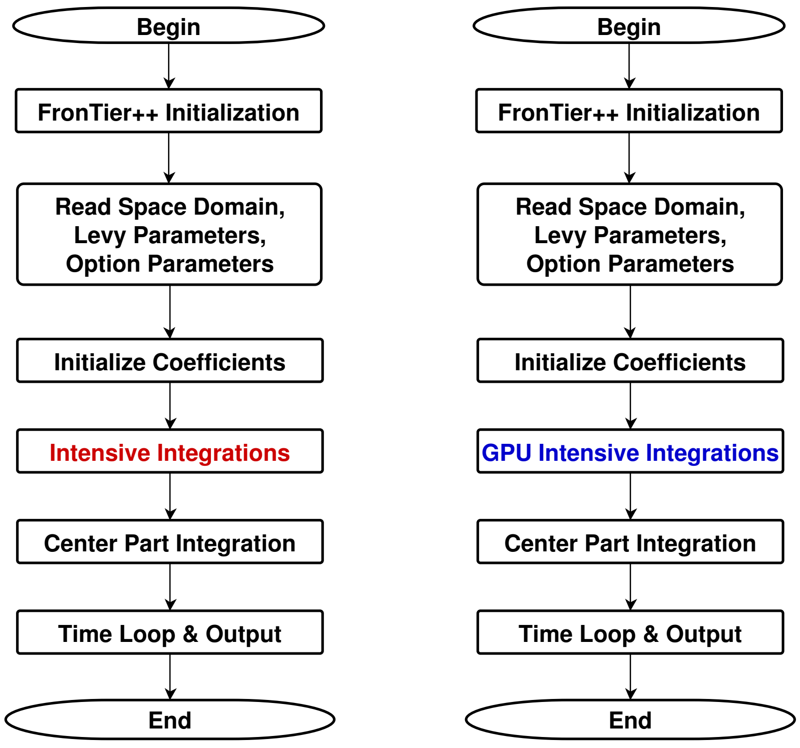
Begin
FronTier++ Initialization
Read Space Domain, Levy Parameters, Option Parameters
Initialize Coefficients
Intensive Integrations
Center Part Integration
Time Loop & Output
End
Begin
FronTier++ Initialization
Read Space Domain, Levy Parameters, Option Parameters
Initialize Coefficients
GPU Intensive Integrations
Center Part Integration
Time Loop & Output
End

Figure 4.6: Without-GPU (Left) and With-GPU (Right) Flow chart for single option.

```
// called by host function only once
{
    // allocate memory for option parameters and copy them from
    host to device
    ... ...
    // calculate option prices for given parameters on each thread
    GHI_kernel <<<grid_size, block_size>>>(N, tempx1, tempx2, dx,
     dev_BlI, dev_BlII, dev_Br, dev_params);
    // copy the results back from device to host
    ... ...
}
```

Listing 4.5: Partial host-side code of the implementation of pricing single option.

```
__global__ void GHI_kernel(
        int N,
        double tempx1,
        double tempx2,
        double dx,
        double *dev_BlI,
        double *dev_BlII,
        double *dev_Br,
        PARAMS *params) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    int j;

    // Thread task managed to avoid execution divergence
    if (i < N - 1) {
        tempx2 = tempx2 - i*dx;
        dev_BlI[i] = dev_GHILI(params, tempx2, 0.0, 1.0, 0.0);
    }
    else if (i < 2 * N - 2) {
        j = i - N + 1;
        tempx2 = tempx2 - j*dx;
        dev_BlII[j] = dev_GHILII(params, tempx2, 0.0, 1.0, 0.0);
    } else if (i < 3 * N - 3) {
        j = i - 2 * N + 2;
        tempx1 = tempx1 - j*dx;
        dev_Br[j] = dev_GHILR(params, tempx1, 0.0, 1.0, 0.0);
    }
}
```

Listing 4.6: Partial device-side code of the implementation of pricing single option.

At each node, three different integral terms are calculated using three threads. Due to the single-instruction, multiple-thread hardware execution style of GPU, the threads are organized as List 4.6. This arrangement ensures that threads in the same warp will follow same paths of control flow and avoid extra execution time result from thread divergence.

The motivation of using heterogeneous computing on single American option pricing is due to the high modeling complexity. In contrast, the pricing of multiple options which involve large problem sizes is also a good candidate for parallel computing. The independence between each single option and corresponding unique parameters enables the whole calculation process of one option being fulfilled in a single thread. This joint CPU/GPU algorithm greatly outperforms pure CPU algorithm, especially in large number of options case, and Fig. 4.7 illustrates this process. Listing 4.7 shows part of the host-side code for the stepping of pricing multiple options.

```
// called by host function only once
```

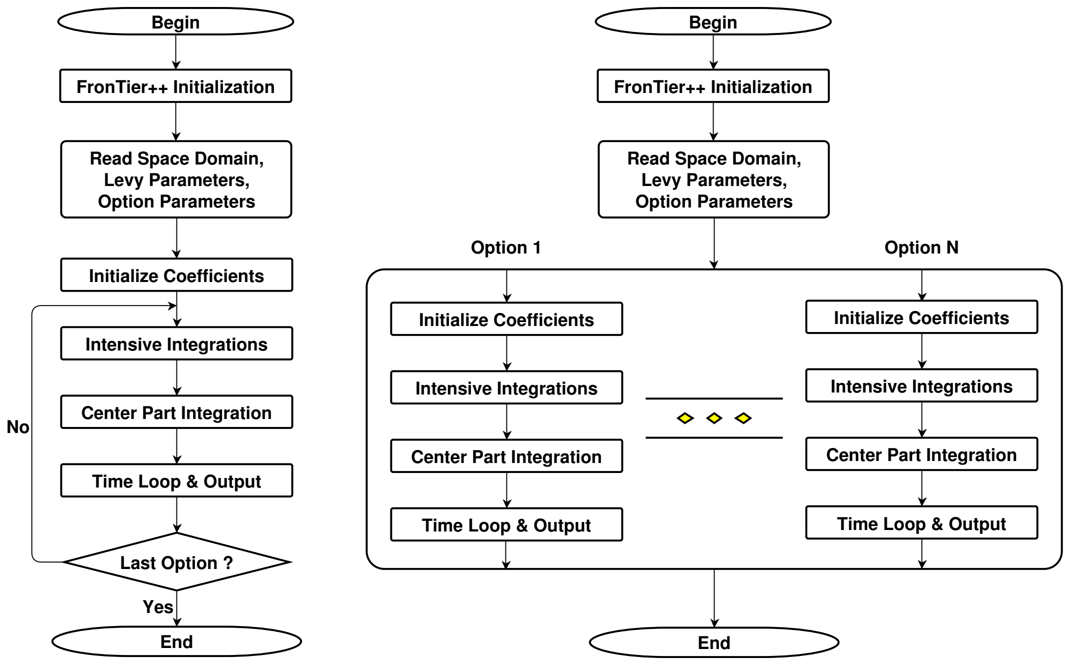

Figure 4.7: Without-GPU (Left) and With-GPU (Right) Flow chart for multiple options.

```
{
    // allocate memory for option parameters and copy them from host to device
    ... ...
    // calculate option prices for given parameters on each thread
    drive_kernel <<<grid_size, block_size>>>(dev_params, dev_lambda, dev_alpha, dev_beta, dev_delta, dev_E, dev_sigma, dev_S0, dev_T, dev_dl, dev_dr, dev_BlI, dev_BlII, dev_Br, dev_B, dev_S, dev_f0, dev_oldf, dev_newf, dev_V, dev_tempBl, dev_tempBr, dev_c, dev_d, num_op);
    // copy the results back from device to host
    ... ...
}
```

Listing 4.7: Partial host-side code of the implementation of pricing multiple options.

4.7 Computation-Memory Ratio

In the previous sections, we explored the hardware limit of GPU computing and derived optimized block size subject to execution resources. The one dimensional gas dynamic model based on the WENO scheme is used to show the calculation process. Only computing time, i.e. the time running on kernel, is considered. However, the three introduced heterogeneous computing models also involve data transfer between CPU and GPU which should be taken into consideration for performance measurement. In chapter 5, we will discuss the advantages and disadvantages of both GPU and CPU, based on the time-including computing time and data transfer time.

Chapter 5

Numerical Results

The system of equations for the spring model is nonlinear and an analytical proof of convergence for this model is very difficult. Therefore proof of convergence through numerical mesh refinement is presented here, e.g., with fixed initial and boundary conditions are used, the displacement, the energy (kinetic energy and potential energy), and the total length (2D) or area (3D), which are functions of time, are convergent under mesh refinement. It is not surprising that the convergence rate is of first order due to the equation of each vertex point in the spring mesh involves only its immediate neighbors and that convergence is not ideal when the surface is compressed or wrinkled.

Proving analytical convergence for the coupled spring-mass model fluid solver system is even more difficult than the spring-mass model itself; as a result, coupled simulation results are compared directly with experiments. One of the main interests of this effort is the short time history response of the parachute during inflation where the drag force can be validated with experimental data. It is demonstrated that the coupled method discussed above captures important properties of the parachute inflation force response.

5.1 Verification of Numerical Convergence

One crucial step in assessing the mathematical validity of the spring-mass model is convergence under mesh refinement. The following question is, if the system is convergent, to what continuum model and partial differential equations the discrete computation of the spring-mass model would converge? The numerical results reveal that the spring model is convergent under the conditions that the total mass of the fabric surface is kept constant and that both the spring tensile stiffness and angular stiffness conform with Young's modulus and the Poisson ratio of the material[19].

