Efficient Numerical Algorithms for Heterogeneous Computation of PDE Extended Systems with Applications

A Dissertation Presented

by

Yiyang Yang

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

May 2015

Stony Brook University

The Graduate School

Yiyang Yang

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

Svetlozar Rachev - Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

Yuefan Deng - Member
Professor, Department of Applied Mathematics and Statistics

Anthony Phillips - Outside Member
Professor, Department of Mathematics

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Efficient Numerical Algorithms for Heterogeneous Computation of PDE Extended Systems with Applications

by

Yiyang Yang

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2015

Partial Differential Equations (PDEs) and corresponding numerical schemes are explored to simulate scientific and engineering problems including parachute simulation and American option pricing. These problems involve appropriate coupling of several equations systems. A revised spring-mass model is used to describe the motion of parachute canopy and string motion which considers both string stiffness and angular stiffness. This model is validated by the material's Young's modulus and Poisson ratio and is proved to be convergent to continuum mechanics. The Navier-Stokes equation is applied to simulate the fluid field and a second-order accurate numerical scheme is used, together with the introduction of the concept "penetration ratio" to simulate fabric porosity which has great impact on the drag performance of the parachute. A partial-integro differential equation based on generalized hyperbolic distribution is built to simulate the price of American option pricing after coupling certain free boundary condition to describe early exercise property.

Due to the complex nature of above applications and the corresponding numerical scheme structure, Graphics Processing Unit (GPU) is introduced to derive efficient heterogeneous computing algorithms. The most computationally intensive and parallelizable parts of the application are identified and accelerated greatly based on the single-instruction multiple data (SIMD) architecture. During the parallelization process, parallel execution, memory hierarchy and instruction usage are optimized to maximize parallelization effect. For the spring-mass system, we achieved $6\times$ speedup and greatly improved the parachute simulation efficiency. The system of one-dimensional gas dynamics equations is solved by the Weighted Essentially Non-Oscillatory (WENO) scheme; the heterogeneous algorithm is 7-$20\times$ faster than the pure CPU based algorithm. For single American option, the numerical integrations are parallelized at grid level and $2\times$ speedup is realized; for multiple option pricing, each thread is in charge of one option and the algorithm reaches $400\times$ speedup.

Key Words: partial differential equations (PDEs), parachute simulation, American option pricing, front tracking, GPU

To my Parents and all loving ones

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to my adviser, Pr. Xiaolin Li. He is the best advisor, who gave me inspiring suggestions on research and patient guidance towards my PhD degree; He is my family, who support me for every decision I made and we enjoy family time together during Thanks-giving and Christmas; He is a good friend, with whom I can share any happiness and sorrow. I feel so proud to be his student and enjoy all the four-year time spent with him.

I really appreciate the encouragement and valuable technical suggestions from Pr. Svetlozar Rachev, Pr. Yuefan Deng and Pr. Anthony Phillips. You are the most outstanding researchers in your field. I feel so excited and grateful to have you in my defense committee.

Many thanks to my colleagues, Dr. JoungDong Kim, Dr. Yan Li, Dr. Yijing Hu, Dr. Qiangqiang Shi, Zheng Gao, Xiaolei Chen, Saurabh Joglekar, Jingfang Qu and Muye Chen. Thanks to Pr. Li, we shared many happy hours every Friday.

During my PhD career, I got a lot support from my parents, family members and close friends. I want to dedicate this thesis to all the people I love and these people knowing I am not perfect but still love me.

Chapter 1

Partial Differential Equations

The research of partial differential equation (abbreviated PDE) dates back to 18th century when Euler, d'Alembert, Lagrange and Laplace used it as a central tool in the description of mechanics of continua and the study of models in physical science [15]. Since the middle of the 19th Century, due to the contribution of Riemann and other scientists, PDE became an essential tool in other branches of mathematics. This duality of viewpoint has been central to the analysis of PDE until today.

Linear PDE is discussed most in textbooks and usually have analytical solutions when coupled with certain boundary condition. However, nonlinear PDE is more important within real world problems and current mainstream research [18].

Nowadays, more complicated partial differential equations or differential equation systems are developed to describe problems in various fields. These equations are almost unable to be solved by analytical methods. At the same time, more advanced computing technique and resources motivated us to solve all these problems numerically. Large scale simulation enables us to get more intuitive idea about the target problem and save great experimental cost.

The complex nature of many practical problem, together with the advanced computing technique and improvement of hardware drive us to explore heterogeneous computing algorithm to solve target problem more efficiently.

1.1 Basic Definitions and Classifications

A *partial differential equation* [64, 32] is an equation involving a function u of several variables and its partial derivatives. For a function $u(x_1, \cdots, x_n)$, the corresponding PDE has the form

$$F(x_1, \cdots, x_n, u_{x_1}, \cdots, u_{x_n}, u_{x_1 x_1}, u_{x_1 x_2}, \cdots) = 0. \tag{1.1}$$

The *order* of the above equation is the order of the highest derivative occurring in the equation. PDE can be classified according to *linearity*:

1. linear: the equation depends linearly on u and its derivatives;
2. semilinear: all derivatives of u occur linearly with coefficients depending only on x;
3. quasilinear: all highest-order derivatives of u occur linearly with coefficients depending only on x, u and lower-order derivative of u.

A simple example of a first-order PDE is

$$u_t + a(u)\,u_x = 0, \tag{1.2}$$

when $a(u) = a$ is a constant, it is a linear equation called *transport equation*; when $a(u) = u$, it is a quasilinear equation called *inviscid Burgers equation*.

After introducing the concept of *Laplacian* or *Laplace operator*

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \cdots + \frac{\partial^2}{\partial x_n^2}, \tag{1.3}$$

we can discuss several second-order PDE

$$u_t = k\Delta u, \tag{1.4}$$

$$u_{tt} = c^2 \Delta u, \tag{1.5}$$

$$\Delta u = 0. \tag{1.6}$$

Eq. (1.4) is called *heat equation* which represents the diffusion of heat through an n-dimensional body; Eq. (1.5) is *wave equation* which represents surface wave when $n = 2$ and sound or light wave when $n = 3$; Eq. (1.6) is n-dimensional Laplace equation. Eq. (1.4) and Eq. (1.5) are *evolutionary equations* because they describe phenomena that may change with time; Eq. (1.6) is satisfied by the *steady-state* solutions of Eq. (1.4) and Eq. (1.5) which is time independent.

In order for a PDE to have *unique* solution, additional *side conditions* should be imposed. Side conditions are usually in the form of *initial conditions*, for example

$$u(x, t_0) = g(x), \tag{1.7}$$

or *boundary conditions*, for example

$$u(x_0, t) = h(t), \tag{1.8}$$

or *combination* of the two.

Eq. (1.4)-Eq. (1.6) are *homogeneous* equations which means if $\phi(x,t)$ is the solution of the PDE, then $k\phi(x,t)$ is also the solution of the PDE. To obtain corresponding *nonhomogeneous* equation, we add function f

$$u_t - k\Delta u = f, \tag{1.9}$$

$$u_{tt} - c^2\Delta u = f, \tag{1.10}$$

$$\Delta u = f. \tag{1.11}$$

In Eq. (1.9) and Eq. (1.10), f can be considered as heat source and external forcing term respectively which lead to nonhomogeneous heat and wave equations. Eq. (1.11) is nonhomogeneous Laplace equation or *Poisson equation.*

Second-order equations are most widely used in many different area. In the case when the derivatives of second-order all occur linearly, the PDE can be represented as

$$a(x,y)\,u_{xx} + b(x,y)\,u_{xy} + c(x,y)\,u_{yy} = d(x,y,u,u_x,u_y) \tag{1.12}$$

and the corresponding characteristic curve is

$$\frac{dy}{dx} = \frac{b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.13}$$

which have three cases:

1. *Hyperbolic* equation, $b^2 > 4ac$, there are *two* characteristics;
2. *Parabolic* equation, $b^2 = 4ac$, there is *only one* characteristic;
3. *Elliptic* equation, $b^2 < 4ac$, there are *no* characteristics.

1.2 PDEs in Science and Engineering

In this section, we discuss three most typical PDEs: wave equation, Laplace equation and heat equation. These PDEs have analytical solutions when coupled with certain side conditions. Even though the forms are quite simple, they can explain many physical and mechanical phenomenon and are the corner stones for understanding more complicated PDE system.

1.2.1 Wave Equation

First, consider *transport* or *advection equation*, which is a special case of wave equation, coupled with *initial boundary condition*

$$\begin{cases} u_t + au_x = 0 \\ u\left(x,0\right) = h\left(x\right) \end{cases}, \tag{1.14}$$

where u is a function of two variables (x,t) and a is a constant. Based on the method of characteristics, the solution can be derived as

$$u(x,t) = h(x - at). \tag{1.15}$$

A more general case is the one-dimensional *wave equation*

$$u_{tt} - c^2 u_{xx} = 0 \tag{1.16}$$

governs wave motion in a string and the *propagation speed* c is a constant. Assume $\mu = x + ct, \eta = x - ct$ transforms Eq. (1.16) to

$$u_{\mu\eta} = 0. \tag{1.17}$$

The general solution of Eq. (1.17) is $u(\mu,\eta) = F(\mu) + G(\eta)$ which gives the equivalent solution

$$u(x,t) = F(x + ct) + G(x - ct). \tag{1.18}$$

Given initial conditions

$$u(x,0) = g(x), \;\; u_t(x,0) = h(x) \tag{1.19}$$

where g and h are arbitrary functions. The d'Alembert's formula for the solution of the initial problem is

$$u(x,t) = \frac{1}{2}\left[g(x+ct) + g(x-ct)\right] + \frac{1}{2c}\int_{x-ct}^{x+ct} h(\epsilon)\, d\epsilon. \tag{1.20}$$

For multiple-dimensional wave equations, that is $n > 1$, they can be solved by mean reduction to the one-dimensional case and use above process to obtain corresponding solution.

Wave equation applies perfectly to model air pollution, dye dispersion or traffic flow with u representing the *density* of the pollutant, dye or traffic at position x and time t [55], as well as the propagation of light and sound.

1.2.2 Laplace Equation

The *Laplace equation*

$$\Delta u = 0,\ in\ \Omega, \tag{1.21}$$

and the corresponding nonhomogeneous equation, *Poisson equation*

$$\Delta u = f,\ in\ \Omega \tag{1.22}$$

are frequently used in the physical science to describe steady-state behavior.

To solve the equation and obtain unique solutions, appropriate *boundary condition* should be coupled:

1. *Dirichlet boundary condition*, state on the boundary $\partial\Omega$ is defined

$$u(x) = h(x),\ x \in \partial\Omega; \tag{1.23}$$

2. *Neumann boundary condition*, normal derivative on the boundary layer $\partial\Omega$ is
$$\frac{\partial u(x)}{\partial \vec{n}} = \vec{n} \cdot \nabla u(x) = h(x), \; x \in \partial\Omega \tag{1.24}$$

3. *Robin boundary condition*, arise naturally from the physical consideration of heat flow
$$\frac{\partial u(x)}{\partial \vec{n}} + \alpha u(x) = h(x), \; x \in \partial\Omega \tag{1.25}$$
where α is constant values and $\vec{n}$ denotes the normal direction to $\partial\Omega$.

For specific domain, separation of variables method can be used to find solution satisfy the boundary condition. Finding Green's function is another typical method to find the solution of Laplace equation. Elliptic equations can be applied to simulate steady irrotational flows. It can also be used to model static electric, magnetic, gravitational and fluid velocity field.

1.2.3 Heat Equation

Regard a physical body with constant heat conductivity k as a bounded region Ω in domain R^n, then the *heat* or *diffusion equation* is

$$u_t = k\Delta u, \; x \in \Omega, \; t > 0 \tag{1.26}$$

where $u(x,t)$ represents the temperature of the body at point x and time t. This equation describes the *propagation* or *diffusion* of heat.

The appropriate side conditions are initial condition which describes the initial temperature:

$$u\left(x,0\right)=g\left(x\right) \tag{1.27}$$

and boundary conditions:

1. *Dirichlet boundary condition*, temperature is controlled on the boundary $\partial\Omega$
$$u\left(x,t\right)=h\left(x,t\right),\ x\in\partial\Omega,\ t>0; \tag{1.28}$$

2. *Neumann boundary condition*, heat flow through boundary $\partial\Omega$ is controlled
$$\frac{\partial u\left(x,t\right)}{\partial\vec{n}}=h\left(x,t\right),\ x\in\partial\Omega,\ t>0 \tag{1.29}$$
where $\vec{n}$ denotes the normal direction to $\partial\Omega$;

3. *Robin boundary condition*, which governs heat flow by Newton's law of cooling
$$\frac{\partial u\left(x,t\right)}{\partial\vec{n}}+\alpha u\left(x,t\right)=h\left(x,t\right),\ x\in\partial\Omega,\ t>0 \tag{1.30}$$
where α is constant values and $\vec{n}$ denotes the normal direction to $\partial\Omega$.

The solution of an initial/boundary heat equation can be obtained using eigenfunctions of the Laplacian; the solution of the pure initial value problem can be derived using Fourier transformation in R^n. Given the pure initial value problem as

$$\begin{cases} u_t=\Delta u,\ t>0,\ x\in R^n \\ u\left(x,0\right)=g\left(x\right),\ x\in R^n \end{cases}, \tag{1.31}$$

if $g(x)$ is bounded and continuous for $x \in R^n$, then

$$u(x,t) = \frac{1}{(4\pi t)^{\frac{n}{2}}} \int_{R^n} e^{-\frac{|x-y|^2}{4t}} g(y)\, dy. \tag{1.32}$$

The temperature $u(x,t)$ changes over time as heat spreads throughout space. The heat equation is used to determine the change in $u(x,t)$. The rate of change of $u(x,t)$ is proportional to the "curvature" of $u(x,t)$. Thus, the sharper the corner, the faster it is rounded off. Over time, the tendency is for peaks to be eroded and valleys to be filled in.

Heat equation has very wide applications. It can be used to simulate particle diffusion process, or the random trajectory of a single particle subject to the particle diffusion equation which is known as Brownian motion. As the name indicated, a direct application is to predict the thermal transfer profiles and the measurement of the thermal diffusivity in polymers. It is also used in financial mathematics to model the price of options and the Black-Scholes equation can be transformed into heat equation. There are also applications in the field of image processing and machine learning.

1.3 Modeling based on PDE extended System

To simulate real world system and realize applications in various fields, much more complicated PDE or even PDE systems should be derived. Ordinary differential equation (abbreviated ODE) or ODE systems are sometimes coupled together to facilitate simulations under certain assumptions. Another extended case is the partial-integro differential equation arise in the financial modeling. In this section, several PDE based or extended models are introduced, together with corresponding applications.

1.3.1 Euler Equations and Navier-Stokes Equations

Euler equations and Navier-Stokes equations are the most important formulas to govern the motion of fluid. Euler equations first appeared in Euler's article in 1757. Navier-Stokes equations were originally derived in the 1840s on the basis of conservation laws and first-order approximations. In the early 1900s, assuming sufficient randomness in microscopic molecular process, they are derived from molecular dynamics [100]. The basic method to derive the both formula is through principles of conservation of mass, momentum and energy [39, 58].

The *conservation of mass* law requires

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \tag{1.33}$$

where $\rho(\vec{x}, t)$ is spatial mass density field and $\vec{u}(\vec{x}, t)$ is associated spatial velocity field.

The *conservation of linear momentum* law requires

$$\rho \left[\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} \right] = \nabla \cdot \mathbf{S} + \rho \vec{b} \tag{1.34}$$

where $\mathbf{S}(\vec{x}, t)$ is the Cauchy stress field and $\vec{b}(\vec{x}, t)$ is the spatial body force per unit mass.

The *conservation of angular momentum* law require that

$$\mathbf{S}^T = \mathbf{S}. \tag{1.35}$$

The *conservation of energy* law requires

$$\rho\left[\frac{\partial \phi}{\partial t} + (\vec{u}\cdot\nabla)\,\phi\right] = \nabla\left(\mathbf{S}\cdot\vec{u}\right) - \nabla\cdot\vec{q} + \rho r \tag{1.36}$$

where $\phi(\vec{x}, t)$ is the internal energy field per unit mass, $\vec{q}(\vec{x}, t)$ is the Fourier-Stokes heat flux vector field, $r(\vec{x}, t)$ is the heat supply field per unit mass.

Different choices of stress fields will lead to equation for different type of fluid. In the world of *ideal fluid* which is inviscid, the Cauchy stress field is given as

$$\mathbf{S}(\vec{x}, t) = -p(\vec{x}, t)\,\mathbf{I}. \tag{1.37}$$

In the case of absent extra body force, *Euler equation* can be derived as

$$\begin{cases} \frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\vec{u}) = 0 \\ \frac{\partial \vec{u}}{\partial t} + (\vec{u}\cdot\nabla)\,\vec{u} = -\frac{1}{\rho}\nabla p \\ \frac{\partial E}{\partial t} + \nabla\cdot\left(\vec{u}\,(E+p)\right) = 0 \end{cases} \cdot \tag{1.38}$$

where ρ is the fluid density, $\vec{u}$ is the velocity of the fluid field, p is the pressure, $E = \rho e + \frac{1}{2}\rho\vec{u}^T\vec{u}$ is the total energy density with e being the specific internal energy per unit mass. Due to the model assumption, Euler equation is most appropriate for simulating compressible inviscid fluid which has large Reynolds number, such as bullet ejection, spacecraft entrance and so on.

In the incompressible *Newtonian fluid* case, mass density field is uniform which means it is constant ρ_0 in the field. Thus, Eq. (1.33) implies

$$\nabla \cdot \vec{u} = 0. \tag{1.39}$$

The Newtonian Cauchy stress field is given as

$$\mathbf{S}(\vec{x}, t) = -p(\vec{x}, t)\,\mathbf{I} + \nu\left(\nabla\vec{u}(\vec{x}, t) + \nabla\vec{u}(\vec{x}, t)^T\right). \tag{1.40}$$

In the case of absent extra body force, *Navier-Stokes equation* for incompressible Newtonian fluid can be derived as

$$\begin{cases} \nabla \cdot \vec{u} = 0 \\ \rho_0\left[\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\,\vec{u}\right] = -\nabla p + \nu \nabla^2 \vec{u} \end{cases} \tag{1.41}$$

where ν is the kinematic viscosity and ρ_0 is the fluid density, $\vec{u}$ is the velocity of the fluid field, p is the pressure. Due to the incompressible and viscous assumption, Navier-Stokes equation has wide applications in physics, geoscience, aerospace and even medical simulation when coupled with certain boundary conditions and other appropriate constraints.

Both Euler equation and Navier-Stokes equation are nonlinear PDE. They aim at describing the fluid motion during evolution. They both have first-order term which governs the advection behavior. However, Euler equation does not have diffusion term due to the lack of viscosity. They both can demonstrate the motion under extra body force by introducing nonhomogeneous source terms. Initial condition specifies the initial state of the fluid field and boundary conditions specifies the behavior of fluid near interface. To realize good simulation result, different initial and boundary conditions should be set according to the requirement of the problem:

1. Inlet boundary conditions: the distribution of all flow variables are specified at *inlet boundaries* mainly flow velocity.

2. Outlet boundary conditions: the distribution of all flow variables are specified at *outlet boundaries* mainly flow velocity.

3. Wall boundary conditions: used to the simulate fluid near solid regions. In viscous flows, no-slip conditions are enforced at wall

   $$\vec{u}_{normal} = \vec{u}_{Wall} \tag{1.42}$$

   $$\vec{u}_{tangential} = 0 \tag{1.43}$$

   which means tangential fluid velocity equal to wall velocity, i.e. "no-slip", and normal velocity is set to be zero.

4. Constant pressure boundary conditions: used when boundary values of pressure are known and the exact details of the flow distributions are unknown. This usually includes pressure inlet and outlet conditions.

5. Periodic boundary conditions: used when physical geometry of interest and expected flow pattern and the thermal solution are of *periodically*

repeating nature.

1.3.2 Spring-Mass Model

ODE system are usually coupled with PDE systems in real application. In this section, an ODE system derived from spring mass model is introduced to simulate fabric motion, such as skin, soft tissue, paper and textile.

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. The fabric surface is represented by a high-quality triangulated mesh. Each mesh point is a mass node and the connection between them are springs. Assume 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. Based on Newton's second law, an ODE system can be derived as

$$m\frac{d\dot{\mathbf{x}}_i}{dt} = F_i \tag{1.44}$$

where F_i is the force from the spring system on mass i. Simple spring model only considers force due to spring elongation and compression. Van Gelder argued [34] 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. Recent publication by Delingette [27] proposed a revision to add angular stiffness. Thus, the force in Eq. (1.44) considers not only tensile stiffness due to elongation, but also angular stiffness due to resistance of angular deformation.

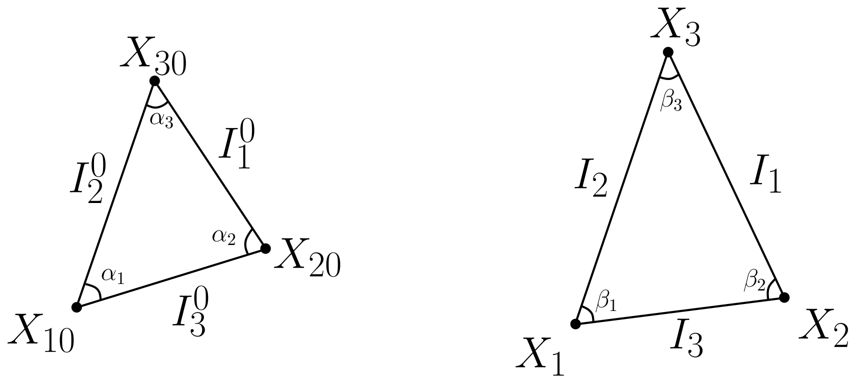


Figure 1.1: (a) Rest triangle $T_{\mathbf{X}_0}$ whose vertices are $\mathbf{X}_{i0}$. (b) Deformed triangle $T_{\mathbf{X}}$ whose vertices are $\mathbf{X}_i$.

