学校代号 10532　　学　　号 B0902S0012

分 类 号 TP391　　密　　级 公开

湖南大學
HUNAN UNIVERSITY

博士学位论文

基于 GPU 的车身结构接触碰撞过程并行计算方法

学位申请人姓名 蔡勇

培 养 单 位 机械与运载工程学院

导师姓名及职称 李光耀　教授

学 科 专 业 车辆工程

研 究 方 向 汽车 CAE 及其并行计算方法

论文提交日期 2013 年 9 月 1 日

学校代号：10532

学　　号：B0902S0012

密　　级：公开

湖南大学博士学位论文

基于GPU的车身结构接触碰撞过程并行计算方法

学位申请人姓名：　蔡勇

导师姓名及职称：　李光耀　教授

培　养　单　位：　机械与运载工程学院

专　业　名　称：　车辆工程

论文提交日期：　2013 年 9 月 1 日

论文答辩日期：　2013 年 11 月 14 日

答辩委员会主席：　刘腾喜　教授

GPU-based Parallel Computing Method for Contact and Impact Problems of Automotive Body

by

CAI Yong

B.E.(Harbin Institute of Technology) 2008

A dissertation submitted in partial satisfaction of the

Requirements for the degree of

Doctor of Engineering

in

Automotive Engineering

in the

Graduate School

of

Hunan University

Supervisor

Professor LI Guangyao

November, 2013

摘 要

汽车车身结构接触碰撞过程有限元计算是汽车 CAE 的重要组成部分，主要涉及汽车碰撞和车身覆盖件成形等工程问题分析，在力学上涉及到材料非线性、几何非线性和接触界面的边界非线性三类非线性问题，经常面临着数值计算量庞大，计算效率低的问题，因而实际应用中对并行计算的需求十分强烈。目前常见的有限元并行计算方法多采用区域分解等粗粒度并行策略，在以 CPU 为计算核心的网络计算机集群上运行，计算效率与计算机节点数直接相关，使用流程复杂且需要昂贵的硬件支持，因此这种并行计算方法的性价比不高。

现代的图形处理器(GPU)是一种内部高度并行的众核处理器，浮点计算能力远高于同时期 CPU 的运算能力。可编程着色器的出现，使得 GPU 具有了通用处理器的特征，并开始应用于通用计算领域，为大数据处理和数值模拟研究带来了新思路和方法。最初的基于 GPU 的通用计算技术(GPGPU)采用 Cg 等高级着色语言编程，并已经应用于各类有限元计算，但是，由于这一时期的 GPGPU 技术只支持单精度计算，数据传输效率也不高，导致有限元 GPU 并行计算的精度低且效率提升有限，工程应用局限性大。统一计算架构(CUDA)的出现，带来了高效、直观的 GPU 并行程序开发工具，基于 CUDA 架构的 GPU 并行计算方法具有计算硬件成本低，计算程序开发简单等特点。

本文以工程应用需求为指导，采用 CUDA 架构研究高精度和高效率的显式有限元细粒度并行计算方法，以及全流程细粒度执行的并行接触算法，最终实现在普通个人计算机上进行汽车车身碰撞仿真和薄板冲压成形仿真两类大规模非线性有限元的快速并行计算。本文的主要工作和成果如下：

(1) 考虑到非线性显式有限元天然的可并行性以及 GPU 的轻量级线程执行模式，开发了具有自主知识产权的基于 GPU 的显式有限元计算平台(发明专利受理号：201210266435.1)。其主要特点在于：建立了线程与单元、线程与节点、线程与自由度三种层次的抽象映射方法，使显式有限元计算与 GPU 线程完美融合。同基于网格分区的粗粒度有限元并行策略相比，该细粒度并行策略没有任何前处理过程，在单块显卡也不存在边界数据处理问题，能够大幅度提升计算效率。因此，可以很方便的实现节点速度、位移计算等显式有限元绝大部分流程在 GPU 上的高效并行计算。

(2) 针对单元计算中节点应力组装在 GPU 平台上难以并行化的技术瓶颈，提出了预索引并行应力组装策略，实现了 BT 四边形单元和 EST 三角形单元两种壳单元在 GPU 上的细粒度并行。提出了 GPU 上基于并行缩减算法的时间步长等单值并行求解方法。实现了显式有限元算法在 GPU 上的全过程计算，减少了 GPU

与 CPU 间数据交换的同时，使程序的计算效率达到最佳化。通过对板壳非线性问题计算表明，该算法的 GPU 并行计算结果与原串行算法在 CPU 中计算的结果完全一致，与同时期同价格的 CPU 相比，计算效率有明显的提升。在 GTX580 显卡上采用 EST 单元进行 185 万个自由度的弹塑性大变形问题求解时，可以达到近 37 倍的计算加速比。

(3) 接触碰撞有限元分析中，接触算法需占用 70%以上的计算时间，为此，本文提出了包含并行级域接触搜寻算法、并行防御节点接触力计算方法和并行罚函数接触力计算方法在内的全流程 GPU 执行的细粒度并行接触算法。级域算法是一种适用于复杂自接触问题的高效搜寻算法，其同一级内接触块的计算独立性也符合 GPU 细粒度计算的要求。本文提出了线程与接触块一一映射策略、GPU 并行排序以及提升 GPU 线程计算粒度等技术手段，实现了测试对在 GPU 上的并行搜寻。在接触对搜寻阶段，本文提出了线程与测试对间的映射策略以实现同一级内接触对的并行搜寻，并采用计算后排序的策略进行上一级与下一级间的数据交换。在接触力计算阶段，本文采用线程与接触对间的映射策略给出了穿透量和接触力细粒度并行计算方法，并采用原子操作来实现接触力的离散。最后，基于自主开发的碰撞仿真软件DYSI3D开发了基于GPU的碰撞过程计算机仿真并行计算软件 CPS-GPU(软件著作权编号：2011SR001966)。采用该软件在 GTX580 显卡上进行 177 万个自由度的白车身碰撞计算时，可以取得 20 倍左右的计算加速比。

(4) 本文提出了完整的薄板冲压成形 GPU 并行计算方法。针对薄板冲压成形对材料流动模拟要求高的有限元计算特征，提出了包含复杂材料本构计算的单元 GPU 并行计算技术以及考虑摩擦的接触力 GPU 并行计算方法。本文提出了一体化接触搜寻算法在 GPU 上的计算策略：引入了计算机图形学中用于实时碰撞检测的广域搜寻方法来完成测试对搜寻，并在建立了相邻接触块信息的前提下，给出了接触后搜寻中接触对细粒度并行更新方法。在自主开发的薄板成形仿真软件 CADEMII 软件的基础上，开发了基于 GPU 的板料成形并行计算软件 CADEM-GPU(软件著作权编号：2010SR052426)，并加入异步数据输出模式以及基于 OpenGL 的实时显示技术，进一步提高了软件的计算效率和实用性。数值算例表明，该软件具有较高的计算精度和计算效率，在 GTX460 显卡上，对于数万网格数的仿真模型，可以取得 20 倍以上的加速比，有效缩短了仿真计算时间。

关键词：图形处理器；统一计算架构；并行有限元；接触/碰撞；板料成形

Abstract

Finite element (FE) simulation of contact and impact process is an important part of the automotive CAE technology. It is widely applied to engineering problems, such as car crash simulation and sheet metal forming simulation. This kind of simulation usually involves material nonlinearity, geometric nonlinearity and nonlinear boundary conditions. Due to these three kinds of nonlinearity, the FE analysis of contact and impact problems faced with enormous computations and low computing efficiency. Therefore, there is a very strong demand for parallel computing in practical applications. Nowadays, the most common parallel computing methods are based on the coarse-grained parallel domain decomposition strategy, and use CPU-based computer network as the computing hardware. In these traditional parallel computing methods, computation efficiency is directly related to the number of computing nodes. Furthermore, in practice, more complex programming and expensive hardware are required for more computing nodes. Therefore, they are not cost effective for both individual and business.

Modern graphics processor unit (GPU) has developed into a kind of multi-core processors with highly internal parallelism, and its float point processing ability is much higher than CPUs at the same period. In the meantime, the appearance of programmable shaders brings several general computing characteristics for GPU. Nowadays, general-purpose computing on GPU (GPGPU) becomes to a novel and effective methods for general large data processing and numerical simulations. The early GPGPU needed to use high-level shading languages to code, such as Cg. Several researchers have tried to use early GPGPU to improve computing efficiency, but these GPU-based FE codes cannot meet the demands of requirements in accuracy and efficiency. This is mainly due to the limited of double float support and the data transfer efficiency. Later, an efficient and intuitive GPGPU program development tools named compute unified device architecture (CUDA) is presented by NVIDIA. CUDA brings an efficient way for GPGPU with low computing cost and general programming language.

In this paper, a GPU-based parallel strategy for explicit FE computing with a full fine-grain parallel contact algorithm is presented to meet the demands of engineering applications. And, the high performance parallel computing of automotive body crash simulation and sheet forming simulation on normal personal computer with a

CUDA-capable device are realized. The main research content and result are as follows:

(1) A GPU-based parallel explicit FE computing platform with independent intellectual property rights based on the characteristics of explicit scheme and lightweight threads parallel computing model of GPU is presented (Patent Pending Number: 201210266435.1). The main advantage of this platform is constructed three kinds of one-to-one mapping relationship between CUDA thread and computing object, including thread-to-element, thread-to-node and thread-to-freedom. Compare to the coarse-grained parallel FE algorithm based on grid partition technology, the fine-grained parallel strategy can enhance calculation efficiency without any pre-treatment processes and boundary data processes. Therefore, the most parts of explicit FE calculation processes involving nodal speed computing and displacement computing can mapped to GPU computing to achieve high efficient.

(2) The nodal force assembling on fine-grained parallel platform has long been a difficult subject. This paper proposed a pre-index strategy to realized parallel assembling on GPU with few additional works. In the meantime, parallel strategies for two kinds of shell element including Belytschlko-Tsay (BT) shell element and Edged-based smoothed triangular (EST) shell element are presented based on the above parallel computing platform. Parallel reduction method is introduced to calculate all kinds of single variables, such as global time step. Finally, an entire parallelized explicit FE iterative process based on GPU is proposed, which can obtain an optimal computational efficiency by reduce the data transfers between CPU and GPU. The numerical examples for nonlinear shell structures show that this method can greatly improve the computational efficiency with the same computing results of serial computing on CPU. For example, about 37 times speedup obtained by GTX580 GPU compare to I7 CPU for an elastic-plastic large deformation problem with 18.5 million degrees of freedom.

(3) During a FE analysis of contact problem, the time consumption of contact algorithm usually occupies more than 70% of the total computation time. Therefore, an entire GPU-based parallel contact algorithm is proposed in this paper, including parallel hierarchy-territory contact-searching algorithm (HITA) and two kinds of parallel contact force calculation algorithms involve parallel penalty function method parallel defense node algorithm. HITA is an efficient contact-searching algorithm and especially suitable for complex problems contain self-contact phenomenon. Furthermore, the computing independence of contact segments searching in the same

hierarchy is suited for GPU parallel computing. Firstly, this paper proposed several technical means to realize the parallel search of test pair on GPU, including thread to segment mapping scheme, the GPU-based sort method and the technology of improve the size of thread granularity. Secondly, in contact pair searching phase, a mapping relationship between thread and test pair is presented to achieve the parallel searching in the same hierarchy. And, a store strategy based on sort is used to realize efficient data transfer between higher-level hierarchies and lower-level hierarchies. In the contact force calculation phase, fine-grained parallel strategy based on thread to contact pair mapping is present to parallel computing contact force, and atomic operation is used to contact force scatter. Based on the above mentioned algorithms, a GPU-based contact process simulation software named CPS-GPU (Software Registered Number: 2011SR001966) is developed based on the self-developed serial contact process simulation software DYSI3D. The numerical examples also demonstrate that this software can get highly accuracy and efficiency. For example, about 20 times speedup can obtain by using GTX580 graphics card to calculate a Body in White (BIW) crash model with 17 million degrees of freedom.

(4) This paper presents a complete GPU parallel computing method to accelerate the FE analysis of sheet metal forming process. According to the requirement of high computing accuracy for material flow in sheet forming simulation, a parallel computing method for shell element with complex material constitutive and friction-considering contact force computation are proposed. In the meantime, the way to parallel a simple contact algorithm integrated in the self-developed sheet metal forming simulation software CADEMII is studied. Firstly, the wide broad search method used in real-time collision detection is introduction to test pair searching during the pre-contact searching. Secondly, a parallel contact pair update method after pre-contact searching is proposed based on the information of adjacent contact segments. Finally, a GPU-based sheet metal forming parallel computing software named CADEM-GPU (Software Registered Number: 2010SR052426) is developed based on CADEMII. To extend the computing efficiency and practicability of this software, several usefully technologies such as data asynchronous transfer method and real-time display technology based on OpenGL are added. Numerical examples show that more than 20 times speedup can be obtained by using GTX460 graphics to calculate sheet metal FE model with tens of thousands of elements.

Key Words: Graphics Processing Unit; Compute Unified Device Architecture; Parallel Finite Element Method; Contact/impact; Sheet Forming

目　录

学位论文原创性声明和学位论文版权使用授权书 I
摘　要 II
Abstract IV
目　录 VII
第 1 章 绪论 1
1.1 引言 1
1.2 接触碰撞有限元及其并行计算方法发展概述 2
1.2.1 基本有限元方法及分析软件 2
1.2.2 基本有限元并行计算技术 4
1.2.3 接触碰撞有限元分析方法概述 5
1.2.4 车身结构中的接触碰撞问题 6
1.3 GPU 通用计算方法的研究现状 7
1.4 本文主要研究内容 9
第 2 章 基于 CUDA 架构的 GPU 并行计算技术 12
2.1 并行计算技术 12
2.1.1 并行计算的基本体系结构 12
2.1.2 并行算法与编程模型 13
2.2 图形处理器的硬件架构 14
2.2.1 图形处理器的发展及其可编程性原理 14
2.2.2 GPU 与 CPU 硬件比较 14
2.2.3 用于高性能通用计算的 Fermi 架构 15
2.3 CUDA 的编程和执行模型 17
2.3.1 CUDA 编程方法 17
2.3.2 CUDA 线程层次结构 18
2.3.3 CUDA 存储器模型 20
2.3.4 CUDA 程序执行模式 21
2.4 并行效率评价准则 22
2.5 GPU 计算性能的初步测试 22
2.5.1 测试平台和测试算例 22
2.5.2 测试结果分析 25
2.6 本章小结 25
第 3 章 基于 GPU 的板壳单元显式并行计算方法 27

3.1 板壳结构非线性有限元分析算法和程序.....27
3.1.1 中心差分格式的显式积分算法.....27
3.1.2 单元技术.....29
3.1.3 显式算法的串行执行流程图.....35
3.1.4 显式算法的并行性分析.....37
3.2 基于 CUDA 的显式有限元并行化策略.....38
3.2.1 计算对象与线程间的映射策略.....38
3.2.2 一维数据存储模式.....39
3.2.3 基于预索引策略的并行内力组装方法.....41
3.2.4 基于并行缩减策略的单值求解方法.....43
3.2.5 并行程序整体流程设计.....43
3.2.6 EST 壳元并行的特殊化处理.....44
3.3 数值算例及分析.....45
3.3.1 数值算例.....46
3.3.2 数值结果分析.....51
3.4 通用显式有限元的 GPU 并行计算平台.....52
3.5 本章小结.....53
第 4 章 基于 GPU 的车身结构碰撞过程并行计算方法.....54
4.1 接触碰撞界面的有限元模型.....54
4.1.1 接触体中接触界面模型.....54
4.1.2 接触界面的离散处理.....55
4.2 接触碰撞问题的有限元算法.....57
4.2.1 级域接触搜寻算法.....57
4.2.2 接触力计算方法.....59
4.3 接触碰撞问题的程序执行流程.....62
4.3.1 程序模块及计算时间分布.....62
4.3.2 串行程序中常用的加速策略.....63
4.4 接触碰撞问题有限元分析并行化策略.....63
4.4.1 单元计算部分的并行化.....63
4.4.2 级域接触搜寻算法的细粒度并行化策略.....64
4.4.3 接触力计算的并行策略.....69
4.5 数值算例及分析.....70
4.5.1 数值算例.....70
4.5.2 结果分析.....74
4.6 本章小结.....75

第 5 章 基于 GPU 的薄板冲压成形并行计算方法 76
5.1 薄板冲压成形问题的有限元仿真方法 76
5.1.1 主要物理现象及仿真方法 76
5.1.2 冲压仿真的模型建立和计算流程 80
5.2 薄板冲压成形关键算法的 GPU 计算 81
5.2.1 包含复杂材料模型的单元并行计算技术 81
5.2.2 简化接触搜寻算法的 GPU 实现 82
5.2.3 考虑摩擦的接触力与压边力并行计算方法 83
5.3 基于 GPU 的薄板冲压成形并行计算系统开发 84
5.3.1 异步数据输出方式 84
5.3.2 计算精度的保证 86
5.3.3 基于 OpenGL 的实时显示技术 87
5.3.4 并行仿真系统的整体计算流程 89
5.4 数值算例及分析 91
5.4.1 数值算例 91
5.4.2 结果分析 94
5.5 本章小结 95
结论与展望 97
参考文献 99
附录 A 攻读学位期间研究成果和发表学术论文情况 114
致 谢 116

第 1 章 绪论

1.1 引言

汽车工业在国民经济发展中起着重要的支柱作用，汽车工业的发展可以推动许多相关工业部门的发展，如：化工、材料、机械制造、电子等多个产业。我国的汽车工业从上世纪中期到现在发展迅速，成为了排名世界前列的汽车制造大国。可是，至今国内大部分的汽车企业都未能掌握到核心的设计和制造技术，仍处于仿造性制造或者合资自主制造阶段。因此，如何提高国内汽车企业的独立自主开发能力和较快的市场反应能力已成为制约我国汽车工业发展的瓶颈问题。计算机辅助工程(Computer Aided Engineering，CAE)已经成为现代汽车企业在日趋激烈的市场竞争中取胜的重要手段，CAE 技术为现代汽车行业的高速发展提供了强大的技术支持，可以有效的降低新产品开发的成本和周期[1-3]。

有限元法是当今科学计算与工程分析中应用最为广泛，理论最为成熟的数值计算方法，具有牢固的理论基础和广泛的应用能力[4]。有限元法的基本思想是将结构离散化成为有限个简单单元，将整体计算离散到单个单元上进行，然后根据变形协调条件形成整体解。因此，有限元单元的数量与结果的准确性直接相关，同时也与数学计算量直接相关。工程应用中，计算结果准确度和计算效率是一把双刃剑：一方面，简化和粗化的模型能减少计算量，提高计算效率，但会损失计算精度；另一方面，采用高精度的计算模型会带来巨大的计算量，降低计算效率，但是可以提高分析结果的准确度。现代汽车工业中，随着汽车结构件形状的复杂度日益增加以及工程人员对真实仿真结果的追求，有限元模型规模的不断增加已经成为汽车结构接触碰撞有限元分析中的普遍现象[5]。图 1.1 所示为乔治华盛顿大学国家碰撞研究中心(NCAC)近几年进行汽车碰撞有限元分析时，有限元计算模型的变化历程[6]。从最初采用 1800 单元的简单有限元模型对新车型的设计理念进行可行性分析，到采用数万个单元进行动态的非线性分析，到 21 世纪，已经发展成采用数十万甚至数百万个单元进行整车耐撞性分析。

有限元计算规模的增加，带来的是对计算效率的挑战。多年以来，CPU 性能的不断发展为有限元计算的提速做出了很大的贡献，不断刷新了有限元分析可采用的计算规模量级。但是 2004 年以后，单个 CPU 受到制造工艺和功耗散热等限制而逐渐放缓步伐，传统基于单核处理器的串行计算设备已经不能满足工程人员对计算速度的要求，从而促使了并行计算技术的产生[7]。并行计算的基本概念是指同时使用多个计算资源解决计算问题的过程，是解决工程计算领域内高要求计算模型的计算效率的有力途径[8]。基于图形处理器的通用计算(General Purpose

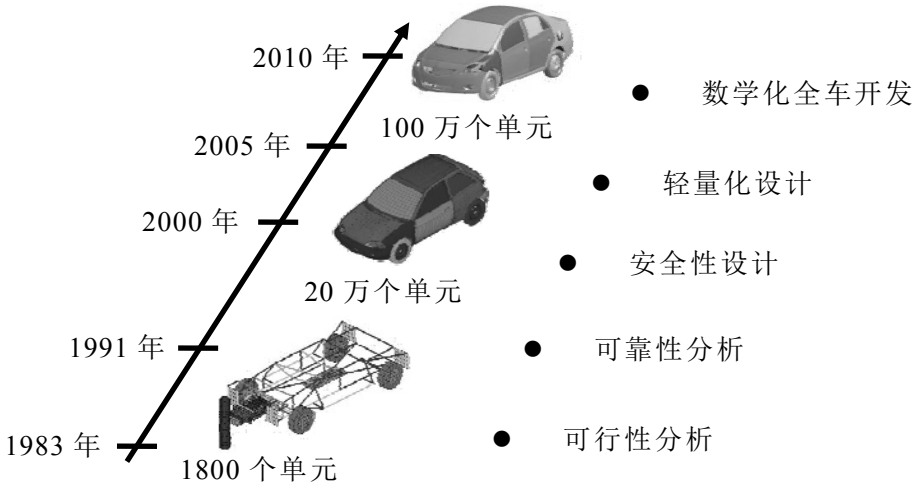


图 1.1 汽车 CAE 应用中有限元计算模型的发展历程

Computation on Graphics Processing Unit, GPGPU)技术是近几年随着可编程图形处理器(Graphics Processing Unit, GPU)的出现，以及 GPU 计算性能的不断提高而出现的一种新颖的并行计算方式。它摆脱了传统并行计算方法硬件成本高、使用和维护复杂等问题，因此很快引起了科研和工程人员的注意，其经过近 10 年的发展，已经形成了比较成熟的编程语言和编程架构。统一计算架构(Compute Unified Device Architecture, CUDA)[9]是 NVIDIA 公司于 2006 年提出的一种基于 NVIDIA 公司系列 GPU 的细粒度并行计算架构和高级编程语言，极大的简化了 GPU 并行计算程序的开发流程，降低了入门门槛，使得 GPU 通用计算方法从理论走向了实际应用。

1.2 接触碰撞有限元及其并行计算方法发展概述

1.2.1 基本有限元方法及分析软件

有限元的概念由 Colugh 在 1960 年首次提出[10]，在这之前， 20 世纪 40 年代 Hrennikoff[11]，McHenry[12]，以及 Courant[13]等学者首先提出了从整体离散到局部计算的基本思想。其中，Hrennikoff 采用的是类似于格子的网格离散求解区域，Courant 的方法则是将区域分解为有限个三角形的子区域，接近于现代有限元中三角形单元。有限元在工业中的应用最早出现在 20 世纪 50 年代中期，Turner 和 Colugh[14]等人利用 3 节点三角形对机翼表面进行模拟，这是有限元在弹性问题中应用的开始。

对弹塑性问题的求解首先出现于 Marcal 和 King[15]等人于 1967 年发表的用于二维应力问题的弹塑性有限元格式。接着，Yamada[16]等人推导了以小变形理论为基础的弹塑性矩阵的显式表达格式，推动了小变形弹塑性有限元的发展和应用。

随后，Hibbit[17]采用增量法建立了全 Largrangian 格式(T.L.格式)的有限元法，用于弹塑性大变形的有限元分析，随后，McMeeking[18]建立了更新的 Largrangian 格式(U.L.格式)的大变形弹塑性有限元表达式。至此，有限元可计算范围由弹性力学问题发展到包含复杂塑性大变形的弹塑性问题[19, 20]，使有限元工程实用性大大增强。现在，随着各类梁单元、杆单元、板单元、壳单元等先进计算单元的出现，有限元分析的真实性、稳定性和计算效率不断提高，并朝多物理耦合[21]、强非线性化[22] 与 CAD 的无缝集成[23]等方向发展。

有限元方法应用到实际工程中的载体是有限元分析软件，伴随着有限元理论的发展，有限元分析软件也得到了很大的发展。开始研究阶段，用于有限元计算的程序没有专门的程序名且功能也比较单一，由各个高校和实验室的研究者自己编程和使用[24]。具有通用功能和接口的第二代有限元程序起源于由 Clough 教授的博士生 Wilson 博士于开发有限元程序 SAP(Structure Analysis Program)[25]。随后，Wilson 的学生 Bathe 开发了第一个非线性结构分析程序 NONSAP[26]，可以进行冲击等瞬态问题求解，该程序后来逐步发展成为商业软件 ADINA。随着 SAP 和 NONSAP 这两个程序 70 年代传入中国，国内许多研究学者对其进行了研究和二次开发，对我国有限元软件技术的发展起到了很大的促进作用[27]。在商业软件的舞台上，早在 60 年代初国际上就已经开始投入大量的力量进行有限元分析程序的开发，近 15 年是有限元分析软件商品化的重要发展阶段，软件开发商在不断提升自身开发实力的同时，还通过并购或收购其它先进有限元方法来提升产品在功能和性能上的优势。工程应用中，汽车 CAE 的需求向来是有限元分析软件的风向标。目前，已经有各类有限元分析软件覆盖了汽车 CAE 中汽车结构强度分析、汽车碰撞分析、车身覆盖件成形分析以及疲劳寿命分析等几乎全部的领域，如表 1.1 所示[28]。

表 1.1 CAE 在汽车工业中的应用领域

序号	应用领域	主要 CAE 软件
1	结构、疲劳、振动分析	Nastran、Abaquas、Fatigue、Sysnoise
2	汽车结构碰撞分析	LS-Dyna、PAM-Crash、VPG
3	多刚体分析	ADAMS
4	热、流体分析	STAR-CD、FLOW-3D、Fluent
5	冲压成形分析	Dynaform、Autoform、PAM-Stamp
6	铸造分析	Procast、Magma
7	塑性成形	Moldflow

国内方面，商业 CAE 软件产业的发展相对落后，虽然在国家的大力的支持下，已经涌现出了一批具有自主知识产权的 CAE 软件，多个大学和研究机构开发有自

已发有限元分析系统，如由大连理工大学开发的工程与科学计算集成化软件平台 SiPESC[29]等。但是，这些 CAE 商业软件的认知度较差，难以与国际上的主流 CAE 软件抗衡。因此，加快推动我国自主知识产权 CAE 软件品牌建设应该成为国民经济建设和发展期中重要的组成部分。

1.2.2 基本有限元并行计算技术

实际工程中，CAE 技术的推广应用需要有限元理论和有限元分析软件的快速发展，但也难以摆脱计算力对它的制约。有限元并行计算技术是一种提高有限元计算效率的有力途径，并行计算通过充分利用现有计算机的计算力可以有效缩减对计算结果的等待时间。并行技术的发展历程中，硬件和软件是一种相互促进，相互影响的关系。并行计算机是并行计算的硬件基础，最早的并行计算机是指具有位并行计算能力的计算机，随后，先后出现了流水线单处理机系统、向量计算机和并行多处理器系统以及 MPP(Massively Parallel Processor)系统和工作站集群等，目前已发展成为可进行每秒数万万亿次运算的超级计算机。并行计算软件包括有并行算法和计算软件两个方面，Noor[30-32]在这方面做出了许多开拓性的工作，他首先提出了子结构方法[33]，并发表了第一篇有限元并行算法的文章。子结构法由 Farhat[34-36]发展成为区域分解法(DDT)。区域分解法是一种早期的粗粒度并行计算方法[37]，适合大部分的并行计算架构。由 Hughes[38]等人在 1983 年提出的 EBE 技术是一种有效用于隐式有限元基本方程求解的并行计算方法，它将整体计算分解到单元计算的策略极大的提升了隐式算法的可并行性[39, 40]。目前，有限元并行计算已经成为有限元领域中一个极为重要的组成部分，对有限元并行计算方法的研究日趋活跃，大到地球与地质科学，小到纳米结构分析和优化，都已经离不开并行计算方法的支持。

我国的并行计算技术一直追随着国际的发展趋势而发展，上世纪 80 年代初期，国防科学大学研制的 YH 系列向量并行计算机为我国第一批并行计算系统，并先后推出有 YH-1 和 YH-2 两代，计算性能峰值达到 400MFLOPS[41]。自主并行计算系统的出现使并行计算在国内各大高校和研究所中逐步普及，周树荃教授[42, 43]等曾基于 YH-1 系统的论述了有限元方程组形成和求解各个步骤的并行计算格式和并行程序设计技术[44, 45]，取得了不错的成果，并与国际上的主流研究方向保持同步。重庆大学的张汝清教授[46, 47]和胡宁教授等对大规模结构的线性和非线性并行分析也作出了重要的贡献。经过 40 多年发展，现在国内的并行计算硬件技术已经达到国际先进水平，由我国完全自主研制的天河一号超级计算机曾一度列于 TOP500 排行榜的首位。可惜的是，我国具有完全自主产权的并行计算软件相对缺乏，导致国内各类并行计算机的利用率不高。因此，提高自主并行计算软件的开发能力是我国未来并行计算技术发展中不可忽视的部分，而本文的研究也旨在

努力为此做出一份贡献。

1.2.3 接触碰撞有限元分析方法概述

接触碰撞过程仿真是汽车 CAE 的重要组成部分，广泛应用于汽车碰撞安全性能分析和车身覆盖件和结构件成形性能分析等工程应用。机械系统中的接触碰撞过程在力学中通常涉及到材料非线性、几何非线性和接触界面的边界非线性三种非线性问题，涉及到的数值计算问题主要有：材料本构计算模型、有限元单元计算模型、接触搜寻算法以及接触力计算等。Bathe[48, 49]首先分别采用 T.L.格式和 U.L.格式建立了用于求解这类两层非线性问题的动态和静态有限元方程，并利用虚功原理建立了基于隐式时间积分的有限元格式。相对于隐式积分方法，显式方法不需要迭代求解高次、非线性的有限元方程组，算法组成简单，计算过程对计算机内存的要求低，同时还具有很好的求解稳定性和收敛性，因此更适用于解决这类的问题[50]。在研究早期，Hughes[51, 52] 等人对接触碰撞等瞬态问题的显式解法研究以及 Belytschko[53]等人对动态非线性问题中显式壳单元的研究是对接触碰撞有限元领域做出了重要的贡献。

接触碰撞问题的有限元分析过程中由两个重要的部分组成：接触搜寻和接触力计算。其中，接触搜寻的目的是确定接触对象之间在动态时间响应历程中的几何不相容情况，找到接触对。Hughes 等[54]对接触算法的研究做出了许多有益的工作，他们提出采用点与点间几何关系来判断接触情况，但是，由于精度有限，现在已经很少用于机械系统接触判断。目前，接触算法大多采用点块接触模型来计算。常见的接触搜寻方法有主从面法、级域法和一体化算法。主从面法由 HALLQUIST 等[55]提出，并被应用于 LS-DYNA 软件中。在此算法中，接触面被指定为主面和从面，并假设从点不容许穿透主面，但是主点可以穿透从面。该方法计算简单快速，但是不适用于自接触问题，对主从面的网格密度也有一定的要求，因此应用范围有限。级域算法由 ZHONG 等[56, 57]基于级和域的概念提出，依据级的概念，搜寻从高一级向低一级逐步开展，当高一级的接触可能性被否定后，则可以免去下一级的接触检测，从而提高搜寻效率。级域法对多接触体的接触系统具有很高的计算效率，同时也已经成功应用于板料成形和汽车碰撞等仿真程序中，证明了其可靠和稳定性[58]。一体化算法由 ZHONG 等[59]提出，在此算法中，采用桶式排序[60]和定位码算法[61]进行全局搜索，采用级域法的思想进行局部搜索，从而进一步提高了接触搜寻的效率。另外，其它常用的算法还有小球算法[62]、光滑接触面算法[63]等。

找到接触对后，就要根据物体的运动规律计算接触对产生的接触力，常用的接触力计算方法有罚函数法和拉格朗日乘子法等。罚函数法由日本的 Yagawa[64]提出，后来得到广泛关注，由 Bathe[65]和 Hughes[54]等继续发展和完善。由于罚函

数法的算法结构简单，且算法和有限元运行方程完全解耦，已经成为 LS-DYNA 等商业软件中最常用的接触力计算方法。拉格朗日乘子法主要用于隐式算法中，需要增加额外的三个未知接触力变量，并且这些接触力变量与有限元运动耦合，因此，在实际工程中较少使用。后来，ZHONG[57]在 1988 年提出了防御节点法，是一种即能精确计算接触力，又能避免求解联立方程组的算法，并成功应用于汽车碰撞仿真和金属板料成形等复杂问题。

1.2.4 车身结构中的接触碰撞问题

在本文中，汽车车身结构中的接触碰撞问题主要包括两个部分：一部分是涉及汽车安全性设计、轻量化设计等先进设计领域的汽车碰撞有限元分析方法，另一部分是关系到车身成形性能的板料成形有限元分析方法。

汽车碰撞有限元分析包含整车和零部件的碰撞性能分析两个部分，最早开始于 Karnal[66]在采用实验-模拟的方法获得大型构件的单元特性分析的前提下，进行结构变形的预测。到了上世纪 70 年代初，逐步形成了基于质点力学理论的弹簧质点整车模型[67]。随后，随着 LS-DYNA 等商业软件的出现，汽车碰撞模拟得到了极大地发展。现阶段该领域主要聚焦于轻量化和新材料的研究[68-71]，碰撞生物力学的研究[72, 73]以及薄壁结构的抗撞性研究[74]等。国内对汽车碰撞安全性的研究如于上世纪八十年代。清华大学，吉林大学，湖南大学，同济大学等在这个方面的研究相对领先，先后建立了自己的碰撞实验设施，并开发了一部分有拥有自主知识产权的汽车碰撞仿真软件。汽车碰撞有限元仿真的计算效率有着较高的要求，因此，需要并行计算方法的支持[75]。对接触碰撞问题显式有限元并行算法的研究开始于 Belytschko 和 Plaskacz 等[76, 77]基于 SIMD 型并行机的显式算法并行化研究。之后，Brown[78]等提出了适用于汽车碰撞过程的一般并行计算策略。Lodsdale[79]等提出了并行计算平台上汽车碰撞有限元软件的开发方法。国内钟志华[80]，黄世霖[81]，寇哲君[82]等研究了可扩展冲击碰撞并行计算方法在汽车碰撞模拟中的应用。现在，伴随着并行计算技术和并行接触碰撞算法的发展，无论是在商业还是学术领域，都涌现出一批采用并行计算技术进行各类碰撞仿真的论文和成果[83, 84]。

用于分析汽车车身成形性能的板料成形的有限元模拟技术是接触碰撞算法另一个主要应用领域。板料成形过程在本质是上能过凹模、凸模、压边圈和板料四个运动体通过接触碰撞作用使板料发生塑性变形的过程。与汽车碰撞过程过程相比，由于模具的运动相对固定，因此接触关系比较简单。但是由于材料间有复杂的塑性变形和摩擦产生，因此对板料成形过程仿真的研究主要集中在材料本构以及壳单元理论等方面。早期，Wang 和 Budiansky[85]采用流动坐标和有限变形理论提出适用于一般板料成形问题的薄膜壳模型。为了提高仿真精度，Nakamachi[86]等采用 Kirchhoff 薄壳模型进行成形性能的分析。随后，板料成形分析进行了蓬勃

发展时期，DYNAFORM，AUTOFORM 等一批优秀的仿真软件相继推出。而早在 1980 年，Tang[87]就应用小变形有限元程序进行车身零件的成形性分析。在国内，上海交通大学林忠钦[88]，湖南大学李光耀[89, 90]等专家学者的工作对国内板料成形数值模型计算的发展起了十分重要的作用。并行计算方法同样可以有效的提高板料成形的计算效率，并且已经有大量基于传统并行计算平台的科研和商业成果涌现[91-94]。

1.3 GPU 通用计算方法的研究现状

传统的并行计算方法，在技术层面上主要采用分布式计算、并行机或多线程等并行处理技术，使用的计算硬件主要是以 CPU 为计算核心的并行计算平台。基于传统并行计算平台的并行有限元分析程序主要存在以下的不足：第一，当有限元模型达到一定规模时，庞大的单元、节点数量会导致分布式计算机之间的通讯消耗提升，使加速比难以提升。第二，传统并行计算中，计算加速比与计算节点数成正比，而计算节点数的增加，会使计算机的硬件成本、使用和维护成本均增加，因而性价比较低。第三，基于计算集群的有限元分析软件，要求软件使用者对计算硬件有较高的熟知度，不能做到完全的透明操作，所以很难在日常工作和科研中普及[95, 96]。因此，在倡导高性能绿色计算的现代社会，很有必要寻找一种具有较高功耗比的计算方法。基于 GPU 的通用计算方法就是符合这一条件的新型并行计算途径。

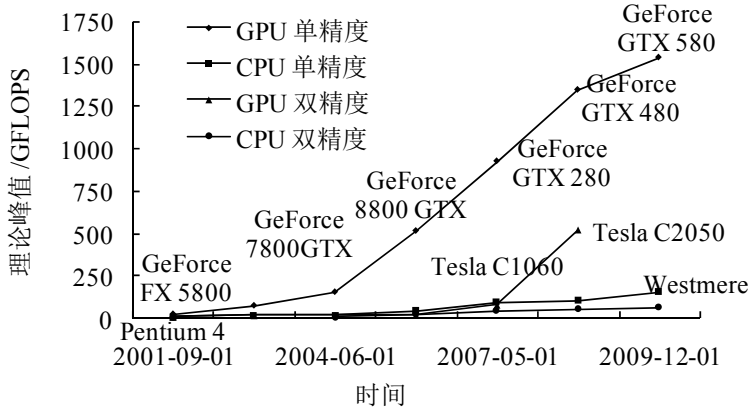


图 1.2 NVIDIA GPU 和 Intel CPU 发展曲线

为满足应用程序对高复杂度、高仿真度图像的渲染要求，GPU 逐渐发展成为一种高度并行化、多线程执行且具有强大浮点数处理计算能力的众核处理器。如图 1.2 中近十年 NVIDIA 公司的 GPU 和 INTEL 公司的 CPU 两种处理器的浮点计算能力发展曲线[97]所示，现代 GPU 的浮点计算能力已经达到了同时期 CPU 的 10 倍以上，外部存储器带宽达到 5 倍以上，并且随着 GPU 的发展，这种计算性能的

差距还在不断扩大。GPU 性能的高速发展是其用于通用计算的主要推动力，而 GPU 通用计算的发展使桌面级的高性能并行计算成为可能。

对 GPU 通用计算的研究是从 2003 年开始的，并首先提出了 GPGPU 的概念。在这之前，NVIDIA 公司在 2001 年发布的 GeForce 3 系列产品中第一次实现了着色器的可编程性，使 GPU 从由若干固定功能单元组成的专用处理器进化为以通用计算单元为主，固定功能单元为辅的全新通用处理器，奠定了 GPGPU 的发展基础[98, 99]。Krüger[100]等人于 2003 年将基于 GPU 的线性代数操作应用于数值分析中，是 GPU 通用计算发展的里程碑。这一时期内 GPGPU 的实现需要使用图形学 API 将通用计算映射成为图形处理过程中的顶点计算[101]，编程十分复杂。后来随着可编程着色器的出现而发布的 Cg（c for graphic）等着色语言编程简化了编程的难度，如 Hillesland 等[102]采用 GPU 进行图像处理过程线性代数操作的加速，Li 等[103]采用 GPU 对基于 Lattice Boltzmann(LB)方法的流体模拟进行加速。国内吴恩华等[104]于 2004 年对基于 GPU 的通用计算进行了探索和讨论，并对其可行性进行了肯定。不过，这些基于“伪装”思路的 GPGPU 实现方法要求程序员对图像处理和 GPU 结构有较深入的理解，且还需要将数据映射到纹理中，使得编程过程对数据的处理变得非常的复杂，对于特定的问题甚至是不可实现的。同时，编程人员还要受到着色指令数和着色器输出寄存器数目[105]等一些硬件上的限制。因此，在这一阶段，GPGPU 技术的应用范围有限，主要还是用于图形处理相关的通用计算领域。

2006 年 NVIDIA 公司发布了更适合于通用计算 G80 架构，与此同时，NVIDIA 公司发布了用于 GPU 通用计算的统一计算架构(CUDA)。CUDA 既是一种适用于 GPU 执行的并行计算架构，又是一种编程语言，程序员可以直接调用 CUDA API 对 GPU 进行控制，简化了通用计算映射到 GPU 上执行的过程，缩短了计算程序的开发难度和周期。CUDA 的推出极大的扩展了 GPU 通用计算的应用范围，现在在 Google 学术搜索引擎中以“CUDA parallel”为关键字，可以得到超过 2 万条搜索记录，其应用领域包括有：计算结构力学[106, 107]，生物信息学与生命科学[108, 109]，医学成像[110, 111]，天气与太空[112, 113]，数据挖掘与分析[114-116]，成像与计算机视觉[117, 118]，计算金融[119, 120]，计算流体力学[121, 122]，电磁学与电动力学[123, 124]，分子动力学[125, 126]等。在国内，也有越来越多的研究学者加入到了 GPU 通用计算的研究行列，研究涉及到并行数值算法[127]、并行性能分析[128]、并行应用[129]等领域，对 GPU 通用计算技术起到了很大的推动作用。其中，中国科学院和清华大学由于其在 GPU 高性能计算方面的突出贡献，于 2009 年被 NVIDIA 公司授予 CUDA 卓越中心(CCOE)的称号[130]。众多商业 CAE 软件也意识到了 GPU 的通用计算能力，Abaquas/Standard 从 V6.12 版本开始加入对 GPU 的支持，用于加速稀疏方程的直接求解，可以获得 1.5-2.5 倍的加速比[131]。Ansys 从 Release 14.0 版本

也开始采用 GPU 用于加速稀疏方程的直接和迭代求解，可以获得 2-3 倍的加速比[132]。其它，如 LS-DYNA/Implicit 在 CentOS Linux 下的测试版，MSC Nastran 的 Version 2013 版本以及 Marc 的 Version 2012 版本等也都陆续采用 GPU 提升其结构力学方程的求解效率[133]。

经过近 10 年的发展，GPU 通用计算领域已经形成了比较成熟的编程语言和编程架构。除 NVIDIA 公司的 CUDA 外，AMD 公司在 2006 年也发布了用于 GPU 通用计算的 CTM(Close to Metal)语言[134]，允许程序员随意访问 AMD 公司系列 GPU 上大量的并行计算元件，拉开了 AMD 开展 GPU 通计算研究的序幕。随后，AMD 又推出了 ATI Stream SDK[135]，并在 ATI Stream SDK 中新增了 Brook+[105]，Brook+是一种与 CUDA 类似的高级开发语言，它的出现形成了 AMD 公司的第二代 GPU 通用计算解决方案，并出现了一些相关应用成果。但是，相对而言，Brook+使用没有 CUDA 灵活，且限制较多，后期修护也没有 NVIDIA 公司积极。

NVIDIA 公司的 CUDA 和 AMD 公司的 Brook+均为私有规范，只支持自家 GPU，这显然不利于 GPU 通用计算技术的发展。直只到 2008 年，由 Apple 公司牵头，联合 AMD、IBM、Intel 和 NVIDIA 等计算机业界的主要研发者发布了名为开放计算语言(Open Computing Language，OpenCL)[136]的异构平台计算框架。OpenCL 的编程规则和 CUDA 类似，且具有硬件和软件上的平台无关性，支持由多核心 GPU、GPU、Cell 类型架构以及数学信号处理器(DSP)等多种并行处理器组成的各类异构计算平台。

虽然目前 GPU 通用计算平台已经出现了多种计算模型，但是，本文坚持采用 CUDA 作为算法和程序的开发平台。一方面，是由于 NVIDIA 公司系列 GPU 的通用计算能力最强，NVIDIA 公司也坚持将自己新产品朝通用计算靠近；另一方面，CUDA 作为 GPU 通用计算业界的首个公开架构，架构发展成熟、应用最广泛，具有极大的业界影响力。最后，CUDA 本身提供了多个并行计算库可以解决线性代数计算、稀疏矩阵计算和随机数生成等常用数值计算问题，国际上还有许多基于 CUDA 的开源项目，可以为并行算法的研究和并行程序的开发提供很多便利。Karimi[137]和 Du[138]等发表的文献也表明在同一个显卡上采用 CUDA 编程可以获得比 OpenCL 更高的计算性能。需要注意的是，目前最新版本的 CUDA 已经包含相关编译器，支持将 CUDA 格式的并行计算代码直接编译成 OpencCL 格式的程序，这为以后将本文的研究成果扩展其它计算平台上提供了有力的保证。