A sequence of numerical simulations of a string with fixed boundary in a two-dimensional (2-D) domain and a membrane with fixed boundary in a three-dimensional (3-D) domain are carried out. In both 2-D and 3-D simulations, the spring constant of the string (2-D) or the membrane (3-D) is conformed with the material's Young's modulus and Poisson ratio. The total mass, which is the summation of all point mass $M_{total} = Nm$, is keep constant where N is the total number of points and m is the mass of each mass point. Therefore, as the computational mesh is refined and the total number of mass points increases, the point mass m is decreased in proportion to the reciprocal of N.

The simulations are computed in domains of $1\times1m^2$ in 2-D and $1\times1\times1m^3$ in 3-D. In the first test case, the dynamic motion of a swinging string with one end fixed is simulated. The initial length of the string is $0.583m$ and a weight of $0.5g$ as shown in Fig. 5.1. The total number of grid points is increased and the point masses for each experiment varied accordingly. The details of each simulation are presented in Table 5.1. The total length and total kinetic energy of the string as a function of time and their Cauchy errors are displayed in Fig. 5.2 and the numerical errors are shown in Table 5.2. From Table 5.2, it is clear that the errors in the Cauchy sequence are reduced approximately by half each time as we reduce the average mesh size by half. This indicates that the sequence is convergent to first order.

In the second case, a circular vibrating membrane with radius of $0.4m$ and a total weight of $380g$ is simulated. The membrane is linearly perturbed initially from the center to the fixed boundary . Fig. 5.3 shows the membrane at $t = 1sec$ and $t = 2sec$ in a sequence of three different mesh refinements. The refinement experiments include four levels of refinements corresponding to the mesh sizes of 15^3, 30^3, 60^3, and 120^3, respectively. The details of each simulation are presented in Table 5.1. The errors of the total area in the Cauchy sequence are shown in Fig. 5.4 and Table 5.3. The convergence of the membrane is time synchronized as in the string chord case. From Fig. 5.4 and Table 5.3, we can see that the errors of Cauchy sequence are reduced approximately by half in each step as the average mesh size is reduced by half. This implies that the membrane spring-mass system is also convergent in first order.

Swing	Mesh size	Stiffness(N/m)	Number of points	Average mass(g)
case 1	50^2	5000	60	0.008333
case 2	100^2	10000	123	0.004065
case 3	200^2	20000	248	0.002016
case 4	400^2	40000	498	0.001004
Drum	Mesh size	Stiffness(N/m)	Number of points	Average mass(g)
case 1	15^3	1000	266	1.428571
case 2	30^3	1000	990	0.383838
case 3	60^3	1000	3790	0.100264
case 4	120^3	1000	14841	0.025605

Table 5.1: Initial configuration of the 2-D and 3-D simulations

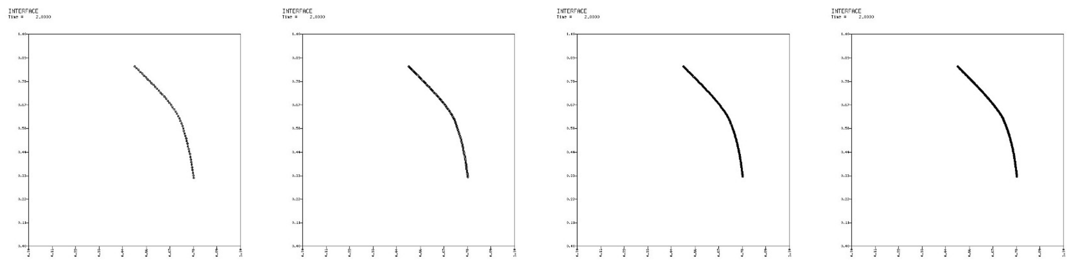

Figure 5.1: Convergence test of the string chord under mesh refinement. From left to right the total number of points in the string are 60, 123, 248, 498 respectively. The total mass of the string chord as well as the payload at the lower end are kept constant in the simulations.

5.2 Verification of Young's Modulus and Poisson's Ratio

Young's modulus, also known as the tensile modulus or elastic modulus, is a measure of the stiffness of an elastic material and is a parameter used to

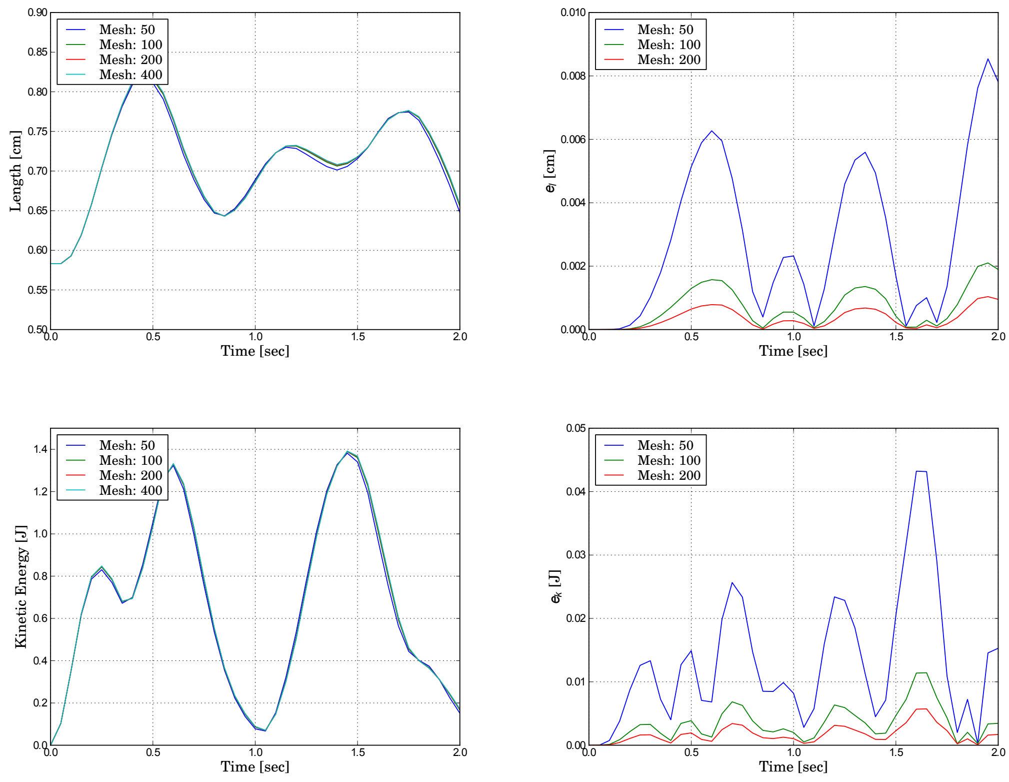

Figure 5.2: Convergence test results of spring model on string chord. In this test the string chord is fixed at one end while the other end has a payload and is free to move. The simulations are on the sequences with 60, 123, 248, 498 points respectively. The total mass $M = Nm$ is a constant in the simulation. The upper left plot shows the string lengths during the four simulations and the upper right plot is the Cauchy error of the sequences. The lower plots are for the total kinetic energy of the system.

characterize materials. It is defined as the ratio of the stress along an axis over the strain along that axis in the range of stress in which Hooke's law is valid. In solid mechanics, the slope of the stress-strain curve at any point is

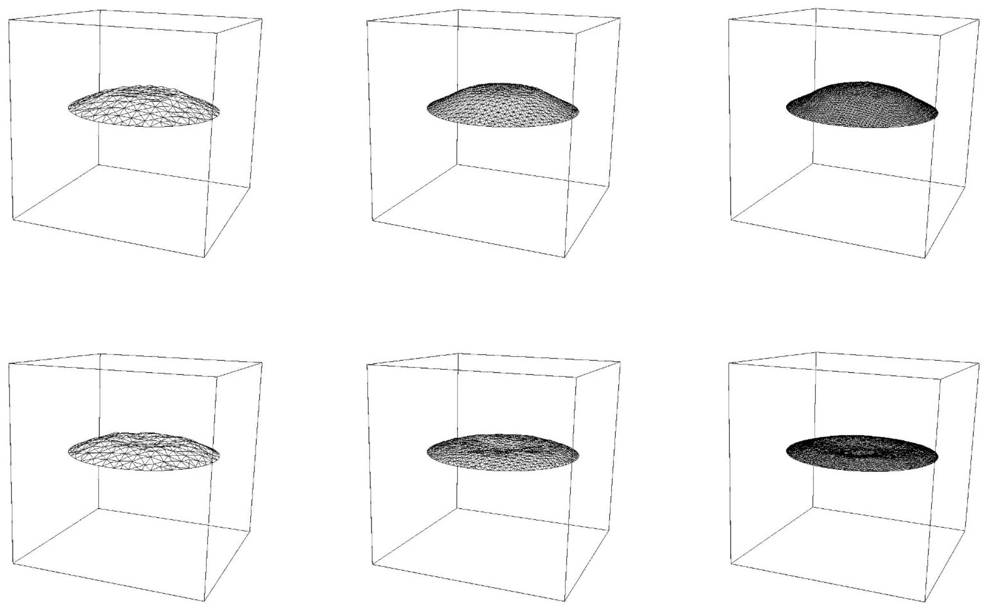

Figure 5.3: Convergence test of the drum membrane under mesh refinement. From left to right the computational mesh of the domain are 15^3, 30^3, and 60^3 respectively. The total mass of the membrane is kept constant in the simulations. The upper three plots show the membrane position at $t = 1sec$ and the lower plots show the membrane position at $t = 2sec$.

called the tangent modulus. The tangent modulus of the initial, linear portion of a stress-strain curve is called Young's modulus. Young's modulus, E, can be calculated by dividing the tensile stress by the tensile strain in the elastic portion of the stress-strain curve:

$$E = \frac{\text{tensile stress}}{\text{tensile strain}} = \frac{\sigma}{\epsilon} = \frac{F/A_0}{\Delta l/l_0} = \frac{F l_0}{A_0 \Delta l} \tag{5.1}$$

