物体势流绕流问题的有限元并行计算

摘要

本文把有限元法与并行计算工具箱 PETSc (Portable，Extensible Toolkit for Scientific Computation) 结合，对理想不可压流体的物体势流绕流问题进行并行数值计算。并行工具箱 PETSc 为线性和非线性代数方程组的并行计算提供了可移植、可扩展的工具，非常适合偏微分方程离散后大型代数方程组的并行数值求解。本文首先利用 PETSc 线性方程组求解工具箱对大型线性方程组进行并行计算，验证该工具箱的并行效率。然后，把有限元法与 PETSc 结合，对理想不可压流体的二维和三维物体势流绕流问题，进行有限元离散和并行计算。计算区域用有限单元离散，单元形式为三角形或四边形单元(二维)，以及四面体单元(三维)，单元形函数为 Lagrange 插值函数。有限元积分离散后的线性代数方程组，用 PETSc 库函数进行并行求解。本文采用上述方法，对以下几种情况的物体势流绕流问题进行有限元并行计算：首先是二维无限域内圆柱绕流、NACA0010-66 机翼绕流以及 NACA0012 机翼绕流的并行数值计算；其次是二维无限域内双圆柱、双机翼的并行数值计算；最后是三维无限域内圆球绕流的并行数值计算。通过对不同情况下的绕流流场的数值模拟，给出了绕流速度势分布、流线分布、速度分布以及压力分布图，分析了进程数与并行计算效率的关系，部分绕流流场速度分布和物体表面压力分布的数值结果与解析解或经验值比较，两者吻合很好。数值计算结果表明，有限元法与 PETSc 结合可以高效率实现理想不可压流体物体势流绕流问题的并行数值求解，为今后开展复杂粘性流体流动问题的并行计算奠定了基础。

关键词：势流，有限元，绕流，PETSc，并行计算

FEM PARALLEL COMPUTING FOR POTENTIAL FLOWS AROUND BODIES

ABSTRACT

Parallel numerical computations for the potential flows aorund bodies are carried out using the coupled finite element method and PETSc (Portable, Extensible Toolkit for Scientific Computation) parallel toolkit. The PETSc provides portable and extensible tools for the numerical solution of partial differential equations on high-performance computers. In this thesis, first, the solution to large-scale linear equations is carried out to check the parallel efficiency by using PETSc toolkit in a PC cluster. Then, numerical simulations for the flows around two-dimensional and three-dimensional bodies in the ideal incompressible fluid based on the coupled PETSc-FEM are given. The following types of objects are taken into account: a cylinder, the NACA0010-66 wing, the NACA0012 wing, two cylinders, two wings and three-dimensional sphere. The numerical results of the distributions of potential function, stream function, velocity and pressure in the flow field are presented. The relationship between number of processors and the efficiency of parallel computing is analysed. The computing results agree very well with the corresponding analytic results (or experiential data). The high efficiency of the coupled PETSc-FEM parallel computing for the potential flows around bodies is verified in several numerical experiments. It is expected that the coupled PETSc-FEM could be applied effecently to the more complex viscous flow.

Key words: potential, FEM, flow, PETSc, parallel computing

目 录

第一章 绪论

1.1 课题背景

绕流现象因其有着广泛的应用背景，近二十年来一直是中外流体力学学者研究的热点。流场与流场中的各种结构体(如建筑物、桥梁、桥墩、机翼、烟囱等)之间的相互作用现象大量出现在各种实际问题中，如海岸工程、化学工程、地面交通、航空航天以及城市规划等众多领域，研究此类问题有着很强的实际意义。

一方面当流体流过结构体时会产生分离的现象，从而对结构体产生持续的作用力；同时在结构体的后部会有旋涡脱落的现象的产生，特别当结构体后部两侧交替产生的旋涡周期性的脱落会对结构体产生与来流方向垂直的周期性作用力。持续和脉动流体力的作用会使结构体产生疲劳、由于振动而产生噪声，严重时甚至会产生灾难性的破坏。1940年11月7日，美国华盛顿州新启用的Tacoma大桥在一场风速达67km/h的暴风雨中，由于大桥受到空气流经时产生的周期作用力与大桥的特征频率发生共振，使得大桥断裂。2001年11月12日，美国航空公司一架型号为A300的飞机于纽约坠机。坠机的原因就怀疑是与前起飞的日航B747客机相继起飞的间隔时间太短所致。这种四引擎的巨无霸喷射客机B747从喷射梢喷出两股旋风式成螺旋形下降的气流而形成强力尾流，使后起飞的A300受到了强烈的旋涡冲击导致坠机。此外，由流-固的相互作用引起的破坏还有很多，例如：高压电塔间的电缆线处于空气和雨水作用的流场中，流体在缆线表面产生流动作用，这种作用导致缆线表面形成自由剪切层，由于压力差的作用产生漩涡流动，缆线表面的力便会和惯性力、高压电塔作用力等相结合，再与缆线两端的阻尼力、恢复力产生作用，使缆线产生往复振动的现象，长久之后便导致结构疲劳而造成断裂。另外一方面，结构体同样会反过来影响流场的分布。在城市规划以及建筑设计中，特别是在城市建设日益向高度化发展的今天，由于城市建筑物向高层、密集化发展，就会产生“城市街区峡谷”。在“城市街区峡谷”中产生“城市急流”、“气流死区”，由此引发一系列的环境问题：比如在建成的街区内出现风口，或是局部风速过大，严重时将影响到行人以及附近建筑物：由于风速和风向的改变，在有火灾等紧急情况发生时会出现烟道效应，加速灾害的传递，增加灾害损失；局部街区可能出现流动迟滞现象，造成严重的局部空气污染[1]。

绕流问题出现在各种工程领域，不容忽视，在流体力学基础研究上，绕流问题是复杂的非线性问题，对于这一问题的研究无论在流体力学基础研究还是弄清实际工问题方面都是非常有意义的。由于实际工程的需要，很长时间以来，人们从实验、理论分析和数值模拟等角度进行了大量的研究。在数值模拟方面，这类问题的研究涉及到流体分析、势流物体分析以及流体、势流物体的混合分析，势流物体分析技术目前已十分成熟，流体分析尤其是流体和结构体之间的混合分析还远不如势流物体分析那样成熟，原因是控制流体运动的粘性不可压N-S方程是非线性的偏微分方程，这种方程不存在压力的显示形式、微分方程组是混合型的，不属于椭圆型、双曲型和抛物型方程中的任何一种。连续方程只充当动量方程的一个约束条件，求解时不能像可压缩流那样将各个量直接沿时间推进，压力场的确定成了求解方程的关键。

流场中的势流物体主要分为两类，一类是非流线型物体，在这类研究中，圆柱体研究得较多。这不仅因为在工程中的实际构件大多是圆柱形的，同时也因为圆柱绕流包含了此类物

体绕流的几乎所有复杂现象。在流体力学中，这类问题的研究已经成为基础研究和工程应用研究中最有意义的课题之一。另一类是流线体，如机翼、桨片、叶栅等，这一领域的研究内容主要集中在飞行器和叶轮机械设计，翼型绕流、边界层的非定常分离特性、翼型的尾迹流动以及尾迹涡系的结构等都包含了丰富的旋涡运动学和动力学问题，这些是涡动力学的重要研究对象。

1.2 国内外研究概况

目前，对于绕流现象的研究也有一百多年的历史了，我们可以较多的了解到关于圆柱体绕流，多圆柱绕流，方柱绕流，翼型绕流，叶栅绕流等方面的研究。随着计算机速度的提高、计算方法和网格技术的发展，数值模拟成为研究许多部件内部流动规律的重要手段。

1.2.1 非流线型物体绕流

非流线型物体绕流的研究有一百多年的历史。然而，非流线型物体绕流中的许多流动现象和物理机制仍未得到透彻的认识，一直是国内外流体力学学者的研究课题。此类绕流问题中研究最多的是圆柱绕流问题，而本文正是以圆柱绕流为例进行研究。

1.2.1.1 单圆柱绕流

决定该流场流态的重要参数是雷诺数(Re)。随着Re的变化，流场会呈现不同的流动状态。当Re <40时，尾迹中将出现对称的涡结构；当Re >40的时候，对称结构被打破，涡开始交替脱落，在圆柱后面形成卡门涡街。国内关于单圆柱绕流的研究有实验和数值模拟方面的，也有理论分析方面的。北京大学的邓庆增利用非线性动力学的概念和方法从实验和数值模拟两个方面研究了粘性不可压流体的二维圆柱绕流问题，计算了定常流失稳以及出现混沌的临界雷诺数值；上海理工大学的叶春明等利用适当修正的差分格式求解N-S方程，成功模拟出圆柱突然起动初期旋涡精细结构及长时间演化后的卡门涡街；中国科技大学的陆夕云和庄礼宪从直接求解非定常N-S方程出发，对旋转振荡圆柱粘性绕流问题进行了数值模拟，从流线图的演变规律方面给出了与实验结果相吻合的结果[2]。

1.2.1.2 双圆柱绕流

对多圆柱柱绕流问题的研究最早可以追述到上世纪初Pannel等人侧关于多细线的实验以及Bier和Herrnstein同飞机支杆的实验. 随着试验技术手段和计算机技术的日益成熟和发展，目前人们对圆柱绕流问题特别是对横向排列和流向排列双柱绕流的流场结构和圆柱受力特性已经有了较为清楚的认识。2003年澳大利亚Li Chen etc. 用LES方法对并列双圆柱的尾流进行了3-D数值模拟(Re=750)，数值计算结果与实验结果相一致。2003年周志勇等利用快速离散涡方法及流场可视化技术分析了二维粘性不可压缩均匀流中并列圆柱体旋涡脱落现象，使用边界元方法及引入一个新的涡量边界条件计算表面涡量，利用系数影响矩阵技术获得无网格方法，采用自适应快速多极子算法快速计算涡元速度。数值计算结果与早期实验结果吻合得很好。清华大学的刘松运用有限体积方法对绕串列放置双圆柱的二维不可压缩流动进行了数值计算，计算结果表明，流动特性在很大程度上取决于间距的大小，且间距存在临界值。华中理工大学的廖俊使用表面涡方法研究了高雷诺数下不同排列方式双圆柱绕流的流动状态，计算了双圆柱在并列、级列情况下的各种流动结构、涡街的变化以及作用在圆柱上的受力情况[3]。

1.2.2 流线型物体绕流

研究流线型物体绕流时以机翼问题最具代表性。机翼问题一支受到国内外广大学者的关注。机翼绕流属于三维湍流流动，具有非稳定、带旋转等特点，特别是在机翼交接处形成的分离涡流动，它包含着丰富的流体特性和机理。像机翼与机身、涡轮发动机叶片和轮毂、发动机短舱与机身、甚至是桥墩与河床之间的干扰均属于该范畴。机翼绕流的三维流场不仅决定着机翼本身的流体动力性能，而且还会对其周围的流场产生一定的影响。因此研究此类流

动现象的机理及流场压力分布不仅有一定的理论价值，而且对于工程实践有很强的实际意义。

目前对机翼绕流的研究主要在实验和数值模拟两方面进行。在实验方面，Hebbark在低速情况下用热线风速仪对零度攻角对称翼型的尾流进行了详细的实验测量；Yao利用粒子图象测速技术分别对自然尾流翼型的近尾迹流动以及小攻角下机翼周围的流场进行了测量；Thomas对低雷诺数情况下二维翼型绕流的分离特性进行了研究，发现机翼攻角和雷诺数对于翼型绕流的分离特性起到决定性作用。机翼绕流的实验研究虽然取得了一定的成果，但是一般要耗费巨大的资金和时间。另外，还有一些机翼绕流问题很难进行实验，如高速飞行的飞机的机翼绕流问题。因此，通过实验对机翼绕流问题进行研究有着一定的局限性。

在数值模拟方面，Gadel-Hak对俯仰翼型进行了研究，发现当升力面在低速和高速流场内俯仰振动时，绕升力面的流场及气动力在其上仰和下俯相同迎角瞬间是不同的；Jameson等人用中心有限体积、多步Runge-Kutta时间步长格式求解三维N-S方程，研究了风洞侧壁干扰的问题；钱炜棋等采用求解低速流动的SIMPLE算法，用标准k-ε两方程模型、非线性k-ε两方程模型和一种改进的k-ε两方程模型来对绕两种翼型（A翼型和GAW-1翼型）流动在接近失速攻角情况下的低速湍流流动进行了数值模拟。计算结果表明，非线性k-ε两方程模型和改进的k-ε。两方程模型较好地模拟出了翼型表面上的分离流动特性，较准确地预测出了分离点位置以及翼型的气动力系数，对翼型与机翼的工程设计具有一定的参考价值。数值模拟能够解决机翼绕流的实验研究中遇到的困难，数值模拟能够简便的施加边界条件，并且具有快速、灵活、不受场地限制的优点。它从基本的物理方程入手，在一定条件下可以模拟出物理模型不能精确测量和观察到的对象，具有丰富的创造性，进而成为研究机翼绕流问题的流动特性的重要手段。近年来随着CFD的飞速发展以及计算机性能和精度的不断提高，数值模拟越来越受到学者们的青睐[4]。

1.3 主要研究内容及研究意义

本课题主要研究二维圆柱、二维机翼以及三维圆球存在于流场时流体与上述势流物体之间的相互作用现象。这些绕流问题与诸多实际问题紧密相关，因此，通过本课题的研究对于了解工程实际中的相关问题的物理机制有着及其重要的意义，可以避免由于流-固之间相互作用所产生的不良后果。

本课题采用有限元并行计算的方法，运用 PETSc 并行工具以及 Gambit 网格生成工具分别对单圆柱、NACA0010-66 翼型、NACA0012 翼型、双圆柱、双机翼及圆球的绕流进行了细致认真的研究。

第二章 有限元方法简介

2.1 引言

流体力学的研究方法主要分为现场观测、理论分析、实验测量和数值计算四种。现场观测法要花费大量的物力、财力和人力，现场流动现象的发生往往不能控制，发生条件几乎不可能完全重复出现，这会影响到现场观测法对流动现象和规律的研究；在理论分析上，流体力学的控制方程是非线性的或混合型的，具有各种数学奇异性，研究问题中还可能存在气穴、激波、自由表面、湍流等特殊现象，这些复杂性给理论分析法增加了巨大的困难和麻烦；实验方法能够较准确地反映真实流场情况，然而，也存在着耗资昂贵、周期长、受限于模拟条件、难以显示、误差难以避免以及实验条件不能同时满足几个相似准则等不足之处。随着计算机科学技术和近代数学的发展，为用数学模型来表达各种复杂流动问题的数值计算提供了可能性，数值地求解流体力学中各种各样的问题，已成为现今流体力学研究方法中不可替代的方法之一，从而催生了计算流体力学这门交叉学科。

计算流体力学（CFD，Computational Fluid Dynamics）是一门用数值计算方法直接求解流动主控方程（Euler或Navier-Stokes方程）以发现各种流动现象规律的学科。它综合了计算数学、计算机科学、流体力学、科学可视化等多种学科。广义的CFD包括计算水动力学、计算空气动力学、计算燃烧学、计算传热学、计算化学反应流动，甚至数值天气预报也可列入其中。

自二十世纪六十年代以来CFD技术得到飞速发展，其原动力是不断增长的工业需求，而航空航天工业自始至终是最强大的推动力。传统飞行器设计方法试验昂贵、费时，所获信息有限，迫使人们需要用先进的计算机仿真手段指导设计，大量减少原型机试验，缩短研发周期，节约研究经费。四十年来，CFD在湍流模型、网格技术、数值算法、可视化、并行计算等方面取得飞速发展，并给工业界带来了革命性的变化。如在汽车工业中，CFD和其它计算机辅助工程（CAE）工具一起，使原来新车研发需要上百辆样车减少为目前的十几辆车；国外飞机厂商用CFD取代大量实物试验，如美国战斗机YF-23采用CFD进行气动设计后比前一代YF-17减少了60％的风洞试验量。目前在航空、航天、汽车等工业领域，利用CFD进行的反复设计、分析、优化已成为标准的必经步骤和手段。当前CFD问题的规模为：机理研究方面如湍流直接模拟，网格数达到了10^9（十亿）量级，在工业应用方面，网格数最多达到了10^7（千万）量级。

2.2 计算机数值计算基本方法

计算机数值计算的基本方法主要有以下几种，本节将分别予以介绍：

（1）有限差分法(Finite Difference Method)

有限差分方法是计算机数值模拟最早采用的方法，至今仍被广泛运用。该方法将求解域划分为差分网格，用有限个网格节点代替连续的求解域。有限差分法用Taylor级数展开，把控制方程中的导数用网格节点上的函数值的差商代替进行离散，从而建立以网格节点上的值为未知数的代数方程组。该方法是一种直接将微分问题变为代数问题的近似数值解法，数学概念直观，表达简单，是发展较早且比较成熟的数值方法。对于有限差分格式，从格式的精度来划分，有一阶格式、二阶格式和高阶格式。从差分的空间形式来考虑，可分为中心格式和逆风格式。考虑时间因子的影响，差分格式还可以分为显格式、隐格式、显隐交替格式等。

目前常见的差分格式，主要是上述几种形式的组合，不同的组合构成不同的差分格式。差分方法主要适用于有结构网格，网格的步长一般根据实际地形的情况和柯朗稳定条件来决定。构造差分的方法有多种形式，目前主要采用的是泰勒级数展开方法。

（2）有限体积法(Finite Volume Method)[5]

有限体积法又称为控制体积法。其基本思路是：算区域划分为一系列不重复的控制体积，并使每个网格点周围有一个控制体积，将待解的微分方程对每一个控制体积积分，便得出一组离散方程。其中的未知数是网格点上的因变量的数值。为了求出控制体积的积分，必须假定值在网格点之间的变化规律，即假设值的分段的分布剖面。从积分区域的选取方法来看，有限体积法属于加权剩余法中的子区域法；未知解的近似方法来看，有限体积法属于采用局部近似的离散方法。简言之，子区域法属于有限体积法的基本方法。有限体积法的基本思路易于理解，并能得出直接的物理解释。离散方程的物理意义，就是因变量在有限大小的控制体积中的守恒原理，如同微分方程表示因变量在无限小的控制体积中的守恒原理一样。有限体积法得出的离散方程，要求因变量的积分守恒对任意一组控制体积都得到满足，对整个计算区域，自然也得到满足。这是有限体积法吸引人的优点。

（3）有限分析法(Finite Analytic Method)[6]

有限分析法是美国籍华裔科学家陈景仁教授在70年代末提出的。有限分析法的基本思路是将待求解的总体区域划分为有限个子区域。在这些子区域内做一些简化，然后求解简化的偏微分方程在该区域中的解析解，再从解析解中导出一个代数方程，使子区域上的内结点值与相邻的结点值联系起来；接着把所有的局部解析解汇集在一起，就得到所求问题的有限分析数值解。有限分析法具有自动迎风性质，能准确的模拟对流项，同时不存在是指扩散现象，计算稳定性较好，收敛较快。

（4）特征线法(Method of characteristic curves)[7]

特征线是在x-t平面上的一组有向线段，其斜率表示沿特征线速度和水深的变化规律，在特征线的交点上确定因变量来求解方程，引入特征线理论形成了特征线法(MOC)。MOC是针对系统的各节点建立独立的维数较低的非线性方程组进行单独运算，所需计算机内存较小。由于对时间离散和空间离散一起处理，MOC数学分析严谨，物理概念明确，能较好地反映信息沿特征传播的性质，计算精度较高，是解决双曲型方程和抛物型方程的较好的方法。但MOC时间步长和空间步长的比值受到一定稳定条件地限制，求解时只能取较小的时间步长，若分析过程较长，需要消耗很长的时间。

（5）边界元法(Boundary Element Method)[8]

边界元法是一种类似于FEM的求解微分方程初边值问题的数值计算方法，在数学原理上和FEM不同，但是离散求解方法和FEM类似。边界元法是一种近似的边界积分方法，微分方程的初边值问题要通过某种途径转化为等价的边界积分方程，然后以此方程为对象进行离散求解，将微分方程转化为边界积分方程，通常有两种途径:直接法和间接法。边界元法对边界进行分块逼近，因此近似求解问题的维数降为一维，计算工作量减小。随着高速电子计算机的出现，边界元法已经广泛应用于弹性力学、势流、粘性层流、地下渗流等问题。边界元法要采用解析函数的基本解，目前只适用于线性问题以及基本解已知的问题。对于非线性问题和半无限域问题，特别是区域的角点等处理还在研究之中。