As illustrated in Fig. 1.1, for a triangle in equilibrium $T_{\mathbf{X}_0}$, the initials states are given by area $A_{\mathbf{X}_0}$, angles α_i, and lengths $l_i^0, i = 1, 2, 3$ 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 as

$$W\left(T_{\mathbf{X}_0}\right) = \sum_{i=1}^{3} \frac{1}{2} k_i^{T_{\mathbf{X}_0}} \left(dl_i\right)^2 + \sum_{\substack{i=1 \\ j=(i+1)\,mod3 \\ k=(i+2)\,mod3}}^{3} \gamma_i^{T_{\mathbf{X}_0}} dl_j dl_k \tag{1.45}$$

where $k_i^{T_{\mathbf{X}_0}}$ and $\gamma_i^{T_{\mathbf{X}_0}}$ are tensile and angular stiffness respectively

$$k_i^{T_{\mathbf{X}_0}} = \frac{\left(l_i^0\right)^2 \left(2\cot^2\alpha_i \cdot (\lambda + \mu) + \mu\right)}{8A_{\mathbf{X}_0}} \tag{1.46}$$

$$\gamma_i^{T_{\mathbf{X}_0}} = \frac{l_j^0 l_k^0 \left(2 \cot \alpha_j \cot \alpha_{ik} \cdot (\lambda + \mu) - \mu\right)}{8A_{\mathbf{X}_0}} \tag{1.47}$$

with λ and μ as the Lame coefficients of the material.

Apply Rayleigh-Ritz method to analyze the fabric surface, which states that system evolve by minimizing its membrane energy, the force can be represented as

$$\mathbf{F}_i = \sum_{j=1}^{N} \eta_{ij} \mathbf{F}_{ij} \tag{1.48}$$

and

$$\mathbf{F}_{ij} = \left(\tilde{k}_{ij} dl_{ij} + \tilde{\gamma}_{ij} dl_{ij}\right) \mathbf{e}_{ij} \tag{1.49}$$

where

$$\tilde{k}_{ij} = k_{ij}^{T_1} + k_{ij}^{T_2},\ \tilde{\gamma}_{ij} = \frac{\gamma_i^{T_1} dl_{im} + \gamma_j^{T_1} dl_{jm} + \gamma_i^{T_2} dl_{in} + \gamma_j^{T_2} dl_{jn}}{dl_{ij}} \tag{1.50}$$

and $\mathbf{e}_{ij}$ is the unit vector from $\mathbf{X}_i$ to $\mathbf{X}_j$.

Eq. (1.44) is a second-order ODE, which can also be written as a first-order ODE system

$$\begin{cases} \frac{d}{dt}\mathbf{x}_i = \mathbf{u}_i \\ \frac{d}{dt}\mathbf{u}_i = \mathbf{F}_i \end{cases} \tag{1.51}$$

which is linear and nonhomogeneous.

1.3.3 Partial-Integro Differential Equation

A seminal application of heat equation is to simulate option price which gives the well-known *Black-Scholes Equation* [8]

$$\frac{\partial V}{\partial t}(t,S) + rS\frac{\partial V}{\partial t}(t,S) + \frac{\sigma^2 S^2}{2}\frac{\partial V}{\partial t}(t,S) - rV(t,S) = 0 \tag{1.52}$$

where $V(t,S)$ gives the option price at time t and underlying asset price S, r is the risk-free rate, σ is the volatility of underlying asset. Given the property of heat equation, the value of option smooths over time. The most important assumption is that the underlying asset S follows geometric Brownian motion

$$\frac{dS_t}{S_t} = \alpha dt + \sigma dW_t. \tag{1.53}$$

This diffusion process contains Brownian motion $(W_t)_{t\geq 0}$. Eq. (1.53) implies that the log return of underlying asset satisfies normal distribution.

Empirical market observation demonstrates that normal distribution which fails to capture skewness and asymmetry properties of underlying asset [52]. A possible solution is to use exponential Lévy model to simulate the price of the underlying asset which represents the extreme returns as discontinuities in the prices [22].

Lévy process is a right-continuity and left-limits stochastic process with values in R^d such that $X_0 = 0$ and satisfies following properties:

1. Independent increments: for every increasing sequence of times $t_0, \cdots, t_n$, the random variables $X_{t_0}, X_{t_1} - X_{t_0}, \cdots, X_{t_n} - X_{t_{n-1}}$ are independent;

2. Stationary increments: the law of $X_{t+h} - X_t$ does not depend on t;

3. Stochastic continuity: $\forall \epsilon > 0$, $\lim_{h \to 0} P\left(|X_{t+h} - X_t| \geq \epsilon\right) = 0$.

Based on *Lévy-Khinchin* theorem, any Lévy process on R^d can be uniquely represented by characteristic triplet (A, ν, γ) through characteristic function

$$E\left[e^{izX_t}\right] = e^{t\psi(z)},\ z \in R^d \tag{1.54}$$

$$\psi(z) = -\frac{1}{2} z^T A z + i\gamma^T z + \int_{R^d} \left(e^{iz^T x} - 1 - iz^T x 1_{|x| \leq 1}\right) \nu(dx). \tag{1.55}$$

The form of characteristic function implies that *infinitely divisible* distributions can be used to construct Lévy process. There are some research focusing on pricing options under exponential Lévy process assumption. Barndorff [6] proposes the Lévy model under normal inverse Gaussian process. Madan, Carr and Chang [60] price European options under exponential variance gamma process. In this paper, exponential Lévy process is constructed with *generalized hyperbolic distribution* which has density function

$$f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu) = a(\lambda, \alpha, \beta, \delta) \left[\delta^2 + (x - \mu)^2\right]^{\frac{\left(\lambda - \frac{1}{2}\right)}{2}}$$

$$\times K_{\lambda - \frac{1}{2}} \left(\alpha \sqrt{\delta^2 + (x - \mu)^2}\right) \exp\left[\beta (x - \mu)\right] \tag{1.56}$$

where

$$a(\lambda, \alpha, \beta, \delta) = \frac{\left(\alpha^2 - \beta^2\right)^{\frac{\lambda}{2}}}{\sqrt{2\pi} \alpha^{\lambda - \frac{1}{2}} \delta^\lambda K_\lambda \left(\delta \sqrt{\alpha^2 - \beta^2}\right)} \tag{1.57}$$

and $K_\lambda(\cdot)$ is the modified Bessel function of the third kind

$$K_\lambda(z) = \frac{1}{2}\int_0^\infty y^{\lambda-1}\exp\left[-\frac{z}{2}\left(y+\frac{1}{y}\right)\right]dy. \tag{1.58}$$

The density function depends on five parameters: $\lambda \in \mathbb{R}$ characterizes certain sub-classes of generalized hyperbolic distribution; $\alpha > 0$ determines the shape; $0 \le |\beta| < \alpha$ represents the skewness; $\delta > 0$ is the scaling parameter; $\mu \in \mathbb{R}$ indicates the location. As illustrated in Fig. 1.2, the shape of probability density function is quite flexible and can better describe the distribution of market return. The corresponding Lévy measure $\nu(x)$ is

$$\nu(x) = \begin{cases} \frac{e^{\beta x}}{|x|}\left(\int_0^\infty \frac{\exp\left(-\sqrt{2y+\alpha^2}|x|\right)}{\pi^2 y\left[J_\lambda^2\left(\delta\sqrt{2y}\right)+Y_\lambda^2\left(\delta\sqrt{2y}\right)\right]}dy + \lambda e^{-\alpha|x|}\right) & \lambda \ge 0 \\ \frac{e^{\beta x}}{|x|}\int_0^\infty \frac{\exp\left(-\sqrt{2y+\alpha^2}|x|\right)}{\pi^2 y\left[J_{-\lambda}^2\left(\delta\sqrt{2y}\right)+Y_{-\lambda}^2\left(\delta\sqrt{2y}\right)\right]}dy & \lambda < 0 \end{cases}, \tag{1.59}$$

where $J_\lambda(\cdot)$ and $Y_\lambda(\cdot)$ are the Bessel functions of the first and second kind respectively.

Assume the price of a financial asset is the exponential of a Lévy process

$$S_t = S_0 e^{rt+X_t} \tag{1.60}$$

where X_t is a Lévy process with Lévy triplet (σ, ν, γ) and interest rate r. Then, the discounted price of underlying asset $e^{-rt}S_t = S_0 e^{X_t}$ is a martingale if and only if

$$\int_{|y|>1} \nu(dy)\, e^y < \infty, \tag{1.61}$$

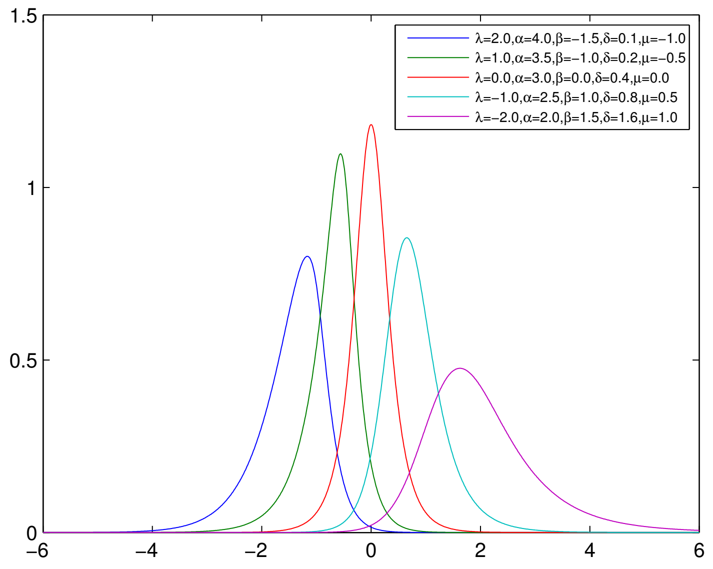


Figure 1.2: Density function of generalized hyperbolic distribution with different parameters. From left to right: (1) $\lambda = 2.0$, $\alpha = 4.0$, $\beta = -1.5$, $\delta = 0.1$, $\mu = -1.0$; (2) $\lambda = 1.0$, $\alpha = 4.0$, $\beta = -1.5$, $\delta = 0.2$, $\mu = -0.5$; (3) $\lambda = 0.0$, $\alpha = 3.0$, $\beta = 0.0$, $\delta = 0.4$, $\mu = 0.0$; (4) $\lambda = -1.0$, $\alpha = 2.5$, $\beta = 1.0$, $\delta = 0.8$, $\mu = 0.5$; (5) $\lambda = -2.0$, $\alpha = 2.0$, $\beta = 1.5$, $\delta = 1.6$, $\mu = 1.0$.

$$\gamma + \frac{\sigma^2}{2} + \int \left(e^y - 1 - y1_{|y|\leq 1}\right) \nu \left(dy\right) = 0. \tag{1.62}$$

Based on the equivalent martingale measure transformation, the price of the option can be represented as

$$V_t = E\left[e^{-r(T-t)} H\left(S_T\right) | \mathcal{F}_t\right] \tag{1.63}$$

where $H\left(S_T\right)$ is the payoff of the option at maturity. The Partial-Integro differential equation (abbreviated PIDE) can be obtained directly from applying the Ito's formula

$$\frac{\partial V}{\partial t}(t,S) + rS\frac{\partial V}{\partial S}(t,S) + \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2}(t,S) - rV(t,S)$$

$$+\int \nu(dy)\left[V(t,Se^y) - V(t,S) - S(e^y-1)\frac{\partial V}{\partial S}(t,S)\right] = 0. \tag{1.64}$$

Assume $f(\tau,x) = P(t,S),\ \tau = T-t$ and $x = \ln\frac{S}{S_0}$, denote

$$\mathcal{L}f(\tau,x) = f_\tau(\tau,x) - \left(r - \frac{\sigma^2}{2}\right)f_x(\tau,x) - \frac{\sigma^2}{2}f_{xx}(\tau,x) + rf(\tau,x)$$

$$-\int \nu(dy)\left[f(\tau,x+y) - f(\tau,x) - (e^y-1)f_x(\tau,x)\right], \tag{1.65}$$

where r is the risk-free rate, σ^2 is the volatility of underlying asset, S_0 is the initial price of underlying asset and τ denotes time to maturity.

This PIDE is a linear and evolutionary equation which has both first-order advection term and second order diffusion term. As a result, given the initial condition, the solution will smooth over time. The integral term makes this formula nonhomogeneous and can be regarded as an extra source term. After coupling appropriate initial and boundary conditions, it can be applied to price different types of options:

1. European put option:

$$\mathcal{L}f(\tau,x) = 0, \tag{1.66}$$

$$f(0,x) = \max\left[K - S_0e^x, 0\right], \tag{1.67}$$

$$f(\tau, x) = \max\left[Ke^{-r\tau} - S, 0\right], \ x \to \infty, \tag{1.68}$$

where K is the strike price of the option.

2. American put option:

$$\mathcal{L}f(\tau, x) = 0, \ \tau > 0, x > x_f(\tau) \tag{1.69}$$

$$f(\tau, x) = K - S_0 e^x, \ \tau > 0, x \leq x_f(\tau) \tag{1.70}$$

$$f(\tau, x) \geq \max\left[K - S_0 e^x, 0\right], \ \tau > 0 \tag{1.71}$$

$$\mathcal{L}f(\tau, x) \geq 0, \ \tau > 0 \tag{1.72}$$

$$f(0, x) = \max\left[K - S_0 e^x, 0\right] \tag{1.73}$$

where

$$x_f(\tau) = \inf\left\{x \in R \,|\, f(\tau, x) > \max\left[K - S_0 e^x, 0\right]\right\} \tag{1.74}$$

is the transformed free boundary and K is strike price of the option.

Chapter 2

Numerical Scheme and Application Implementation

With more advanced computational techniques, it is possible to derive numerical results of PDEs which have no analytical solution or too complicated to be solved analytically. In this chapter, general numerical schemes for classic PDE forms are discussed, together with the treatment of different boundary conditions. Then, three major applications and corresponding numerical schemes are illustrated in detail. The first case is in the computational fluid dynamics area. The numerical scheme for solving Euler equation and Navier-Stokes equation are given, as well as the special treatment of boundary condition to simulate porous boundary case. The second case is to solve spring mass model numerically. Through appropriate coupling, they can be used to simulate the interaction between fluid and interface which lead to a lot of applications in different area, and in this particular case, we focus on the simulation of parachute deployment process. The third application is to price American option numerically which solves PIDE with an efficient algorithm.

2.1 General Numerical PDE Schemes

General finite difference schemes for different PDEs are discussed concisely in this section, together with the treatment of different boundary conditions [94]. This should be the foundation for solving more complicated PDEs and PDE extended systems numerically.

2.1.1 Wave Equation

The simplest wave equation is the first order advection equation

$$u_t + a u_x = 0 \tag{2.1}$$

where a is the wave speed and $x \in (-\infty, \infty)$. Assume $R = a\frac{\Delta t}{\Delta x}$, some typical numerical schemes and corresponding accuracy and stability conditions are:

1. Upwind Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + 1_{a>0} \cdot a\frac{u_j^n - u_{j-1}^n}{\Delta x} + 1_{a<0} \cdot \frac{u_{j+1}^n - u_j^n}{\Delta x} = 0 \tag{2.2}$$

   this scheme is $O(\Delta x, \Delta t)$; stability requires $-1 \le R \le 0$ for $a < 0$ and $0 \le R \le 1$ for $a > 0$.

2. Central Explicit Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} = 0 \tag{2.3}$$

   this scheme is $O(\Delta x^2, \Delta t)$ and unconditionally unstable.

3. Central Implicit Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\Delta x} = 0 \tag{2.4}$$

this scheme is $O\left(\Delta x^2, \Delta t\right)$ and unconditionally stable.

4. Lax-Wendroff Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} - \frac{1}{2}a^2\Delta t\left(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}\right) = 0 \tag{2.5}$$

this scheme is $O\left(\Delta x^2, \Delta t\right)$ and stability requires $|R| \leq 1$.

2.1.2 Heat Equation

For parabolic diffusion equation

$$u_t = bu_{xx} \tag{2.6}$$

where $b > 0$ and assume $r = b\frac{\Delta t}{\Delta x^2}$. Some typical numerical schemes and corresponding accuracy and stability conditions are:

1. Central Explicit Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = b\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2} \tag{2.7}$$

this scheme is $O\left(\Delta x^2, \Delta t\right)$ and stability requires $r < \frac{1}{2}$.

2. Central Implicit Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = b\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2} \tag{2.8}$$

this scheme is $O\left(\Delta x^2, \Delta t\right)$ and uncondionally stability.

3. Crank-Nicolson Scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{b}{2}\left(\frac{u_{j+1}^{n} - 2u_j^{n} + u_{j-1}^{n}}{\Delta x^2} + \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2}\right) \tag{2.9}$$

this scheme is $O\left(\Delta x^2, \Delta t^2\right)$ and uncondionally stability.

For implicit and Crank-Nicolson scheme, a *tridiagonal system of equations* should be solved

$$Ax = d \tag{2.10}$$

where coefficient matrix A is a tridiagonal matrix and d is a vector

$$A = \begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{pmatrix}, d = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}. \tag{2.11}$$

Thomas or *double-sweep* algorithm can be used to solve the equation in $O(8n)$ operations.

2.1.3 Poisson Equation

Consider two dimensional Poisson's equation, assume $u(x,t)$ is the exact solution

$$\triangle u = u_{xx} + u_{yy} = f(x,y) \tag{2.12}$$

on a domain of unit square and $u_{i,j}$ is the numerical solution on a rectangle mesh with size $\Delta x = \Delta y = h$ in both x and y directions, the finite difference equation can be written as

$$\Delta_h u_{i,j} = \frac{u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - 4u_{i,j}}{h^2} = f_{i,j}. \tag{2.13}$$

Eq. (2.13) can be expressed in the matrix form $Au = d$, where $A = \{a_{ij}\}$ with $a_{i-1,j} = a_{i,j-1} = a_{i+1,j} = a_{i,j+1} = -\frac{1}{4}$ and $a_{i,j} = 1$. The right hand side is $-\frac{1}{4}h^2 f_{i,j}$. A general iteration algorithm is to split the matrix A into two matrices, that is let $A = B - C$, then the equation becomes $Bu = Cu + d$. The iteration takes the form

$$Bu^{k+1} = Cu^k + d, \tag{2.14}$$

or equivalently

$$u^{k+1} = B^{-1}Cu^k + B^{-1}d. \tag{2.15}$$

Define $e^k = u^{k+1} - u^k$, then

$$e^k = B^{-1}Cu^k - B^{-1}Cu^{k-1} = B^{-1}Ce^{k-1} = \left(B^{-1}C\right)^k e^0 \tag{2.16}$$

For the iteration algorithm to converge, it requires the spectral radius of $B^{-1}C$ to be smaller than 1, that is

$$\rho\left(B^{-1}C\right) < 1$$

which requires the magnitude of eigenvalue of $B^{-1}C$. be smaller than 1.

There are several iteration methods:

1. Jacobi iteration

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^{k} + u_{i,j-1}^{k} + u_{i+1,j}^{k} + u_{i,j+1}^{k}\right) - \frac{1}{4}h^2 f_{i,j} \tag{2.17}$$

The equivalent matrix form is $Au = b$ and $A = B - C$, $B = I$, $C = -\left(L + U\right)$.

2. Gauss-Seidel iteration

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^{k} + u_{i,j+1}^{k}\right) - \frac{1}{4}h^2 f_{i,j} \tag{2.18}$$

The equivalent matrix form is $Au = b$ and $A = B - C$, $B = I + L$, $C = -U$.

3. SOR iteration

$$u_{i,j}^{k+1} = \omega u_{i,j}^{k} + \frac{1}{4}\left(1-\omega\right)\left(u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i+1,j}^{k} + u_{i,j+1}^{k}\right) - \frac{1}{4}\left(1-\omega\right)h^2 f_{i,j} \tag{2.19}$$

The equivalent matrix form is $\left(1-\omega\right)Au = \left(1-\omega\right)b$, which implies

$(1-\omega)A = B - C$

$$B = I + (1-\omega)L,\ C = \omega I - (1-\omega)U. \tag{2.20}$$

2.1.4 Conservation Law

Consider conservation law

$$u_t + f(u)_x = 0$$

several classical numerical schemes are:

1. Lax-Friedrich Scheme

$$u_j^{n+1} = \frac{1}{2}\left(u_{j-1}^n + u_{j+1}^n\right) - \frac{\Delta t}{2\Delta x}\left(f\left(u_{j+1}^n\right) - f\left(u_{j-1}^n\right)\right) \tag{2.21}$$

2. Lax-Wendroff Scheme

$$u_{j+\frac{1}{2}}^{n+\frac{1}{2}} = \frac{1}{2}\left(u_{j-1}^n + u_{j+1}^n\right) - \frac{\Delta t}{2\Delta x}\left(f\left(u_{j+1}^n\right) - f\left(u_j^n\right)\right) \tag{2.22}$$

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(f\left(u_{j+\frac{1}{2}}^{n+\frac{1}{2}}\right) - f\left(u_{j-\frac{1}{2}}^{n+\frac{1}{2}}\right)\right) \tag{2.23}$$

3. Godunov Scheme

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(f\left(u_{j+\frac{1}{2}}^R\right) - f\left(u_{j-\frac{1}{2}}^R\right)\right) \tag{2.24}$$

where $u^R_{j+\frac{1}{2}}$ is the Riemann solution at $x_{j+\frac{1}{2}}$

$$u^R_{j+\frac{1}{2}} = V\left(u^n_j, u^n_{j+1}, 0\right), \; u^R_{j-\frac{1}{2}} = V\left(u^n_{j-1}, u^n_j, 0\right) \tag{2.25}$$

and $V(l, r, 0)$ is the Riemann solution obtained by using l as the left initial state and r as the right initial state.