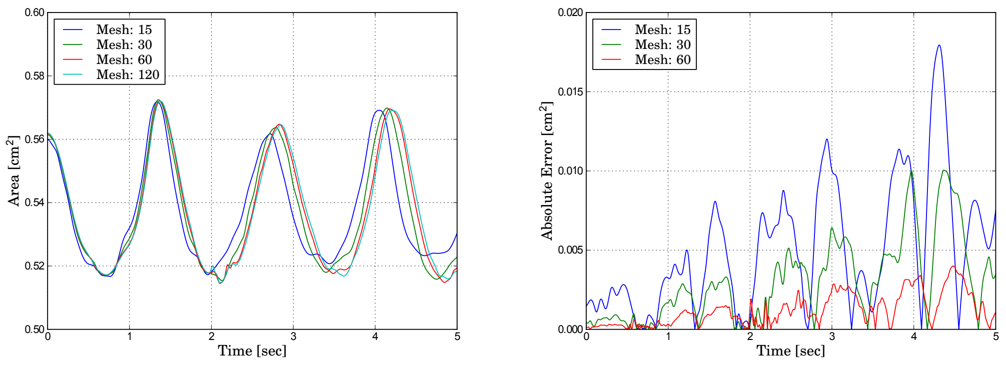

Figure 5.4: Convergence test results of spring model on membrane. The boundary of the membrane is fixed. The simulations are on the sequences with 15^3, 30^3, 60^3 and 120^3 mesh for the computational domain respectively. The total mass $M = Nm$ is a constant in the simulation. The left plot shows the total area during the four simulations and right plot shows the Cauchy errors of the sequences.

mesh size	e_l	e_k	e_p
50 and 100	0.00374	0.01664	0.03464
100 and 200	0.00184	0.00821	0.01726
200 and 400	0.000918	0.00406	0.008614

Table 5.2: Convergence tests of spring model for a swing chord. In the computational sequences, the total mass of the swing chord is fixed. As the number of points increases, the point mass is reduced accordingly. Cauchy error is calculated on two consecutive mesh sequences. Column e_l, e_k, and e_p are errors of total length, total kinetic energy and total spring potential energy respectively. The numerical results show the first order convergence on each of them.

mesh size	e_A	e_k	e_p
15 and 30	0.02528	1.91810	1.97967
30 and 60	0.01507	1.21784	1.21772
60 and 120	0.00604	0.53550	0.52837

Table 5.3: Convergence tests of spring model for a fabric drum. In the computational sequences, the total mass of the membrane is fixed. As the number of points increases, the point mass is reduced accordingly. Cauchy error is calculated on two consecutive mesh sequences. Column e_A, e_k, and e_p are errors of total area, total kinetic energy and total spring potential energy respective. The numerical results show the first order convergence on each of them.

where E is Young's modulus, F is the force exerted on the object under tension, A_0 is the original cross-sectional area through which the force is applied, Δl is the change of length of the object from the original length l_0 of the object.

Poisson's Ratio is the negative ratio of transverse strain to axial strain. When a material is stretched, it usually tends to contract in the directions transverse to the direction of stretching. The Poisson ratio is the ratio of relative contraction to relative stretching.

$$\nu = -\frac{d\epsilon_{\text{trans}}}{d\epsilon_{\text{axial}}} = -\frac{d\epsilon_y}{d\epsilon_x} \tag{5.2}$$

where ν is the Poisson's Ratio, ϵ_{trans} is transverse strain and ϵ_{axial} is axial strain. Theoretically, in the case of small deformations Poisson's Ratio ν was computed by Eq. (5.3) and in the case of large deformation it was computed by Eq. (5.4).

$$\nu = -\frac{\Delta w / w_0}{\Delta l / l_0} \tag{5.3}$$

$$\nu = -log_{(1+\Delta l/l_0)}(1 + \Delta w/w) \tag{5.4}$$

To verify that the spring model can catch isotropic elastic material's Young modulus and Poisson ratio, we carried out a set of simulations by stretching fabric surfaces with different Young's modulus and Poisson ratios. These simulations start with a fabric surface which has its original length $l_0 = 0.1m$ and original width $w_0 = 0.02m$. The fabric is then pulled along the direction of the longer side of it with a distributed force.

Firstly, three groups of simulations which stretch fabric surfaces with Poisson ratios of -0.14, -0.22, and -0.30, respectively are carried out. Each group with five different values of Young's modulus, that is $0.1GPa$, $0.2GPa$, $0.3GPa$, $0.4GPa$, and $0.5GPa$ respectively. The fabric surface is pulled at one end. Young's modulus and Poisson ratio are calculated from the deformation of the surface and the force added on it. The results are summarized by Table 5.4 which shows that the spring-mass model nicely reproduces the values of Young's modulus and the Poisson ratio from the input.

Next, a group of simulations, stretching the fabric surface whose Young's modulus and Poisson ratio are fixed at $0.5Gpa$ and -0.14 respectively, are carried out. The total number of triangles of the triangulations use in these simulations change from 1143, 1590, 2468, to 4520. The strain of the elongation change from 0.002, 0.004, 0.006, 0.008 to 0.01. The measured Young's modulus

Group 1	Young's Modulus results (GPa)		Poisson Ratio results	
	Input	Numerical	Input	Numerical
case 1	0.1	0.0963016606	-0.14	-0.141887
case 2	0.2	0.1926061554	-0.14	-0.141920
case 3	0.3	0.2888656609	-0.14	-0.141910
case 4	0.4	0.3852341348	-0.14	-0.141913
case 5	0.5	0.4815205552	-0.14	-0.141905
Group 2	Young's Modulus results (GPa)		Poisson Ratio results	
	Input	Numerical	Input	Numerical
case 1	0.1	0.0957361901	-0.22	-0.223342
case 2	0.2	0.1914918931	-0.22	-0.223332
case 3	0.3	0.2870650506	-0.22	-0.223382
case 4	0.4	0.3829895385	-0.22	-0.223349
case 5	0.5	0.4784196259	-0.22	-0.223365
Group 3	Young's Modulus results (GPa)		Poisson Ratio results	
	Input	Numerical	Input	Numerical
case 1	0.1	0.0953452395	-0.30	-0.305362
case 2	0.2	0.1906619377	-0.30	-0.305342
case 3	0.3	0.2860538596	-0.30	-0.305343
case 4	0.4	0.3814011184	-0.30	-0.305359
case 5	0.5	0.4767297379	-0.30	-0.305355

Table 5.4: Young's modulus and Poisson ratio verification

and Poisson ratio from these simulations are demonstrated in Fig. 5.5.

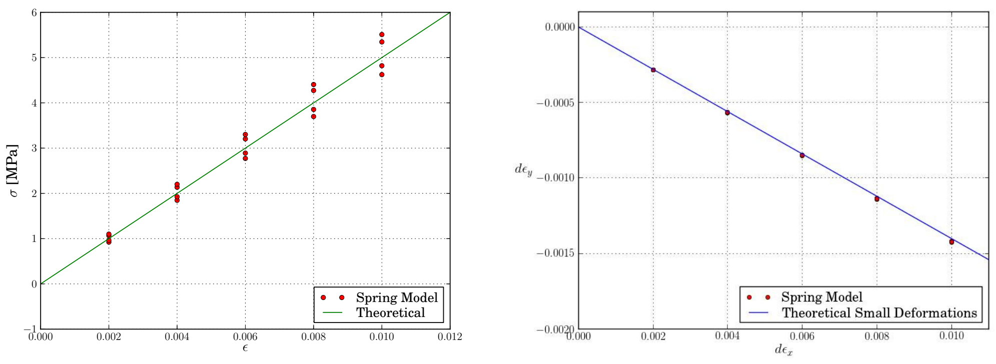

Figure 5.5: Young's Modulus (Left) and Poisson's Ratio (Right). We tested the spring model by stretching the fabric surface to different lengths. The numerical results show that the spring-mass model catches the fabric's Young's Modulus and Poisson ratio nicely in the linear regime of strain.

The numerical solutions imply that the revised spring-mass model based on the derivation by Delingette [19] can accurately simulate an isotropic elastical membrane in the linear region with strain up to 0.01. The error between the spring model and continuum model increases when elongation is too large and the deformation reaches the nonlinear regime. For parachute simulation, the strain of the canopy, even during the most dynamic phase of inflation, should still be in the linear regime. Therefore the spring model is an excellent model for such simulations.

It should be mentioned that in the case in which the fabric surface is compressed and it adjusts itself with wrinkles, the convergence is not obvious. In some cases, a small perturbation of the initial condition may result in substantially different folding and wrinkling patterns. How to describe such case and define its mathematical convergence remain as an open question to the model. Fig. 5.6 shows the draping of a table cloth due to gravitational force on a circular table. The right plot shows the wrinkled edge of the cloth.

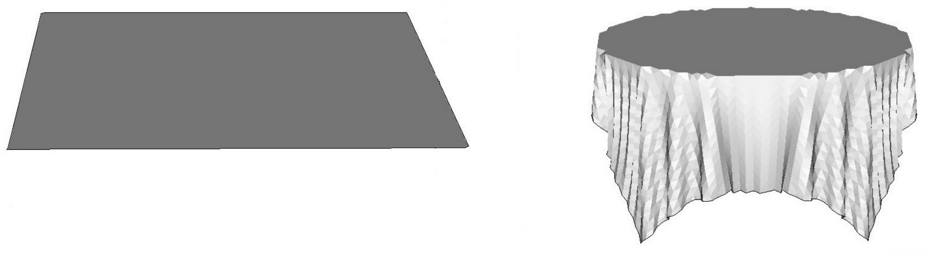

Figure 5.6: Simulation of a table cloth draping under the action of the gravity. The fabric constraint automatically adjusts the regions of the cloth. The spring model of the fabric gives a realistic motion of the cloth. The characteristic eigen frequency for the fabric model in this simulation is $\sqrt{k/m} = 1000$ and the friction constant is $\kappa = 0.1$.