（6）有限元方法(Finite Element Method)

有限元方法最早应用于结构力学，后来随着计算机的发展慢慢用于流体力学的数值模拟。有限元法的优点是:网格剖分灵活，能够以较少的网格获得较高的精度，能够处理较复杂几何边界问题，且算法程序具有通用性。本文采用数值模拟的方法研究势流物体绕流问题，考虑到其边界复杂的特点，本文将运用有限元方法对控制方程离散。在下节中将重点介绍有限元方法的原理及优点。

2.3 有限元基本原理[9,10]

有限元方法的基础是变分原理和加权余量法，其基本求解思想是把计算域划分为有限个互不重叠的具有一定规则几何形状的子区域（如：在二维问题中可以划分为三角形或四边形；在三维问题中可以划分为四面体或六面体等），这些子区域被称为单元。单元之间以结点相联结。在每个单元内，选择一些合适的结点作为求解函数的插值点，将微分方程中的变量改写成由各变量或其导数的结点值与所选用的插值函数组成的线性表达式，借助于变分原理或加权余量法，考虑到边界条件，将微分方程离散求解。采用不同的权函数和插值函数形式，便构成不同的有限元方法。其求解思路概括地来说就是“分块逼近”。由于单元的几何形状是规则的，因此在单元上构造基函数可以遵循相同的法则，每个单元的有限元方程都具有相同的形式，可以用标准化的格式表示，其求解步骤变得很规范，即使是求解域剖分各单元的尺寸大小不一样，其求解步骤也不用改变，这就为利用计算机编制通用程序进行求解带来了方便。

数值模拟中，常见的有限元计算方法是由变分法和加权余量法发展而来的里兹法和伽辽金法、最小二乘法等。根据所采用的权函数和插值函数的不同，有限元方法也分为多种计算格式。从权函数的选择来说，有配置法、矩量法、最小二乘法和伽辽金法，从计算单元网格的形状来划分，有三角形网格、四边形网格和多边形网格，从插值函数的精度来划分，又分为线性插值函数和高次插值函数等。不同的组合同样构成不同的有限元计算格式。对于权函数，伽辽金法(Galerkin)是将权函数取为逼近函数中的基函数；最小二乘法是令权函数等于余量本身，而内积的极小值则为对代求系数的平方误差最小；在配置法中，先在计算域内选取N个配置点，令近似解在选定的N个配置点上严格满足微分方程，即在配置点上令方程余量为0。插值函数一般由不同次幂的多项式组成，但也有采用三角函数或指数函数组成的乘积表示，但最常用的是多项式插值函数。有限元插值函数分为两大类，一类只要求插值多项式本身在插值点取已知值，称为拉格朗日(Lagrange)多项式插值；另一种不仅要求插值多项式本身，还要求它的导数值在插值点取已知值，称为哈密特(Hermite)多项式插值。单元坐标有笛卡尔直角坐标系和无因次自然坐标，有对称和不对称等。常采用的无因次坐标是一种局部坐标系，它的定义取决于单元的几何形状，一维看作长度比，二维看作面积比，三维看作体积比。

对于有限元方法，其基本思路和解题步骤可归纳为：

(1) 建立积分方程，根据变分原理或方程余量与权函数正交化原理，建立与微分方程初边值问题等价的积分表达式，这是有限元法的出发点。

(2) 区域单元剖分，根据求解区域的形状及实际问题的物理特点，将区域剖分为若干相互连接、不重叠的单元。区域单元划分是采用有限元方法的前期准备工作，这部分工作量比较大，除了给计算单元和节点进行编号和确定相互之间的关系之外，还要表示节点的位置坐标，同时还需要列出自然边界和本质边界的节点序号和相应的边界值。

(3) 确定单元基函数，根据单元中节点数目及对近似解精度的要求，选择满足一定插值条件的插值函数作为单元基函数。有限元方法中的基函数是在单元中选取的，由于各单元具有规则的几何形状，在选取基函数时可遵循一定的法则。

(4) 单元分析：将各个单元中的求解函数用单元基函数的线性组合表达式进行逼近；再将近似函数代入积分方程，并对单元区域进行积分，可获得含有待定系数(即单元中各节点的参数值)的代数方程组，称为单元有限元方程。

(5) 总体合成：在得出单元有限元方程之后，将区域中所有单元有限元方程按一定法则进行累加，形成总体有限元方程。

(6) 边界条件的处理：一般边界条件有三种形式，分为本质边界条件(狄里克雷边界条

件)、自然边界条件(黎曼边界条件)、混合边界条件(柯西边界条件)。对于自然边界条件，一般在积分表达式中可自动得到满足。对于本质边界条件和混合边界条件，需按一定法则对总体有限元方程进行修正满足。

(7) 解有限元方程：根据边界条件修正的总体有限元方程组，是含所有待定未知量的封闭方程组，采用适当的数值计算方法求解，可求得各节点的函数值。

目前，在数值计算中常用的方法有有限元法和有限差分法。在此就以有限差分法为例，通过两者的比较说明有限元方法的主要优点:

(1) 有限元方法对于求解区域的单元剖分没有特别的限制,这对处理具有复杂边界区域的工程实际问题格外方便，而且可以完全根据问题的物理特点，在求解区域中安排单元网格的疏密。

(2) 有限元方法是将区域进行分片离散，对每一个单元而言，它的近似解是解析的。

(3) 对于事先未知的自由边界或求解区域内部不同介质的交界而言，比较容易处理。

2.4 本章小结

流体力学数值计算以其诸多优点成为流体力学理论研究和流体工程设计的重要手段。基于本课题的研究内容以及有限元法的优点，本课题的研究工作运用有限元法进行离散求解。本章介绍了数值计算的基本方法并重点给出有限元法的基本原理。

第三章 并行计算及 PETSc 简介

3.1 并行计算[11]

随着科学技术的迅猛发展，需要解决的工程实际问题越来越复杂，需要处理的信息量越来越多，使得计算量剧增，对计算机性能要求也越来越高。近年来相继发展的巨型计算机广泛采用多 CPU 结构和群集式并行结构，获得了最优性能。但是，计算机性能的提高受到物理限制，不可能无限提高，且巨型机价格十分昂贵，普及困难。可见，通过提高单个处理器的运算速度和采用传统的串行计算技术已难以胜任。因此，需要有功能更强大的新的计算机系统和计算技术来支撑。并行计算机及并行计算技术应运而生。

3.1.1 并行计算定义

所谓并行计算是指同时对多个任务或多条指令、或对多个数据项进行处理。它分为时间上的并行和空间上的并行；时间上的并行就是指流水线技术，而空间上的并行则是指用多个处理器并发的执行计算。完成此项处理的计算机系统称为并行计算机系统，它是将多个处理器（可以几个、几十个、几千个、几万个等）通过网络连接以一定的方式有序地组织起来（一定的连接方式涉及网络的互联拓扑、通信协议等，而有序的组织则涉及操作系统、中间件软件等）。并行计算的主要目的：一是为了提供比传统计算机快的计算速度；二是解决传统计算机无法解决的问题。

3.1.2 并行计算平台

3.1.2.1 并行计算机的控制结构

对并行计算机的分类有多种方法，其中最著名的是 1966 年由 M.J.Flynn 提出的分类法，称为 Flynn 分类法。Flynn 分类法是从计算机的运行机制进行分类的。根据指令流和数据流的不同组织方式，Flynn 把计算机系统的结构分为以下四类：单指令流单数据流（SISD）、单指令流多数据流（SIMD）、多指令流单数据流（MISD）、多指令流多数据流（MISD）。

SISD 就是普通的顺序处理的串行机。SIMD 和 MIMD 是典型的并行计算机。MISD 代表何种计算机存在较大分歧。

3.1.2.2 并行计算机的地址空间

从地址空间的角度，并行计算机被分为两类：消息传递体系结构和共享地址空间体系结构。

在消息传递结构的并行机中，通常每个处理器有自己的存储器。该存储器只能被该处理器访问而不能被其它处理器直接访问，因此这种存储器通常被称为局部存储器或私有存储器。当处理器 A 需要向处理器 B 传送数据时，A 把被传送的数据以消息的形式发送给 B。

在共享地址空间体系结构的并行机中，通过硬件支持，使得系统中只有唯一的一个地址空间。所有的处理器共享该地址空间。共享地址空间并不意味着系统中必须存在一个在物理上共享的存储器。共享地址空间可以通过一个物理上共享的存储器来实现，也可以通过分布式存储器来实现。在某些并行系统中，存储器分布在各结点内，通过硬件和软件的方法维护一个单一的地址空间。当存储器要访问不在本结点内的内存时，由系统硬件和软件为它找到所需访问的内存。

3.1.2.3 并行计算机的系统结构

从系统结构角度，并行计算机一般可以分为：单指令流多数据流计算机 SIMD、并行向量处理机 PVP、对称多处理机 SMP、大规模并行处理机 MPP、工作站机群 COW、分布式

共享存储多处理机 DSM。上述几种中，除了 SIMD 外，其余的均属于 MIMD 计算机。PVP 的处理器和互联网络都是定制的，SMP、MPP 多使用商用处理器，MPP 的互联网络是定制的，COW 基本都使用商品部件。

对称多处理机 SMP、大规模并行处理机 MPP 以及机群系统 COW 是当代主流并行机。

3.1.3 并行计算的软件条件

3.1.3.1 并行平台——Linux 操作系统

从性能和费用等方面考虑，在支持网络并行计算的操作系统中优先选择的系统应是 Linux。Linux 作为一个新兴的操作系统，以它为并行环境来构建集群有很多优点。Linux 系统是一个 32 位的操作系统。它遵循 POSIX 规范.对硬件要求很低，配置 80386 以上档次 CPU 并具有 2MB 以上内存的 PC 机均可安装运行。Linux 系统是一个免费软件，它的内核完全自行开发而成，系统程序也全部由免费软件构成，因此便于推广。许多网络并行计算均可在 Linux 系统上运行，这些是我们选择 Linux 作为微机的并行计算平台的主要原因。

与 Windows 以及其它商品化操作系统相比，Linux 的一个显而易见的优势就是廉价。硬件的花销加上很少的软件费用就可以拥有一个 PC 工作站或服务器，这方面显然是其它操作系统无法比拟的，而且 Linux 对于硬件的要求比 Windows 要低得多，一般的用户也可以利用 Linux 来构造一个高性能的集群来进行科学计算，在很大程度上它可以替代以往昂贵的大型计算机。开放源码为提高性能提供了更加广阔的空间。开发者可以看到这个系统是怎样运行起来的，然后在操作系统一级进一步提高性能便成为可能。而在 Windows 或者 AIX 这样的操作系统中，得到它们的源码已是很不容易，要想从操作系统着手来优化上层的大型应用更是难上加难。

目前，由于还比较缺乏对 Linux 的性能和功能评价的系统科学研究，在同等硬件配置和应用环境下，Linux 与其他操作系统相比孰优孰劣还不太明了。但是，已经有不少数据说明，作为工作站或小型服务器。Linux 已经可以与它的对手一较高低了，尤其是它的网络性能以及可靠性都备受称赞，而这些正是一个高效集群不可缺少的。

3.1.3.2 并行环境——消息传递接口 MPI[12]

MPI 于 1994 年问世，是目前最重要的并行编程工具之一。它吸收了其他多种并行环境的优点，具有移植性好、功能强大、效率高等优点，在短短几年里迅速普及，成为消息传递并行编程模式的标准。MPI 是一个接口库，遵循所有对库函数的调用规则，可以被 FORTRAN77、FORTRAN90、C、C++程序调用。由于 MPI 是一个库而不是编程语言，因此使用 MPI 时必须和特定的程序设计语言结合起来。FORTRAN 是科学和工程计算语言，C 是目前被广泛使用的系统和程序设计语言。在 MPI 中明确提出了与 FORTRAN,C 的绑定.并且给出了通用接口和针对 FORTRAN 与 C 的专用接口说明。

MPI 拥有上百个调用接口，是一个宏大的软件平台，具有功能齐全的接口调用，但是它又是一个十分小巧灵活的系统，MPI 的所有消息传递功能可以用 6 个基本 MPI 库函数来实现。

MPI具有如下主要特点:

- 通用性。MPI是可移植的标准平台，在一个厂家的MPI的环境下开发的应用程序，可以在其他厂家的MPI环境下更新和运行，称为源程序级的可移植性，但不支持异构网络和异质结构下MPI的应用程序间的通信。
- 通信方式。MPI在点到点通信和集合通信两方面都提供了更多的通信函数和更强大的通信功能。在点到点通信方式中，发送和接收都有阻塞和非阻塞两种。另外，也提供对于不同类型的数据建立导出数据类型的函数。MPI又提供了非阻塞的缝合通信及单向通信函数，更增强了MPI的通信功能。
- 任务控制和分配。MPI对任务的分配提供了按照一定的逻辑互连拓扑结构分配计算资源

的函数，如按照网络、超立方体等，由MPI保证利用低层的物理网络拓扑给予最优的消息传递。

- 安全通信的上下文和多线程。MPI提供第三个标签以区别用户消息和库函数发送的消息，称其为通信的上下文。同时，MPI引人通信子的概念，即一组具有同一上下文的任务与通信上下文的绑定。MPI通过引入通信子，支持并发库的开发，支持多线程应用。

3.1.4 并行效率

并行计算主要是为了解决串行程序计算速度或内存不足而设计的。因此对并行效率的评价也应该根据实际需要从程序执行时间和单机内存的减少来判断。只有理想的具有p个处理器的并行系统的加速比能够达到p，在实际中，这种理想情形是不可能达到的，因为在执行并行算法时，处理器不可能把它们100%的时间都用来执行算法。

加速比定义为：

$$\mathbf{S_p} = \frac{\text{串行算法在处理机的执行时间}}{\text{并行算法在具体有p台处理机的系统上的执行时间}} \qquad (3\text{-}1)$$

效率被用来衡量一个处理器的计算能力被有效利用的比率。在一个理想的并行系统中，加速比等于p，而效率等于1，但是在实际系统中，加速比小于p，而效率在0到1之间取值，它描述了处理器被有效利用的程度，用E来表示效率。

并行算法效率定义为：

$$E_p = \frac{S_p}{p} \qquad (3\text{-}2)$$

3.2 PETSc简介[13]

PETSc（ Portable, Extensible Toolkit for Scientific Computation ）是美国Argonne国家实验开发的可移植可扩展科学计算工具箱，目的是在高性能计算机上数值求解偏微分方程及相关问题。PETSc包含一套正在扩充的并行线性、非线性方程解法器和时间积分器，它们可能在用FORTRAN、C和C++编写的应用代码中被用到。PETSc提供并行应用代码中所需的许多机制，如并行矩阵和向量聚集程序。本程序库采用分级组织，使用户可以选择适合的结构层次来解决他的特定问题，由于使用了面向对象编程技术，PETSc使其内部功能部件的使用非常方便；PETSc完全兼容MPI系统，即在PETSc程序中，可任意调用MPI函数；PETSc对所有消息传递通信均使用MPI标准。

PETSc包含许多库（类似于c++中的类），每一部分处理一个特殊的对象类(如向量)以及可在此对象上执行的运算。PETSc的模块处理主要有:索引集，包括用于向量索引的置换，重新计数等；向量；矩阵（一般是稀疏的）；分布阵列(对正规的基于网格问题的并行化有用)； Krylov子空间方法；预条件子，包括多重网络和稀疏直接解法器；非线性解法器；和解时间相关(非线性)PDEs的时间步进解法器。每一个部分都包含一个抽象的接口(一组调用序列)和一个或多个用特殊数据结构的实现。因此，PETSc对解PDEs的各个阶段提供了清晰且有效的代码，对每个问题均用统一的方法。这种设计易于比较和使用不同的算法(例如，用不同Krylov子空间方法、预条件子或截断Newton方法等进行试验)。因此，PETSc对模型科学应用以及快速算法设计和开发提供了一个丰富的环境。这些库使得算法和实现的定制和扩展都更容易。这种方法促进了代码的再利用与及其灵活性，并将并行性问题与算法的选择分离开来。

可移植可扩展科学计算工具箱(PETSc)已成功地表明:使用现代编程典范可使 Fortran C和C++编写的大规模科学应用代码变得更容易。PETSc具备一般库软件具备的高性能、可

移植优点，而且面向对象技术使得PETSc内部功能部件的使用非常方便，接口简单而又适用面广，大量缩短开发周期和减少工作量。所以，PETSc是一个值得去学习和使用的强大的数值计算的工具。

3.3 PETSc 并行测试

本文测试 PETSc 程序求解大型五对角线性方程组的能力，以及其并行性能。本次测试在 PC-Cluster 上进行，具体内容是分别测试不同维数线性方程组在不同进程数下求解的结果及性能。

本文求解线性方程组的系数矩阵为五对角矩阵，五根对角线上的值从左到右依次为：-80，-1，2，-1，80，其余元素均为 0. 右手项向量的值为（81，80，0，0，……，0，0，-80，-79），准确解为（1，1，……，1，1）。本文共测试了系数矩阵维数分别为 50000，100000，200000 的线性系统在不同进程数下的计算性能。下面将重点介绍通过 PETSc 程序并行求解维数为 100000 的线性方程组所得的结果及性能，以及不同系数矩阵维数求解结果比较。

从本次测试的求解环境可以看出，线性求解器所使用的方法为广义最小残差法 gmres，使用古典的（未经修改的）Gram-Schmidt 正交方法，每 30 步重开始一次。最大迭代步数为 100000，初始给定解向量为零向量。相对收敛误差为 1e-07，绝对收敛误差为 1e-50，收敛判断标准为：

$$\|r(k)\| < \max(relative \cdot \|b\|, absolute) \tag{3-3}$$

发散相对误差为 10000，发散判断标准为：

$$\|r(k)\| > divergence \cdot \|b\| \tag{3-4}$$

其中 b 为右手项，r(k)为迭代第 k 步后的残差，向量范数取 2—范数。

迭代矩阵使用左预处理方式，而预处理方式为 Jacobi，线性迭代矩阵与需经预处理的矩阵相同。矩阵存储方式为 aij，求解时使用单进程，因此矩阵类型为 seq（sequential）。

通过运行程序时得到相关的程序运行性能报告“-log_summary”，该报告包含了程序运行的性能表现，其中主要的内容包括：

（1）总体计算性能情况：时间、目标个数（包括 viewer, vec, mat, ksp, pc 等）、浮点运算次数（一次实数的加、减、乘或除运算定义为一次浮点运算）、每秒浮点运算次数、通信次数、通信长度、归约（MPI Reductions，是一种将不同进程数据执行相同运算得到的操作）等各统计变量的最大值、平均值、总值等。

（2）不同阶段的计算信息：时间、浮点运算、通信、通信长度、归约等变量的平均值或次数，以及占整个计算过程的百分比。

（3）各计算阶段内各子程序的计算信息。

（4）内存占用情况和目标的创建和注销情况：目标的创建次数、注销次数、相应的内存占用情况等。

（5）程序运行环境的相关信息等。

3.3.1 系数矩阵维数为 100000

从计算结果中可以看出，随着进程数的增加，每秒浮点数运算次数增大，使得程序运行的时间逐渐减小，但由于通信数和通信长度与进程数成比例的增加，通信开销增大，导致实际运行时间变长。对于小计算量的并行求解，并不是进程数越大越好，应根据线性系统的规模选择合适的进程数，以达到最小的计算时间。对于特定的线性系统，浮点数的运

算总量是一定的，运行时间与每秒浮点运算次数成反比，同时，误差范数与收敛迭代步数也与进程数无关。

表 3-1 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	4.466e+03	2.433e+03	1.644e+03	1.234e+03
Objects:	4.300e+01	5.000e+01	5.000e+01	4.900e+01
Flops:	1.120e+12	1.120e+12	1.120e+12	1.120e+12
Flops/sec:	2.508e+08	4.603e+08	6.815e+08	9.080e+08
MPI Message:	0.000e+00	2.984e+05	5.970e+05	8.957e+05
MPI Message Lengths:	0.000e+00	1.277e+07	2.555e+07	3.833e+07
Norm of error:	3.58727e-05	3.56711e-05	3.57807e-05	3.57755e-05
Iterations:	144420	144403	144441	144453
	5 procs	6 procs	7 procs	8 procs
Time (sec):	9.155e+02	7.856e+02	6.773e+02	6.910e+02
Objects:	5.000e+01	5.000e+01	5.000e+01	5.000e+01
Flops:	1.120e+12	1.120e+12	1.120e+12	1.120e+12
Flops/sec:	1.224e+09	1.426e+09	1.653e+09	1.621e+09
MPI Message:	1.194e+06	1.492e+06	1.791e+06	2.090e+06
MPI Message Lengths:	5.110e+07	6.388e+07	7.665e+07	8.943e+07
Norm of error:	3.57803e-05	3.58227e-05	3.58381e-05	3.58231e-05
Iterations:	144438	144421	144389	144424