4. WENO Scheme

$$u_t|_{x=x_j} = \frac{1}{\Delta x}\left(f_{j+\frac{1}{2}} - f_{j-\frac{1}{2}}\right) \tag{2.26}$$

where $\left\{f_{j+\frac{1}{2}}\right\}$ are reconstructed value of $f(u)$ at $u_{j+\frac{1}{2}}$ with WENO scheme [45]. Discretize the space into uniform intervals of size Δx and denote $x_j = j\triangle x$, $u_j = u(x_j)$ and $f_j = f(u_j)$. For a general flux $f(u)$, it can be splitted into two parts

$$f(u) = f^+(u) + f^-(u) \tag{2.27}$$

where $\frac{df^+(u)}{du} \geq 0$ and $\frac{df^-(u)}{du} \leq 0$ are monotone functions. This can be realized by the formula

$$f^{\pm}(u) = \frac{1}{2}\left(f(u) \pm \alpha u\right) \tag{2.28}$$

where $\alpha = \max\left|f'(u)\right|$ and the maximum is taken over the whole relevant range of u. The reconstruction for $f^+(u)$ uses a biased stencil with one more point to the left and that for $f^-(u)$ uses a biased stencil with one more point to the right to obey correct upwinding. Let $\hat{f}^+_{j+\frac{1}{2}}$ and

$\hat{f}^{-}_{j+\frac{1}{2}}$ be the numerical fluxes obtained from the positive and negative parts respectively, then

$$\hat{f}_{j+\frac{1}{2}} = \hat{f}^{+}_{j+\frac{1}{2}} + \hat{f}^{-}_{j+\frac{1}{2}}. \tag{2.29}$$

Consider the rth-order approximate of $f^{+}_{j+\frac{1}{2}}$ at r candidate stencils denoted by $S_k,\ k = 0, \cdots, r-1$

$$S_k = \left(x_{j+k-r+1}, \cdots, x_{j+k}\right), \tag{2.30}$$

then the rth-order approximation is

$$q^{r}_{k}\left(f_{j+k-r+1}, \cdots, f_{j+k}\right) = \sum_{l=j+k-r+1}^{j+k} a^{r}_{k,l} f_l \tag{2.31}$$

where$\left\{a^{r}_{k,l}\right\}$ are constant coefficients. When $r = 3$, it gives

$$\begin{aligned} q^{3}_{0} &= \tfrac{1}{3}f_{j-2} - \tfrac{7}{6}f_{j-1} + \tfrac{11}{6}f_j \\ q^{3}_{1} &= -\tfrac{1}{6}f_{j-2} + \tfrac{5}{6}f_{j-1} + \tfrac{1}{3}f_j. \\ q^{3}_{2} &= \tfrac{1}{3}f_{j-2} + \tfrac{5}{6}f_{j-1} - \tfrac{1}{6}f_j \end{aligned} \tag{2.32}$$

WENO reconstructed the value of function $f^{+}(u)$ at $u_{j+\frac{1}{2}}$ uses all the r candidate stencils, which all together contain $(2r-1)$ grid values of f to give a $(2r-1)$th-order approximation

$$\hat{f}^{+}_{j+\frac{1}{2}} = q^{2r-1}_{r-1}\left(f_{j-r+1}, \cdots, f_{j+r-1}\right) \tag{2.33}$$

and it can also be represented as the combination of r polynomials

$$\hat{f}^{+}_{j+\frac{1}{2}} = \sum_{k=0}^{r-1} \omega_k q_k^r \left(f_{j+k-r+1}, \cdots, f_{j+k}\right). \tag{2.34}$$

Through simple algebra, coefficients $\{C_k^r\}$ can be derived from

$$q_{r-1}^{2r-1} \left(f_{j-r+1}, \cdots, f_{j+r-1}\right) = \sum_{k=0}^{r-1} C_k^r q_k^r \left(f_{j+k-r+1}, \cdots, f_{j+k}\right) \tag{2.35}$$

and $\sum_{k=0}^{r-1} C_k^r = 1$. When $r = 3$, this gives

$$C_0^3 = \frac{1}{10}, C_1^3 = \frac{6}{10}, C_2^3 = \frac{3}{10}. \tag{2.36}$$

The choice of weight $\{\omega_k\}$ based on $\{C_k^r\}$ and follows a convex combination

$$\omega_k \geq 0, \ \sum_{k=0}^{r-1} \omega_k = 1 \tag{2.37}$$

for stability and consistency. The weight ω_k for stencil S_k is defined by

$$\omega_k = \frac{\alpha_k}{\alpha_0 + \cdots + \alpha_{r-1}} \tag{2.38}$$

where

$$\alpha_k = \frac{C_k^r}{\left(\epsilon + IS_k\right)^2} \tag{2.39}$$

$\epsilon > 0$ is a small value and usually set as 10^{-6}, $\{IS_k\}$ are the smooth indicators of the stencil S_k which has the interpolation polynomial $q_k(x)$

$$IS_k = \sum_{l=1}^{r-1} \int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} h^{2l-1} \left(q_k^{(l)}\right)^2 dx \tag{2.40}$$

where $q_k^{(l)}$ is the lth-derivative of $q_k(x)$ and Eq. (2.40) is the sum of the L^2 norms of all the derivatives of the interpolation polynomial $q_k(x)$ over the interval $\left(x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}\right)$. When $r = 3$, the corresponding smooth indicators are

$$\begin{aligned} IS_0 &= \tfrac{13}{12}\left(f_{j-2} - 2f_{j-1} + f_j\right)^2 + \tfrac{1}{4}\left(f_{j-2} - 4f_{j-1} + 3f_j\right)^2 \\ IS_1 &= \tfrac{13}{12}\left(f_{j-1} - 2f_j + f_{j+1}\right)^2 + \tfrac{1}{4}\left(f_{j-1} - f_{j+1}\right)^2 \\ IS_2 &= \tfrac{13}{12}\left(f_j - 2f_{j+1} + f_{j+2}\right)^2 + \tfrac{1}{4}\left(3f_j - 4f_{j+1} + 3f_{j+2}\right)^2 \end{aligned} \tag{2.41}$$

which gives fifth-order accurate WENO scheme. After the reconstruction process, the problem is simplified as an ODE

$$\frac{du}{dt} = L(u). \tag{2.42}$$

2.1.5 Boundary Condition

For Dirichlet boundary condition $u(x,t) = h(x,t), x \in \partial\Omega$, it can be numerically represented as

$$u_k^n = h(x_k, t_n). \tag{2.43}$$

For Neumann boundary condition $u_x(x,t) = h(x,t), x \in \partial\Omega$, the first order approximation is

$$u_k^n = u_{k+1}^n - \Delta x h\left(x_k, t^n\right) \tag{2.44}$$

and the second order approximation is

$$u_{k-1}^n = u_{k+1}^n - 2\Delta x h\left(x_k, t^n\right) \tag{2.45}$$

where $x_k \in \partial\Omega$, $x_{k+1} \in \Omega$ and x_{k-1} is a ghost point.

2.1.6 Numerical ODE Scheme

ODE has wide applications and usually couples with PDE in simulations. A simple example is Newton's second law of motion which describes the relationship between displacement x and time t of the object. The order of ODE is determined by highest-order derivative. However, ODE with higher-order can be transformed into equivalent first-order system. So we only look into numerical methods for first-order ODEs

$$\vec{y}_t = f(t, \vec{y}), \tag{2.46}$$

for simplicity, we consider only single scalar ODE. There are several typical methods:

1. Taylor Series Methods

(a) Forward Euler scheme

$$y_{k+1} = y_k + hf\left(t_k, y_k\right) \tag{2.47}$$

first order accuracy and stability requires $|1 + h\lambda| < 1$.

(b) Backward Euler scheme

$$y_{k+1} = y_k + hf\left(t_{k+1}, y_{k+1}\right) \tag{2.48}$$

implicit first order accuracy scheme and unconditionally stable.

(c) Second-order scheme

$$y_{k+1} = y_k + hf\left(t_k, y_k\right) + \frac{h^2}{2}\left[f_t\left(t_k, y_k\right) + f_y\left(t_k, y_k\right) f\left(t_k, y_k\right)\right] \tag{2.49}$$

2. Runge-Kutta methods: single-step methods with similar motivation with Taylor series methods, but avoid calculating higher derivatives by evaluating f several times between t_k and t_{k+1}

(a) Second-order Heun's method

$$y_{k+1} = y_k + \frac{h}{2}\left(k_1 + k_2\right) \tag{2.50}$$

where

$$\begin{aligned} k_1 &= f\left(t_k, y_k\right) \\ k_2 &= f\left(t_k + h, y_k + hk_1\right) \end{aligned} \tag{2.51}$$

(b) Fourth-order Runge-Kutta method

$$y_{k+1} = y_k + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k4\right) \tag{2.52}$$

where

$$\begin{aligned} k_1 &= f\left(t_k, y_k\right) \\ k_2 &= f\left(t_k + \tfrac{h}{2}, y_k + \tfrac{h}{2}k_1\right) \\ k_3 &= f\left(t_k + \tfrac{h}{2}, y_k + \tfrac{h}{2}k_2\right) \\ k_4 &= f\left(t_k + h, y_k + hk_3\right) \end{aligned} \tag{2.53}$$

3. Multistep methods

$$y_{k+1} = \sum_{i=1}^{m} \alpha_i y_{k+1-i} + h \sum_{i=0}^{m} \beta_i f\left(t_{k+1-i}, y_{k+1-i}\right) \tag{2.54}$$

where $\{\alpha_i\}$ and $\{\beta_i\}$ are determined by polynomial interpolation.

2.2 Numerical Methods for PDE extended Systems

2.2.1 Numerical Solver for Advection Term

Advection Solver

A lot of complicated PDE systems involve advection term Eq. (2.55)

$$u_t + af(u)_x = 0. \tag{2.55}$$

A usual numerical method is to reconstruct the second term and transform it into an ODE

$$u_t = L\left(u\right). \tag{2.56}$$

Eq. (2.56) can be derived from reconstructed based on PDE (ex: Euler equation) or directly from ODE (ex: spring model case). Then a 4th-order Runge-Kutta method can be used to solve the equation numerically

$$\begin{array}{c} u^{(1)} = u^n + \frac{1}{2}\Delta t L\left(u^n\right) \\ u^{(2)} = u^n + \frac{1}{2}\Delta t L\left(u^{(1)}\right) \\ u^{(3)} = u^n + \Delta t L\left(u^{(2)}\right) \\ u^{n+1} = \frac{1}{3}\left(-u^n + u^{(1)} + 2u^{(2)} + u^{(3)}\right) + \frac{1}{6}\Delta t L\left(u^{(3)}\right) \end{array}. \tag{2.57}$$

Solve Euler Equation based on WENO

WENO scheme is used to reconstruct the derivative term and Euler equation is solved to demonstrate the process of advection solver. High-order numerical methods have been widely used to effectively resolve complex flow features such as turbulent or vertical flows [31]. High-order shock-capturing schemes such as the Essentially Non-Oscillatory (ENO) and Weighted ENO (WENO) [59, 46] 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.

To numerically solve one-dimensional Euler equation

$$\mathbf{U}_t + F\left(\mathbf{U}\right)_x = 0 \tag{2.58}$$

where

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \quad F(\mathbf{U}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (E+p)\,u \end{pmatrix} \tag{2.59}$$

$$E = \rho\left(e + \frac{u^2}{2}\right), \quad p = \rho e\,(\gamma - 1), \quad \gamma = 1.4 \tag{2.60}$$

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

$$\mathbf{A} = \frac{\partial F(\mathbf{U})}{\partial \mathbf{U}} \tag{2.61}$$

The eigenvalues of the Jacobian matrix A are

$$\lambda_1 = u - c, \quad \lambda_2 = u, \quad \lambda_3 = u + c \tag{2.62}$$

where $c = \sqrt{\frac{\gamma p}{\rho}}$ is the sound speed, and the corresponding right eigenvectors are

$$\mathbf{r}_1 = \begin{pmatrix} 1 \\ u - a \\ H - ua \end{pmatrix}, \quad \mathbf{r}_2 = \begin{pmatrix} 1 \\ u \\ \frac{u^2}{2} \end{pmatrix}, \quad \mathbf{r}_3 = \begin{pmatrix} 1 \\ u + a \\ H + ua \end{pmatrix} \tag{2.63}$$

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

$$H = \frac{(E+p)}{\rho} = \frac{1}{2}u^2 + h, \quad h = e + \frac{P}{\rho} \tag{2.64}$$

We denote the matrix whose columns are eigenvectors in Eq. (2.63) by

$$\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) \tag{2.65}$$

and denote $\mathbf{L} = \mathbf{R}^{-1}$. The eigenvalue decomposition of $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{L}^{-1}\boldsymbol{\Lambda}\mathbf{L} \tag{2.66}$$

where $\boldsymbol{\Lambda} = diag\,(u-c, u, u+c)$. Then, Eq. (2.58) can be transformed as

$$(\mathbf{LU})_t + \Lambda\,(\mathbf{LU})_x = 0 \tag{2.67}$$

which consists of three independent one-dimensional hyperbolic equations. Each equation can be transformed using WENO scheme then solved by advection solver.

2.2.2 Projection Method for Navier-Stokes equation

Projection method is an effective method to numerically solve time-dependent incompressible fluid flow problems. The method is pioneered by Chorin [19, 20] based on the observation that the left-hand side of the momentum equation

$$\rho_0\left[\frac{\partial\vec{u}}{\partial t} + (\vec{u}\cdot\nabla)\,\vec{u}\right] = -\nabla p + \nu\nabla^2\vec{u} \tag{2.68}$$

is a Hodge decomposition. Thus, there exists a project operator P which projects a vector field onto the space of divergence-free vector fields with appropriate boundary condition. The advantage of the projection method is that the computation of velocity and the pressure fields are decoupled. Some approximation to the momentum equation Eq. (2.68) is used to determine an intermediate velocity $\vec{u}^{*}$, then an elliptic equation is solved to enforce the divergence constraint required by incompressible property

$$\nabla \cdot \vec{u} = 0. \tag{2.69}$$

In some variations, the advection term and viscous term are calculated separately in Eq. (2.68). 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 because of observed weak instabilities and the desire to use more complicated or adaptive finite difference meshes. Additionally, as with all fractional step methods, a crucial issue is how boundary conditions are determined for some or all of the intermediate variables.

In the 1980s, several second-order accurate projection methods were proposed, including Goda [38], Bell [7], Kim and Moin [48] and Van Kan [98]. These methods are mostly based on the second-order, time-discrete semi-implicit forms of Eq. (2.68) and Eq. (2.69)

$$\frac{u^{n+1} - u^{n}}{\Delta t} + \frac{1}{\rho}\nabla p^{n+\frac{1}{2}} = -\left[(u \cdot \nabla)\, u\right]^{n+\frac{1}{2}} + \frac{\nu}{2\rho}\nabla^{2}\left(u^{n+1} + u^{n}\right) \tag{2.70}$$

$$\nabla \cdot u^{n+1} = 0 \tag{2.71}$$

where $[(u \cdot \nabla)u]^{n+1/2}$ represents a second-order approximation to the advective derivative term at time level $t^{n+1/2}$ which is usually computed explicitly. Spatially discretized versions of the coupled Eq. (2.70) and Eq. (2.71) 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. (2.70) 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 can be summarized as [16]:

Step 1: Solve for the intermediate field u^*

$$\frac{u^* - u^n}{\Delta t} + \frac{1}{\rho}\nabla q = -\left[(u \cdot \nabla)\, u\right]^{n+1/2} + \frac{\nu}{2\rho}\nabla^2 \left(u^* + u^n\right) \tag{2.72}$$

$$B\left(u^*\right) = 0 \tag{2.73}$$

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

Step 2: Perform the projection step

$$u^* = u^{n+1} + \frac{\Delta t}{\rho}\nabla\phi^{n+1} \tag{2.74}$$

$$\nabla \cdot u^{n+1} = 0 \tag{2.75}$$

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

Step 3: Update the pressure

$$p^{n+1/2} = q + L\left(\phi^{n+1}\right) \tag{2.76}$$

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 which will not 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:

1. pressure approximation q in Eq. (2.72)
2. boundary condition $B(u^*) = 0$ in Eq. (2.73)
3. pressure-update equation $L(\phi^{n+1})$ in Eq. (2.76)

An important issue is that given the boundary condition $B(u^*) = 0$ for u^* and boundary condition $u^{n+1}|_{\partial\Omega} = u_b^{n+1}$ for u_b^{n+1}, the boundary condition for ϕ can be derived through Eq. (2.74).

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 $B(u^*) = 0$ to include a nontrivial approximation of $\nabla\phi^{n+1}$ in Eq. (2.74). Regarding the third step of the method, substituting Eq. (2.76) into Eq. (2.72), eliminating u^*, and comparing with Eq. (2.70) yield a formula for the pressure-update in second-order accuracy

$$p^{n+1/2} = q + \phi^{n+1} - \frac{\nu \Delta t}{2} \nabla^2 \phi^{n+1}. \tag{2.77}$$

The last term of Eq. (2.77) 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.

The implementation of the numerical solver for the Navier-Stokes equation can be summarized as following:

1. Solve advection term: uses advection solver introduced in Sec. 2.2.1
2. Solve diffusion term: uses Crank-Nicolson scheme Eq. (2.9)
3. Perform projection: solve Poisson equation to obtain ϕ
4. Update new velocity and pressure

2.2.3 Numerical Solver for Spring Model

Eq. (1.51) implies that the spring model can be represented as a first-order system of equations

$$\frac{d}{dt} \begin{pmatrix} \mathbf{x} \\ \mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{u} \\ \frac{\mathbf{F}}{m} \end{pmatrix} \tag{2.78}$$

where **x** is the location of the mass point, **u** is the velocity of mass point and **F** is the force on the mass point which come from the spring system and is calculated from the information of neighbor mass points via Eq. (1.48). This is an ODE system and can naturally be solved using the numerical solver introduced in Sec. 2.2.1.

To simulate the motion fabric, this equation is calculated for each mass point in the spring system. The most time consuming part is the calculation of force F_i on mass point i for all mass points. However, since the force on each mass point can be calculated independently through spring elongation and angular deformation, they can be computed simultaneously to accelerate the program. This drives us to use heterogeneous computing technique and offload parallelable part to GPU. A detailed algorithm will be given in Sec. 3.3.2.

2.2.4 Numerical Solver for PIDE

The numerical scheme of PIDE is explored in the domain $[0,T]\times[-A,A]$. The discrete grid on the domain is:

$$\tau_n = n\triangle t, \quad n = 0,\cdots,M, \quad \triangle t = \frac{T}{M}; \tag{2.79}$$

$$x_k = -A + k\triangle x, \quad k = 0,\cdots,N, \quad \triangle x = \frac{2A}{N}. \tag{2.80}$$

And the *initial condition* can be transformed as

$$f(\tau_0, x_k) = \begin{cases} K - S_0 e^{x_k}, & x_k < \log\frac{K}{S_0} \\ 0, & x_k \geq \log\frac{K}{S_0} \end{cases}, \quad k = 0,\cdots,N. \tag{2.81}$$

Choose A such that $-A < \log \frac{K}{S_0} < A$, then the *boundary condition* for European option is

$$f(\tau, x) = \begin{cases} Ke^{-r\tau} - S_0 e^x, & (\tau, x) \in [0, T] \times (-\infty, -A] \\ 0, & (\tau, x) \in [0, T] \times [A, \infty) \end{cases}, \tag{2.82}$$

and for American option is

$$f(\tau, x) = \begin{cases} K - S_0 e^x, & (\tau, x) \in [0, T] \times (-\infty, -A] \\ 0, & (\tau, x) \in [0, T] \times [A, \infty) \end{cases}. \tag{2.83}$$

Due to the constant coefficient, the advection term can be computed with central implicit scheme using Eq. (2.4) and the diffusion term can be computed with central implicit scheme Eq. (2.8). This gives

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

$$\frac{\sigma^2}{2} \frac{f_{k+1}^{n+1} - 2f_k^{n+1} + f_{k-1}^{n+1}}{(\triangle x)^2} - r f_k^n + \int_R g(y)\left[f(\tau_n, x_k + y) - f(\tau_n, x_k)\right] dy, \tag{2.84}$$

assume

$$I_1 = \int_R g(y)(e^y - 1)\, dy \tag{2.85}$$

$$I_2^{n,k} = \int_R g(y)\left[f(\tau_n, x_k + y) - f(\tau_n, x_k)\right] dy \tag{2.86}$$

which should be computed using efficient numerical method.

The Lévy measure of generalized hyperbolic distribution is $g(y)$ which is singular at 0 and $g(y)\, e^{-\beta y}$ is even function. Then the calculation of I_1 can be simplified as

$$I_1 = \int_{-\infty}^{\infty} g(y)\left(e^{y}-1\right) dy$$

$$= \int_{-\infty}^{0} g(y)\left(e^{y}-1\right) dy + \int_{0}^{\infty} g(y)\left(e^{y}-1\right) dy$$

$$= \int_{0}^{\infty} g(-y)\left(e^{-y}-1\right) dy + \int_{0}^{\infty} g(y)\left(e^{y}-1\right) dy$$

$$= \int_{0}^{\infty} g(y)\left[e^{-2\beta y}\left(e^{-y}-1\right) + e^{y}-1\right] dy. \tag{2.87}$$

$I_2^{n,k}$ is the integration term which causes the major difference from traditional Black-Scholes equation. The formula of $I_2^{n,k}$ is

$$I_2^{n,k} = \int_{-\infty}^{\infty} \left[f\left(\tau_n, x_k + y\right) - f\left(\tau_n, x_k\right)\right] g(y)\, dy \tag{2.88}$$

which contains numerical results $\{f_k^n\}$ and should be updated explicitly in each time step. Divide the original integration domain into five sub-intervals, then calculate each part with appropriate methods.

$$\int_{-\infty}^{\infty} \left[f\left(\tau_n, x_k + y\right) - f\left(\tau_n, x_k\right)\right] g(y)\, dy$$

$$= \int_{-\infty}^{x_0 - x_k} \left[f\left(\tau_n, x_k + y\right) - f_k^n\right] g(y)\, dy$$

$$+ \int_{x_0 - x_k}^{-\triangle x} \left[f\left(\tau_n, x_k + y\right) - f_k^n\right] g(y)\, dy$$

$$+ \int_{-\triangle x}^{\triangle x} \left[f\left(\tau_n, x_k + y\right) - f\left(\tau_n, x_k\right)\right] g(y)\, dy$$

$$+\int_{\triangle x}^{x_N-x_k}\left[f\left(\tau_n,x_k+y\right)-f_k^n\right]g\left(y\right)dy$$

$$+\int_{x_N-x_k}^{\infty}\left[f\left(\tau_n,x_k+y\right)-f_k^n\right]g\left(y\right)dy. \tag{2.89}$$

The integration in the center sub interval $[-\triangle x,\triangle x]$ is

$$\int_{-\triangle x}^{\triangle x}\left[f\left(\tau_n,x_k+y\right)-f\left(\tau_n,x_k\right)\right]g\left(y\right)dy$$

$$=\int_{-\triangle x}^{\triangle x}\left[f\left(\tau_n,x_k\right)+yf_x\left(\tau_n,x_k\right)+O\left(y^2\right)-f\left(\tau_n,x_k\right)\right]g\left(y\right)dy$$

$$\approx\int_{-\triangle x}^{\triangle x}f_x\left(\tau_n,x_k\right)yg\left(y\right)dy\approx\frac{f_{k+1}^n-f_{k-1}^n}{2\triangle x}\int_{-\triangle x}^{\triangle x}yg\left(y\right)dy, \tag{2.90}$$

where $\int_{-\triangle x}^{\triangle x}yg\left(y\right)dy$ can be calculated as $\int_0^{\triangle x}y\left(1-e^{-2\beta y}\right)g\left(y\right)dy$.