1.4 本文主要研究内容

本文在国家重点基础研究发展(973 计划)课题：产品功能和性能高效仿真优化理论与方法研究(项目编号：2010CB328005)；以及国家自然科学基金重点项目：面向重大工程需求的高效计算方法(项目编号：61236014)等项目的资助下，针对

汽车工程中接触碰撞过程有限元分析的计算效率问题，研究了基于 GPU 的汽车车身接触碰撞过程的并行计算方法，并应用于汽车碰撞和薄板冲压成形等实际工程问题。本文的主要研究内容：

(1) 研究了 GPU 的基本硬件架构和 CUDA 架构下 GPU 并行程序的开发流程。首先，本文介绍了通用图形处理器的可编性原理，通过对比 GPU 和 CPU 的硬件架构，说明了 GPU 具有极强并行浮点计算能力的原因以及 GPU 通用计算的优劣性，并着重研究了 Fermi 架构的硬件组成和性能指标。其次，对 CUDA 架构下的编程方法，线程层次结构、存储器模型和程序执行模式进行了探讨，依此引出了 GPU 通用计算的基本流程。同时，对本文采用的并行程序效率评估方法进行了简要介绍。在对结构分析基础上，本文采用三种类型的算例对 GPU 进行浮点数计算和实际科学问题的求解性能进行了测试。

(2) 提出了显式有限元全流程的 GPU 并行计算方法，并应用于板壳结构的非线性有限元分析。采用计算无关性的概念对显式有限元算法的可并行性进行了分析，证明了显式算法具有很好的天然并行性。研究了 GPU 上显式有限元计算过程中的数据表示方法，提高了并行计算过程中内存读取效率。通过建立线程与单元、线程与节点、线程与自由度三种层次的一一映射方法，使算法与 CUDA 的线程计算模式完全融合，实现了算法中大部分流程的细粒度并行。研究了 BT 四边形单元[53]和新型边光滑三角形壳单元 EST[139, 140]两种单元技术的 GPU 并行计算方法。针对单元计算中节点应力组装难以并行的问题，提出了预索引的策略，在 GPU 上实现了高效的并行组装过程。提出了时间步长计算等计算单值在 GPU 上的并行求解方法，实现了 GPU 上的全过程计算，减少了 GPU 与 CPU 间数据交换的同时，使程序的计算效率达到最佳化。

(3) 接触碰撞有限元分析中，接触算法需占用 70%以上的计算时间，为此，本文提出了包含并行级域接触搜寻算法、并行防御节点接触力计算方法和并行罚函数接触力计算方法在内的全流程 GPU 执行的细粒度并行接触算法。级域算法是一种适用于复杂自接触问题的高效搜寻算法，其同一级内接触块的计算独立性也符合 GPU 细粒度计算的要求。在测试对搜寻阶段，提出了线程与接触块一一对应的细粒度执行方法，实现了级域算法中同一级内的并行搜寻方法。在接触对搜寻阶段，本文采用线程与测试对一一映射的方式实现了同一级内的接触对搜寻，并采用计算后排序的方式实现上一级与下一级间的数据交换。在接触力计算阶段，本文提出了基于 GPU 的并行罚函数法和并行防御节点法两种接触力计算方法。在罚函数法中，提出了线程与接触对一一对应的方式同时计算每个接触对中接触点和接触面间发生的穿透，并计算接触力，实现了接触力计算的流程并行计算。在防御节点法中，提出在每级中分别采用线程与接触对一一对应的细粒度并行计算方法，同时计算每个接触对中接触点和防御节点间的接触力，并采用原子操作的

方式实现接触力对接触块中节点的离散过程。结合以上理论，本文基于 DYSI3D 软件开发了基于 GPU 的碰撞过程计算机仿真并行计算软件 CPS-GPU(软件著作权编号：2011SR001966)。

(4) 本文将 GPU 并行计算引入到薄板冲压成形有限元计算中，提出了完整的薄板冲压成形 GPU 并行计算方法。针对薄板冲压成形对材料模型要求较高的问题，提出了适用于 GPU 并行计算的包含复杂材料本构计算的单元并行计算技术。自主开发的板料成形软件 CADEMII 采用简化的一体化接触搜寻算法极大的降低冲压成形仿真中接触算法的计算时间，为此，本文研究了该简化接触算法在 GPU 上的实现方法：在测试对搜寻阶段，引入了计算机图形学中用于实时碰撞检测的广域搜寻方法，具有较高的计算效率。并在建立了相邻接触块信息的前提下，给出了接触后搜寻中并行接触对更新方法。由于摩擦力对板料成形性能有直接影响，为此，本文提出了包含摩擦力计算的接触力 GPU 并行计算方法。在 CADEMII 软件的基础上，本文开发了适用于 GPU 并行计算的金属薄板冲压成形仿真分析系统 CADEM-GPU(软件著作权编号：2010SR052426)，并加入异步数据输出模式以及基于 OpenGL 的实时显示技术，提高了软件的实用性。

第 2 章 基于 CUDA 架构的 GPU 并行计算技术

基于 CUDA 的 GPU 通用计算技术是一种硬件成本低、软件实现容易且计算效率高的绿色并行计算方法，也是本文研究的基础。在建立并行算法和系统之前，首先需要对基于 CUDA 的 GPU 通用计算技术所采用的软硬件平台以及编程模型进行探索和分析。同时，为了验证 GPU 是否适合于有限元等高精度运算的需求，本章采用基本的浮点数操作和一个双精度 N 体问题对 GPU 进行浮点数计算的性能进行了测试。

2.1 并行计算技术

2.1.1 并行计算的基本体系结构

计算机体系结构常采用 Flynn[141]分类法进行分类，Flynn 根据指令流和数据流的多倍性概念将计算机系统结构分成了四大类：单指令单数据流(Single Instruction Single Data, SISD)，多指令多单数据流(Multiple Instruction Single Data, MISD)，单指令多数据流(Single Instruction Multiple Data, SIMD)，多指令多数据流(Multiple Instruction Multiple Data, MIMD)[142]。其中，指令流是指机器执行指令序列，数据流是指令流调用的数据序列，多倍性是指系统结构的瓶颈部件上所允许同时执行的指令和数据的最大个数。四大类中，SISD 就是指传统的串行计算机，它不支持任何形式的并行计算，所有指令和数据都顺序处理。MISD 是一种不真实存在的结构，在此不做讨论。SIMD 是一类非常有效且较易实现的并行计算架构，目前 INTEL 和 AMD 的商业处理器被已经集成了数个用于实现 SIMD 计算的扩展指令集，如：MMXTM、SSE、SSE2 和 SSE3 等[143]。MIMD 则将指现在最常用到的多核处理器架构，它将多个处理核心集成到单个芯片中，每个处理核心都是一个独立处理器，并拥有独立或者共享的缓存空间[144]。SIMD 和 MIMD 具有强大的硬件支持，是目前最常用的并行计算架构，也是国际上传统并行计算领域的热点研究对象[145-148]。

对于高性能大规模并行计算所采用的大型并行机系统，其体系结构按照存储体系结构和计算节点的分布方式进行分类有：共享存储(Shared Memory, SM)、分布存储(Distribution Memory, DM)以及分布式共享存储(Distributed Shared Memory, DSM)架构[149]等三种体系结构。这三种体系结构同属于 MIMD 架构，如图 2.1 所示，三种架构的组成部分一样，只是连接的形式不同。多个具有独立存储空间的单核处理器通过网络连接而构成 DM 架构；多个处理器通过连接网络共享一块存储器空间则构成 SM 架构。如果将 DM 架构上计算节点扩展成为微型 SM 架构，

就形成了 DSM 架构。DM 和 DSM 架构是目前最常见的并行机组成方式，但是，这两种架构并不适应于计算单元粒度小且需要频繁通信的并行算法。一方面由于现代网络通信技术的限制，通信的时间成本非常高，另一方面，为提高计算力而盲目增加计算节点，会导致系统维护和使用成本过高。在 GPU 中，所有计算单元均可以通过总线访问同一个内存空间，不需要通过网络进行数据交换，因为内存延迟较低。但是，由于 GPU 中每一个处理核心拥有的硬件资源很少[150]，所以更适于细粒度的并行算法[151]。

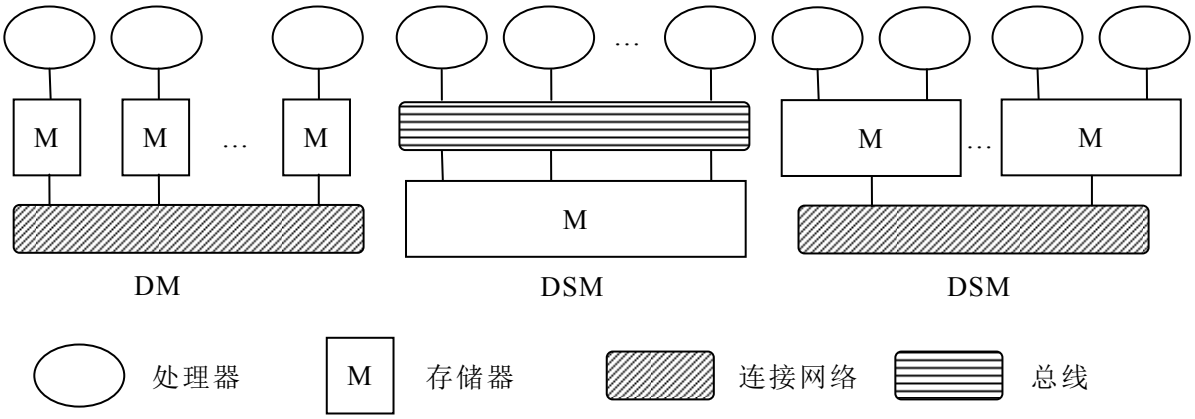


图 2.1 并行计算机体系示意图

2.1.2 并行算法与编程模型

并行算法是指用并行计算求解问题的方式，并行算法的实现依靠编程模型。针对不同硬件架构，需要采用不同的编程模型来实现并行算法，并行编程模型有消息传递模型、数据并行模型和共享变量模型三种。

消息传递模型主要用于在 MIMD 架构上开发粗粒度并行算法[152, 153]，一般需要通过网络设备执行各计算单元之间的消息传递、协调同步等操作。目前广泛使用的消息传递模型有两个：PVM(Parallel Virtual Machine)[154]和 MPI(Message Passing Interface)[155]。并行计算时，PVM 和 MPI 会分划出许多进程，每个一个计算节点负责一个进程的执行，并且可以同时在不同的数据空间中执行不同代码。但是，消息传递模型中的大部分工作都要编程者手动完成，如数据划分和进程间的同步，因此，编程比较麻烦，需要特别注意数据依赖、死锁等问题[156]。

数据并行模型和共享变量模型主要用于在 SIMD 架构和 MIMD 架构上开发细粒度的并行算法[157, 158]。在多核处理器中，这两种并行模型可以采用 SEE 指令集实现并行程序的开发。同时，OpenMP[159]也是这两种模型的一个主要程序开发标准，OpenMP 由 OpenMP Architecture Review Board 于 1997 年推出，目前已支持包括 C 和 Fortran 在内多种高级编程语言。相对于 PVM 和 MPI 等接口，OpenMP 的编程比较简单，通常情况下，它只是一个编译指导语言，告诉处理器接下来的

工作要由多线程同时进行，而线程具体如何执行将由处理器自己控制。事实上，这种多线程的并行执行方式也是 GPU 并向计算所采用的执行模式。

2.2 图形处理器的硬件架构

2.2.1 图形处理器的发展及其可编程性原理

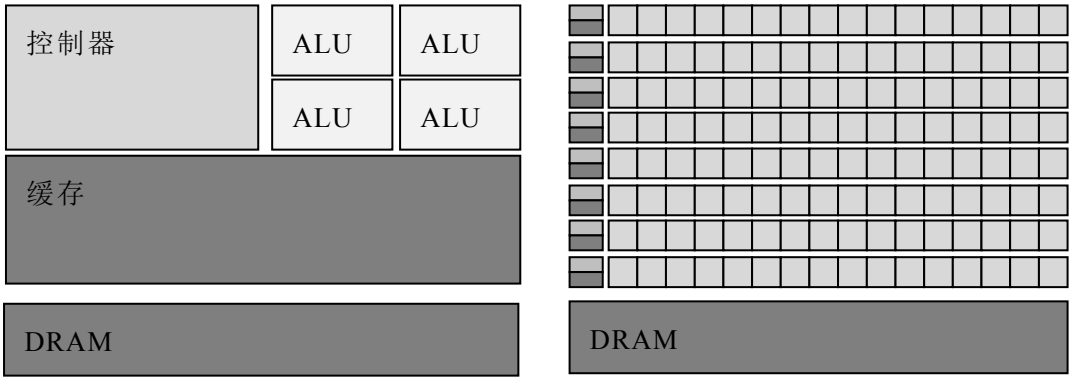
最早的图形处理器是只用于单色和彩色点陈文字输出的一类显示硬件。随着计算机图形学的提出，20 世纪 80 年代逐渐出现了专门进行图形计算的处理芯片以及承载其工作的显卡，并用于军用运输工具仿真模拟器的视景生成系统等各类军事领域研究。从 90 年代开始，伴随着应用程序对 2D 和 3D 图像处理质量的要求提高，出现了以 NVIDIA TNT2，ATI Rage 和 3dfx Voodoo3 为代表的平民化的图形处理芯片，即常说的图形加速器。NVIDIA 公司在 1999 年发布 GeForce256 系列图形处理芯片时，里程碑式的率先将硬件加速的多边形转换与光源处理(Transforming and Lighting，T&L)整合到显示芯片中，并第一次提出了 GPU 的概念：GPU 是拥有完整的转换、光照、三角形设置和渲染引擎等四种处理引擎的处理器。GPU 可以比 CPU 更快速的实现高质量的 3D 几何变换和光照计算，降低显卡对 CPU 的依赖。

同样在对复杂图形处理能力的追求的下，2001 年到 2006 年间，图形处理硬件中出现了着色器(shader)的概念，实现了对像素和顶点的可编程，并且使 GPU 开始向 SIMD 处理器方向发展，具有了部分流式处理器特征。随后，也许是 NVIDIA 公司对显卡通用计算能力的刻意追求，2006 年 NVIDIA 发布的 G80 架构采用了统一着色模型(Unified Shader Model)，将传统 GPU 架构中顶点着色器和片源着色器更新设计为功能更全统一逻辑运算单元(ALU)，即现在 NVIDIA 所称的流处理器(Stream Processors，SP)。统一着色模型最大的好处在于每一个着色器都可以完成原本只能由某种特定着色器完成的工作，提高了着色器的利用率。此时，GPU 发展成为了一种 MIMD 架构的通用众核处理器。

2.2.2 GPU 与 CPU 硬件比较

通常情况下，通用处理器内部由控制单元(control)、逻辑运算单元(ALU)和存储单元(DRAM)等 3 个部分组成，在晶体管数目基本一致的前提下，三者的数量配比将直接影响到处理器的性能。由于设计目标的不同，GPU 和 CPU 两者在硬件架构上有着很大的不同，这种不同也就体现在上述三个部分的数量配比上。如图 2.2 中(a)所示，CPU 将更多的晶体管用于逻辑控制器和高速缓存，只有少量的用作逻辑运算单元。这种设计使 CPU 拥有强大逻辑控制能力，可以适应于各类复杂运算和各种系统突发事情，如密集控制、复杂的流程控制和不规则内存访问等。相比之下，GPU 的设计与 CPU 完全相反。如图 2.2 中(b)所示，GPU 将大部分的

晶体管设计成为逻辑运算单元，控制单元和存储单元较少。这样的数量配比使得GPU 拥有了强大的浮点数并行计算能力，适合于具有较高算术密度，且逻辑分支简单的大规模并行负载。



(a) CPU　　(b) GPU

图 2.2 CPU 和 GPU 的晶体管对比

如上文所述，GPU 采用 MIMD 的并行计算模式，适用于进行数据并行模型下多线程轻量并行计算，依靠计算核心的数目优越可以保证在同样时间周期内比CPU 处理更多的线程。但是，尽管 GPU 的核心数要远大于 CPU，GPU 计算能力却难以按核心数的倍数增长，这是由于 GPU 中核心只是一个简单的标量处理器，缺少和 CPU 核心一样多级缓存系统和分支预测功能，主频也相对较低。为了尽量降低单个处理核心的计算力对整体计算能力以及更合理的管理并行线程，从 G80 架构开始，NVIDIA 就开始在自家的 GPU 中引入超级多线程(Gigathread)技术，使得成千个独立而同步的进程可以同时在 GPU 的相关着色器中运行，并通过动态检测每一个流水线的工作状态来保证着色器的负载平衡。同时，Gigathread 技术还可以有效地隐藏显存的存取动作紊乱所引用的延迟，其原理是：当发现有着色器因为等待纹理数据而空闲时，控制器会马上将这个着色器的执行线程冻结并放到临时寄存器中，然后立即提交新的线程给这个着色器，保证着色器在高压下的满负荷运行。

2.2.3 用于高性能通用计算的 Fermi 架构

如上文在描述图形处理器的发展历程中所述，2006 年以后，NVIDIA 公司一直坚持统一着色架构的理念，不断的巩固其在 GPU 通用计算领域的地位。G80 架构是 NVIDIA 公司对统一图形与计算处理器应有面貌的最初愿景。随后的发布的 GT200 则扩展了 G80 的性能与功能。最近发布的 Fermi[160]架构的系列显卡则可以称为是 NVIDIA 专为 GPU 通用计算而发布的产品，如表 2.1 所示，Fermi 架构在双精度计算方面和缓存设置等方面进行了大量的优化和改进，主要表现在如

下三个方面：

(1) 流处理器的数量增加，计算能力增强。流处理器的数量首次达到 512 个，并且引入 CUDA 核心的名字，表示其在通用计算能力上的增强。事实上，Fermi 架构中的每个 CUDA 核心都符合 IEEE 754-2008 浮点计算标准并支持完整的 32 位整数算法。更为重要的是，双精度浮点数的计算性能也大大提升，峰值执行率可达到单精度浮点的 1/2，有力的提高了 GPU 在有限元分析中的计算精度和计算效率。

(2) 建立了全新的缓存构架。Fermi 架构第一次在 GPU 引入硬件意义上的缓存，缓存可以减少线程对全局显存读写时的延迟，特别是对于非对齐内存访问情况。基于 Fermi 架构编写 CUDA 程序时，可以弱化对对齐内存访问的要求，从而使编程更加灵活和方便。详细的来说，Fermi 架构的 GPU 同时集成了 6 个内存控制器，以便获得高带宽和低延迟。缓存方面，一个统一的 768KB 二级缓存架构负责整个芯片的线程加载、存储和纹理操作。每组 SM 里四个纹理单元，共享使用 12KB 一级纹理缓存，并和整个芯片共享 768KB 二级缓存。

(3) 显存空间支持 64 位寻址，并加入 ECC 校验功能。Fermi 架构对 64 位寻址的要求，意味着可以在单块显卡上进行更大的规模的计算，而 ECC 校验功能可以极大的减少由于显存的读取错误而引起的计算错误。

表 2.1 NVIDIA 公司近三代 GPU 架构对比

性能指标		G80	GT200	Fermi
整体性能指标	晶体管	6.8 亿	14 亿	30 亿
	流处理器	128 个	240 个	512 个
	双精度计算力	不支持	支持	支持
	ECC 内存支持	不支持	不支持	支持
	kernel 并发执行	不支持	不支持	最多 16 个
	并行线程数	12288 个	30720 个	24576 个
	存储位宽	32 位	32 位	64 位
流处理器性能指标	warp 策略	1	1	2
	专用函数单元	2	2	4
	共享显存	16 KB	16 KB	48KB or 16KB
	L1 缓存	无	无	48KB or 16KB
	L2 缓存	无	无	768KB

CUDA 已经可以使程序员在完全脱离 GPU 架构的情况编写 GPU 通用计算程序，但是，对 GPU 架构的了解，却可以为 CUDA 程序优化提供便利，CUDA 程序优化是 CUDA 程序编写过程中重要的一步，因此，有必要对 GPU 硬件架构有

一个粗略的了解。本文以 Fermi 架构为例，如图 2.3 所示。Fermi 架构从大到小可以分为三层，第一层为图形处理团簇(GPC)层级，每个 GPC 单元可以被看作是一个独立的四核心处理器，由 4 个 SM，一个光栅引擎(Raster Engine)和一个多形体引擎(PolyMorph Engine)组成，共有 4 个 GPC。第二层为 SM 层级，每个 SM 包含 32 个 CUDA 核心，最多同时支持 48 个 warp 的处理。第三层为 ROP 层级，为最小的一级处理单元，称为光栅处理单元。Fermi 架构详细的性能参数可参见表 2.1。

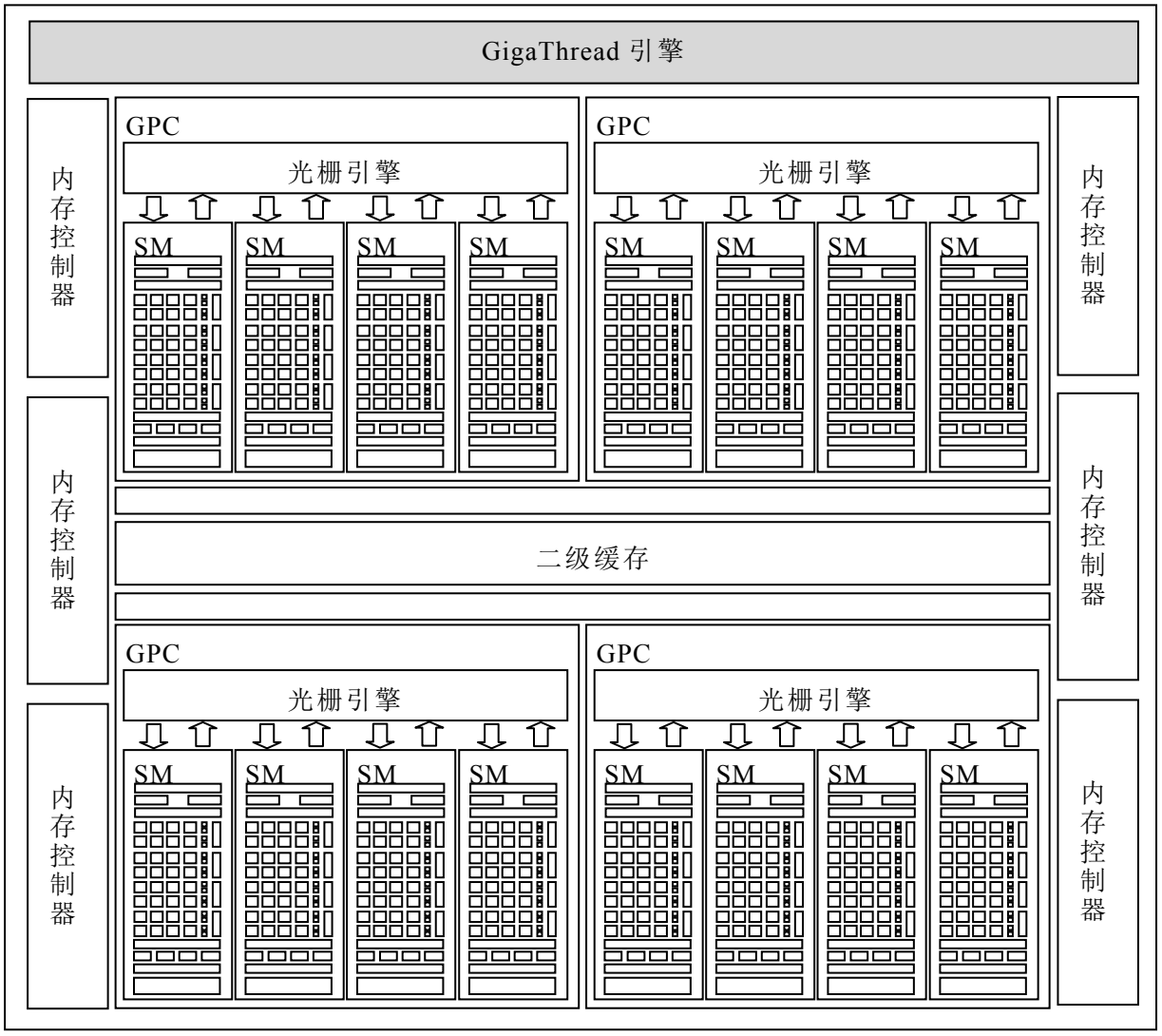


图 2.3 Fermi 架构

2.3 CUDA 的编程和执行模型

2.3.1 CUDA 编程方法

CUDA C 是 CUDA 提供的高级编程语言，事实上，它只是 C 语言一个小范围的扩展集，遵循 C 和 C++语言大多数的语法规则和特性[161]。采用 CUDA C 进行编程有三种方法可以选择，由顶向下分别为 CUDA 库，CUDA 运行时 API 和设备

驱动 API。CUDA 库是基于 CUDA 技术所开发的应用开发库，主要包含有用于线性代数运算的 CUBLAS 以及用于离散快速傅里叶变换 CUFFT。最新版本的 CUDA 中，为了方便的 CUDA 在经济和优化领域中的应用，还包括有用于 GPU 上随机数的生成的 CURAND 等。适当的采用各类 CUDA 库函数，可以在保证计算效率的前提下，极大的简化编程流程。

CUDA 运行时 API 是由 cudart 动态库提供的，它提供了应用开发接口和运行时组件，其 API 函数以 *cuda* 前缀作为标识，可以用于数据类型的定义、内存管理、设备访问和执行调度等操作。CUDA 开发的程序代码在实际执行中分为两种，一种是运行在 CPU 上的主机端代码(Host Code)，一种是运行在 GPU 上的设备端代码(Device Code)，不同类型代码的运行物理位置以及能够访问到的资源都不相同。因此，CUDA 定义有一系列的函数类型限定符，用于区为两类代码。其中，*__device__*限定符声明的函数只能在设备上执行，且只能由设备调用。*__global__* 限定符表示该函数只能在设备上执行，且只能由主机调用。并且该类函数必须是 *void* 类型，也就是说由 GPU 计算的结果不能直接由设备返回到主机中，需要通过 *cudaMemcpy()*等显式数据复制函数*__host__*限定符表明该函数只能由主机调用且只能在主机上执行。

驱动 API 由 nvcuda 动态库提供，以 *cu* 前缀作为标识，提供了硬件设备的统一的抽象访问接口，运行时所提供的各类功能都由这一层来实现，其功能比运行时更加丰富，增加了上下文管理以及模块管理等功能。驱动 API 要求极为严格，使用复杂，因此，一般情况下并不会单独采用该类 API，但可以采用设备驱动 API 和运行时 API 混合使用的方法达到对计算设备最佳访问。

基于 CUDA 开发的并行计算程序，通过 NVCC 的编译和链接所生成的机器代码可以直接运行在支持 CUDA 计算的 GPU 上。为了保证计算程序的通用性，CUDA 采用设备计算能力(compute capability)的概念来协调不同 GPU 的硬件性能差异。对于同一个 CUDA 程序，NVCC 在进行编译会根据当前硬件环境参数来自动修正编译结果，使程序可以在任何支持 CUDA 的 GPU 上执行，例如，当一个采用双精度浮点数编写的程序要在一个计算能力仅为 1.0 的设备上运行时，由于 1.0 的设备不支持双精度浮点数计算，NVCC 会自动将程序编译为单精度浮点格式。

2.3.2 CUDA 线程层次结构

如上文所述，CUDA 是基于轻量级线程的并行执行模型，目前的 GPU 设备可并行执行成千上万个线程。事实上，CUDA 可以支持的线程数目是十分庞大的，基本可以满足现代工程科学中对并行线程数的要求。对于大规模计算时有可能出现的庞大的线程，CUDA 通过 3 个矢量将线程组织成为三个不同级别的层次组，如图 2.4 所示。最底层的为线程块(Block)层次，线程块以三维坐标的形式进行线

程的索引。由于硬件资源的限制，每个线程块容许容纳的线程数目是有限的，如：计算能力为 2.0 的设备最大容纳数目为 1024 个，而计算能力为 2.0 以下的设备最大只能容纳 512 个线程。中间层次为块网格(Grid)，块网络同样以一个三维坐标进行线程块的索引，三个维度分别有不同的上限，但已经足够庞大，如一个计算能力为 2.0 的设备，可以容纳 1024×1024×64，共计 670 多万个线程块。块网格也同样由一个三维矢量组织，是最高级别的层次，且三个维度上的限定值更高，最终可以形成一个庞大的线程组。

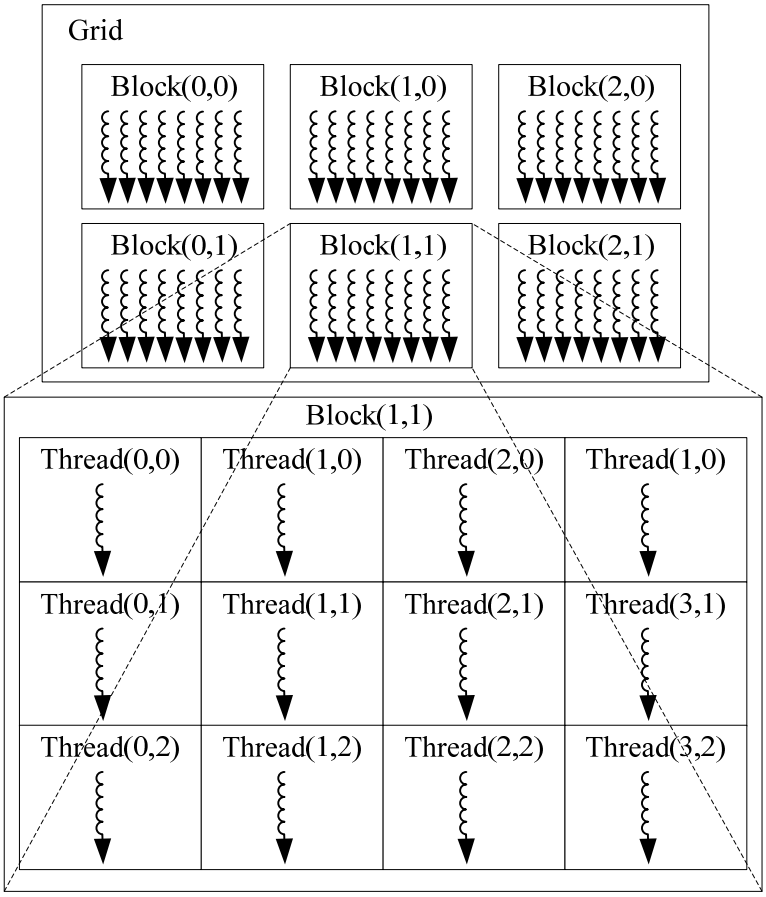


图 2.4 线程块网格

CUDA 程序中，用于 GPU 并行执行的设备端代码称为 kernel 函数，调用 kernel 函数时，程序员可以通过形如<<<*NumofBlocks,NumofThreads*>>>的代码来指定由多少个线程来执行 *kernel* 函数中的指令，其中 NumofThreads 是指每个线程块内的线程数，与最低级线程组对应，而 NumofBlocks 则用于指定线程块的数目，与中间级线程组对应。在 *kernel* 函数中，为了便于线程的访问，CUDA 通过 *blockIdx*，*blockDim* 和 *threadIdx* 这三个内置的三维变量为每个线程指定一个索引号。如图 2.4 所示的二维线程结构，只有一个块网络，块网络的规模为 2×3，而线程块的规模为 3×3。进行线程访问时，一个线程由(i,j)标识，具体的计算方式如框 2.1

所示。另外，在<<<>>>符号内可以配置 *kernel* 计算时的执行流和可使用的共享显存空间大小等。

框 2.1 kernel 函数中线程的访问

```
…
//线程索引号计算
int i = blockIdx.x * blockDim.x + threadIdx.x;
int j = blockIdx.y * blockDim.y + threadIdx.y。
a[i][j] = 1; //线程操作
```

2.3.3 CUDA 存储器模型

在传统并行计算方法中，由于处理器和存储器之间性能差，内存墙 (Memory Wall)是异构计算平台上影响处理器性能发挥的重要因素[162]。CPU 架构中常采用建立较大缓存空间和复杂缓存控制逻辑的方式来减小内存墙的影响。但是，在 GPU 中通常没有或只设有非常有限的缓存空间，不足以降低内存墙的影响，GPU 中缓存空间的主要目的是用于线程对显存空间的随机访问优化和减轻全局存储器带宽压力[163]。同时，由于 GPU 计算时只能访问到显卡上固定的显存空间，因此，GPU 采用多层次的存储模型来解决内存墙对处理器性能的影响。

如图 2.5 所示为现代 GPU 体系中的多层次存储器模型，依据每个存储器特性的不同，CUDA 建立有严格的存储器使用规定。在 GPU 上执行的所有线程都可以访问全局存储器和常数存储器，其中，全局存储器占据了显存的大部分空间，可被线程读取和改写，但是访问速度较慢。CUDA 提供有相应的 API 函数进行全局显存空间的分配、释放、读写操作以及与主机内存之间进行数据传递。常数存储器具有只读属性，容量小但是访问速度快，适合于存储不可变的参数等变量。在同一个线程块内的线程共同使用一块可读写但容量有限的共享存储器，共享存储器的读写速度较快。共享存储器与缓存的作用类似，但需要人为的分配和控制，计算时，可以先将数据从全局存储器中读取到共享存储器中，中间计算结果也可存储在共享存储器中，计算结束后再将结果复制到全局存储器，从而达减少对全局内存访问的目的。另外，每一个线程固定有一定数量的寄存器，寄存器也相当于每一个线程私有的缓存，访问速度最快。寄存器不可以在程序中显式控制，一般由编译器控制用于存储中间结果，但是，可以通过调整编译参数控制寄存器的使用数目，以便满足更多线程同时计算的需要。

与管理程序函数的方式一样，CUDA 同样采用限定符来指定一个变量的类型以及其在设备上的存储位置。*__device__*限定符声明的变量表示该变量为设备上变量，只能由设备访问，并且与应用程序具有相同的生命周期。*__constant__*限定

符声明的变量位于常数存存储器中，同样具有与应用程序相同的生命周期。*__shared__*限定符是存储于共享存储器中的共享变量，生命周期和线程块一样。

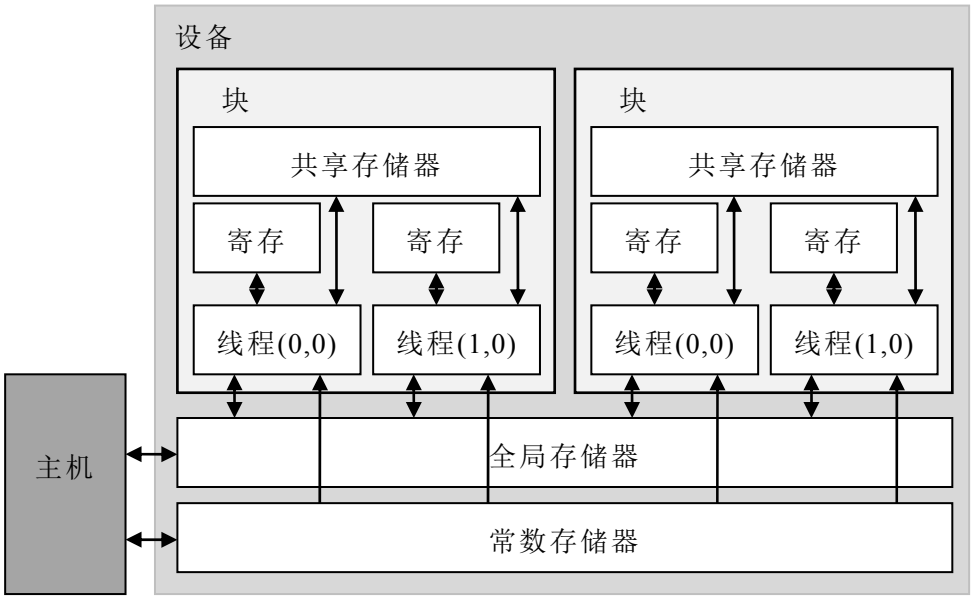


图 2.5 GPU 存储器架构

2.3.4 CUDA 程序执行模式

本文采用的计算平台是至少含有一块支持 CUDA 计算的显卡的个人计算机，从本质上来说，该平台是由主机(Host)和设备(Device)组成的一种 CPU/GPU 异构的计算平台。如图 2.6 所示，平台中主机是指 CPU 和内存，设备则是指支持 CUDA 计算的显卡。采用这种基于 GPU 的 CPU/GPU 异构计算平台进行并行计算的基本流程如图 2.6 所示，共包含有如下四个步骤：

(1)数据准备及初始化工作，即在 CPU 端和 GPU 端分配计算所需的存储空间，并对数据空间进行清零等初始化工作；

(2)将计算所需数据从 CPU 中复制到 GPU 的显存空间中；

(3)基于显存空间的数据，在 GPU 端采用内核函数进行并行计算；

(4)将计算结果从 GPU 的显存空间中传回到 CPU 中。

通过以上步骤可以看出，在 CPU 和 GPU 组成的异构计算平台中， CPU 的主要职责是负责程序的流程控制和执行一些不易并行化的串行代码，GPU 则主要执行并行计算函数，以提高程序的执行效率。

以上 CPU 串行计算、GPU 并行执行以及主机与设备间的数据传输是此类异构计算平台中的三种主要操作形式。并且，由于 GPU 计算时采用的是异步并行[164]的计算模式，即 GPU 的计算可以脱离 CPU 的控制进行，因此，这三种操作在时间上是可重叠的，这是异构平台计算的优势之一。

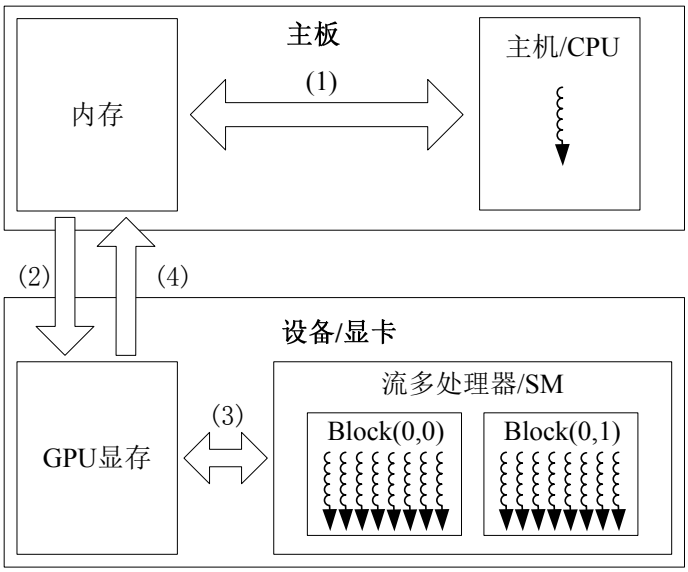


图 2.6 CUDA 架构计算的基本流程

2.4 并行效率评价准则

加速比是评价并行算法和并行机器性能的重要指标，可分为绝对加速比和相对加速比两种类型。绝对加速比是指将同一计算过程的串行算法在串行计算机上运行与并行算法在并行计算机上运行的时间之比。对于 GPU 并行计算平台，即指将程序的 CPU 执行时间 t_{CPU} 和 GPU 执行时间 t_{GPU} 相除，得到 GPU 计算平台的相对计算加速比 S_{pa}，如式(2.1)所示。

$$S_{pa}=\frac{t_{\mathrm{CPU}}}{t_{\mathrm{GPU}}} \tag{2.1}$$

相对加速比主要应用于传统的多节点并行计算机，是指同一并行算法在单个节点上的运行时间与在多节点并行计算机上的运行时间之比，其数学表达式为：

$$S_{pr}=\frac{t_{\mathrm{s}}(I)}{t_{p}(N)} \tag{2.2}$$

本文在以后的章节中，将统一采用式(2.1) 所示的绝对加速比来平衡算法的发性能。同时，为了测试并行程序的稳定性，分析计算规模对加速比的影响，对同一个算法和程序，本文对不同的计算模型将采用不同的单元节点规模分别计算。

2.5 GPU 计算性能的初步测试

2.5.1 测试平台和测试算例

在进行深入研究前，为了验证 GPU 是否适用于进行有限元这类高精度科学计算的需求，本文首先采用简单算法对 GPU 的实际计算性能进行测试。测试采用的

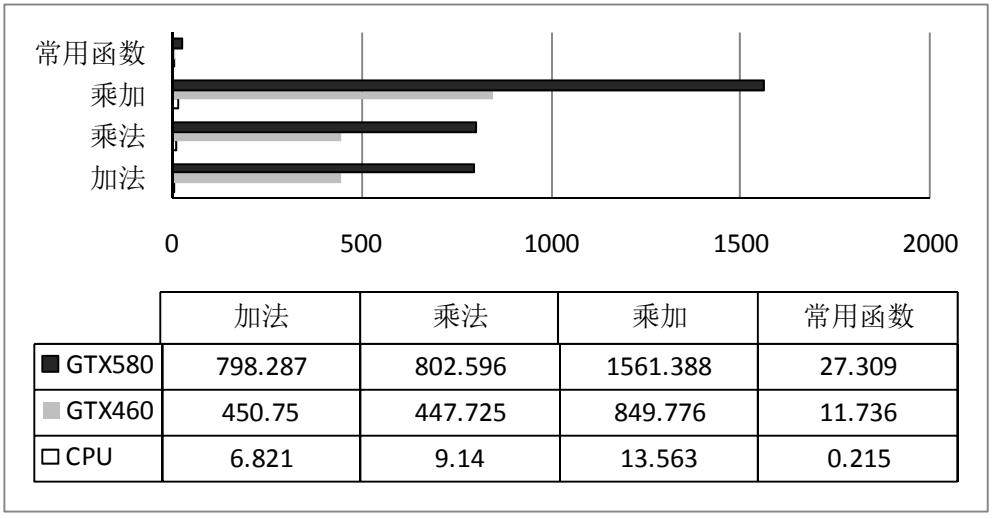
平台如表 2.2 所示，其中，GTX460 和 GTX580 为两种不同规格、采用 Fermi 架构的显卡，同时也是本文以后研究主要采用的计算设备。，其中 I7-930 的市场价格为 2500 元，GTX460 的市场价格为 1000 元，GTX580 的市场价格为 3000 元。

表 2.2 测试平台

名称	型号	主要性能参数
CPU	Intel Core I7-930	主频：2.80GHz，6GB 内存
GPU-1	NVIDIA GTX 460	384 个 CUDA 核，核心频率:675Mhz，1Gb 的显存
GPU-2	NVIDIA GTX 580	512 个 CUDA 核，核心频率:832Mhz，3Gb 的显存
操作系统	Windows 7 64 位	
开发环境	Microsoft VC++ 2010, CUDA 5.0	

2.5.1.1 单精度浮点数运算

单精度浮点数是科学中最基本的浮点数格式，占用 4 个字节的存储空间，数值表示范围为-3.4×10^{38}至3.4×10^{38}。由于通用处理器的设定，单精度浮点数在保证所需计算精度的同时可以获得较好的计算效率。采用的测试方式为计算四种基本的数学操作的操作吞吐量，包括对两个浮点数进行加，减，乘加操作以及几中常用的函数(平方根，幂，正弦和对数)。图 2.7 所示为测试结果，数值单位为 GFLOPS。



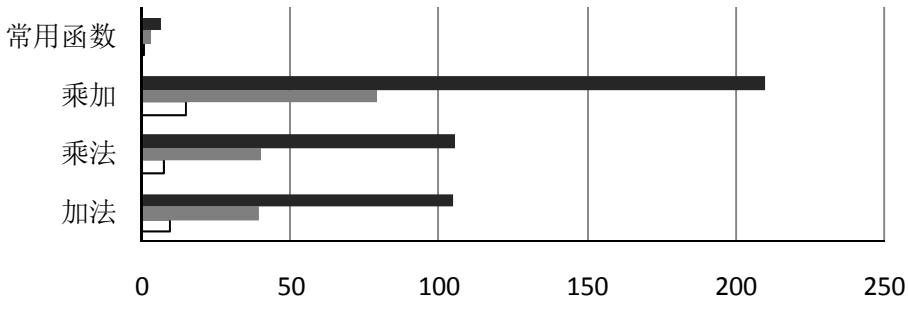
	加法	乘法	乘加	常用函数
GTX580	798.287	802.596	1561.388	27.309
GTX460	450.75	447.725	849.776	11.736
CPU	6.821	9.14	13.563	0.215