3.3.2 不同系数矩阵维数求解结果比较

从下表可以看出，本文的计算时间与系数矩阵维数的平方成正比，矩阵维数越大，所需的计算时间越长；对于同一维数矩阵而言，随着进程数的增加，计算时间逐渐减少。

表 3-2 不同系数矩阵维数计算时间比较

维数	1 proc	2 procs	3 procs	4 procs
50000×50000	1.037e+03	5.546e+02	3.598e+02	2.892e+02
100000×100000	4.466e+03	2.433e+03	1.644e+03	1.234e+03
200000×200000	1.842e+04	1.038e+04	6.812e+03	5.026e+03

通过对不同维度的线性系统在不同进程下的求解结果及性能的比较可以得出：

（1）PETSc 应用程序具有良好的并行性能，适合大规模具有稀疏系数矩阵的线性系统的并行求解；

（2）对于特定维数的线性方程组求解，浮点运算量为定值，迭代收敛步数也与进程数无关；随着进程数的增加，进程间通信量成比例的增加，系统开销增大；

（3）在相同进程数下，随着系数矩阵维数的增加，迭代收敛步数与系统矩阵维数正相关。

3.4 并行计算在计算流体力学中的应用[14]

随着计算机技术的迅猛发展，计算流体力学（CFD）得以迅速发展和普及。单机性能的提高使过去根本无法解算的问题在普通微机上可以解算，从而推动了 CFD 成为尖端工业、乃至一般过程工业的基本设计分析手段，从而大大激发了其应用。但是，CFD 研究人员发现硬件性能的提高随时会被无止境的需求所吞没，所以他们永远在作突破硬件性能的努力，尽管改进串行程序使用的数值算法可以提高计算效率，但并行化却几乎是唯一的大幅度提高计算效率的手段。

在20世纪80年代，向量计算机大大改善了CFD的计算速度，但是这种改善是以很高的费用为代价的。精简指令集计算机（RISC）在20世纪90年代的出现，提高了性能价格比，但是基于总线共享内存并行操作的规模不能超过8个处理器，原因是总线带宽限制了多CPU同时读取共享内存的速度。后来分布式内存（DMP）和分布共享内存（DSMP）机器的发展突破了这个瓶颈，并行计算流体力学（Parallel CFD）得到迅速发展。

Parallel CFD一般采用物理区域分割并行方法，在编程上采用单控制流多数据流（SPMD）模型，采用MPI或PVM实现消息传递，几乎适用于所有的并行机体系结构：如向量机、MPP、集群系统、SMP及其构成的星群系统，甚至是局域网连接的工作站/PC群。并行原理是：将整个流动区域分割成N个子区域分配给N个CPU计算，把子区域的初始流场信息、几何信息（网格坐标、标识号）分别装载入各子区域对应的CPU的内存中，在每一个CPU中启动计算进程，由主进程调度各CPU的计算。在每一次全场的扫描过程中，由各CPU完成子区域的计算并在边界完成数据交换（各CPU间的通信），由主进程收集全场数据完成收敛准则判别，并按需要进行写盘等其它操作。在物理模型和数值算法确定的条件下，计算速度主要取决于CPU个数、CPU性能、内存、CPU-内存访问带宽、结点互连带宽、网格质量及分区质量。每一个特定问题、每一台特定机器对应于一个最佳分区数，大量的实践会对同一类问题总结出一个最佳网格数/CPU比值。分区数过多，CPU间通信量增大，分区数增大到一定程度会反而降低计算速度；分区数过少，没有充分利用更多的CPU参与计算，也会影响计算速度；分区质量差，各CPU负载不均匀，CPU有等待现象，也影响速度。通常成熟的商业软件包如FLUENT都至少提供两种分区方法：自动分区和手动分区。自动分区的好处在于操作简便，适宜工程应用，但各CPU的负载平衡不一定能保证。手动分区恰好相反。值得注意的是，对于稳态流动，在任何一个计算时刻内存中都只保留整场的一套数据信息，故在计算过程中只存在CPU-内存、CPU-CPU（或计算结点-计算结点）之间的通讯，无需访问磁盘。但对于非定常问题，每一个时间步均产生一套场信息，每一时间步计算结束后必须将各计算结点的场信息会聚起来，执行内存到磁盘的写操作，通信量骤增，计算速度无疑会放慢。

在流动机理研究方面，区域分割并行方法仍然广为使用。如在研究湍流及其非定常性时就采用了此法。由于湍流尺度极小，故网格分辨率必须小于湍流尺度，从而使整个计算区域的网格数达到数十亿。

3.5 本章小结

并行工具箱 PETSc 为线性和非线性代数方程组的并行计算提供了可移植、可扩展的工具，非常适合偏微分方程离散后大型代数方程组的并行数值求解。本课题采用 PETSc 与有限元相结合求解势流物体绕流问题。本章主要介绍了并行计算及 PETSc 的相关知识，并在计算机机群上进行了 PETSc 的相关测试。

第四章 圆柱势流绕流的有限元并行求解

4.1 原理简介[16] [17] [19]

边长 H=20 的正方形区域（可视为无限流场）中圆柱绕流问题的物理模型（单圆柱均匀流为例）如图 4-1 所示，其中，圆柱半径 R=1，来流速度 V_0=1.0，坐标原点为圆心。

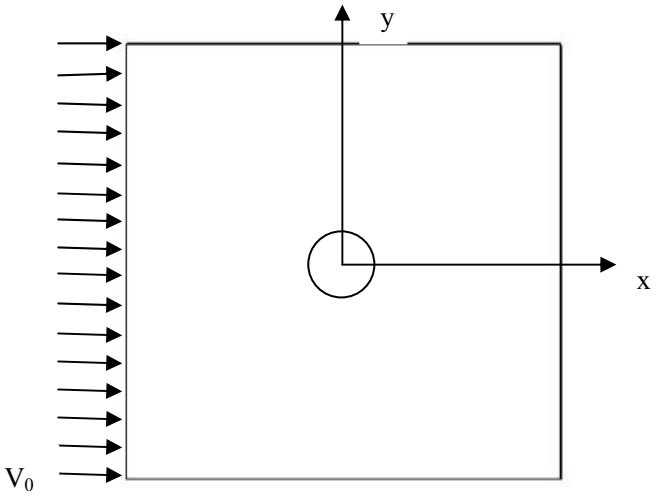

图 4-1 无限流场单圆柱绕流（均匀流）简化模型

4.1.1 控制方程与边界条件

对于二维理想不可压流体的流动，最重要的简化是引进速度势函数，并建立速度势函数所满足的 Laplace 方程。具体控制方程如下：

$$\begin{cases} \dfrac{\partial^2\varphi}{\partial x^2}+\dfrac{\partial^2\varphi}{\partial y^2}=0 & (x,y)\in\Omega \\ \varphi\big|_{\Gamma_1}=\overline{\varphi} & \\ \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_2}=g & \end{cases} \tag{4-1}$$

其中 Ω 为计算流体区域，Γ_1 为本质边界条件，Γ_2 为自然边界条件。

根据方程余量与权函数正交化原理建立起来的控制方程（4-1）的强解积分表达式：

$$\int_\Omega\left(\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}\right)\delta\varphi d\Omega=0 \tag{4-2}$$

对上式进行分部积分，应用Green公式，并带入本质边界条件和自然边界条件，即可得到弱解积分表达式：

$$\int_\Omega\left[\frac{\partial\varphi}{\partial x}\frac{\partial(\delta\varphi)}{\partial x}+\frac{\partial\varphi}{\partial y}\frac{\partial(\delta\varphi)}{\partial y}\right]d\Omega=\int_{\Gamma_2}g\delta\varphi d\Gamma \tag{4-3}$$

4.1.2 单元分析

单元分析是有限元分析中的关键一部分。根据单元类型的不同，选取插值函数的不同，所使用的单元分析方法也不同，对结果的精度也有较大的影响。本文采用三结点三角形单元对二维物体绕流问题进行单元分析，选取Lagrange线性插值基函数。下面就进行简单介绍。

取线性插值基函数：

$$\Phi_i = a_i + b_i x + c_i y \qquad (i=1,2,3) \tag{4-4}$$

根据插值基函数的定义：

$$\Phi_i(x_j^{(e)}, y_j^{(e)}) = \delta_{ij} = \begin{cases} 1 & (i=j) \\ 0 & (i \neq j) \end{cases} \qquad (i,j=1,2,3) \tag{4-5}$$

可求得插值基函数的各个系数（其中下标i、j、k按1、2、3顺序循环取值）：

$$\begin{cases} a_i = \dfrac{1}{D}(x_j^{(e)} y_k^{(e)} - x_k^{(e)} y_j^{(e)}) \\ b_i = \dfrac{1}{D}(y_j^{(e)} - y_k^{(e)}) \\ c_i = \dfrac{1}{D}(x_k^{(e)} - x_j^{(e)}) \\ D = (x_j^{(e)} - x_i^{(e)})(y_k^{(e)} - y_i^{(e)}) - (y_j^{(e)} - y_i^{(e)})(x_k^{(e)} - x_i^{(e)}) = 2A^{(e)} \end{cases} \tag{4-6}$$

其中，$x_i^{(e)}, y_i^{(e)}$ (i=1,2,3)为单元结点坐标值，$A^{(e)}$是三角形单元的面积。

单元内任意一点的速度势可表示为：$\varphi = \varphi_j \Phi_j(x,y)$，令 $\delta\varphi = \Phi_i(x,y)$ 则在任一单元内弱解积分表达式可转化为：

$$[\int_{\Omega^{(e)}} (b_i b_j + c_i c_j) dxdy]\varphi_j = \int_{\Gamma_2^{(e)}} g\Phi_i d\Gamma \qquad (i,j=1,2,3) \tag{4-7}$$

即生成的单元线性方程组为：

$$A_{ij}\varphi_j = f \qquad (i,j=1,2,3) \tag{4-8}$$

其中 $A_{ij} = (b_i b_j + c_i c_j)A^{(e)}$ ，$f_i = \int_{\Gamma_2^{(e)}} g\Phi_i d\Gamma$ 。

对于单元线积分 $f_i = \int_{\Gamma_2^{(e)}} g\Phi_i d\Gamma$，若假定 $\Gamma_2^{(e)} = \overline{23}$，且边值函数g在$\overline{23}$上认为是线性数，则通过局部坐标变换可求得单元线性方程组的右端项向量：

$$\begin{cases} I_1 = 0 \\ I_2 = \dfrac{L}{6}(2g_2 + g_3) \\ I_3 = \dfrac{L}{6}(g_2 + 2g_3) \end{cases} \tag{4-9}$$

其中L为单元自然边界$\overline{23}$的长度，即 $L = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2}$

4.1.3 总体合成

总体合成就是将单元有限元方程中的系数矩阵和右端项各个系数，按照单元结点序号与总体结点序号对应关系，累加到相应的总体有限元方程系数矩阵和右端项的元素中去。假定单元中的结点i，j对应的总体结点的序号是m，n，则进行下面所表示的累加：

$$\begin{cases} A_{mn} = A_{mn} + A_{ij} \\ f_m = f_m + f_i \end{cases} \tag{4-10}$$

4.1.4 边界条件处理

自然边界条件已经在弱解积分表达式（4-3）中得到满足，本质边界条件要通过总体有限元方程的修正得到满足。修正的方法有两种，即“消行修正法”和“对角线项扩大修正法”。这两种方法都是强制本质边界上的函数值满足本质边界条件，下面以消行修正法为例说明本质边界条件的处理过程。

假定第r号结点是本质边界上的结点，边界值为 $\overline{\varphi_r}$ 。将系数矩阵 $\{A_{mn}\}$ 中相应r的对角线元素A_{rr}用1代替，并将它所在的行和列，即第r行和第r列中的其余全部元素置于0；同时将 $\{f_m\}$ 中的第r行元素f_r改写为 $\overline{\varphi_r}$，其余的各个元素f_m均减去 $A_{mr}\overline{\varphi_r}$ ，即把f_m改写为 $f_m - A_{mr}\overline{\varphi_r}(m \neq r)$。

4.1.5 线性代数方程组的并行求解[18]

线性代数方程组的并行求解是采用PETSc的KSP（Krylov Subspace）方法。由有限元方法离散得到系数矩阵为大型稀疏对称矩阵，求解这类线性系统的ksp方法有GMRES（广义最小残差法）、BCGS（双共轭梯度法）、TFQMR（拟最小残差法）、LSQR(最小二乘QR分解)以及SYMMLQ（对称LQ分解）等方法。此外在求解过程中还可以使用Jacobi、ILU以及SOR等预条件子，目前采用较多的适用于并行计算的预条件子为Jacobi。

4.1.6 速度和压力的求解

当各个结点的速度势求出后，就可以根据单元结点速度势求解各个单元的速度。当各个单元的速度值求出后，各个结点上的速度值可取包含该结点的所有单元的速度值的平均值。而单元内或结点上的压力值可通过Bernoulli方程求解。

在三结点三角形单元内，任一点的速度势可表示为：

$$\varphi = \sum_{i=1}^{3} \varphi_j \Phi_j(x,y) = \sum_{i=1}^{3} \varphi_j (a_j + b_j x + c_j y) \tag{4-11}$$

由于速度势插值基函数为线性函数，因此各个单元内的速度值为常数，速度分量的表达式为：

$$\begin{cases} u = \dfrac{\partial \varphi}{\partial y} = \varphi_j c_j \\ v = -\dfrac{\partial \varphi}{\partial x} = -\varphi_j b_j \end{cases} \tag{4-12}$$

其中b_j，c_j的值由方程（4-4）给出。

当各个单元的速度值求出后，各个结点上的速度值可取包含该结点的所有单元的速度值的平均值。当速度值获得后，可通过Bernoulli方程求解压力。考虑到重力场的定常运动的Bernoulli方程为：

$$\frac{v^2}{2} + \frac{P}{\rho} + gz = C \tag{4-13}$$

其中C为常数。

4.1.7 结果后处理

结果的后处理使用GMV（General Mesh Viewer）图形处理软件。GMV由美国Los Alomos国家实验室开发，本文采用GMV软件作流函数分布图、势函数分布图、压力分布图以及速

度矢量图。

4.2 单圆柱绕流

4.2.1 数值模拟结果[17]

单圆柱绕流问题的原理已在上节中详细介绍过了，需要指出的是，正方形区域的上壁面Γ_1、下壁面Γ_2及入流面Γ_3均为自然边界，出流面Γ_4为本质边界。即：

$$\begin{cases} \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_1\Gamma_2} = 0 \\ \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_3} = -1 \\ \varphi|_{\Gamma_4} = 10 \end{cases} \tag{4-14}$$

本文计算单圆柱绕流时使用的计算网格共有11176个结点，22152个三角形单元，在圆柱周围的网格较密。

通过计算所得的势函数分布如图4-2所示，流函数分布如图4-3所示，压力分布如图4-4所示，速度矢量分布如图4-5所示。从图4-2和4-3中可以明显看出整个流场中的势函数和流函数的分布情况，势函数从入流面的φ=-10变化至出流面的φ=10，流函数从下壁面的ψ=-10变化至上壁面的ψ=10；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到圆柱绕流影响的区域压力与速度分别为0和1，没有变化；而在圆柱周围的压力和速度变化就非常明显，下面就着重研究圆柱周围结点的压力以及沿y轴的速度变化。

为了验证结果的准确性，将所得的并行计算结果与解析解进行比较。

由于无限域中均匀流下绕流的静止圆柱具有轴对称性，故其速度势在极坐标中可表示为：

$$\varphi = U_0(r + \frac{a^2}{r})\cos\theta \tag{4-15}$$

相应的速度分量为：

$$\begin{cases} v_r = \dfrac{\partial\varphi}{r} = U_0(1 - \dfrac{a^2}{r^2})\cos\theta \\ v_\theta = \dfrac{1}{r}\dfrac{\partial\varphi}{\theta} = -U_0(1 + \dfrac{a^2}{r^2})\sin\theta \end{cases} \tag{4-16}$$

利用不考虑重力场的定常运动的Bernoulli方程可求得压力的表达式为：

$$p = p_0 + \frac{1}{2}\rho U_0^2(1 - 4\sin^2\theta) \tag{4-17}$$

其中，来流压力 p_0=0，流体介质密度 ρ=1，来流 U_0=1。

首先对圆柱周围的结点压力值进行比较。从图 4-6 中的比较结果可以看出，数值计算解与近似解析解吻合较好。但是需注意到，在 π/2 和 3π/2 处两者之间存在较大误差，这两点是圆柱周围结点压力最大处且它们周围的压力变化较为剧烈。

接下来比较沿 y 轴方向的速度值。本文选取了从 y=1 开始 y 坐标成等比数列（公比为 1.08）的一系列点，比较结果如图 4-7 所示。两者之间存在着一定的误差，这是由于在数值

计算中处理速度的近似性，选取结点速度值为包含该结点的单元速度值。但是又因为网格精度较高，通过此处理方法得出的速度值还是有较高的准确性，因此，从图中可以看出，速度值的数值解与近似解析解基本吻合。从图中可以看出，在 y=4 附近速度接近 1，说明从此处开始流场内远离圆柱的区域受圆柱绕流的影响很小，可以说不受其影响。