The integration on the right sub-interval $[\triangle x,x_N-x_k]$ can be estimated with composite trapezoid method. Trapezoid method is chosen because the numerical value of $\{f_k^n\}$ are known at grid point $\{k\triangle x\}$.

$$\int_{\triangle x}^{x_N-x_k}\left[f\left(\tau_n,x_k+y\right)-f_k^n\right]g\left(y\right)dy$$

$$=\sum_{i=1}^{N-k-1}\int_{i\triangle x}^{(i+1)\triangle x}\left[f\left(\tau_n,x_k+y\right)-f_k^n\right]g\left(y\right)dy$$

$$\approx\sum_{i=1}^{N-k-1}\frac{\left(f_{k+i}^n-f_k^n\right)g\left(i\triangle x\right)+\left(f_{k+i+1}^n-f_k^n\right)g\left(\left(i+1\right)\triangle x\right)}{2}\triangle x$$

$$=\left[\frac{f_{k+1}^n-f_k^n}{2}g\left(\triangle x\right)+\sum_{i=2}^{N-k-1}\left(f_{k+i}^n-f_k^n\right)g\left(i\triangle x\right)\right.$$

$$+\frac{f_N^n - f_k^n}{2} g\left((N-k)\triangle x\right)\Bigg]\Delta x \tag{2.91}$$

The left sub-interval $[x_0 - x_k, -\triangle x]$ can be estimated with similar method

$$\int_{x_0-x_k}^{-\triangle x} \left[f\left(\tau_n, x_k + y\right) - f\left(\tau_n, x_k\right)\right] g\left(y\right) dy$$

$$= \int_{\triangle x}^{x_k - x_0} \left[f\left(\tau_n, x_k - y\right) - f_k^n\right] e^{-2\beta y} g\left(y\right) dy$$

$$= \sum_{i=1}^{k-1} \int_{i\triangle x}^{(i+1)\triangle x} \left[f\left(\tau_n, x_k - y\right) - f\left(\tau_n, x_k\right)\right] e^{-2\beta y} g\left(y\right) dy$$

$$\approx \sum_{i=1}^{k-1} \frac{\left(f_{k-i}^n - f_k^n\right) e^{-2\beta i\triangle x} g\left(i\triangle x\right) + \left(f_{k-i-1}^n - f_k^n\right) e^{-2\beta i\triangle x} g\left((i+1)\triangle x\right)}{2}\triangle x$$

$$= \Bigg[\frac{f_{k-1}^n - f_k^n}{2} e^{-2\beta\triangle x} g\left(\triangle x\right) + \sum_{i=2}^{k-1}\left(f_{k+i}^n - f_k^n\right) e^{-2\beta i\triangle x} g\left(i\triangle x\right)$$

$$+\frac{f_0^n - f_k^n}{2} e^{-2\beta k\triangle x} g\left((N-k)\triangle x\right)\Bigg]\Delta x. \tag{2.92}$$

When $y \in [x_N - x_k, \infty)$, $x_k + y \geq x_N$, then $f\left(\tau_n, x_k + y\right) = 0$ which gives simple numerical estimation

$$\int_{x_N-x_k}^{\infty} \left[f\left(\tau_n, x_k + y\right) - f\left(\tau_n, x_k\right)\right] g\left(y\right) dy$$

$$= -f_k^n \int_{x_N-x_k}^{\infty} g\left(y\right) dy. \tag{2.93}$$

When $y \in (-\infty, x_0 - x_k]$, $x_k + y \leq x_0 = -A$, then $f\left(\tau_n, x_k + y\right) = Ke^{-r\tau_n} - S_0 e^{x_k + y}$ for European option; $f\left(\tau_n, x_k + y\right) = K - S_0 e^{x_k+y}$ for American option. The integration in the left to infinity sub-interval can be estimated as

$$
\begin{aligned}
&\int_{-\infty}^{x_0-x_k}\left[f\left(\tau_n,x_k+y\right)-f\left(\tau_n,x_k\right)\right]g\left(y\right)dy\\
&=\int_{-\infty}^{x_0-x_k}\left[K-S_0e^{x_k+y}-f_k^n\right]g\left(y\right)dy\\
&=\int_{x_k-x_0}^{\infty}\left[K-S_0e^{x_k-y}-f_k^n\right]e^{-2\beta y}g\left(y\right)dy\\
&=\left(K-f_k^n\right)\int_{x_k-x_0}^{\infty}e^{-2\beta y}g\left(y\right)dy-S_0e^{x_k}\int_{x_k-x_0}^{\infty}e^{-(2\beta+1)y}g\left(y\right)dy. \qquad (2.94)
\end{aligned}
$$

To calculate $I_2^{n,k}$, we need $\{f_k^n\}$, $\{g\left(k\triangle x\right)\}$, $\int_{-\triangle x}^{\triangle x}yg\left(y\right)dy$, $\int_{k\triangle x}^{\infty}g\left(y\right)dy$, $\int_{k\triangle x}^{\infty}e^{-2\beta y}g\left(y\right)dy$ and $\int_{k\triangle x}^{\infty}e^{-(2\beta+1)y}g\left(y\right)dy$, where $k=1,\cdots,N-1$. Since the Lévy measure $g\left(y\right)$ in this case has quite complicate form as stated in Eq. (1.59) which is an integration and should be estimated numerically. To accelerate the algorithm, the calculation of $g\left(y\right)$ is limited in the positive half axis. Due to the independence of integration for each grid point, they can be computed simultaneously using heterogeneous algorithm to further accelerate as illustrated in Sec. 3.3.3.

2.3 Application Implementation

In this section, we focus on the parachute simulation based on front tracking method which involves fluid-structure interaction, canopy porosity simulation, clustered parachutes and collision handling. Another important topic is American option pricing based on exponential Lévy process built on generalized hyperbolic distribution.

2.3.1 Front Tracking based on *FronTier*

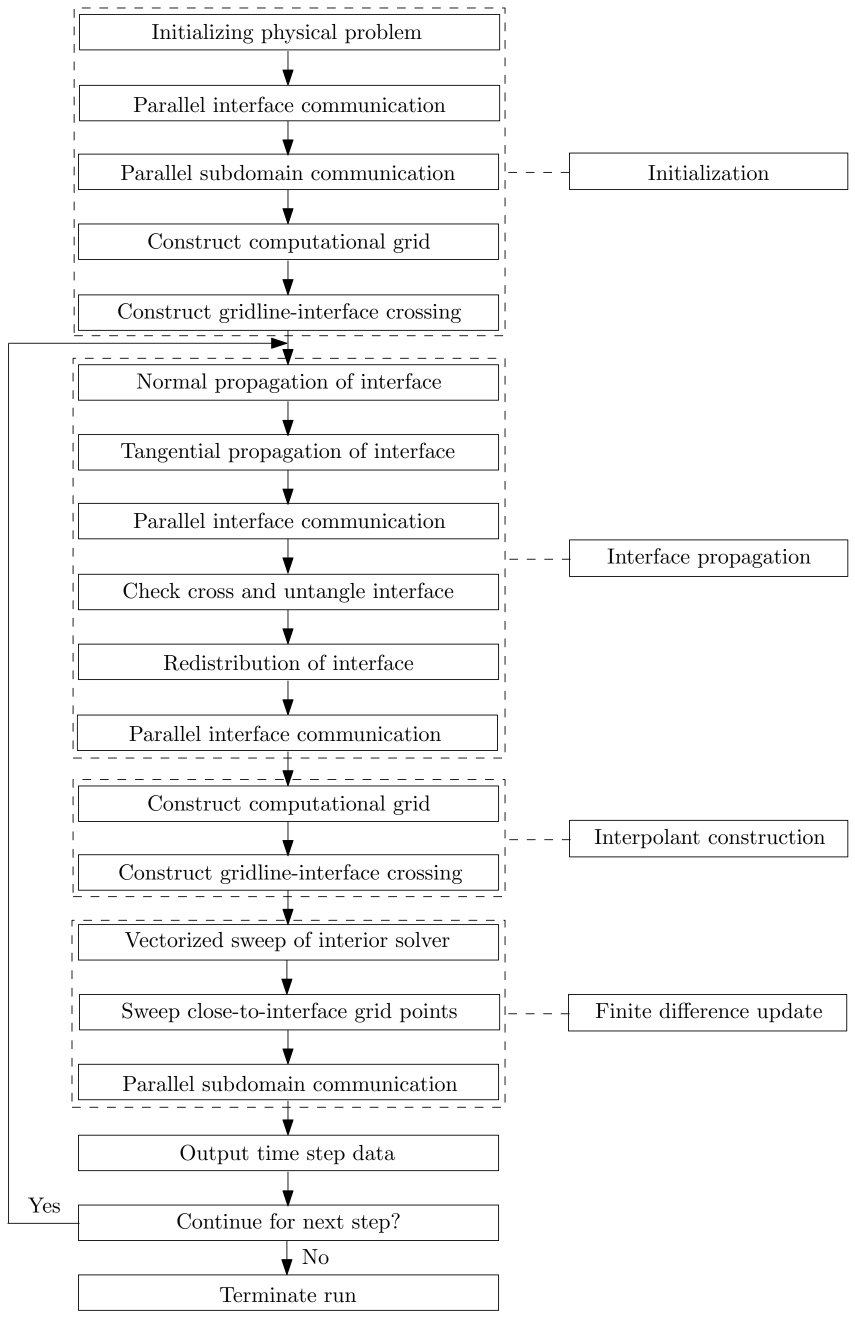

Initializing physical problem
Parallel interface communication
Parallel subdomain communication
Construct computational grid
Construct gridline-interface crossing
Initialization
Normal propagation of interface
Tangential propagation of interface
Parallel interface communication
Check cross and untangle interface
Redistribution of interface
Parallel interface communication
Interface propagation
Construct computational grid
Construct gridline-interface crossing
Interpolant construction
Vectorized sweep of interior solver
Sweep close-to-interface grid points
Parallel subdomain communication
Finite difference update
Output time step data
Yes
Continue for next step?
No
Terminate run

Figure 2.1: The flow chart of *FronTier* library.

We run fluid-structure interacted simulation on the *FronTier* library which sets foundation on front tracking method. Fig. 2.1 shows the major steps of the front tracking process. There are two tasks to be accomplished at each time step: first, dynamically evolve the front which describe the motion of the interface; then, calculate the numerical solutions in smooth regions surrounded by the fronts. More details about the *FronTier* library can be found in [9, 10].

One of the best features of *FronTier* library is the delicate handling of interface geometry which enables reliable fluid-structure interaction simulation. A hyper-surface is treated as a topologically linked set of marker points. The library contains data structure and functionalities to optimize and resolve the hyper-surface mesh with topological consistency. The library has been used for the simulations of fluid interface instabilities [29, 28, 36, 35], diesel jet droplet formation [11], and plasma pellet injection process [75, 76]. 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 simulation of parachute involves quite unique interface handling. The parachute canopy, made up of fabric, is not a separable interface and requires considering of porosity. Fabric surface cannot bifurcate which calls for special collision handling, especially for the case of parachute clusters simulation.

2.3.2 Parachute Simulation

Researchers have studied parachute with different methods including empirical analysis [68], semi-numerical simulation [69] and through experiments [70, 69]. There are many attempts to simulate parachute via computational method. Stein and Benney used finite element method to simulate fluid and structure dynamics [84, 83, 81, 85, 82]. Tezduyar [92, 91, 90, 89, 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 [49, 50], performed simulations at small Reynold number (about 300) case. Yu and Min [101] studied the transient aerodynamic characteristics of the parachute opening process. Karagiozis used the large-eddy simulation to study parachute in Mach 2 supersonic flow [47]. Purvis [71, 72] 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 [87]. 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 [96, 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.

While solving incompressible Navier-Stokes equation, boundary condition plays an important role in simulating fluid structure interaction. For the parachute simulation, the boundary condition consists of two parts, the external boundary and two interior sides of the canopy surface. Three different external boundary conditions are implemented. The periodic boundary requires no special treatment. The Dirichlet boundary is usually preset on the upwind side while the flow-through boundary is on the downwind side. The only approximation is to assume the downwind side of flow-through boundary is a constant extrapolation. This is required for the parabolic equation.

The interaction between fluid and the canopy is the most crucial part of the algorithm for the parachute simulation. The system described by Eq. (1.51) 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. 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, the spring system can be solved and the internal impulse $\mathbf{I}_i^s$ can be derived. 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

$$\mathbf{I}_i = \mathbf{I}_i^e + \mathbf{I}_i^s. \tag{2.95}$$

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

$$\mathbf{I}_i^c = \mathbf{I}_{gi}^c + \mathbf{I}_{pi}^c + \mathbf{I}_{si}^c. \tag{2.96}$$

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

$$\mathbf{I}_i^s = \mathbf{I}_{gi}^s + \mathbf{I}_{si}^s. \tag{2.97}$$

The external impulse due to gravity and pressure is time integrated for each mass point, that is

$$\mathbf{I}_{gi} = \int_0^t m\mathbf{g}dt \tag{2.98}$$

for both canopy and string chord mass points and

$$\mathbf{I}_{pi} = \int_0^t \sigma \left(p^- - p^+\right) \mathbf{n} dt \tag{2.99}$$

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. 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

$$\begin{cases} \frac{d}{dt}\mathbf{x}_i = \mathbf{u}_i \\ \frac{d}{dt}\mathbf{u}_i = \frac{1}{m}(\mathbf{F}_i + \mathbf{f}_i^e) - \kappa \mathbf{v}_i^s \end{cases} \tag{2.100}$$

where $\mathbf{F}_i$ is the internal force from spring system, $\mathbf{f}_i^e$ is the external force, κ

is the damping coefficient and $\mathbf{v}_i^s$ is the velocity component due to the spring impulse $\mathbf{v}_i^s = \frac{\mathbf{I}_i^s}{m}$.

To simulate the reaction of the canopy, the increment of the impulse at each point on the canopy can be calculated by Peskin's delta function method

$$\mathbf{f}\left(\mathbf{x}, t\right) = \int \mathbf{F}\left(s, t\right) \delta\left(\mathbf{x} - \mathbf{X}\left(s, t\right)\right) ds \tag{2.101}$$

where $\mathbf{F}$ is the superposition of three forces from the spring system

$$\mathbf{F}\left(x_i, i\right) = \frac{d}{dt}\left(\mathbf{I}_g + \mathbf{I}_p + \mathbf{I}_s\right). \tag{2.102}$$

2.3.3 Canopy Porosity Simulation

The porosity of parachute canopy greatly affects parachute performance. There are two forms of parachute porosity: for canopy manufactured from solid fabric, the porosity (sometimes referred as fabric permeability) is defined as the airflow through canopy cloth in $ft^3/ft^2/min$ (cubic feet per minute per square feet) at $\frac{1}{2}$ inch water pressure; for slotted canopies which has geometric openings, porosity is defined in percent as the ratio of all open areas to the total canopy area. Most personnel parachutes and main descent parachute for air vehicles use materials with porosity from 80 to 150 $ft^3/ft^2/min$. Gliding parachutes use almost imporous materials from 0 to 5 $ft^3/ft^2/min$. Slotted parachutes use geometric porosity in 10% to 35% range [54].

Porosity influences parachute drag, stability and opening forces. Higher porosity decreases opening forces and oscillation, but reduces drag forces at the same time which is usually not desirable. As a result, parachute canopy porosity is an important factor to consider for both parachute design and parachute simulation. Different types of parachutes have different design characteristics requirements [41, 54, 77].

Given the significance of porosity, there are several attempts to simulate porous canopy parachute motion. Kim and Peskin [51] use immersed boundary method to simulate parachute motion and derive the relative velocity between fluid and canopy interface using Darcy's Law. Tutt [97] simulate parachute performance under LS-DYNA and use Ergun equation to describes the magnitude of porous flow. Wang, Aquelet, Tutt, Do, Chen and Souli [99] simulate the interaction between the fluid and porous medium by a Euler-Lagrange coupling under LS-DYNA framework.

A cloth porosity of 27.4 $ft^3/ft^2/min$ at $\frac{1}{2}$ inch water pressure is equivalent to 1% geometric porosity [54]. This enables comparison between different parachutes and uniform simulations among parachutes which have different porosity types.

We introduce the concept of *penetration ratio* γ, which is a *dimensionless* coefficient $0 \leq \gamma \leq 1$. When $\gamma = 0$, it means no penetration will happen and the fluid on two sides of boundary has no connection; when $\gamma = 1$, it means the interface does not exist; when $0 < \gamma < 1$, part of the fluid can penetrate the canopy and canopy interface is a porous medium. The fluid field is described by Navier-Stokes equation and solved by projection method [16]. Coupling penetration ratio into projection method transforms fluid through porous medium simulation into boundary condition treatment.

Advection Term

The advection term solver in Sec. 2.2.1 is used to proceed calculation which requires information from seven nearby points, i.e.

$$u_i^{k+1} = f\left(u_{i-3}^k, u_{i-2}^k, u_{i-1}^k, u_i^k, u_{i+1}^k, u_{i+2}^k, u_{i+3}^k\right). \tag{2.103}$$

When came across boundary, interpolation is required to obtain *ghost points*. Assume parachute canopy is between u_i and u_{i+1}, then $u_{i+1}^k, u_{i+2}^k, u_{i+3}^k$ should be interpolated as

$$u_{i+p}^{k,ghost} = g_p\left(\left\{u_{1,\cdots,n}^k\right\}\right), p = 1, 2, 3 \tag{2.104}$$

Thus, the calculation of velocity on the boundary is

$$u_i^{k+1} = f\left(u_{i-3}^k, u_{i-2}^k, u_{i-1}^k, u_i^k, u_{i+1}^{k,ghost}, u_{i+2}^{k,ghost}, u_{i+3}^{k,ghost}\right). \tag{2.105}$$

Coupling penetration ratio γ, we set boundary points as

$$u_{i+p}^{k,poro} = \gamma u_{i+p}^k + (1-\gamma)\, u_{i+p}^{k,ghost}, p = 1, 2, 3 \tag{2.106}$$

which leads to new construction formula

$$u_i^{k+1} = f\left(u_{i-3}^k, u_{i-2}^k, u_{i-1}^k, u_i^k, u_{i+1}^{k,poro}, u_{i+2}^{k,poro}, u_{i+3}^{k,poro}\right). \tag{2.107}$$

Diffusion Term

Diffusion part is solved using Crank-Nicolson scheme

$$\frac{u_i^{k+1}-u_i^k}{\triangle t}=\frac{1}{2}\left(\frac{u_{i+1}^{k+1}-2u_i^{k+1}+u_{i-1}^{k+1}}{\triangle x^2}+\frac{u_{i+1}^{k}-2u_i^{k}+u_{i-1}^{k}}{\triangle x^2}\right) \tag{2.108}$$

and given $a=\frac{\triangle t}{2\triangle x^2}$, it can be simplified as

$$-au_{i+1}^{k+1}+(1+2a)\,u_i^{k+1}-au_{i-1}^{k+1}=au_{i+1}^{k}+(1-2a)\,u_i^{k}+au_{i-1}^{k}. \tag{2.109}$$

Assume there is canopy interface between u_i and u_{i+1}, and consider no porosity case, we get the velocity from interface $u_{i+1}^{k,state}$. Coupling with constant velocity boundary condition, we have

$$-au_{i+1}^{k,state}+(1+2a)\,u_i^{k+1}-au_{i-1}^{k+1}=au_{i+1}^{k,state}+(1-2a)\,u_i^{k}+au_{i-1}^{k}. \tag{2.110}$$

Introducing penetration ratio gives

$$-\gamma au_{i+1}^{k+1}+(1+2a)\,u_i^{k+1}-au_{i-1}^{k+1}=2\,(1-\gamma)\,au_{i+1}^{k,state}+\gamma au_{i+1}^{k}+(1-2a)\,u_i^{k}+au_{i-1}^{k}. \tag{2.111}$$

Projection Step

After obtaining intermediate velocity, projection step is carried out to obtain pressure and new velocities. Solve Poisson equation gives pressure

$$p_{i+1}-2p_i+p_{i-1}=\frac{\rho\triangle x^2}{\triangle t}div\,(u_i^*)\,. \tag{2.112}$$

Considering the case which has canopy interface between u_i and u_{i+1}, and using Neumann boundary will give the formula

$$-p_i + p_{i-1} = \frac{\rho \triangle x^2}{\triangle t} div\left(u_i^*\right). \tag{2.113}$$

Introducing penetration ratio γ gives

$$\gamma p_{i+1} - (1+\gamma)\, p_i + p_{i-1} = \frac{\rho \triangle x^2}{\triangle t} div\left(u_i^*\right). \tag{2.114}$$

Update new velocity uses

$$u_i^{k+1} = u_i^* - \frac{\triangle t}{2\rho \triangle x}\left(p_{i+1} - p_{i-1}\right) \tag{2.115}$$

when came across boundary between u_i and u_{i+1}, the new formula coupled with penetration ratio is

$$u_i^{k+1} = u_i^* - \frac{\gamma \triangle t}{2\rho \triangle x}\left(p_{i+1} - p_{i-1}\right). \tag{2.116}$$

Relationship with Darcy's Law

The penetration ratio γ is a dimensionless parameter which controls the volume of fluid go through canopy via influencing boundary condition. In nonporous case, the boundary conditions used for intermediate velocity and pressure in transportation and projection process respectively are

$$u_b^* = u_{intf} \tag{2.117}$$

$$\nabla p_b = 0 \tag{2.118}$$

where u_{intf} is the velocity of interface, i.e. parachute canopy. This gives the boundary condition of fluid at interface through projection formula

$$u^* = u + \frac{\Delta t}{\rho} \nabla p \Rightarrow u_b = u_{intf} \tag{2.119}$$

which means the velocity of fluid at interface is the velocity of the interface.