图 2.7 单精度浮点数的运算性能对比

2.5.1.2 双精度浮点数运算

有限元分析等高精度科学计算问题中通常需要采用双精度浮点数进行计算，以保证计算精度。因此，相对于单精度浮点数，本文更为关心 GPU 对双精度浮点数的计算能力。双精度浮点数是一种高精度的浮点数格式，需要占用 8 个字节的存储空间，数值表示范围达到-1.7×10^{308}至1.7×10^{308}，拥有 15 或 16 个有效的十进制数字。本次计算采用的数学等式与单精度测试时相同，因此，只需在程序中将

float 型改成 double 型即可。但是，由于只有计算能力达到 1.3 以上的设备才支持双精度数的计算，所以在采用 CUDA 进行程序编译需要将设备计算能力设备 SM=1.3 或者更高，否则，CUDA 编译器还会强行将 double 型计算编译为 float 型。最终所取的测试结果如图 2.8 所示，数值单位为 GFLOPS。

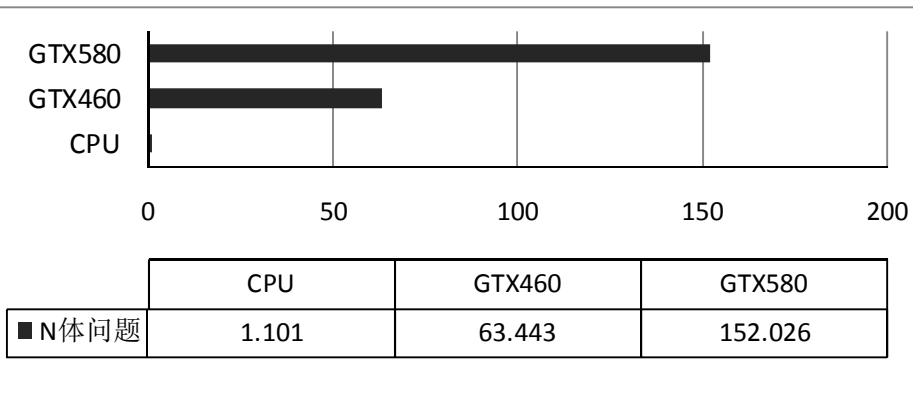


	加法	乘法	乘加	常用函数
GTX580	104.532	104.932	209.516	6.367
GTX460	39.348	39.504	78.864	2.372
CPU	9.121	7.268	14.898	0.138

图 2.8 双精度浮点数的运算性能对比

2.5.1.3 N 体模拟测试

最后本文对 CUDA SDK 中自带的一个 N 体问题程序的计算性能进行测试。N 体问题是天文数学上一个历史悠久的问题，是求解已知位置和初速度的 N 个物体在考虑万有引力时的运动形式。这类问题的每个物体间具有较好的细粒度并行性，同时也需要考虑到各个物体间的相互影响，因此，可以看成是一个极度简化的接触碰撞问题。图 2.9 所示为 10000 个物体采用双精度浮点数时的计算性能对比，结果中的数值单位仍然为 GFLOPS。



	CPU	GTX460	GTX580
N体问题	1.101	63.443	152.026

图 2.9 双精度 N 体问题的计算性能对比

2.5.2 测试结果分析

通过对图 2.7 到图 2.9 所示测试结果的分析可以得到如下几点结论：

(1) 无论是进行单纯的浮点数计算还是实际问题的求解，GPU 的计算性能都要远高于 CPU。对于单精度浮点数计算，两者间的性能比可以达到 100 倍以上。对于双精度浮点数的计算，GPU 的计算性能的下降幅度要大于 CPU，但是两者间的性能比也能达到 50 倍以上，这符合前文对 Fermi 架构的双精度浮点数的峰值执行率为单精度浮点数的 1/2 的描述。由 N 体问题的分析可知，对于实际科学问题，GPU 的计算性能也远大于 CPU，说明了基于 CUDA 架构的 GPU 设备完全适用于通用科学计算。

(2) 对于同一个问题，采用 GTX460 和 GTX580 这两种显卡的吞吐量也不同。尽管 GTX460 和 GTX580 均采用 Fermi 架构，但是两者拥有的 CUDA 核心以及核心频率均不相同，这对两者的计算性能必然有较大的影响。GTX580 的 CUDA 核心数和核心频率均高于 GTX460，所以对于以上的计算问题其性能都高于 GTX460 两倍左右。同时，GTX580 拥有的显存也更大，因此，更利于计算大规模的科学问题。

(3) 以上 3 个测试算例基本涉及到了显式有限元计算过程中会涉及到的数值运算。由 GPU 具有优势的计算性能可以预见采用 GPU 进行显式有限元的计算必然可以在保证计算精度的同时取得较好的计算加速比。

2.6 本章小结

本章首先简单介绍了 SISD，SIMD，MISD，MIMD 等现代计算机结构以及 DM、SM 和 DSM 等三种并行计算机架构和它们主要应用领域。对并行算法中常用的编程模型也作了简要说明，并指明 GPU 并行计算所对应的硬件架构和编程模型。

其次，本章简单的描述了 GPU 的发展历史和其可编程性原理，分析说明了 CPU 和 GPU 在计算性能上的差异。GPU 和 CPU 在性能上的差异主要来源于两者设计目标的不同，而导致的控制单元、逻辑运算单元和存储单元三者的晶体数量配比上的差异，CPU 主要将晶体管用于控制单元，而 GPU 则将大部分的晶体管用于计算单元。

CUDA 是由 NVIDIA 公司针对 GPU 通用计算推出的统一计算架构，是本文研究并行计算方法所采用计算架构和编程语言。本章对 CUDA 的编程方法、线程管理和程序执行模型等内容进行了详细的介绍。GPU 通用计算是采用大量轻量线程并发执行的并行计算模式，CUDA 提供了多级线程管理模式，既适应于 GPU 的硬件特征也方便程序员对大规模线程进行管理。现代 GPU 提供了多层存储器模型用

于减小内存墙对计算效率的影响，CUDA 提供了相应的类型限定符对不同存储空间进行管理。采用 CUDA 计算时，计算数据需要由主机端复制到设备端，由设备端并行计算得到计算结果后再显式传回。

最后，本文采用三种类型的算例对 GPU 进行浮点数计算和实际科学问题的求解性能进行了测试。测试结果表明，相比于同时期的 CPU，对于单精度浮点数的计算，GPU 可以取得 100 倍以上的性能优势，对于双精度浮点数可以取得近 50 倍以上的优势，这满足 Fermi 架构中双精度浮点数和单精度浮点数两倍性能差异。同时，对于实际科学问题 GPU 同样具有较大的计算性能优势。这说明了采用 GPU 加速显式有限元的计算必然可以取得较好的结果。

第 3 章 基于 GPU 的板壳单元显式并行计算方法

板壳结构是汽车车身的常用结构，板壳单元是板壳结构有限元分析的基本构成，对板壳单元的并行化计算方法的研究是板壳结构有限元并行计算方法研究的基础。显式算法是解决板壳结构非线性问题的常用算法，因此，本文首先研究基本显式单元算法在 GPU 上的并行计算方式，并从中研究和提取适用于 GPU 计算的基本显式有限元并行计算策略。在不考虑接触界面的影响下，本章主要研究 BT 四边形壳单元以及 EST 边光滑三角形壳元这两种显式格式的壳单元在 GPU 上的并行计算方法。

3.1 板壳结构非线性有限元分析算法和程序

3.1.1 中心差分格式的显式积分算法

对于任何一个结构动力学分析问题，采用有限元方法进行求解时，本质上都可归纳为如下的常微分方程的初值问题

$$\boldsymbol{K}\boldsymbol{u}(t)+\boldsymbol{C}\dot{\boldsymbol{u}}(t)+\boldsymbol{M}\ddot{\boldsymbol{u}}(t)=\boldsymbol{F}(t) \tag{3.1}$$

式中 $\ddot{\boldsymbol{u}}(t)$ 表示加速度，$\dot{\boldsymbol{u}}(t)$ 为速度，$\boldsymbol{u}(t)$ 表示位移，$\boldsymbol{F}(t)$ 表于载荷，$\boldsymbol{K}$ 表示刚度矩阵，$\boldsymbol{C}$ 表示阻尼矩阵，$\boldsymbol{M}$ 表示质量矩阵。有限元计算的数值问题主要包括有：线性代数方程求解、代数特征值的计算、积分算法等，求解这类问题有两种方法：显式算法和隐式算法。对于非线性动态问题的求解，常用的隐式积分算法有 Wilson-θ，Newmark-β 方法等，但是，由于需要每个迭代步内都要求解复杂的非线性方程组，计算时间和计算空间的花费很高，同时，隐式积分算法的稳定性也差，所以较少应用于动态问题的求解。相比之下，显式算法由于引入对角的集中质量矩阵，不需要求解方程组，计算花费小，因而比隐式算法更适合于解决大规模动态线性和非线性问题[50]。同时，显式算法具有极强的计算稳健性，能够处理非常复杂的接触和变形问题，因此，本文主要研究显式格式的接触碰撞问题的有限元算法。

目前，常用的显式积分格式有单边差分和中心差分两种。单边差分格式只具有一阶精度的格式，计算公式简单，但受精度所限在实际工程中的应用不多。中心差分格式是一种二阶精度的显式积分方法，根据其思想，首先需要将时间域 $[0,T]$ 离散成为 N 个时间步，每个时间步为 Δt_n

$$\Delta t_n = t_{n+1}-t_n, n=[1,N] \tag{3.2}$$

式中，t_n 分别代表第 n 时间步的时间点。

基于以上的离散思想，在中心差分思想中，速度的计算是可以通过位移的一阶中心差分给出

$$\boldsymbol{v}_{n+\frac{1}{2}}=\frac{\boldsymbol{x}_{n+1}-\boldsymbol{x}_n}{t_{n+1}-t_n}=\frac{\boldsymbol{x}_{n+1}-\boldsymbol{x}_n}{\Delta t_n} \tag{3.3}$$

式中，$\boldsymbol{x}_n$ 分别代表第 n 时间步的位移。通过移项变换可以得到第 $n+1$ 时间步内的位移积分公式，成而实现由差分公式到积分的变换

$$\boldsymbol{x}_{n+1}=\boldsymbol{x}_n+\boldsymbol{v}_{n+\frac{1}{2}}\Delta t_n \tag{3.4}$$

考虑到一般有限元问题的基本运动方程

$$\boldsymbol{M}\boldsymbol{a}^n+\boldsymbol{F}_{\text{int}}^n=\boldsymbol{F}_{\text{ext}}^n \tag{3.5}$$

式中，$\boldsymbol{M}$ 为质量矩阵，$\boldsymbol{a}$ 为加速度矢量，$\boldsymbol{F}_{\text{int}}$ 为内力矢量，$\boldsymbol{F}_{\text{ext}}$ 为外力矢量，n 代表当前时间步。进行空间离散后，得到对时间的二阶常微分方程

$$\boldsymbol{M}\boldsymbol{a}^n=F^n=f_{ext}(x_n,t_n)-f_{\text{int}}(x_n,t_n) \tag{3.6}$$

结合式(3.3)，式(3.4)和式(3.6)可得

$$\boldsymbol{v}_{n+\frac{1}{2}}=\boldsymbol{v}_{n-\frac{1}{2}}+\Delta t\boldsymbol{M}^{-1}F^n \tag{3.7}$$

显式积分过程中，对于任意时间步 n，由已知位移 $\boldsymbol{x}_n$ 通过应变-位移公式、本构方程和节点外力计算，可以确定节点外力 F^n。这样式(3.7)的右边已经全部已知，从而可以计算得到 $\boldsymbol{v}_{n+\frac{1}{2}}$，然后利用式(3.4)获得下一时间点位移 $\boldsymbol{x}_{n+1}$。

时间步长是影响中心差分法收敛性的重要因素[165]，为了使计算过程稳定收敛，根据单元的物理特性，存在有一个临界值 Δt_{crit}，如果时间步长取值大于 Δt_{crit}，其计算结果将会发散，趋近于无穷大。

$$\vartriangle t\le\vartriangle t_{crit}=\frac{T_{\min}}{\pi} \tag{3.8}$$

式中，$T_{\min}$ 代表当前有限元系统的最小固有频率。对于壳单元而言，最小单元的最小固有频率为

$$T_{\min}=\frac{L_{\min}}{c}\pi \tag{3.9}$$

式中，$L_{\min}$ 表示有限元系统中尺寸最小单元的特征长度，常用的计算方式如下

$$L_{\min}=\frac{(1+\beta)A}{\max\left(L_1,L_2,L_3,(1-\beta)L_4\right)} \tag{3.10}$$

式中，A 表示壳单元的面积；L_1，L_2，L_3 和 L_4 为壳单元的边长；β 单元形状系数，通常情况下，壳单元为四边形时 $\beta=1$，为三角形时 $\beta=0$；另外，公式(3.9)中 c 变

材料内的声速，计算公式为

$$c=\sqrt{\frac{E}{\rho(1-\upsilon^2)}} \tag{3.11}$$

其中，E为材料弹性模量；ρ材料质量密度；υ为泊松比。

综合式(3.8)到(3.11)，中心差分法的稳定性条件式可以表示为

$$\Delta t \le \Delta t_{crit}=L_{\min}\sqrt{\frac{\rho(1-\upsilon^2)}{E}} \tag{3.12}$$

由此可知，时间步长决定于最小单元的尺寸，当单元尺寸变小时，会使临界时间步长变小，从而增加整个计算时间。在有限元分析程序中，时间步长的计算有定时间步长和变时间步长两种方式，对于本文所关注的大变形问题，每个时间步内单元变形较大，因而变时间步长是十分必要的，即在每一个时间步都需要根据当前构形单元中的最小单元重新计算时间步长。

3.1.2 单元技术

单元技术是有限元分析的重要组成部分。薄膜是壳体结构最简单的近似，因此，薄膜单元首先被用于壳体结构的分析。但是，薄膜单元在计算时只考虑到中面的变形，不考虑弯曲效应，这大大简化了计算过程，却不适用于含有复杂弯曲等壳体变形的模拟。

后来，逐步提出了壳单元的概念，壳单元算法满足板壳结构中剪切力自锁，能够消除零能模式以及计算效率高等要求。目前常用的壳单元理论有柯西霍夫(Kirchhoff)理论和梅得林 (Mindlin)理论两种。Kirchhoff 理论假设板壳变形前后垂直于中面的法线在变形后仍然保持为直线且与上面保持垂直，即忽略了厚度方向上的剪切变形，因此一般情况下仅适用于厚度曲率比超过 1/20 的薄板。然而，实际工程中很难严格的区分所使用的材料是属于薄板还是中厚板，并且由于忽略厚度方向上的剪切变形，Kirchhoff 理论对中厚板壳的计算结果有一定的误差。因此，本文研究主要采用 Mindlin 厚板理论的壳单元，Mindlin 厚板理论既适用于薄板，也适用于中厚板，它假设变形后垂直于中面的法线变形后仍然保持直线，但是不一定垂直于变形后的中面。计算时，Mindlin 厚板理论将厚度方向的位移和转动分开计算，因此，得到的单元计算形式简单、高效。

3.1.2.1 Belytschlko-Tsay 四节点壳单元

基于退化壳元理论和一点积分方法的 Belytschlko-Tsay(BT)四边形单元[53]是显式有限元分析中最常用的壳单元，其计算流程和其它类型的壳单元具有很好的共性基础。为了了解壳单元的并行计算性能，本文对 BT 单元的计算流程进行简要的讨论。

首先是进行局部随动坐标系的设定，如图 3.1 所示，该坐标系的基矢量定义

为

$$\hat{e}_3 = \frac{r_{31} \times r_{42}}{|r_{31} \times r_{42}|} \tag{3.13}$$

$$\hat{e}_2 = \frac{r_{21} - (r_{21} \cdot \hat{e}_3)\hat{e}_3}{|r_{21} - (r_{21} \cdot \hat{e}_3)\hat{e}_3|} \tag{3.14}$$

$$\hat{e}_2 = \hat{e}_3 \times \hat{e}_1 \tag{3.15}$$

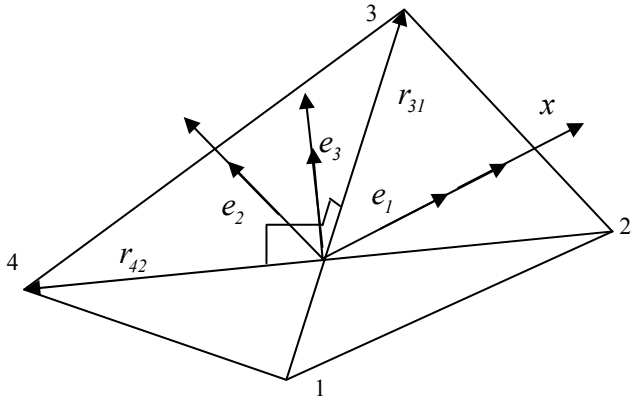


图 3.1 局部随动坐标系

根据 Mindlin 理论，BT 壳体中任一节点的速度可以表示为

$$V = V^m - \hat{Z}\hat{e}_3 \times Q \tag{3.16}$$

式中，V^m表示参考面上的速度，Q表示角速度，$\hat{Z}$表示沿厚度方向的距离。结合式(3.16)可以得到速度应变的各个分量为

$$\begin{aligned}
&\hat{d}_1 = \frac{\partial \hat{v}_1^m}{\partial \hat{x}_1} + \hat{x}_3 \frac{\partial \hat{Q}_1}{\partial \hat{x}_1}, \hat{d}_2 = \frac{\partial \hat{v}_2^m}{\partial \hat{x}_2} + \hat{x}_3 \frac{\partial \hat{Q}_2}{\partial \hat{x}_2} \\
&2\hat{d}_{12} = \frac{\partial \hat{v}_1^m}{\partial \hat{x}_2} + \frac{\partial \hat{v}_2^m}{\partial \hat{x}_1} + \hat{x}_3 \left(\frac{\partial \hat{Q}_2}{\partial \hat{x}_1} - \frac{\partial \hat{Q}_1}{\partial \hat{x}_2}\right) \\
&2\hat{d}_{23} = \frac{\partial \hat{v}_3^m}{\partial \hat{x}_2} - \hat{Q}_1, 2\hat{d}_{13} = \frac{\partial \hat{v}_3^m}{\partial \hat{x}_2} + \hat{Q}_2
\end{aligned} \tag{3.17}$$

结合式(3.5)和虚功原理，可得到：

$$\partial Q_I^{eT} m_I^e + \partial v_I^{eT} f_I^e = \int_{v^e} \hat{\delta}^T \hat{\sigma} dv \tag{3.18}$$

式中

$$\begin{aligned}
&\hat{\delta}^T = \left[\hat{d}_{11}, \hat{d}_{22}, 2\hat{d}_{12}, 2\hat{d}_{13}, 2\hat{d}_{33}\right] \\
&\hat{\sigma}^T = \left[\hat{\sigma}_{11}, \hat{\sigma}_{22}, \hat{\sigma}_{12}, \hat{\sigma}_{13}, \hat{\sigma}_{23}\right] \\
&f_I^{eT} = \left[f_{1I}, f_{2I}, f_{3I}\right], m_I^{eT} = \left[m_{1I}, m_{2I}, m_{3I}\right]
\end{aligned} \tag{3.19}$$

对于一点积分单元，即在一个单元中，弯矩 m_I^e 和膜力 f_I^e 只计算一次。因此，为了保证计算精度，对于弹塑性材料，沿厚度方向可以取 3 到 7 个积分点，并通过插值进行计算。

如图 3.2 所示的退化四节点壳单元，单元中的任一节点可以由节点坐标及节点矢量通过形函数插值得到

$$x_i = \bar{N}^a (X_i^a + \frac{\zeta_3}{2} V_i^a) \tag{3.20}$$

式中，X_i^a 为中面上节点 a 的坐标，$\bar{N}^a$ 为插值形函数，ζ_3 为随动坐标系分量，V_i^a 为节点从下表面指向上表面的矢量

$$V_i^a = X_{i(upper)}^a - X_{i(lower)}^a \tag{3.21}$$

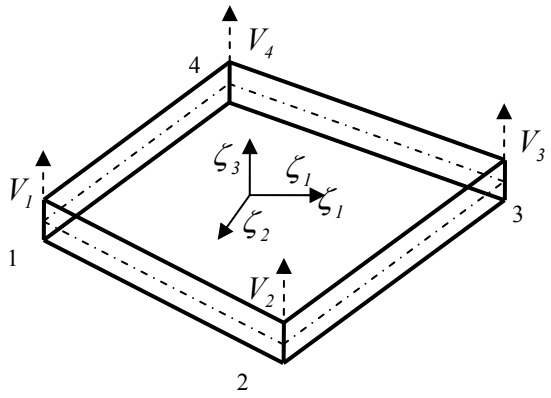


图 3.2 四节点退化壳单元

壳单元内任一点的位移也从节点插值得到。壳单元中每个节点有五个自由度，分别为沿三个方向上平动自由度和沿平面轴的两个转动自由度。由于计算时一个时间步很小，所以可以认为转动度很小，矢量 V_i^a 仍然保持直线方向，因此，位移可以表示为

$$u_i = N_{ik}^a U_k^a \quad , \quad (k = 1,2,3,4,5) \tag{3.22}$$

式中，U_k^a 为节点位移，N_{ik}^a 为插值函数。同样速度也采用相同的形函数进行插值。

$$N_{ik}^a = \begin{cases} \delta_{ik} \bar{N}^a, k = 1,2,3 \\ -\frac{h}{2} \bar{N}^a \zeta_3 \delta_{2i}, k = 4 \\ \frac{h}{2} \bar{N}^a \zeta_3 \delta_{1i}, k = 5 \end{cases} \tag{3.23}$$

式中，h 为板壳的厚度，δ_{ik} 为 Kronecker 符号。

最终可以得到单元节点内力的具体表达式为

$$\begin{aligned} f_{1I} &= A(B_{1I}f_1 + B_{2I}f_{12}) \\ f_{2I} &= A(B_{2I}f_2 + B_{1I}f_{12}) \\ f_{3I} &= A\bar{k}(B_{1I}f_{13} + B_{1I}f_{12}) \\ m_{1I} &= A(B_{2I}\,\mathrm{m}_2 + B_{1I}\,\mathrm{m}_{12} - \frac{1}{4}\bar{k}f_{23}) \\ m_{2I} &= A(B_{1I}\,\mathrm{m}_1 + B_{2I}\,\mathrm{m}_{12} - \frac{1}{4}\bar{k}f_{13}) \\ m_{3I} &= 0 \end{aligned} \tag{3.24}$$

式中，A 为单元面积，$\bar{k}$ 为剪切修正系数，$B_{1I} = \partial N_I / \partial \hat{x}_1$，$B_{2I} = \partial N_I / \partial \hat{x}_2$。

需要注意的是，尽管一点积分可以有效的避免“锁定”问题，但是由于单元秩不足，会引起零能模式，即沙漏模式。为此，BT 单元通过加入人为的应变来控制沙漏，通过选取合适的沙漏控制系数既能控制沙漏又不致引起“锁定”。

3.1.2.2 边光滑三角形壳单元

在工程应用中进行有限元分析时，前处理将占用大量的时间，特别是像诸如汽车碰撞仿真此类大规模接触碰撞问题的仿真，在极端情况下，前处理中的网格划分时间甚至会超过有限元计算的时间。现有商业有限元分析软件中，三角形单元具有较好的网格自动生成和网格大小自适用能力，但是，它们的计算精度却很难满足工程需求。因此，在工程问题分析中，为了保证计算精度，还是需要采用四边形单元进行计算。但是，四边形单元很难完成复杂曲面的构形，并且由于积分点的增加，计算效率也较低。为此，本文也将研究由李光耀[139, 140]开发的高精度边光滑三角壳片(Edged-based smoothed triangular shell element, EST)。EST 单元是采用边光滑技术构造的一种不需要增加额外自由度和参数的高精度、低阶三角形单元。

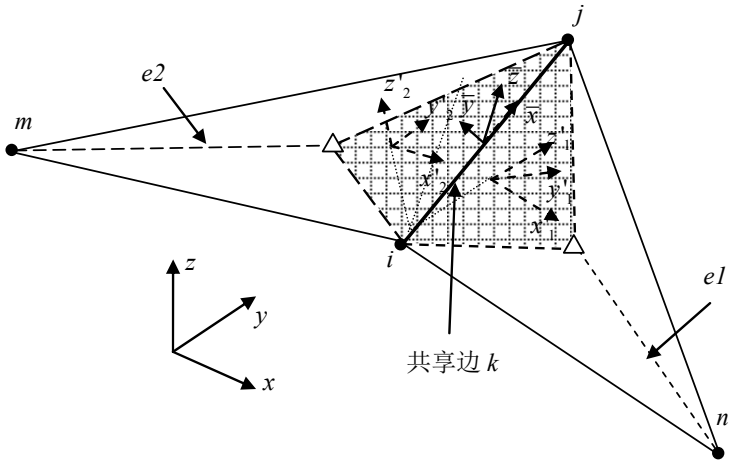


图 3.3 三角形壳单元的边光滑域

采用三角形壳单元进行平面问题分析时，几何形状被离散为一系列的三角形

单元，但是，由于各个三角形单元不在同一个平面上，因而需要采用等参计算的方式，在每个单元内建立一个局部坐标系，依照李光耀[166]的思想，该局部坐标系的组成如下：$\bar{x}$ 与边 k 的方向相同，$\bar{z}$ 取两个相邻单元法向的平均值，$\bar{y}$ 由 $\bar{z}$ 和 $\bar{x}$ 的叉积得到，如图 3.3 所示。图中所示为相邻两个单元 e_1 和 e_2 基于共享边 k 的构造成的光滑域 Ω_k，即图中有填充部分。

在图中基于边的局部坐标系下，壳单元的膜应变 $\hat{\boldsymbol{\varepsilon}}_m$、曲率 $\hat{\boldsymbol{\kappa}}$ 和剪应变 $\hat{\boldsymbol{\gamma}}$ 可分别表示为：

$$\bar{\boldsymbol{\varepsilon}}_m = \mathbf{R}_{m1}\mathbf{R}_{m2}\hat{\boldsymbol{\varepsilon}}_m \tag{3.25}$$

$$\bar{\boldsymbol{\kappa}} = \mathbf{R}_{b1}\mathbf{R}_{b2}\hat{\boldsymbol{\kappa}} \tag{3.26}$$

$$\bar{\boldsymbol{\gamma}} = \mathbf{R}_{s1}\mathbf{R}_{s2}\hat{\boldsymbol{\gamma}} \tag{3.27}$$

式中，$\mathbf{R}_{m1}$、$\mathbf{R}_{m2}$、$\mathbf{R}_{b1}$、$\mathbf{R}_{b2}$、$\mathbf{R}_{s1}$ 和 $\mathbf{R}_{s2}$ 为转换矩阵：

$$\mathbf{R}_{m1} = \mathbf{R}_{b1} = \begin{bmatrix} c_{\bar{x}x}^2 & c_{\bar{x}y}^2 & c_{\bar{x}z}^2 & c_{\bar{x}x}c_{\bar{x}y} & c_{\bar{x}y}c_{\bar{x}z} & c_{\bar{x}x}c_{\bar{x}z} \\ c_{\bar{y}x}^2 & c_{\bar{y}y}^2 & c_{\bar{y}z}^2 & c_{\bar{y}x}c_{\bar{y}y} & c_{\bar{y}y}c_{\bar{y}z} & c_{\bar{y}x}c_{\bar{y}z} \\ 2c_{\bar{x}x}c_{\bar{y}x} & 2c_{\bar{x}y}c_{\bar{y}y} & 2c_{\bar{x}z}c_{\bar{y}z} & c_{\bar{x}x}c_{\bar{y}y}+c_{\bar{y}x}c_{\bar{x}y} & c_{\bar{x}z}c_{\bar{y}y}+c_{\bar{y}z}c_{\bar{x}y} & c_{\bar{x}x}c_{\bar{y}z}+c_{\bar{y}x}c_{\bar{x}z} \end{bmatrix} \tag{3.28}$$

$$\mathbf{R}_{s1} = \begin{bmatrix} 2c_{\bar{x}x}c_{\bar{z}x} & 2c_{\bar{x}y}c_{\bar{z}y} & 2c_{\bar{x}z}c_{\bar{z}z} & c_{\bar{x}x}c_{\bar{z}y}+c_{\bar{z}x}c_{\bar{x}y} & c_{\bar{x}z}c_{\bar{z}y}+c_{\bar{z}z}c_{\bar{x}y} & c_{\bar{x}x}c_{\bar{z}z}+c_{\bar{z}x}c_{\bar{x}z} \\ 2c_{\bar{y}x}c_{\bar{z}x} & 2c_{\bar{y}y}c_{\bar{z}y} & 2c_{\bar{y}z}c_{\bar{z}z} & c_{\bar{y}x}c_{\bar{z}y}+c_{\bar{z}x}c_{\bar{y}y} & c_{\bar{y}z}c_{\bar{z}y}+c_{\bar{z}z}c_{\bar{y}y} & c_{\bar{y}x}c_{\bar{z}z}+c_{\bar{z}x}c_{\bar{y}z} \end{bmatrix} \tag{3.29}$$

$$\mathbf{R}_{m2} = \mathbf{R}_{b2} = \begin{bmatrix} c_{\hat{x}x}^2 & c_{\hat{y}x}^2 & c_{\hat{x}x}c_{\hat{y}x} \\ c_{\hat{x}y}^2 & c_{\hat{y}y}^2 & c_{\hat{x}y}c_{\hat{y}y} \\ c_{\hat{x}z}^2 & c_{\hat{y}z}^2 & c_{\hat{x}z}c_{\hat{y}z} \\ 2c_{\hat{x}x}c_{\hat{x}y} & 2c_{\hat{y}x}c_{\hat{y}y} & c_{\hat{x}x}c_{\hat{y}y}+c_{\hat{x}y}c_{\hat{y}x} \\ 2c_{\hat{x}y}c_{\hat{x}z} & 2c_{\hat{y}y}c_{y'z} & c_{\hat{x}y}c_{\hat{y}z}+c_{\hat{x}z}c_{\hat{y}y} \\ 2c_{\hat{x}x}c_{\hat{x}z} & 2c_{\hat{y}x}c_{y'z} & c_{\hat{x}x}c_{\hat{y}z}+c_{\hat{x}z}c_{\hat{y}x} \end{bmatrix} \tag{3.30}$$

$$\mathbf{R}_{s2} = \begin{bmatrix} c_{\hat{x}x}c_{\hat{z}x} & c_{\hat{y}x}c_{\hat{z}x} \\ c_{\hat{x}y}c_{\hat{z}y} & c_{\hat{y}y}c_{\hat{z}y} \\ c_{\hat{x}z}c_{\hat{z}z} & c_{\hat{y}z}c_{\hat{z}z} \\ c_{\hat{x}x}c_{\hat{z}y}+c_{\hat{x}y}c_{\hat{z}x} & c_{\hat{y}x}c_{\hat{z}y}+c_{\hat{y}y}c_{\hat{z}x} \\ c_{\hat{x}z}c_{\hat{z}y}+c_{\hat{x}y}c_{\hat{z}z} & c_{\hat{y}z}c_{\hat{z}y}+c_{\hat{y}y}c_{\hat{z}z} \\ c_{\hat{x}x}c_{\hat{z}z}+c_{\hat{x}z}c_{\hat{z}x} & c_{\hat{y}x}c_{\hat{z}z}+c_{\hat{y}z}c_{\hat{z}x} \end{bmatrix} \tag{3.31}$$

式中，$c_{\bar{x}x}$ 表示 $\bar{x}$ 轴和 x 轴夹角的余弦。

根据三角形壳单元的插值理论，可以得到边局部坐标系下应变与单元局部坐标下节点位移向量之间的关系：

$$\bar{\boldsymbol{\varepsilon}}_m = \mathbf{R}_{m1}\mathbf{R}_{m2}\hat{\mathbf{B}}_m\hat{\mathbf{d}} \tag{3.32}$$

$$\bar{\boldsymbol{\kappa}} = \mathbf{R}_{b1}\mathbf{R}_{b2}\hat{\mathbf{B}}_b\hat{\mathbf{d}} \tag{3.33}$$

$$\boldsymbol{\gamma} = \mathbf{R}_{s1}\mathbf{R}_{s2}\hat{\mathbf{B}}_s\hat{\mathbf{d}} \tag{3.34}$$

式中，$\hat{\mathbf{B}}_m$、$\hat{\mathbf{B}}_b$和$\hat{\mathbf{B}}_s$为单元局部坐标系下的应变位移矩阵，$\hat{\mathbf{d}}$为局部坐标系下的节点位移向量：

$$\hat{\mathbf{d}}_i = \hat{\mathbf{T}}\mathbf{d}_i \tag{3.35}$$

$\mathbf{d}_i$为全局坐标系下的节点位移向量，转换矩阵$\hat{\mathbf{T}}$可表示为：

$$\hat{\mathbf{T}} = \begin{bmatrix} \mathbf{T}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{2\times3} & \mathbf{T}_{2\times3} \end{bmatrix} \tag{3.36}$$

其中，

$$T_{ij} = c_{\hat{i}\hat{j}} \tag{3.37}$$

如图 3.3 所示，沿共享边 k 构造的积分域由相邻两单元两个子域构造而成，将在基于边的局部坐标系下的第 I 个子域 Ω_{kI} 内的膜应变，曲率和剪切应变分别定义为$\bar{\boldsymbol{\varepsilon}}_m^{(I)}$，$\bar{\boldsymbol{\varepsilon}}_b^{(I)}$和$\bar{\boldsymbol{\varepsilon}}_s^{(I)}$。由于采用线性插值，因而应变值为常数，则积分域内的平均应变可以由下式得到

$$\bar{\boldsymbol{\varepsilon}}_{m(k)} = \frac{1}{A_k}\sum_I A_{kI}\bar{\boldsymbol{\varepsilon}}_m^{(I)} \tag{3.38}$$

$$\bar{\boldsymbol{\varepsilon}}_{b(k)} = \frac{1}{A_k}\sum_I A_{kI}\bar{\boldsymbol{\varepsilon}}_b^{(I)} \tag{3.39}$$

$$\bar{\boldsymbol{\varepsilon}}_{s(k)} = \frac{1}{A_k}\sum_I A_{kI}\bar{\boldsymbol{\varepsilon}}_s^{(I)} \tag{3.40}$$

式中，A_{kI}是子区域 Ω_{kI}的面积。

由式(3.35)-(3.40)得到积分域内的平均应变与全局坐标系下位移向量的关系为：

$$\bar{\boldsymbol{\varepsilon}}_{m(k)} = \bar{\mathbf{B}}_{m(k)}\mathbf{d} \tag{3.41}$$

$$\bar{\boldsymbol{\varepsilon}}_{b(k)} = \bar{\mathbf{B}}_{b(k)}\mathbf{d} \tag{3.42}$$

$$\bar{\boldsymbol{\varepsilon}}_{s(k)} = \bar{\mathbf{B}}_{s(k)}\mathbf{d} \tag{3.43}$$

式中，

$$\bar{\mathbf{B}}_{m(k)} = \frac{1}{A_k}\sum_I A_{kI}\mathbf{R}_{m1(k)}\mathbf{R}_{m2}^{(I)}\mathbf{B}_m'^{(I)}\mathbf{T}^{(I)} \tag{3.44}$$

$$\bar{\mathbf{B}}_{b(k)} = \frac{1}{A_k}\sum_I A_{kI}\mathbf{R}_{b1(k)}\mathbf{R}_{b2}^{(I)}\mathbf{B}_b'^{(I)}\mathbf{T}^{(I)} \tag{3.45}$$

$$\bar{\mathbf{B}}_{s(k)}=\frac{1}{A_k}\sum_I A_{kI}\mathbf{R}_{s1(k)}\mathbf{R}_{s2}^{(I)}\mathbf{B}_s^{(I)}\mathbf{T}^{(I)} \tag{3.46}$$

3.1.3 显式算法的串行执行流程图

根据以上的理论分析，采用显式有限元算法对板壳结构进行非线性有限元分析程序的基本计算流程如框 3.1 所示[167]。主要包含以下几个步骤：

框 3.1 显式时间积分的流程图

1. 初始条件和初始化：

读取输入文件(模型数据，边界条件，材料参数等)；

设定状态参数初始值：初始运动速度 v^0，初始时间步长 σ^0 等；

计算质量矩阵 $\boldsymbol{M}$

2. 计算节点 $\boldsymbol{F}^n$

3. 计算加速：$\boldsymbol{a}_n=\boldsymbol{M}^{-1}(\boldsymbol{F}^n-\boldsymbol{C}\boldsymbol{v}_{n-\frac{1}{2}})$

4. 计算时间更新：$t_{n+1}=t_n+\Delta t_{n+\frac{1}{2}}$，$t_{n+\frac{1}{2}}=\frac{1}{2}(t_n+t_{n+1})$

5. 更新节点速度：$v_{n+\frac{1}{2}}=v_n+(t_{n+\frac{1}{2}}-t_n)a_n$.

6. 更新节点位移 $d_{n+1}=d_n+\Delta t_{n+\frac{1}{2}}v_{n+\frac{1}{2}}$

7. 单元计算给出作用力 $\boldsymbol{F}^n$

8. 更新时间步长

9. 迭代次数更新 $n=n+1$

10. 中间结果和后处理数据的输出

11. 收敛判断，如果没达到收敛条件，返回 4

12. 收敛结束

(1) 程序准备部分，由第 1 步组成，主要完成输入文件的读取，一个完全的有限元分析输入文件应该要包含有模型的单元节点信息，载荷和运动信息，约束信息以及完全的材料信息。同时还需要对程序计算所需的各个数组和参数进行置零。

(2) 有限元计算部分。包含由第 2、3、5、6、8 步组成的节点物理量计算部分，首先计算节点力，用节点力除以质量得到节点的加速度，再由加速度和时间步长计算得到当前速度，由速度计算当前节点位移，最后由位移更新节点的位置。由第 7 步组成的单元计算部分，通过单元计算得到节点的作用内力是整个程序流程中

最复杂且耗时最多的部分。对应于本文 3.1.2 节的介绍的单元技术部分，可以得到如框 3.2 所示的基本计算流程。计算流程主要由局部坐标系计算、变形量计算、本构计算、单元内力计算以及内力离散等部分组成。最后，要对时间步长进行更新。

(3) 数据输出部分。由第 10 步组成，主要将计算所得的各类所需的中间结果存储到输出文件中，以用于有限元后处理的需要。

(4) 显式迭代控制部分。由第 4、9、11 步组成，用于控制程序的执行，主要工作包括：更新当前仿真的时间点以及迭代步数总和，同时，根据输入文件的设置，对收敛情况以及程序终止条件进行判断。

通常情况下，在对程序进行并行化处理前，首先需要对计算流程进行详细的分类和分析，目的在于通过其自身的指令特点，找到适合于并行执行的部分。其次，对于不适合并行的部分，还需要分析计算步前后计算过程间的数据依靠关系，判断该部分的串行执行是在主机还是设备上进行，因为，为了减少不必要的数据传输，应该将小规模的串行执行直接放在设备上执行，这样有利于建立合理高效的并行计算程序架构。

框 3.2 单元计算基本流程

1. 初始节点内力 $\boldsymbol{f}_{\text{int}}^{n} = 0$

2. 计算当前的总体节点外力 $\boldsymbol{f}_{ext}^{n}$

3. 对所有的单元循环，执行相同计算：

(1)集合当前单元节点位移和速度

(2)对积分点循环：

a. 如果 n=0，转到 d

b. 计算变形的度量：$\boldsymbol{D}^{n-\frac{1}{2}}(\xi_Q)$，$\boldsymbol{F}^{n}(\xi_Q)$，$\boldsymbol{E}^{n}(\xi_Q)$

c. 通过本构方程计算应力 $\sigma^{n}(\xi_Q)$

d. $\boldsymbol{f}_{e}^{\text{int}} = \boldsymbol{f}_{e}^{\text{int}} + \boldsymbol{B}^{T}\sigma^{n}\varpi_Q J|_{\xi_Q}$

结束积分点循环

4. 离散单元内力 $\boldsymbol{f}_{e}^{\text{int}}$ 到内力节点 $\boldsymbol{f}_{\text{int}}^{n}$

5. 结束单元循环

6. 计算节点速度：$v_{n+\frac{1}{2}} = v_n + (t_{n+\frac{1}{2}} - t_n)a_n$.