5.3 Computation of Stress

The geometrical deformation of a fabric surface is the major source of stress on the material. The stress causes surface tension in parachute canopy and exerts a normal component of force which the parachute canopy interacts with the surrounding fluid to produce drag of the deceleration. The stress is also an important engineering variable in the parachute safety design. Therefore accurate computation of stress will not only help to understand the fluid dynamics of the parachute, but also to ensure the material will not be ripped apart during the deceleration. The stress has been computed on a rectangular and a triangular fabric surface being stretched from point forces at their corners. The magnitude of von-Mises stress on each individual triangle of the spring mesh is given in Fig. 5.7. In the rectangular fabric simulation, the total mass of the membrane is $13g$ and the spring constant is $1000N/m$. A total of 5461 mass points are in the triangulated mesh. The color plot shows that the largest stress is near the pulling corner while the central part is relatively non-stressed. The right plot shows the stress computed using Eq. (2.19). In the second case, the total mass is also $13(g)$ with 1123 mass points. Fig. 5.8 demonstrates that the advanced spring-mass model can capture the front of the shock wave clearly when a rectangular fabric surface was stretched along the horizontal axis. Fig. 5.9 shows the most stressful areas of a C-9 canopy during the inflation stage of the parachute. Here the peaks in stress are located at the canopy-string connection points.

5.4 Comparison of Drag in Inflation

Parachute inflation is one of the most crucial stages for the deceleration system. Both experimental data and numerical simulation results showed that there exists a time period during which the parachute canopy experiences a peaked drag force that can be as high as 3-4 times of the payload. Potvin recorded the drag force personally from his jumps. The experimental results (Fig. 5.10) reveal that the peak of the drag force climbed during the first second after the canopy deployment. The drag force reached about 700 lbs. at roughly $t = 1.3 - 1.5sec$). At this time the air from bottom opening

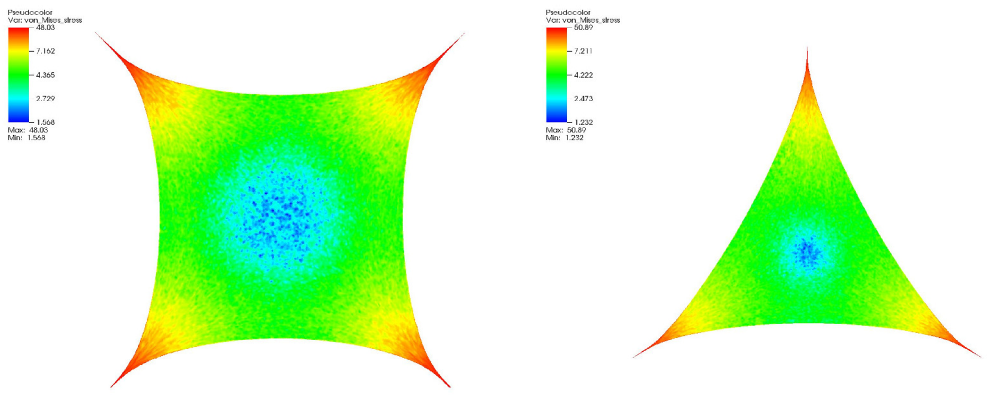

Figure 5.7: The von-Mises formula Eq. (2.19) is used to calculate the fabric stress in the spring model. The left plot shows the von-Mises stress of a rectangular membrane when pulled from the four corners. Similarly, the right plot shows the von-Mises stress of the triangular membrane pulled from its three vertices.

rushes to fill the volume of the canopy causing expansion in both vertical and horizontal directions. The ballooning canopy exerts the majority of the drag force force to all the stiffer chords. In the simulations, the parachute drag force is recorded as the sum of forces on each chord. The total force as a function of time is compared with the experimental data by Potvin. We have carried out simulations with different initial shapes of the canopy. The flat and partially closed canopies show similar peaked drag at $t = 1.2 - 1.6sec$, but the canopy with an angled initial condition demonstrated a peak at the later time of $t = 2sec$. Early in the simulations the drag force appears oscillatory for several seconds following the highest peak. The dynamic evolution of the

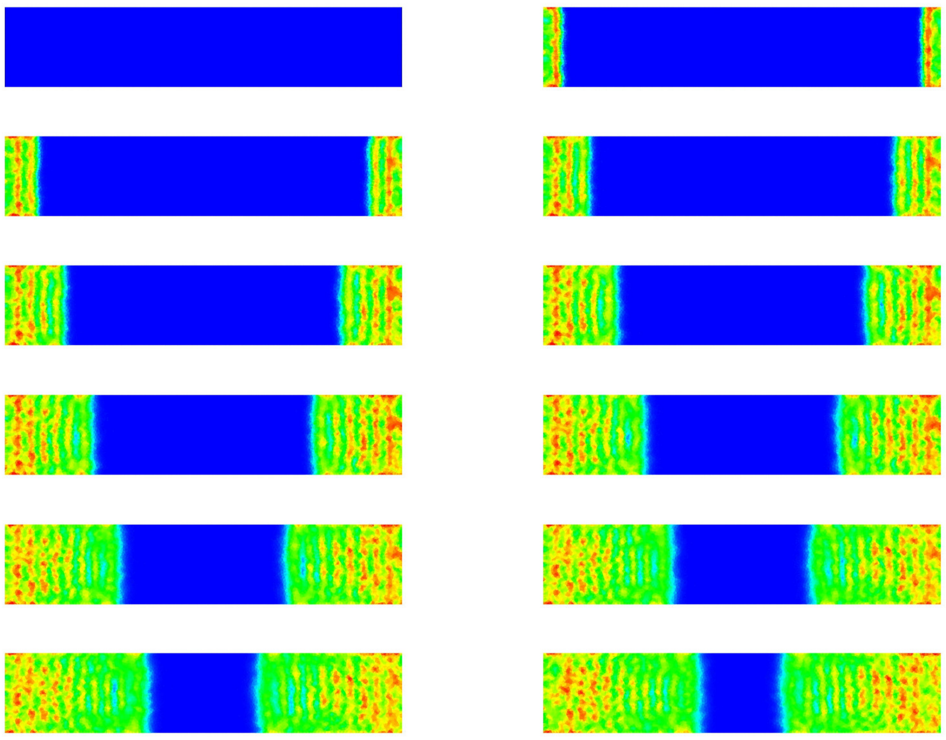

Figure 5.8: The von-Mises formula Eq. (2.19) is used to calculate the fabric stress in the spring model. The plots show the von-Mises stress of a rectangular membrane stretched along the horizontal axis. The shock wave has been captured clearly.

canopy geometry at these times corresponds to to the oscillatory motion of the horizontal motion of the string chord and the breathing motion of the canopy. Such oscillation is reduced when a transverse damping force is added to the string chord.

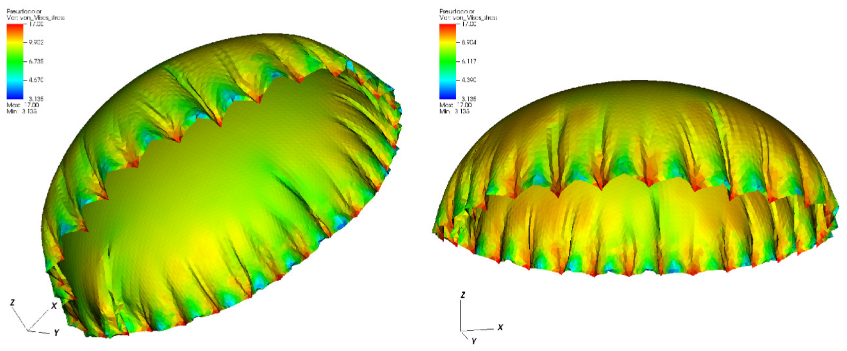

Figure 5.9: Von-Mises stress on the parachute canopy during its inflation. The red color shows regions with high stress. The figure shows that the areas near the canopy-string connection points are the most stressful part of the parachute during inflation.

5.5 Simulation of Angled Drop

The majority of parachute malfunctions occur during the inflation sequence. One of the most harmful malfunctions is the canopy "inversion" which occurs when one or more gore sections near the skirt of the canopy blows between the suspension lines on the opposite side of the parachute and then catches air [72]. That portion then forms a secondary lobe with the canopy inverted. The condition may work out or may become a complete inversion i.e. the canopy turns completely inside out [60]. Inversion during parachute inflation is dangerous as it can completely shut up the inlet of the canopy and prevent the creation of an air volume under the canopy, thus reduces the drag force to essentially zero and results in a free fall.