After coupling penetration ratio, the boundary condition of intermediate velocity is derived from interpolation

$$u_b^* = \begin{cases} (1-\gamma)\, u_{intf} + \gamma u_{i+1}, \;\; for\ u_i \\ (1-\gamma)\, u_{intf} + \gamma u_i, \;\; for\ u_{i+1} \end{cases} \tag{2.120}$$

and the boundary condition for pressure is

$$\nabla \tilde{p}_b = \frac{\tilde{p}_{i+1} - p_i}{\triangle x} = \gamma \nabla p_b. \tag{2.121}$$

Substitute above equations into projection formula gives

$$u_b = \begin{cases} u_{intf} + \gamma \left(u_{i+1} - u_{intf}\right) + \gamma \frac{\triangle t}{\rho} \nabla p_b, \;\; for\ u_i \\ u_{intf} + \gamma \left(u_i - u_{intf}\right) + \gamma \frac{\triangle t}{\rho} \nabla p_b, \;\; for\ u_{i+1} \end{cases} . \tag{2.122}$$

The difference between the fluid velocity at boundary and the boundary interface velocity is regarded as velocity due to porosity, which can be represented as

$$u_{poro} = -\gamma\frac{\Delta x^2}{2}\Delta u_b - \gamma\frac{\triangle t}{\rho}\nabla p_b. \tag{2.123}$$

Darcy's law describes the fluid through porous medium, which has the form

$$q = -\frac{\kappa}{\mu}\nabla p \tag{2.124}$$

where ∇p is the pressure gradient, κ is the intrinsic permeability of medium, μ is the fluid viscosity and q is the flux. The fluid velocity is related to flux q by void fraction ϕ

$$u_{poro} = \frac{q}{\phi}. \tag{2.125}$$

Equivalence the fluid velocity derived from numerical scheme with Darcy's law gives

$$\gamma = \frac{\frac{\kappa}{\phi\mu}\nabla p}{\frac{\triangle t}{\rho}\nabla p + \frac{\triangle x^2}{2}\triangle u} \tag{2.126}$$

assume $\triangle x^2 \ll \triangle t$ gives simplified form

$$\gamma \approx \frac{\kappa\rho}{\triangle t\phi\mu}. \tag{2.127}$$

2.3.4 Parachute Clusters and Collision Handling

Parachute clusters have been used in a wide range of applications and the design has been significantly improved over the years. Compared with excessively large single canopy, parachute cluster systems have a number of benefits, including the ability to rig and manufacture the system, backup protection, and excellent stability characteristics. The major deficiency of parachute clusters is 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 [63, 21, 54, 33, 80, 56]. 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. 2.2 demonstrates a cluster consisting of three G11 parachutes.

The major obstacle in parachute cluster simulation is collision handling. The *FronTier* library usually resolves collision and contact of surface by merging or bifurcation. However, to deal with fabric surface, a major revision has been added to the library and the resolving collision process can be summarized as:

1. Detect the intersection of two surfaces as triangle lists.
2. Find all triangles located on two corresponding surfaces within the intersection lists.
3. Lift two surfaces properly based on the type of interface.

Fig. 2.3 shows the difference of collision handling between the fluid-fluid interface and the fabric-fabric surface.

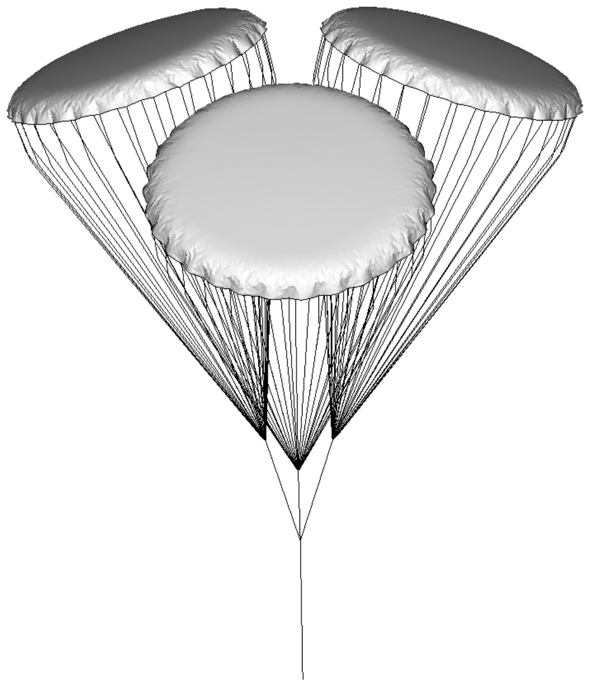


Figure 2.2: A cluster of three G11 parachutes.

2.3.5 American Option Pricing

The majority of exchange-traded options are American and there are many attempts to solve American option pricing problems which has no closed form solution. Based on the seminal work of Black and Scholes [8], Brennan and Schwartz [65] proposed the finite difference scheme for American options. Cox, Ross and Rubinstein [25] introduced the binomial tree model. Regression method proposed by Broadie and Detemple [14] tend to find the approximate solution. Boyle [12] propose the option pricing method based on Monte Carlo simulation.

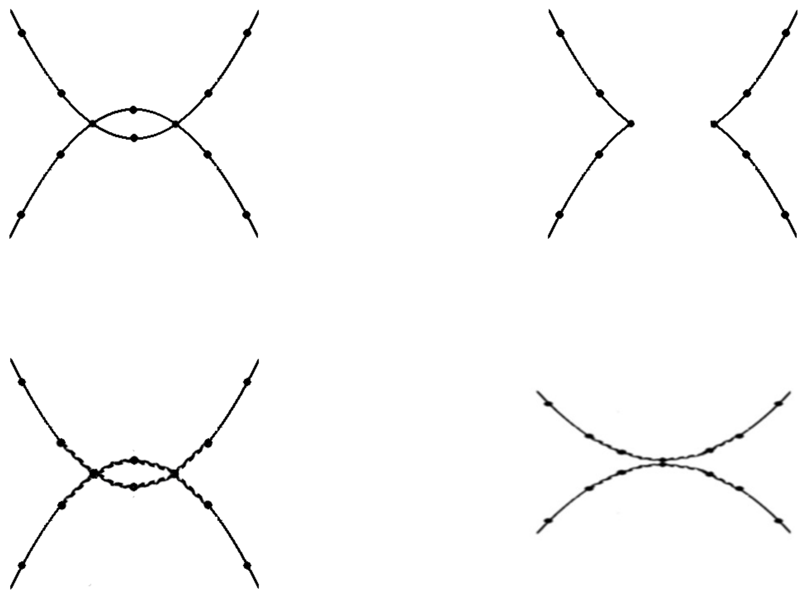


Figure 2.3: The difference between the collision handling for fluid interface and the fabric surface. The upper two plots show the topological bifurcation of fluid interface. The lower two plots show the repulsion of the fabric collision.

There are many recent papers focus on the option pricing based on exponential Lévy model. Barndorff [6] propose the Lévy model under normal inverse Gaussian process. Madan, Carr and Chang [60] obtained closed form solution for the return density and the prices of European options under variance gamma process. Cont and Voltchkova [23, 24] propose a general finite difference method for solving PIDE and prove the convergence of the scheme. Almendral and Oosterlee [3, 4, 5] numerically solve Merton's model and PIDE based on CGMY process by estimating the integration term with Fast Fourier

Transformation. Hirsa and Madan [43] price American options under variance gamma distribution and treat the integro-term by various sub-intervals. Matache, Petersdorff and Schwab [62] use θ-scheme for time stepping and a wavelet-Galerkin discretization of the integro-differential operator under variance gamma and CGMY processes. Sachs and Strauss [73] develop a second order scheme for Merton's Model through transforming the PIDE to eliminate convection term and solve dense linear system with conjugate gradient method. Briani, Natalini and Russo [13] develop an implicit-explicit Runge-Kutta methods to obtain higher order accuracy scheme for jump-diffusion model.

Eq. (1.65) can be solved with certain type of boundary condition to price specific option. In this specific case, we price American option based on exponential Lévy process under generalized hyperbolic distribution. To accelerate the program, we use heterogeneous algorithm to price both single option and multiple option prices.

Chapter 3

GPU Application

3.1 Introduction to GPU

In this section, the architecture of GPU is introduced, together with common methods to improve GPU performance which should be considered carefully while designing heterogeneous algorithms.

3.1.1 GPU Architecture

General Purpose Graphics Processing Units (GPGPU) computing [53] is to use 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 greatly.

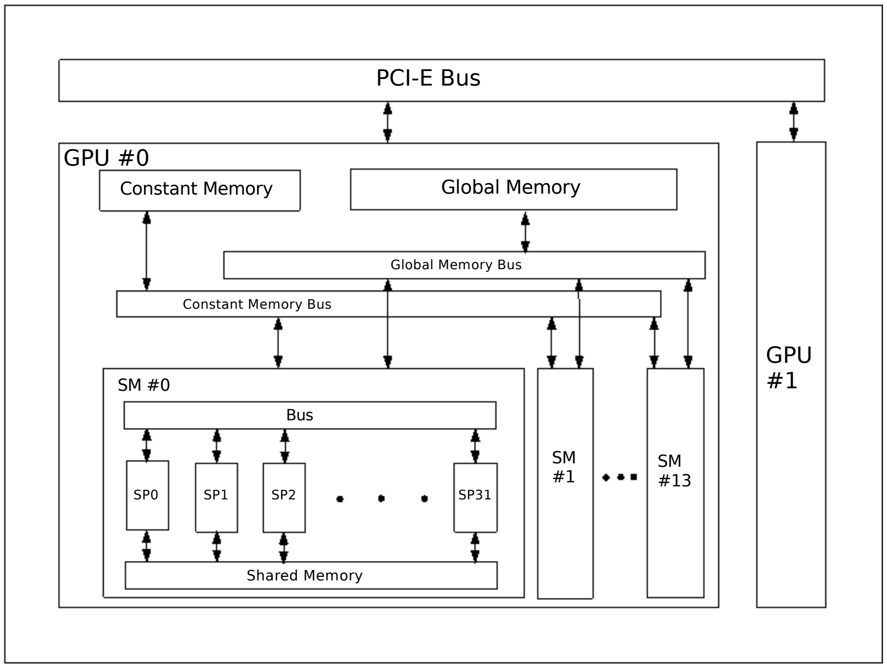


Figure 3.1: Architecture of multi-GPU devices. Each GPU hardware consists of memory (global, constant, shared) and 14 SMs. Each SM consists of 32 SPs and can run 1536 threads simultaneously.

Fig. 3.1 briefly shows the architecture of Modern GPU which can be viewed as a set of independent streaming multiprocessors (SMs) [26]. Each SM contains several scalar processors (SPs) which can execute integer and floating-point calculation, a multi-thread instruction unit and shared memory. When flow control transfer from CPU to GPU, functions in device are triggered which are called *kernels*. After a kernel is invoked, threads which are copies of this kernel will be distributed to all available multiprocessors. Based on the Single Instruction Multiple Data (SIMD) parallel programming style, all threads of a parallel phase will execute the same code. Threads are grouped into *blocks* and blocks are arranged into a *grid*. Blocks and grids can be one-dimensional, two-dimensional or three-dimensional arrays and they are assigned unique coordinates in CUDA framework as *threadId* and *blockId* to distinguish threads during execution. There are some 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:

1. Maximum number of threads per block is 1024.

2. Maximum dimension size of a thread block is $1024 \times 1024 \times 64$.

3. Maximum dimension size of a grid is $65535 \times 65535 \times 65535$.

Threads in a block are executed by processor in a single SM. Given available resources, multiple blocks can be assigned to the same SM. Blocks can be executed in any order which allows transparent scalability in the CUDA kernels. CUDA device bundles 32 threads into a warp and threads in the same warp are handled on the same multiprocessor. At every instruction issue time, a warp scheduler selects a warp that is ready to execute its next instruction, if any, and issues the instruction to the active threads of the warp. The number of clock cycles it takes for a warp to be ready to execute its next instruction is called the latency, and full utilization is achieved when all warp schedulers always have some instruction to issue for some warp at every clock cycle during that latency period, or in other words, when latency is completely ”hidden”.

CUDA threads may access data from multiple memory spaces during their execution [1]:

1. Each thread has *private local memory* or register.
2. Each thread block has *shared memory* visible to all threads of the block and with the same lifetime as the block.
3. All threads have access to the same *global memory.*
4. There are also two additional *read-only* memory spaces accessible by all threads: the *constant* and *texture* memory spaces.

The global, constant and texture memory spaces are optimized for different memory usages. The global, constant, and texture memory spaces are persistent across kernel launches by the same application.

3.1.2 Improving GPU Performance

There are three basic strategies to improve GPU performance:

1. Maximize parallel execution to achieve maximum utilization.
2. Optimize memory usage to achieve maximum memory throughput.
3. Optimize instruction usage to achieve maximum instruction throughput.

The best strategy is determined by the bottleneck of the program. Effort should be spent on the performance limiter after measuring and monitoring.

Maximize Utilization

Maximize utilization means to expose as much parallelism as possible and keep the system busy most of the time. This parallelism utilization can be optimized from different hierarchy. From *application level*, each processor should be assigned the type of work it does best, that is, serial workloads to the host and parallel workloads to the devices. This requires exact measuring of the most time consuming part of the algorithm and efficient algorithm design to realize optimal heterogeneous computing effect. From *device level*, parallel execution should be maximized between the multiprocessors of a device. From *Multiprocessor level*, the utilization efficiency depends on the number of resident warps which determine the number of active threads and parallelism effect. The number of blocks and warps that can reside and be processed together on the multiprocessor for a given kernel depends on the amount of registers and shared memory used by the kernel and the amount of registers and shared memory available on the multiprocessor. During computation, the variable we can specify is block size which will determine the efficiency in multiprocessor level. Thus, utilization optimization in multiprocessor level is transformed into finding the optimal block size. The calculation process can illustrated as following based on the parameter of out computation platform:

1. Maximum blocks per streaming multiprocessor is 8.

2. Maximum registers per streaming multiprocessor is 32768 (unit 32 bit).

3. Maximum shared memory per streaming multiprocessor is 49152 bytes.

4. 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). 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

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

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

The analysis process based on the actual execution resources is given in Sec. 4.5.1 with great detail about how to derive the optimized block size for gas dynamic simulation case. Numerical tests results are provided to support this optimization strategy.

Maximize Memory Throughput

The most important rule is to minimize data transfers with low bandwidth. This means to reduce data transfer between host and the device and also implies minimizing data transfers between global memory and the device by maximizing use of on-chip shared memory and caches. Thus, a typical programming pattern is:

1. Load data from device memory to share memory,
2. *Synchronize* with all other threads of the block,
3. Process the data in *shared memory*,
4. *Synchronize* again if necessary to make sure that shared memory has all updated result,
5. Write the result back to device memory.

To improve *data transfer between host and device*, one way is to move more code from host to device; another way is to batch many small transfers into a single large transfer which should perform better than making each transfer separately. In Sec. 4.5, the time spent on data transfer between host and device is demonstrated for each application cases.

Maximize Instruction Throughput

From the aspect of *arithmetic instructions*, this means use less arithmetic instructions which has low throughput which includes trading precision for speed when it does not affect the end result, such as using intrinsic instead of regular functions, single-precision instead of double-precision or flushing renormalized number to zero.

Threads from a block are bundled into warps for execution and threads within a warp must follow the same execution trajectory due to SIMD principal. All threads must execute the same instruction at the same time. Thus, any *flow control instruction* (if, switch, do, for, while) can significantly impact the effective instruction throughput by causing threads of the same warp to diverge. When came cross conditional branch depend on thread ID, threads in same warp may go to different branches which will give different instructions. Instead of executing different branches simultaneously, the CUDA platform will instruct the warp to execute the branches in order. While executing one branch, all threads that evaluated in other branches are effectively deactivated. As a result, the then and else parts are not executed in parallel, but in serial. This serialization can result in a significant performance loss and is called *thread divergence*.

_syncthreads() can impact performance by forcing the multiprocessor to idle which is called *synchronization instruction* limit.

3.2 GPU Accelerated PDE Scheme

There are some successful GPU implementations of numerical methods for partial differential equations, for example fast multipole methods [40], nodal discontinuous Galerkin methods [42], finite difference method [26], finite element methods [88], Fourier spectral methods and non-Fourier spectral methods [17].

The numerical scheme of specific partial differential equation is usually derived through discretization and can be represented as a large linear system of equations

$$Ax = d. \tag{3.1}$$

Usually, for explicit scheme, A and x are given to calculate d; for implicit scheme, A and d are given to solve x.

3.2.1 Explicit Scheme

Explicit time discretization has some benefits for parallelization. Due to its form, the parallelization process is much easier and intuitive. Kruger and Westermann [57] introduce numerical techniques for solving partial differential equations based on efficient representations of vectors and matrices on the GPU. Phillips and Massimiliano [67] demonstrate how to solve Himeno benchmark on clusters with GPUs using Jacobi relaxation and improve efficiency through optimizing memory bandwidth utilization. Alias etc. [2] introduce the solution of two-dimensional partial differential equations using parallel Gauss Seidel and Red-Black Gauss Seidel. Parand, Zafarvahedian and

Hossayni [66] using GPU to solve transient diffusion type equation by stable and explicit finite difference method and propose an optimal synchronization arrangement. Giles etc. [37] discuss the implementation of one-factor and three-factor PDE models on GPUs using both explicit and implicit schemes.

The calculation of Ax can be realized with the efficient matrix vector multiplication under the GPU framework. However, due to the property of finite difference scheme, A is usually a sparse matrix. As a result, it will be a waste to employ simply matrix vector product structure and a more efficient method is to assign the computing task of one or several specific grid points to each thread.

3.2.2 Efficient Tridiagonal Solver

As mentioned in Sec. 2.1.2, tridiagonal system

$$Ax = \begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix} = d \qquad (3.2)$$

can be numerically solved with $O(8n)$ operations. As for the parallel algorithm, Hockney [44] proposed the odd-even reduction and Stone [86] introduced the recursive doubling algorithm. These algorithm enable each processor to process exactly one row of the tridiagonal matrix. We refer the method in [30]. The basic idea is to eliminate variables from adjacent equations and reduce the system recursively until a single equation remains. Consider equation i with upper and lower equation

$$a_{i-1}x_{i-2} + b_{i-1}x_{i-1} + c_{i-1}x_i = d_{i-1} \tag{3.3}$$

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \tag{3.4}$$

$$a_{i+1}x_i + b_{i+1}x_{i+1} + c_{i+2}x_{i+2} = d_{i+1} \tag{3.5}$$

$-\frac{a_i}{b_{i-1}}$Eq. (3.3)+Eq. (3.4)$-\frac{c_i}{b_{i+1}}$Eq. (3.5) gives

$$a_i^{(1)}x_{i-2} + b_i^{(1)}x_i + c_i^{(1)}x_{i+2} = h_i^{(1)} \tag{3.6}$$

where $\alpha_i = -\frac{a_{i-1}}{b_{i-1}}$ and $\gamma_i = -\frac{c_i}{b_{i+1}}$

$$a_i^{(1)} = -\alpha_i a_{i-1} \tag{3.7}$$

$$b_i^{(1)} = b_i + \alpha_i c_{i-1} + \gamma_i a_{i+1} \tag{3.8}$$

$$c_i^{(1)} = \gamma_i c_{i+1} \tag{3.9}$$

$$h_i^{(1)} = h_i + \alpha_i h_{i-1} + \gamma_i h_{i+1}. \tag{3.10}$$

This transformation decompose the system of equations into two sub system of equations one involves $\{x_{2i}\}$, the other involves $\{x_{2i+1}\}$

$$\begin{pmatrix} b_1^{(1)} & c_1^{(1)} & 0 & \ddots \\ a_3^{(1)} & b_3^{(1)} & \ddots & 0 \\ 0 & \ddots & \ddots & c_{2k-3}^{(1)} \\ \ddots & 0 & a_{2k-1}^{(1)} & b_{2k-1}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \\ \vdots \\ x_{2k-1} \end{pmatrix} = \begin{pmatrix} d_1^{(1)} \\ d_3^{(1)} \\ \vdots \\ d_{2k-1}^{(1)} \end{pmatrix} \tag{3.11}$$

$$\begin{pmatrix} b_2^{(1)} & c_2^{(1)} & 0 & \ddots \\ a_4^{(1)} & b_4^{(1)} & \ddots & 0 \\ 0 & \ddots & \ddots & c_{2k-2}^{(1)} \\ \ddots & 0 & a_{2k}^{(1)} & b_{2k}^{(1)} \end{pmatrix} \begin{pmatrix} x_2 \\ x_4 \\ \vdots \\ x_{2k} \end{pmatrix} = \begin{pmatrix} d_2^{(1)} \\ d_4^{(1)} \\ \vdots \\ d_{2k}^{(1)} \end{pmatrix}. \tag{3.12}$$

Assume there are $2^q - 1$ variables, recursively apply above process and the result in level l can be represented as

$$a_i^{(l)} = \alpha_i a_{i-2^{l-1}}^{(l-1)} \tag{3.13}$$

$$b_i^{(l)} = b_i^{(l-1)} + \alpha_i c_{i-2^{l-1}}^{(l-1)} + \gamma_i a_{i+2^{l-1}}^{(l-1)} \tag{3.14}$$

$$c_i^{(l)} = \gamma_i c_{i+2^{l-1}}^{(l-1)} \tag{3.15}$$

$$h_i^{(l)} = h_i^{(l-1)} + \alpha_i h_{i-2^{l-1}}^{(l-1)} + \gamma_i h_{i+2^{l-1}}^{(l-1)} \tag{3.16}$$

where

$$\alpha_i = -\frac{a_i}{b_{i-2^{l-1}}},\ \gamma_i = -\frac{c_i}{b_{i+2^{l-1}}} \tag{3.17}$$

and $x_0 = x_{2^q} = 0$. Each thread is responsible for solving one variable, above process is repeated until hit two boundaries x_0 and x_{2^q}. Through this arrangement, each thread takes $O(\log_2 n)$ operations and total operations are increased to $O(n \log_2 n)$. However, consider the parallel effect, the computation operation is $O(\log_2 n)$.

To accelerate the computation efficiency, using low-latency shared memory would be much better than high-latency global memory. As a result, based on the limitation of thread per block, if the dimension of system of equation is less or equal than 1536, all threads can be calculated parallely. If the dimension exceeds the block thread limit, each thread should calculate several rows.

3.3 Heterogeneous Algorithms for Applications

3.3.1 WENO Scheme for Gas Dynamics

We illustrate the heterogeneous algorithm for advection solver based on the computing process of Euler equation. 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 at grid point $x_{j+\frac{1}{2}}$ can be calculated by WENO scheme using following steps:

1. Compute the 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 based on the value 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. Convert the flux calculated based on primitive variables into physical space flux.

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.