7. 加载边界条件

8. 计算作用力 $\boldsymbol{F}^{n} = \boldsymbol{f}_{ext}^{n} - \boldsymbol{f}_{\text{int}}^{n}$

3.1.4 显式算法的并行性分析

显式算法中每个单元都有具有计算独立性，因而形成了很好的天然并行性。本文引用计算无关性[168]的概念来评价一个算法的可并行性。计算无关性的数学评价公式为

$$y \leftarrow x_1 \circ x_2 \tag{3.47}$$

式中，x_1和x_2分别表示输入元，即程序函数入口中的输入变量，y代表输出元，即函数的返回值。符号∘指计算元，用于代表某种计算关系、逻辑运算或者是函数主体。根据式(3.47)，首先对计算独立性进行约定：如果由多个计算元组的计算组是独立的，那么这组中每个计算元的输出元都是单一的；反之，如果计算元组中一个计算元的输入元为另一个计算元的输出元则这个计算组是相关的。如果一个算法包含大量具有计算独立性的计算，则称该算法具有内在并行性。

对框 3.1 的分析可知，显式有限元算法主要包括节点量计算和单元量计算两大部分，以位移、速度等节点物理量为例，假设一个节点从计算加速度、计算速度到计算位移为一个计算元，则由全体节点对应的计算元形成的计算组具有很好的内在并行性，如式(3.48)所示。

$$\boldsymbol{x}:(y_1, y_2 \ldots y_N) \leftarrow \chi(x_1, x_2 \ldots x_N) \tag{3.48}$$

式中，输出元$\boldsymbol{x}$为节点更新后的位移，N为节点整数。输入元x_1主要包括节点质量和节点力，算子χ为框 3.1 中第 3、5、6 步对应的计算公式。参照以上的分析方法，对于框 3.2 中单元计算过程，对所有单元间的循环也可以完全并行展开，以一个单元内形成单元内力的计算为计算元进行并行化计算。

由于节点和单元已经是有限元模型的基本构成，粒度很小，所以显式算法的计算无关性是一种细粒度的无关性，具有很好的细粒度并行性。显式有限元的细粒度并行性可以很好的适用于 GPU 轻量级的线程并行计算。在本文研究的开始阶段，Elsen[169]等利用 GPU 对流体模拟中纳维-斯托克斯(N-S)方程进行了显式求解。王建华[170]等采用早期的 GPGPU 技术实现了简单弹性问题在 GPU 上的并行计算，但是编程复杂，取的加速比也有限，而且由于 Cg 语言的限制，大部分的计算核心代码只能采用单精度浮点数进行，所以计算精度有限。在医学领域中，Joldes[171]等在 GPU 上采用显式方法实验了脑组织变形的实时模拟效果。总的来说，已有的这些利用 GPU 加速的算法大多数是基于早期 GPGPU 的计算平台，实现方式复杂，取得的加速比有限，且对计算精度的关注度不高。针对现有研究成果中的不足，本章的研究目标是在 CUDA 架构下建立适应于 GPU 计算的显式有限元并行计算策略，形成相应的分析程序，并应用于板壳结构的非线性有限元分析。

3.2 基于 CUDA 的显式有限元并行化策略

3.2.1 计算对象与线程间的映射策略

如上文所述，显式有限元大部分计算流程以单元或节点为计算单位，可以分为单元串行计算模式、单元并行计算模式、节点串行计算模式、节点并行计算模式、自由度串行计算模式、自由度并行计算模式等六种计算模式。在 GPU 中执行时，将计算单位和线程一一对应是一种可行且高效的并行策略，针对计算模式的不同，共有三种情况：一种是采用线程与单元对应的方式，称为 TFE(Thread For Element)模式；一种是采用线程与节点对应的计算方式，称为 TFN(Thread For Node)模式，以及线程与自由度对应的计算模式，称为 TFD(Thread For Degree)。图 3.4 为基于一维线程编号的 TFE 模式示意图。所谓一维线程编程是指 CUDA 线程的三级层次均以一维形式来索引，图 3.4 中，$block(x)$ 为线程块的索引，CUDA 程序中通过内建变量 blockId.x 表示；$t(y)$ 为线程在线程块内的编程，在 CUDA 由内建变量 threadIdx.x 表示；[x]表示单元编号。参考框 2.1 中二维线程索引的计算方式，一维 CUDA 线程索引号 T_n 的计算公式为

$$T_n = block(x) * gridDim + t(y) \tag{3.49}$$

式中，$gridDim$ 代表线程块总数，以第[257]号单元例，该单元由第 257 号线程计算，线程计算时，式中的 $x = 1$，$y = 2$。

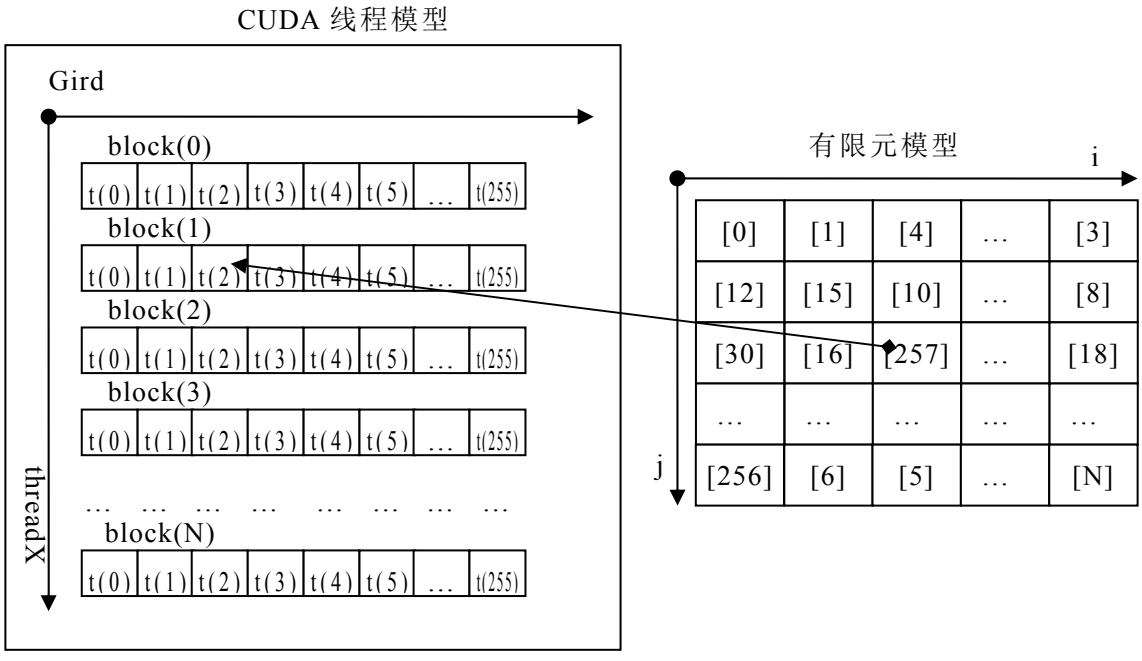


图 3.4 单元与线程间的一一映射关系

采用以上三种并行策略计算时，首先需要建立和计算对象数量一样的线程数，因此在执行 kernel 函数前，需要根据计算对象的数量 *numElements* 计算得到 CUDA 执行时线程块的数量，分两步计算：首先，需要确定每个线程块内的线程数 *numThreads*，根

据 CUDA 架构采用的 half-warp 执行模式，*numThreads* 的取值最好为 16 的倍数，但也不能超过硬件所容许的最大数目，图 3.4 中的示例每个线程块采用 256 个线程。然后，根据块内线程数计算出数程块的个数 *numBlocks*，数程块数必须为整数，计算出现小数时需要向上取整，计算代码如框 3.3 所示。

框 3.3 线程数及线程块数的确定

```
// 块内线程数
const unsigned int numThreads=256;
// 线程块数
const unsigned int numBlocks=(numElements+numThreads-1)/numThreads;
```

以上代码一般出现在 kernel 函数的执行配置阶段，参见本文的 2.3.2 节。由于 GPU 计算完全依赖于显卡上已经固定了的集成硬件资源，因此，合理的对硬件资源进行分配，可以保证计算程序效率的最大化，具体措施本文不再仔细描述，可以参考 CUDA 的编程手册《NVIDIA CUDA C Programming Guide》[172]。

3.2.2 一维数据存储模式

如本文 2.2 节所述，由于 CPU 强大的逻辑控制能力和较大的缓存空间使其可以高效的进行不规则的内存访问，因而常用的有限元分析等通用计算程序可以直接采用二维数组或多维数组存储数据，以节点位移的存储为例，对于一个包含 32 个 6 自由度节点的有限元模型，可以建立维度大小为[6][32]的二维数组存储。然而，采用 GPU 计算时，为了顾及到并行计算时进行并行内存访问时的吞吐量，现阶段 CUDA 对二维及以上维度数组的支持较差。为了实现数组访问，CUDA 提供了 *cudaMallocPitch()*，*cudaMalloc2D()*和 *cudaMalloc3D()*等函数可以采用自动对齐的方式在全局存储器中实现二维和三维数组的操作，如框 3.4 所示。

框 3.4 线程数及线程块数的确定

```
// 分配空间
cudaMallocPitch(&devPtr, &pitch, width * sizeof(float), height);
// 内存访问方式
__global__ void Kernel(float* devPtr, size_t pitch, int width, int height)
{ for (int r = 0; r < height; ++r) {
        float* row = (float*)((char*)devPtr + r * pitch);
        for (int c = 0; c < width; ++c) {
            float element = row[c];
    } } }
```

但是，基于这些函数分配的多维数组，在 *kernel* 函数中使用时每次访问都要

计算其起始地址等，程序代码复杂，不易维护，因而使用率不高。因此，本文采用一维数据存储模式，相对而言，这种存储方式的访问相对最为容易，并且可以与一维线程执行模型很好的对应。图 3.5 所示为在全局存储空间中以一维形式存储 6 自由度节点的速度和位移数据的示意图，图中，箭头指向为一个线程计算需要访问到的空间。通常情况下，我们为在显存空间中开辟的空间加上“_d”后缀，以便于编程时进行识别。

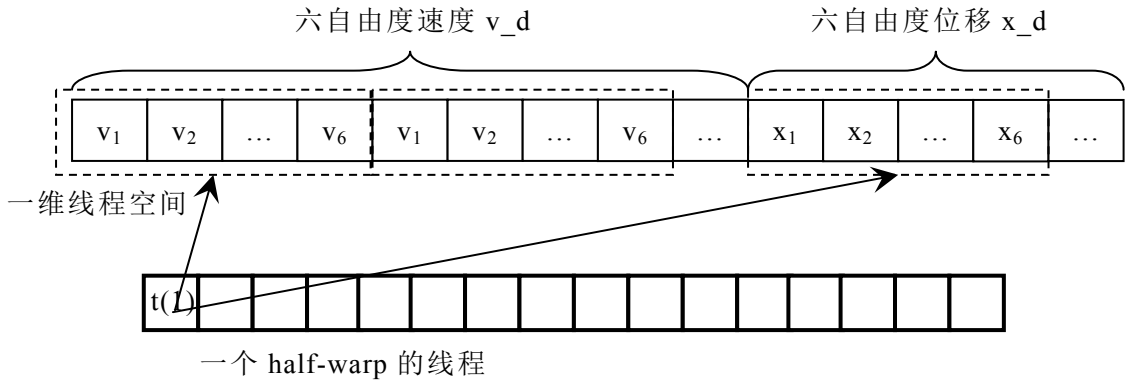


图 3.5 一维模式的存储和数据访问示意图

GPU 计算时，每个线程块内的线程首先按顺序按批分为不同的 SIMD 线程组，称为 warp 块，每个 warp 块包含 32 线程，但是，一个 warp 块只有一半的线程能同时运行，所以称为 half-warp 执行模式，并且每次并发执行的线程必须满足 SIMD 模式，当出现由于逻辑判断等操作出现不同指令时，不同指令间需要分批、串行执行。如果一个 half-warp 内的线程对全局内存访问的地址是连续的，就实现了对齐访问(Coalesced Access)，对齐访问是一种最优的全局内存访问模式，理论上可以达到最高的内存带宽。但是，保证程序中内存对齐访问会极大的增加程序代码的难度，降低代码的可读性和可维护性。幸运的是，对于计算能力 1.2 及以上的 GPU 已经能够自动完成全局存储器的对齐工作，计算能力 2.0 及以上的 GPU 引入的缓存机制可以有效降低全局存储器的乱序访问对程序性能的影响。因此，尽管图 3.4 中所示的内存访问一般情况并不能满足对齐访问的要求，但是对程序计算效率的影响有限。除了对齐访问可以优化 GPU 中内存访问外，还有如下两点因素对保证程序内存访问效率和整体计算效率是非常有利的：

(1) 充分应用共享内存空间。kernel 函数中，密集计算前，可以将计算所需的数据以线程块为单位从全局内存中读出并缓存于共享显存中，从而达到利用共享内存的极高的访问速度来提高计算效率，减少对全局内存的访问延迟。对共享显存访问也需要考虑到对齐性，由于共享内存的局部性，这种对齐访问一般是自然满足和容易实现的。

(2) 提高并发线程数和并发线程块数。在基于线程的并行计算模式下，提高

并行线程数是最直接地效率提升方法，而且，提高同一线程块中并发执行的线程数，可以有效地隐藏内存访问延迟。GPU 并行计算时，每个线程块可以使用的存储资源是固定的，这些资源由线程块内的所有线程共同使用，所以，当单个线程占用的存储资源过多时，并发执行的线程数就会减少。因此，应该合理进行各类存储空间的分配，使用共享存储器和寄存器要权衡内存访问效率提升和并发线程数提升对效率的影响，建立合理的并发线程数。在考虑提高并发线程数的同时，还要注意提高并行线程块数，因为，当线程块数为 GPU 中多处理器数的两倍以上时，可以使闲置和不闲置的线程块在时间上重叠，从而达到隐藏内存访问延迟的目的。

3.2.3 基于预索引策略的并行内力组装方法

显式算法中进行单元内力计算时，绝大部分单元技术是以单元为计算个体进行内力计算的，通过单元计算得到单元内力后再将单元内力离散到该单元所有节点中，形成节点内力，如框 3.2 所示。该离散过程是一种典型的离散-加(scatter-add)[173]操作，考虑到同一个单元有可能共享多个节点，当某一时刻这些单元同时将自己的单元内力离散并累加到该节点上时，即会引起并行“竞写”错误，这是显式算法并行化过程中的主要技术瓶颈之一[174, 175]。

为此，本文提出了一种名为“预索引”的预处理机制。通过该机制，单元力离散的过程由离散-加操作转换成为具有较高并行性的并行聚集(gather)操作，数学计算公式如下：

$$\boldsymbol{f}_j = \sum_{e=1}^{E} f_e^k \tag{3.50}$$

式中，$\boldsymbol{f}_j$ 代表节点力，$\boldsymbol{f}_e$ 代表单元力，E 代表所有包含节点 j 的单元，k 代表节点 j 是单元 e 的第几号节点。因此，“预索引”的主要含义是找到一个节点归哪些单元共有以及它在这些单元中的编号等索引信息。

该索引信息的建立流程如图 3.6 所示，以图中的 5 号节点为例，它由四个节点所有，分别为 1 号单元的 3 号节点，2 号单元的 2 号节点，3 号单元的 4 号节点以及 4 号单元的 1 号节点。这些索引信息由两个数组存储，一个数组以基于节点的形式存储该节点对应的单元节点力在单元力数组中的索引号，计算公式为：

$$I(x,y) = (x-1) * E_{node} + (y-1) \tag{3.51}$$

式中，E_{node} 为一个单元的节点数量，x 是以 1 为起始编号的单元号，y 是以 1 为起始编号的节点号。5 号节点对应的索引号分别为 2，5，11 和 12。另一个数组则用于以基于节点的形式存储索引号的个数，如 5 号节点共对应 4 个索引号。有了这些索引信息，节点力离散计算过程则可由 TFE 模式转换为基于节点的 TFN 并行计算模式，由一个线程对应一个节点的小循环，即将线程的计算粒度进行小的

提升，就可以根据索引号从存储单元节点力的全局数组中读取对应索引号位置上单元力并累加当前线程所对应的节点的节点力上。由于不同线程容许对同一个全局内存空间进行读取，从而避免了并行计算过程的“竞争”错误。

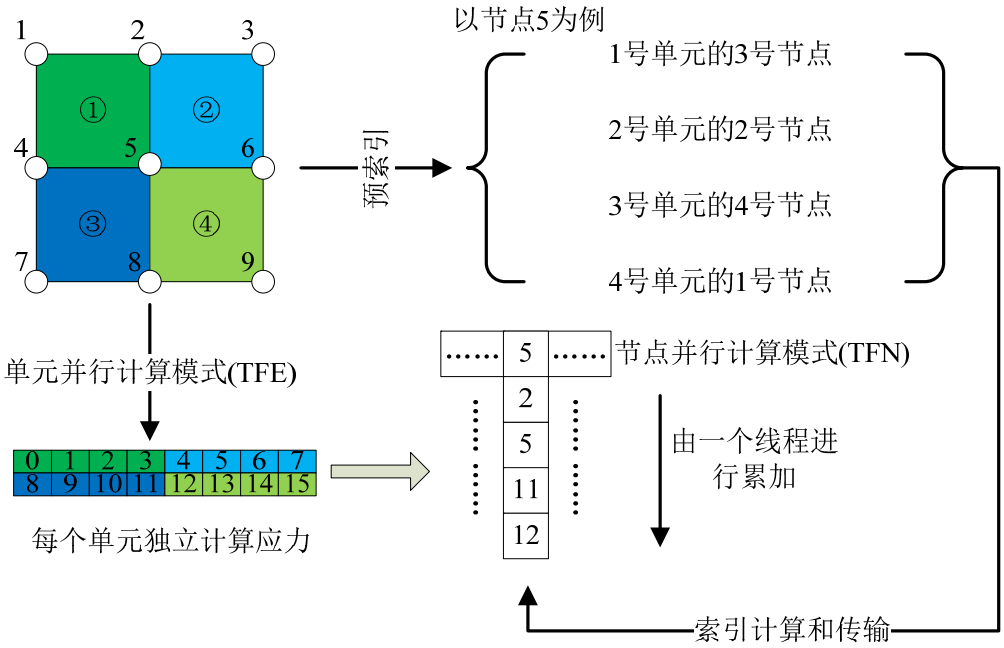


图 3.6 预索引机制

图 3.6 所示的预索引机制的程序代码如框 3.5 所示。对于较小规模的有限元模型该部分代码可以由 CPU 完成，然后将索引数组复制到 GPU 的全局显存中即可。对于大规模的有限元模型，由于索引信息的建立过程中涉及到循环次数分别为到单元总数 *gmeshl* 和节点总数 *gmdshl* 的两层嵌套循环，计算复杂度为 $O(n^2)$，耗时较长。为此，通过将外层循环展开的方法，实现了 GPU 中的并行索引信息搜寻，此时，计算复杂度缩减为 $O(n)$，计算时间有较大的减少。

框 3.5 预索引机制的程序代码

```
for(int j=0;j<gmeshl;j++){
        for(int k=0;k<4;k++){
            for(int i=0;i<gmdshl;i++){
                if( gix[k][j] == i){
                    gNum[i] = gNum[i]   + 1;
                    gNodeToElement[i*10 + gNum[i] -1] = j*4+k;
    }}}}
```

3.2.4 基于并行缩减策略的单值求解方法

单值是本文对计算过程中需要通过对数组的极值搜寻或对数组的求和等规约计算获取的数值的总称。以时间步长的计算为例，时间步长的串行计算方法为：首先，循环计算每个单元特征长度，找到最小的单元特征长度；然后根据式(3.12)计算时间步长。假设每个单元的特征长度为计算元，那么在单元之间寻找最小特征长度则属于完全计算相关，并行性很差。为此，本文提出采用并行缩减算法[176]进行单值求解，缩减算法是一种类似于二分搜索的计算方法，求解思路简单且适合于并行求解，如图 3.7 所示。基于 GPU 的并行缩减算法可以在 GPU 上实现数组极值和数组求和的快速并行化求解。

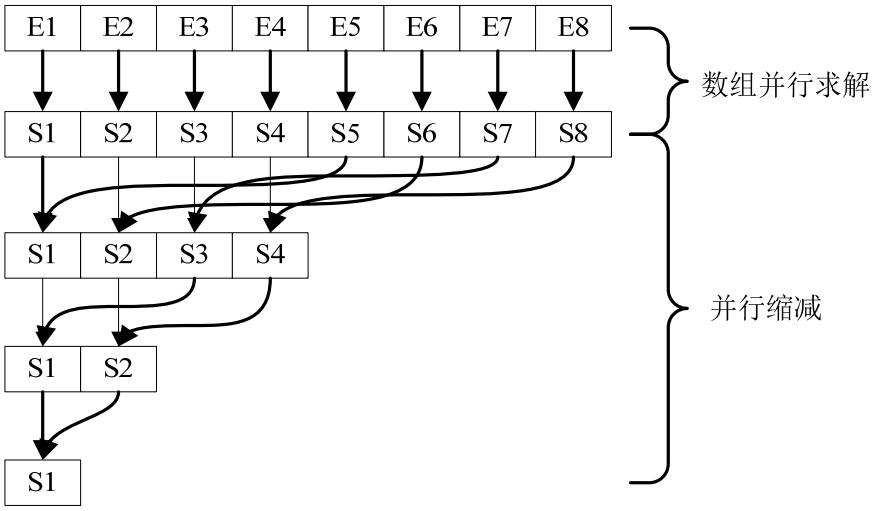


图 3.7 并行缩减算法

采用并行缩减算法进行时间步长求解时，可先采用 TFE 模式并行计算每个单元的特征长度，并存储到一个整体数组中，然后对这个数组采用并行缩减算法找到最小值，并传回到 CPU 中，最后，在 CPU 采用式(3.8)计算得到当前的时间步长。该算法具有很好的并行度，并且主机与设备仅需进行一个数值的传递，对计算效率的影响可以忽略。

3.2.5 并行程序整体流程设计

如果一个计算程序中，除了数据文件的读入和写出外，其它涉及数值求解的过程都可以在 GPU 上并行执行，那么，本文称这个程序具有完全 GPU 并行化计算能力。通过 3.2.1 节到 3.2.2 节的研究和分析可知，采用以上的并行化策略，显式有限元算法可以实现完全 GPU 并行化计算。本文开发的显式有限元并行分析程序的流程如图 3.8 所示，程序共分为数据准备、GPU 并行计算、后处理 3 个部分。程序设计坚持“更多并行，更少数据传输，更快加速比”的思想，由 CPU 负责

GPU 设备管理，调用 GPU 进行数值求解以及进行程序收敛判断等程序流程控制。GPU 则作为辅助处理器，完成所有数值计算，如图中所示的各类 kernel 函数。这种 CPU 处理器为主，GPU 为辅的程序符合 GPU/CPU 异构平台，具有很高的计算效率。

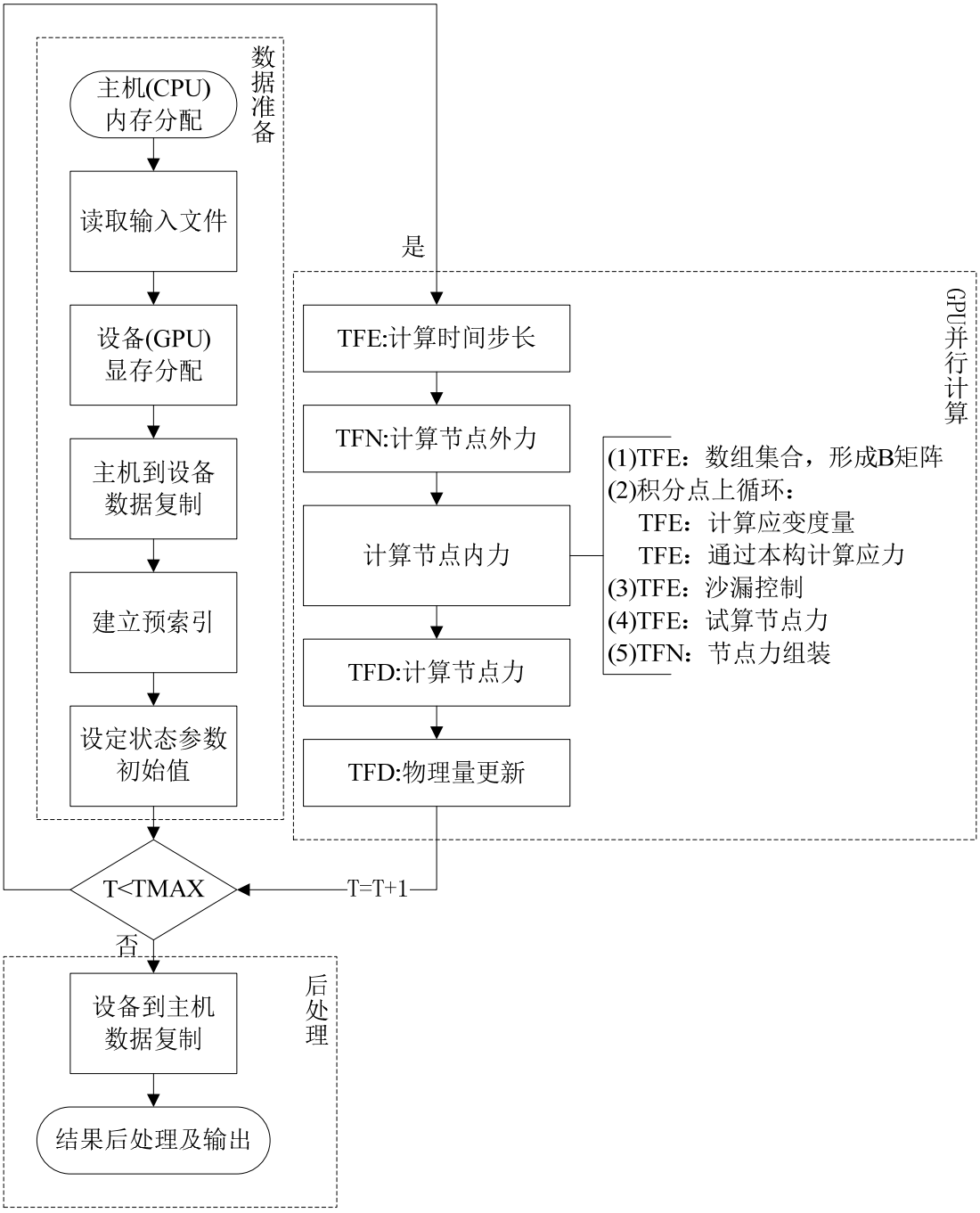


图 3.8 显式有限元并行分析程序流程图

3.2.6 EST 壳元并行的特殊化处理

以上对并行化策略的描述是以 BT 单元为对象进行的，这类单元共同的特点

是积分域和几何域相同，因此，单元计算过程相对简单，也无需采用特殊的并行计算策略。但是，EST 单元属于边光滑单元，应力计算的积分是在由单元边构成的边光滑域中进行的，因而积分计算时是以边作为最小计算单元。为此，考虑到基本边的积分计算的本质与在单元域相同,同样具有很好的细粒度并行性,为 EST 壳元需要引入另外一种细粒度并行策略，线程与边的一一映射关系，称为 TFG(Thread For edGe)，即计算过程中由一个线程完成一个边域的计算，同样满足细粒度并行的要求。

此外，EST 壳元计算过程中应力的离散也同样需要采用预索引策略，只是，与 BT 单元不相同的是，EST 单元需要建立的预索引信息不再是单元与节点间从属关系，而是节点与边之间的索引信息。应力计算时，首先采用 TFG 策略计算得到“边应力”，再根据边与点间的索引信息，采用 TFN 的策略由节点去读取属于自己的“边应力”分量并累加。

通过，本文 3.1.2 小节的理论分析可知，单个 EST 单元计算的计算复杂度比 BT 单元要高，同时，基于边光滑的策略也会带到并行计算的难点。这是因为，在边域积分时,由于边通常由多个单元共享,而且每条边共享单元的个数也不同。对于这类不等于且次数不明确的两层嵌套循环，通常是无法完全展开执行进行细粒度并行的，即使采用 CUDA 的二维线程执行模式也无法完全发挥出 GPU 计算能力。本文采用的策略是将并行粒度加粗，对于边域计算这样里外层循环次数差异明显的算法，加粗的过程即将循环的外层展开，进行 TFG 策略的执行，里层则在线程内进行循环。由于内层循环很小，因此，这样加粗其实是很小的，其本可以忽略，仍然满足细粒度并行的特性，因此，相比于 CPU 串行计算，计算效率还是会有比较明显的提升。

3.3 数值算例及分析

本文采用球壳模型[177]和受冲击载荷圆柱板模型[53]对本章研究的显式有限元并行计算方法进行验证，这两个计算模型是有限元算法理论中常用的算例，可以有效地检测显式壳单元计算精度和计算效率。测试采用的计算平台的硬件及编译环境如表 3.1 所示。

表 3.1 计算平台

名称	型号	主要性能参数
CPU	Intel Core I7-930	主频：2.80GHz，6GB 内存
GPU	NVIDIA GTX 580	512 个 CUDA 核，核心频率:832Mhz，3Gb 的显存
操作系统	Windows 7 64 位	
开发环境	Microsoft VC++ 2010, CUDA 5.0	

3.3.1 数值算例

3.3.1.1 球壳问题

首先采用球壳模型对弹性本构下本文提出算法的计算精度和计算效率进行评估。计算模型如图 3.9 所示，在球面对称边和自由边的交点处分别加载大小为 1.0N 的集中力。模型的几何参数为：半径 R 为 10.0m，厚度 t 为 0.04m。材料参数为：杨氏模量 E=6.825×10^7 N/m^2，泊松比 v=0.3。

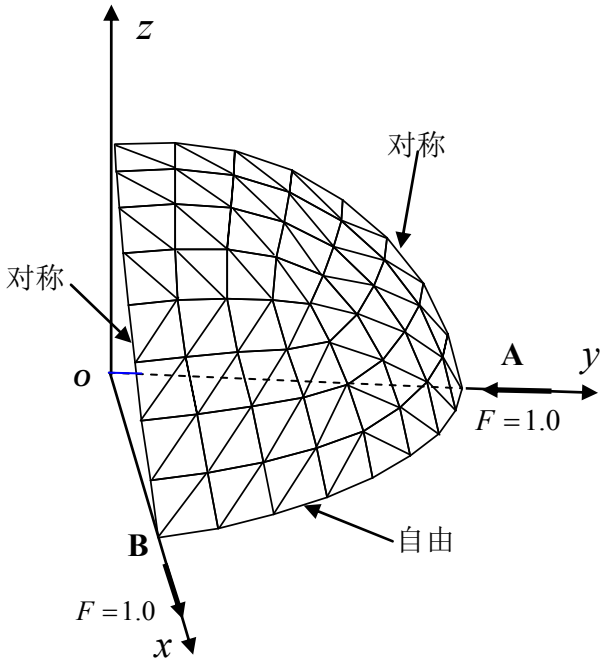


图 3.9 球壳模型的几何模型描述

为了同时对比 BT 四边形单元和 EST 三角形单元的计算效率和计算精度，对该算例分别采用 BT 四边形单元和 EST 三角形单元进行建模，并分别在 GPU 和 CPU 中进行计算。计算后，可以得到如图 3.10 所示的 CPU 和 GPU 计算的加载点 B 点的位移动态响应曲线。由图 3.10 可以得到两点结论：

(1) 无论是采用 BT 单元还是 EST 单元，GPU 并行计算的动态响应曲线和 GPU 计算的动态响应曲线都十分吻合，说明 GPU 的计算结果与 CPU 完全一致。

(2) 在网格密度大致相同的条件下，EST 三角形单元的计算结果要好于 BT 四边形单元，这符合 EST 单元低阶、高精度的特点，说明了 EST 三角形进行大规模复杂模型计算时的优势。当然，对 EST 单元计算精度的探讨，并不是本文的研究重点，崔向阳[139]已经从理论角度进行了详细的说明。

另外，为了评价并行算法的计算效率，分析计算规模与计算效率间的关系，本文对该球壳模型采用不同大小的网格进行建模和计算。设定仿真终止时间为 0.05 秒时两种单元的计算效率和计算加速比如表 3.2 和图 3.11 所示。

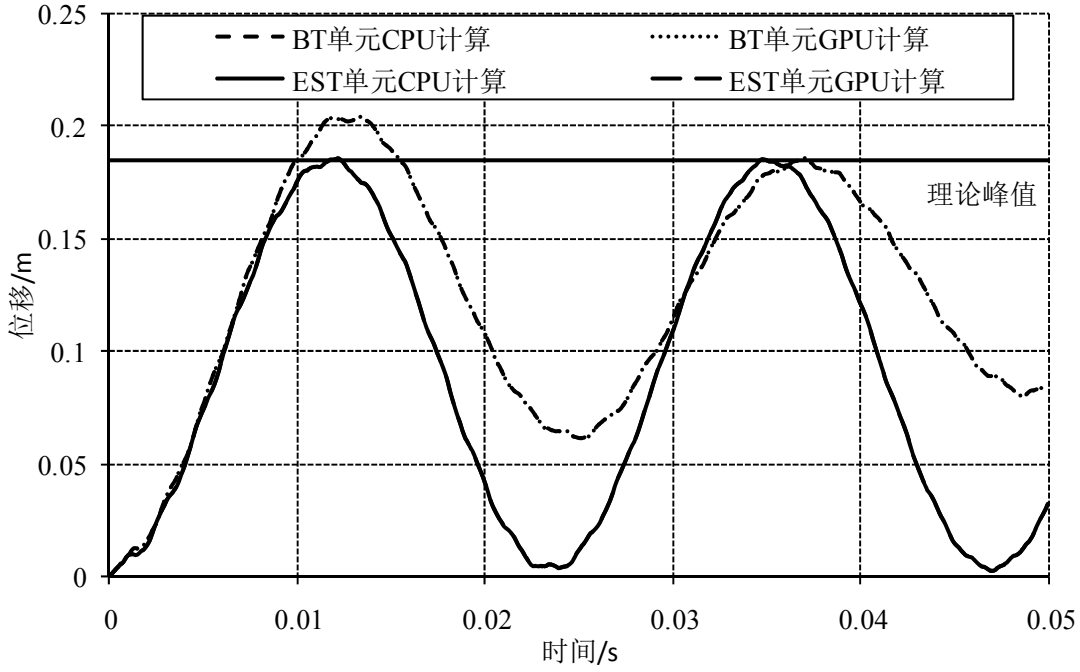


图 3.10 球壳模型的位移动态响应对比

表 3.2 所示为采用 BT 四边形单元计算的各个规模有限元模型的结果，给出对应规模下 CPU 和 GPU 计算单个迭代步的平均计算时间以及 GPU 计算可以取得的加速比。由表可知，在 GPU 和 CPU 计算所需要的迭代步数完全一致前提下，GPU 的单个迭代步的平均计算时间远少于 CPU 计算所需，因此，GPU 的计算效率要好于 CPU。通过对比两者的总计算时间，可以得到 GPU 执行的计算加速比，例如，对于近 500 万个自由度的模型，可以取得近 28 倍的计算加速比，加速效果明显。

表 3.2 球壳模型采用 BT 单元的计算时间及加速比

序号	有限元模型			单个迭代步计算时间/s		加速比
	单元数	节点数	自由度数	CPU	GPU	
1	3072	3169	19014	0.00742	0.00131	5.65
2	12288	12481	74886	0.03213	0.00283	11.33
3	49152	49537	297222	0.13414	0.00821	16.35
4	196608	197377	1184262	0.69613	0.02798	24.88
5	786432	787969	4727814	3.52979	0.12683	27.83

图 3.11 所示为采用 EST 三角形壳单元计算时，GPU 单个迭代步的平均计算时间以及总程序执行时间加速比随自由度变化的曲线。可以看出，对于 EST 三角形单元，GPU 同样可以有效的减少计算时间，并取得较好的计算加速比。例如，对于近 500 万个自由度的计算模型，GPU 计算可以取得 30 倍左右的加速比，加

速比略高于 BT 单元。

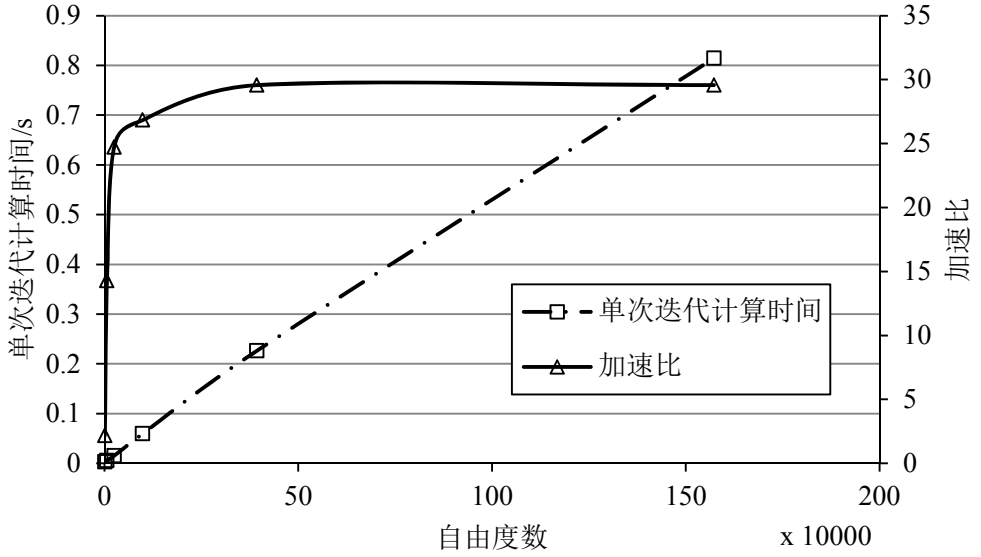


图 3.11 球壳模型采用 EST 单元的计算时间及加速比

3.3.1.2 受冲击载荷的圆柱板问题

采用弹塑性材料的圆柱板受到冲击载荷后会发生弹塑性大变形，采用此算例，可以检验本章所提并行算法对求解此类弹塑性大变形问题的精度和效率。模型的几何形态和几何参数如图 3.12 所示，在长为 12.56in，开角为 120° 的圆柱板上，部分区域受到大小为 5650in/s 的法向冲击载荷。具体材料参数为：杨氏模量 E=1.05×10^{7}psi，密度 ρ=2.5×10^{-4} psi×s^2/in^2，泊松比 v=0.3，屈服应力 σ=44000psi，弹性模量 E_p=0。程序中，采用 von Mises 本构进行计算。

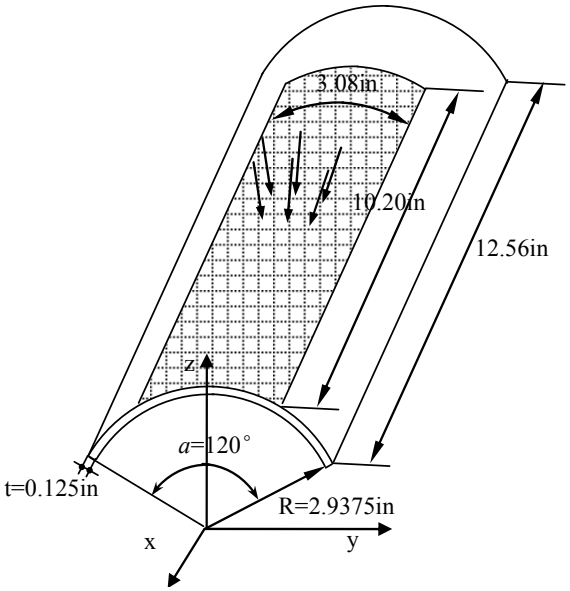
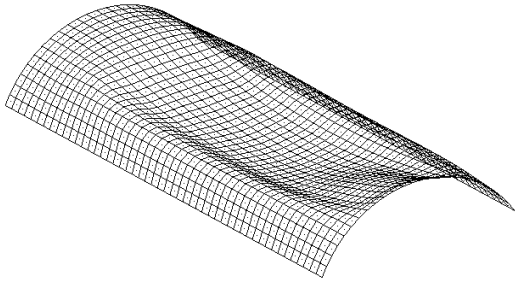


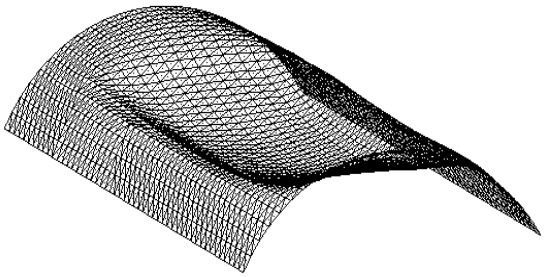
图 3.12 圆柱板模型

对本算例，同样分别采用 BT 单元和 EST 单元在 CPU 和 GPU 上进行计算。

得到的圆柱板最终弹塑性变形结果如图 3.13 所示。图 3.14 所示为两种单元采用 CPU 和 GPU 计算得到的圆柱板对称线中点的位移动态响应曲线。由图可知，CPU 和 GPU 的结果曲线完全重合，因此，可以证明本文所提并行计算方法对于弹塑性大变形问题同样具有和 CPU 完全一样的计算精度。同时，对于此算例，EST 单元仿真结果同样比 BT 单元更接近于实验结果。



(a) BT 单元结果



(b) EST 单元结果

图 3.13 圆柱板的弹塑形变形结果

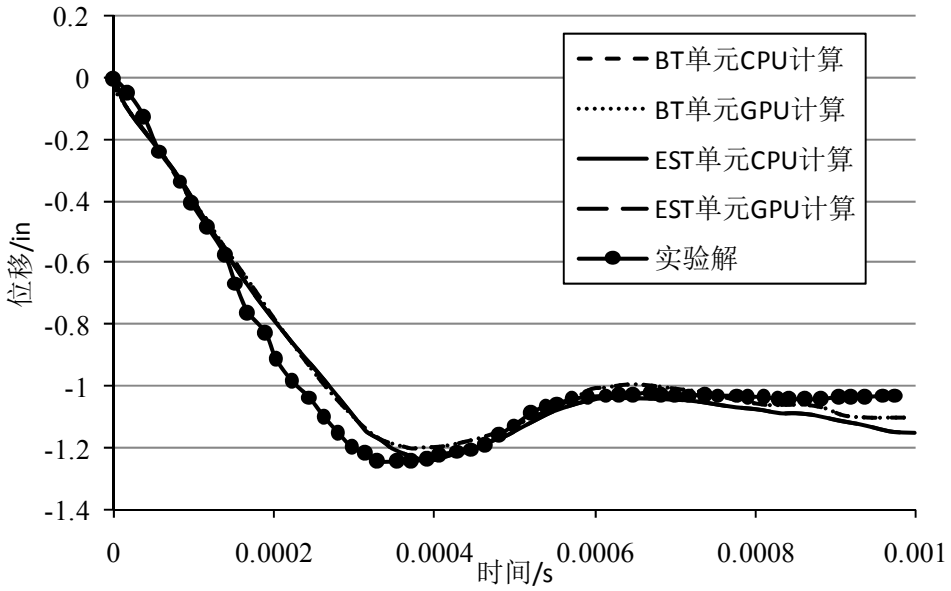


图 3.14 对称面与圆柱板交线中点的位移动态响应曲线对比

同样采用不同规模的计算模型来检验 GPU 的计算效率，计算模型的规模跨度从数千自由度到数百万自由度。首先，表 3.3 给出了圆柱板模型在不同计算规模下，采用 BT 四边形单元计算时，单个迭代步的平均计算时间和程序总计算时间的加速比。为了对比 GPU 与不同 CPU 对比的计算效率，在这里，加入了 Q6600 的计算时间，Q6600 是 CPU 发展历史中很有代表性的双核处理器，由表可以看出，Q6600 的计算性能与 GTX580 差距更加明显，最高可以取得 44 倍的计算加速比，与 I7 GPU 对比，对于近 200 万自由度的模型，加速比也可以达到 27 倍左右。同时，通过对比表 3.2 中弹性问题的计算效率可以发现，在相同自由度下，弹塑性问题的计算加速比要高于弹性问题的计算加速比。能过本文的实验发现，这是一个一般性的规律，即在并行性基本相同的情况下，复杂度高的计算往往可以取得更高的计算加速比。

表 3.3 圆柱板模型采用 BT 单元的计算时间及加速比

序	有限元模型信息			单个迭代步计算时间			加速比	
号	单元数	节点数	自由度数	Q6600	I7	GTX 580	Q6600	I7
1	1200	1275	7650	0.01173	0.00660	0.00217	5.42	3.05
2	4800	4949	29694	0.04668	0.02603	0.00246	18.97	10.58
3	19200	19497	116982	0.17215	0.10710	0.00573	30.02	18.68
4	76800	77393	464358	0.67858	0.42311	0.01586	42.77	26.67
5	307200	308385	1850310	2.61416	1.58434	0.05899	44.32	26.86

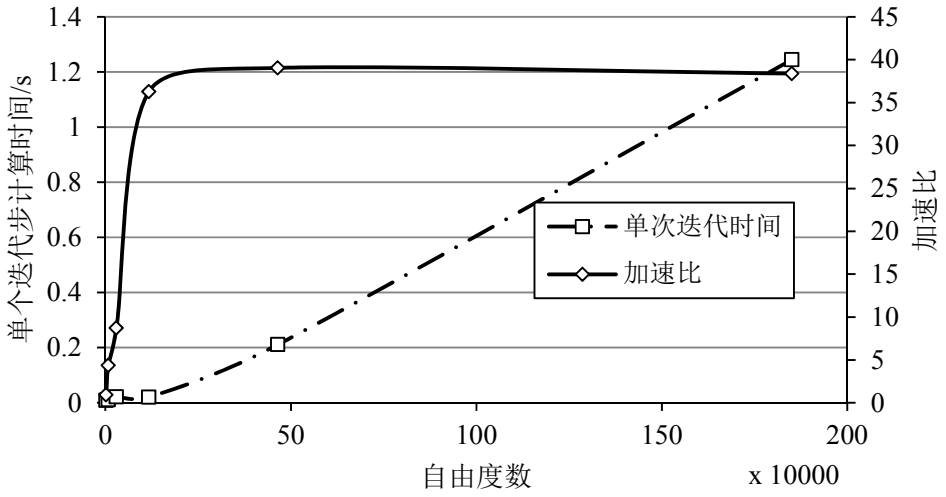


图 3.15 不同规模模型下 BT 和 EST 单元的加速比

图 3.15 所示为采用 EST 三角边单元计算时，计算时间和加速比的对比，可以看到，对于近 200 万自由度的模型，可以取得近 40 倍的计算加速比。由此可知，

对于弹塑性模型，EST 三角形单元的计算加速比仍高于 BT 四边形单元，这与弹性问题一致。同时，相同规模下，含弹塑性本构的 EST 单元的计算加速度也高于弹性本构的计算加速比。

同时，我们也发现，虽然 EST 单元的加速比要高于 BT 单元，在自由度数相同的情况下，EST 单元单个迭代步的平均计算时间却要高于 BT 单元。通过分析，我们认为这主要是两方面的因素造成的。一方面，是因为相同自由度下，三角形单元的数目要多于四边形网格，例如，在此算例中，同样的自由度下，三角形的单元数为四边形的两倍。另一方面，如本文 3.2.6 小节中的分析，由于边域的引入， EST 单元的计算复杂度的确是要高于 BT 单元。但是，考虑到 EST 三角形单元在网格划分上的巨大优势，以及 EST 单元往往在少数自由度的前提下，就可以达到和多数自由度 BT 单元同样的自由度一样的计算结果。因此，在本文下阶段进行车身结构碰撞仿真和金属板料成形仿真的研究中，仍然会加入 EST 单元的应用研究。

3.3.2 数值结果分析

由以上两个算例的计算结果可知，本文提出的显式并行算法可以有效地在 GPU 通用计算平台上对板壳单元和板壳结构非线性问题进行求解。首先，在相同计算模型下，GPU 并行计算的结果与 CPU 串行计算的结果完全一致，因此，在证明了该并行算法无误的同时，也证明了 GPU 通用计算技术在有限元分析中的可行性。其次，采用 GPU 并行计算时，相对于 CPU 串行计算模式，计算效率有十分明显的提高，这说明了本章所提出的显式有限元细粒度并行执行策略可以很好适用于 GPU 执行。而且，弹塑性本构的计算加速比要高于弹性本构的加速比，EST 单元的加速比要高于 BT 单元，这说明，复杂度越高的算法采用 GPU 加速比效果更明显。再次，我们发现，GPU 计算的加速比在一定区间内与计算规模成正比关系，但是，这种正比关系并不是一直存在的，如对表 3.2 中球壳模型的计算时间分析，当网格数超过 5 万个时，加速比不再增加，而趋于稳定。经过我们的研究发现，这种计算加速比先增长后趋于平稳的特性是符合 GPU 的硬件特性的，因为，GPU 中用于并行计算的 CUDA 核心以及可并发执行的线程数是有限的，当有限元模型增长时，并发执行线程数出逐渐达到所容许的最大值，这时加速比将不会增加，如下式

$$\lim_{m\to N}\delta\frac{nO(e)}{nO(e)/m}=\delta N,\quad \delta=(0,1) \tag{3.52}$$