Numerical solution becomes a valuable tool for the parachute design if

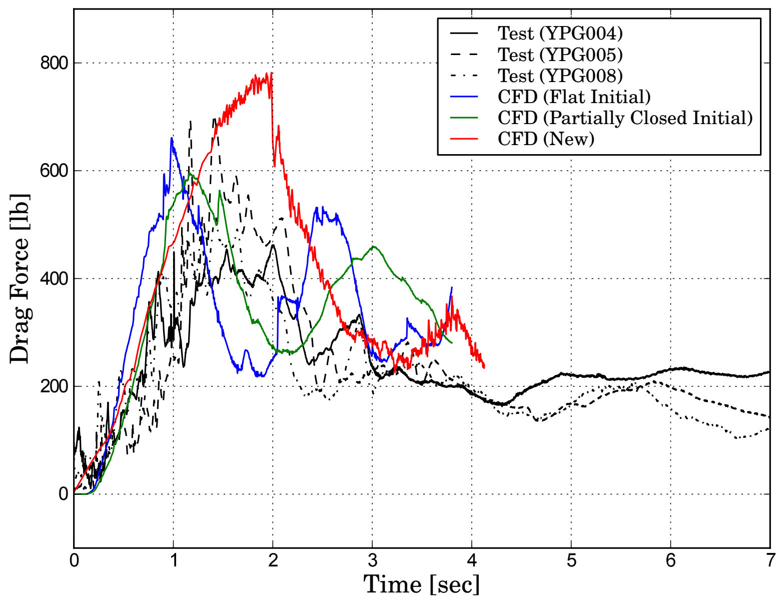

Figure 5.10: Drag force time history during the inflation phase of a C-9 personnel parachute. The experimental data is provided by Dr. Jean Potvin at St. Louis University.

computer simulation can reveal and predict malfunctions of the parachute canopy during the deployment. A group of different drops in which the initial alignments of the parachute form different angles with the direction of the fluid velocity are simulated here. Fig. 5.11 shows the case in which the alignment of canopy-string-payload forms a $15°$ angle with the fluid velocity. In this case, the canopy only slightly loses its symmetry during the inflation, but the inflation is normal. The total adjustment to vertical fall takes a longer time, but the opening of the canopy is just on time. In the case of the parachute forming a $60°$ angle with the flow, as shown in Fig. 5.12, the side of the parachute facing the flow is dented and wrapped up and the opening time is increased. In the case in which the parachute forms a 75° angle with the flow, the complete inversion of the canopy happens at approximately $t = 2.0sec$. Fig. 5.13 shows such inverted canopy.

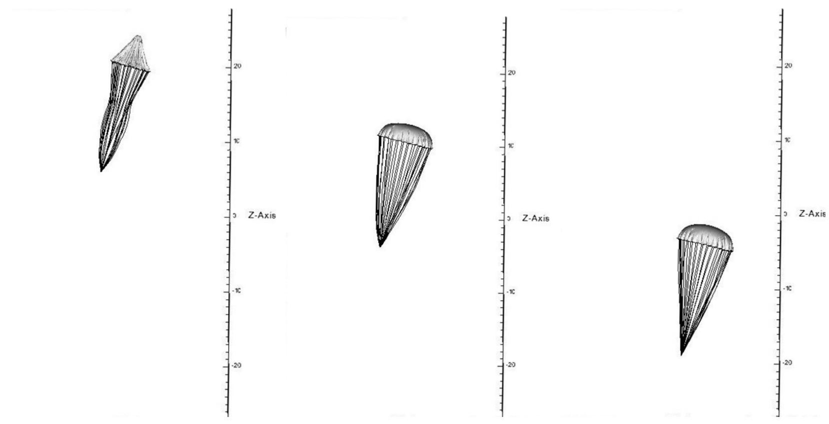

Figure 5.11: Angled deployment of C-9 parachute with the flow. The deployment starts with a 15° angle between the initial parachute and the direction of flow. The parachute experiences only slight asymmetry of the canopy. The plots show the parachute at (from left to right) $t = 0sec$, $t = 1.5sec$ and $t = 3.0sec$ respectively.

5.6 Enhancement of the Efficiency

5.6.1 Gas Dynamics Results

The test case for the Euler equations is the shock-tube problem. This problem is a well-known Riemann problem introduced by Sod [76]. The solu-

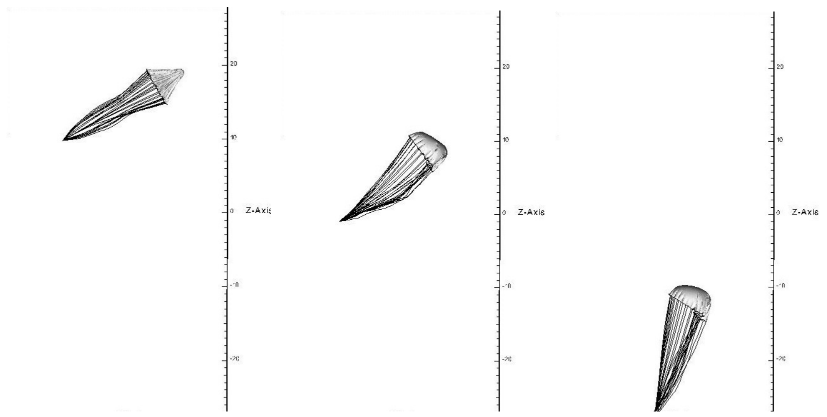

Figure 5.12: Angled deployment of C-9 parachute with the flow. This sequence starts with a 60° angle between the initial parachute and the direction of the flow. In this case, the canopy skirt is dangerously wrapped at the lower side of the canopy. The plots show the parachute at (from left to right) $t = 0sec$, $t = 1.5sec$ and $t = 3.0sec$ respectively.

tion domain is $[-1, 1]$, and the initial conditions are Eq. (5.5).

$$(\rho, u, p) = \begin{cases} (1, 0, 1), & x \leqslant 0 \\ (0.125, 0, 0.1), & x > 0 \end{cases} \tag{5.5}$$

The results are demonstrated in Fig. 5.14 and the operation time recorded by the CPU clock and time for GPU intensive computational part are collected in Table 5.5. Figure. 5.15 displays the GPU time and CPU time for each step. From Table 5.5 and Fig. 5.15 we can conclude that the application of the GPUs has clear advantage in the computation of the flux using the WENO scheme.

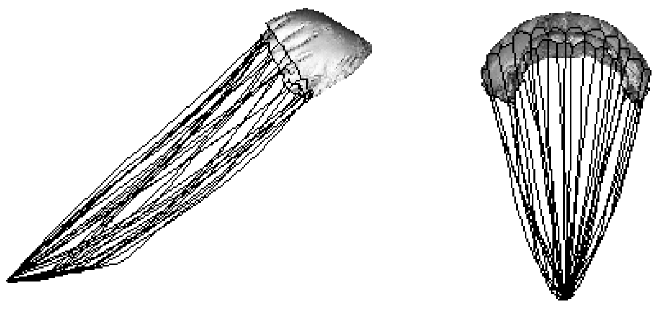

Figure 5.13: Inversion of the parachute canopy during an angled drop. The alignment of the parachute started with a 75° angle with the direction of the velocity. A complete inversion occurs at $t = 2sec$. The two plots are views of the inverted canopy from different directions.

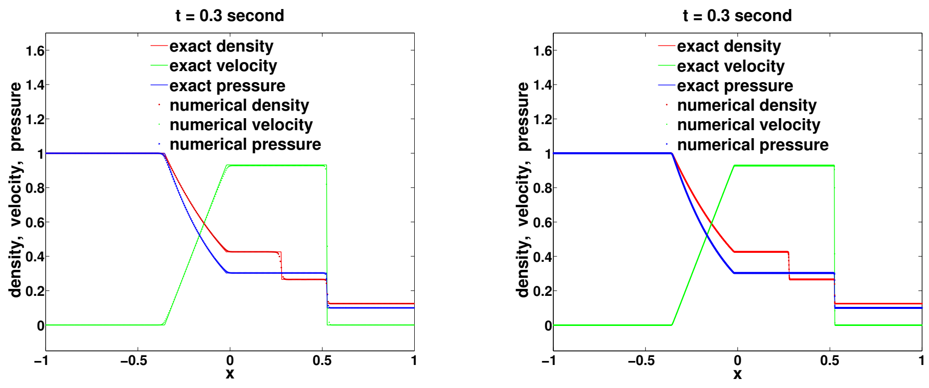

Figure 5.14: Sod problem results, the solution domain is $[-1, 1]$. The mesh size of the left figure is 400 while the right one's is 3200.

5.6.2 Spring Model Results

The test case of the spring model is stretching a rectangular fabric surface. In this simulation, the total mass of the membrane is $13g$ and the spring

Mesh	CPU Time ($micros$)	GPU Time ($micros$)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
1024	3427	336	118	272	726	29.0424	4.7204
2048	6896	349	146	273	768	46.9726	8.9297
3072	10053	365	152	283	800	65.6053	12.4650
4096	13912	340	183	365	888	75.3989	15.5383
5120	16774	411	215	321	947	77.4930	17.5935
6144	20126	397	249	308	954	80.2811	20.9539
7168	23463	525	271	396	1192	86.0996	19.5747
8192	25376	509	363	365	1237	69.5344	20.4050

Table 5.5: GPU and CPU computing time of solving one dimensional Euler equations by the fifth order WENO scheme in one step. Eight different mesh sizes (1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192) were tested with both pure CPU code and hybrid (CPU and GPU) code. Based on the analysis and experiments in section 4.4, we choose one of the best block size 128 here. The hybrid code is 8-20$\times$ faster than the pure CPU code for the computation of the intensive part when the mesh size is larger than 2048. In the table, "Time of copy data from Host to Device" and "Time of copy data from Device to Host" denoted by "H2D" and "D2H", respectively.

constant is $1000N/m$.

The total operation time recorded by CPU clock and time for GPU intensive computational part are collected in Table 5.6.

The left plot in Fig. 5.7 shows that the largest stress is near the pulling

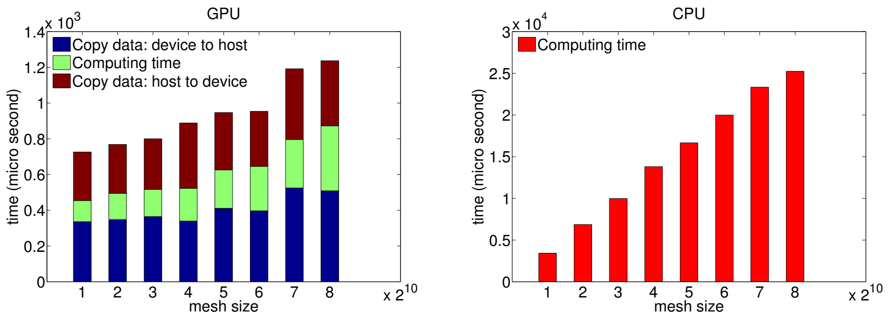

Figure 5.15: GPU (left) and CPU (right) time per step. The GPU computing time is 29-86× faster than CPU's. However, GPU total time is only 4-20× faster than CPU's. This is due to the time used to transfer the data between the host and the device in GPU. From the left plot, we can clearly see that the time spent on copying data is at least twice larger than the time spent on computing.