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.

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. 3.2 demonstrates the difference between cases without and with shared memory usage.

3.3.2 Spring Mass Model

Fig. 3.3 is the complete flow chart of the parachute simulation algorithm. Upon testing, we identified that solving spring model is the most time consuming and parallelizable part.

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 model.

3. Calculate the acceleration of each vertex.

4. Update the location of the vertex.

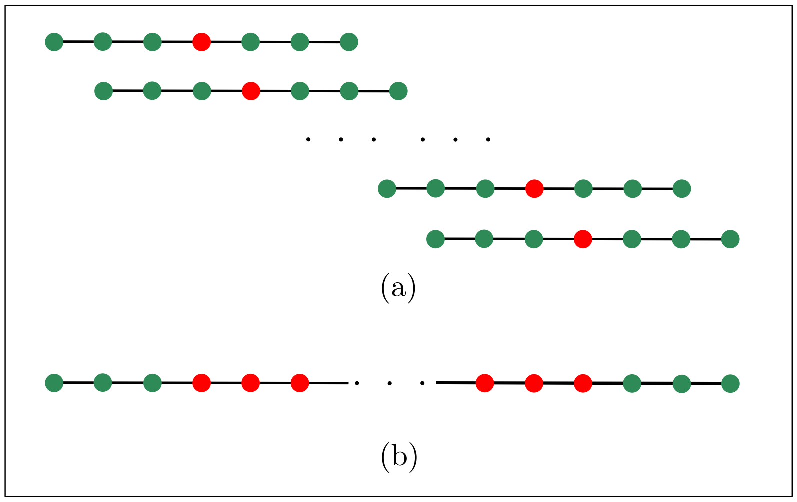


Figure 3.2: Fifth order WENO scheme stencils. Each point represents a computational node. Red points are updated by the threads while the green points are only used as data source. Each thread updates one red point only. (a) Without shared memory usage, each thread reads seven points' information. (b) 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 $512 \times 7 = 3584$ points' information; with shared memory, only $512 + 6 = 518$ points' information is necessary.

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 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:

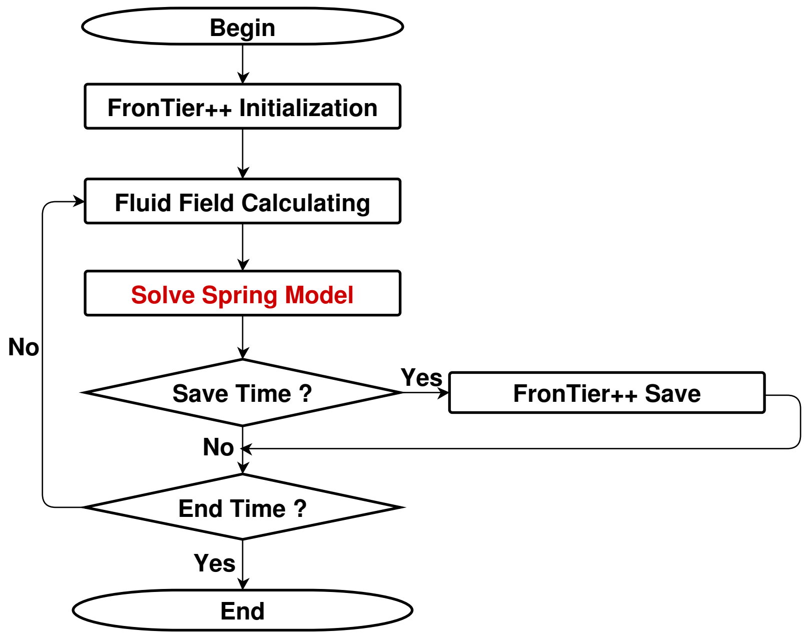


Figure 3.3: Flow chart of the complete algorithm. The computation of the spring model, which is marked by red color, is the most time consuming section. This part is calculated in parallel by the GPU device with multiple threads in order to improve the computational efficiency

1. Copy the position and velocity of each point to the device.

2. Invoke GPU kernel function to duplicate current position and velocity of each vertex for further calculation.

3. Invoke GPU kernel function to calculate acceleration of each point.

4. Invoke 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 to host.

The difference between this case and the gas dynamics case is that the mesh here is unstructured triangulated mesh while the mesh used in the gas dynamics problem is structured uniform mesh. Therefore, we can not use the shared memory of GPUs to further enhance the performance of the computation.

3.3.3 American Option Pricing

The flow chart in Fig. 4.21 (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. 4.21 (right). 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 to ensure 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 scale 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. Another major advantage is that this application involves little data transfer and spent most time intensive floating calculation. Thus, the joint CPU/GPU algorithm greatly outperforms pure CPU algorithm, especially in large number of options case, and Fig. 3.5

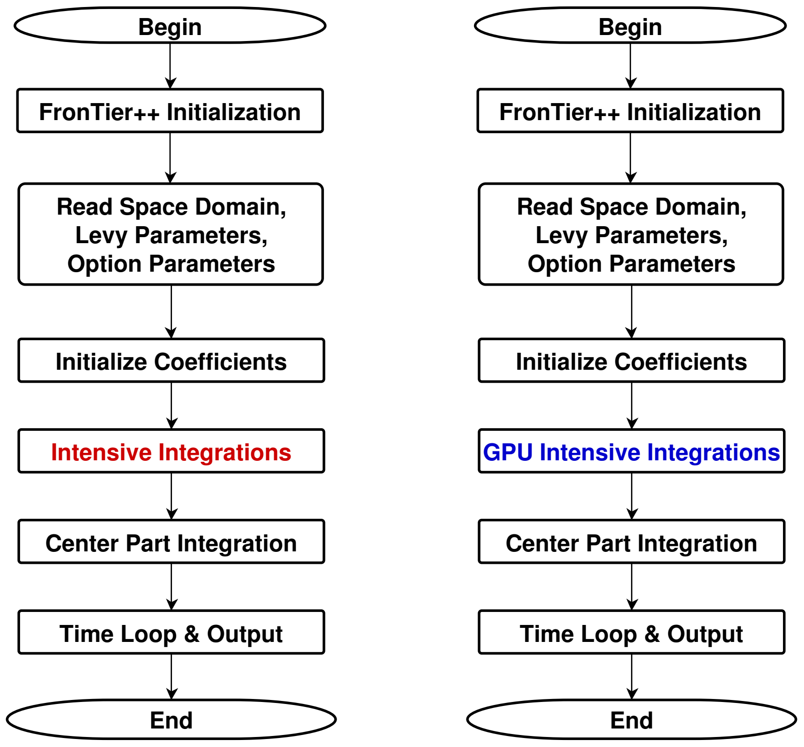

Begin
FronTier++ Initialization
Read Space Domain, Levy Parameters, Option Parameters
Initialize Coefficients
Intensive Integrations
Center Part Integration
Time Loop & Output
End
Begin
FronTier++ Initialization
Read Space Domain, Levy Parameters, Option Parameters
Initialize Coefficients
GPU Intensive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

Figure 3.4: Without-GPU (Left) and With-GPU (Right) flow charts for single American option pricing.

illustrates this process.

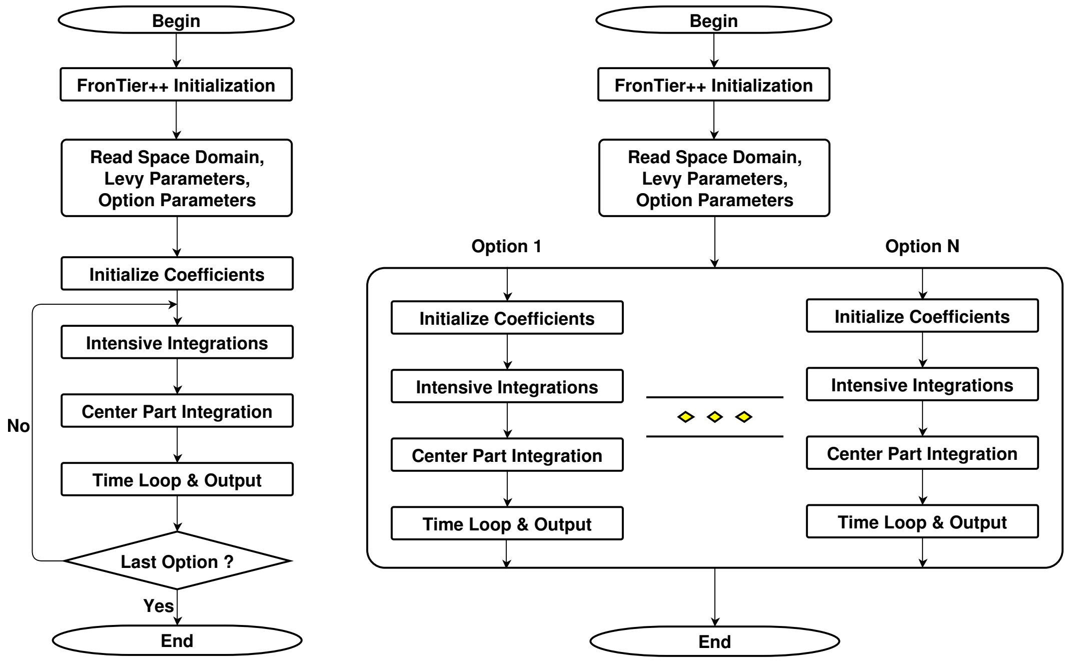


Figure 3.5: Without-GPU (Left) and With-GPU (Right) flow charts for multiple American options pricing.

Chapter 4

Numerical Results

4.1 Experiment Platform

Numerical experiments based on both the CPU and the GPU are 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. Fedora 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.

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.

4.2 Advection Solver for Gas Dynamic

The test case for the Euler equations is the shock-tube problem. This problem is a well-known Riemann problem introduced by Sod [79]. The solution domain is $[-1, 1]$, and the initial conditions are Eq. (4.1).

$$(\rho, u, p) = \begin{cases} (1, 0, 1)\,, & x \leqslant 0 \\ (0.125, 0, 0.1)\,, & x > 0 \end{cases} \tag{4.1}$$

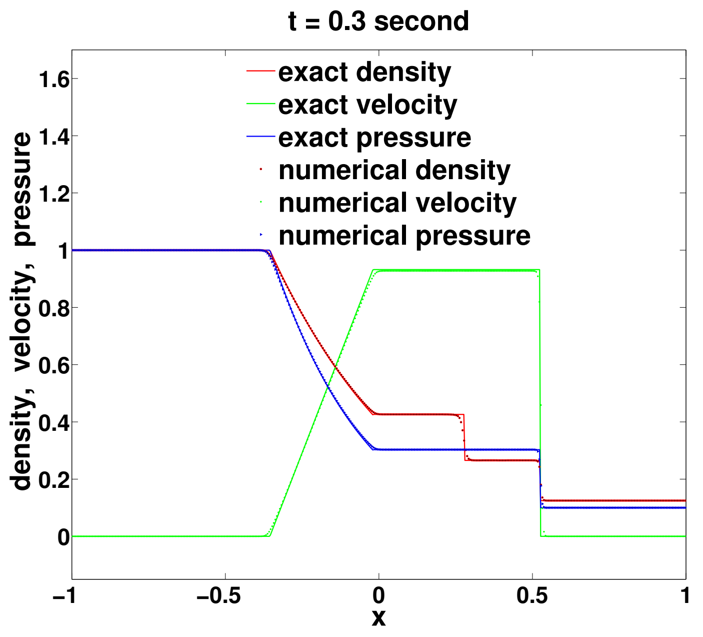
The results are demonstrated in Fig. 4.1. As we can see, this algorithm is numerically convergent under mesh refinement test and the numerical solution obtains high accuracy when compared with analytical solution.

4.3 Parachute Simulation

In this section, several parachute based simulation results are demonstrated. Refer [78] for the convergence test of spring model and the verification and validation of parachute inflation simulation.

4.3.1 Angled Deployment

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 [74]. 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 [61]. 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



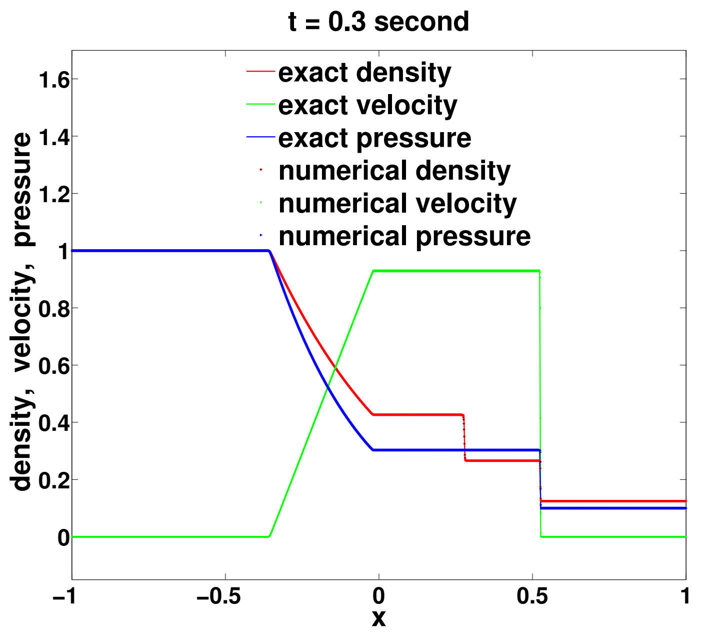


Figure 4.1: Sod problem results, the solution domain is $[-1, 1]$. The mesh size of the first figure is 400 and the second one's is 3200.

force to essentially zero and results in a free fall.

Numerical solution becomes a valuable tool for the parachute design if 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. 4.2 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. 4.3, 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. 4.4 shows such inverted canopy.

4.3.2 Canopy Porosity Simulation

The motion of fluid around a porous arch-shaped interface is simulated in the penetration ratio range $\gamma = 0, 0.25, 0.5, 0.75$ based on the model introduced in Sec. 2.3.3.

The density of fluid is 1.2 and the viscosity is 0.000628; the simulation region is $[0, 12] \times [-3, 12]$. Fig. 4.5 shows the magnitude of velocity in x direction; Fig. 4.6 shows the magnitude of velocity in y direction; Fig. 4.7 shows the vorticity around the parachute canopy. We can conclude that higher

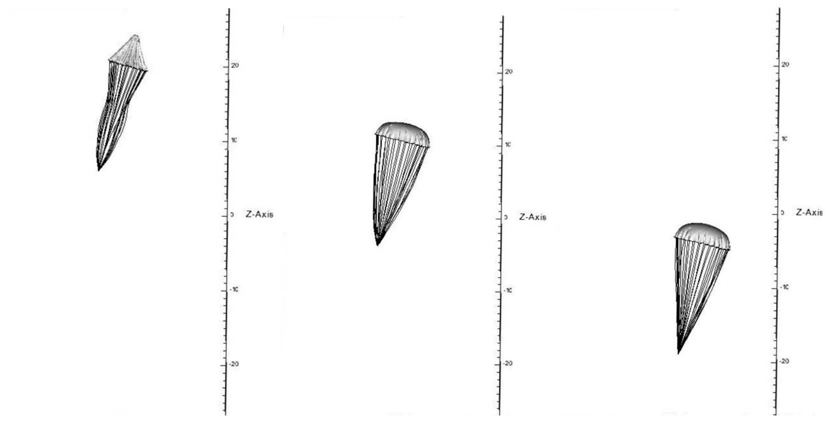


Figure 4.2: Angled deployment of C-9 parachute with the flow. The deployment starts with a 15° angle between the initial parachute and the direction of flow. The parachute experiences only slight asymmetry of the canopy. The plots show the parachute at (from left to right) $t = 0sec$, $t = 1.5sec$ and $t = 3.0sec$ respectively.

penetration ratio, as a result of high porosity, should lead to higher stability and less vorticity. This simulation result is consistent with experiments and observations.

4.4 American Option Pricing

In the first experiment, we try to prove the convergence of the algorithm and estimate the error via computing European put options based on various processes. Table 4.2, Table 4.3, Table 4.4 and Table 4.5 record the numerical result under variance gamma process, CGMY model (tempered stable process), normal inverse Gaussian process and generalized hyperbolic process respectively. In each table, we list the total error l_∞ , error at strike price

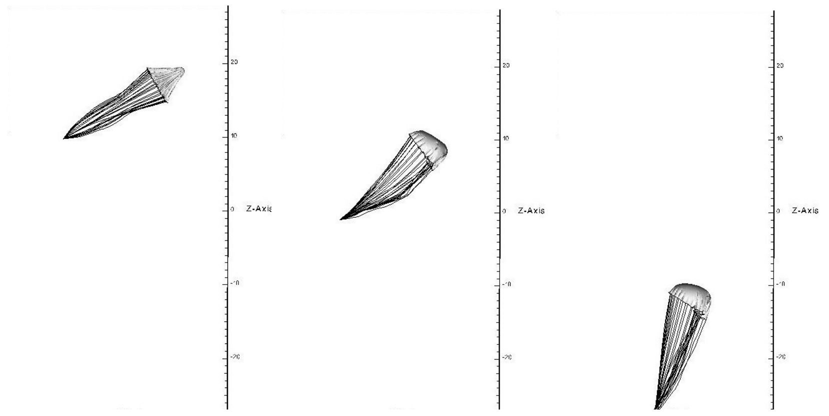


Figure 4.3: Angled deployment of C-9 parachute with the flow. This sequence starts with a 60° angle between the initial parachute and the direction of the flow. In this case, the canopy skirt is dangerously wrapped at the lower side of the canopy. The plots show the parachute at (from left to right) $t = 0sec$, $t = 1.5sec$ and $t = 3.0sec$ respectively.

K and CPU operation time for different number of grids. The parameters of objective option is $r = 0.1$, $\sigma^2 = 0.4$, $K = 1$ and $\tau = 1$. Since the error convergence rate is between 2 and 4 when we double the grid, the algorithm is convergent and the accuracy is between first and second order. And the CPU operation time is quite ideal for variance gamma process, CGMY model and normal inverse Gaussian process. For generalized hyperbolic process, we have to estimate double numerical integration which takes more operation time when compared with other processes.

Fig. 4.8 shows part of the European put option prices under different underlying processes with same parameters as above tables, as well as the Black-Scholes result. Among all five lines, the European put option price derived from Black-Scholes is the lowest. Since PIDE are based on exponential

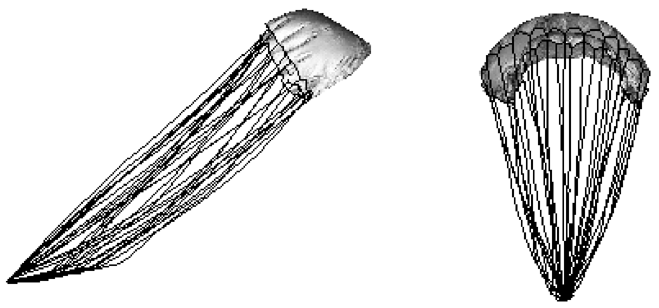


Figure 4.4: Inversion of the parachute canopy during an angled drop. The alignment of the parachute started with a 75° angle with the direction of the velocity. A complete inversion occurs at $t = 2sec$. The two plots are views of the inverted canopy from different directions.

Lévy model which characterizes jumps, the results will be higher than corresponding Black-Scholes result.

N	M	l_∞-error	error at K	CR	CPU
100	25	9.8400e-04	7.7600e-04	-	0.01
200	50	4.1900e-04	3.1100e-04	2.4952	0.03
400	100	1.7100e-04	1.2200e-04	2.5492	0.12
800	200	5.6000e-05	3.9000e-05	3.1282	0.69

Table 4.2: errors and convergence of the variance gamma process and parameters are $c = 1.0$, $\lambda_+ = 7.0$ and $\lambda_- = 9.0$

The second experiment is to explore the effects of four parameters of the generalized hyperbolic process on option prices. In Fig. 4.9, Fig. 4.10,

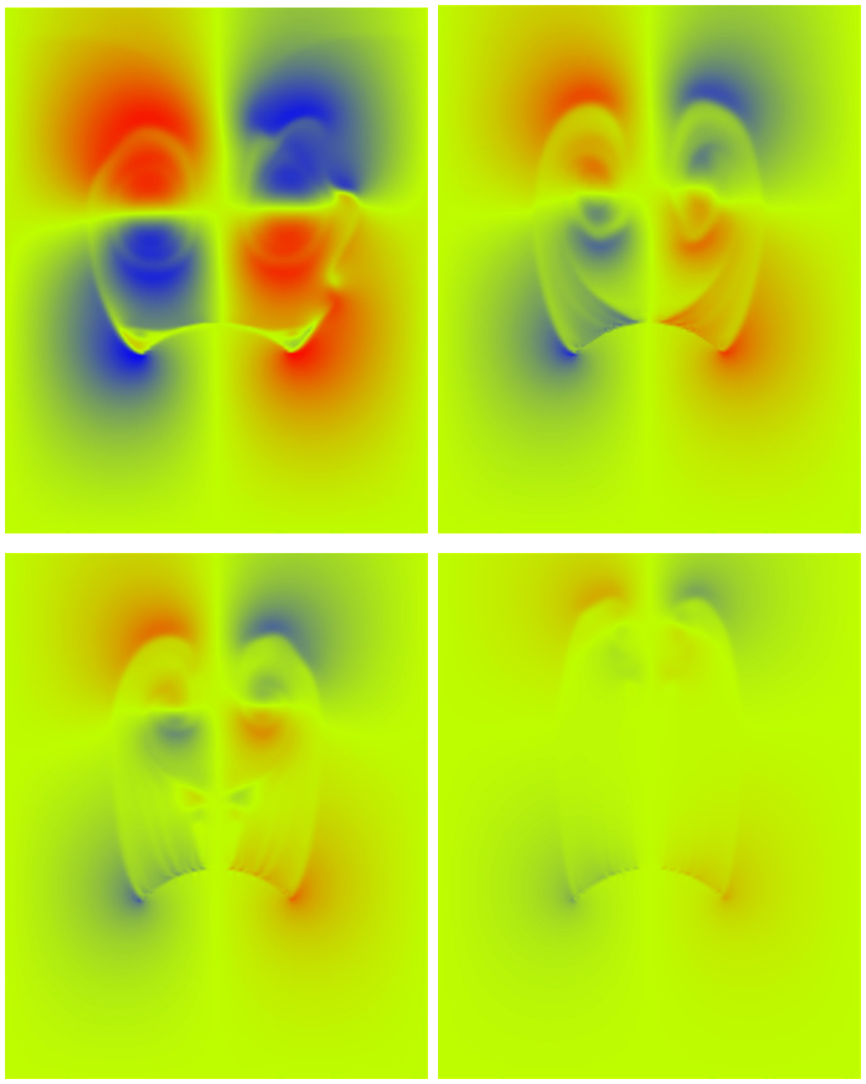


Figure 4.5: Porosity simulation, velocity in x direction, with penetration ratio from left to right, top to bottom: $\gamma = 0, 0.25, 0.5, 0.75$

Fig. 4.11, Fig. 4.12 and Fig. 4.13, we demonstrate American put option prices with three fixed parameters and one flexible parameter; then, these option prices are compared with the prices derived from the Black-Scholes equation quantitatively.