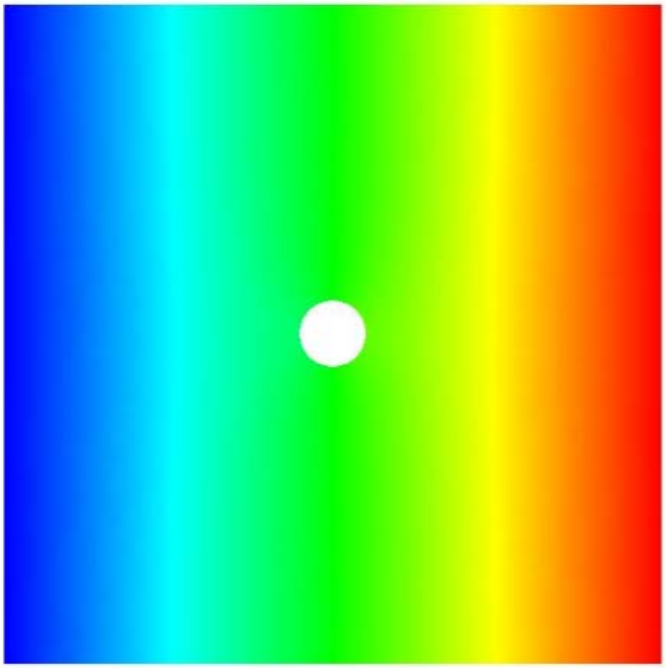

图 4-2 势函数分布图

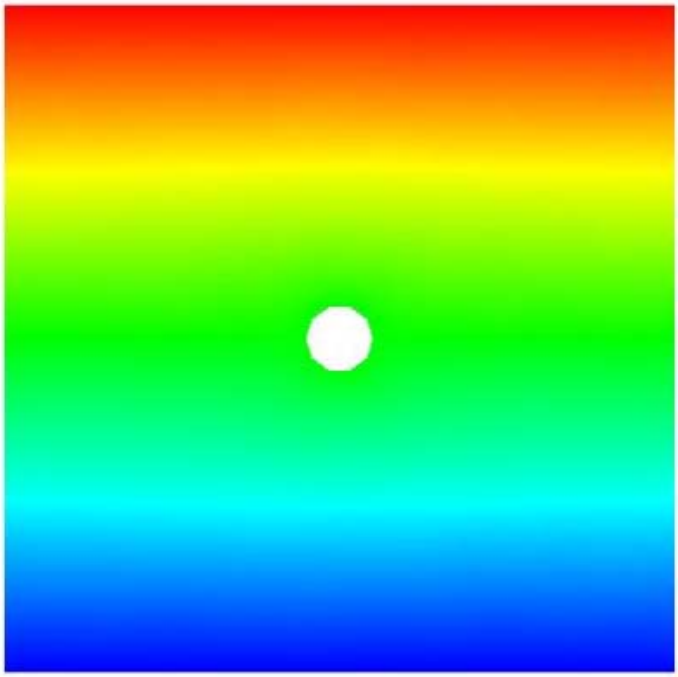

图 4-3 流函数分布图

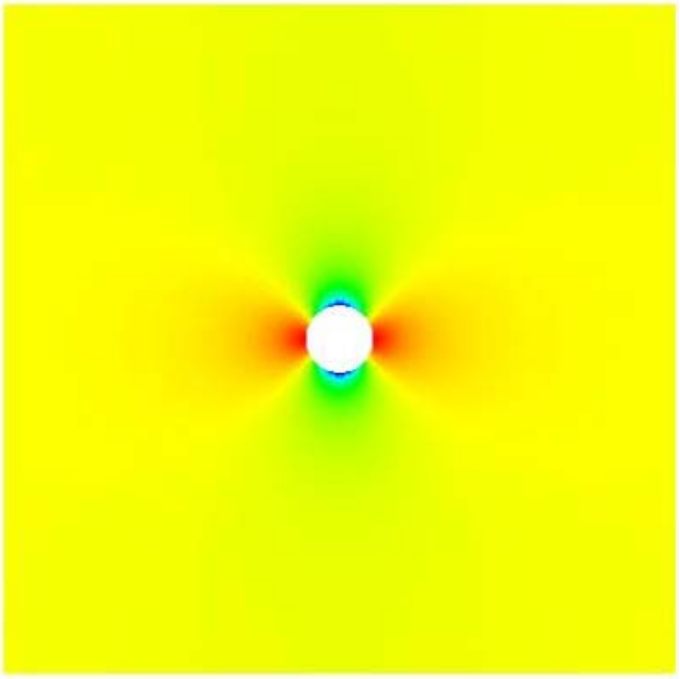

图 4-4 压力分布图

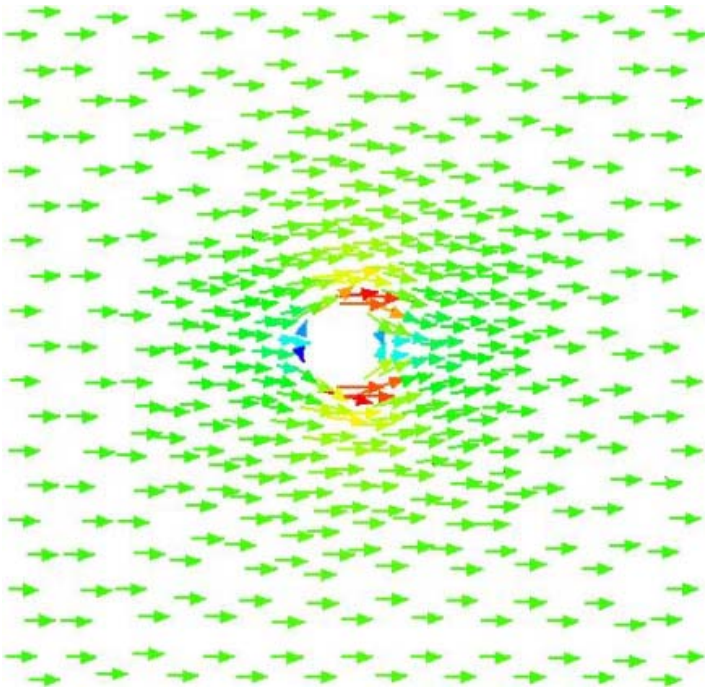

图 4-5 速度矢量分布图

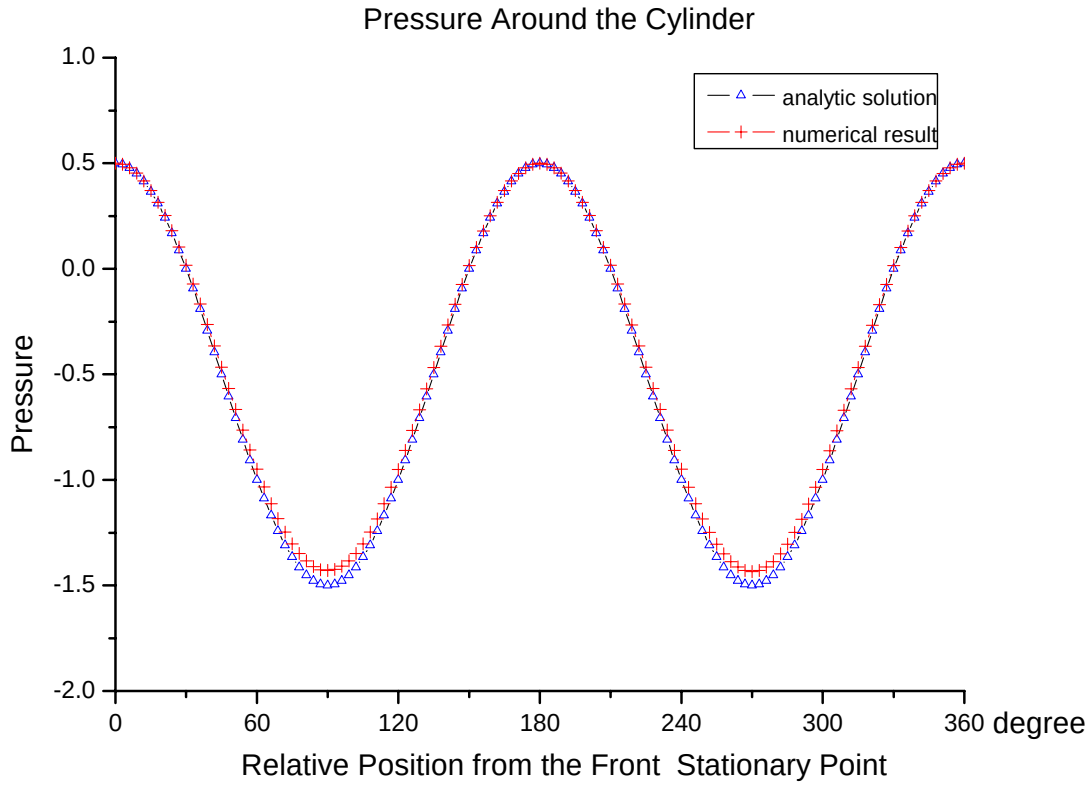

图 4-6 计算压力值与近似解析解的比较

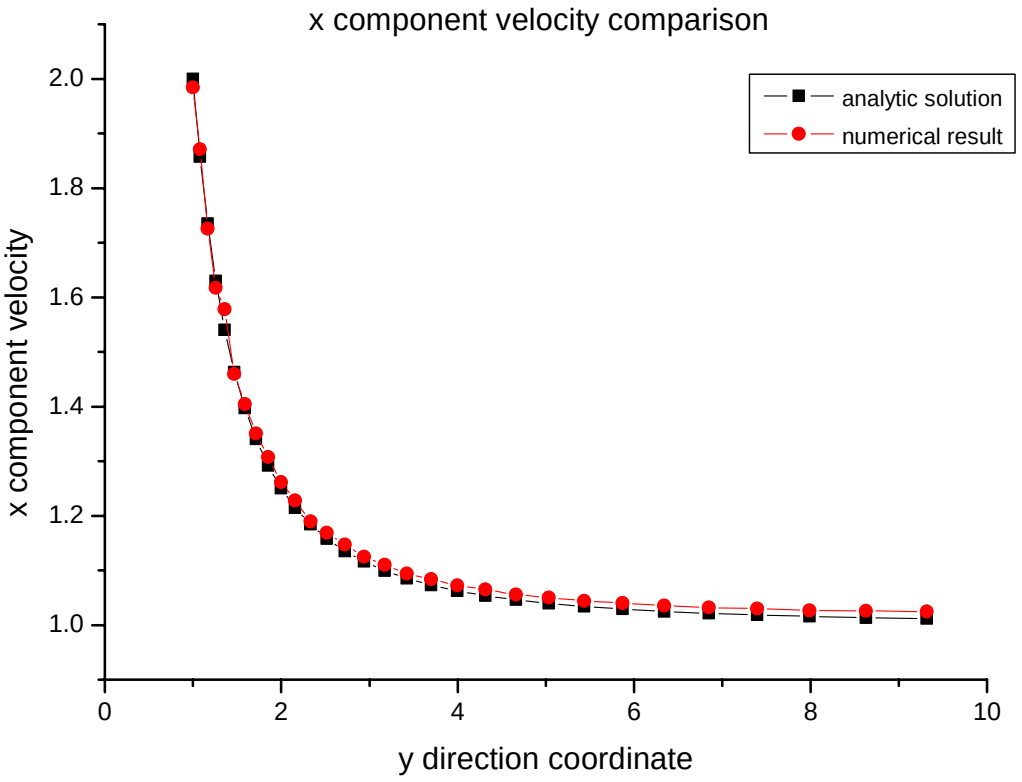

图 4-7 计算速度值与近似解析解的比较

4.2.2 并行效率验证

本文通过在 PC-Cluster 上实现并行计算来求解单圆柱绕流问题，下面就分别测试在不同进程数下求解的性能，以期得到最理想的并行效率。

表 4-1 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	3.919e+01	1.507e+01	9.224e+00	7.347e+00
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	1.669e+09	8.353e+08	5.568e+08	4.177e+08
Flops/sec:	4.259e+07	7.115e+07	9.391e+07	1.040e+08
MPI Message:	0.000e+00	1.899e+03	3.798e+03	5.697e+03
MPI Message Lengths:	0.000e+00	7.917e+06	1.454e+07	1.531e+07
MPI Reductions:	3.743e+03	1.875e+03	1.250e+03	9.375e+02
	5 proc	6 procs	7 procs	8 procs
Time (sec):	6.835e+00	6.644e+00	6.308e+00	6.973e+00
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	3.341e+08	2.785e+08	2.387e+08	2.089e+08
Flops/sec:	9.550e+07	8.327e+07	7.927e+07	5.749e+07
MPI Message:	5.705e+03	7.604e+03	7.612e+03	9.511e+03
MPI Message Lengths:	1.564e+07	1.590e+07	1.574e+07	1.581e+07
MPI Reductions:	7.500e+02	6.250e+02	5.357e+02	4.688e+02

从程序运行性能的结果比较中可以看出，随着进程数的增加，通信开销增大，浮点运算总量减小，每秒浮点运算次数先增大后减小。这就导致了开始几个进程中（进程数从 1 到 4）随着进程数增加，由于每秒浮点运算次数在增加，计算时间缩短得比较明显；当从进程数为 5 开始，随着进程数的增加，由于每秒浮点运算数减小，计算时间缩短的较为平缓直至计算时间反而有所增加。因此，对于此并行求解并非进程数越大越好，应根据实际情况选取适合的进程数，以获得最小的计算时间。从图 4-8 可以明显看出，进程数为 7 时，所需时间最少。

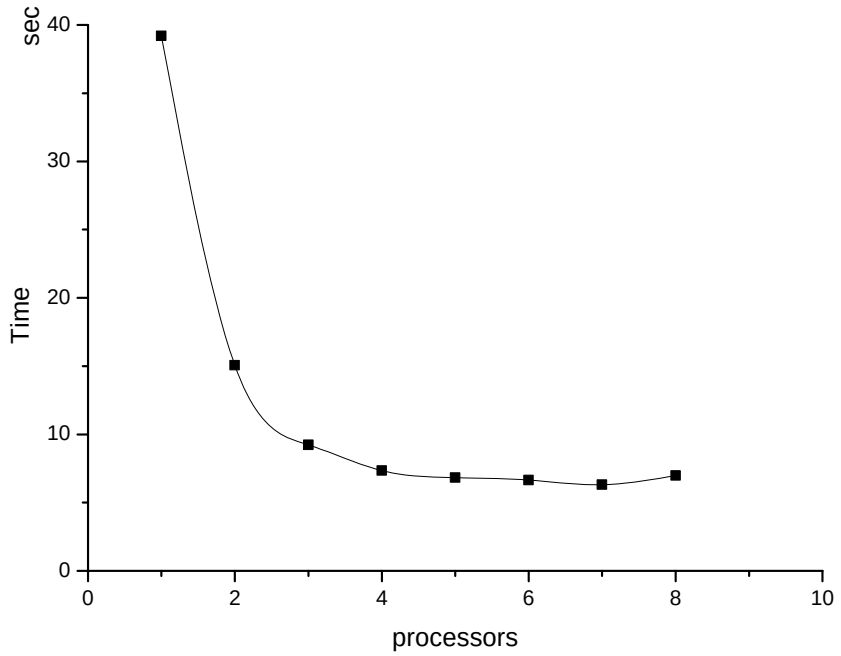

图 4-8 不同进程数下的计算时间的比较

4.3 双圆柱绕流

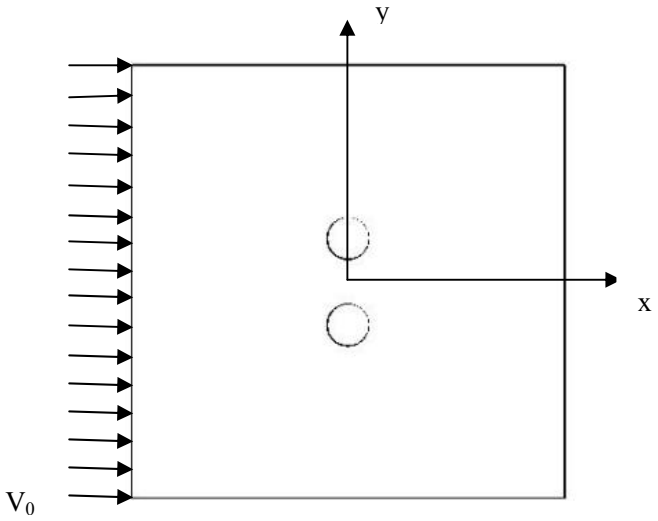

图 4-9 无限流场双圆柱绕流（均匀流）简化模型

4.3.1 数值模拟结果

双圆柱绕流的物理模型类似于单圆柱绕流（图 4-9），本文研究对称的并列双圆柱。其中，正方形区域的边长 H=20；圆柱均为半径 R=1，来流速度 V_0=1.0，两圆柱圆心相距 d=4，坐标原点为两圆柱的对称中心。

双圆柱绕流问题的原理已在4.1节中详细介绍过了，需要指出的是，正方形区域的上壁面Γ_1、下壁面Γ_2及入流面Γ_3均为自然边界，出流面Γ_4为本质边界。即：

$$\begin{cases} \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_1\Gamma_2} = 0 \\ \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_3} = -1 \\ \left.\varphi\right|_{\Gamma_4} = 10 \end{cases} \tag{4-18}$$

本文计算双圆柱绕流时使用的计算网格共有20032个结点，39746个三角形单元，在圆柱周围的网格较密。

通过计算所得的势函数分布如图4-10所示，流函数分布如图4-11所示，压力分布如图4-12所示，速度矢量分布如图4-13所示。从图4-10和4-11中可以明显看出整个流场中的势函数和流函数的分布情况，势函数从入流面的φ=-10变化至出流面的φ=10，流函数从下壁面的ψ=-10变化至上壁面的ψ=10；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到双圆柱绕流影响的区域压力与速度分别为0和1，没有变化；而在圆柱周围的压力和速度变化就非常明显，由于对称性，本文只需研究x轴上方的圆柱即可。下面就着重研究圆柱周围结点的压力以及沿y轴的速度变化。

首先，通过并行计算求出圆柱周围结点压力值，其结果如图4-14；从此图看出圆柱周围压力变化类似于单圆柱，两种情况最大区别在于，两波谷值（π/2和3π/2处）压力不等，由于两圆柱相互作用导致3π/2处圆柱的压力最大，压力分布图也验证了这点。接下来计算求出沿y轴方向的速度值，本文选取了从y=3开始纵坐标成等比数列（公比为1.05）的一系列点，结果如图4-15所示；从图中可以看出，在y=7附近速度接近1，说明从此处开始流场内远离圆

柱的区域受圆柱绕流的影响很小，可以说不受其影响。最后研究两圆柱之间沿y轴方向的速度分布，选取从坐标原点开始，纵坐标成等差数列（公差为0.05）的一系列点，结果如图4-16所示，从图中可以看出，越接近圆柱，速度值越大。