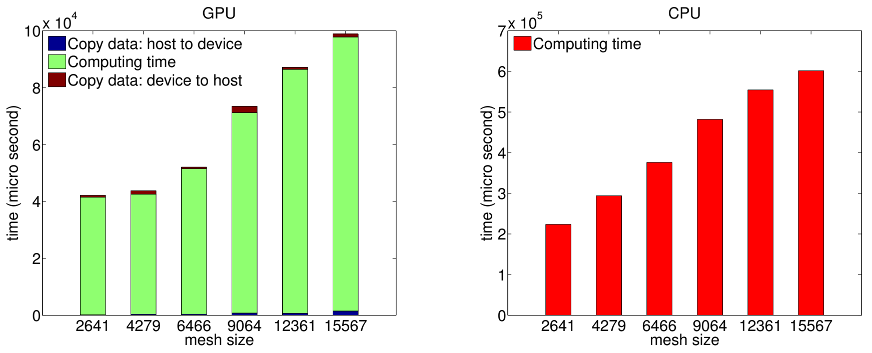

Figure 5.16: GPU (left) and CPU (right) time per step. The GPU computing time and total time are 5-6× faster than CPU's. This is due to the negligible time used to transfer the data between the host and the device.

corner while the central part is relatively non-stressed. This spring model has also been used to calculate the stress on a C-9 parachute canopy during

Mesh	CPU Time (*micros*)	GPU Time (*micros*)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
2641	223565	171	41328	644	42143	5.4095	5.3049
4279	294037	340	42200	1239	43734	6.9751	6.7233
6466	376137	376	51180	530	52086	7.3493	7.2215
9064	481941	786	70444	2241	73471	6.8415	6.5596
12361	554545	687	85744	727	87158	6.4674	6.3625
15567	601600	1503	96310	1066	98879	6.2465	6.0842

Table 5.6: GPU and CPU computing time of three dimensional spring model. Spring models with eight different mesh sizes of 2641, 4279, 6466, 9064, 12361, 15567 were tested with both pure CPU code and hybrid (CPU and GPU) code. The hybrid code is 5-6$\times$ faster than the pure CPU code for computing the intensive part.

inflation [75]. The right plot in Fig. 5.7 shows that the most stressful areas of a C-9 parachute canopy during the inflation stage located at the canopy-string connection points.

5.6.3 American Option Pricing Results

After several tests, we found that for single American option pricing the intensive integrations are the most time consuming part when mesh size is not too large. Fortunately, we can calculate these integrations in parallel using a GPU. The total operation time and time for intensive integrations measured in micro-seconds for computing single option with parameters $\tau = 1, r =$

$0.1, \sigma^2 = 0.4, K = 1$ are collected in Table 5.7.

Mesh	CPU Time ($micros$)	GPU Time ($micros$)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
128	942539	752740	540774	55	1293569	1.7429	0.7286
256	2825369	757925	1034025	59	1792009	2.7324	1.5766
512	8088534	757555	2124162	115	2881832	3.8079	2.8067
1024	22214927	780483	3867521	79	4648083	5.7440	4.7794
2048	55254249	810256	6475537	87	7285880	8.5328	7.5837
4096	134558431	755945	11294322	256	12050523	11.9138	11.1662

Table 5.7: Operation time of single option pricing under the generalized hyperbolic distribution, parameters $\lambda = 1.0, \alpha = 8.15, \beta = -2.5, \delta = 0.767$

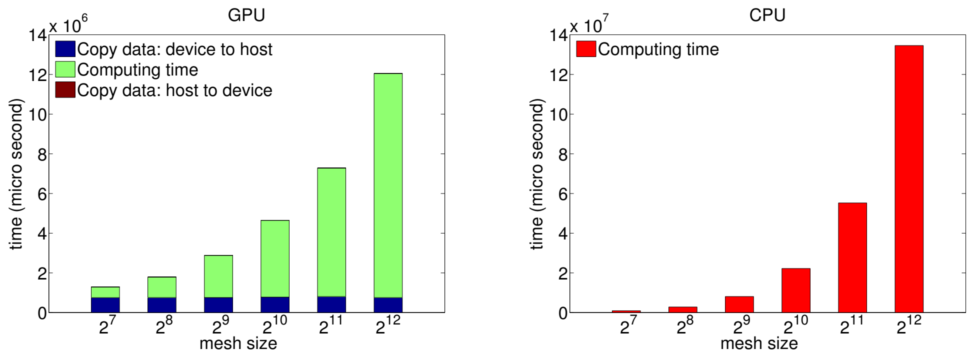

Figure 5.17: GPU (left) and CPU (right) time per step. The GPU computing time is 1.7-12× faster than CPU's. However, the performance of GPU is worse when the mesh size is small. Because the time used to transfer the data dominates GPU calculation time.

From Table 5.7 and Fig. 5.17 we can conclude that GPU has an obvious advantage in computing intensive integrations, which leads to less total operation time when the mesh size is relatively large. However, when the mesh size increases, the operation time of other parts, especially time iteration steps, increases gradually and undermines the effect of parallel computing. There is a trade-off between operation time and accuracy which depends on the mesh size. In this example, when we require lower accuracy and the mesh size is relatively small, CPU algorithm is the better choice; when higher accuracy is required and the mesh size is relatively large, CPU/GPU joint algorithm is the better choice. But this advantage of CPU/GPU joint application will decrease as the mesh becomes more refined.

To meet the requirements of timely and efficiently pricing of multiple options, we transform the CPU algorithm to joint CPU/GPU algorithm. Table 5.8 compares the operation time on pricing multiple options without GPU and with GPU respectively at a given mesh size of 128. As we can see, the operation time with GPU is relatively stable and is around 4 seconds when the number of options is below 2048. The ratio of CPU operation time and CPU/GPU joint operation time improves greatly as the number of options increases. In summary, the algorithm with GPU has an overwhelming advantage over the algorithm without GPU on pricing a large number of options.

Number of options	CPU Time ($micros$)	GPU Time ($micros$)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
32	35361760	2902	3892648	96	3895646	9.0842	9.07735
64	70723520	3484	3900757	324	3904565	18.1307	18.11303
128	141447040	2752	3954747	195	3957694	35.7664	35.73986
256	282894080	3199	3955152	382	3958733	71.5255	71.46086
512	565788160	3200	4097343	1429	4101972	138.0866	137.93087
1024	1131576320	3232	4366600	1064	4370896	259.1436	258.88896
2048	2263152640	3257	4767045	1945	4772247	474.7496	474.23218
4096	4526305280	3328	9540157	3352	9546837	474.4477	474.11579

Table 5.8: Operation time of multiple options pricing under the generalized hyperbolic distribution, parameters $\lambda = 1.0, \alpha = 8.15, \beta = -2.5, \delta = 0.767$

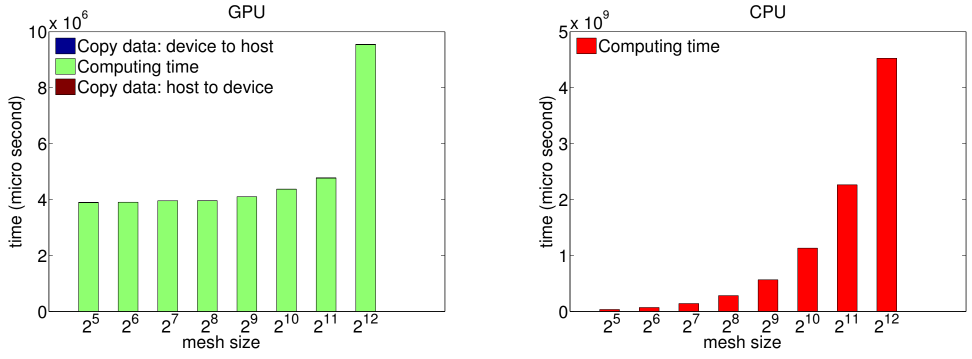

Figure 5.18: GPU (left) and CPU (right) time per step. The GPU computing time is relatively stable when the number of options is below 2048.

Chapter 6

Conclusions

A spring model for the simulation of fabric surface is discussed based on the modifications of Delingette. New components to the parachute module have been included including the distinction of gore boundary, more sophisticated collision handling and porosity. A dramatic computational speed up the spring model simulation is achieved through heterogeneous computations where the most expensive portion of calculation is performed by the GPU. Numerical experiments verified first order convergence of the spring model under the conditions that the total mass of the membrane is kept constant and that both the tensile stiffness and angular stiffness of the spring conform with the material's Young's modulus and the Poisson ratio. However, the convergence is weak in the case when the fabric is under compression and forms wrinkles.

The verification of the spring model on the material properties of the fabric membrane gives exciting results. Both Young's modulus and Poisson ratio agree with the theory excellently in the linear regime in which the relative displacement $\Delta l/l$ is less than 10 percent. The difference increases as it reaches the nonlinear regime in which the relative displacement exceeds 10 percent. However, since both parachute canopy and string chords are stiff materials and the relative stretches are both smaller than 10 percent even during the most dynamic inflation phase, the spring model is sufficient for the simulation of parachute deceleration system.