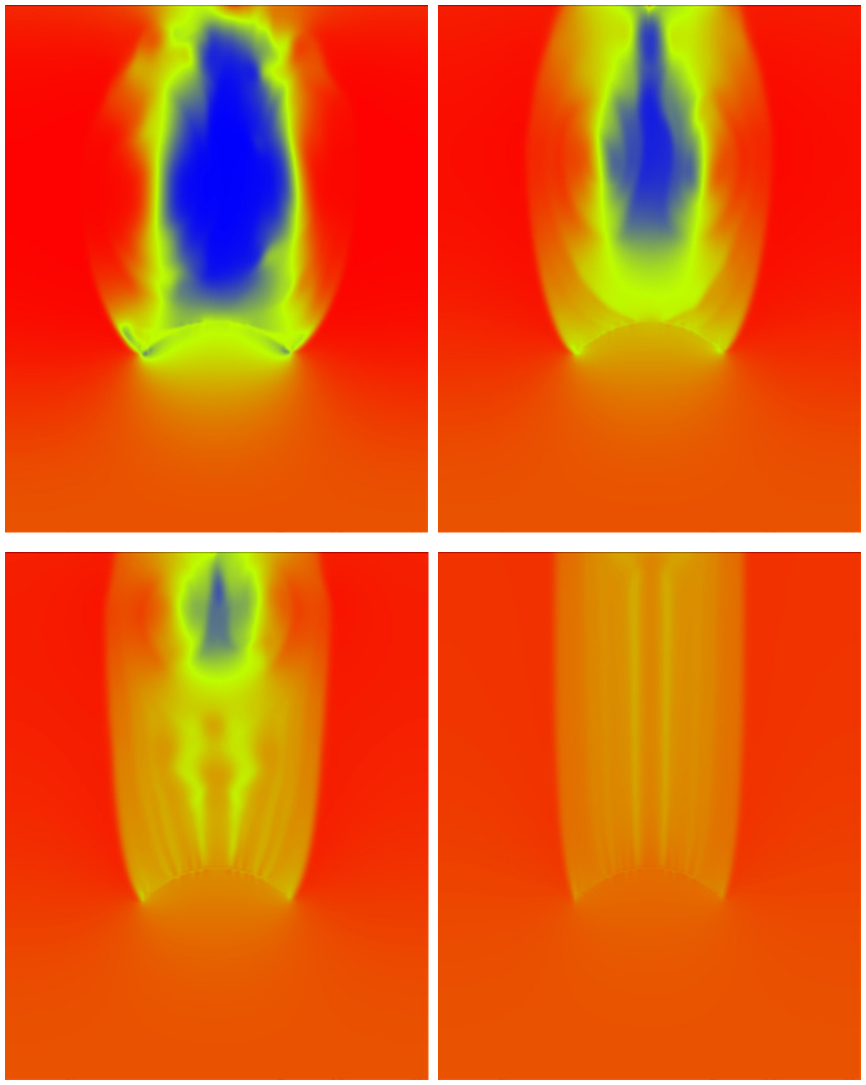


Figure 4.6: Porosity simulation, velocity in y direction, with penetration ratio from left to right, top to bottom: $\gamma = 0, 0.25, 0.5, 0.75$

In Fig. 4.9, α, β, δ are fixed and larger λ will lead to higher option prices. In Fig. 4.10, λ, β, δ are fixed and larger α leads to lower option prices. In Fig. 4.11 and Fig. 4.12, fixed parameters are λ, α, δ and when $\beta > 0$, larger β will lead to higher option prices; when $\beta < 0$, lower β will lead to higher

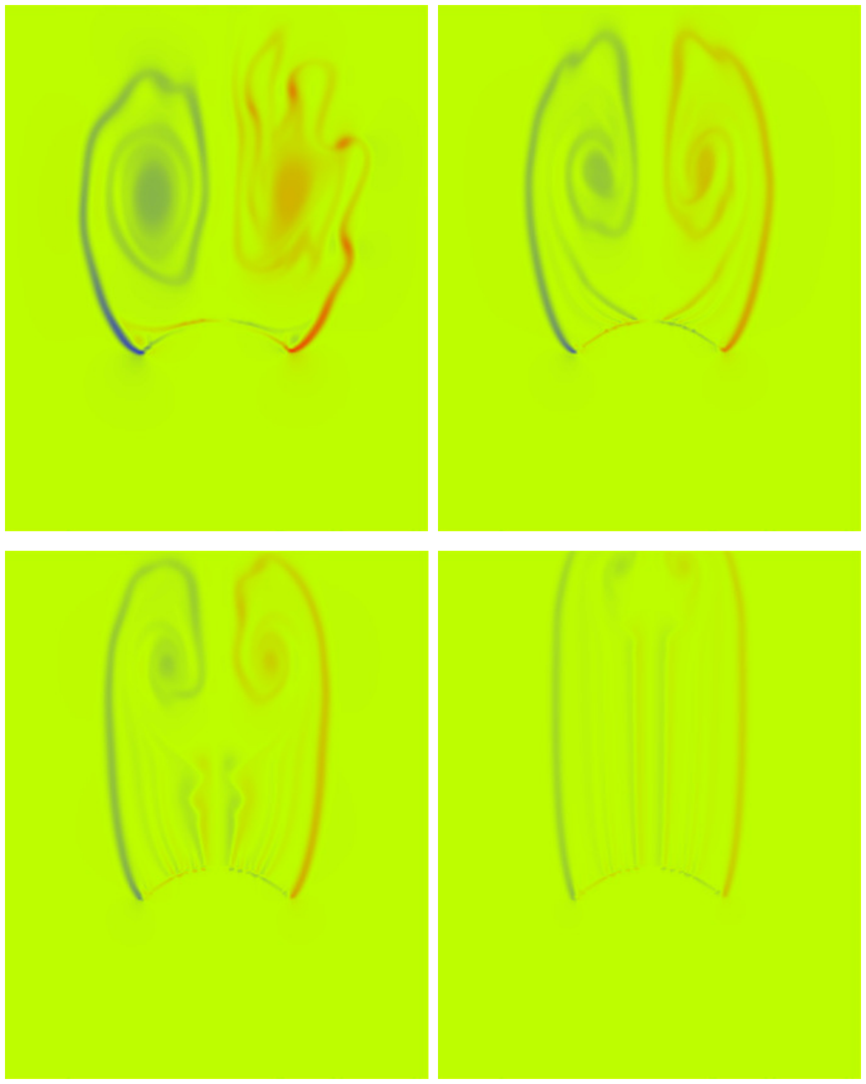


Figure 4.7: Porosity simulation, vorticity, with penetration ratio from left to right, top to bottom: $\gamma = 0, 0.25, 0.5, 0.75$

option prices. In Fig. 4.13, we fix parameters λ, α, β and find out larger δ will lead to higher option prices.

The third experiment is to compute the early exercise boundary for American put options. In Fig. 4.14, Fig. 4.15 and Fig. 4.16, we fix three parameters

N	M	l_∞-error	error at K	CR	CPU
100	25	0.0030520	0.0020610	-	0.01
200	50	0.0012890	8.2100e-04	2.5104	0.01
400	100	5.0000e-04	3.0800e-04	2.6656	0.12
800	200	1.5100e-04	9.0000e-05	3.4222	0.69

Table 4.3: Errors and convergence of the CGMY model and parameters are $C = 1.0$, $G = 7.0$, $M = 9.0$ and $Y = 0.7$

N	M	l_∞-error	error at K	CR	CPU
100	25	0.0011290	9.5900e-04	-	0.00
200	50	5.3100e-04	4.3000e-04	2.2302	0.02
400	100	2.2900e-04	1.8000e-04	2.3889	0.11
800	200	7.6000e-05	5.9000e-05	3.0508	0.66

Table 4.4: Errors and convergence of the normal inverse Gaussian process and parameters are $\alpha = 8.15$, $\beta = -2.5$ and $\delta = 0.767$

N	M	l_∞-error	error at K	CR	CPU
100	25	0.0026030	0.0016540	-	0.30
200	50	0.0012290	7.8600e-04	2.1043	0.84
400	100	5.3000e-04	3.4100e-04	2.3050	1.90
800	200	1.7400e-04	1.1000e-04	3.1000	4.88

Table 4.5: Errors and convergence of the generalized hyperbolic process and parameters are $\lambda = 1$, $\alpha = 8.15$, $\beta = -2.5$ and $\delta = 0.767$

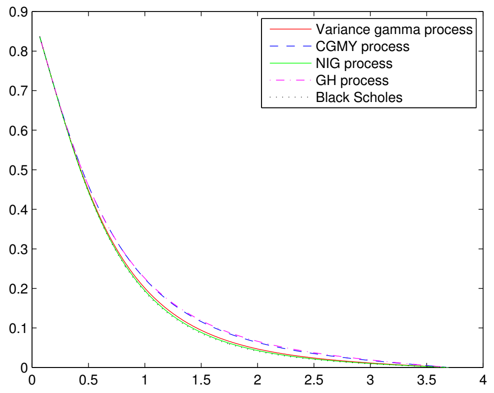


Figure 4.8: European option prices based on the variance gamma process with parameters $c = 1.0$, $\lambda_+ = 7.0$ and $\lambda_- = 9.0$; the CGMY model with parameters $C = 1.0$, $G = 7.0$, $M = 9.0$ and $Y = 0.7$; the normal inverse Gaussian process with parameters $\alpha = 8.15$, $\beta = -2.5$ and $\delta = 0.767$; the generalized hyperbolic process with parameters $\lambda = 1$, $\alpha = 8.15$, $\beta = -2.5$ and $\delta = 0.767$.

and verify the effect of the fourth parameter on the early exercise boundary of American put options. At fixed τ, given same α, β, δ, larger λ will lead to lower early exercise prices. Given same λ, β, δ, larger α will lead to higher early exercise prices. Given fixed parameters λ, α, δ, when $\beta > 0$, larger β will lead to lower early exercise prices; when $\beta < 0$, smaller β will lead to lower exercise prices. Fix λ, α, β, larger δ will lead to lower early exercise prices.

4.5 Acceleration Effect of Heterogeneous Algorithms

4.5.1 Optimized Resource Allocation

In Sec. 3.1.2, 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 gas dynamic simulation case. 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 4.6, 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 parallelization. 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.

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 4.6: Execution capacity analysis. In this application, each thread needs 63 registers and each block needs $(block\ size + 2 \times ghost\ size) \times (number\ of\ shared\ doubles\ per\ thread) \times sizeof(double)$ shared memory where $ghost\ size = 3$, number of shared doubles per thread is decided by the algorithm which is 11 and $sizeof(double) = 8$ is decided by the compiler. Given these parameters and the size of registers and shared memory per SM, maximum number of blocks can be calculated for given block size.

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. Fig. 4.17 shows the results. In Table 4.6 we conclude that the effective threads per device is 3584 when the block size is 32. In the the right upper subplot (subplot (2)) of Fig. 4.17, 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 $7168 = 3584 \times 2 < 8192$, 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. 4.17. In sum, from the consistency of the analysis in the Table 4.6 and the results in Fig. 4.17, 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.

4.5.2 GPU Application for Gas Dynamics

The operation time spent on solving sod problem measured by the CPU clock and the time for GPU intensive computational part are collected in Table 4.7. Fig. 4.18 displays the GPU time and CPU time for each step. From Table 4.7 and Fig. 4.18 we can conclude that the application of the GPUs has clear advantage in the computation of the flux using the advection solver based on WENO scheme and Runge-Kutta method. The pure computation time accelerated for 29-86$\times$. For this application, data fetching costs even more time than computation. However, consider time spent on both the data transfer and computation, the heterogeneous algorithm is still 4-20$\times$ more efficient than pure CPU algorithm.

Mesh	CPU Time ($micros$)	GPU Time ($micros$)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
1024	3427	336	118	272	726	29.04	4.72
2048	6896	349	146	273	768	46.97	8.93
3072	10053	365	152	283	800	65.61	12.47
4096	13912	340	183	365	888	75.40	15.54
5120	16774	411	215	321	947	77.49	17.59
6144	20126	397	249	308	954	80.28	20.95
7168	23463	525	271	396	1192	86.10	19.57
8192	25376	509	363	365	1237	69.53	20.41

Table 4.7: 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 Table 4.6, we choose one of the best block size 128 here. The hybrid code is 8-20× 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.

4.5.3 GPU Application for Spring Model

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 constant is $1000N/m$.

The total operation time recorded by CPU clock and time for GPU intensive computational part are collected in Table 4.8, together with the barplot Fig. 4.19 which shows the time spent on each part. As we can see, this application spent little time on data transfer and the heterogeneous algorithm achieves 5-6× speedup.

Mesh	CPU Time (*micros*)	GPU Time (*micros*)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
2641	223565	171	41328	644	42143	5.41	5.30
4279	294037	340	42200	1239	43734	6.98	6.72
6466	376137	376	51180	530	52086	7.35	7.22
9064	481941	786	70444	2241	73471	6.84	6.56
12361	554545	687	85744	727	87158	6.47	6.36
15567	601600	1503	96310	1066	98879	6.25	6.08

Table 4.8: 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× faster than the pure CPU code for computing the intensive part.

4.5.4 GPU Application for American Option Pricing

Single American Option Pricing

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 4.9 and an intuitive comparison is given in Fig. 4.20.

Mesh	CPU Time ($micros$)	GPU Time ($micros$)				CPU/GPU	
		H2D	Compute	D2H	Total	Compute	Total
128	942539	752740	540774	55	1293569	1.74	0.73
256	2825369	757925	1034025	59	1792009	2.73	1.58
512	8088534	757555	2124162	115	2881832	3.80	2.80
1024	22214927	780483	3867521	79	4648083	5.74	4.78
2048	55254249	810256	6475537	87	7285880	8.53	7.58
4096	134558431	755945	11294322	256	12050523	11.91	11.17

Table 4.9: Operation time of single option pricing under the generalized hyperbolic distribution, parameters $\lambda = 1.0, \alpha = 8.15, \beta = -2.5, \delta = 0.767$

From Table 4.9 and Fig. 4.20 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.

Multiple American Options Pricing

To meet the requirements of timely and efficiently pricing of multiple options, we transform the CPU algorithm to joint CPU/GPU algorithm. Table 4.10 and Fig. 4.21 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	9.08	9.08
64	70723520	3484	3900757	324	3904565	18.13	18.11
128	141447040	2752	3954747	195	3957694	35.77	35.74
256	282894080	3199	3955152	382	3958733	71.53	71.46
512	565788160	3200	4097343	1429	4101972	138.09	137.93
1024	1131576320	3232	4366600	1064	4370896	259.14	258.89
2048	2263152640	3257	4767045	1945	4772247	474.75	474.23
4096	4526305280	3328	9540157	3352	9546837	474.45	474.12

Table 4.10: Operation time of multiple options pricing under the generalized hyperbolic distribution, parameters $\lambda = 1.0, \alpha = 8.15, \beta = -2.5, \delta = 0.767$

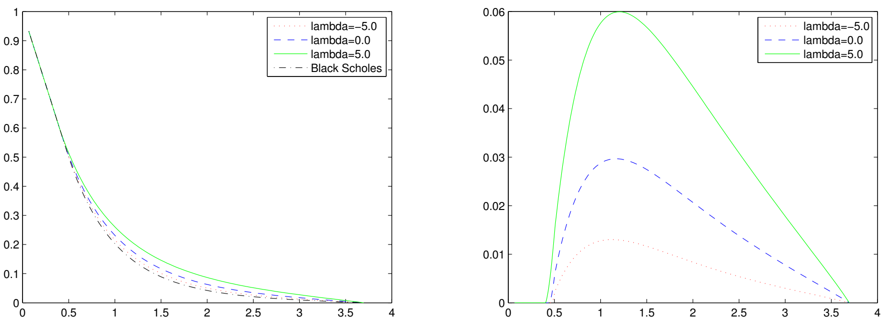


Figure 4.9: Left: American option prices under the generalized hyperbolic distribution; Right: difference between option prices based on PIDE and Black--Scholes. Other parameters are $\alpha = 8.15, \beta = -2.5, \delta = 0.767$.

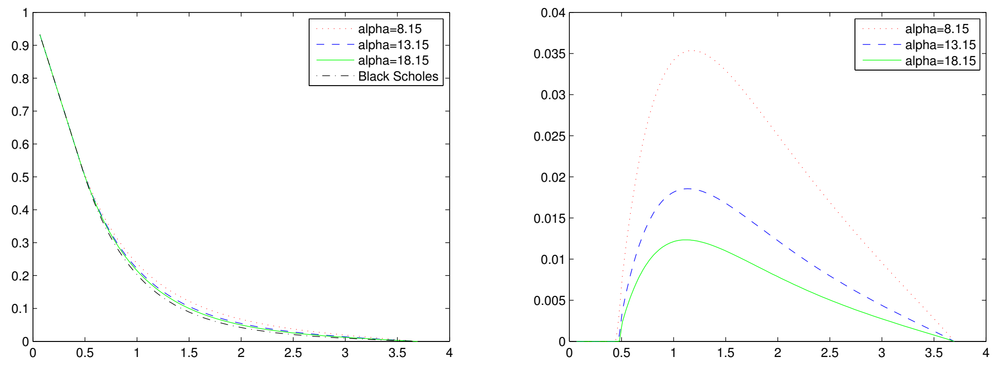


Figure 4.10: Left: American option prices under the generalized hyperbolic distribution; Right: difference between option prices based on PIDE and Black--Scholes. Other parameters are $\lambda = 1.0, \beta = -2.5, \delta = 0.767$.

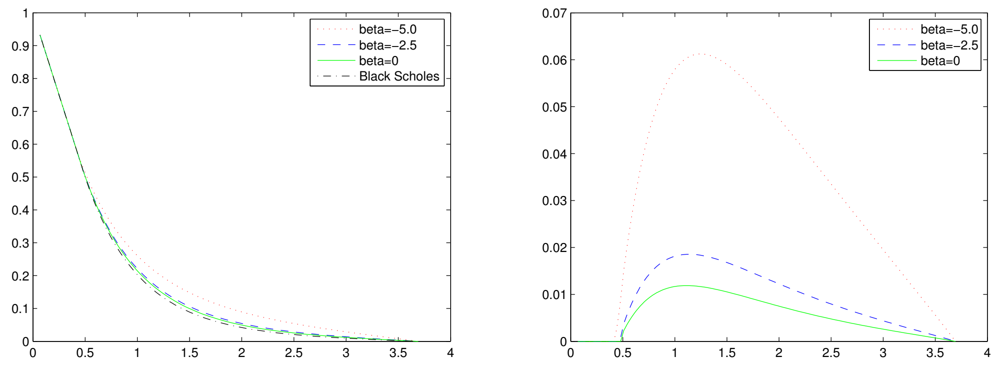


Figure 4.11: Left: American option prices under the generalized hyperbolic distribution; Right: difference between option prices based on PIDE and Black--Scholes. Other parameters are $\lambda = 1.0, \alpha = 8.15, \delta = 0.767$.

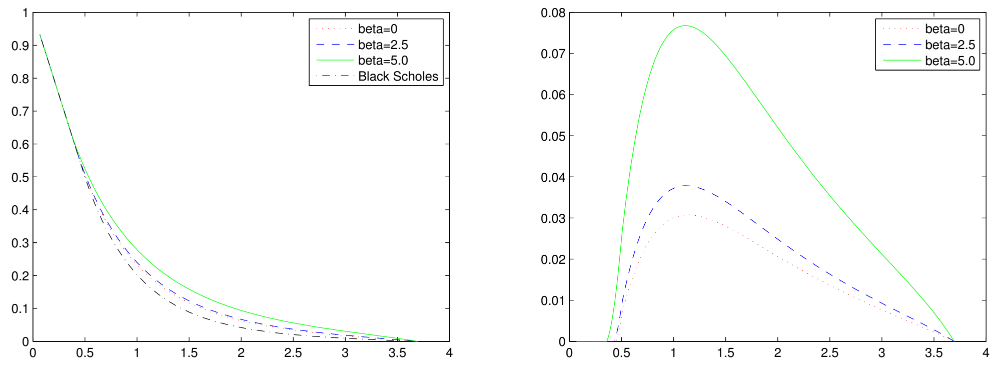


Figure 4.12: Left: American option prices under the generalized hyperbolic distribution; Right: difference between option prices based on PIDE and Black--Scholes. Other parameters are $\lambda = 1.0, \alpha = 8.15, \delta = 0.767$.

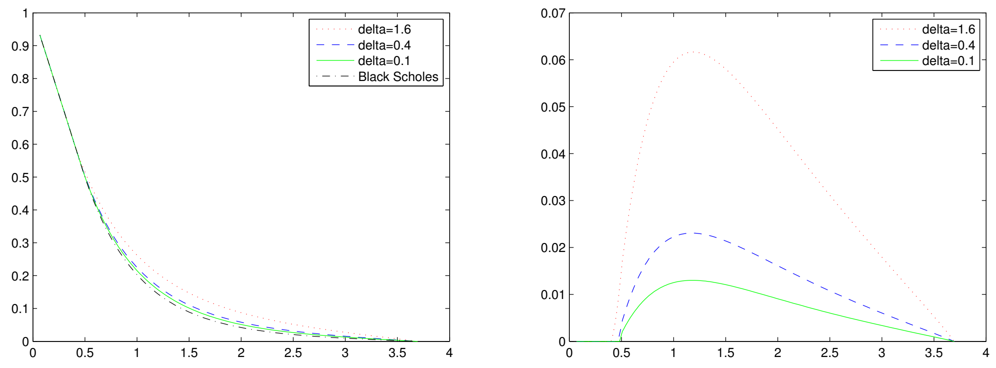


Figure 4.13: Left: American option prices under the generalized hyperbolic distribution; Right: difference between option prices based on PIDE and Black--Scholes. Other parameters are $\lambda = 1.0, \alpha = 8.15, \beta = -2.5$.