式中 n 为总单元数，$O(e)$ 为单个单元的计算复杂度，m 为当前并发执行的线程线，N 为容许的并发执行的最大线程数。δ 是由 GPU 和 CPU 间的计算频率差异和 GPU 计算额外操作等因素决定的一个因子，取值范围为 0 到 1 之间。

3.4 通用显式有限元的 GPU 并行计算平台

通过对本章研究内容的总结，可以得到一个具有通用属性的显式有限元 GPU 并行计算平台。该计算平台的硬件为含支持 CUDA 计算的显卡的个人计算机。软件平台的基本架构可以从图 3.7 中提取，架构包括计算前处理、内存操作、CPU 迭代流程控制和并行计算核心。如图 3.16 所示，该架构为一种典型的异构执行架构，在显式迭代过程中，GPU 作为并行计算核心的执行设备，而由 CPU 进行迭代流程控制，其中包括中间结果和最终结果的传回控制。

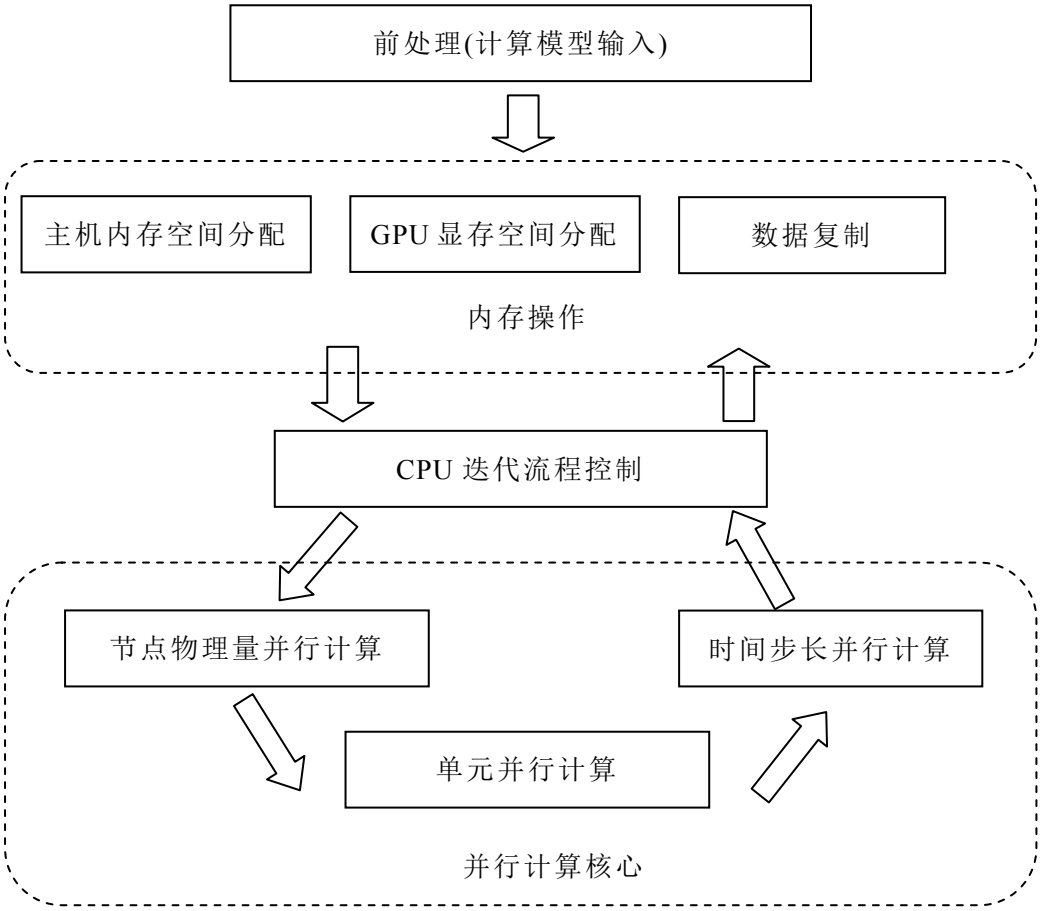


图 3.16 通用显式有限元的 GPU 并行计算程序架构

通过与传统计算平台的对比可知，该采用 GPU 进行计算的平台最大的好处在于排除了有限元网络分区系统以及计算节点通讯两大部分，这两大部分往往是并行计算性能的瓶颈所在。对该 GPU 并行计算平台和相关的并行计算方法已经申请了国家发明专利，专利名称为“基于 GPU 的有限元显式并行求解仿真方法”，受理号为 201210266435.1，目前处于实质审查阶段。

3.5 本章小结

本章首先对板壳结构非线性有限元分析显式有限元方法和显式算法求解非线性的程序流程进行了介绍，并引用计算无关性的概念对显式算法和单元计算的可并行性进行了分析，证明了它们具有很好的天然并行性。在这些理论基础上，本章提出了 CUDA 架构下适应于 GPU 执行的显式有限元并行算法以及单元并行算法，具体并行策略如下：

(1) 针对显式有限元算法中的计算独立性，提出了 TFE、TFN、TFD 和 TFG 四种计算对象与 CUDA 线程间的一一映射模式的基本细粒度并行策略，实现了有限元算法中绝大部分计算流程的细粒度并行，并且可以很好适用于 GPU 基于 SIMD 的轻量化并行计算模式。同基于网格分区的粗粒度有限元并行策略相比，该细粒度并行策略没有任何前处理过程，在单块显卡也不存在边界数据处理问题，能够大幅度提升计算效率。

(2) 针对单元内力离散过程中，离散加操作有可能会引发并行“竞写”错误，提出了基于预索引策略的内力并行组装方法。

(3) 提出了基于并行缩减策略的并行单值求解方法。

(4) 针对 EST 三角形单元中采用边光滑域进行积分的特点，对 EST 三角形单元并行计算所需的特殊化处理进行了详细的说明。

在以上并行策略的基础上，建立了适合于 GPU 计算的整体程序流程，并形成了并行计算程序。通过数值算例表明，采用 GPU 加速的显式算法可以取得与 CPU 完全一样的计算结果，因此，证明了 GPU 通用计算技术的计算精度可以满足有限元计算的要求。同时，通过不同规模下，CPU 和 GPU 计算时间的对比，证明了本章提出的显式并行算法可以很好的适用于 GPU 执行。采用 GTX580 进行弹性问题分析时，BT 四边形单元可以获得近 28 倍的计算加速比，EST 三角形单元可以获得近 30 倍的计算加速比。同样，采用 GTX580 进行弹塑性问题分析时，可以获得更高的加速比，BT 单元和 EST 单元可以获得计算加速比分别为 27 倍和 40 倍。

最后，通过并行计算程序流程的分析，建立了具有通用属性的显式有限元 GPU 并行计算异构平台，相比于传统并行计算方法，该平台剔除了网格分区，计算节点通讯等部分，具有很好的实用性和较高的计算效率。

总的来说，本章的主要贡献是，在极低的硬件成本上，实现了壳单元显式有限元算法的快速计算方法，为有限元工程应用中板壳结构非线性问题分析提供了一种简单、高效的新途径。本章研究成果已经申请了国家发明专利，专利名称：基于 GPU 的有限元显式并行求解仿真方法，受理号：201210266435.1，目前处于实质审查阶段。

第 4 章 基于 GPU 的车身结构碰撞过程并行计算方法

车身结构碰撞过程有限元分析是汽车 CAE 的重要组成部分，是汽车结构设计和车身设计过程中重要的有限元依据。接触碰撞界面非线性是接触碰撞问题中除材料非线性和几何非线性外，第三个重要的非线性特征。在脱离外界载荷的影响下，接触变形的作用力唯一来源于接触体间接触和摩擦作用。接触碰撞界面计算主要就是指根据接触算法找到接触碰撞体系中发生接触的接触对，然后计算出计算接触力，最后进行接触约束，从而形成一个完整的、符合实际规则的接触碰撞过程。在现代碰撞仿真分析中，接触算法的时间通常会占用到 70%左右的计算时间，因此，本章在第三章的研究基础上，主要研究显式算法中接触碰撞界面的 GPU 并行计算方法，包括基于 GPU 的并行级域接触搜寻算法以及并行罚函数法、并行防御节点法两种并行接触力计算方法。最终，构成了全流程 GPU 执行的细粒度并行接触算法。本章的研究基于自主开发的碰撞仿真计算程序 DYSI3D，在建立完整并行算法后，开发了基于 GPU 的碰撞过程计算机仿真并行计算软件 CPS-GPU(软件著作权编号：2011SR001966)。

4.1 接触碰撞界面的有限元模型

4.1.1 接触体中接触界面模型

一个由两个接触物体 A 和 B 构成的接触系统如图 4.1 所示。

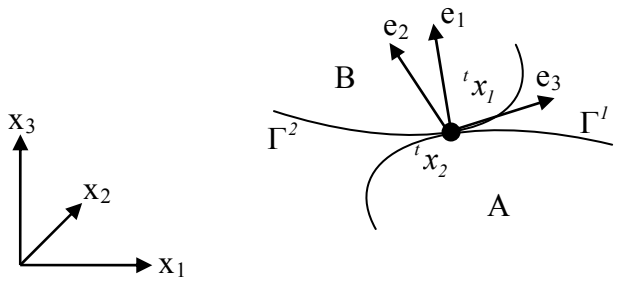


图 4.1 接触体系

接触体系中不同接触体在接触碰撞界面上相互接触的质点应该满足如下三个接触约束条件[178]。

(1) 运动学约束条件：在某一时刻 t，接触体 B 的边界 Γ^2 上，点 ${}^t x^2$ 处的单位矢量 $\boldsymbol{e}_1$ 和单位切向量 $\boldsymbol{e}_2$、$\boldsymbol{e}_3$ 组成的局部坐标系如图 4.1 所示，其中 $\boldsymbol{e}_1 = \boldsymbol{e}_2 \times \boldsymbol{e}_3$。点 ${}^t x^2$ 为接触体 A 的边界 Γ^1 上的点 ${}^t x^1$ 在边界 Γ^2 上的正交投影，在 t 时刻两接触体相互接触，则在两接触体之间的接触界面上 ${}^t x^1$ 和 ${}^t x^2$ 重合，即

$$ {}^{t}x^{1} = {}^{t}x^{2} \tag{4.1} $$

用 ${}^{t}x^{1}$ 相对于 ${}^{t}x^{2}$ 沿 $\boldsymbol{e}_1$ 方向的穿透量 ${}^{t}g_{e1}$ 表示接触体发生接触时的运动学约束条件，即互不穿透条件

$$ {}^{t}g_{e1} = ({}^{t}x^{1} - {}^{t}x^{2}) \cdot \boldsymbol{e}_1 \tag{4.2} $$

(2) 动力学约束条件：将接触体系中第 n 个接触体的点 ${}^{t}x^{n}$ 处的面积上的接触面力记为 ${}^{t}p^{n,c}$，设 ${}^{t}p_{ei}^{1,c}$ 为 ${}^{t}p^{1,c}$ 在 $\boldsymbol{e}_i$ 方向的分量，

$$ {}^{t}p_{ei}^{1,c} = {}^{t}p^{1,c} \cdot \boldsymbol{e}_i \tag{4.3} $$

如不考虑两接触质点之间的粘附和冷焊，法向接触面力 ${}^{t}p_{e1}^{1,c}$ 只能为压力，即要满足动力学约束条件：

$$ {}^{t}p_{e1}^{1,c} \geq 0 \tag{4.4} $$

切向接触力 ${}^{t}p_{e2}^{1,c}$，${}^{t}p_{e3}^{1,c}$ 的合力 ${}^{t}p_{et}^{1,c}$ 即为摩擦力

$$ {}^{t}p_{et}^{1,c} = \sqrt{({}^{t}p_{e2}^{1,c})^2 + ({}^{t}p_{e3}^{1,c})^2} \tag{4.5} $$

如果采用库仑摩擦模型，当 $\left|{}^{t}p_{et}^{1,c}\right| \leq \left|u_s\ {}^{t}p_{e1}^{1,c}\right|$ 时，点 ${}^{t}x^{1}$ 和点 ${}^{t}x^{2}$ 相对静止

$$ {}^{t}v_t = ({}^{t}v^{1} - {}^{t}v^{2}) - \left[({}^{t}v^{1} - {}^{t}v^{2}) \cdot e_1\right] e_1 = 0 \tag{4.6} $$

当 $\left|{}^{t}p_{et}^{1,c}\right| = \left|u_d\ {}^{t}p_{e1}^{1,c}\right|$ 时，点 ${}^{t}x^{1}$ 和点 ${}^{t}x^{2}$ 相对滑移

$$ {}^{t}v_t \neq 0 \quad {}^{t}v_t \bullet \left[{}^{t}p^{1,c} - ({}^{t}p^{1,c} \cdot e_1) \cdot e_1\right] \leq 0 \tag{4.7} $$

式中 u_s，u_d 分别为静、动摩擦因数，${}^{t}v^{n}$ 为 ${}^{t}x^{n}$ 的速度，${}^{t}v_t$ 为点 ${}^{t}x^{1}$ 相对于点 ${}^{t}x^{2}$ 的切向相对速度。由牛顿第三定律

$$ {}^{t}p^{2,c} = -{}^{t}p^{1,c} \tag{4.8} $$

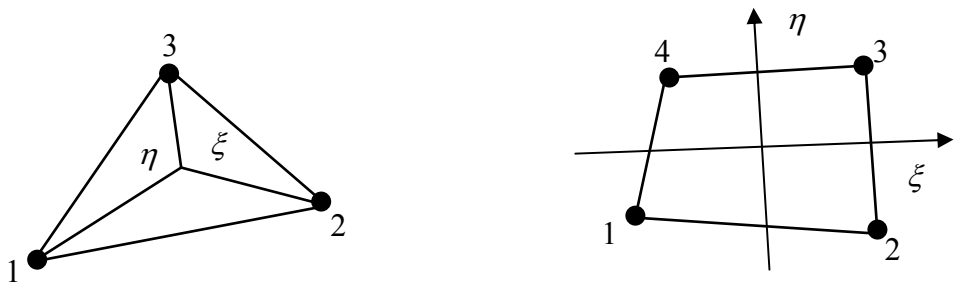
(3) 单一接触条件：

$$ {}^{t}g_{e1}\ {}^{t}p_{et}^{1,c} = 0 \tag{4.9} $$

4.1.2 接触界面的离散处理

对接触界面进行有限元分析时，与单元计算类似，首先也需要将接触体离散成为有限个单元的集合。对于板壳结构的接触碰撞问题分析，最常见的有三节点单元和四节点单元，如图 4.2 所示。为了方便接触搜寻和接触力计算，这些离散单元通常对应于有限元中的低阶单元，例如，本文第三章所介绍的 BT 四边形单元和 EST 三角形单元都能很好的用于接触界面的离散和计算。其中，三角形单元

由于具有更好的构形能力和网络自动划分能力，因此在大规模复杂结构的接触碰撞有限元分析中优势更大。在接触体系中，根据 WHIRLEY[179]的定义方法，这些离散单元称为接触面，接触面上的边称为接触边，接触边上的点称为接触点。当两边界接触时，把其中的一个边界叫做主动接触边界，而另一边界则称为被动接触边界。主接触边界上的接触块、接触边和接触点分别称为主接触块、主接触边和主接触点。对应的，被动接触边界上的离散单元称为从接触块、从接触边和从接触点。与有限元计算的思想一样，计算接触力时，以接触块为单位来考虑，只要能计算出任意一接触块上的接触力所做的虚功，就能计算出全部接触力。



(a) 三节点接触块　　(b) 四节点接触块

图 4.2 常用接触块及其等参坐标

假设一个接触块上有 m 个接触点，并以 ${}^{t}x^{n}$ 表示 t 时刻第 n 个节点接触点的坐标。同时假设可在接触块上定义一个等坐标系 $\xi-\eta$ 以使得接触块上的任一点都可用一组 $\xi-\eta$ 坐标来表示，参见图 4.2。那么接触块的几何形状可描述为

$$ {}^{t}x(\xi,\eta)=\sum_{n=1}^{m}\phi_n(\xi,\eta){}^{t}x^{n} \tag{4.10}$$

式中 ${}^{t}x(\xi,\eta)$ 表示任一点 (ξ,η) 处的坐标，$\phi_n(\xi,\eta)$ 表示对于 ${}^{t}x(\xi,\eta)$ 点的插值函数，相当于有限元中的形函数。

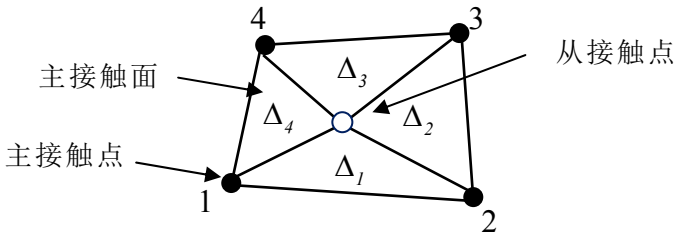


图 4.3 四节点接触块的面积坐标

通常情况下，有多种方法可以建立不同类型接触块的形函数[57]。以四节点接触块为例，本文采用如图 4.3 所示的面积坐标建立形函数，面积坐标形函数的定义分别为

$$\phi_1 = \Delta_2\Delta_3 / \Delta \tag{4.11}$$

$$\phi_2 = \Delta_3\Delta_4 / \Delta \tag{4.12}$$

$$\phi_3 = \Delta_4\Delta_1 / \Delta \tag{4.13}$$

$$\phi_4 = \Delta_1\Delta_2 / \Delta \tag{4.14}$$

式中，Δ_1，Δ_2，Δ_3和Δ_4分别为图 4.3 中所示子域的面积，Δ的定义如下

$$\Delta = (\Delta_1+\Delta_3)(\Delta_2+\Delta_4) \tag{4.15}$$

假设从接触点与主接触面间的接触力为 f，则通过以上的形函数可以计算得到离散到每个主接触点 i 上的接触力为

$$f_i = f \times \phi_i \tag{4.16}$$

4.2 接触碰撞问题的有限元算法

4.2.1 级域接触搜寻算法

级域算法是一种适用于复杂自接触问题的高效搜寻算法，其通过引入级和域的概念可以极大地提高算法的搜寻效率。同时，搜寻过程中，同一级内接触块的具有很好的计算独立性，符合 GPU 细粒度计算的要求。因此，本文计算采用级域接触搜寻算法，并主要研究该算法在 GPU 上并行化计算方法。

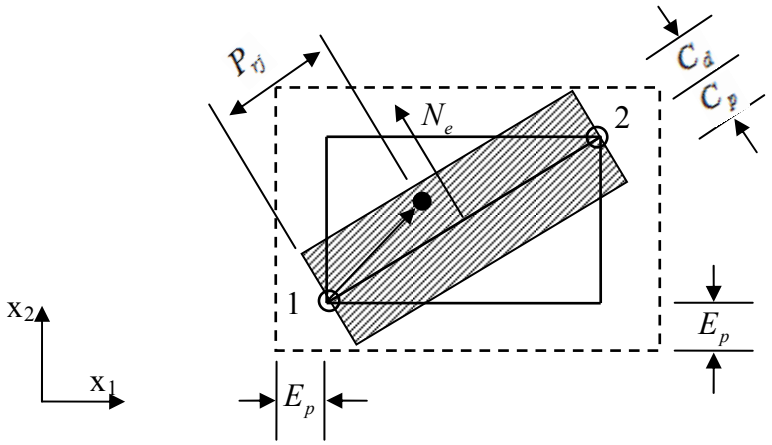


图 4.4 接触块的块域，接触域及扩展块域

首先，对一个通用的接触搜寻流程进行简要的介绍，以如图 4.4 所示的一个全局坐标系 x_1-x_2 中二维接触块为例。接触搜寻时，首先需要定义其接触块域，接触块域的定义为包含接触块，且边平行于全局坐标轴的最小矩形，如图 4.4 中的实线框所示。接触块域的数学表达式为

$$T = \left\{ (x_1, x_2) \,|\, x_i^a \le x_i \le x_i^b, i = 1,2 \right\} \tag{4.17}$$

其中

$$x_i^a = \min\left(x_i^1, x_i^2\right), i = 1,2 \tag{4.18}$$

$$x_i^b = \max\left(x_i^1, x_i^2\right), i = 1,2 \tag{4.19}$$

式中，x_i^1和x_i^2分别代表节点 1 和节点 2 的坐标分量。

接触块域的作用是对所有接触点进行一个初步的筛选，如果某个接触点要与接触块发生接触，首先必须是在其块域内。考虑到计算机计算的舍入误差，还需要建立统一的接触对判断标准，因此，提出了接触域的定义，如图 4.4 中填充矩形所示，其中C_d为接触的控制误差，C_p为容许的最大穿透量。搜寻过程中，所有落入接触域内的节点都认为与该接触块发生了接触。定义接触域前，需要先定义接触面的正反面，只有正面内的接触点才有可能发生接触，一般情况下，我们假定与接触面的外法向同侧的面为正面。

有了接触块域和接触域的定义，就可以得到如下的接触判断标准：

$$0 \le P_{rj} \le R_L \quad \text{和} \quad -C_p \le d \le C_d \tag{4.20}$$

其中，P_{rj}为节点在接触块上的投影距离，R_L为接触块的长度，d为节点到接触块的法向距离。

由图 4.4 可以发现接触块域没有完全包含接触域，因此，在接触搜寻的初始阶段只使用接触块域是不够的，还需要引入接触块扩展域的定义，如图中虚线矩形所示。接触块扩展域T_e的数学定义为：

$$T_e = \left\{\left(x_1, x_2\right) \mid x_i^a - E_p \le x_i \le x_i^b + E_p, i = 1,2\right\} \tag{4.21}$$

其中，x_i^a和x_i^b的定义为式(4.18)和式(4.19)所示。E_p为接触块域的扩展量，一般情况下，E_p的大小取决于由接触域的大小和接触块的空间取向

$$E_p \ge \max\left(C_d, C_p\right) \tag{4.22}$$

在建立好包含以上几个域的接触搜寻几何模型后，为了方便接触搜寻算法的描述，还需要引入两个接触搜寻中重要的概念：接触对和测试对。如果一个接触点落入一个接触块的扩展域内则两者构成一个测试对，如果一个接触点落入接触块的接触域内则两者构成一个接触对。显然，进行接触搜寻时，按空间大小，应该先搜索测试对，然后在测试对中寻找接触对。形成接触对的条件如式(4.20)所示，而形成测试对的条件如下

$$x_i^a - E_p \le x_i^h \le x_i^b + E_p, i = 1,2 \tag{4.23}$$

式中，x_i^h为接触点的坐标，x_i^a和x_i^b为接触块域的上、下限，它们的定义如式(4.18)和式(4.19)所示。

以上的讨论形成了一个二维问题的接触搜寻方法。对于三维问题，只需要适当的扩充即可。三维情况下的接触块域的形状如图 4.5 中(a)所示，为包含接触块且各面都平行于全局坐标轴平面的最小正六面体，其数学定义成为

$$T=\left\{\left(x_1,x_2,x_3\right)|\,x_i^{\min}\le x_i\le x_i^{\max},i=1,2,3\right\} \tag{4.24}$$

其中

$$x_i^{\min}=\min\left(x_i^1,x_i^2,...x_i^N\right) \tag{4.25}$$

$$x_i^{\max}=\max\left(x_i^1,x_i^2,...x_i^N\right) \tag{4.26}$$

其中 N 表示接触块的节点总数。相应的，三维情况下的扩展域可定义为：

$$T_e=\left\{\left(x_1,x_2,x_3\right)|\,x_i^{\min}-E_p\le x_i\le x_i^{\max}+E_p,i=1,2,3\right\} \tag{4.27}$$

三维情况下，接触域的定义相对复杂，如图 4.5 中(b)所示，其数学定义为

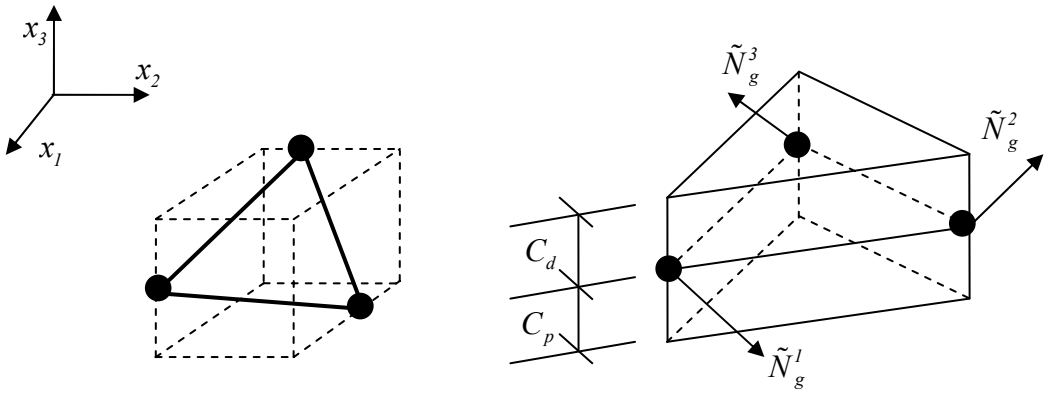
$$T_c=\left\{x\,|-C_p\le d(x)\le C_d,P_i(x)\le 0,(i=1,2,3)\right\} \tag{4.28}$$

式中

$$d(\boldsymbol{x})=(\boldsymbol{x}-\boldsymbol{x}_1)\cdot\boldsymbol{N}_1 \tag{4.29}$$

$$P_i(\boldsymbol{x})=(\boldsymbol{x}-\boldsymbol{x}^i)\cdot\tilde{\boldsymbol{N}}_g^i,(i=1,2,3) \tag{4.30}$$

其中 $\boldsymbol{x}$ 表示接触域中任意一点的坐标，$\boldsymbol{N}_1$ 表示接触块的单位法矢量，$\boldsymbol{x}^i$ 表示接触块上三个接触点的坐标，$\tilde{\boldsymbol{N}}_g^i$ 表示接触块上三个接触边的单位法向量。



(a)接触块域　　(b)接触域

图 4.5 三维状态下的三角形接触块

4.2.2 接触力计算方法

在所有的接触力计算方法中，罚函数法和防御节点法均具有接触力计算和有

限元运动方程完全解耦的特征，适用于显式有限元计算。同时，显式计算特征也使这两种方法均具有较高的计算独立性，适用于在 GPU 上的并行计算。

在罚函数法中，接触点允许穿透它的接触块，并假设作用在接触点上的接触力与它的穿透量成正比。计算时，在每一个时间步内先检查各个从节点是否穿透接触面，没有穿透则对该从节点不作任何处理。如果穿透，则在该从节点与被穿透主表面之间引入一个较大的反向作用力，其大小与穿透深度和接触面的体积刚度成正比

$$F_s = k_i d_i \tag{4.31}$$

式中，k_i为接触刚度因子，d_i为穿透量。目前常用的接触刚度因子k_i有如下三种计算方法

$$k_i = \frac{f_s \times A_i^2 \times K}{V_i} \tag{4.32}$$

$$k_i = \frac{f_s \times A_i \times K}{\min(diagonal)} \tag{4.33}$$

$$k = \frac{f_s \times m_i}{\Delta t^2} \tag{4.34}$$

式中，f_s为罚因子，一般取值 0.1，A_i是接触块的面积，V_i是接触块的体积，m_i为接触点质量，Δt为时间步长，K为接触块材料的体何模量，与材料的杨氏模量E和泊松比v相关

$$K = \frac{E}{3(1-2v)} \tag{4.35}$$

三种计算方法中，式(4.32)只适用实体单元的计算。因此，本文采用式(4.33)和式(4.34)其上，对于汽车碰撞等接触体间有明显材料参数差异的模型采用式(4.34)更加合适。

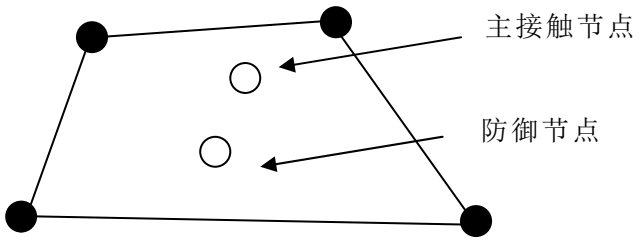


图 4.6 一对主接触节点和防御节点

防御节点法是一种既能精确计算接触力，又能避免求解联立方程组的接触力计算方法。如图 4.6 所示，在防御节点法中，每个接触对都增加了一个虚拟的接触节点即所谓的防御节点，防御节点具有和普通节点同样的属性，如位移、速度、

加速度、节点力和质量等物理量。

防御节点所具有的物理量均由插值法得到，以图 4.7 所示的两节点接触体系为例，实心点为从接触点，空心点为防御节点。假设从接触点的质量、法向加速度、法向节点力和法向接触力分别为 M_1、a_1、F_1 和 f_1，而防御节点上对应量记为 M_2、a_2、F_2 和 f_2。那么两者的运行方程可以写为：

$$M_1 a_1 = F_1 + f_1 \tag{4.36}$$

$$M_2 a_2 = F_2 + f_2 \tag{4.37}$$

为避免求解方程组，采用中心差分法，可能将式(4.36)和式(4.37)写为：

$$M_1\left[({}^t u_1 - u_1)/\Delta t - {}^l v_1\right]/\Delta t = F_1 + f_1 \tag{4.38}$$

$$M_2\left[({}^t u_2 - u_2)/\Delta t - {}^l v_2\right]/\Delta t = F_2 + f_2 \tag{4.39}$$

式中 ${}^t u_1$ 和 ${}^t u_2$ 分别代表从接触点和防御节点在下一时间步的位移，而 u_1 和 u_2 则代表在当前时间步的位移，${}^l v_1$ 和 ${}^l v_2$ 则代表从接触点和防御节点在上一时间步的速度。

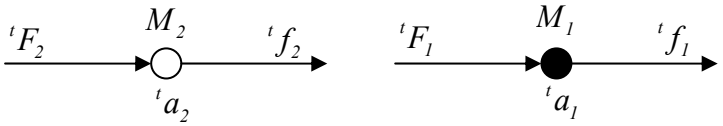


图 4.7 计算从接触点和防御节点间的接触力

考虑到式(4.9)所示的单一接触条件，从接触点和防御节点的法向间距可以表示为：

$$p = {}^l p + u_2 - {}^t u_2 - (u_1 - {}^t u_1) = 0 \tag{4.40}$$

其中，${}^l p$ 为上一时间步内的穿透量。综合式(4.36)到式(4.40)可得从接触点和防御节点间的接触力为：

$$f_1 = -f_2 = M_1 M_2 \left(\frac{F_2}{M_2} - \frac{F_1}{M_1} + \frac{{}^l v_2}{\Delta t} - \frac{{}^l v_1}{\Delta t} + \frac{{}^l p}{\Delta t^*} \right) / \left(M_1 + M_2 \right) \tag{4.41}$$

式中：

$$\Delta t^* = \Delta t(\Delta t + \Delta \tau)/2 \tag{4.42}$$

在得到接触块上防御节点的接触力后，则可以按照式(4.16)将防御节点接触力离散到接触块的每个主接触点上。

4.3 接触碰撞问题的程序执行流程

4.3.1 程序模块及计算时间分布

接触碰撞问题有限元分析程序的基本模块和流程如图 4.8 所示，主要包括如下几个部分：

(1) 接触前处理在显式迭代前进行，主要目的是形成接触点、接触边、接触面信息，以及接触面的相邻接触面、共享边、共享边上的节点信息等接触搜索过程中常用的信息，以避免迭代计算过程中的重复计算。

(2) 有限元计算用于计算当前接触状态下单元外力和单元内力，这部分的并行计算方法已经在第三章中讨论。

(3) 接触搜寻一般分为两部分，一部分称为接触前搜寻，另一部分称为接触后搜寻。在显式算法中，接触前搜寻是针对当前未处于接触状态的节点而言的，目的是检测是否存在新的接触点，而接触后搜寻是针对前一次时间步中已处于接触状态的节点而言的，其目的是检测这些节点是否仍然处于接触状态。

(4) 找到接触对后则进行接触力计算，随后需要将接触力离散到节点合力中，最后，根据节点合力进行节点的物理更新，如节点速度计算和坐标更新等。

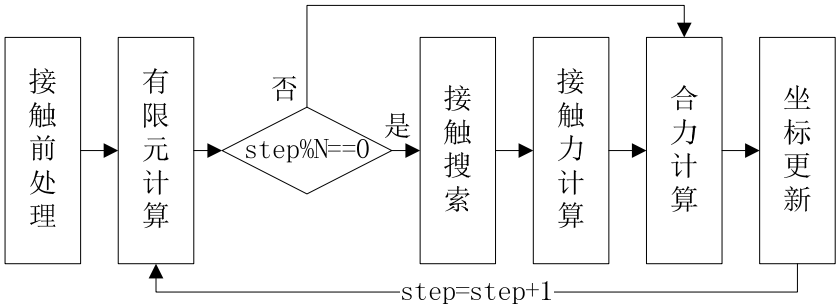


图 4.8 接触碰撞问题有限元分析模块

有限元分析程序的计算通常效率取决于单元方程的求解时间。但是，在进行接触碰撞问题分析时，接触算法往往取代单元计算成为主要的耗时点，接触算法的执行效率往往成为整个程序效率的主要瓶颈。表 4.1 所示为一个由 69625 个节点，65177 个单元组成的白车身碰撞模型采用 DYSI3D 程序[180]在 CPU 上进行串行计算的过程中，某个迭代步内计算时间的分布情况。

可知，接触搜索算法占用了 70%以上的总计算时间，其次为有限元计算即单元计算和节点物理量更新的时间，接触力计算所需要的时间则相对较少。因此，为了使程序的计算效率最大化，接触搜寻算法和有限元计算必然需要进行并行化计算。但是，考虑到接触力计算本身优质的可并行性，以及接触力计算与上下文间庞大的数据依靠要求，本文的并行化研究对象涉及到图 4.8 中除接触前处理外

的每一个部分，从而形成一个用于汽车车身结构接触碰撞问题有限元分析的全流程并行计算方法。

表 4.1 接触碰撞问题的计算时间分布

功能模块	时钟周期个数	占用比率
有限元计算	140	21.88%
接触搜索	478	74.51%
接触力计算	14	2.19%
其它	8	1.25%

4.3.2 串行程序中常用的加速策略

在实际应用中，考虑到显式有限元计算过程中的时间步长往往很小，接触点在一个时间步内位移量有限，通常情况下并不会引起接触状态的改变。因此，接触搜寻并不需要在每个时间步内都进行，可以设定一个 N 值，使程序每隔 N 个迭代步才进行一次接触搜寻，如图 4.8 所示。

本文在进行细粒度并行开发时，仍然会保留这些有效的加速策略，在开发的程序中，默认情况下设定 N 值为 10，当对计算的结果精度要求较高或者出现计算结果错误时，则可以缩小搜寻间隔到 3-5 个迭代步，只有在极端情况下才需要每步都进行接触搜寻。采用隔步搜寻的策略可以有力的缩减计算时间，但是在网格较密或者接触体系中接触体间相对速度较大的情况，该策略对接触搜寻的结果影响较大。

尽管以上的加速策略可以在一定程序上提高计算效率，但是，对于整车碰撞等大规模计算模型，单个迭代步计算所需的时间仍然过长，引起计算时间仍然十分可观，因此，本文继续探讨接触碰撞问题有限元分析各个模块的细粒度并行计算方法。

4.4 接触碰撞问题有限元分析并行化策略

4.4.1 单元计算部分的并行化

如表 4.1 所示，有限元计算的计算时间在接触碰撞问题的仿真分析中也占有较大的比例，因此缩减该部分的计算时间对整体计算效率的提升是有必要的。由于接触碰撞问题一般采用显式算法，因此，可以直接采用本文第三章介绍的显式有限元并行计算方法。通用的接触碰撞问题一般对单元计算等没有特殊要求，因此可以直接将基于 GPU 的显式有限元并行计算方法引入到串行碰撞有限元计算中。图 4.9 所为某薄壁梁撞刚性墙有限元计算过程中，加入并行单元计算后，关键步骤的时间分布情况和整体计算时间对比，该有限元分析模型由 4140 节点和

4080 个单元构成。

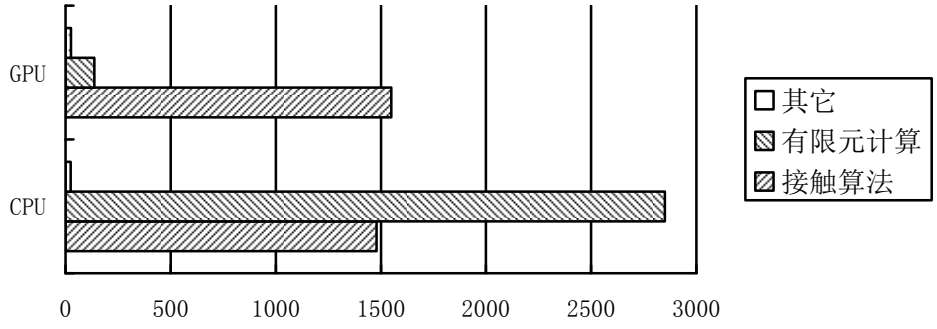


图 4.9 采用 GPU 进行有限元计算加速后的计算时间分布

由图可知，由于该薄壁梁算例的规模有限且接触关系简单，采用 CPU 串行计算时，有限元单元计算仍为主要的耗时部分。采用 GPU 进行单元计算部分的加速后，程序主要耗时部分也由单元计算部分变为接触算法部分。然而，即使是对这类初始有限元计算时间为主导的算例，只对有限元计算部分进行加速，接触算法仍然占用比较多的计算时间，程序整体计算效率的提升也十分有限。

4.4.2 级域接触搜寻算法的细粒度并行化策略

综合上文中对接触算法的理论分析，本文分步对算法的并行化进行研究，首先将一个级域接触搜寻算法归纳为如下三个主要步骤：

(1) 接触块的扩展域计算；

(2) 找到落入接触块扩展域内的接触点，形成测试对；

(3) 从测试对找到接触对；

其中，接触块的护展域计算具有很好的块独立性，因此，在 CUDA 架构下，可以采用一个 CUDA 线程对应于一个接触块的计算方式，即可取得很好的计算效率。

4.4.2.1 CUDA 架构下测试对的细粒度并行搜寻方法

在 DYSI3D 中，三维碰撞问题的测试对搜寻通常采用在长边方向上排序的策略，由此可以排除大量无意义的接触点和接触块对比，成而提高搜寻效率。该策略的串行执行流程如下：

首先，计算整个接触体系的最长边方向，将接触点和接触块扩展域的下限点在长边方向的坐标 x_n^k 和 x_s^k 按前部分接触块扩展域后半部分接触点的顺序存入同一个一维数组中，如图 4.10(a)所示。混合后，按从小到大的顺序排序，得到图 4.10 中(b)所示的结果，图中黑点代表接触块扩展域的下限点，空心点代表接触点坐标。

然后，在排序表中进行测试对第一次搜寻。从排序表的第一个元素开始逐个

检测，当发现一个扩展域下限点后就计算出扩展域在长边方向上跨度。并在排序表找到落入该跨度内的接触点号。例如，图 4.10 中落入第 2 个接触块扩展域在长边方向跨度 L_s^2 的接触点有 1，2，3 三个节点。

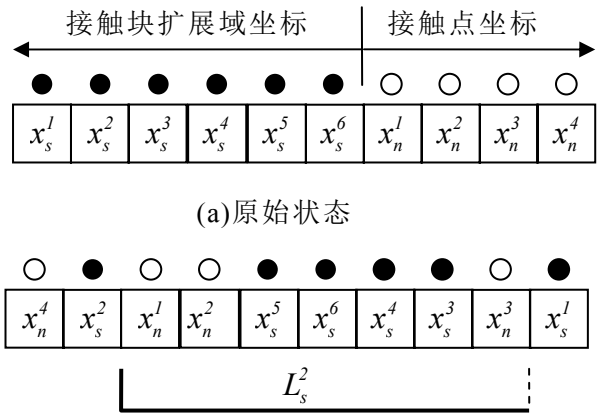


图 4.10 接触点与接触块扩展域混合排序方法

最后，在长边方向的搜索结束后，得到包括接触块扩展域和接触点从属关系的数组后，就可以通过直接比较另外两个方向上坐标值的方法，检测接触点是否真正在该接触块的扩展域内。如果接触点另外两个方向的坐标值也处于接触块扩展域内，就构成一个测试对。以上的测试对搜寻过程，可以总结为图 4.11 所示的计算流程。

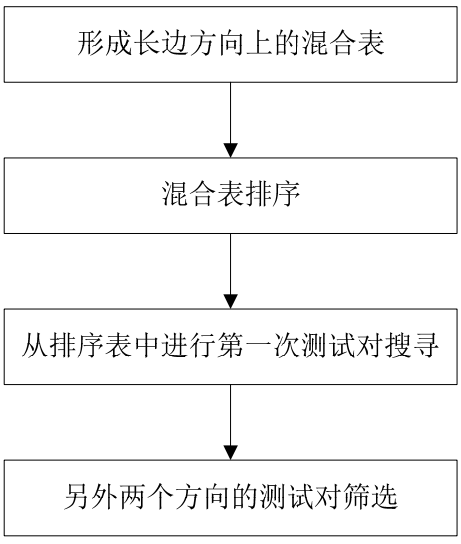


图 4.11 测试对搜寻流程

根据该搜寻流程，本文制定的细粒度并行执行策略如下：

(1) 混合表的形成可以在接触块的并行扩展域计算过程上完成，具有很好的并行性。并行计算时，采用一个线程对应一个接触块的方式并行计算每个扩展域，

并存入到全局显存中用于存储混合表的一维数组中。

(2) 并行排序方法是并行计算中的热门讨论对象，在 GPU 实现并行排序已经有多种方法发表，最常用的是基于本文 3.3.4 节所述的并行缩减算法的并行排序方法。在本程序中，为了降低编程难度，结合 CUDA 的版本进化，本文采用由 CUDA 后期提供的 Thrust 库[181]来实现排序过程。Thrust 是一个类似于 STL 的专门用于 GPU 并行计算的 C++模板库，使得程序员可以在 CUDA 程序中快速实现并行排序等操作。采用 Thrust 库实现混合排序只需要用到一个函数就可以，十分方便，代码如框 4.1 所示。

框 4.1 基于 Thrust 库的混合排序程序代码

```
…
// 包含 Thrust 头文件
#include <thusrt/sort.h>;
#include <thusrt/device_ptr.h>;
…
// 形成包含长边方向上接触点和接触块扩展域下限的数组 d_xsgd
// 数组大小 nums=接触点总数+接触块总数
// 形成一个整型的索引数组 d_ksgd，对应于 d_xsgd 坐标所属节点号和块号
// 排序，按 d_xsgd 数据从小到大排序，d_ksgd 也跟随变化
thrust::sort_by_key(thrust::device_ptr<double>(d_xsgd),
                    thrust::device_ptr<double>(d_xsgd+nums),
                    thrust::deivce_ptr<int>(d_ksgd));
…
```

(3) 长边方向上第一次测试对搜寻的串行算法由两层循环组成，在最差的情况下时间复杂度为 $O(n^2)$，经测试，占用了接触搜寻中 90%以上的计算时间。考虑到 GPU 细粒度执行模式以及两层循环的依赖关系，本文采用的方法就是将两层循环中的外层循环展开到每个 CUDA 线程上，此时，最高时间复杂度降为 O(n)。

该方法的基本思路如图 4.12 所示，其中主要的难点在于如何在全局显存空间上紧密的存储搜寻到的测试对。如图 4.12 所示，在 GPU 上采用一个线程对应于一个接触扩展域的方式执行测试对搜寻时，单个线程存在有：(1)不是接触块下限点，(2)没有找到测试对，(3)找到一个测试对以及(4)找到多个测试对等四种情况。情况(1)和情况(2)的线程不需要记录测试对，因此，不涉及写全局显存的操作。但是，在有测试对产生的情况下，出现情况(3)和情况(4)的线程需要进行类似于压栈的操作，这在并行计算中就需要引入原子操作。