Our comparison of the inflation drag with the experimental data by Potvin shows agreement on the peak drag force at approximately $t = 1.2 - 1.6 sec$. However, simulations demonstrate oscillatory variation of the drag force, or after-shock. By adding transverse damping of string chord such oscillation can be reduced, but do not fully explain the discrepancies between experiment and simulations. Future work will focus on the effects due to porosity and the wake of parachutist body or payload. Parachute malfunctions during the inflation stage is dangerous and our numerical simulations show that parachute canopy can undergo inversion when the parachute is dropped with an angle exceeding 60° between the parachute and the free fluid stream velocity.

Bibliography

[1] R. M. Aileni, D. Farima, and M. Ciocoiu. Simulating cloth in realistic way using vertices model based. *7th International Conference-TEXSCI*, 2010.

[2] M. Aono, D. Breen, and M. Wozny. Fitting a woven cloth model to a curved surface: mapping algorithms. *Computer-Aided Design*, 26(4):278–292, April 1994.

[3] M. Aono, P. Denti, D. Breen, and M. Wozny. Fitting a woven cloth model to a curved surface: dart insertion. *IEEE Computer Graphics and Applications*, 16(5):60–70, September 1996.

[4] Ole Barndorff-Nielsen. Exponentially decreasing distributions for the logarithm of particle size. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 353(1674):401–419, 1977.

[5] J. B. Bell, P. Colella, and H. M. Glaz. An efficient sceond-order projection method for viscous incompressible flow. *Proceedings of the Tenth AIAA Computational Fluid Dynamics Conference*, AIAA:360, 1991.

[6] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.

[7] W. Bo, X. Liu, J. Glimm, and X. Li. Primary breakup of a high speed liquid jet. *ASME Journal of Fluids Engineering*, submitted, 2010.

[8] D. Breen. *A particle-based model for simulating the draping behavior of woven cloth.* PhD thesis, Rensselaer Polytechnic Institute, 1993.

[9] D. Breen, D. House, and M. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of ACM SIGGRAPH 94*, pages 365–372. ACM Press, 1994.

[10] D. Brown, R. Cortez, and M. Minion. Accurate projection method for the incompressible Navier Stokes equations. *J. Comp. Phys.*, 168:464–499, 2001.

[11] M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, pages 99–104. ACM Press, 1992.

[12] K.-J. Choi and H.-S. Ko. Stable but responsive cloth. *ACM Transactions on Graphics*, 21:604–611, 2002.

[13] A. J. Chorin. Numerical solution of the Navier Stokes equations. *Math. Comp*, 22:745–762, 1968.

[14] A. J. Chorin. On the convergence of discrete approximations to the Navier-Stokes equations. *Math. Comp*, 23:341, 1969.

[15] Cedric Cochrane, Maryline Lewandowski, and Vladan Koncar. A flexible strain sensor based on a conductive polymer composite for in situ measurement of parachute canopy deformation. *Sensors (14248220)*, 10:8291 – 8303, 2010.

[16] David J Cockrell and Alec David Young. The aerodynamcis of parachutes. Technical report, DTIC Document, 1987.

[17] Rama Cont and Peter Tankov. *Financial modelling with jump processes*, volume 2. CRC Press, 2004.

[18] John C Cox, Stephen A Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of financial Economics*, 7(3):229–263, 1979.

[19] Herve Delingette. Triangular springs for modeling nonlinear membranes. *IEEE Transactions on Visualization and Computer Graphics Volume 14 Issue 2*, pages 723–731, March 2008.

[20] Hamid J. Desabrais K.J. Aerodynamics of parachute opening. *ADA411095, ARO37594.6-EG. Mechanical Engineering Department, Worcester Polytechnic Institute*, pages 1–131, 2002.

[21] J. Du, B. Fix, J. Glimm, X. Jiao, X. Li, Y. Li, and L. Wu. A simple package for front tracking. *J. Comput. Phys.*, 213:613–628, 2006.

[22] Jian Du, Brian Fix, James Glimm, Xicheng Jia, Xiaolin Li, Yunhua Li, and Lingling Wu. A simple package for front tracking. *J. Comput. Phys.*, 213:613–628, 2006.

[23] S. Dutta, E. George, J. Glimm, J. Grove, H. Jin, T. Lee, X. Li, D. H. Sharp, K. Ye, Y. Yu, Y. Zhang, and M. Zhao. Shock wave interactions in spherical and perturbed spherical geometries. *Nonlinear Analysis*, 63:644–652, 2005. University at Stony Brook preprint number SB-AMS-04-09 and LANL report No. LA-UR-04-2989.

[24] S. Dutta, E. George, J. Glimm, X. L. Li, A. Marchese, Z. L. Xu, Y. M. Zhang, J. W. Grove, and D. H. Sharp. Numerical methods for the determination of mixing. *Laser and Particle Beams*, 21:437–442, 2003. LANL report No. LA-UR-02-1996.

[25] B. Eberhardt, M. Meißner, and W. Straßer. Knit fabrics. In D. House and D. Breen, editors, *Cloth Modeling and Animation*, pages 123–144. A.K. Peters, 2000.

[26] B. Eberhardt, A. Weber, and W. Straßer. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, September 1996.

[27] John A. Ekaterinaris. High-order accurate, low numerical diffusion methods for aerodynamics. *Progress in Aerospace Sciences*, 41(34):192 – 300, 2005.

[28] EG Ewing, HW Bixby, and TW Knacke. Recovery systems design guide. Technical report, DTIC Document, 1978.

[29] C. L. Gardner, J. Glimm, J. W. Grove, O. McBryan, R. Menikoff, D. H. Sharp, and Q. Zhang. A study of chaos and mixing in Rayleigh-Taylor and Richtmyer-Meshkov unstable interfaces. In M. Duong-van and B. Nichols, editors, *Proceedings of the International Conference on 'The Physics of Chaos and Systems Far from Equilibrium' (CHAOS' 87), Monterey, CA, USA, Jan. 11–14, 1987*. 1988. Special Issue of Nuclear Physics B (proceedings supplements section).

[30] A. Van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *J. Graphics Tools*, 3(2):21–41, March 1998.

[31] E. George, J. Glimm, X. L. Li, Y. H. Li, and X. F. Liu. The influence of scale-breaking phenomena on turbulent mixing rates. *Phys. Rev. E*, 73:016304, 2006.

[32] E. George, J. Glimm, X. L. Li, A. Marchese, and Z. L. Xu. A comparison of experimental, theoretical, and numerical simulation Rayleigh-Taylor mixing rates. *Proc. National Academy of Sci.*, 99:2587–2592, 2002.

[33] James M. Gere. *Mechanics of materials, sixth edition.* Bill Stenquist, 2004.

[34] J. Glimm, X.-L. Li, R. Menikoff, D. H. Sharp, and Q. Zhang. A numerical study of bubble interactions in Rayleigh-Taylor instability for compressible fluids. *Phys. Fluids A*, 2(11):2046–2054, 1990.

[35] J. Glimm, X. L. Li, W. Oh, A. Marchese, M.-N. Kim, R. Samulyak, and C. Tzanos. Jet breakup and spray formation in a diesel engine. In *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics*. Cambridge, MA, 2003. SUNY Stony Brook preprint No. susb-ams-02-20.

[36] J. Glimm, O. McBryan, R. Menikoff, and D. Sharp. Front tracking applied to Rayleigh-Taylor instability. *SIAM J. Sci. Stat. Comput.*, 7:230–251, 1986.

[37] James Glimm, M.-N. Kim, X.-L. Li, R. Samulyak, and Z.-L. Xu. Jet simulation in a diesel engine. In *Computational Fluid and Solid Mechanics 2005*. Elsevier Science, 2005.

[38] Katuhiko Goda. A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows. *J. Comput. Phys.*, 30:76–95, 1979.

[39] S.-W. Hsiao and R.-Q. Chen. A method of drawing cloth patterns with fabric behavior. *5th WSEAS International Conference on Applied Computer Science*, pages 635–640, 2006.

[40] A. H. Barr J. C. Platt. Constraints methods for flexible models. In *SIGGRAPH '88 Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 279–288, 1988.

[41] F. Ji, R. Li, and Y. Qiu. Simulate the dynamic draping behavior of woven and knitted fabrics. *J. Industrial Textiles*, 35:201–214, 2006.

[42] G. Jiang and C.-W. Shu. Efficient implementation of weighted ENO schemes. *J. Comput. Phys.*, 126:202–228, 1996.

[43] Guang-Shan Jiang and Chi-Wang Shu. Efficient implementation of weighted ENO schemes. *Journal of Computational Physics*, 126(1):202 – 228, 1996.

[44] Vinay Kalro and Tayfun E. Tezduyar. A parallel 3D computational method for fluid-structure interactions in parachute systems. *Computer Methods in Applied Mechanics and Engineering*, 190:321–332, 2000.

[45] K. Karagiozis, R. Kamakoti, F. Cirak, and C. Pantano. A computational study of supersonic disk-gap-band parachutes using large-eddy simulation coupled to a structural membrane. *Journal of Fluids and Structures*, 27(2):175–192, 2011.

[46] S. Kawabata. The standardization and analysis of hand evaluation. *J. Text. Mach. Soc. Japan*, 1980.

[47] J. Kim and P. Moin. Application of a fractional-step method to incompressible Navier-Stokes equations. *J. Comput. Phys.*, 59:308, 1985.

[48] J.-D. Kim, Y. Li, and X.-L. Li. Simulation of parachute FSI using the front tracking method. *Journal of Fluids and Structures*, 37:101–119, 2013.

[49] Y. Kim and C. S. Peskin. 2-D parachute simulation by the immersed boundary method. *SIAM J. Sci. Comput.*, 28:2294–2312, 2006.

[50] Y. Kim and C. S. Peskin. 3-D parachute simulation by the immersed boundary method. *Comput. Fluids*, 38:1080–1090, 2009.