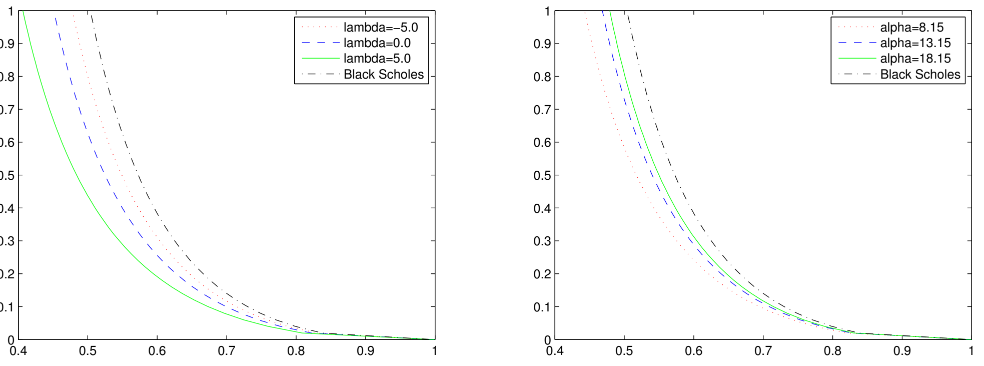


Figure 4.14: Early exercise boundaries for American options based on the generalized hyperbolic distribution (x-axis: exercise price; y-axis: time to maturity). Parameters are (Left) $\alpha = 8.15, \beta = -2.5, \delta = 0.767$; (Right) $\lambda = 1.0, \beta = -2.5, \delta = 0.767$.

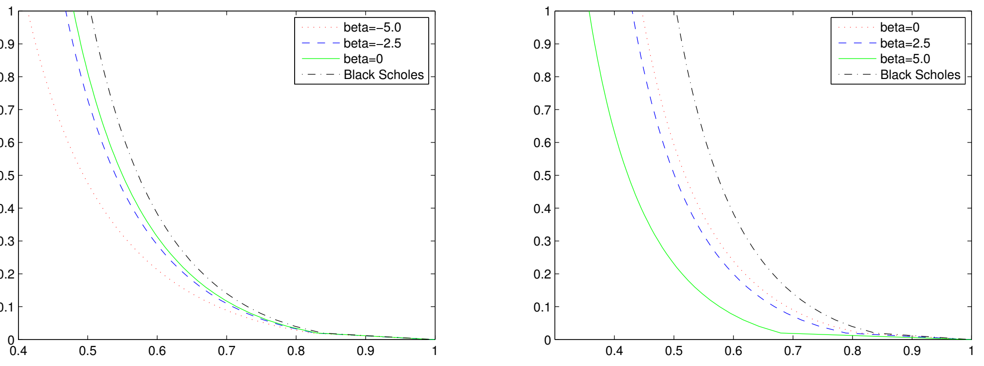


Figure 4.15: Early exercise boundaries for American options based on the generalized hyperbolic distribution (x-axis: exercise price; y-axis: time to maturity). Parameters are $\lambda = 1.0, \alpha = 8.15, \delta = 0.767$.

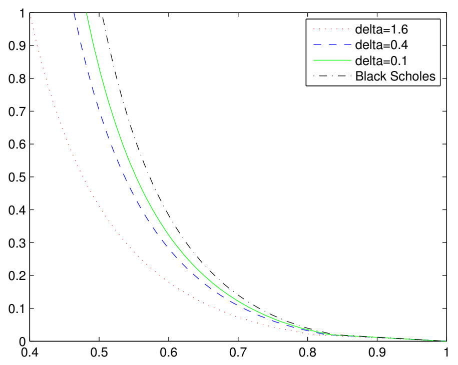


Figure 4.16: Early exercise boundaries for American options based on the generalized hyperbolic distribution (x-axis: exercise price; y-axis: time to maturity). Parameters are $\lambda = 1.0, \alpha = 8.15, \beta = -2.5$.

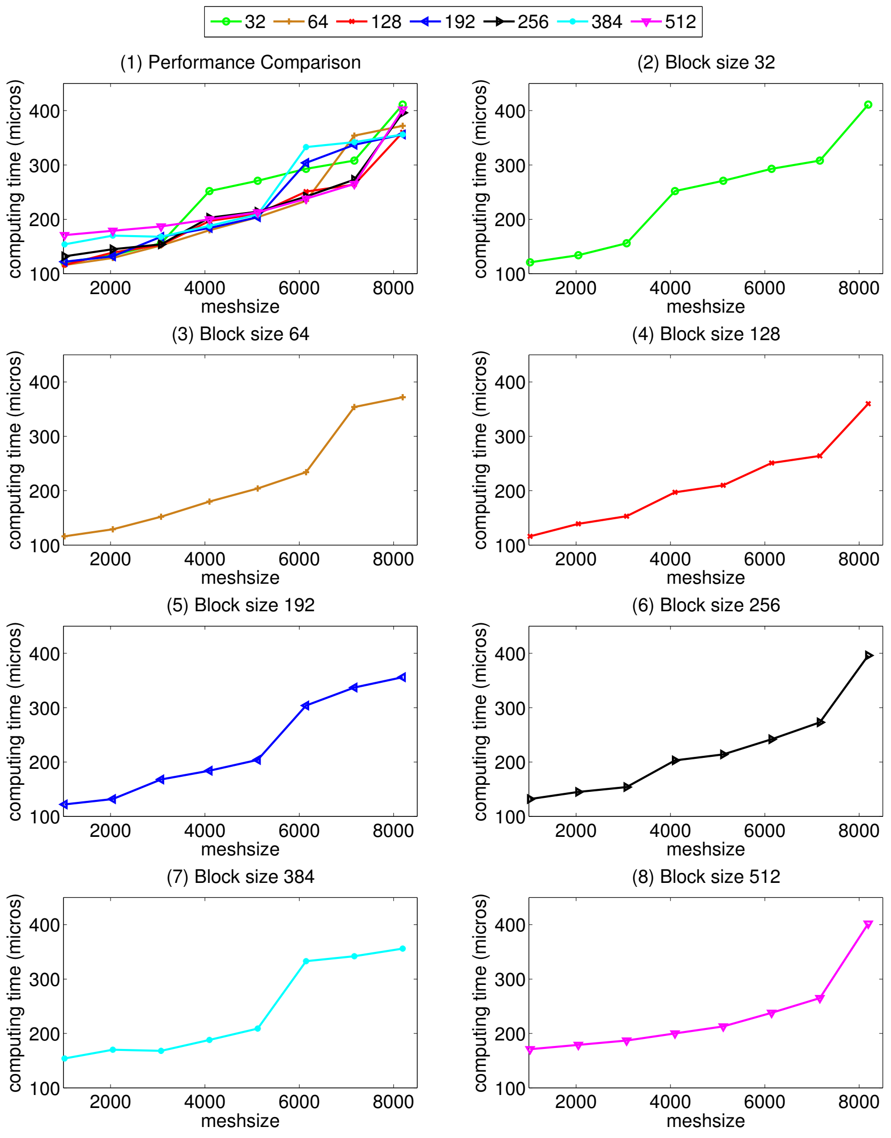


Figure 4.17: GPU computing performance on different block sizes and mesh sizes for Sod problem.

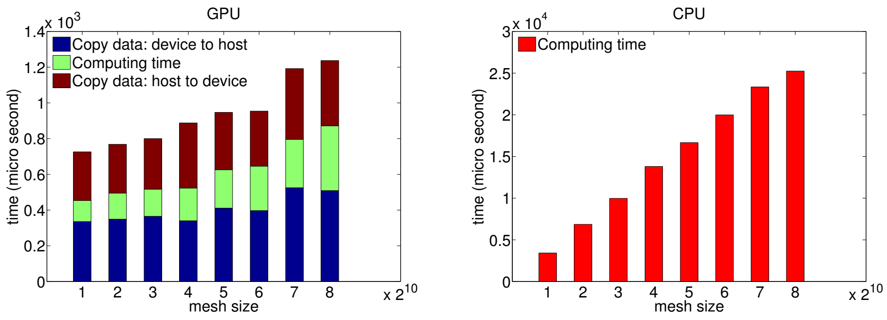


Figure 4.18: GPU (left) and CPU (right) time per step for gas dynamic simulation based on WENO. The GPU computing time is 29-86× faster than CPU's. However, GPU total time is only 4-20× faster than CPU's. This is due to the time used to transfer the data between the host and the device in GPU. From the left plot, we can clearly see that the time spent on copying data is at least twice larger than the time spent on computing.

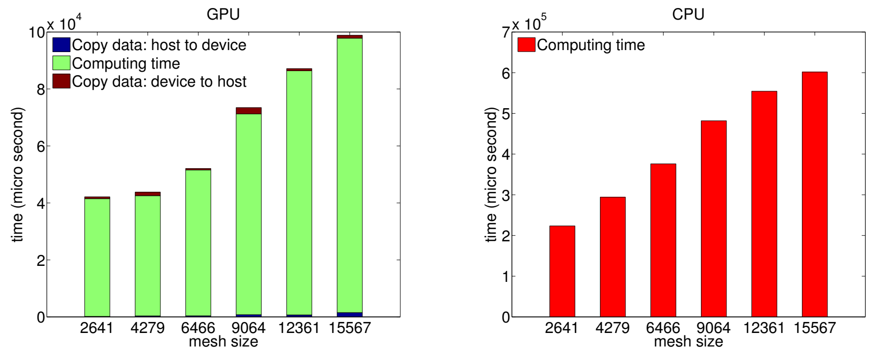


Figure 4.19: GPU (left) and CPU (right) time per step for spring model simulation. The GPU computing time and total time are 5-6× faster than CPU's. This is due to the negligible time used to transfer the data between the host and the device.

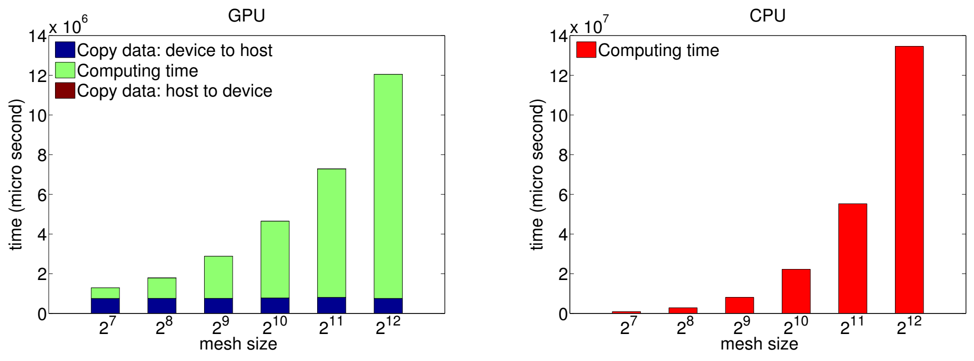


Figure 4.20: GPU (left) and CPU (right) time per step for single option pricing. The GPU computing time is 1.7-12× faster than CPU's. However, the performance of GPU is worse when the mesh size is small. Because the time used to transfer the data dominates GPU calculation time.

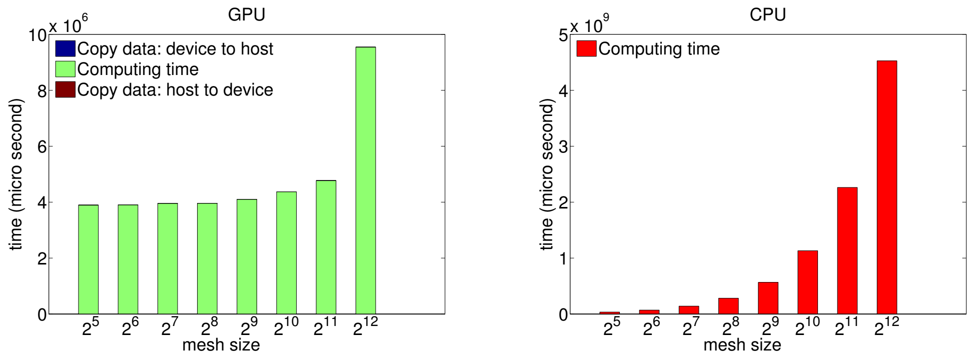


Figure 4.21: GPU (left) and CPU (right) time per step for multiple option pricing. The GPU computing time is relatively stable when the number of options is smaller or equal than 2048.

Chapter 5

Conclusions

A spring model based on the modifications of Delingette is used for fabric surface simulation which considers both tensile stiffness and angular stiffness. This model is convergent to elastic membrane model in continuum mechanics and is ideal for parachute canopy simulation. Navier-Stokes equation is solved by projection method to describe the incompressible viscous fluid field for personnel and cargo parachute. These two models are coupled together using impulse method and achieve realistic parachute deployment simulation. The porosity of canopy affects the stability, descent rate and drag force of the parachute. A concept ”penetration ratio” is introduced to transform the porosity simulation into boundary treatment. This simulation gives a conclusion that highly porous canopy will lead to more stable performance, but less drag force, which is consistent with observations. During the parachute simulation process, the solving spring model part is identified as most time-consuming and it is accelerated greatly by using heterogeneous computing techniques based on GPU.

Instead of using the geometric Brownian motion to describe the behavior of underlying asset, exponential Lévy process is used and a partial integro-differential equation can be derived for option pricing. Compared with the Black-Scholes equation, PIDE contains an extra integral term and needs special treatment. The integration domain is divided into five parts: the calculation of the first and the fifth parts are simplified through coupling boundary condition in advance; trapezoidal rule is adopted to calculate the second and fourth terms which enables reusing of grid information and pre-calculated value of Lévy measure; the center part is calculated by applying Taylor expansion and is pre-calculated to improve efficiency. This algorithm can be applied to PIDE derived from any distributions. We demonstrate the process with the model constructed from generalized hyperbolic distribution, due to the flexibility of its density function. This algorithm is proved to be numerically convergent and the accuracy is between first and second order. To further accelerate the application, two different heterogeneous algorithms are designed to accommodate different cases. For single option pricing, integral value at each grid point are calculated parallely with GPU; for multiple option pricing case, dramatic efficiency improvement is achieved by simultaneously assigning each thread the task of pricing one option.

A heterogeneous advection solver is developed based on WENO scheme and Runge- Kutta method. This algorithm intends to solve the Euler equation in gas dynamic simulation for compressible fluid cases such as space shuttle landing parachute. The WENO reconstruction part is accelerated using GPU techniques. The process of finding optimized block size is demonstrated which enables best parallelization effect under certain execution resource capacity. Uniform mesh based on finite difference scheme is used during reconstruction and this allows the usage of shared memory, rather than global memory to reduce the data fetch latency. Finally, each thread follows exactly same instruction which avoids potential latency due to threads divergence. This heterogeneous algorithm realize substantial acceleration and can be easily applied to advection form equations.

In sum, three heterogeneous algorithms are discussed to realize numerical simulations in scientific and engineering fields. These algorithms are accelerated through coupling GPU techniques appropriately and have wide applications in solving many different PDE extended systems.

Bibliography

[1] Cuda c programming guide. 2014.

[2] Norma Alias, Noriza Satam, Roziha Darwis, Norhafizah Hamzah, A Ghaffar, Zarith Safiza, Md Islam, et al. Some parallel numerical methods in solving partial differential equations. 2010.

[3] Ariel Almendral. Numerical valuation of american options under the cgmy process. *Exotic option pricing and advanced Lévy models*, pages 259–276, 2005.

[4] Ariel Almendral and Cornelis W Oosterlee. Numerical valuation of options with jumps in the underlying. *Applied Numerical Mathematics*, 53(1):1–18, 2005.

[5] Ariel Almendral and Cornelis W Oosterlee. Accurate evaluation of european and american options under the cgmy process. *SIAM Journal on Scientific Computing*, 29(1):93–117, 2007.

[6] Ole E Barndorff-Nielsen. Processes of normal inverse gaussian type. *Finance and stochastics*, 2(1):41–68, 1997.

[7] John B. Bell, Phillip Colella, and Harland M. Glaz. A second-order projection method for the incompressible navier-stokes equations. *Journal of Computational Physics*, 85:257–283, 1989.

[8] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The journal of political economy*, pages 637–654, 1973.

[9] W. Bo, B. Fix, J. Glimm, X. Li, X. Liu, R. Samulyak, and L. Wu. Frontier and applications to scientific and engineering problems. *Proceedings in Applied Mathematics and Mechanics*, 2007.

[10] W. Bo, B. Fix, J. Glimm, X. L. Li, X. T. Liu, R. Samulyak, and L. L. Wu. Frontier and applications to scientific and engineering problems.

Proceedings of International Congress of Industrial and Applied Mathematics, pages 1024507–1024508, 2008.

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

[12] Phelim P Boyle. Options: A monte carlo approach. *Journal of Financial Economics*, 4(3):323–338, 1977.

[13] Maya Briani, Roberto Natalini, and Giovanni Russo. Implicit–explicit numerical schemes for jump–diffusion processes. *Calcolo*, 44(1):33–57, 2007.

[14] Mark Broadie and Jerome Detemple. American option valuation: new bounds, approximations, and a comparison of existing methods. *Review of Financial Studies*, 9(4):1211–1250, 1996.

[15] Felix Browder et al. Partial differential equations in the 20th century. *Advances in Mathematics*, 135(1):76–144, 1998.

[16] David L Brown, Ricardo Cortez, and Michael L Minion. Accurate projection methods for the incompressible navier-stokes equations. *Journal of Computational Physics*, 168(2):464–499, 2001.

[17] Feng Chen and Jie Shen. A gpu parallelized spectral method for elliptic equations.

[18] Gui-Qiang G. Chen. Partial differential equations: Origins, developments and roles in the changing. Technical report, Oxford Mathematical Institute: The Secrets of Mathematics, 2014.

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

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

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

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

[23] Rama Cont and Ekaterina Voltchkova. A finite difference scheme for option pricing in jump diffusion and exponential lévy models. *SIAM Journal on Numerical Analysis*, 43(4):1596–1626, 2005.

[24] Rama Cont and Ekaterina Voltchkova. Integro-differential equations for option prices in exponential lévy models. *Finance and Stochastics*, 9(3):299–325, 2005.

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

[26] Duy Minh Dang, Christina Christara, and Kenneth R Jackson. A parallel implementation on gpus of adi finite difference methods for parabolic pdes with applications in finance. *Available at SSRN 1580057*, 2010.

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

[28] 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.

[29] 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.

[30] Daniel Egloff. High performance finite difference pde solvers on gpus. *QuantAlea GmbH, Zurich, Switzerland2010*, 2010.

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

[32] Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Soc., 2010.

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

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

[35] 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.

[36] 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.

[37] Mike Giles, Endre László, István Reguly, Jeremy Appleyard, and Julien Demouth. Gpu implementation of finite difference solvers. In *Proceedings of the 7th Workshop on High Performance Computational Finance*, pages 1–8. IEEE Press, 2014.

[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] Oscar Gonzalez and Andrew M. Stuart. *A First Course in Continuum Mechanics*. Cambridge University Press, 2008.

[40] Nail A Gumerov and Ramani Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227(18):8290–8313, 2008.

[41] Helmut G Heinrich and Eugene L Haak. Stability and drag of parachutes with varying effective porosity. Technical report, DTIC Document, 1971.

[42] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, volume 54. Springer Science & Business Media, 2007.

[43] Ali Hirsa and Dilip B Madan. Pricing american options under variance gamma. *Journal of Computational Finance*, 7(2):63–80, 2004.

[44] Roger W Hockney. A fast direct solution of poisson's equation using fourier analysis. *Journal of the ACM (JACM)*, 12(1):95–113, 1965.

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

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

[47] 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.

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

[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] Yongsam Kim and Charles S Peskin. 2-d parachute simulation by the immersed boundary method. *SIAM Journal on Scientific Computing*, 28(6):2294–2312, 2006.

[52] 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.

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

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

[55] Roger Knobel. *An Introduction to the Mathematical Theory of Waves*. American Mathematical Soc., 2000.

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

[57] Jens Kruger and Rudiger Westermann. *GPU Gems 2*. Pearson Addison Wesley Prof, 2005.

[58] C. C. Lin and L. A. Segel. *Mathematics Applied to Deterministic Problems in the Natural Sciences*. Society for Industrial and Applied Mathematics, 1988.

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

[60] Dilip B Madan, Peter P Carr, and Eric C Chang. The variance gamma process and option pricing. *European Finance Review*, 2(1):79–105, 1998.

[61] 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.

[62] Ana-Maria Matache, Tobias Von Petersdorff, and Christoph Schwab. Fast deterministic pricing of options on lévy driven assets. *ESAIM-Mathematical Modelling and Numerical Analysis*, 38(1):37–72, 2004.

[63] 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.

[64] Robert C McOwen. *Partial differential equations: methods and applications*. Pearson Education Inc., 2003.

[65] 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.

[66] K Parand, Saeed Zafarvahedian, and Sayyed A Hossayni. Gpu-acceleration of parallel unconditionally stable group explicit finite difference method. *arXiv preprint arXiv:1310.3422*, 2013.

[67] Everett H Phillips and Massimiliano Fatica. Implementing the himeno benchmark with cuda on gpu clusters. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–10. IEEE, 2010.

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

[69] 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.

[70] 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.

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

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

[73] EW Sachs and AK Strauss. Efficient solution of a partial integro-differential equation in finance. *Applied Numerical Mathematics*, 58(11):1687–1703, 2008.

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

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

[76] 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.

[77] WP Shepardson. Problems of parachute design and their relation to textiles. Technical report, DTIC Document, 1954.

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] Harold S Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM (JACM)*, 20(1):27–38, 1973.

[87] 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.

[88] Toru Takahashi and Tsuyoshi Hamada. Gpu-accelerated boundary element method for helmholtz'equation in three dimensions. *International journal for numerical methods in engineering*, 80(10):1295–1321, 2009.

[89] 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.

[90] 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.

[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] J. W. Thomas. *Numerical Partial Differential Equations*. Springer Science and Business Media, 1995.

[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 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.

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

[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] Jason Wang, Nicolas Aquelet, Benjamin Tutt, Ian Do, Hao Chen, and Mhamed Souli. Porous euler-lagrange coupling: Application to parachute dynamics. In *9th International LS-DYNA Users Conference*, 2006.

[100] Stephen Wolfram. *New Kind of Science: Notes from the Book*. Wolfram Media Inc., 2002.

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