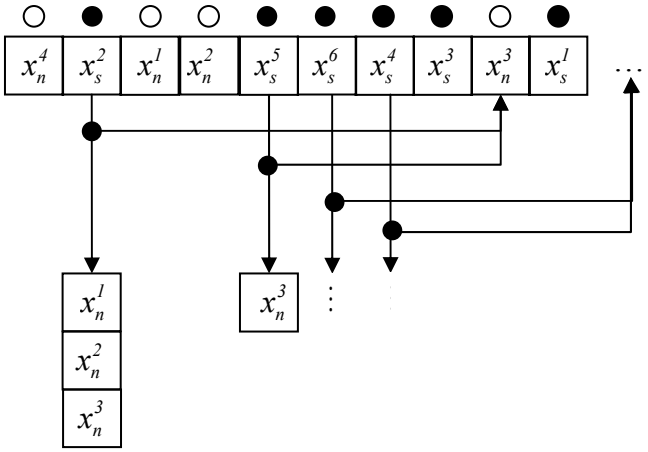


图 4.12 测试对搜寻的细粒度执行模式

采用原子操作进行搜寻结果存储的并行执行方式如图 4.13 所示，假设为 10 线程执行模式，同时，为了方便描述，没有考虑情况(4)的存在。图中 e1 是指判断当前线程对应的排序中的位置是否为接触块的下限点，如果不是，则终止。e2 是与排序表中当前线程对应位置后的值对比，判断是否可以找到一个符合条件的接触点，如果可以，则进行 c1 计算。c1 是指由原子加函数 *atomicAdd()*计算得到当前可写入的全局显存位置，并存入找到的接触点。其中，根据线程到达时间戳的不同，接触点存入的目标位置是随机的，如线程 1 对应的测试对存到 M1 位置，而线程 6 存入 M2 位置。

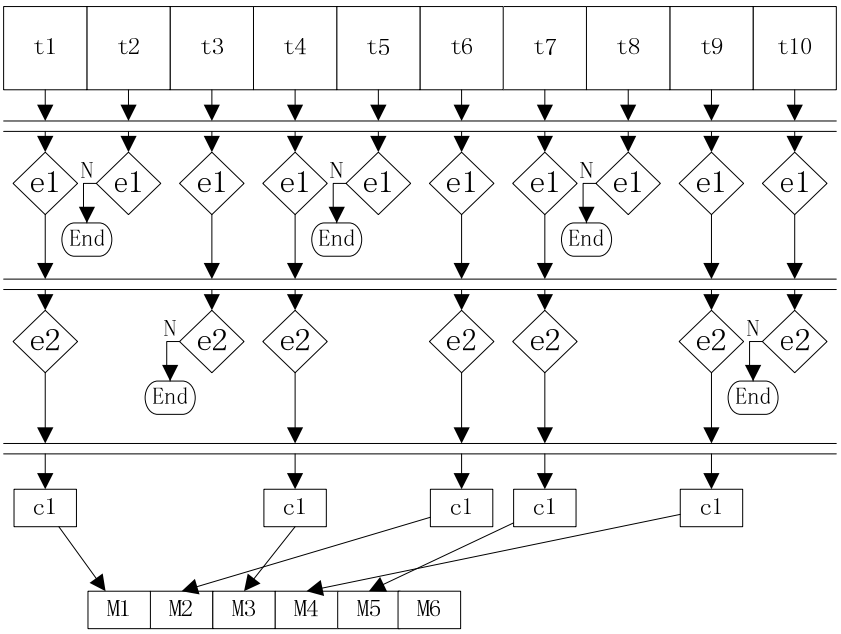


图 4.13 基于原子操作的结果存储方法

由于原子操作的顺序执行特征，这种执行方法对计算效率会有一定的影响，但是可以在不引入任何附加计算的前提下，避免全局显存空间的写入冲突，同时保证结果的顺序存储。

4.4.2.2 CUDA 架构下接触对的细粒度并行搜寻方法

级域算法中，根据如图 4.14 所示级的概念，接触对搜寻首先从面级开始，面级构成接触对的测试对再进入到边级进行检测，边级检测成功的进入到点级的接触检测，最终形成真正的接触对关系。这种基于级的检测方法，可以有效的减少无用检测的次数，提高检测效率。

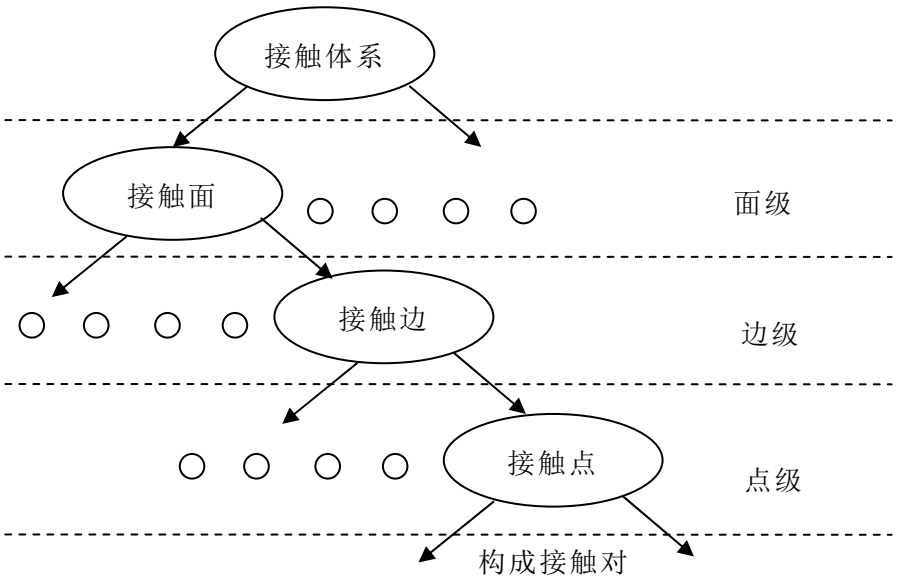


图 4.14 级的概念

在每一级的接触检测中，均需要以测试对为基本计算单元计算测试对中接触点与对应级的几何关系，以此来判断是否满足接触关系。在同一级内，这种几何关系的计算具有较好的细粒度并行性，可以采用一个线程与一个测试对对应的方式在 GPU 中实现并行计算。

为了节约显存空间，同时减少低效率的全局显存空间操作，级与级之间测试对的更新均在同一个数组空间内进行更新操作。为了保证上一级更新后的数组在下一级可以满足线程的顺序执行，而不出现跳跃，本文提出了一种基于计算后排序思想的搜寻策略。采用此策略的接触对并行搜寻方法如图 4.15 所示，假设有 10 个测试对 TP，首先并行进行 e1 判断，e1 是指式(4.20)或式(4.28)的计算和判断，如果当前测试对满足接触条件，则将该测试对的编号保存，如果不满足，则将该测试对编号的设为一个极大数 MX，这样得到一个新的测试对数组。然后，采用 Thrust 函数进行从小到大的排序，这样满足接触条件的接触对就全部排到了数组的前面部分，通过寻找第一个 MX 出现的位置即可以知道共有多少个接触对产生，同时也满足顺序存储的要求。这样得到的数组和接触对总数信息有利于进行下一级接触对程序的编写，并提高程序的执行效率。其中，MX 大于总测试对数即可，通常设 MX 的大小为总测试数+1。

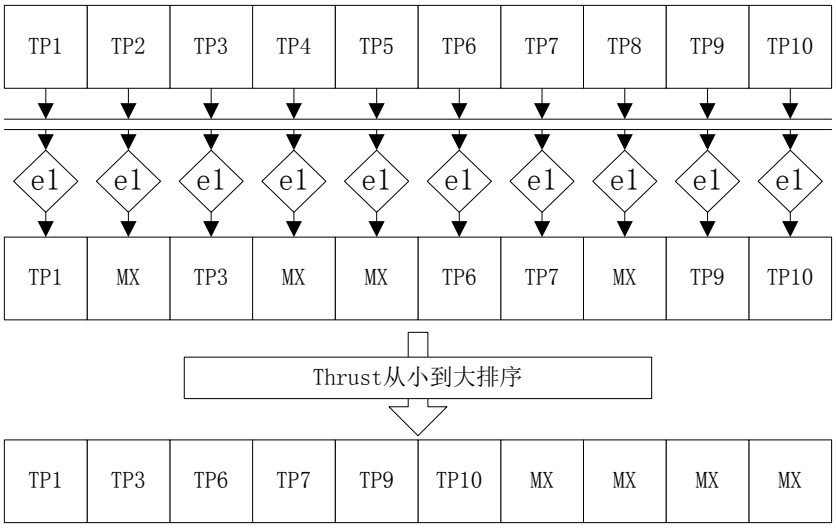


图 4.15 接触对搜寻的细粒度执行模式

4.4.3 接触力计算的并行策略

本文采用的罚函数法和防御节点法虽然在计算细节上有所不同，但是，整体上一般由两部分组成：首先，计算穿透量，并由穿透量计算惩罚力，如式(4.31)和式(4.39)所示；然后，将从接触点和主接触面之间产生的接触力组合到整体合力矩阵中。因此，两种方法的并行计算策略基本一致，所以在这里统一进行介绍。

框 4.2 双精度原子加操作

```
__device__ double atomicAdd(double* address, double val)
{
    unsigned long long int* address_as_ull = (unsigned long long int*)address;
    unsigned long long int old = *address_as_ull, assumed;
    do {
        assumed = old;
        old = atomicCAS(address_as_ull, assumed,
                        __double_as_longlong(val +
                        __longlong_as_double(assumed)));
    } while (assumed != old);
    return __longlong_as_double(old);
}
```

事实上，继承显式算法的天然并行性，穿透量和惩罚力的计算也同样满足细粒度并行的要求，并行实现方法与并行单元计算过程类似。同时，接触力离散的过程也与内力离散过程类似，将接触块上的力离散到节点力上的过程直接并行同样会存在“竞写”的并行冲突。预索引的方法也同样可以应用于此，但是，考虑

到此部分涉及到计算量较小，本文因此直接采用原子加操作来实现该组装过程。由于目前 CUDA 不支持双精度浮点数的原子加操作，因而需要能过 *atomicCAS()* 函数来间接实现，函数代码如框 4.2 所示。

4.5 数值算例及分析

综合以上的并行算法，本章基于自主开发的碰撞仿真计算程序 DYSI3D，开发了基于 GPU 的碰撞过程计算机仿真并行计算软件 CPS-GPU(软件著作权编号：2011SR001966)。为了验证该软件和本文所提算法的计算效率和计算精度，进行了多个数值分析，并将计算结果与碰撞仿真程序 DYSI3D 的 CPU 计算结果进行对比，以说明计算结果的正确性。同时，对 CPU 和 GPU 的执行时间对比，计算 GPU 的加速比，说明 GPU 并行计算的高效性。本章测试平台采用 Intel Core I7 930 CPU、6GB 内存、Nvidia GTX 580 GPU、3GB 显存、Microsoft Windows 7 64 位操作系统，编程软件为 Visual Studio 2010 以及 CUDA 5.0。

4.5.1 数值算例

4.5.1.1 受冲击载荷的悬臂梁

首先对弹性材料碰撞过程的计算精度进行验证，采用的算例由 Oldenburg[182] 设计，由具有一定初始速度的弹性球去撞击同样材质的悬臂梁，如图 4.16 所示。模型的几何参数为：R=0.01m，B=0.24m，L=1.0m，h=0.0015m；材料参数为：杨氏模量 E=200Gpa，泊松比 v=0.3；弹性球的初始速度为 3m/s。采用 BT 四边形单元进行计算，并采用防御节点法进行接触力的计算。

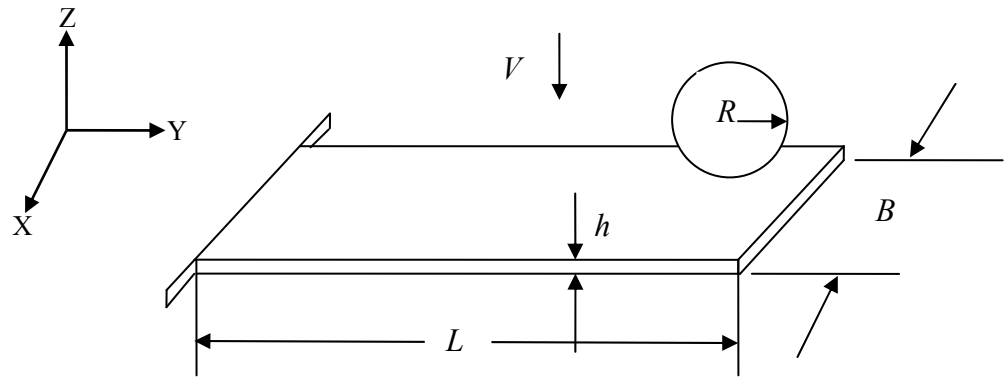


图 4.16 受冲击载荷的悬臂梁

图 4.17 所示为采用本文算法的 GPU 并行计算和 DYSI3D 程序 CPU 串行计算的球面上某节点的位移动态响应曲线对比。图中两条响应曲线完全重合，可知，GPU 与 CPU 的计算结果完全一致。同时，图 4.18 所示为采用 GPU 计算的变形动态响应过程，可知仿真可以较好的反应真实的碰撞过程。

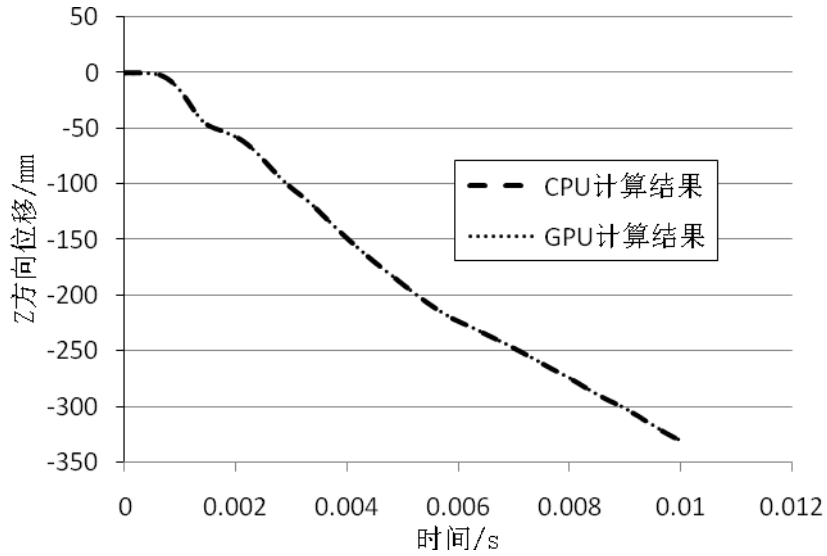


图 4.17 Z 方向位移的动态响度曲线对比

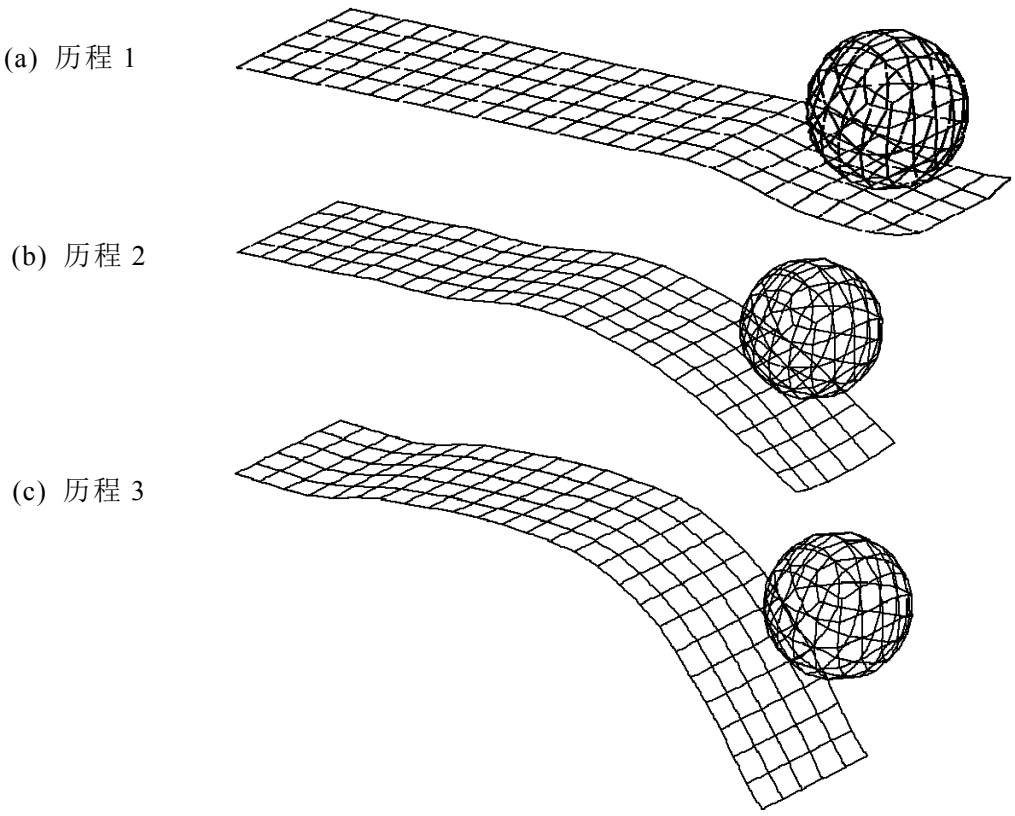
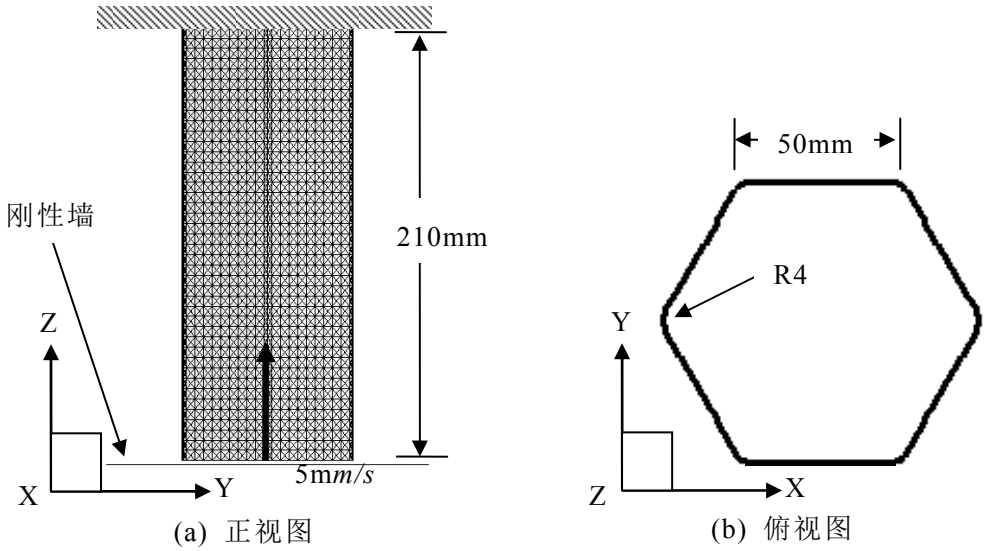


图 4.18 受冲击载荷的悬臂梁的变形历程

4.5.1.2 薄壁梁和刚性墙的碰撞

薄壁梁是汽车结构中重要的承载和吸能构件，汽车在发生碰撞过程中，碰撞动能主要通过车身前部发动机舱处的薄壁梁的塑性变形来吸收，以保证驾驶舱的安全。因此，本文采用如图 4.19 所示带圆角的正六边形薄壁梁对算法计算这类工

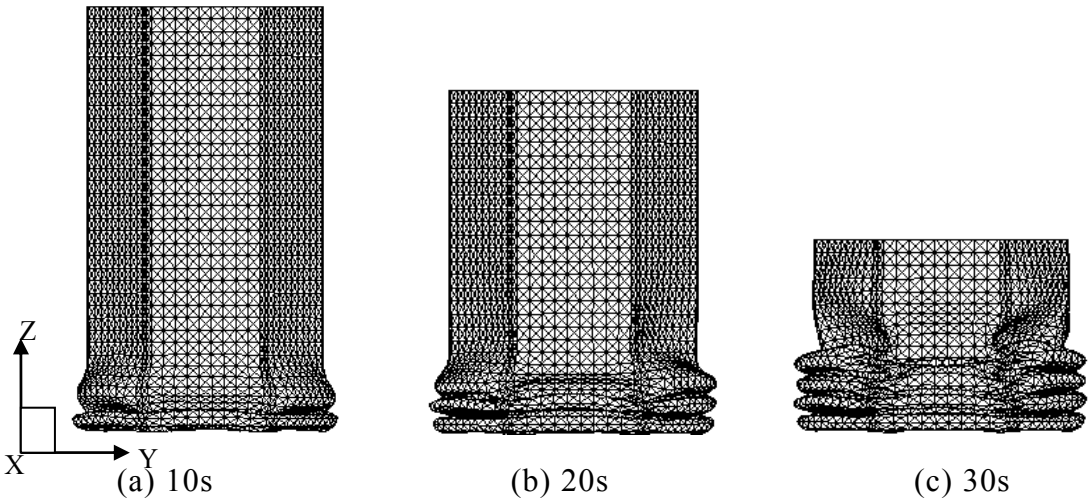
程问题的精度和效率进行验证。在 0-30 秒的时间内将薄壁梁的一端固定，用具有 5mm/s 线性速度的刚性墙撞击薄壁梁。模型的塑性材料参数具体为：杨氏模量 E=68.9Mpa，泊松比 v=0.3，屈服应力 σ_y=0.178Mpa，密度为 2.86×10^{-6}kg/mm^3，板料厚度为 2.5mm。



(a) 正视图　　(b) 俯视图

图 4.19 薄壁梁和刚性墙的碰撞

对几何模型采用 EST 三角形壳元进行建模，得到由 13420 个单元，6819 个节点构成的有限元模型，同样分别在 CPU 和 GPU 上进行计算。采用 GPU 计算时，可以得到如图 4.20 所示为，EST 单元计算得到的变形历程，可知仿真可以较好的反应真实的压缩过程。



(a) 10s　　(b) 20s　　(c) 30s

图 4.20 薄壁梁和刚性墙碰撞的 EST 单元计算结果

图 4.21 所示为 CPU 和 GPU 计算得到的 Y-Z 中截面轮廓线对比。两者的轮廓线完全重合，可见采用 EST 三角形单元时，CPU 和 GPU 计算的结果同样是完全一致的。计算效率方面，该计算模型共需要 62924 个计算迭代步，并且为了检测

搜索搜寻的稳定性和效率，每步均进行接触搜寻，即前文中所提的 N 值设为 1，并采用罚函数法进行接触力计算，最终，采用 GTX 580 取得的计算加速比为 7.16 倍。

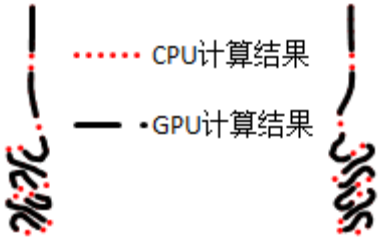


图 4.21 CPU 和 GPU 计算的 Y-Z 截面轮廓线对比

4.5.1.3 白车身碰撞

作为接触碰撞有限元分析的重要应用领域，汽车碰撞仿真是现代汽车企业研究汽车碰撞安全性的重要手段，可以有效的加快研发效率并降低研发成本。但是，由于汽车车身模型往往比较庞大和复杂，因此，有限元计算模型的单元数也十分可观，计算量庞大。为了检测本章提出的并行算法对于大规模接触碰撞模型的求解效率，采用如图 4.22 所示白车身碰撞模型进行碰撞分析。将该车以 16m/s 的初始速度撞向刚性墙，车身为塑性材料，具体材料参数为：杨氏模量 E=0.21×10^{6}Mpa，泊松比 v=0.3，屈服应力 σ_y=206Mpa，密度为 2.45×10^{-6}kg/mm^{3}。

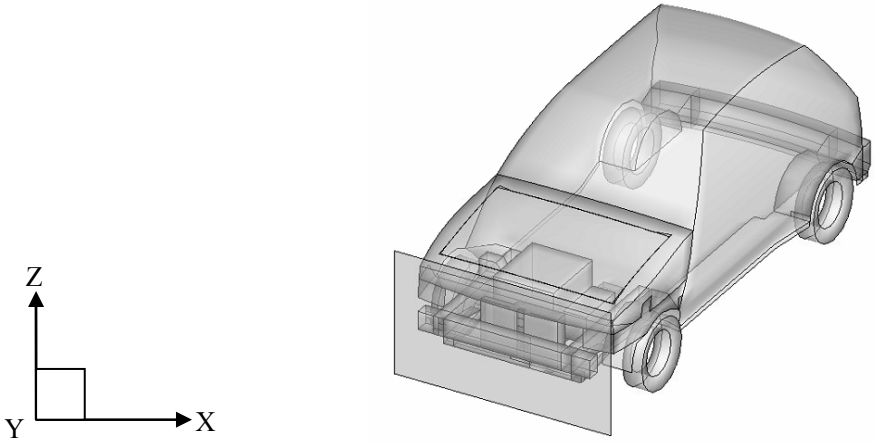


图 4.22 白车身模型

同样采用不同大小网格对车身进行建模，仿真终止时间为 0.05 秒，采用每 10 步进行一次接触搜寻的加速策略，即设 N 值为 10。三种不同规模型的有限元模型信息如表 4.2 所示。

三种规模分别在 CPU 和 GPU 中进行计算，其中，规模 2 的计算仿真结果如图 4.23 所示，该规模下，整个计算流程的迭代步数为 143028 步。图 4.23 所示为

三种计算规模下，采用 GPU 和 CPU 计算时，总单元计算加速比、单元计算加速比以及总加速比对比。可以发现，GPU 并行可以有效的提升该仿真过程的计算效率，程序的总加速比最高可以达到 22 倍。

表 4.2 白车身的三个有限元模型信息

	单元数	节点数	自由度总数
规模 1	20597	22404	134424
规模 2	73680	73727	442362
规模 3	294720	294803	1768818

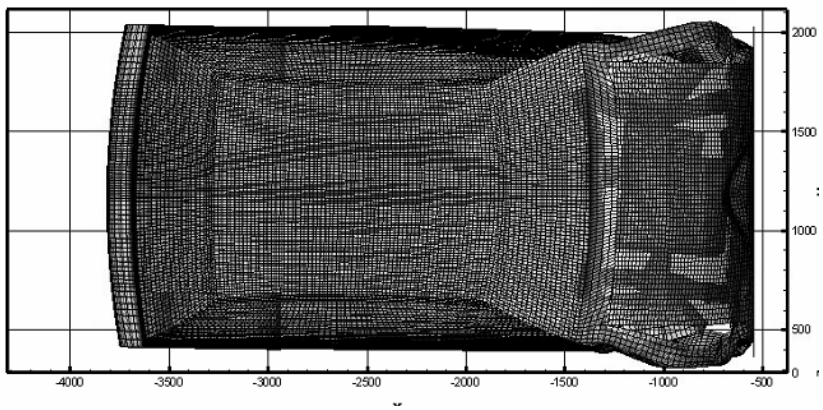


图 4.23 规模 2 的有限元仿真结果

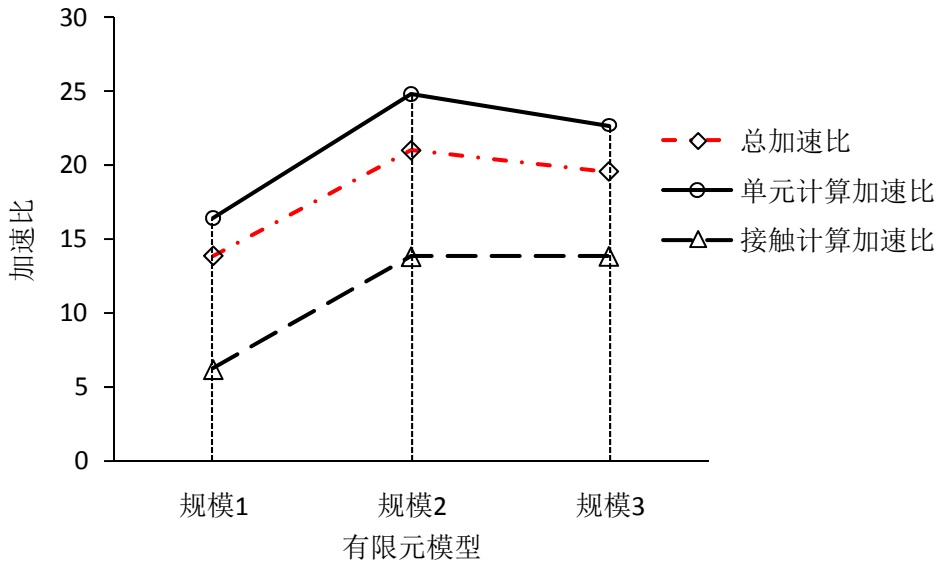


图 4.24 三种规模的计算加速度对比

4.5.2 结果分析

由以上三个算例可以看出，本章提出的并行计算方法具有很高的计算精度，

无论是采用 BT 四边形壳单元还是采用 EST 三角形壳单元对弹性问题和弹塑性接触碰撞问题进行分析时，GPU 并行计算与 CPU 计算结果完全一致，因此可以证明，GPU 并行计算可以满足接触碰撞问题有限元仿真对计算精度的要求。同时，对 GPU 和 CPU 的计算时间对比可知，本文所提出的并行算法具有较高计算效率，可以有效的缩短接触碰撞问题的有限元分析时间，采用 GTX580 进行白车身碰撞分析时，可以取得 20 倍左右的计算加速比。

4.6 本章小结

本章首先介绍了接触碰撞问题有限元计算的接触界面的离散方法，以及基于离散界面的接触搜寻过程和接触力计算方法。通过对接触碰撞问题有限元串行计算程序的基本构成和计算时间分布进行了分析，证实了接触算法才是此类仿真过程中主要的耗时部分。然后，对接触碰撞问题各部分算法在 GPU 上执行的可行性进行了分析，并提出了相对应的适应于 GPU 执行的细粒度并行化执行策略。其中，有限元计算采用了第三章中提出的基于 GPU 基本有限元显式并行计算方法，而基于 GPU 的并行接触算法中，主要包含并行接触搜寻和并行接触力计算两个部分：

(1) 提出了基于 GPU 的并行级域接触搜寻算法，采用线程与接触块一一对应的方法实现了测试对搜寻的细粒度并行，保证了程序高效、正常的执行。其中，为了提高程序的编程效率，采用 Thrust 库进行接触块与接触点的混合排序，采用原子操作来避免测试对存储时的竞写错误。

(2) 在同一级提出了采用线程与测试对一一映射的接触对细粒度并行搜寻方法。同时，提出采用计算后排序的方法来保证级与级的无缝数据传输，同时节约了显存空间。

(3) 提出了基于 GPU 的并行罚函数法和并行防节点法两种接触力计算方法。这两种接触力计算方法中，穿透量和惩罚力的计算比较独立，可以方便的细粒度执行。对于接触力的离散，直接采用基于 *atomicCAS()*函数的双精度原子加操作来实现。

最终，基于自主开发的碰撞仿真串行计算软件 DYSI3D 开发了基于 GPU 的碰撞过程计算机仿真并行计算软件 CPS-GPU(软件著作权编号：2011SR001966)。数值算例表明该软件在 GPU 上采用 BT 四边形单元和 EST 三角形单元进行汽车车身结构接触碰撞问题仿真计算时，均具有很好的计算精度和计算效率。例如，在 GTX580 显卡上进行近 200 万个自由度的白车身碰撞仿真计算时，可以取得 20 倍左右的计算加速比，极大的缩减了计算时间。

第 5 章 基于 GPU 的薄板冲压成形并行计算方法

薄板冲压成形是现代汽车产业中一种非常重要的制造技术，具有材料利用率高，生产成本低，容易实现自动化生产等优点，广泛应用于汽车车身制造以及薄壁结构件的成形等。汽车结构接触碰撞过程仿真一个重要的应用领域即车身覆盖件和板壳结构件的薄板冲压成形仿真分析。因此，本章基于第四章的研究内容，着重研究薄板冲压成形的并行仿真计算方法。本章基于自主开发的板料成形软件 CADEMII 开发了 CADEMII 的 GPU 并行计算版本 CADEM-GPU(软件著作权编号：2010SR052426)。在 CADEM-GPU 的开发过程中，并不是对原串行计算流程的生硬套用，而是从理论出发，基于前面两章的研究成果，提出了一系列适用于 GPU 计算的薄板冲压成形并行计算方法。同时，根据 GPU 的计算特点，研究了软件的结果数据输出和计算精度等问题，并加入实时显示技术，扩展了软件功能。

5.1 薄板冲压成形问题的有限元仿真方法

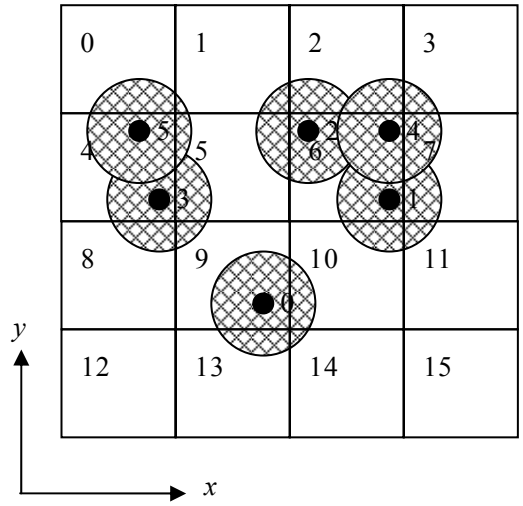
5.1.1 主要物理现象及仿真方法

冲压成形是指利用金属板料塑性变形的特点，通过模具对板料施加作用力，使其在外力的作用下产生塑性变形，从而获得与模具表面一致的形状。从力学角度来看，可以用一个多体接触体系的运动和相互作用过程来表示任一冲压成形过程，包括有接触碰撞现象、弹塑性变形现象、摩擦磨损现象等主要的物理现象[183]。通用接触碰撞现象和弹塑性变形现象的并行数值仿真技术已经在本文的第三章和第四章中介绍。但是，针对冲压成形问题，对这种物理现象的分析时，也存在有其特殊性。

(1)可简化的接触碰撞现象。在 CADEMII 软件中，采用一体化算法[58]进行接触搜寻，再采用罚函数法进行接触力的计算。一体化接触搜寻算法是一种高效的接触搜寻算法，其基本思路是采用子域分类法，将包含所有网格节点的空间，用许多相同大小的子域对空间进行区域划分，并按一定的顺序方式将空间中的每个网格和节点划分到各个子域中。由于接触发生时，接触点和接触块必然在同一个子域内或在相邻的子域内，因此，通过子域的划分就完成了测试对的搜寻。图 5.1(a)所示为一个简单二维接触体系的子域划分，设该接触体 x 方向上边上为 D_x，y 方向上边上为 D_y，并假设我们拟在两个方向划分的子域数分别为 N_y 和 N_y，那么子域在两个方向上长度分别为：

$$d_x = D_x / N_x \tag{5.1}$$

$$d_y = D_y / N_y \tag{5.2}$$



(a) 二维接触体系子域的划分

子域号	接触点数	接触点号
0	0	
1	0	
2	0	
3	0	
4	2	3,5
5	0	
6	3	1,2,4
7	0	
8	0	
9	1	0
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	

(b) 接触点的归纳结果

图 5.1 二维接触体系的子域划分

对于划分好的子域可以采用二维整数坐标系 *IJ* 进行管理，每个子域的编号可上下式计算得到：

$$L = J \times N_x + I \tag{5.3}$$

根据整数坐标系，可以计算得到每一个接触点分属于那个子域，接触点的整数坐标计算方法如下：

$$I = \mathrm{int}[(x - x_{\min}) / d_x] + 1 \tag{5.4}$$

$$J = \mathrm{int}[(y - y_{\min}) / d_y] + 1 \tag{5.5}$$

式中，$x_{\min}$ 和 $y_{\min}$ 为接触体系两个方向上的最小坐标，*int* 代表对数值的取整操作。图 5.1(b)中的表格即为接触点按整数坐标系对接触点进行归纳的结果。

同样的方法也可以用于计算扩展域与哪些子域相交。首先，将接触块扩展域的上、下限点也赋予整数坐标。设上限点的整数坐标为(I_{max},J_{max})，下限点的整数坐标为(I_{min},J_{min})，对于子域(I,J)，如果满足式(5.6)条件就代表该子域与扩展域相交。

$$\mathrm{I}_{min} \le I \le \mathrm{I}_{max} \ and \ J_{min} \le J \le J_{max} \tag{5.6}$$

接下来，通过对该子域内的接触点与扩展域进行比较就可以得到测试对。

冲压成形过程中，由于模具的刚度远远大于板料刚度，在成形过程中模具基本上不变形，因此，模具通常当成刚体来对待。同时，模具的运动通常是已知的，以一单动拉延模为例，假设凸模沿着 Z 轴方向运动，凹模固定不动，压力圈则只能沿着 Z 轴方向上下运动。考虑到冲压模具这种刚性特征以及模具和板料的运动趋势相对稳定的特点，在每个迭代步内，可以不再进行以上基于盒形排序的接触前搜寻，只要进行接触后搜寻即可。通过数值分析证明，这种简化在板料成形分析中是可行的，并且具有较高的搜索精度[184]。

(2)材料的高精度塑性变形现象。由于板料是冲压过程中唯一的变形体，因而有限元计算时板料的应力应变关系对仿真结果有直接的影响。成形过程中，材料之所以在模具卸载后仍然会保持变形后的形状，而不是恢复到原始状态，主要是由于塑性变形的存在。本构关系建立了材料的塑性变形现象中的屈服准则和流动准则，确定了板料在复合应力状态下何时屈服，以及材料屈服后如何塑性流动。由于金属板料的各向异性，需要采用各向异性的材料屈服准则。常用的屈服准则有各向同性的 Von Mises 屈服准则、厚向异性的 HILL 屈服准则[185]以及正交各相异性的 Barlat 屈服准则[186]。

HILL 屈服准则最早是 Hill 在 1948 提高的 HILL48 二次屈服准则，屈服函数如下：

$$\sigma_{Hill48} = \sqrt{\frac{1}{2}\left(H\left(\sigma_{11}-\sigma_{22}\right)^2 + F\left(\sigma_{22}-\sigma_{33}\right)^2 + G\left(\sigma_{33}-\sigma_{11}\right)^2 + 2N\sigma_{12}^2\right)} \tag{5.7}$$

式中，$\sigma_{ii}, ii = 1,2,3$ 代表双向拉伸状下的屈服应力，F，G，H，N分别为板料各向异性系数，当F，G，H，N的取值分别为：1，1，1，3 时，式(5.1)等于 Von Mises 屈服准则函数。对于 HILL48 屈服准则，F，G，H，N的取值如下：

$$H = \frac{1}{2}\left(\frac{1}{R_{22}^2} + \frac{1}{R_{11}^2} + \frac{1}{R_{33}^2}\right) \tag{5.8}$$

$$F = \frac{1}{2}\left(\frac{1}{R_{22}^2} + \frac{1}{R_{33}^2} - \frac{1}{R_{11}^2}\right) \tag{5.9}$$

$$G=\frac{1}{2}\left(\frac{1}{R_{11}^2}+\frac{1}{R_{33}^2}-\frac{1}{R_{22}^2}\right) \tag{5.10}$$

$$N=\frac{3}{2R_{12}^2} \tag{5.11}$$

其中，

$$R_{11}=1;R_{22}=\sqrt{\frac{r_{90}(r_0+1)}{r_0(r_{90}+1)}};R33=\sqrt{\frac{r_{90}(r_0+1)}{r_0+r_{90}}} \tag{5.12}$$

式中，r_0，r_{45}，r_{90}为试样与轧制方向的角度分别成$0°$，$45°$和$90°$时的厚向异性系数。

针对 HILL 屈服准则只考虑了厚向异性，而板面内仍假设为各向同性的不足。Barlat 等人 1989 年提出了可以考虑面内剪切应力的 Barlat 屈服准则：

$$F=a\left|K_1+K_2\right|^m+a\left|K_1-K_2\right|^m+2\left(2-a\right)\left|2K_2\right|^m=2\bar{\sigma}^m \tag{5.13}$$

其中，

$$K1=\frac{\sigma_{xx}+h\sigma_{yy}}{2} \tag{5.14}$$

$$K2=\sqrt{\left(\frac{\sigma_{xx}+h\sigma_{yy}}{2}\right)^2+p2\sigma_{xy}^2} \tag{5.15}$$

式中，x，y为板料的轧制方向和横向。m为与材料晶体结构相关的指数，a，h和p为塑性各向异性系数。$\bar{\sigma}$为轧制方向上的等效应力。

HILL 屈服准和 Barlat 屈服准则是板料成形中最常用的理想材料模型，进行有限分分析时，除了输入密度ρ，杨氏模量E，泊松比v等基本材料属性外，还应根据选择的材料模型，输入相关的附加材料参数，如以上两个准则均涉及的各向异性系数r_0，r_{45}和r_{90}。

另外，材料进入塑性状态后，其应力应变呈非线性关系，且并不唯一，而是与加载路径和加载历史有关。这时需要引入合适的流动法则来规定塑性应变增量的分量和应力分量以及应力增量分量之间的关系。当出现塑性形变时，需要采用流动法则来计算塑性变形的大小。塑性应变率为

$$\dot{\varepsilon}_{ij}^{p}=\dot{\lambda}\frac{\partial Q}{\partial\sigma_{ij}} \tag{5.16}$$

其中，$\dot{\lambda}$为塑性率参数，Q为塑性流动势能，$\partial Q/\partial\sigma_{ij}$为塑性流动方向。如果流动方向与屈服面的法向成正比，则认为塑性流动与屈服函数是相关的，否则，认为两者是不相关联的。Von Mises 流动法则是一种常用的关联流动法则，可表示为

$$\dot{\varepsilon}_{ij}^{p}=\dot{\lambda}\frac{\partial f}{\partial \sigma_{ij}} \tag{5.17}$$

或者可以表示为塑性变形率的形式

$$D_{ij}^{p}=\dot{\lambda}\frac{\partial f}{\partial \sigma_{ij}} \tag{5.18}$$

(3)摩擦现象。摩擦是接触碰撞问题中不可避免的物理现象，指两个物体表面接触时产生的阻碍两表面相对切运动的一种现象。在板料成形仿真中，摩擦力对板料的材料流动有很大的影响，直接影响到板料的成形性。常用的摩擦力计算基于经典的库伦摩擦定律，摩擦力计算公式为：

$$F=F_n u \tag{5.19}$$

式中，F_n是作用于物体表面法向合力，对应于接触碰撞问题中的法向接触合力。u为摩擦系数，库伦定律通过来静摩擦系数u_s和动摩擦系数u_d来约定物体间的摩擦运动，并且，u_s大于u_d。当摩擦力小于临界值F_c时，不发生相对运动，式(5.19)中u等于u_s，当大于临界值F_c时，发生相对运动，式(5.19)中u等于u_d，临界值F_c的计算公式为：

$$F_c=F_n u_s \tag{5.20}$$

5.1.2 冲压仿真的模型建立和计算流程

冲压仿真的模型由模具和板料两部分组成。其中，汽车车身结构件的冲压成形所采用的板料通常为金属薄板，在成形仿真的发展历程中，先后有三种不同类型的单元可用于建立板料的有限元模型：膜单元，壳单元和实体单元。其中，膜单元的理论最简单，计算效率高，但是由于没有考虑抗弯作用，计算结果较差。实体单元的计算精度高，但是由于板料在厚度方向上的尺寸限制，会形成很小的临界时间步长，导致计算时间过长，计算成本高。壳单元具有以上两种单元的优势，同时，网格划分也较简单，所以，逐步成为板料成形仿真计算中最常用的单元。

除板料的有限元模型外，模具的有限元模型也是有限元仿真模型的一个重要组成部分，一套成形模具通常包括凹模，凸模和压边圈这三个部分。如上文的分析，如果没有特殊的要求，通常将模具看成为刚体，采用壳单元或者实体单元进行网格划分。本文将模具中的三个部分都设定为可移动的刚性表面，并且，为了简化仿真过程，不考虑模具的磨损等物理性质。在实际使用过种，考虑到模具的型面通常比较复杂，因此可以采用三角形和四边形单元混合的方式进行网格的划分，这样并不影响到程序的计算效率，并且可以极大的缩减前处理的时间，如图5.2 所示为采用混合单元划分的某车内饰件凸模模型。