[51] Young Shin Kim, Svetlozar T Rachev, Michele Leonardo Bianchi, and Frank J Fabozzi. Financial market models with lévy processes and time-varying volatility. *Journal of Banking & Finance*, 32(7):1363–1378, 2008.

[52] David B Kirk and W Hwu Wen-mei. *Programming massively parallel processors: a hands-on approach*. Morgan Kaufmann, 2010.

[53] Theo W Knacke. Parachute recovery systems design manual. Technical report, DTIC Document, 1991.

[54] AC Knoell. Alaa 2nd aerodynamic deceleration systems conference. 1968.

[55] Hans Petter Langtangen, Kent-Andre Mardal, and Ragnar Winther. Numerical methods for incompressible viscous flow. *Advances in Water Resources*, 25(8):1125–1146, 2002.

[56] Long Lee and Randall J LeVeque. An immersed interface method for incompressible navier–stokes equations. *SIAM Journal on Scientific Computing*, 25(3):832–856, 2003.

[57] Y. Li, I-Liang Chern, J.-D. Kim, and X.-L. Li. Numerical method of fabric dynamics using front tracking and spring model. *Communications in Computational Physics*, 14(5):1228–1251, 2013.

[58] Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200 – 212, 1994.

[59] Abhijit Majumdar. *Soft computing in textile engineering*. Elsevier, 2010.

[60] Jr. Manley C.Butler and Michael D.Crowe. The design, development and testing of parachutes using the bat sombrero slider. *15th CEAS/AIAA Aerodynamic Decelerator Systems Technology Conference*, 1999.

[61] Maciej Matyka. Solution to two-dimensional incompressible navier-stokes equations with simple, simpler and vorticity-stream function approaches. driven-lid cavity problem: Solution and visualization. *arXiv preprint physics/0407002*, 2004.

[62] Randall C Maydew, Carl W Peterson, and Kazimierz J Orlik-Rueckemann. Design and testing of high-performance parachutes (la conception et les essais des parachutes a hautes performances). Technical report, DTIC Document, 1991.

[63] Robert C Merton, Michael J Brennan, and Eduardo S Schwartz. The valuation of american put options. *The Journal of Finance*, 32(2):449–462, 1977.

[64] Paulius Micikevicius. 3d finite difference computation on gpus using cuda. pages 79–84, 2009.

[65] Carl W. Peterson. The fluid physics of parachute inflation. *Physics Today*, 46:32, 1993.

[66] J. Potvin. Parachute inflation. *McGraw-Hill Yearbook of Science and Technology*, 1998.

[67] J. Potvin, K. Bergeron, G. Brown, R. Charles, K. Desabrais, H. Johari, V. Kumar, M. McQuilling, A. Morris, G. Noetscher, and B. Tutt. The road ahead: A white paper on the development, testing and use of advanced numerical modeling for aerodynamic decelerator system design and analysis. *AIAA paper 2011-2501*, May 2011.

[68] Jean Potvin and Mark McQuilling. The bi-model: Using cfd in simulations of slowly-inflating low-porosity hemispherical parachutes. *AIAA paper 2011-2542*, May 2011.

[69] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proceedings of Graphics Interface (GI 1995)*, pages 147–154. Canadian Computer-Human Communications Society, 1995.

[70] J. W. Purvis. Prediction of line sail during lines-first deployment. *AIAA 21st Aerospace Sciences Meeting*, 1983.

[71] J. W. Purvis. Numerical prediction of deployment, initial fill, and inflation of parachute canopies. *8th AIAA Aerodynamic Decelerator and Balloon Technology Conference*, 1984.

[72] Douglas S.Adams. Lessons learned and flight experience from planetary parachute development. *7th International Planetary Probe Workshop (IPPW7)*, 2010.

[73] R. Samulyak, T. Lu, and P. Parks. A hydromagnetic simulation of pellet ablation in electrostatic approximation. *Nuclear Fusion*, 47:103–118, 2007.

[74] R. Samulyak, T. Lu, P. Parks, J. Glimm, and X. Li. Simulation of pellet ablation for tokamak fuelling with itaps front tracking. *Journal of Physics: Conf. Series*, 125:012081, 2008.

[75] Qiangqiang Shi, Daniel Reasor, Zheng Gao, Xiaolin Li, and Richard D. Charles. On the verification and validation of a spring fabric for medeling parachute inflation. *Submitted to Journal of Fluids and Structures*, 2014.

[76] Gary A Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1 – 31, 1978.

[77] GA Solt Jr. Performance of and design criteria for deployable aerodynamic decelerators. *US Air Force Flight Dynamics Lab Report, ASD-TR-61-579*, page 357, 1963.

[78] K. Stein, R. Benney, V. Kalro, T. E. Tezduyar, J. Leonard, and M. Accorsi. Parachute fluid-structure interactions: 3-D computation. *Comput. Methods Appl. Mech. Engrg*, 190:373–386, 2000.

[79] K. Stein, R. Benney, T. Tezduyar, and J. Potvin. Fluid-structure interactions of a cross parachute: numerical simulation. *Computer Methods in Applied Mechanics and Engineering*, 191(6-7):673–687, 2001.

[80] K. Stein, T. Tezduyar, V. Kumar, S. Sathe, R. Benney, E. Thornburg, C. Kyle, and T. Nonoshita. Aerodynamic interactions between parachute canopies. *J. Appl. Mech.*, 70:50–57, 2003.

[81] K. R. Stein, R. J. Benney, V. Kalro, A. A. Johnson, and T. E. Tezduyar. Parallel computation of parachute fluid-structure interactions. *14th Aerodynamic Decelerator Systems Technology Conference*, 1997.

[82] K. R. Stein, R. J. Benney, E. C. Steeves, Development U.S. Army Natick Research, and Engineering Center. *A computational model that couples aerodynamic and structural dynamic behavior of parachutes during the opening process.* Technical report (U.S. Army Natick Laboratories). United States Army Natick Research, Development and Engineering Center, Aero-Mechanical Engineering Directorate, 1993.

[83] K. R. Stein, R. J. Benney, T. E. Tezduyar, J. W. Leonard, and M. L. Accorsi. Fluid-structure interactions of a round parachute: Modeling and simulation techniques. *J. Aircraft*, 38:800–808, 2001.

[84] J. H. Strickland, V. L. Porter, G. F. Homicz, and A. A. Gossler. Fluid-structure coupling for lightweight flexible bodies. *17th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, 2003.

[85] K. Takizawa, C. Moorman, S. Wright, T. Spielman, and T. E. Tezduyar. Fluid-structure interaction modeling and performance analysis of the orion spacecraft parachutes. *International Journal for Numerical Methods in Fluids*, 65:271–285, 2011.

[86] Kenji Takizawa, Timothy Spielman, and Tayfun E. Tezduyar. Space-time FSI modeling and dynamical analysis of spacecraft parachutes and parachute clusters. *Computational Mechanics*, 48:345–364, 2011.

[87] Eric Yu Tau. A second-order projection method for the incompressible navier-stokes equations in arbitrary domains. *Journal of Computational Physics*, 115(1):147–152, 1994.

[88] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, December 1988.

[89] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscoelasticity, plasticity, fracture. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, pages 269–278. ACM Press, July 1988.

[90] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, pages 205–214. ACM Press, July 1987.

[91] T. E. Tezduyar, S. Sathe, R. Keedy, and K. Stein. Space-time finite element techniques for computation of fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 195:2002–2027, 2006.

[92] T. E. Tezduyar, S. Sathe, M. Schwaab, J. Pausewang, J. Christopher, and J. Crabtree. Fluid-structure interaction modeling of ringsail parachutes. *Computational Mechanics*, 43:133–142, 2008.

[93] T. E. Tezduyar, K. Takizawa, C. Moorman, S. Wright, and J. Christopher. Space-time finite element computation of complex fluid-structure interactions. *International Journal for Numerical Methods in Fluids*, 64:1201–1218, 2010.

[94] Tayfun E. Tezduyar and Sunil Sathe. Modelling of fluid-structure interactions with the space-time finite elements: Solution techniques. *International Journal for Numerical Methods in Fluids*, 54(6-8):855–900, 2007.

[95] B. Tutt, S. Roland, R. D. Charles, and G. Noetscher. Finite mass simulation techniques in LS-DYNA. *21st AIAA Aerodynamic Decelerator Systems Technology conference and Seminar*, 2011.

[96] B. A. Tutt. The application of a new material porosity algorithm for parachute analysis. *9th International LS-DYNA Users Conference*, 2006.

[97] B. A. Tutt and A. P. Taylor. The use of LS-DYNA to simulate the inflation of a parachute canopy. *18st AIAA Aerodynamic Decelerator Systems Technology conference and Seminar*, 2005.

[98] J Van Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM Journal on Scientific and Statistical Computing*, 7(3):870–891, 1986.

[99] P. Volino, M. Courchesne, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of ACM SIGGRAPH 95*, pages 137–144. ACM Press, 1995.

[100] Yiyang Yang, Qiangqiang Shi, and Xiaolin Li. A gpu enhanced numerical algorithm for american option pricing under generalized hyperbolic distribution. *Submitted to applied numerical mathmatics*, 2013.

[101] CAO YIHUA, SONG QIANFU, WU ZHUO, and JOHN SHERIDAN. Flow field and topological analysis of hemispherical parachute in low angles of attack. *Modern Physics Letters B*, 24:1707 – 1725, 2010.

[102] Li Yu and Xiao Ming. Study on transient aerodynamic characteristics of parachute opening process. *Acta Mechanica Sinica*, 23:627–633, 2007.

[103] Peter Zaspel and Michael Griebel. Solving incompressible two-phase flows on multi-gpu clusters. *Computers & Fluids*, 80(0):356 – 364, 2013.