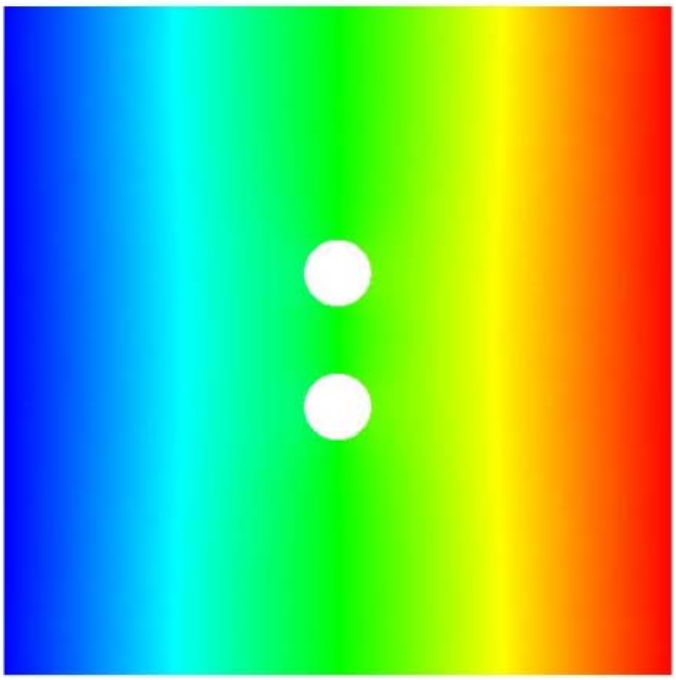

图 4-10 势函数分布图

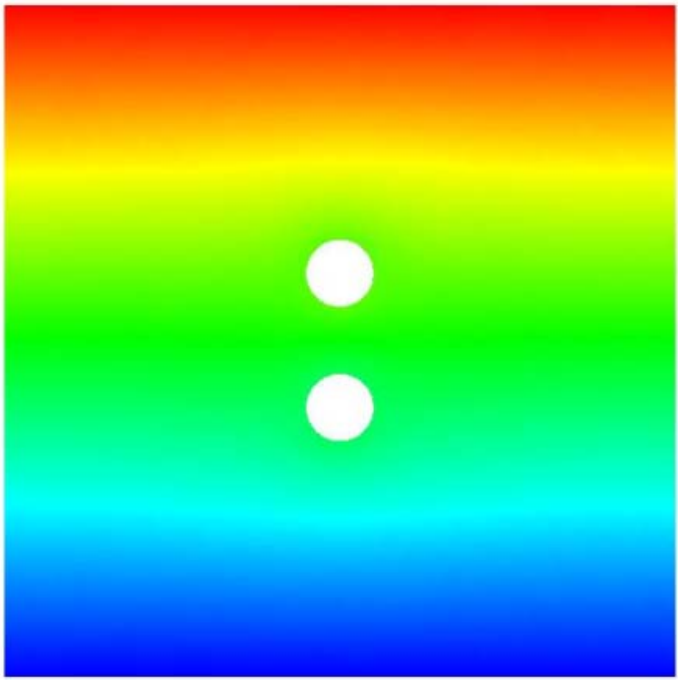

图 4-11 流函数分布图

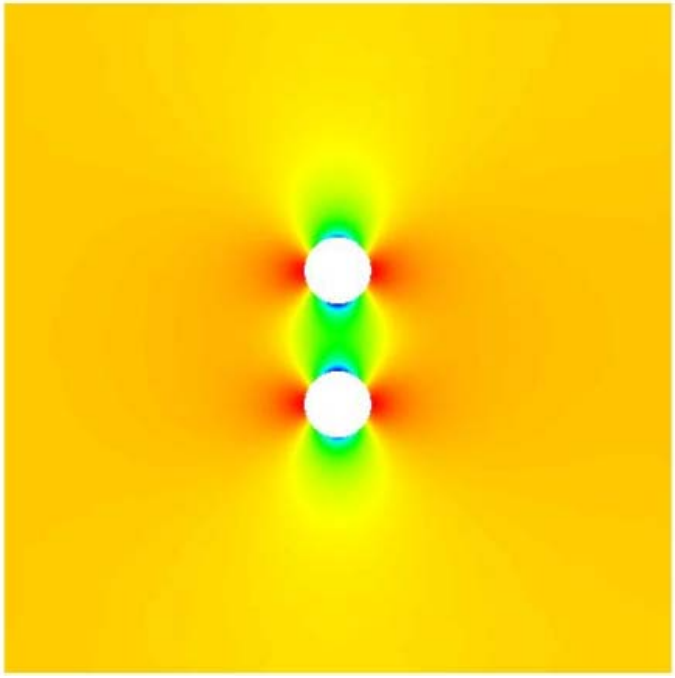

图 4-12 压力分布图

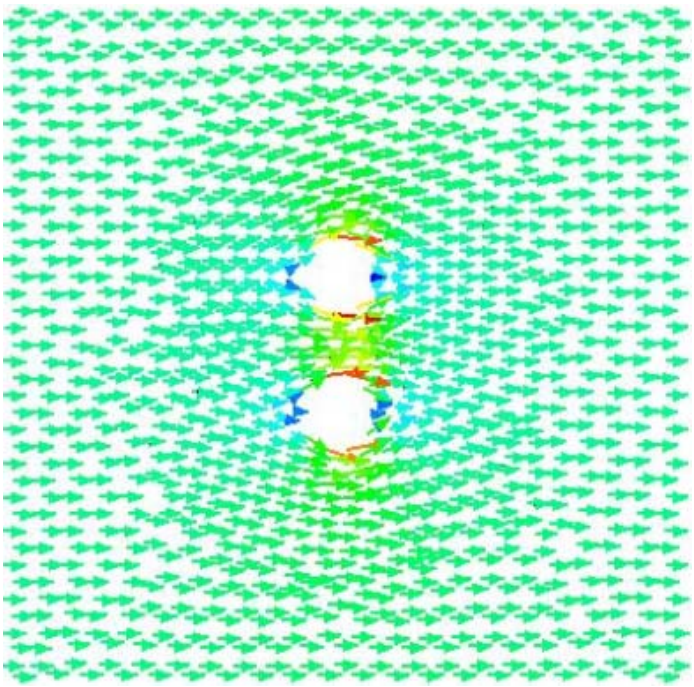

图 4-13 速度矢量分布图

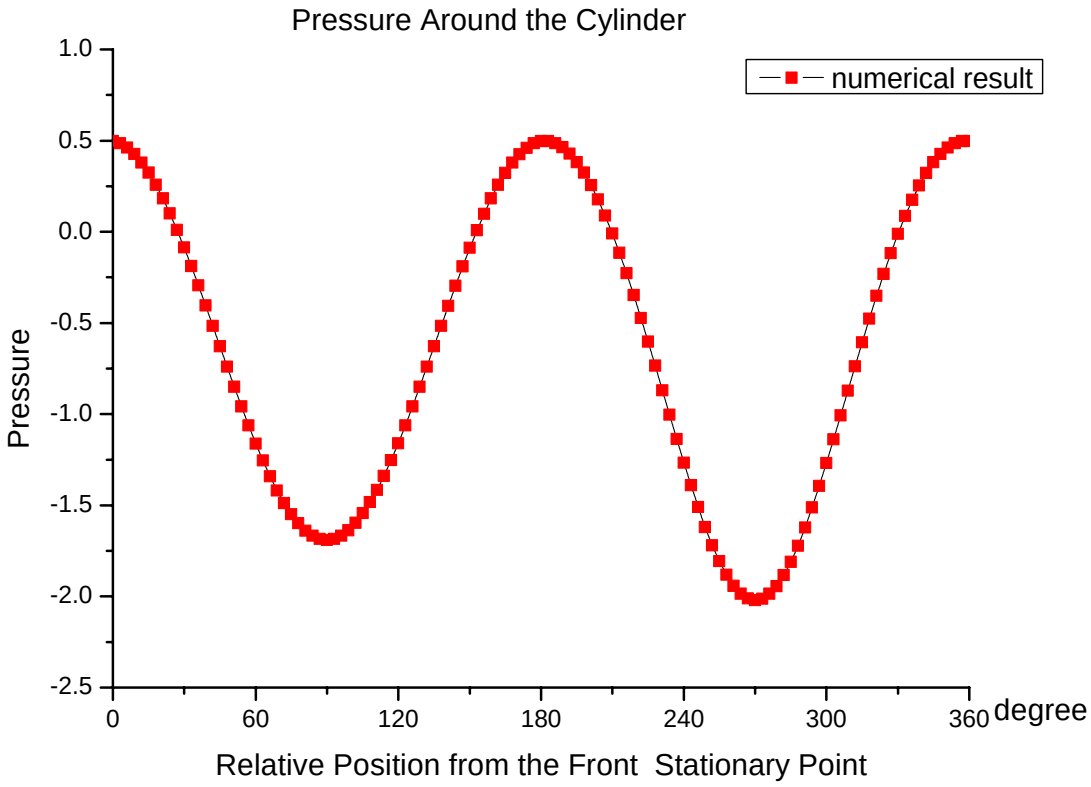

图 4-14 计算压力值

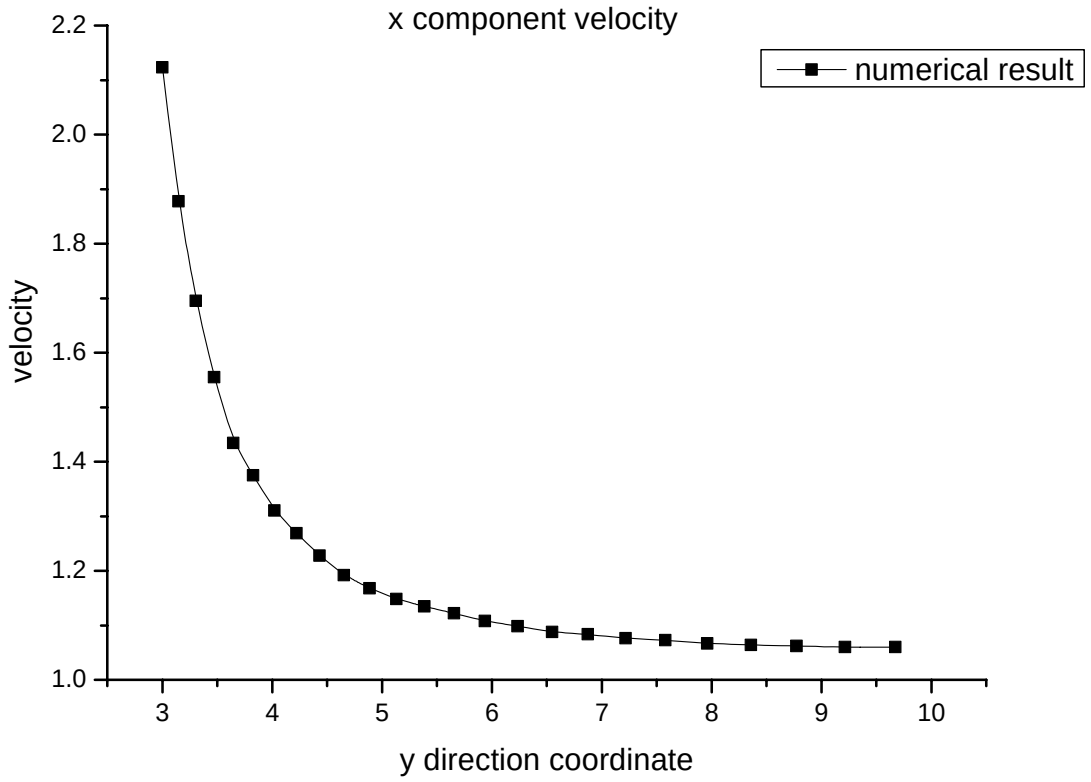

图 4-15 计算速度值

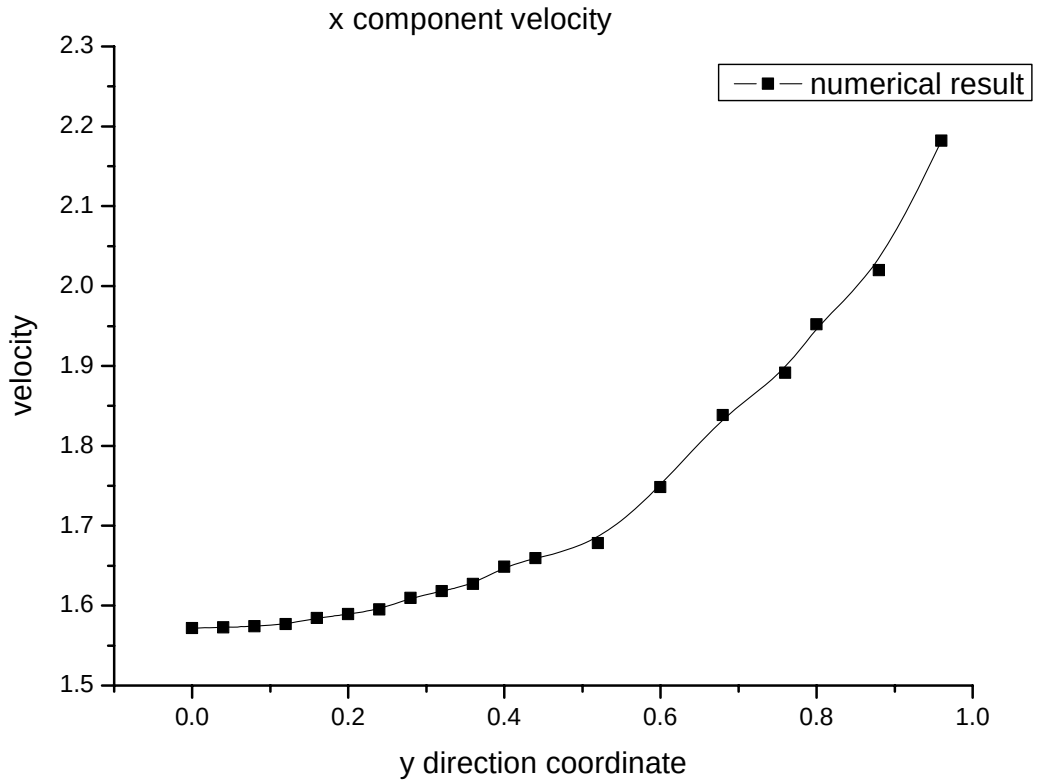

图 4-16 计算速度值

4.3.2 并行效率验证

本文通过在 PC-Cluster 上实现并行计算来求解双圆柱绕流问题，下面就分别测试在不同进程数下求解的性能，以期得到最理想的并行效率。

表 4-2 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	2.258e+02	8.019e+01	6.583e+01	4.997e+01
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	8.066e+09	2.824e+09	2.999e+09	1.678e+09
Flops/sec:	3.572e+07	4.024e+07	5.395e+07	4.213e+07
MPI Message:	0.000e+00	3.578e+03	1.139e+04	8.506e+03
MPI Message Lengths:	0.000e+00	2.423e+07	7.160e+07	5.574e+07
MPI Reductions:	1.006e+04	3.526e+03	3.740e+03	2.094e+03
	5 procs	6 procs	7 procs	8 procs
Time (sec):	4.276e+01	3.517e+01	3.573e+01	3.820e+01
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	1.183e+09	9.880e+08	7.928e+08	7.168e+08
Flops/sec:	3.645e+07	3.942e+07	3.169e+07	2.556e+07
MPI Message:	1.126e+04	1.128e+04	1.407e+04	1.455e+04
MPI Message Lengths:	4.831e+07	4.849e+07	4.605e+07	4.759e+07
MPI Reductions:	1.478e+03	1.233e+03	9.893e+02	8.948e+02

从程序运行性能的结果比较中可以看出，当程序由单进程运行变为双进程运行时，运行时间大大缩短了。随着进程数的增加，通信开销增大，浮点运算总量减小，每秒浮点运算次

数变化有所波动，总体上是先增大后减小。这就导致了开始几个进程中（进程数从 1 到 4）随着进程数增加，由于每秒浮点运算次数在增加，计算时间缩短得比较明显；当从进程数为 5 开始，随着进程数的增加，由于每秒浮点运算数减小，计算时间缩短的较为平缓直至计算时间反而有所增加。因此，对于此并行求解并非进程数越大越好，应根据实际情况选取适合的进程数，以达到最小的计算时间。从图 4-17 可以明显看出，进程数为 6 时，所需时间最少。

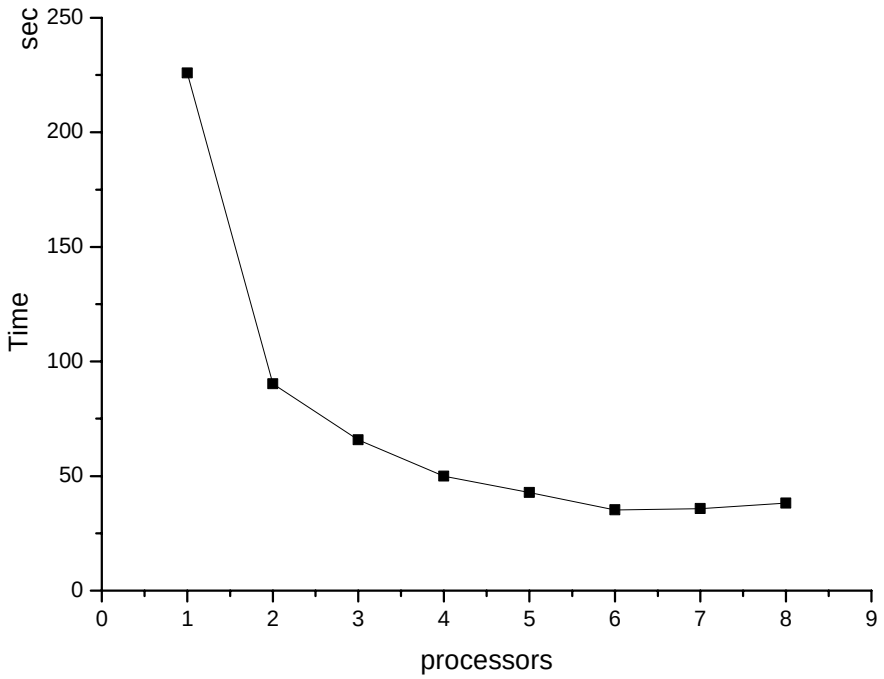

图 4-17 不同进程数下的计算时间的比较

4.4 本章小结

本章首先介绍了圆柱绕流数值模拟的原理及方法，分别通过对单圆柱及双圆柱的绕流流场的数值模拟，给出了绕流速度势分布、流线分布、速度分布以及压力分布图，并分析了进程数与并行计算效率的关系，同时给出了单圆柱绕流流场速度分布和物体表面压力分布的数值结果与解析解比较，两者吻合很好。证明了本课题采用的有限元法与 PETSc 相结合能高效求解圆柱（包括单双圆柱）绕流问题。

第五章 机翼势流绕流的有限元并行求解

5.1 单机翼（NACA0010-66）绕流

单机翼绕流的物理模型类似于单圆柱绕流（图 5-1）。其中，正方形区域的边长 H=5，来流速度 V_0=1.0，坐标原点为机翼中心。

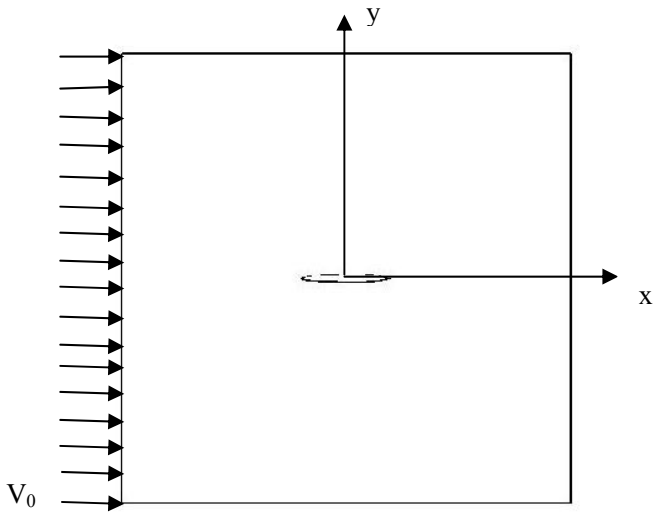

图 5-1 无限流场单机翼绕流（均匀流）简化模型

单机翼绕流问题的原理已在4.1节中详细介绍过了，需要指出的是，正方形区域的上壁面Γ_1、下壁面Γ_2及入流面Γ_3均为自然边界，出流面Γ_4为本质边界。即：

$$\begin{cases} \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_1\Gamma_2} = 0 \\ \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_3} = -1 \\ \left.\varphi\right|_{\Gamma_4} = 2.5 \end{cases} \tag{5-1}$$

本文计算单机翼（翼型NACA0010-66）绕流时使用的计算网格共有18936个结点，37672个三角形单元，在机翼周围的网格较密。

5.1.1 无攻角来流下数值模拟结果

通过计算所得的势函数分布如图5-2所示，流函数分布如图5-3所示，压力分布如图5-4所示，速度矢量分布如图5-5所示。从图5-2和5-3中可以明显看出整个流场中的势函数和流函数的分布情况，势函数从入流面φ=-2.5变化至出流面φ=2.5，流函数从下壁面的ψ=-2.5变化至上壁面的ψ=2.5；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到机翼绕流影响的区域压力与速度分别为0和1，没有变化；而在机翼周围的压力和速度变化就非常明显。下面就着重研究机翼周围结点的压力、速度值以及沿y轴的速度变化。