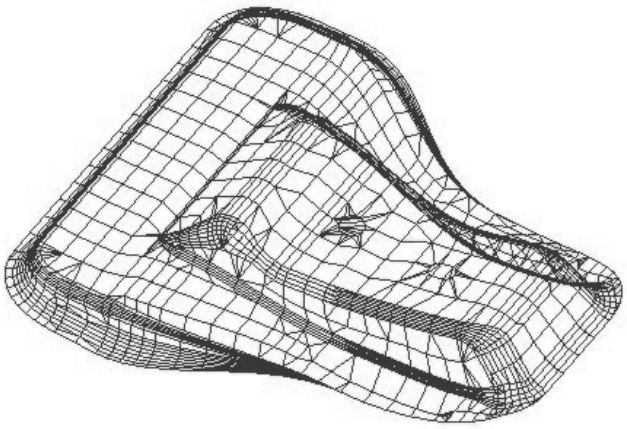


图 5.2 采用混合网格划分的某凸模模型

采用有限元方法进行冲压仿真的流程与通用有限元分析基本一致，冲压仿真流程由以下几种部分组成：

(1) 建立几何模型。根据给定的产品 CAD 等数模建立仿真的几何模型，一个完整的仿真几何模型包含有板料，凹模，凸模和压边圈等四个部分。CAD 模型的建立一般采用 UG 等 CAD 软件完全，本文不详细讨论。

(2) 建立有限元模型。仿真第二步为有限元模型的建立、全部模具和板料的网格划分以及材料和运动属性的定义等。

(4) 有限元求解。建立好有限元模型后，再根据冲压成形工序建立模具和板料的工序和接触关系，即可以提交给求解器分析。

(3) 仿真结果分析。求解结束后，即可以根据输出文件，以图形和动画的方式直观的对计算结果进行分析。

5.2 薄板冲压成形关键算法的 GPU 计算

本文第三章和第四章已经建立了比较好完整的接触碰撞问题的细粒度并行策略，可以很好的适用于 GPU 执行。因此，同样采用显式算法的薄板冲压成形仿真可以借鉴其中大部分的并行策略，所以，本章只对薄板冲压成形仿真中增加和改变的计算流程进行并行化说明，当然，大部分细粒度并行策略的核心是相同的，即 GPU 线程与计算对象的抽象融合。

5.2.1 包含复杂材料模型的单元并行计算技术

薄板冲压成形仿真中常用的时间积分算法同样有隐式和显式两种，隐式积分方法在板料成形中，可以很好的用于回弹计算[187]，因此也有比较广泛的应用，不过仍然避免不了收敛性和计算稳定性的问题。因此，本文沿用 CADEMII 软件的算法，采用显式算法进行冲压过程的计算。

成形过程中，为了真实的仿真材料在塑性大变形过程中材料流动和应力应变关系，除了上节中所述的屈服准则外，还需要选用合适的材料的应力应变关系曲线，材料硬化准则和材料流动准则等材料模型。这些材料模型的引入，极大了增加了有限元计算中单元计算部分的计算复杂度，使单元计算时间增加。如表 5.1 所示，为 CADEMII 软件采用 HILL48 屈服准进行成形仿真时，单个迭代步的平均时间分布。由表可知，由于简化接触算法的引入，以及单元计算复杂度的提升，单元计算成为了主要的耗时部分。因此，减少单元计算时间，将对整个程序计算效率的提升有明显的效果。

表 5.1 成形仿真中单个迭代步的计算时间分布

序号	流程	时间比
1	单元计算	86%
2	接触计算	11%
3	其它	3%

因此，本文基于第三章的研究成果，将 BT 单元和 EST 单元两种单元 GPU 并行计算算法引入到板料成形仿真中，并加入了 HILL48 和 Barlat89 两种屈服准则。由于显式算法优越的并行性，屈服准则计算仍然是以单个单元为对象，因此，加入的屈服准则并不会影响到单元的细粒度并行性。对于屈服的判断可以采用 TFE 或 TFG 的策略，由每个线程独立、并行的对每个单元的当前屈服状态进行计算和判断。

5.2.2 简化接触搜寻算法的 GPU 实现

通过表 4.1 与表 5.1 可知，采用 CADEMII 进行板料成形计算的时间分布与通用接触碰撞问题计算的时间分布有比较大的差异。CADEMII 所采用的简化一接触算法，极大地减少了接触算法所需的时间。但是，由于接触算法涉及到的数据十分广泛，为了降低计算过程中 GPU 和 CPU 间数据交换对程序总计算效率的影响，研究板料成形中的接触算法仍然是十分必要的。

(1) 接触测试对的并行搜寻。在简化的一体化接触搜寻算法中，尽管接触测试对搜寻只进行一次，但是，对于大规模的成形分析，该过程的串行执行会占用比较长的时间。一体化算法中基于子域排序的接触搜寻方法，经常应用于计算机图形学中的实时碰撞检测[188]，可以快速对三维空间中物体进行广域搜索。目前，计算图形学领域中已经发表了许多基于 GPU 的实时碰撞检测算法，因此，本文直接使用 CUDA5.0 自带的 SDK 程序“particels”，通过适当的修改后，实现了 CUDA 下的用于板料成形分析的并行桶式排序接触搜寻方法，具体的执行流程可以参见参考文献[189, 190]。

(2) 接触对的并行更新方法。在简化的接触搜寻算法中，在建立接触对关系后，迭代过程中将只在如图 5.3 所示的相邻接触块中进行。

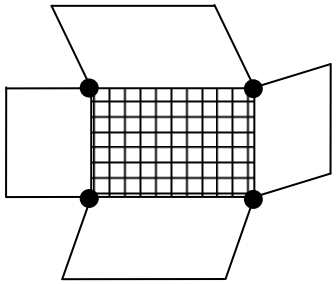


图 5.3 接触块相邻关系

接触块的相邻关系会在显式迭代前搜寻，并复制到 GPU 的全局显存中。在知道接触相邻关系的前提下，接触对的更新同样具有良好的并行性。本文采用一个线程对应一个接触点的方式，判断该接触点对应的接触块是否仍然接触，如果仍满足关系则该线程计算结束，如果当前接触方式已经不满足，则继续与相邻的接触块进行接触检测。如果找到，则更新接触对信息；如果没有找到满足条件的接触块，则与找到距离最近的接触块形成虚拟接触对信息。虚拟接触对不会产生接触力，但是，在下一个迭代步中，虚拟接触对信息仍然参于接触对更新的计算。

5.2.3 考虑摩擦的接触力与压边力并行计算方法

本文采用罚函数的进行薄板冲压成形中的接触力计算，并采用如式(5.19)所示的经典摩擦力计算方法。由于该摩擦力计算方法具有完全的节点独立性，因此，可以直接融入第四章采用的并行罚函数算法中。执行时，并行计算策略没有改变，只需要在计算得到接触力后，再根据接触力的大小，采用 TFN 的策略计算每个发生接触节点的摩擦力。最后，由于摩擦力始终与接触面的法向垂直，因此，还需要采用 TFE 的策略计算当前接触块的法向，将摩擦力按切线方向累加到节点合力矩阵中。

在板料成形过程中，压边力与接触力直接相关，在数值上是板料上所有接触反力之和，其计算过程对应于程序中对一个数组的求和操作。如前文介绍，采用并行缩减的方法可以实现数组的求和，为了简化程序编写，在开发板料成形仿真系统时，本文采用由 CUBLAS 提供的函数来实现。

程序中从接触力计算到压边力计算的代码流程如框 5.1 所示，可见，采用 CUDA 进行并行程序开发时，适当的采用 CUDA 所自带各种数值计算库可以在保证计算效率的同时，极大的提高程序开发效率。

框 5.1 接触力和压边力程序代码

```
…
// 计算接触力
// 1. 计算穿透量 d_ds
CalculateDsGPU<<<numBlocksNode,numThreadsNode>>>(gmdshl,gmsrf3,is,4,gtol,d_x,d_ignd
,ncyc,d_ib,d_tct1,d_rn,d_rnc,d_thkNode,d_ngsg,d_xt,d_ndsg, d_contactGPU,d_ds);
// 2. 由穿透量 d_ds 计算接触力
double dds = gvmax*gdt1;
    CalculateContactForceGPU<<<numBlocksNode,numThreadsNode>>>(gmdshl,gmsrf1,is,gef
,gfu,dds, ncyc, d_tct1, d_nd2,d_x,d_ff, d_rnc, d_fcontact0 ,d_ds, d_alf0,d_ft00,d_ft10,
    d_dut00, d_dut10, d_temp1_0,d_temp2_0,d_temp1_1,d_temp2_1,d_contactGPU,d_ignd);
// 3. 由接触力计算压边力 gblkf
// sumtemp0 为负方向，sumtemp1 为正方向
    cublasDasum(handle,gmdshl,d_temp1_0,1,&sumtemp0);
    cublasDasum(handle,gmdshl,d_temp1_1,1,&sumtemp1);
    gblkf = gblkf - (-sumtemp0 + sumtemp1);
    cublasDasum(handle,gmdshl,d_temp2_0,1,&sumtemp0);
    cublasDasum(handle,gmdshl,d_temp2_1,1,&sumtemp1);
    gblkf = gblkf - (-sumtemp0 + sumtemp1);…
…
```

5.3 基于 GPU 的薄板冲压成形并行计算系统开发

5.3.1 异步数据输出方式

结果数据的输出是有限元分析系统中重要的组成部分，通常包含有中间结果和最终结果两部分。板料成形仿真过程中，由于后处理的需要，计算过程中需要将应力应变值、厚度值和位移值等各类中间结果保存到硬盘文件中。考虑 GPU 计算的异构性，主机与设备间通过 PCI-E 接口的数据传输速度较慢，将造成整个计算程序的暂停等待，对于大规模的计算模型，等待的时间将会非常可观。因此，尽管并不要求每个迭代步都输出中间结果，但对程序整体计算效率的影响还是十分明显的。为此，本文采用异步执行的方式进行中间结果的输出。所谓异步执行，在 CUDA 架构中是指 GPU 操作开始执行后，不用等 GPU 操作结束，CPU 就可以继续进行它的操作，实现两者同时进行各自的操作而互不影响。

CUDA 采用流的概念实现程序的异步执行，同一流中的指令需要按顺序执行，但是不同流中的指令，如 *kernel* 函数和内存复制等操作，在硬件容许的前提下可

以同时执行。图 5.4 所示为本文实现计算结果输出和 GPU 计算异步执行的示例。在 T3 时刻，GPU 执行完 *kernel* 函数后，CPU 会继续执行其它串行计算函数，同时，如果需要，GPU 也同时会将计算结果从 GPU 的全局显存空间复制到内存空间中，并写入到输出文件中。

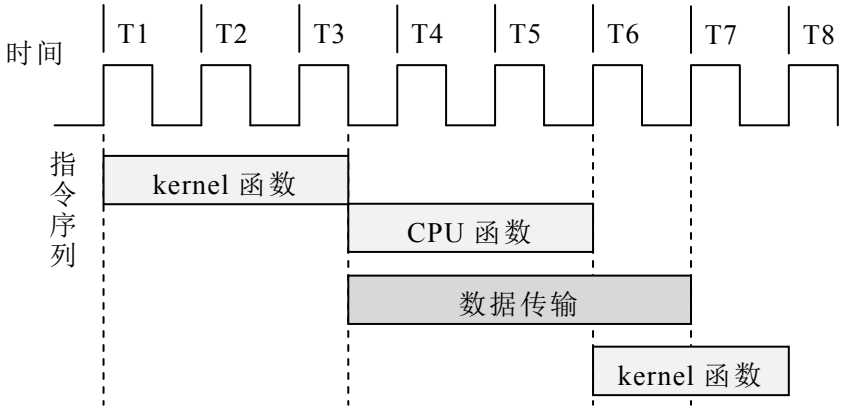


图 5.4 数据传输和 GPU 计算的异步执行模式

框 5.2 基于流的异步执行程序代码

```
…
// 创建流
cudaStream_t stream1,stream2;
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);
for(int i=0;i<step;i++){
cudaMemset(f_d,0,sizeof(double)*NumN);
// GPU 计算
CalculateInnerForecGPU<<<numB,numT,0,stream1>>>(numel,fe_d,f_d);
…
// 将中间计算结果从 GPU 中输出
if(i % iOut ==0){
// 数据复制
cudaMemcpyAsync(f,f_d,sizeof(double)*NumN,cudaMemcpyDeviceToHost,stream2);
// 写到文本文件中
fprintf(watch_d,"%4d,%15.12e\n",i,f[0]);
    }}
…
```

图 5.4 的执行代码如框 5.2 所示，首先建立了两个流，其中，一个流用于一

个迭代过程中所有 kernel 函数的执行，另一个流用于一个迭代计算完成后计算结果的输出。

表 5.2 所示异步执行效率对比，其中，串行执行模型设为每 10 步进行一次数据输出。可以看出，程序执行中数据传输所需的时间得到了有效的隐藏，减少 83.8%，同时，程序总执行时间也减少了 18.8%。因此，可知当对数据输出要求较多时，异步执行方式对提升程序效率有很大的帮助。

表 5.2 异步执行效率

时间	串行执行时间/*s*	异步执行时间/*s*	时间减少量/%
数据传输传输	15.8	2.6	83.8
总时间	70.3	57.1	18.8

5.3.2 计算精度的保证

有限元分析软件的数值结果的稳健性十分重要，现阶段商业显卡的双精度计算能力普遍较差，不仅计算速度上较慢，甚至有部分早期 GPU 对双精度的计算精度都不能完全满足 IEEE 的标准，因此，由于 GPU 的乱序执行本质，会导致明显的计算结果随机性[191]。显式算法中，初始微小的计算误差，在经过数万甚至数十万个迭代累加后，将产生极大的计算误差，本文研究过程就多次遇到计算结果不稳定的问题。通过多次数值实验，本文发现造成这种计算结果不稳定的原因主要有两个：

(1) 浮点数计算的顺序相关性。由于计算机的特点，浮点数计算具有很高的顺序敏感性，不同的计算顺序会产生的不同的计算结果。表 5.3 所示为在 GTX280 显卡执行三个双精度浮点数时产生的计算误差，需要注意的是，表中的三个双精度数并不是随机取值，而是本文在实际有限元分析中出现的一种情况。

表 5.3 浮点数计算的顺序相关性

计算代码	结果	误差

double a = 9.7217471672231739e-003;
double b = 1.6217239847627001e-002;
double c = -9.4916858578941826e-005;
double r1 = (a+b)+c;
double r2 = a+(b+c);
	r1 = 2.5844070156271232e-002;
	r2 = 2.5844070156271236e-002;
		4.0e-17

(2) FMAD 计算误差。早期 GPU 对于类似于 $A\times B+C$ 的操作时，为了提高计算效率，会采用混合乘加(Fused Multiply-ADD,FMAD)的计算模型，这种计算模型会在计算 $A\times B$ 后产生一个有截断误差的乘积再与 C 相加，从而影响了计算精度，如图 5.5 所示。

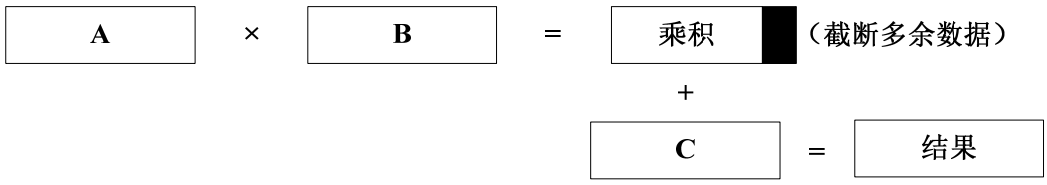


图 5.5 FMAD 计算模式

为了解决这些问题，本文在编写程序时，对于计算精度敏感部位采用 CUDA 提供的内建函数代替传统的符号计算函数。内建函数可以使编译器按程序员的主观意思进行程序编译，例如：乘加操作不再编译成为 FMAD 操作，同时，也有利于保证计算顺序。比如，“+”计算符在程序就由内建函数*__dadd_rn(double x, double y)*代替，“*”计算符则由内建函数*__dmul_rn(double x, double y)*代替。

表 5.4 所示为采用内建函数进行乘加计算的数值算例。由表所示，当采用计算符表达式时，CPU 的计算结果 r_CPU 和 GPU 的计算结果 r_GPU 并不相同。但是，当在 GPU 中使用内建函数时，计算结果 r_IF 与 r_CPU 的计算结果一致。

表 5.4 使用内建函数进行计算

计算代码	结果
double a = 1.1564987445311563 2115e-15;	
double b = 2.3156875135486112 5466e-5;	
double c = 3.23614523287643e2;	f_CPU=2.9975604431208402e-2
r_CPU=a+b*c+b*c+b*c+b*c;	f_GPU=2.9975604431208405e-2
r_GPU=a+b*c+b*c+b*c+b*c;	f_IF =2.9975604431208402e-2
r_IF=a+__dmul_rn(b,c)+__dmul_rn(b,c)+	
__dmul_rn(b,c)+ __dmul_rn(b,c);	

5.3.3 基于 OpenGL 的实时显示技术

有限元分析系统中的实时显示技术可以使软件使用者在计算过程中实时观测到当前的计算状态，具有重要的应用价值。GPU 通用计算过程中，计算数据存储于显存空间内，因此，通过 CUDA 与 OpenGL 等图形支持库间的相互操作，可以方便的实现科学计算可视化[192]。本文实现 CUDA 计算时的实时显示技术采用的基本原理是使用 CUDA 生成数据，然后使用 OpenGL 在屏幕上绘制出数据所表示的图形。

从 CUDA 编程的角度出发，OpenGL 在 GPU 上创建并管理的内存区域通为缓存对象，基中 CUDA 可以使用的缓存对象主要有两种 PBO(Pixel Buffer Object，像素缓冲对角)和 VBO(Vertex Buffer Object，顶点缓冲区对象)。其中，PBO 是存储像素的空间，一般用于 2D 图像的显示，CUDA 可以映射到 PBO，在 PBO 中生

成或修改像素，再由 OpenGL 显示这些像素。VBO 是 OpenGL 用于存储 3D 向量的一段内存，CUDA 程序映射到 VBO 后，可以生成或修改 3D 位置信息，之后 OpenGL 可以将这些网格渲染成彩色表面、3D 图像等[193]。

图 5.6 所示为 CADEM-GPU 中 CUDA 与 OpenGL 交互程序流程图，图中大部分流程与常规可视化程序一样，只是需要通过 *cudaGraphicsGLResgisterBuffer()* 将 VBO 注册为 CUDA 可以访问的数据。同时，每个迭代步都需要将计算所得到的模具和板料的新的节点位置信息通过 *cudaGLMapBufferObject()* 函数映射到 VBO 中，这样，新的节点位置就可以由 OpenGL 直接使用并渲染显示出来。

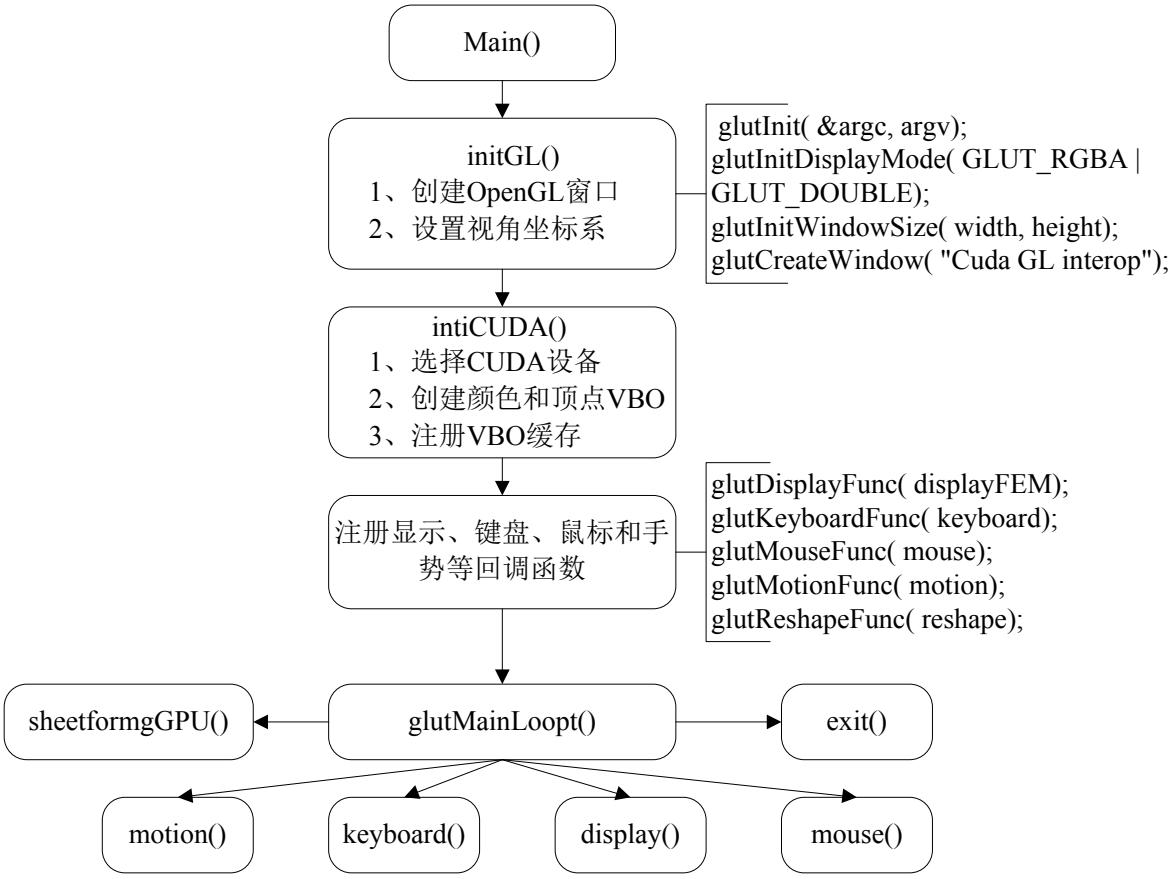


图 5.6 CUDA 与 OpenGL 交互程序流程图

图 5.7 所示为本文在仿真系统中实现的实时显示界面。在板料成形仿真程序中加入可视化，可以实时观察模具的运动情况，确保模具的正常动作。同时，可以实时观察板料的成形质量，当有程序运动错误或者有板料起皱、拉裂等成形缺陷出现时，可以马上终止程序运行。本文开发的界面具有一定的人机交互功能，可以选择需要显示的部件，改变部件的显示方式(渲染或网络)，还可以进行放大、旋转等操作。并且可以实时显示当前显式迭代状态，如当前时间步长大小，当前仿真的时间节点以及迭代步数等信息。

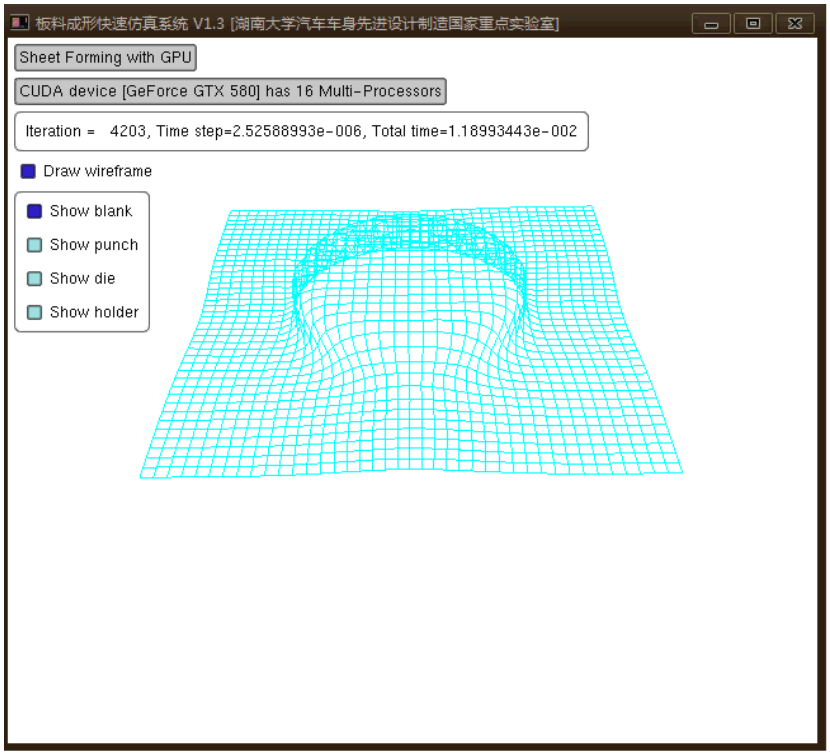


图 5.7 成形仿真实时显示

5.3.4 并行仿真系统的整体计算流程

基于以上的算法分析以及异构计算平台的特点，本文对原 CADEMII 软件的计算流程进行了一定的调整和改进，建立了适合于 GPU 并行计算的软件流程，如图 5.8 所示，并给出了每步详细的计算方法。并且，开发了 CADEMII 的 GPU 并行计算版本 CADEM-GPU(软件著作权编号：2010SR052426)。

该计算流程基于本文所建立的通用显式有限元 GPU 并行计算平台，采用了由 CPU 进行主控制运算，包括计算时间结点控制、模具位置判断以及冲压工序控制等，由 GPU 进行有限元单元计算、节点的数学物理量求解以及接触算法计算等，并加入实时显示技术等。

在图中的计算流程中，灰色的流程块代表采用 GPU 进行并行计算的部分，不同的计算部分根据线程计算单元的不同以及计算规模自动计算最佳的执行参数，以保证最佳的计算效率。这种异构的计算模式在最大化减少数据传输的同时也可以充分利用硬件资源，有效保证计算效率。

在输入输出文件方面，本程序所采用的输入文件可以直接从商业有限元前处理软件 HyperMesh 中导出。同时，支持输出用于自主开发的 CADEM 后处理软件使用的格式，以及支持输出用于 Tecplot 使用的三维图形数据以及 HyperGraph 使用的各类动态响应曲线数据。

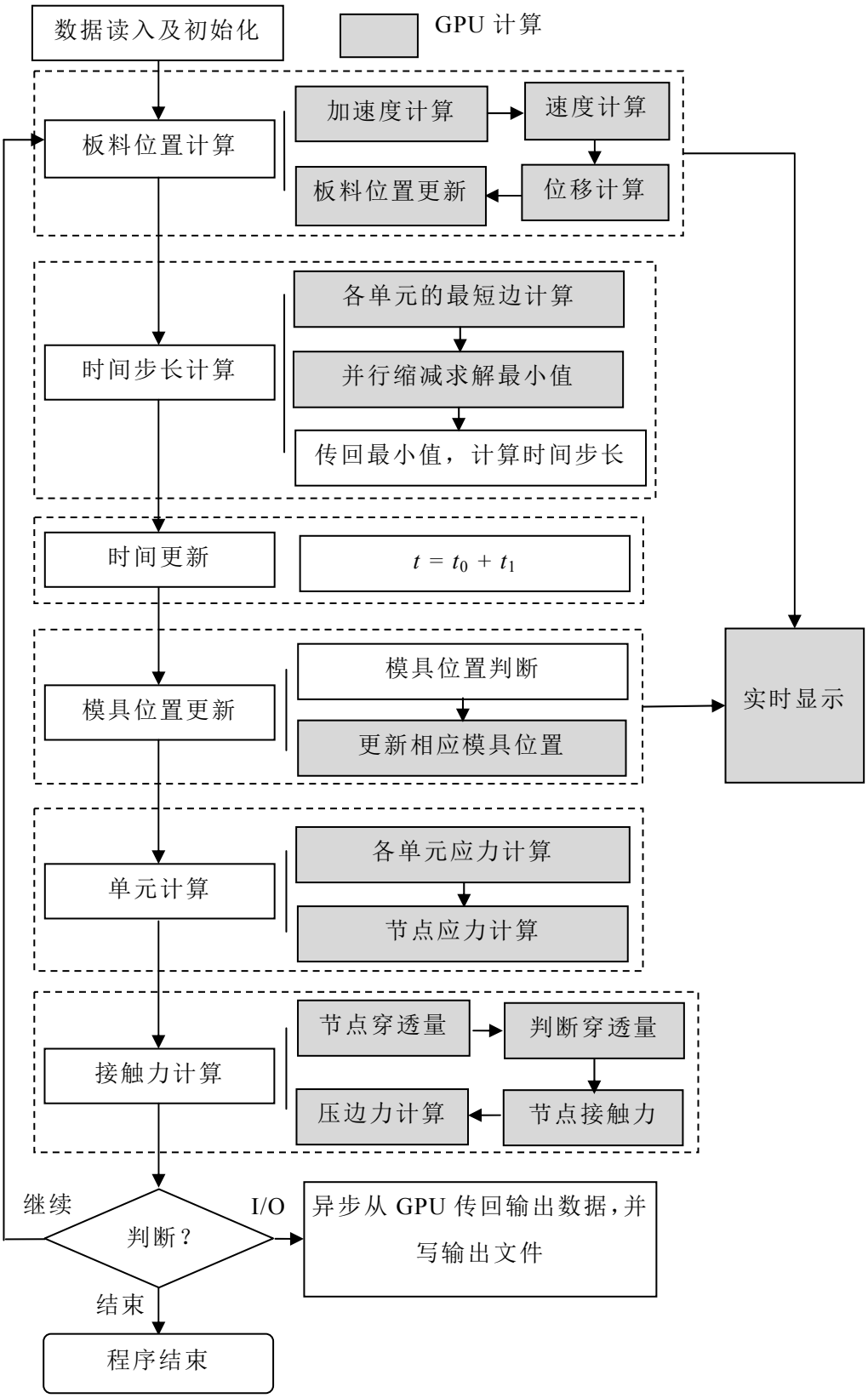


图 5.8 基于 GPU 的薄板冲压成形并行计算流程

5.4 数值算例及分析

本节对本文提出的并行计算方法和开发的板料成形仿真系统进行验证，测试平台如表 5.5 所示。

表 5.5 计算平台

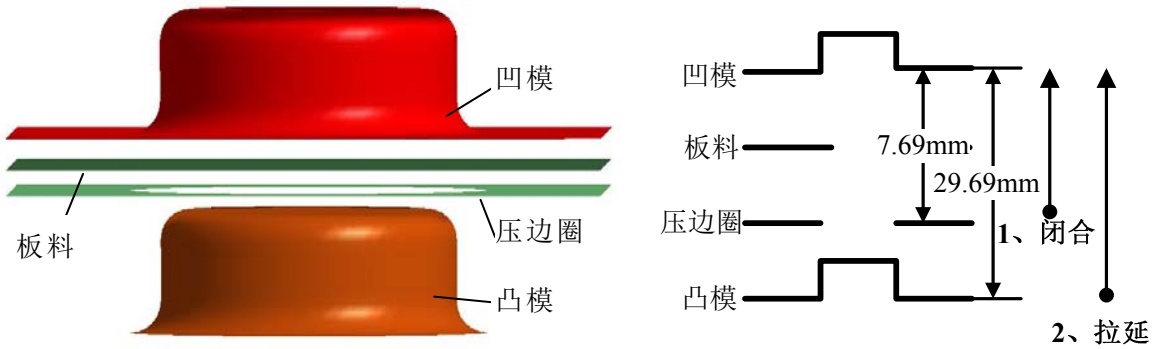
名称	型号	主要性能参数
CPU	酷睿 2 四核 Q8200	主频：2.33GHz，4GB 内存
GPU	NVIDIA GTX 460	384 个 CUDA 核，核心频率:675Mhz，1Gb 的显存
操作系统	Windows 7 32 位	
开发环境	Microsoft VC++ 2010, CUDA 4.0	

其中，Q8200 和 GTX460 目前的市场价格均在 1000 元左右，是目前个人计算机的主流配置，因此，具体较好的代表性。

5.4.1 数值算例

5.4.1.1 杯突成形分析

首先采用如图 5.9 所示杯突成形模型对 EST 三角形单元在成形仿真中的计算效率和精度进行分析。计算模型中，板料厚度为 1mm，厚度方向上为 3 个积分点，冲压速度为 5m/s。板料采用弹塑性的材料本构，HILL48 屈服准则，具体材料参数为：杨氏模量 E=10.5×10^6 Mpa，泊松比 v=0.3，弹性模量 E_p=0.22×10^6 Mpa，密度为 2.45×10^{-6} kg/mm^3，屈服应力 σ_y=24000 Mpa。



(a) CAE 模型　　(b) 模拟工序

图 5.9 杯突成形的计算模型

模型采用 EST 三角形单元建模后，分别采用 CADEM-GPU 和 CADEMII 软件在 GPU 和 CPU 中进行计算，计算结果对比如图 5.10 和图 5.11 所示。其中，图 5.10 所示为 CPU 和 GPU 计算所得板料成形后厚度分布云图，由图可知，两种方

式所得的厚度云图分布完全一致。另外，图 5.11 给出了网格变形对比，根据该计算模型的对称性，可知两种方式所得的网格变形也保持了很好的一致性。

(a) CPU 计算结果

(b) GPU 计算结果

图 5.10 CPU 和 GPU 计算的板厚对比

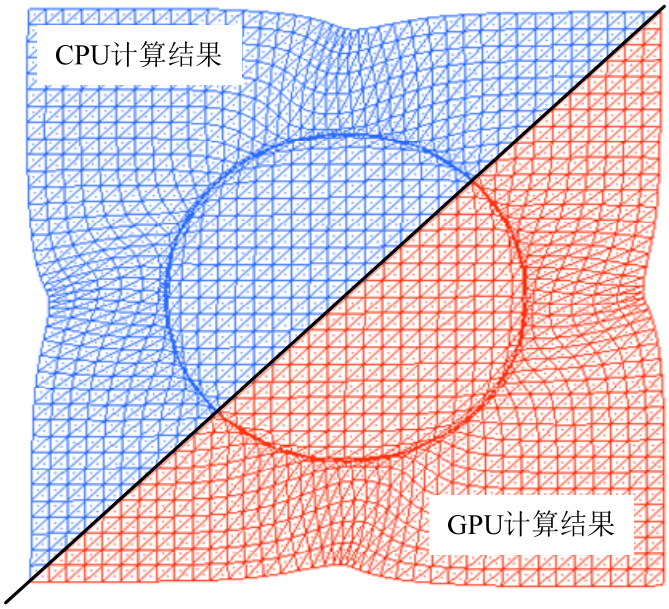


图 5.11 CPU 和 GPU 计算的网格变形对比

CPU 和 GPU 的计算时间和计算加速比如表 5.6 所示。可知，采用 GPU 计算

的时间远低于 CPU 计算的时间，并且，随着单元数目的增加，计算加速比逐渐增大，最大可以达到 37 倍的计算加速比，并逐渐平稳，这符合第三章对计算加速比发展趋势的研究结果。

表 5.6 EST 单元成形的计算效率比较

单元数/个	CPU 耗时/s	GPU 耗时/s	加速比
800	176.370	31.972	5.5
3200	2609.139	120.252	21.7
6400	8576.550	252.360	34.0
12000	18405.620	522.334	35.2
24000	40215.360	1095.996	36.7

5.4.1.2 车顶盖成形仿真

该算例采用 BT 四边形单元对某汽车的车顶盖进行成形仿真。仿真模型如图 5.12 所示。板料采用 BT 四边形单元进行网格划分，同样采用弹塑性的材料本构，HILL48 屈服准则，板料厚度为 0.8mm，具体材料参数为：杨氏模量 E=10.5×10^6 Mpa，泊松比 v=0.3，弹性模量 E_p=0.22×10^6 Mpa，密度为 2.45×10^{-6} kg/mm^3，屈服应力 σ_y=24000 Mpa。

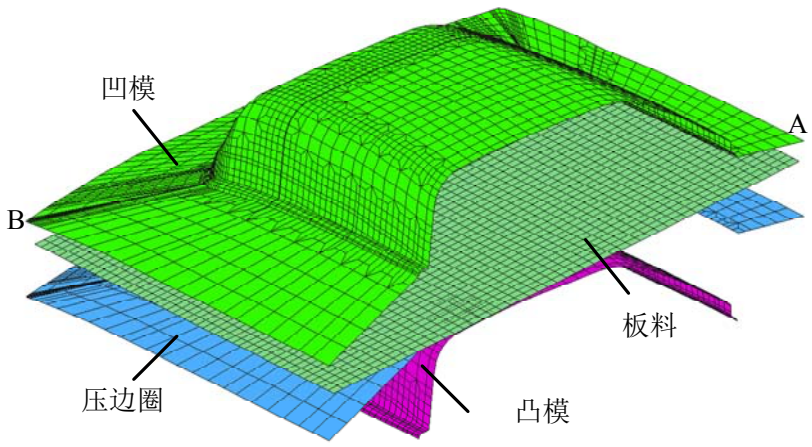


图 5.12 车顶盖成形仿真模型

首先对计算结果进行对比，图 5.13 所示为成形后板料截面上厚度对比，截面线由图 5.12 中的 A 点到 B 点构成，取 100 个点。同时，图 5.14 所示为板料上某点在仿真过程中节点合力的动态响应曲线对比。如图可知，截面线上厚度变化完全一致，同时，节点合力的动态响应曲线也完全重合，因此，可以证明采用 GPU 对基于 BT 四边形单元的析料成形模型进行计算同样可以取得很好的计算精度，计算结果与 CPU 一致，满足实际工程对计算精度的要求。

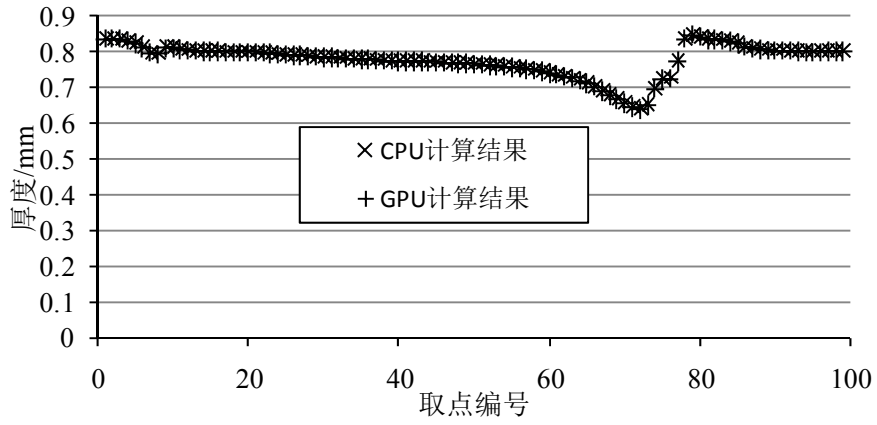


图 5.13 成形后板料截面上的厚度对比

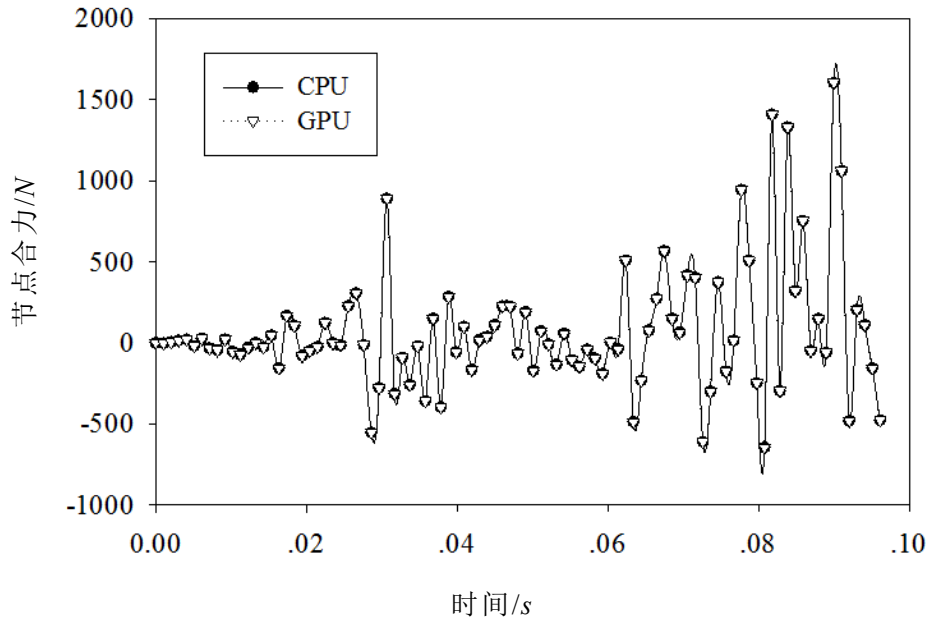
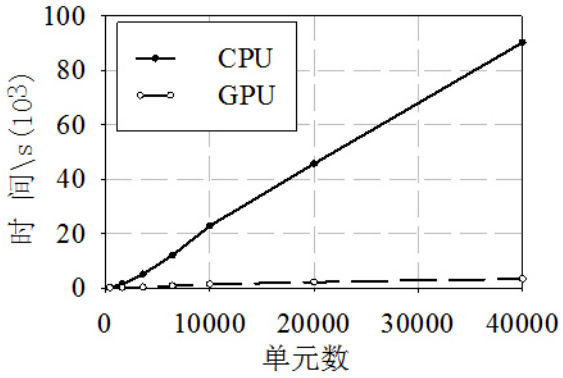


图 5.14 板料上某节点的节点合力动态响应曲线对比

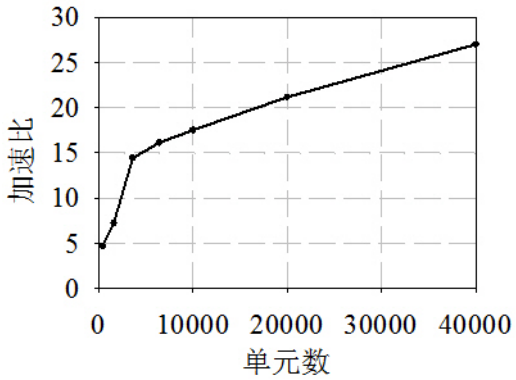
其次，对计算效率进行分析，CPU 和 GPU 计算的总时间如图 5.15(a)所示，可知 GPU 计算的时间要远少于 CPU 的计算时间，可以明显提高计算效率。不同计算规模下 GPU 计算所得的计算加速比如图 5.15(b)所示，计算加速比同样与计算规模成正比关系，图中，对于 40000 个网格的计算模型，可以取得 27 倍的加速比。

5.4.2 结果分析

通过以上的算例分析可知，本文提出的板料成形并行计算方法在引入到 CADEMII 软件形成 CADEM-GPU 软件后，具有较高的计算精度，计算结果与原 CADEMII 软件的计算结果完全一致。无论采用 EST 三角形单元还是采用 BT 四边



(a) 计算时间对比



(b) 加速比

图 5.15 基于 BT 单元的成形仿真的计算效率对比

形单元在 GPU 上进行薄板冲压成形仿真时，在取得与 CPU 计算完全一致的仿真结果的前提下，计算的时间明显缩短，有效提升了计算效率。

5.5 本章小结

本章基于自主开发的板料成形软件 CADEMII 研究了基于 GPU 的薄板冲压成形并行计算方法。在第三章和第四章的研究基础上，针对薄板成形的部分特殊性，对基本接触碰撞问题有限元分析的并行进行了一定的修改和扩充，形成了具有高效、高精度特征的冲压成形并行计算方法：

(1) 提出了包含复杂材料模型的单元并行计算技术。

(2) 对 CADEMII 软件采用的简化一体化算法进行了并行化分析，并实现了其全流程的 GPU 执行。

(3) 提出了包含摩擦力的并行接触力计算方法，以及采用 CUBLAS 编程的压边力并行计算方法。

在以上并行算法的基础上，根据 GPU 的异构计算的特点，对在 GPU 通用计

算平台上进行软件开发遇到的数据输出和计算精度的问题进行了说明，具体如下：

(1) 由于 GPU 计算的异构性，通过 PCI-E 接口的数据传输速度较慢，大量、频繁的数据交换会降低程序计算效率。因此，本文提出了基于流的异步数据输出方式，成功隐藏了数据输出对计算效率的影响。

(2) 由于 GPU 的双精度浮点数计算能力有限以及并行计算的无序性，有时无法保证有限元的计算精度。本文提供采用内建函数代替传统计算符的程序编写方式，使编译器可以按自己的主观意思进行程序编译，从而保证计算结果的稳健性。

(3) 有限元计算的实时显示有较高的实用价值。根据 GPU 的计算特点，GPU 计算过程中计算数据均存储在显存上，可以直接渲染和显示，因此，本文提出了基于 OpenGL 的 GPU 计算过程中结果的实时显示方法。

最终，开发了薄板冲压成形并行计算软件 CADEM-GPU (软件著作权编号：2010SR052426)，实现了采用 GPU 进行板形成形模拟的全流程并行计算。通过数值算例证明，该软件可以有效的提高板料成形仿真的计算效率，采用 GTX460 进行计算，对于近 25000 个网格组成的 EST 三角形模型，可以取得近 37 倍的计算加速比，对于近 40000 个网络组成的 BT 四边形模型，可以取得近 27 倍的计算速比。

由于依托项目的需求，该系统由国家应用软件产品质量监管检验中心进行了检验(软件检测编号：G21101901)，证明了软件的稳定性，并具有较高的计算效率。

结论与展望

汽车车身结构接触碰撞过程有限元分析中，对计算精度和计算效率的双重追求一直是研究领域和工程领域中存在的主要矛盾。并行计算方法是调和这种矛盾的有力手段，但是传统的并行计算方法需要有昂贵的硬件支持，使用流程复杂，最终取得的加速比也非常有限。立足于这些问题，本文提出采用通用图形处理器进行板壳结构非线性有限元的并行计算，主要研究汽车碰撞仿真和薄板冲压成形问题的高效求解方法，实现了此类问题在普通个人计算机上的快速计算，为汽车结构接触碰撞过程有限元仿真技术在实际工程中的应用提供了全新的、高效的且具有极高性价比的并行计算方法。

1 本文主要创新点

本文的研究开始于 GPU 通用计算领域刚刚兴起的时期，因此，本文的研究内容绝大多数属于开创性的工作，具体的创新点如下：

(1) 考虑到非线性显式有限元天然的可并行性以及 GPU 的轻量级线程执行模式，开发了具有自主知识产权的基于 GPU 的显式有限元计算平台。其主要特点在于：建立了线程与单元、线程与节点、线程与自由度三种层次的抽象映射方法，使显式有限元计算与 GPU 线程完美融合。同基于网格分区的粗粒度有限元并行策略相比，该细粒度并行策略没有任何前处理过程，在单块显卡也不存在边界数据处理问题，因此计算效率大幅度提升。

(2) 针对单元计算中节点应力组装在 GPU 平台上难以并行化的技术瓶颈，提出了预索引并行应力组装策略，实现了 BT 四边形单元和 EST 三角形单元两种壳单元在 GPU 上的细粒度并行。提出了 GPU 上基于并行缩减算法的时间步长等单值并行求解方法。实现了显式有限元算法在 GPU 上的全过程计算，减少了 GPU 与 CPU 间数据交换的同时，使程序的计算效率达到最佳化。

(3) 提出了包含并行级域接触搜寻算法、并行防御节点接触力计算方法和并行罚函数接触力计算方法在内的全流程 GPU 执行的细粒度并行接触算法。级域算法是一种适用于复杂自接触问题的高效搜寻算法，其同一级内接触块的计算独立性也符合 GPU 细粒度计算的要求。在测试对搜寻阶段，本文采用线程与接触块一一映射的细粒度并行策略用于接触块扩展域和接触域的计算，并采用 GPU 并行排序的方法快速完成扩展域和接触点的混合排序。通过对 GPU 线程计算粒度的微小提升，采用线程与表元素一一映射的方法实现了排序中测试对的并行搜寻。在接触对搜寻阶段，本文采用线程与测试对一一映射的方式实现了同一级内接触对的

并行搜寻，并采用计算后排序的方式实现上一级与下一级间的数据交换。在接触力计算阶段，本文采用线程与接触对一一映射的策略给出了穿透量和接触力细粒度并行方式，并采用原子操作来实现接触力的并行离散。

(4) 本文将 GPU 并行计算引入到薄板冲压成形有限元计算中，提出了完整的薄板冲压成形 GPU 并行计算方法。提出了适用于 GPU 并行计算的包含复杂材料本构计算的单元并行计算技术以及包含摩擦力计算的接触 GPU 并行计算方法。自主开发的板料成形软件 CADEMII 采用简化的一体化接触搜寻算法极大的降低冲压成形仿真中接触算法的计算时间，为此，本文研究了该简化接触算法在 GPU 上的实现方法：在测试对形成阶段，引入了计算机图形学中用于实时碰撞检测的广域搜寻方法，具有较高的计算效率。并在建立了相邻接触块信息的前提下，给出了接触后搜寻中并行接触对更新方法。

2 进一步研究展望

本文的工作不仅仅在于构造适用于 GPU 并行计算的接触碰撞过程有限元算法，更是将目标定于开发具有自主知识产权的汽车 CAE 分析软件。由于时间的限制，以及个人在软件开发能力上的限制，本文目前所形成的相关有限元分析系统在功能上比较单一，在使用便利性上也不够完善。基于本论文的现有研究成果和工程应用中的需求，作者认为未来可以进一步开展的研究工作有：

(1) 本文目前的研究主要是针对壳单元进行，实际上，汽车整车碰撞仿真过程中还会经常用到杆单元和各类实体单元等，因此，研究这类单元的 GPU 并行计算方法是十分必要的。

(2) 本文在研究过程中发现，级域法在测试对搜索阶段采用的混合后排序的方法虽然相对比于原程序在计算效率上有比较大的提升。但是，这种方法并不适用于在 GPU 上的并行执行，因为，搜寻过程中除了长边方向外，还有另外两个方向需要采用串行的方法执行。目前，国内外也没有发表专门针对机械系统，基于 GPU 的高精度并行接触搜寻方法。因此，可以进一步研究专门适用于 GPU 并行执行的接触搜寻方法。

(3) 目前本文的研究均是基于单块显卡，受于显存空间的限制，导致计算规模也受到限制。同时，考虑到目前国内外有较多基于多 GPU 的异构并行计算平台的研究，硬件平台比较成熟，因此，有必要将本文提出的并行计算方法与 OpenMP 和 MPI 等传统并行计算方法相结合，提出适用多 GPU 的并行算法，以提高计算规模，进一步提升算法的实用性。

(4) 完善所开发的 CAE 软件，建立更加方便的前后处理接口，以易于与相关商业软件配套使用。更可以开发自主的前后处理软件，形成一套完整 CAE 分析系统，并将其推广应用。

参考文献

[1] CHEN C J, USMAN M. Design optimisation for automotive applications. International Journal of Vehicle Design, 2001, 25(1): 126-141.