为了验证结果的准确性，将所得的并行计算结果与经验值进行比较。

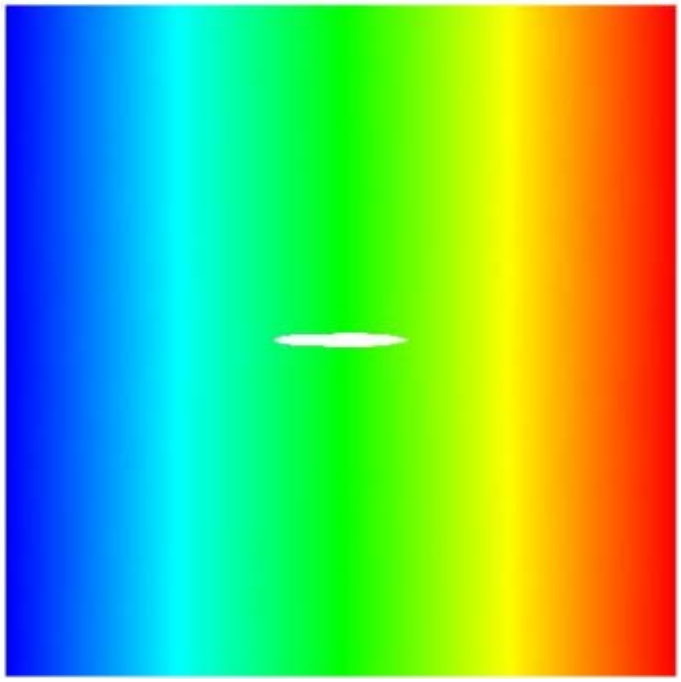

图 5-2 势函数分布图

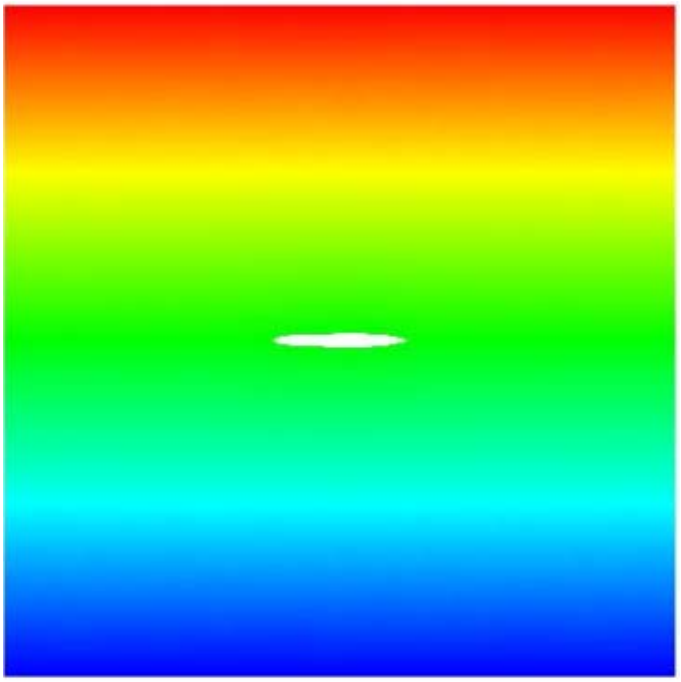

图 5-3 流函数分布图

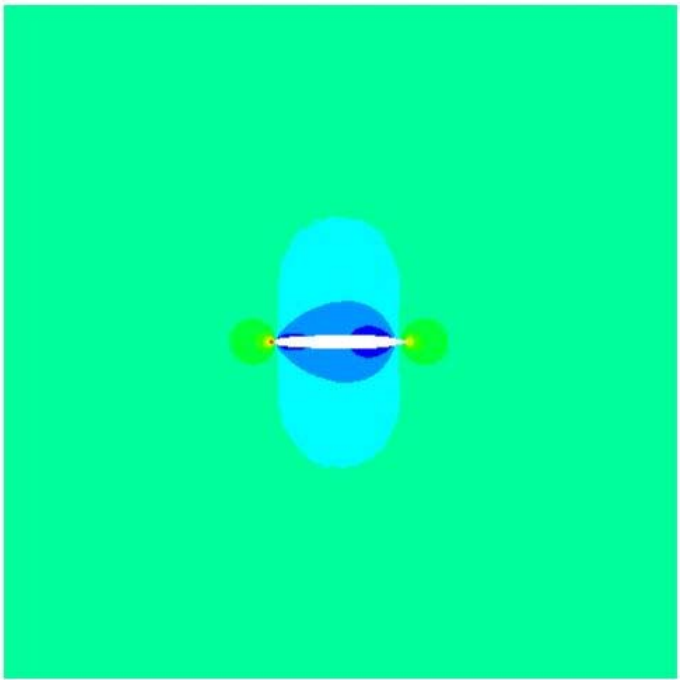

图 5-4 压力分布图

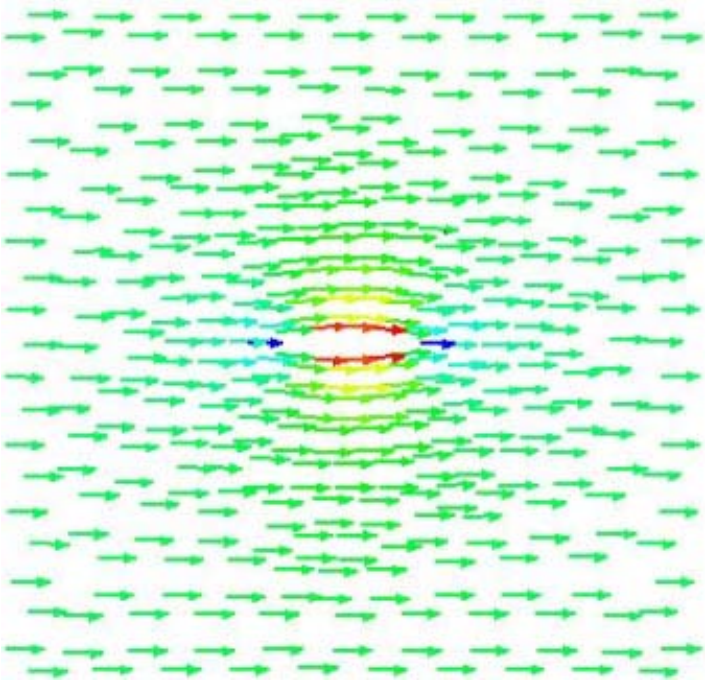

图 5-5 速度矢量分布图

首先对机翼周围点的压力值进行比较。从图 5-6 中的比较结果可以看出，数值计算解与经验值吻合较好。接下来对机翼周围点的速度值进行比较。从图 5-8 中的比较结果可以看出，数值计算解与经验值吻合较好。最后比较沿 y 轴方向的速度值。本文选取了从 y=0.05 开始

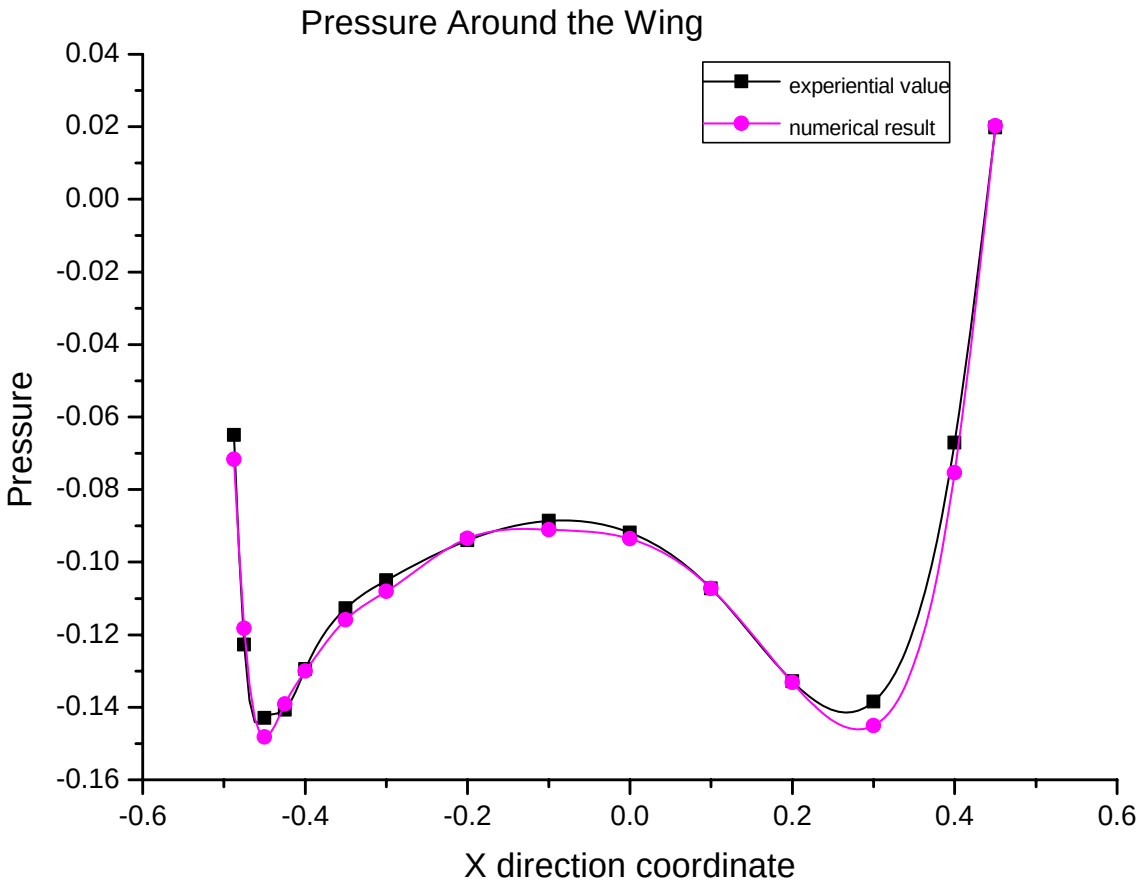

图 5-6 计算压力值与经验值的比较

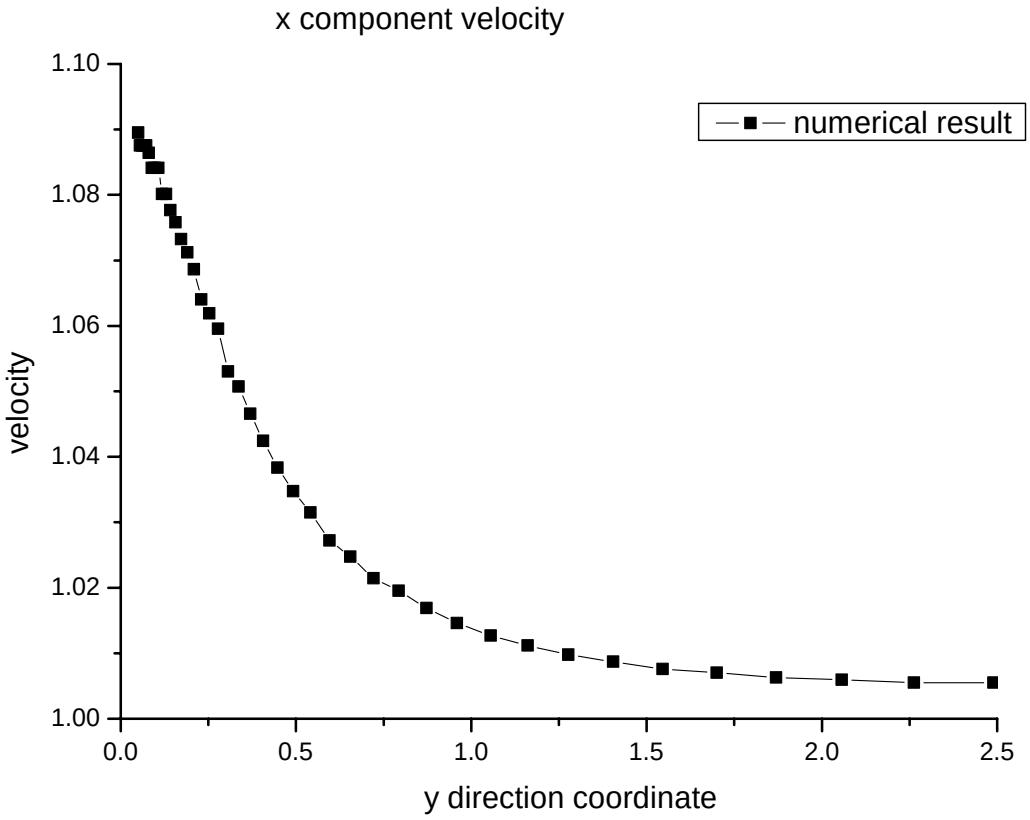

图 5-7 计算速度值

y 坐标成等比数列（公比为 1.1）的一系列点，比较结果如图 5-7 所示。从图中可以看出，速度值沿着远离机翼的方向逐渐减小，在 y=1.5 附近速度接近 1，说明从此处开始流场内远离机翼的区域受圆柱绕流的影响很小，可以说不受其影响。

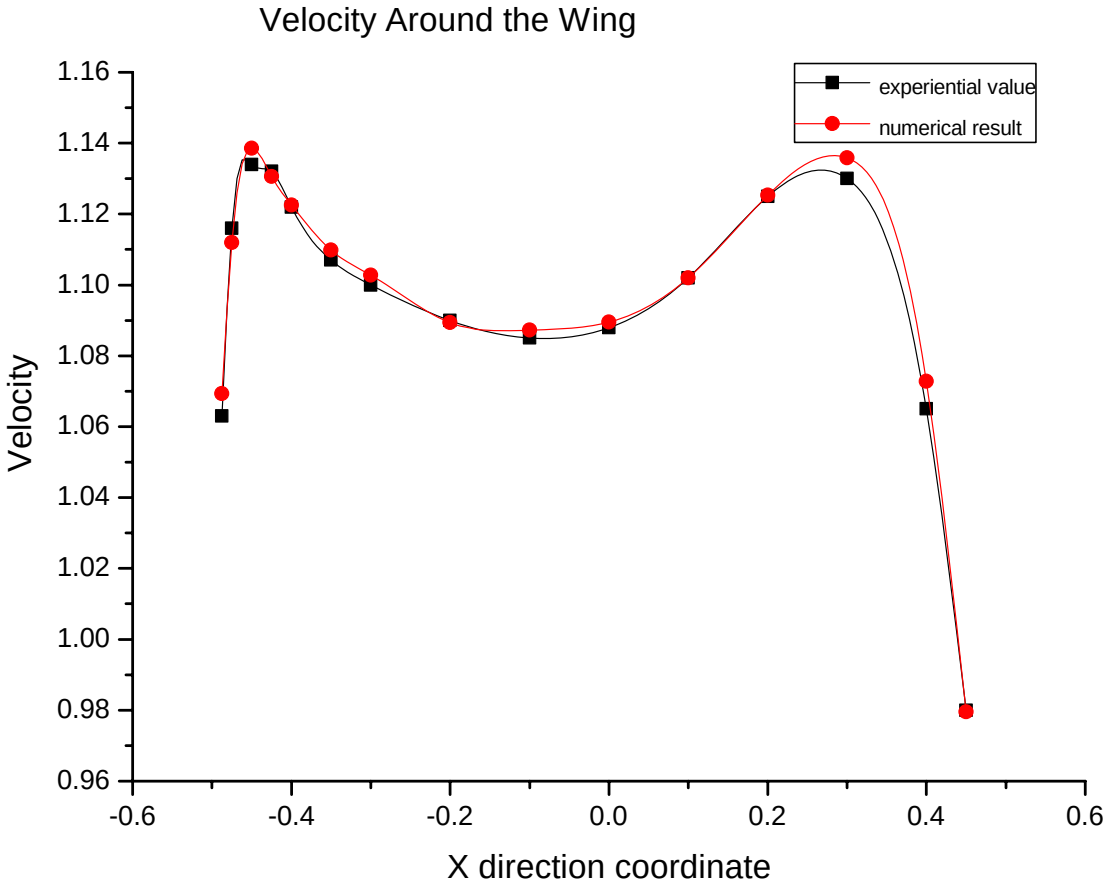

图 5-8 计算速度值与经验值的比较

5.1.2 有攻角来流下数值模拟结果

通过计算所得的势函数分布如图 5-9 所示，流函数分布如图 5-10 所示，压力分布如图 5-11 所示，速度矢量分布如图 5-12 所示。从图 5-9 和 5-10 中可以明显看出整个流场中的势函数和流函数的分布情况，它与无攻角来流区别在于等势线和流线不再竖直和水平而变为斜向；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到机翼绕流影响的区域压力与速度分别为 0 和 1，没有变化；而在机翼周围的压力和速度变化就非常明显。下面就着重研究在不同攻角均匀来流下机翼周围结点的压力值以及沿 y 轴的速度变化，本文分别选取了无攻角、5^o 攻角以及 10^o 攻角这三种情况进行比较。

首先对在不同攻角均匀来流下机翼周围点的压力值进行比较，机翼上半部分压力比较结果如图 5-13，机翼下半部分压力比较结果如图 5-14。接着比较沿 y 轴方向的速度值。本文分别选取了从 y=0.05 及 y=-0.05 开始 y 坐标成等比数列（公比为 1.1）的一系列点，比较结果如图 5-15 及图 5-16 所示。从图中可以看出，不同攻角下速度变化趋势相同，速度值都是沿着远离机翼的方向逐渐减小。

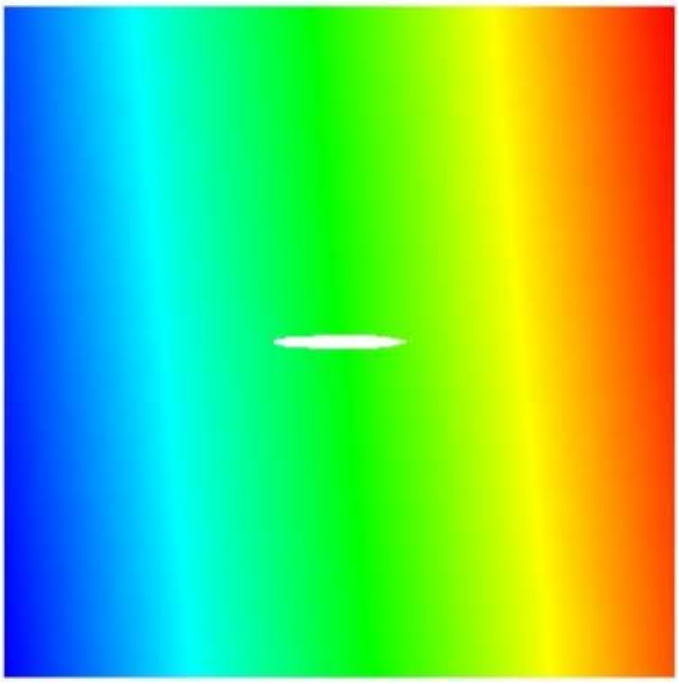

图 5-9 势函数分布图

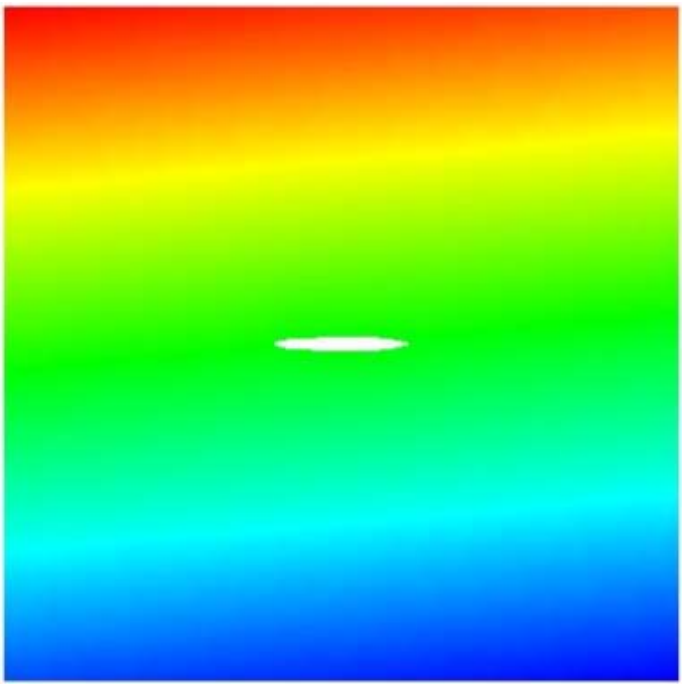

图 5-10 流函数分布图

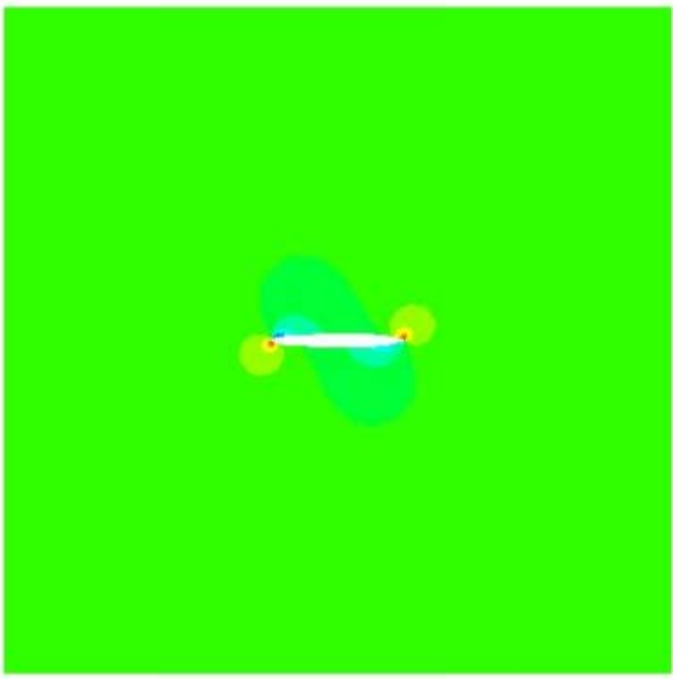

图 5-11 压力分布图

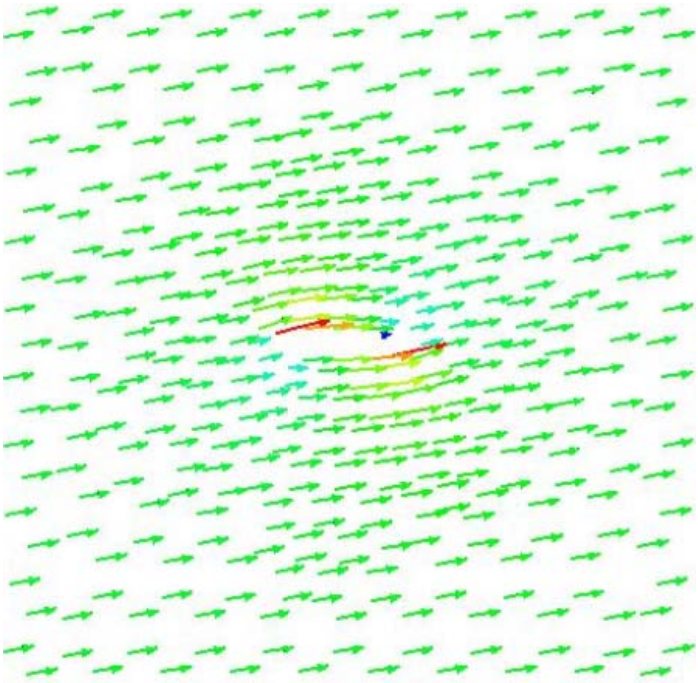

图 5-12 速度矢量分布图

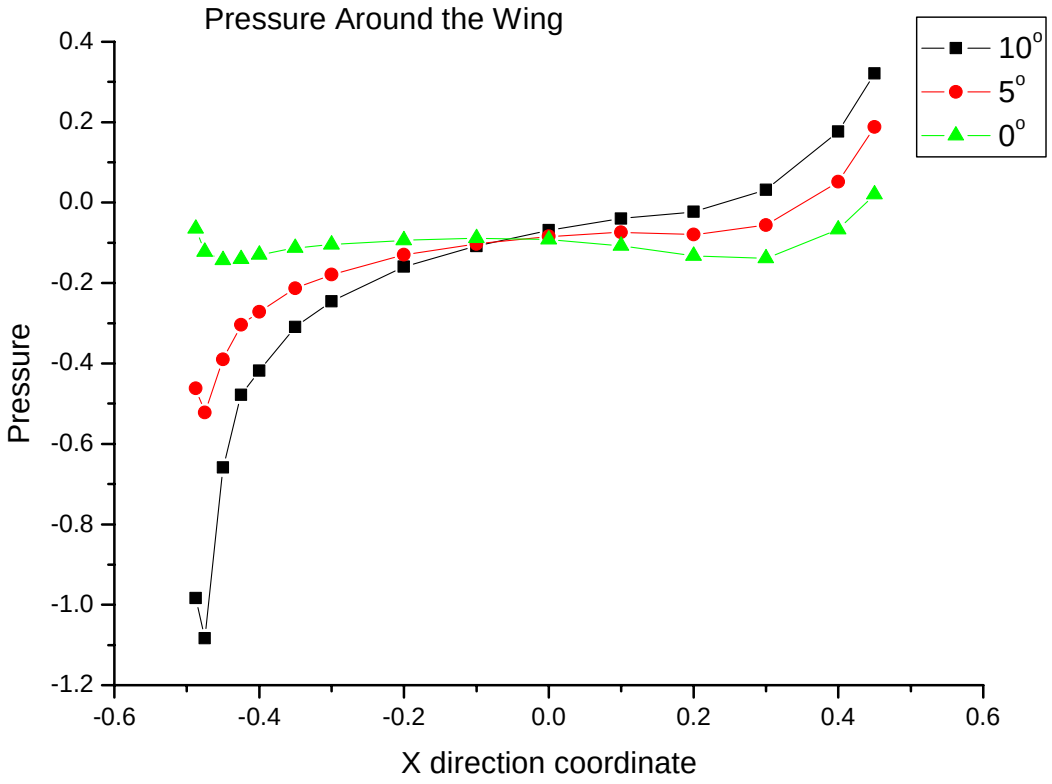

图 5-13 不同攻角来流下计算压力值比较

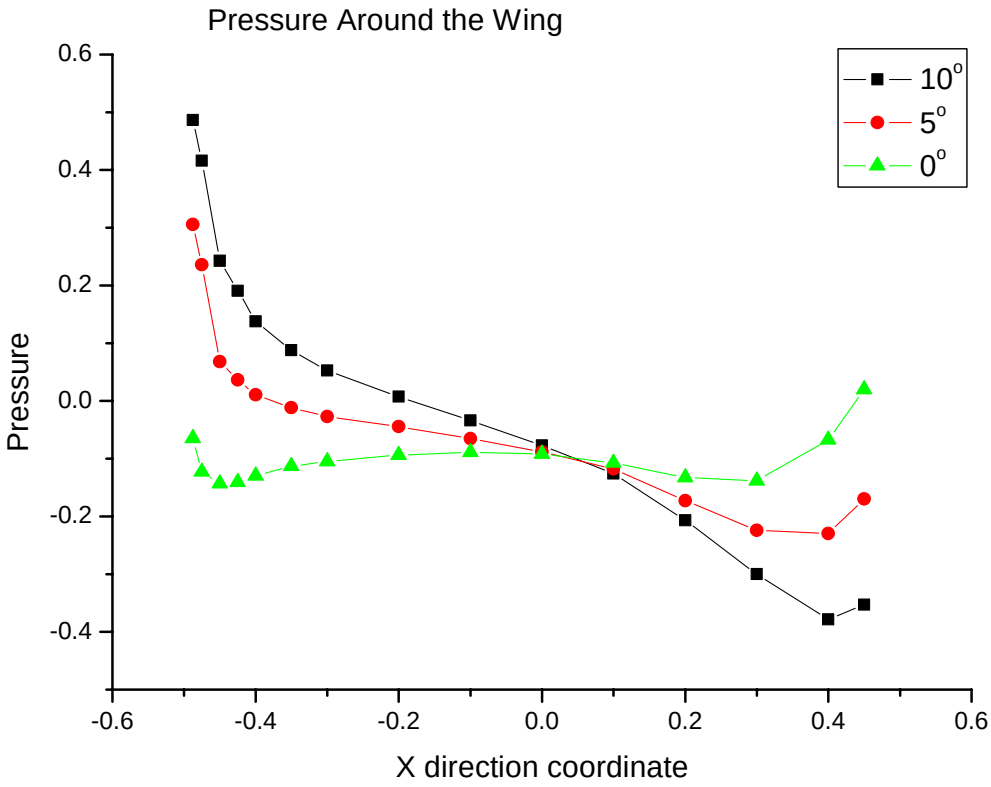

图 5-14 不同攻角来流下计算压力值比较

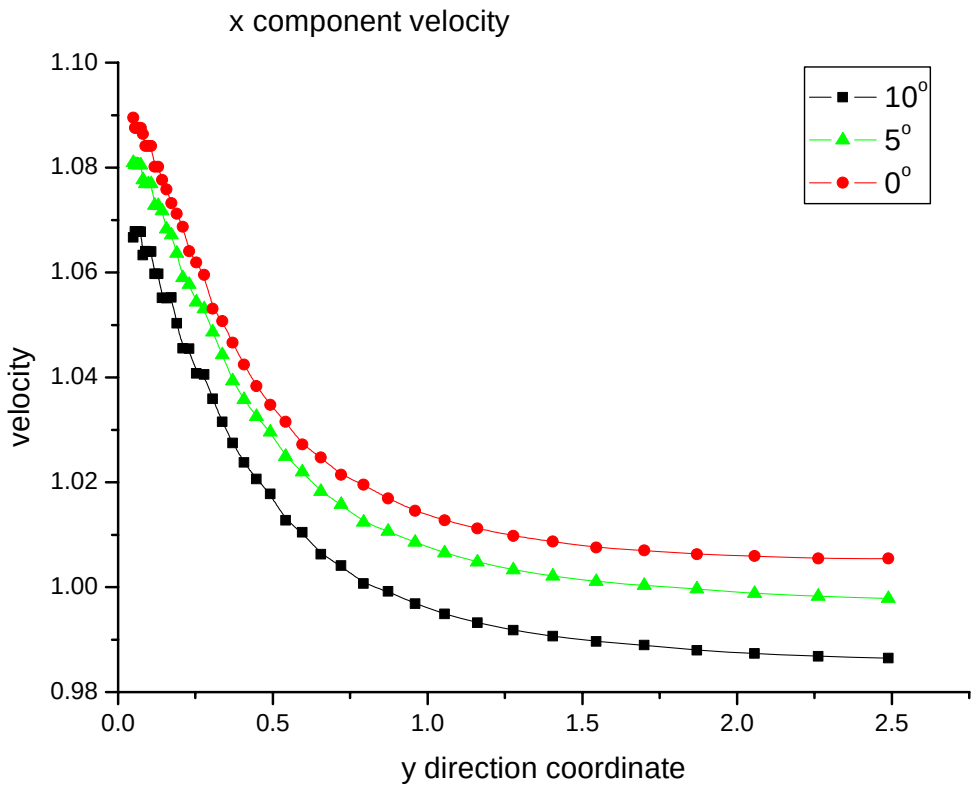

图 5-15 不同攻角来流下计算速度值比较

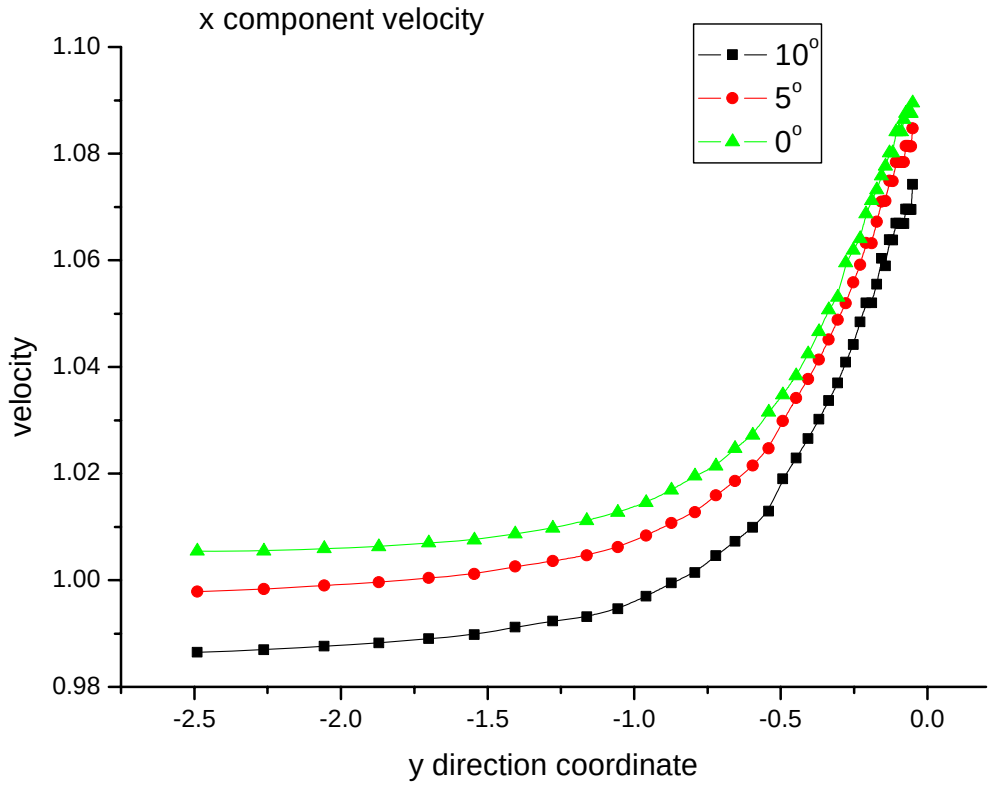

图 5-16 不同攻角来流下计算速度值比较

5.1.3 并行效率验证

本文通过在 PC-Cluster 上实现并行计算来求解翼型 NACA0010-66 单机翼绕流问题，下面就分别测试在不同进程数下求解的性能，以期得到最理想的并行效率。

表 5-1 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	1.863e+02	6.816e+01	4.657e+01	4.069e+01
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	4.510e+09	2.299e+09	1.679e+09	1.271e+09
Flops/sec:	2.420e+07	3.890e+07	4.490e+07	4.060e+07
MPI Message:	0.000e+00	3.083e+03	6.756e+03	6.826e+03
MPI Message Lengths:	0.000e+00	1.504e+07	3.076e+07	3.269e+07
MPI Reductions:	5.964e+03	3.040e+03	2.220e+03	1.680e+03
	5 procs	6 procs	7 procs	8 procs
Time (sec):	3.368e+01	3.147e+01	2.852e+01	3.270e+01
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	9.560e+08	8.201e+08	7.104e+08	6.203e+08
Flops/sec:	3.913e+07	3.682e+07	3.718e+07	2.648e+07
MPI Message:	9.611e+03	9.907e+03	1.334e+04	1.332e+04
MPI Message Lengths:	3.142e+07	3.179e+07	3.243e+07	3.280e+07
MPI Reductions:	1.262e+03	1.083e+03	9.393e+02	8.192e+02

从程序运行性能的结果比较中可以看出，当程序由单进程运行变为双进程运行时，运行时间大大缩短了。随着进程数的增加，进程间通信量也急剧增加，浮点运算次数减小，每秒浮点运算次数是先增大后减小。这就导致了开始几个进程中（进程数从 1 到 3）随着进程数增加，由于每秒浮点运算次数在增加，计算时间缩短得比较明显；从进程数 4 开始，随着进程数增加，由于每秒浮点运算数减小，计算时间缩短的较为平缓直至计算时间反而有所增加。因此，对于此并行求解并非进程数越大越好，应根据实际情况选取适合的进程数，以达到最小的计算时间。从图 5-17 可以明显看出，进程数为 7 时，所需时间最少。

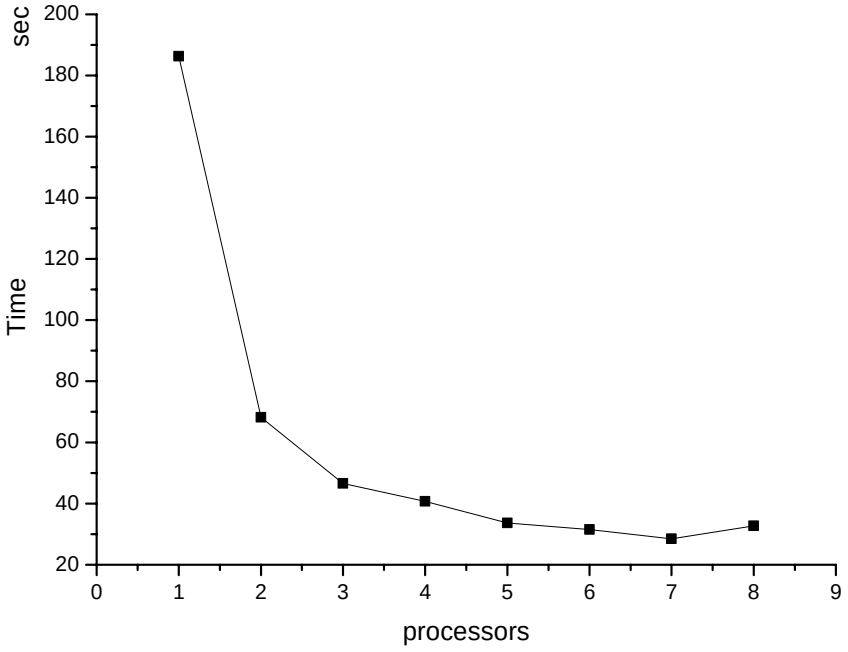

图 5-17 不同进程数下的计算时间的比较

5.2 单机翼（NACA0012）绕流

5.2.1 无攻角来流下数值模拟结果

单机翼绕流的原理（包括模型、边界条件等）在上文中已详细给出，在此不再鳌述。

本文计算单机翼（翼型NACA0012）绕流时使用的计算网格共有9354个结点，18508个三角形单元，在机翼周围的网格较密。

通过计算所得的势函数分布如图5-18所示，流函数分布如图5-19所示，压力分布如图5-20所示，速度矢量分布如图5-21所示。从图5-18和5-19中可以明显看出整个流场中的势函数和流函数的分布情况，势函数从入流面的φ=-2.5变化至出流面的φ=2.5，流函数从下壁面的ψ=-2.5变化至上壁面的ψ=2.5；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到机翼绕流影响的区域压力与速度分别为0和1，没有变化；而在机翼周围的压力和速度变化就非常明显。下面就着重研究机翼周围结点的压力、速度值以及沿y轴的速度变化。

为了验证结果的准确性，将所得的并行计算结果与解析解进行比较。

首先对机翼周围点的压力值进行比较。从图 5-22 中的比较结果可以看出，数值计算解与经验值吻合较好。接下来对机翼周围点的速度值进行比较。从图 5-24 中的比较结果可以看出，数值计算解与经验值吻合较好。最后比较沿 y 轴方向的速度值。本文选取了从 y=0.06 开始 y 坐标成等比数列（公比为 1.1）的一系列点，比较结果如图 5-23 所示。从图中可以看出，速度值的数值解与经验值基本吻合。从图中可以看出，速度值沿着远离机翼的方向逐渐减小，在 y=1.5 附近速度接近 1，说明从此处开始流场内远离机翼的区域受圆柱绕流的影响很小，可以说不受其影响。