[2] ONDA H, SAKANASHI T, MIHARA K, et al. Casting CAE for automotive casting parts. Nissan Technical Review, 2003, 52: 21-26.

[3] LAKSHMINARAYAN V, WANG H, WILLIAMS W, et al. Application of CAE nonlinear crash analysis to aluminum automotive crashworthiness design. SAE Technical Paper 951080, 1995.

[4] 王勖成. 有限单元法. 第一版. 北京: 清华大学出版社有限公司, 2003, 117-120.

[5] THOLE C A, STUBEN K. Industrial simulation on parallel computers. Parallel Comput, 1999, 25(13-14): 2015-2037.

[6] NCAC. Finite Element Model Archive, http://www.ncac.gwu.edu/vml/models.html, 2011-12-1.

[7] TOMOV S, MCGUIGAN M, BENNETT R, et al. Benchmarking and implementation of probability-based simulations on programmable graphics cards. Computers & Graphics, 2005, 29(1): 71-80.

[8] ALMASI G S, GOTTLIEB A. Highly parallel computing. Redwood City: Benjamin-Cummings Publishing Co., 1989, 106-110.

[9] NVIDIA C. Compute unified device architecture programming guide, 2007, 1-5.

[10] CLOUGH R W. Thse Finite Element Method in Plane Stress Analysis. In: ASCE 2nd Conference on Electronic Compuataion. Reston: American Society of Civil Engineers, 1960, 345-378.

[11] HRENNIKOFF A. Solution of problems of elasticity by the framework method. Journal of Applied Mechanics, 1941, 8(4): 169-175.

[12] MCHENRY D. A lattice analogy for the solution of plane stress problems. Journal of Insitute of Civil Engineers, 1943, 21(2): 59-82.

[13] COURANT R. Variational methods for the solution of problems of equilibrium and vibrations. Bull Amer Math Soc, 1943, 49(1): 1-23.

[14] TURNER M J, CLOUGH R W, MARTIN H C, et al. Stiffness and deflection analysis of complex structures. Journal of Aeronautical Society, 1956, 23(9): 805-823.

[15] MARCAL P V, KING I P. Elastic-Plastic Analysis of Two-dimensional Stress Systems by the Finite Element Method. International Journal of Mechanical Sciences, 1967, 9: 143-155.

[16] YAMADA Y, YISHIMURA N, SAKURAI T. Plastic Stress-Strain Matrix and its Application for the Solution of Elastic-Plastic Problems by the Finite Element Method. International Journal of Mechanical Sciences, 1968, 10: 343-354.

[17] HIBBITT H D, MARCAL P V, RICE J R. A finite element formulation for problems of large strain and large displacement. International Journal of Solids and Structures, 1970, 6(8): 1069-1086.

[18] MCMEEKING R M, RICE J R. Finite-element formulations for problems of large elastic-plastic deformation. International Journal of Solids and Structures, 1975, 11(5): 601-616.

[19] OH S I, ALTAN T. Metal forming and the finite-element method. Oxford: Oxford University Press, 1989: 210-216.

[20] REBELO N, NAGTEGAAL J, HIBBITT H. Finite element analysis of sheet forming processes. International journal for numerical methods in engineering, 1990, 30(8): 1739-1758.

[21] LI J, YANG Q, NIU P, et al. Analysis of Thermal Field on Integrated LED Light Source Based on COMSOL Multi-physics Finite Element Simulation. Physics Procedia, 2011, 22: 150-156.

[22] REDDY J. Nonlinear finite element analysis. Oxford: Oxford University Press, 2004: 90-93.

[23] LEE S H. A CAD–CAE integration approach using feature-based multi-resolution and multi-abstraction modelling techniques. Computer-Aided Design, 2005, 37(9): 941-955.

[24] CLOUGH R W, WILSON E L. Early finite element research at Berkeley. In: Proceedings of the Fifth US National Conference Computational Mechanics. Boulder, 1999, 1-35.

[25] WILSON E L. SAP: A General Structural Analysis Program, Report to Walla Walla District U.S. Engineers Office. Structural Engineering Laboratory, University of California, 1970.

[26] BATHE K J, WILSON E L, IDING R H. NONSAP: a structural analysis program for static and dynamic response of nonlinear systems. Springfield, VA: National Technical Information Service, 1974, 3-5.

[27] 袁明武, 陈璞. 微机结构分析通用程序 SAP84(版本 4.0). 计算结构力学及其应用, 1995, 12(3): 298-300.

[28] 张早明. CAE 在汽车工业中的应用. 汽车科技, 2008, 5: 7-11.

[29] 张洪武,陈飙松,李云鹏,张盛,彭海军.面向集成化 CAE 软件开发的 SiPESC 研发工作进展. 计算机辅助工程, 2011, 20(2): 39-49.

[30] NOOR A. Parallel processing in finite element structural analysis. In: Parallel computations and their impact on mechanics. USA: NASA, 1987, 253-277.

[31] NOOR A K, LAMBIOTTE JR J J. Finite element dynamic analysis on CDC STAR-100 computer. Computers & Structures, 1979, 10 (1–2): 7-19.

[32] NOOR A K, PETERS J M. Element stiffness computation on CDC CYBER 205 computer. Communications in Applied Numerical Methods, 1986, 2(3): 317-328.

[33] NOOR A K, KAMEL H A, FULTON R E. Substructuring techniques—status and projections. Computers & Structures, 1978, 8(5): 621-632.

[34] FARHAT C, WILSON E. A new finite element concurrent computer program architecture. International journal for numerical methods in engineering, 1987, 24(9): 1771-1792.

[35] FARHAT C. A simple and efficient automatic FEM domain decomposer. Computers & Structures, 1988, 28(5): 579-602.

[36] FARHAT C, LESOINNE M. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. International journal for numerical methods in engineering, 1993, 36(5): 745-764.

[37] FARHAT C, ROUX F-X. An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems. SIAM Journal on Scientific and Statistical Computing, 1992, 13(1): 379-396.

[38] HUGHES T J, LEVIT I, WINGET J. Element-by-element implicit algorithms for heat conduction. Journal of Engineering Mechanics, 1983, 109(2): 576-585.

[39] BARRAGY E, CAREY G F. A parallel element‐by‐element solution scheme. International journal for numerical methods in engineering, 2005, 26(11): 2367-2382.

[40] CAREY G, BARRAGY E, MCLAY R, et al. Element‐by‐element vector and parallel computations. Communications in Applied Numerical Methods, 1988, 4(3): 299-307.

[41] 李晓梅, 程建钢, 李明瑞. 并行计算环境与有限元分析并行算法. 中国科学基金, 1995, 9(1): 15-21.

[42] 周树荃, 梁维泰, 邓绍忠. 有限元结构分析并行计算. 北京: 科学出版社, 1997, 15-20.

[43] 周树荃, 邓绍忠. 有限元结构分析并行计算的若干研究进展. 南京航空航天大学学报, 1995, 27(1): 27-32.

[44] 周树荃, 邓绍忠. 变带宽大型稀疏线性方程组的并行直接解法及其在 YH-1 上的实现. 在：航空科学基金论文集(3). 北京:航空工业出版社, 1993, 216-220.

[45] 邓绍忠, 周树荃. 不规则结构分析有限元方程组的并行迭代解法及其实现. 在：全国第三届并行算法学术会议论文集. 武汉: 华中理工大学出版社, 1992, 116-120.

[46] 张汝清. 并行计算结构力学. 重庆大学出版社, 1993, 2-5.

[47] 张汝清. 并行计算结构力学的发展和展望.力学进展, 1994, 24(4): 511-517.

[48] BATHE K J, RAMM E, WILSON E L. Finite element formulations for large deformation dynamic analysis. International Journal for Numerical Methods in Engineering, 1975, 9(2): 353-386.

[49] BATHE K-J, BOLOURCHI S. A geometric and material nonlinear plate and shell element. Computers & structures, 1980, 11(1): 23-48.

[50] KEY S W. A finite element procedure for the large deformation dynamic response of axisymmetric solids. Computer Methods in Applied Mechanics and Engineering, 1974, 4(2): 195-218.

[51] HUGHES T, LIU W. Implicit-explicit finite elements in transient analysis. I-Stability theory. II- Implementation and numerical examples. Journal of Applied Mechanics(ASME Transactions), 1978, 45: 371-378.

[52] HUGHES T J, PISTER K S, TAYLOR R L. Implicit-explicit finite elements in nonlinear transient analysis. Computer Methods in Applied Mechanics and Engineering, 1979, 17: 159-182.

[53] BELYTSCHKO T, LIN J I, CHEN-SHYH T. Explicit algorithms for the nonlinear dynamics of shells. Computer methods in applied mechanics and engineering, 1984, 42(2): 225-251.

[54] HUGHES T J, TAYLOR R L, SACKMAN J L, et al. A finite element method for a class of contact-impact problems. Computer Methods in Applied Mechanics and Engineering, 1976, 8(3): 249-276.

[55] HALLQUIST J, GOUDREAU G, BENSON D. Sliding interfaces with contact-impact in large-scale Lagrangian computations. Computer Methods in Applied Mechanics and Engineering, 1985, 51(1): 107-137.

[56] ZHI-HUA Z, NILSSON L. A contact searching algorithm for general contact problems. Computers & Structures, 1989, 33(1): 197-209.

[57] ZHONG Z-H. Finite element procedures for contact-impact problems. Oxford: Oxford university press, 1993, 345-360.

[58] 钟志华, 李光耀. 薄板冲压成型过程的计算机仿真与应用. 北京: 北京理工大学出版社, 1998, 72-75.

[59] ZHONG Z-H, NILSSON L. A unified contact algorithm based on the territory concept. Computer Methods in Applied Mechanics and Engineering, 1996, 130(1): 1-16.

[60] BENSON D J, HALLQUIST J O. A single surface contact algorithm for the post-buckling analysis of shell structures. Computer Methods in Applied Mechanics and Engineering, 1990, 78(2): 141-163.

[61] OLDENBURG M, NILSSON L. The position code algorithm for contact searching. International journal for numerical methods in engineering, 1994, 37(3): 359-386.

[62] BELYTSCHKO T, NEAL M O. Contact‐impact by the pinball algorithm with penalty and Lagrangian methods. International journal for numerical methods in engineering, 1991, 31(3): 547-572.

[63] CHAMORET D, SAILLARD P, RASSINEUX A, et al. New smoothing procedures in contact mechanics. Journal of Computational and applied Mathematics, 2004, 168(1): 107-116.

[64] K Yamazaki. M Mori. Analysis of an elastic contact problem by the boundary element method (Anapproach by the penalty function method). JSME Int. J. Series 1, 1989, 32(4): 508-513.

[65] BATHE K J, CHAUDHARY A. A solution method for planar and axisymmetric contact problems. International Journal for Numerical Methods in Engineering, 1985, 21(1): 65-88.

[66] KAMAL M M. Analysis and simulation of vehicle to barrier impact. SAE Technical Paper 700414, 1970.

[67] TANI M. Study of Automobile Crashworthiness. SAE Technical Paper 700175, 1970.

[68] CARRUTHERS J J, KETTLE A, ROBINSON A. Energy absorption capability and crashworthiness of composite material structures: A review. Applied Mechanics Reviews, 1998, 51(10): 635-649.

[69] LANGSETH M, HOPPERSTAD O, BERSTAD T. Crashworthiness of

aluminium extrusions: validation of numerical simulation, effect of mass ratio and impact velocity. International Journal of Impact Engineering, 1999, 22(9): 829-854.

[70] JACOB G C, FELLERS J F, STARBUCK J M, et al. Crashworthiness of automotive composite material systems . Journal of applied polymer science, 2004, 92(5): 3218-3225.

[71] ZAREI H, KR GER M, ALBERTSEN H. An experimental and numerical crashworthiness investigation of thermoplastic composite crash boxes. Composite structures, 2008, 85(3): 245-257.

[72] PERRONE N. Crashworthiness and biomechanics of vehicle impact. Washington, D.C. : Catholic University of America, 1970, 10-13.

[73] AMBR SIO J, SILVA M. Structural and biomechanical crashworthiness using multi-body dynamics. Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 2004, 218(6): 629-645.

[74] HU K, HU P. Parameters Optimization Method for Hot Forming Panels Based on the Crashworthiness and Lightweight. Applied Mechanics and Materials, 2013, 327: 318-321.

[75] LONSDALE G, CLINCKEMAILLIE J, VLACHOUTSIS S, et al. Communication requirements in parallel crashworthiness simulation. Lecture Notes in Computer Science, 1994, 796: 55-61.

[76] BELYTSCHKO T, PLASKACZ E, CHIANG H-Y. Explicit finite element methods with contact-impact on SIMD computers. Computing Systems in Engineering, 1991, 2(2): 269-276.

[77] PLASKACZ E J, BELYTSCHKO T, CHIANG H-Y. Contact-impact simulations on massively parallel SIMD supercomputers. Computing Systems in Engineering, 1992, 3(1): 347-355.

[78] BROWN K, ATTAWAY S, PLIMPTON S, et al. Parallel strategies for crash and impact simulations. Computer Methods in Applied Mechanics and Engineering, 2000, 184(2): 375-390.

[79] LONSDALE G, PETITET A, ZIMMERMANN F, et al. Programming crashworthiness simulation for parallel platforms. Mathematical and computer modelling, 2000, 31(1): 61-76.

[80] 钟志华. 汽车耐撞性分析的有限元法. 汽车工程, 1994, 16(1): 1-6.

[81] 黄世霖, 张金换, 王晓冬. 汽车碰撞与安全. 北京: 清华大学出版社, 2000, 15-20.

[82] 寇哲君. 可扩展冲击-接触并行计算及其在汽车碰撞模拟中的应用: [清华大学博士学位毕业论文]. 北京: 清华大学, 2003, 60-65.

[83] MEI L, THOLE C. Data analysis for parallel car-crash simulation results and model optimization. Simulation modelling practice and theory, 2008, 16(3): 329-337.

[84] PARTL A M, MASELLI A, CIARDI B, et al. Enabling parallel computing in CRASH. Monthly Notices of the Royal Astronomical Society, 2011, 414(1): 428-444.

[85] WANG N-M, BUDIANSKY B. Analysis of sheet metal stamping by a finite-element method. Journal of Applied Mechanics, 1978, 45: 73-76.

[86] NAKAMACHI E, SOWERBY R. A numerical analysis of bulging and punch stretching of circular disks based on Kirchhoff plate theory. Advanced Technology of Plasticity, 1984, 1: 666-671.

[87] TANG S C. Computer prediction of the deformed shape of a draw blank during the binder-wrap stage. Journal of Applied Metalworking, 1980, 1(3): 22-29.

[88] 林忠钦. 车身覆盖件冲压成形仿真. 北京: 机械工业出版社, 2005, 35-40.

[89] 李光耀. 三维板料成形过程的显式有限元分析. 计算结构力学及其应用, 1995, 13(3): 253-268.

[90] 陈涛, 李光耀. 覆盖件拉延模工艺补充及压料面的参数化设计新方法. 机械工程学报, 2006, 42(5): 69-74.

[91] BUBAK M, CHROBAK R, KITOWSKI J, et al. Parallel finite element calculation of plastic deformations on Exemplar SPP1000 and on networked workstations. Journal of materials processing technology, 1996, 60(1): 409-413.

[92] XIE H, ZHONG Z, LI G, et al. Parallel computation and application of sheet forming numerical simulation. Zhongguo Jixie Gongcheng/China Mechanical Engineering, 2003, 14(21): 1842-1844.

[93] NIKISHKOV G, KAWKA M, MAKINOUCHI A, et al. Porting an industrial sheet metal forming code to a distributed memory parallel computer [J]. Computers & Structures, 1998, 67(6): 439-449.

[94] MORI K-I, OTOMO Y, YOSHIMURA H. Parallel processing of 3D rigid-plastic finite element method using diagonal matrix. Journal of Materials Processing Technology, 2006, 177(1): 63-67.

[95] GODDEKE D, STRZODKA R, MOHD-YUSOF J, et al. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. Parallel

Computing, 2007, 33(10–11): 685-699.

[96] YANG C-T, HUANG C-L, LIN C-F. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters. Computer Physics Communications, 2011, 182(1): 266-269.

[97] EKMAN M, WARG F, NILSSON J. An in-depth look at computer performance growth. SIGARCH Comput Archit News, 2005, 33(1): 144-147.

[98] LINDHOLM E, KILGARD M J, MORETON H. A user-programmable vertex engine. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. New York: ACM, 2001, 149-158.

[99] PURCELL T J, BUCK I, MARK W R, et al. Ray tracing on programmable graphics hardware. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. San Antonio: ACM, 2002, 703-712.

[100] KR GER J, WESTERMANN R. Linear algebra operators for GPU implementation of numerical algorithms. In: Proceedings of the ACM Transactions on Graphics (TOG). New York: ACM, 2003, 908-916.

[101] OWENS J D, LUEBKE D, GOVINDARAJU N, et al. A Survey of General-Purpose Computation on Graphics Hardware. Computer Graphics Forum, 2007, 26(1): 80-113.

[102] HILLESLAND K E, MOLINOV S, GRZESZCZUK R. Nonlinear optimization framework for image-based modeling on programmable graphics hardware. In: ACM SIGGRAPH 2005 Courses. Los Angeles: ACM, 2005, 224-233.

[103] LI W, WEI X, KAUFMAN A. Implementing lattice Boltzmann computation on graphics hardware. Visual Comp, 2003, 19(7-8): 444-456.

[104] 吴恩华, 柳有权. 基于图形处理器(GPU)的通用计算. 计算机辅助设计与图形学学报, 2004, 16(5): 601-612.

[105] BUCK I, FOLEY T, HORN D, et al. Brook for GPUs: stream computing on graphics hardware. In: ACM SIGGRAPH 2004 Papers. Los Angeles: ACM, 2004, 777-786.

[106] GODDEKE D, WOBKER H, STRZODKA R, et al. Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU. International Journal of Computational Science and Engineering, 2009, 4(4): 254-269.

[107] GARCIA M, GUTIERREZ J, RUEDA N. Fluid–structure coupling using lattice-Boltzmann and fixed-grid FEM. Finite Elements in Analysis and Design, 2011, 47(8): 906-912.

[108] HAIXIANG S, SCHMIDT B, WEIGUO L, et al. Accelerating error correction

in high-throughput short-read DNA sequencing data with CUDA. In: Proceedings of the Parallel & Distributed Processing 2009. Los Alamitos: IEEE, 2009, 11-18.

[109] LIGOWSKI L, RUDNICKI W. An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In: Proceedings of the Parallel & Distributed Processing 2009. Los Alamitos: IEEE, 2009, 1-8.

[110] SCHERL H, KECK B, KOWARSCHIK M, et al. Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA). In: Proceedings of the Nuclear Science Symposium Conference Record 2007. Los Alamitos: IEEE, 2007, 4464-4466.

[111] STONE S S, HALDAR J P, TSAO S C, et al. Accelerating advanced MRI reconstructions on GPUs. Journal of Parallel and Distributed Computing, 2008, 68(10): 1307-1318.

[112] GOVETT M, MIDDLECOFF J, HENDERSON T. Running the NIM Next-Generation Weather Model on GPUs. In: Proceedings of the Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference. Los Alamitos: IEEE, 2010, 792-796.

[113] MICHALAKES J, VACHHARAJANI M. GPU ACCELERATION OF NUMERICAL WEATHER PREDICTION. Parallel Processing Letters, 2008, 18(4): 531-548.

[114] FANG R, HE B, LU M, et al. GPUQP: query co-processing using graphics processors. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. Los Angeles: ACM, 2007, 1061-1063.

[115] CHEN S, QIN J, XIE Y, et al. A Fast and Flexible Sorting Algorithm with CUDA. In: 9th International Conference, ICA3PP 2009. Berlin: Springer Berlin Heidelberg, 2009, 281-290.

[116] MA W, AGRAWAL G. A compiler and runtime system for enabling data mining applications on gpus. SIGPLAN Not, 2009, 44(4): 287-288.

[117] CATANZARO B, SU B-Y, SUNDARAM N, et al. Efficient, high-quality image contour detection. In: Proceedings of the Computer Vision, 2009 IEEE 12th International Conference. Los Alamotis: IEEE, 2009, 2381-2388.

[118] ALLUSSE Y, HORAIN P, AGARWAL A, et al. GpuCV: A GPU-Accelerated Framework for Image Processing and Computer Vision. In: Proceedings of the 4th International Symposium on Visual Computing. Berlin: Springer Berlin

Heidelberg, 2008, 430-439.

[119] LEE M, JEON J H, BAE J, et al. Parallel implementation of a financial application on a GPU. In: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology. Los Angeles: ACM, 2009, 1136-1141.

[120] SINGLA N, HALL M, SHANDS B, et al. Financial Monte Carlo simulation on architecturally diverse systems. In: Proceedings of the High Performance Computational Finance 2008. Los Alamitos: IEEE, 2008, 1-7.

[121] CORRIGAN A, CAMELLI F F, L HNER R, et al. Running unstructured grid-based CFD solvers on modern graphics hardware. International Journal for Numerical Methods in Fluids, 2011, 66(2): 221-229.

[122] APPLEYARD J, DRIKAKIS D. Higher-order CFD and interface tracking methods on highly-Parallel MPI and GPU systems. Computers & Fluids, 2011, 46(1): 101-105.

[123] DE DONNO D, ESPOSITO A, TARRICONE L, et al. Introduction to GPU Computing and CUDA Programming: A Case Study on FDTD. Antennas and Propagation Magazine, 2010, 52(3): 116-122.

[124] TAKADA N, SHIMOBABA T, MASUDA N, et al. High-speed FDTD simulation algorithm for GPU with compute unified device architecture. In: Proceedings of the Antennas and Propagation Society International Symposium 2009. Los Alamitos: IEEE, 2009, 1-4.

[125] ANDERSON J A, LORENZ C D, TRAVESSET A. General purpose molecular dynamics simulations fully implemented on graphics processing units. Journal of Computational Physics, 2008, 227(10): 5342-5359.

[126] LIU W, SCHMIDT B, VOSS G, et al. Molecular Dynamics Simulations on Commodity GPUs with CUDA. In: High Performance Computing HiPC 2007. Berlin: Springer Berlin Heidelberg, 2007, 185-196.

[127] 柳有权, 尹康学, 吴恩华. 大规模稀疏线性方程组的 GMRES-GPU 快速求解算法. 计算机辅助设计与图形学学报, 2011, 23(4): 553-560.

[128] 李建江, 路川, 张磊. 基于指导语句的 CUDA 程序性能分析工具研究与实现. 电子科技大学学报, 2012, 41(2): 280-284.

[129] 刘小虎, 胡耀国, 符伟. 大规模有限元系统的 GPU 加速计算研究. 计算力学学报, 2012, 29(1): 146-152.

[130] XU J, WANG X, HE X, et al. Application of the Mole-8.5 supercomputer: Probing the whole influenza virion at the atomic level. Chin Sci Bull, 2011, 56

(20): 2114-2118.

[131] HAN L, HIPWELL J, TAYLOR Z, et al. Fast deformation simulation of breasts using GPU-based dynamic explicit finite element method. Digital Mammography, 2010, 6236: 728-735.

[132] KRAWEZIK G P, POOLE G. Accelerating the ANSYS direct sparse solver with GPUs. In: Proceedings of the 2010 Symposium on Application Accelerators in High Performance Computing (SAAHPC'10). Illinois, 2010, 1-3.

[133] NVIDIA. Popular GPU-accelerated Applications, http://www.nvidia.com/object/gpu-accelerated-applications.html, 2012-10-1.

[134] HENSLEY J. Amd ctm overview. In: SIGGRAPH'07. New York: ACM, 2007, 1-26.

[135] BAYOUMI A, CHU M, HANAFY Y, et al. Scientific and engineering computing using ati stream technology. Computing in Science and Engineering, 2009, 11(6): 92-97.

[136] STONE J E, GOHARA D, SHI G. OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering, 2010, 12(3): 66-72.

[137] KARIMI K, DICKSON N G, HAMZE F. A performance comparison of CUDA and OpenCL. arXiv:10052581, 2010.

[138] DU P, WEBER R, LUSZCZEK P, et al. From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. Parallel Computing, 2012, 38(8): 391-407.

[139] CUI X, LIU GR, LI GY, et al. Analysis of plates and shells using an edge-based smoothed finite element method. Computational Mechanics, 2010, 45(2): 141-156.

[140] ZHENG G, CUI X, LI G, et al. An edge-based smoothed triangle element for non-linear explicit dynamic analysis of shells. Computational Mechanics, 2011, 48(1): 65-80.

[141] FLYNN M J. Some Computer Organizations and Their Effectiveness. Computers, IEEE Transactions on, 1972, C21(9): 948-960.

[142] 陈国良. 并行算法的设计与分析. 北京: 高等教育出版社, 1994, 11-15.

[143] SIEWERT S. Using Intel® Streaming SIMD Extensions and Intel® Integrated Performance Primitives to Accelerate Algorithms. http://software.intel.com/en-us/articles/using-intel-streaming-simd-extensions-and-intel-integrated-perfo

rmance-primitives-to-accelerate-algorithms, 2009-11-04.

[144] REINDERS J. Intel threading building blocks: outfitting C++ for multi-core processor parallelism. Sebastopol,CA: O'Reilly Media, 2007, 20-25.

[145] SHAPIRO B A, WU J C, BENGALI D, et al. The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. Bioinformatics, 2001, 17(2): 137-148.

[146] CANT -PAZ E. A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 1998, 10(2): 141-171.

[147] YUAN F, LIAO G, FAN W, et al. An interactive 3D visualization system based on PC using Intel SIMD, 3D texturing and thinning techniques. International Journal of Pattern Recognition and Artificial Intelligence, 2006, 20(3): 393-416.

[148] CLARK T W, MCCAMMON J A. Parallelization of a molecular dynamics non-bonded force algorithm for MIMD architecture. Computers & Chemistry, 1990, 14(3): 219-224.

[149] KENNEDY K, BENDER C F, CONNOLLY J W D, et al. A nationwide parallel computing environment. Commun ACM, 1997, 40(11): 62-72.

[150] WITTENBRINK C M, KILGARIFF E, PRABHU A. Fermi GF100 GPU architecture. IEEE Micro, 2011, 31(2): 50-59.

[151] NICKOLLS J, DALLY W J. The GPU computing era. IEEE Micro, 2010, 30 (2): 56-69.

[152] EWING R E, SHARPLEY R C, MITCHUM D, et al. Distributed computation of wave propagation models using PVM. Parallel & Distributed Technology: Systems & Applications, IEEE, 1994, 2(1): 26-31.

[153] UEHARA H, TAMURA M, YOKOKAWA M. An MPI benchmark program library and its application to the Earth Simulator. In: Proceedings of the High Performance Computing. Berlin: Springer, 2002, 219-230.

[154] GEIST A, BEGUELIN A, DONGARRA J, et al. PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing. Cambridge: MIT press, 1994, 87-95.

[155] SNIR M, OTTO S, HUSS-LEDERMAN S, et al. MPI: The Complete Reference (Vol. 1): Volume 1-The MPI Core. Cambridge: MIT press, 1998, 1-3.

[156] 张武生, 薛巍, 李建江. MPI 并行程序设计实例教程. 北京: 清华大学出版社, 2009, 2-5.

[157] HILLIS W D, STEELE JR G L. Data parallel algorithms. Communications of

the ACM, 1986, 29(12): 1170-1183.

[158] HATCHER P J, QUINN M J, LAPADULA A J, et al. Data-parallel programming on MIMD computers. Parallel and Distributed Systems, IEEE Transactions on, 1991, 2(3): 377-383.

[159] DAGUM L, MENON R. OpenMP: an industry standard API for shared-memory programming. Computational Science & Engineering, IEEE, 1998, 5(1): 46-55.

[160] GLASKOWSKY P N. NVIDIA's Fermi: the first complete GPU computing architecture. http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf, 2009-9-1.

[161] ATALLAH M J. Algorithms and Theory of Computation Handbook. Boca Raton: CRC press, 1998, 17-20.

[162] WULF W A, MCKEE S A. Hitting the memory wall: Implications of the obvious. ACM SIGARCH Computer Architecture News, 1995, 23(1): 20-24.

[163] KANTER D. Nvidia's gt200: Inside a parallel processor, http://www.realworldtech.com/gt200/, 2008-9-8.

[164] GODEL N, NUNN N, WARBURTON T, et al. Scalability of higher-order discontinuous Galerkin FEM computations for solving electromagnetic wave propagation problems on GPU clusters. Magnetics(IEEE Transactions), 2010, 46(8): 3469-3472.

[165] BELYTSCHKO T, HUGHES T J. Computational methods for transient analysis (Mechanics and Mathematical Methods-Series of Handbooks). Amsterdam: North-Holland, 1983, 84-87.

[166] 崔向阳. 机械结构分析中的新型低阶高精度单元理论研究: [湖南大学博士学位论文].长沙:湖南大学, 2011, 61-65.

[167] BELYTSCHKO T, LIU W K, MORAN B. Nonlinear finite elements for continua and structures. 1 edition. Singapore: Wiley, 2000, 104-108.

[168] 王琥. 基于并行计算的金属塑性成形仿真分析中关键技术研究: [湖南大学博士学位毕业论文].长沙:湖南大学, 2006, 14-20.

[169] ELSEN E, LEGRESLEY P, DARVE E. Large calculation of the flow over a hypersonic vehicle using a GPU. Journal of Computational Physics, 2008, 227 (24): 10148-10161.

[170] 王建华, 李光耀, 李胜等. 基于 GPU 弹性问题的快速计算方法. 中国机械工程, 2011, 22(8): 932-937.

[171] JOLDES G R, WITTEK A, MILLER K. Real-time nonlinear finite element computations on GPU–Application to neurosurgical simulation. Computer Methods in Applied Mechanics and Engineering, 2010, 199(49): 3305-3314.

[172] COORPERATION N. NVIDIA CUDA C programming guide 4.0. Santa Clara: NVIDIA Corporation, 2011, 10-20.

[173] AHN J H, EREZ M, DALLY W J. Scatter-add in data parallel architectures. In: HPCA '05 Proceedings of the 11th International Symposium on High-Performance Computer Architecture.Washington:IEEE Computer Society Washington, 2005, 132-142.

[174] HE B, GOVINDARAJU N K, LUO Q, et al. Efficient gather and scatter operations on graphics processors. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing. Reno, Nevada: ACM, 2007, 1-12.

[175] GENAUD S, GIERSCH A, VIVIEN F. Load-balancing scatter operations for grid computing. Parallel Computing, 2004, 30 (8): 923-946.

[176] KIRK D B, WEN-MEI W H. Programming massively parallel processors: a hands-on approach. San Francisco: Morgan Kaufmann, 2010, 2-5.

[177] BELYTSCHKO T, LEVIATHAN I. Physical stabilization of the 4-node shell element with one point quadrature. Computer Methods in Applied Mechanics and Engineering, 1994, 113(3): 321-350.

[178] 钟阳, 钟志华, 李光耀等. 机械系统接触碰撞界面显式计算的算法综述. 机械工程学报, 2011, 47(13): 44-58.

[179] WHIRLEY R G, ENGELMANN B E. Automatic contact in DYNA-3D for vehicle crashworthniess. In: Crashworithniess and Occupant Protection in Transportation System, Proceedings of the 1993 ASME Winter Annunal Meeting. New York:ASME Applied Mechanics Division, 1993, 15-29.

[180] 谢晖, 钟志华, 李光耀等. 板料冲压数值模拟的并行计算与应用. 中国机械工程, 2003, 14(21): 1842-1844.

[181] BELL N, HOBEROCK J. GPU Computing Gems: Jade Edition. GPU Computing Gems: Jade Edition. San Francisco: Morgan Kaufmann, 2011, 359-372.

[182] OLDENBURG M. Finite element analysis of thin-walled structures subjected to impact loading: [Doctoral thesis]. Porsön: Luleå University of Technologyy,1988, 3-8.

[183] Makinouchi A. Sheet metal forming simulation in industry. Journal of Materials processing Technology, 1996, 60(1-4): 19-26.

[184] 郑刚. 汽车覆盖件冲压成形中拉延筋模型及其参数反演研究:[湖南大学博士学位论文]. 长沙:湖南大学, 2008.

[185] HILL R. A theory of the yielding and plastic flow of anisotropic metals. Proceedings of the Royal Society of London Series A Mathematical and Physical Sciences, 1948, 193(1033): 281-297.

[186] BARLAT F, LIAN K. Plastic behavior and stretchability of sheet metals. Part I: A yield function for orthotropic sheets under plane stress conditions. International Journal of Plasticity, 1989, 5(1): 51-66.

[187] PAPELEUX L, PONTHOT JP. Finite element simulation of springback in sheet metal forming. Journal of Materials Processing Technology, 2002, 125: 785-791.

[188] ERICSON C. Real-time collision detection. San Francisco: Morgan Kaufmann, 2004, 1-6.

[189] GREEN S. Particle simulation using CUDA. Santa Clara: NVIDIA Corporation, 2010, 1-3.

[190] LE GRAND S. Broad-phase collision detection with CUDA, GPU Gems.Boston: Addison-Wesley Professional, 2007, 3697-3721.

[191] Göddeke D, STRZODKA R, TUREK S. Accelerating double precision FEM simulations with GPUs. In: Proceedings of ASIM 2005 - 18th Symposium on Simulation Technique. Erlangen, 2005, 1-6.

[192] KUTTER O, SHAMS R, NAVAB N. Visualization and GPU-accelerated simulation of medical ultrasound from CT images. Computer methods and programs in biomedicine, 2009, 94(3): 250-266.

[193] FARBER R. CUDA application design and development. Waltham: Morgan Kaufmann, 2011, 207-239.

附录 A 攻读学位期间研究成果和发表学术论文情况

1 已发表和录用的期刊论文

1. Yong Cai, Guangyao Li, Hu Wang, Gang Zheng, Sen Lin. Development of parallel explicit finite element sheet forming simulation system based on GPU architecture, Advances in Engineering Software, 2012, 45(1): 370-379.(SCI, EI)
2. Yong Cai, Guangyao Li, Hu Wang. A Parallel Node-based Solution Scheme for Implicit Finite Element Method using GPU, Procedia Engineering, 2013, 61C:318-324. (accepted) (EI 源刊)
3. 蔡勇，李光耀，王琥．GPU 通用计算平台上中心差分格式显式有限元并行计算，计算机研究与发展，2013，50(2): 412-419. (EI)
4. 蔡勇，王琥，李光耀，崔向阳，郑刚．基于边光滑三角形壳元和统一计算架构的板料成形仿真并行计算方法，机械工程学报，2012，48(6): 32-38. (EI)
5. 蔡勇，李光耀，王琥．基于多重网格法和 GPU 并行计算的大规模壳结构快速计算方法，工程力学. (录用) (EI 源刊)
6. 蔡勇，李光耀，王琥．基于 CUDA 架构的并行粒子群优化算法的设计与实现，计算机应用研究，2013, 30(8): 2415-2418. (CSCD)

2 会议论文和报告

1. Wang, Hu; Li, Guangyao; Cai, Yong. Accelerate Parallel calculation of sheetforming by GPU，Proceedings of the 10th International Conference on Numerical Methods in Industrial Forming Processes Dedicated to Professor O. C. Zienkiewicz (1921-2009) (NUMIFORM 2010). Pohang, Republic of Korea, 13-17 June 2010.(会议报告)
2. Cai, Yong； Li, Guangyao. Accelerate Parallel calculation of sheetforming by GPU，Proceedings of the 10th International Conference on Numerical Methods in Industrial Forming Processes Dedicated to Professor O. C. Zienkiewicz (1921-2009) (NUMIFORM 2010). Pohang, Republic of Korea, 13-17 June 2010.(会议报告)
3. Cai, Yong； Li, Guangyao. Development of CUDA-based Parallel FE Explicit Code for Sheet Forming Simulation，International Conference on Computational Methods, Zhangjiajie, China, November 15-17, 2010. (会议报告)

3 软件著作权与专利

1. 蔡勇，李光耀. 基于 GPU 的板料成形并行计算软件(CADEM-GPU V1.0)，软件著作权(登记证书编号：2010SR052426).
2. 蔡勇,李光耀. 基于 GPU 的碰撞过程计算机仿真并行计算软件(CPS-GPUV1.0)，软件著作权(登记证书编号：2011SR001966).
3. 蔡勇，李光耀. 基于 GPU 的车身结构优化设计系统(CBO-GPU V1.0)，软件著作权(登记证书编号：2011SR001981).
4. 蔡勇，李光耀，王琥，郑刚. 基于 GPU 的有限元显式并行求解仿真方法，发明专利(已受理，受理号：20120266425.1).
5. 王琥，李光耀，蔡勇，龚志辉，郑刚. 基于响应函数的空间映射的冲压成形优化方法，发明专利(已授权，专利号：ZL201010255593.8).

致　谢

本论文的工作是在李光耀教授的悉心指导下完成，硕博连读的五年期间，李光耀老师严谨和实事求是的科研作风、渊博的学识和领导能力，令我深深敬仰，并使我受益终身。感谢李光耀老师对我的谆谆教诲、悉心关怀和殷切期望，并给予了我很多宝贵的科研和实践机会。基于 GPU 通用计算方法在我开始研究之初，属于完全新兴的研究方向，感谢李光耀老师为我提供和创造的优秀科研平台，使我可以排除疑虑，始终坚持。也深切感谢李光耀老师和师母张英老师对我、我爱人和我女儿的关心照顾。

感谢王琥副教授在科研方法、科学论文写作和并行计算方法上的指导，以及在日常生活中对我和我家人的支持。感谢郑刚副教授、崔向阳副教授、龚志辉副教授和孙光永讲师在板料成形分析和有限元计算方法上的指导。感谢研究团队中每位师兄弟给予的大量帮助。

感谢北京大学李胜副教授在并行计算方法上的指导。

感谢汽车车身先进设计制造国家实验室行政办韩琪老师给予了我许多成长和锻炼的机会，并在生活上给予我极大的支持，让我可以安心学习和科研。同时，感谢行政办张翠环老师对我学习和科研中琐事的热情帮助。

在我五年的硕博连读期间，家庭始终是我坚强的后盾，感谢我的父母和岳父岳母对我生活和学习上的支持。感谢我的妻子彭燕在背后的默默付出以及在精神上对我的支持。感谢我的女儿蔡佳琪，她的到来给我的生活带来了无尽的快乐和动力。

最后，感谢评审本论文付出辛勤劳动的全体专家学者。

蔡勇

2013 年 11 月