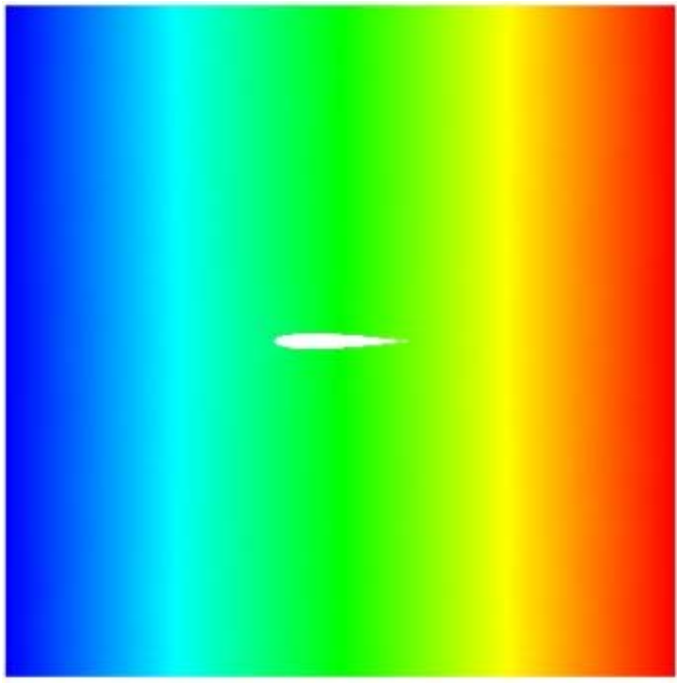

图 5-18 势函数分布图

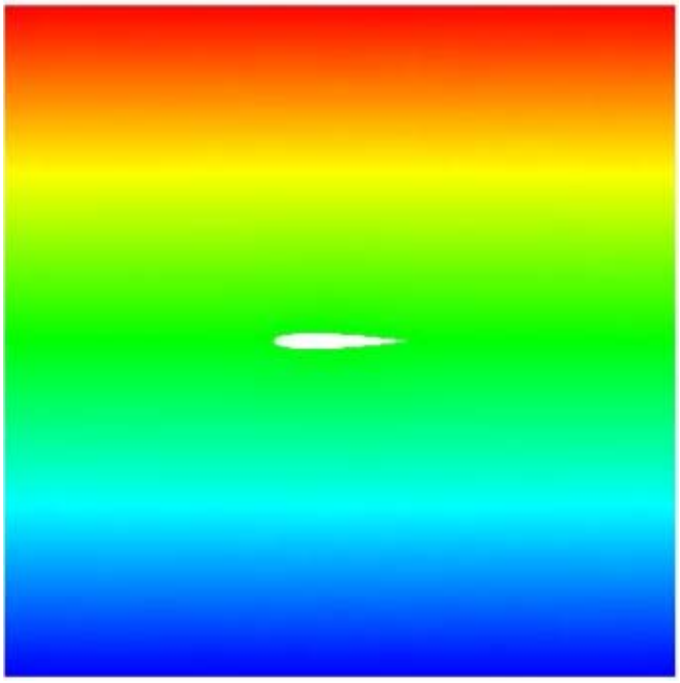

图 5-19 流函数分布图

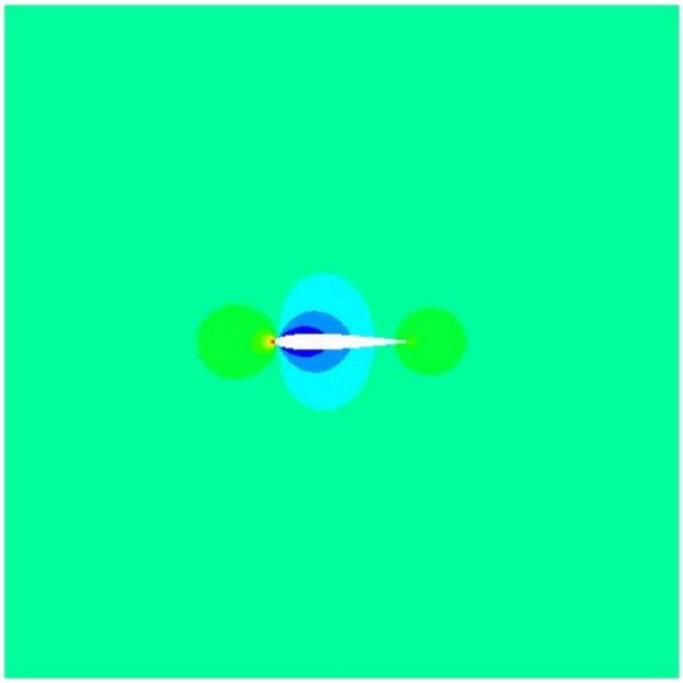

图 5-20 压力分布图

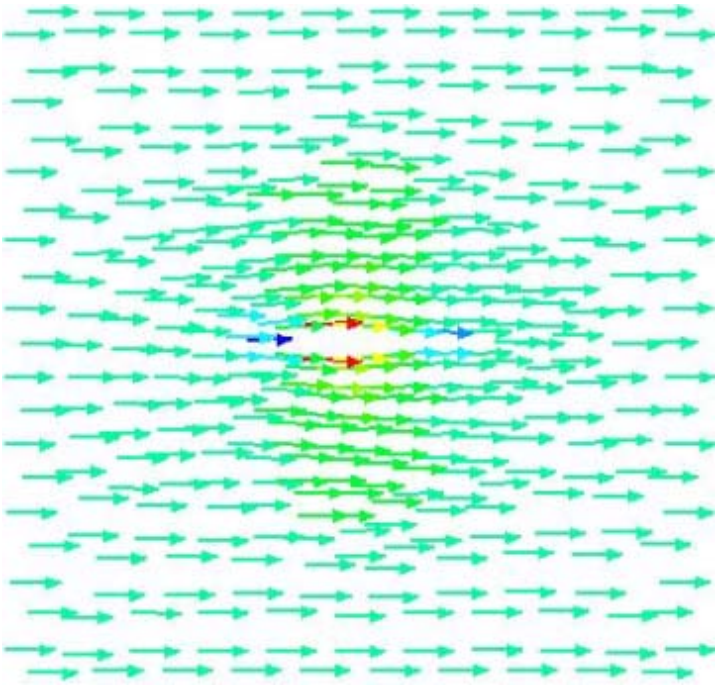

图 5-21 速度矢量分布图

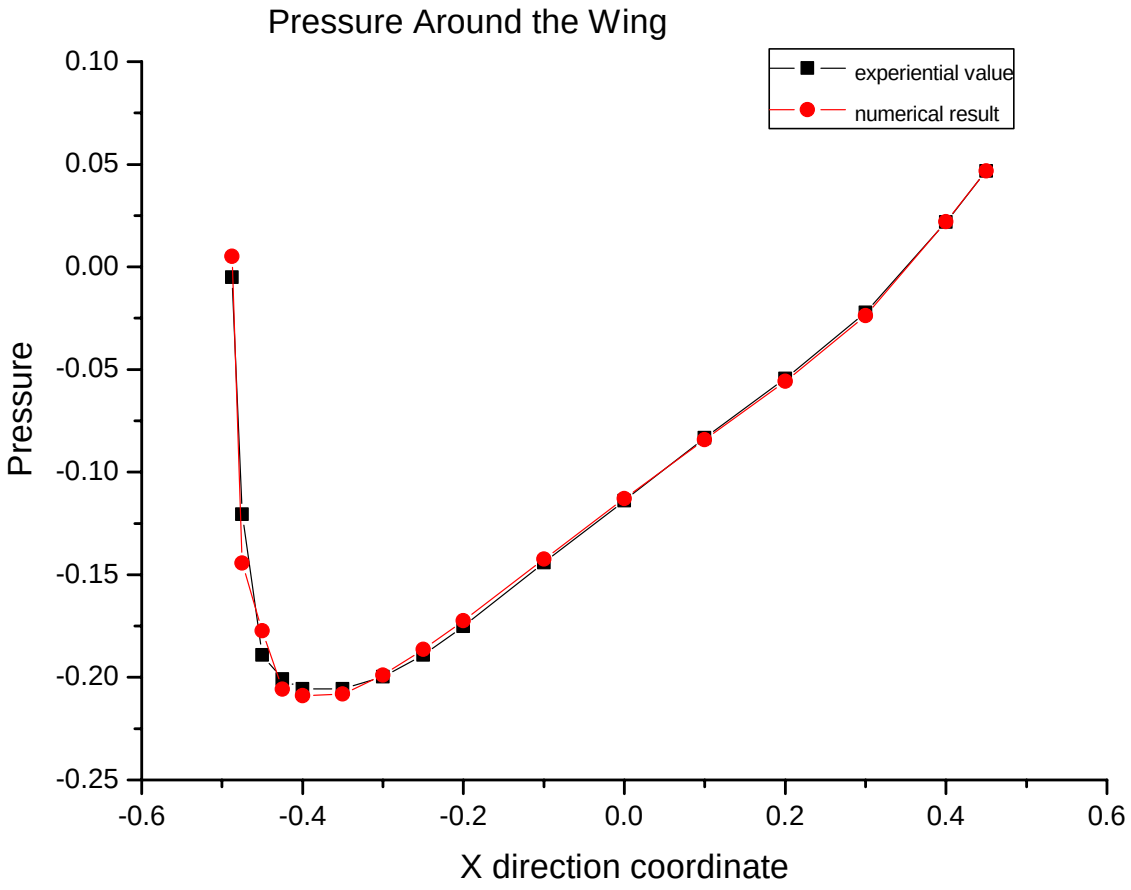

图 5-22 计算压力值与经验值的比较

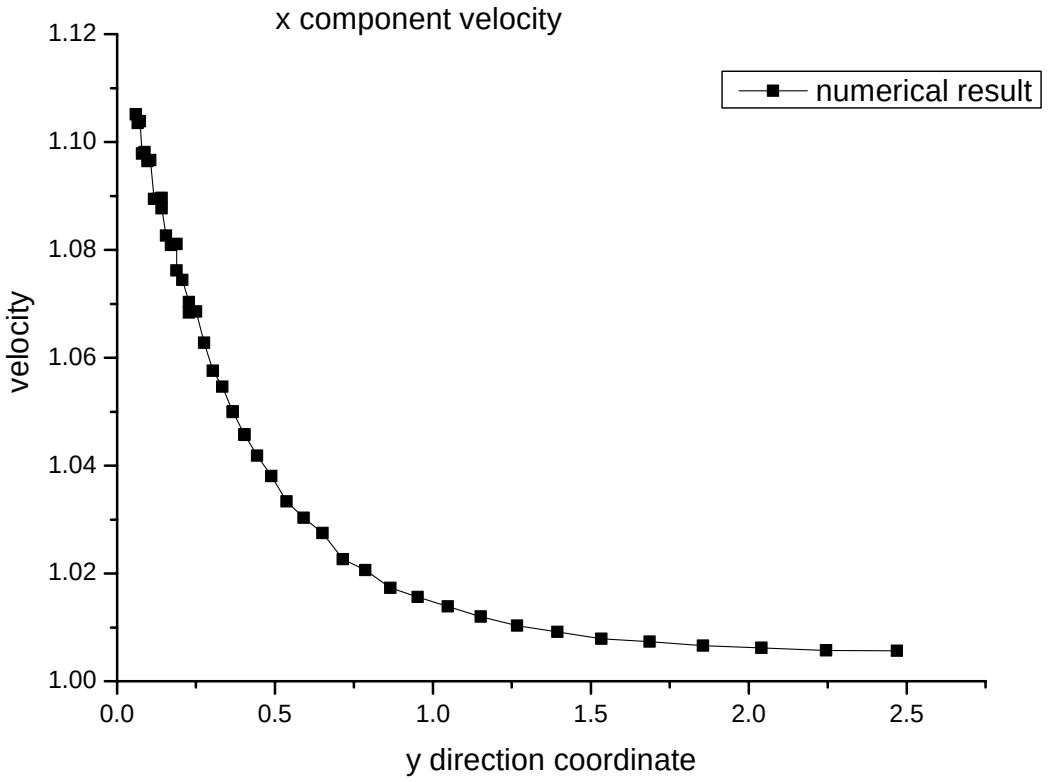

图 5-23 计算速度值

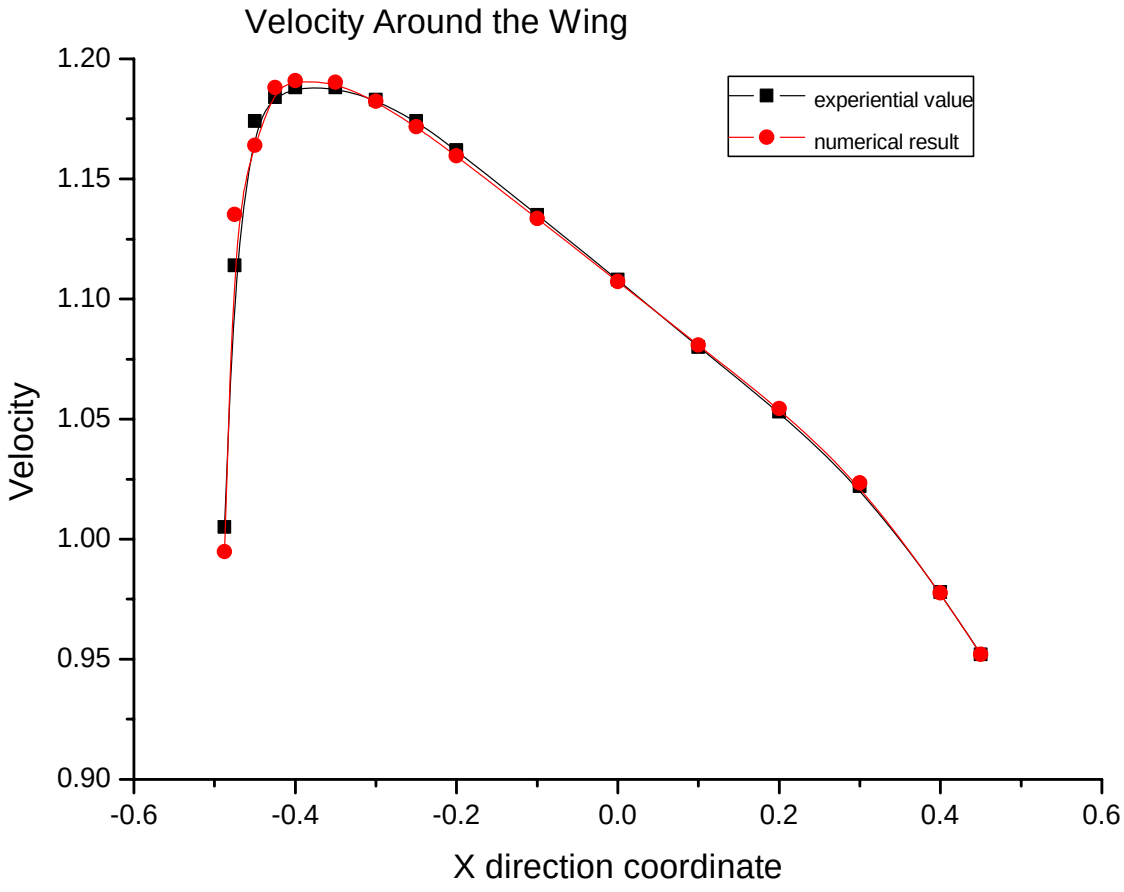

图 5-24 计算速度值与经验值的比较

5.2.2 有攻角来流下数值模拟结果

通过计算所得的势函数分布如图 5-25 所示，流函数分布如图 5-26 所示，压力分布如图 5-27 所示，速度矢量分布如图 5-28 所示。从图 5-25 和 5-26 中可以明显看出整个流场中的势函数和流函数的分布情况，它与无攻角来流区别在于等势线和流线不再竖直和水平而变为斜向；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到机翼绕流影响的区域压力与速度分别为 0 和 1，没有变化；而在机翼周围的压力和速度变化就非常明显。

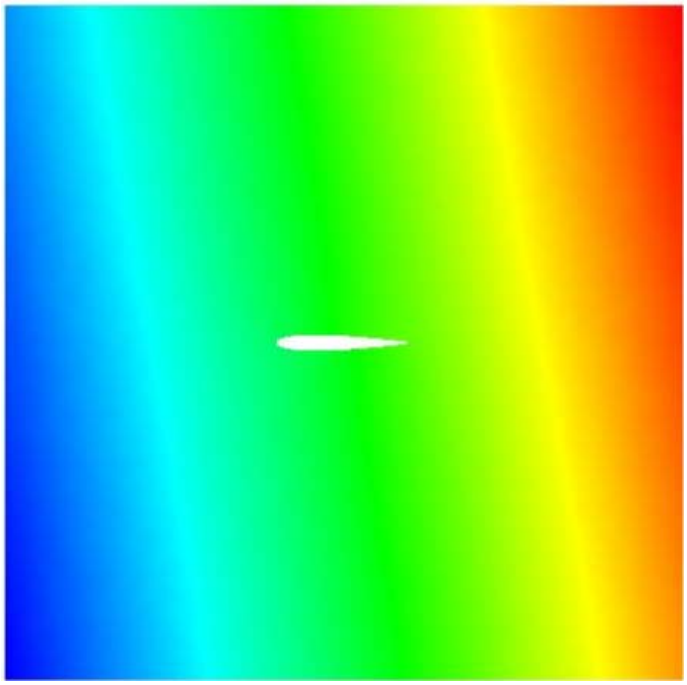

图 5-25 势函数分布图

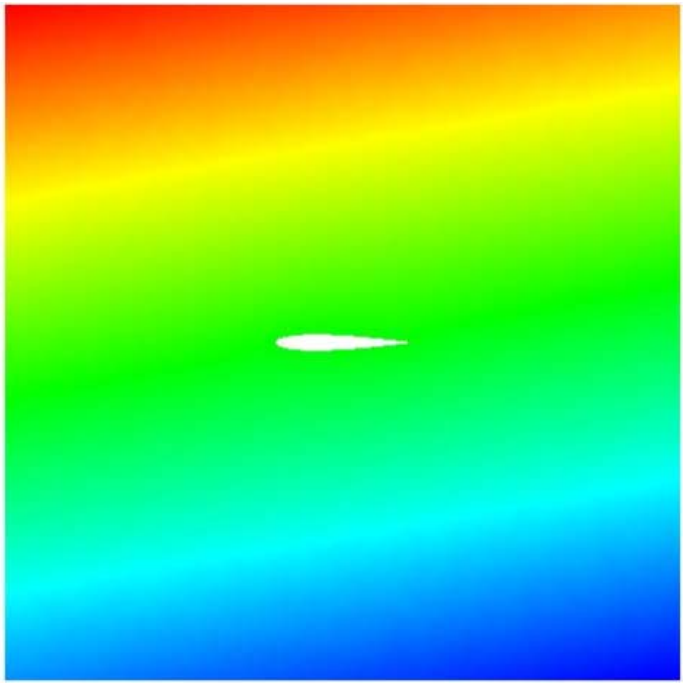

图 5-26 流函数分布图

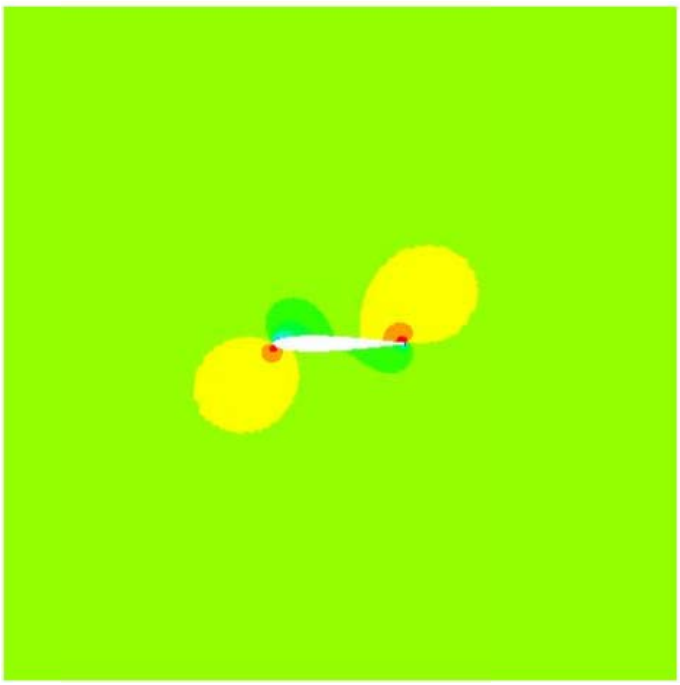

图 5-27 压力分布图

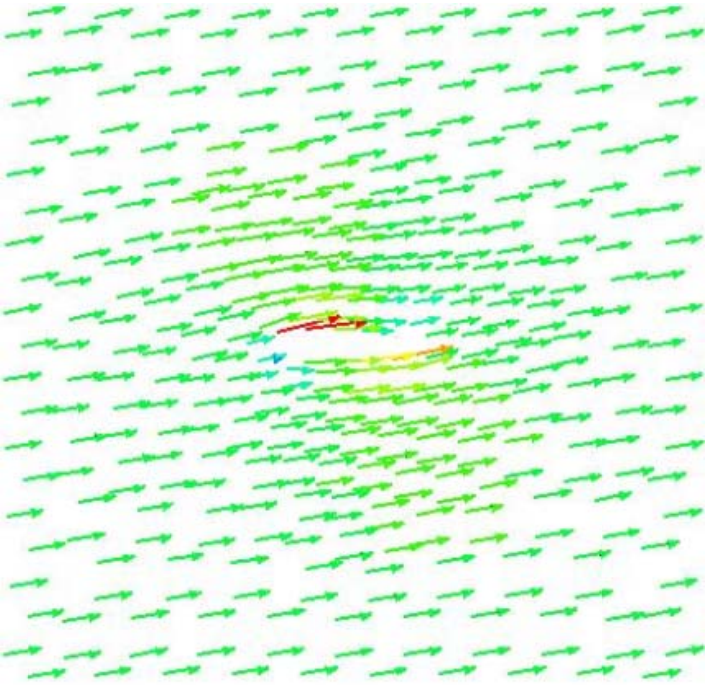

图 5-28 速度矢量分布图

下面就着重研究在不同攻角均匀来流下机翼周围结点的压力值以及沿 y 轴的速度变化，本文分别选取了无攻角、5^o 攻角以及 10^o 攻角这三种情况进行比较。

首先对在不同攻角均匀来流下机翼周围点的压力值进行比较，机翼上半部分压力比较结果如图 5-29，机翼下半部分压力比较结果如图 5-30。

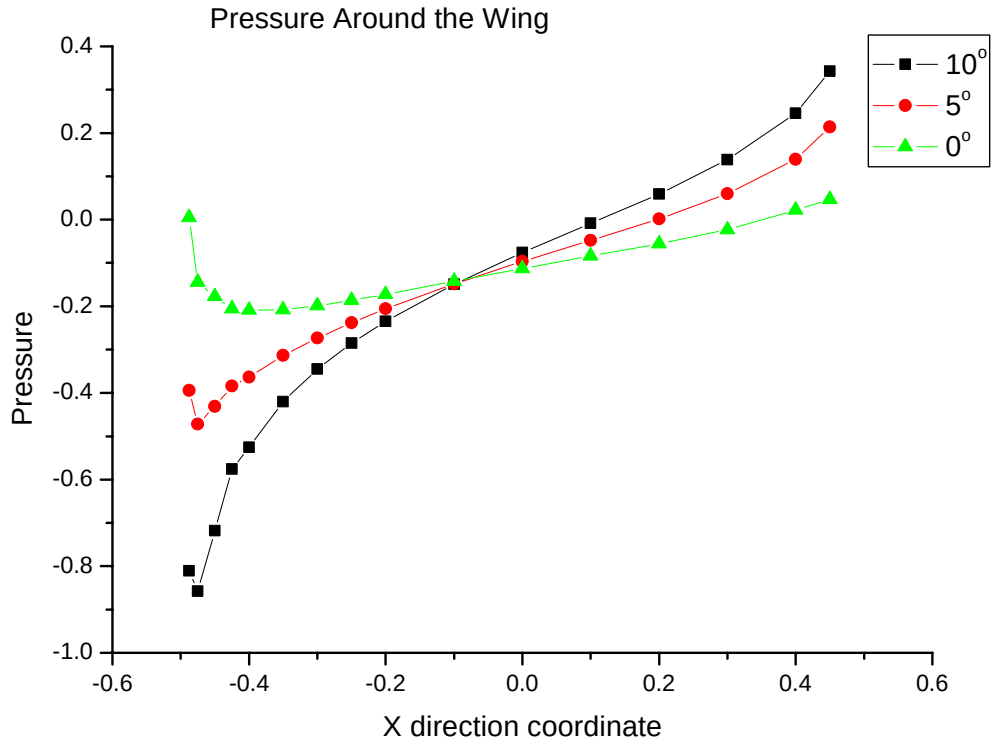

图 5-29 不同攻角来流下计算压力值比较

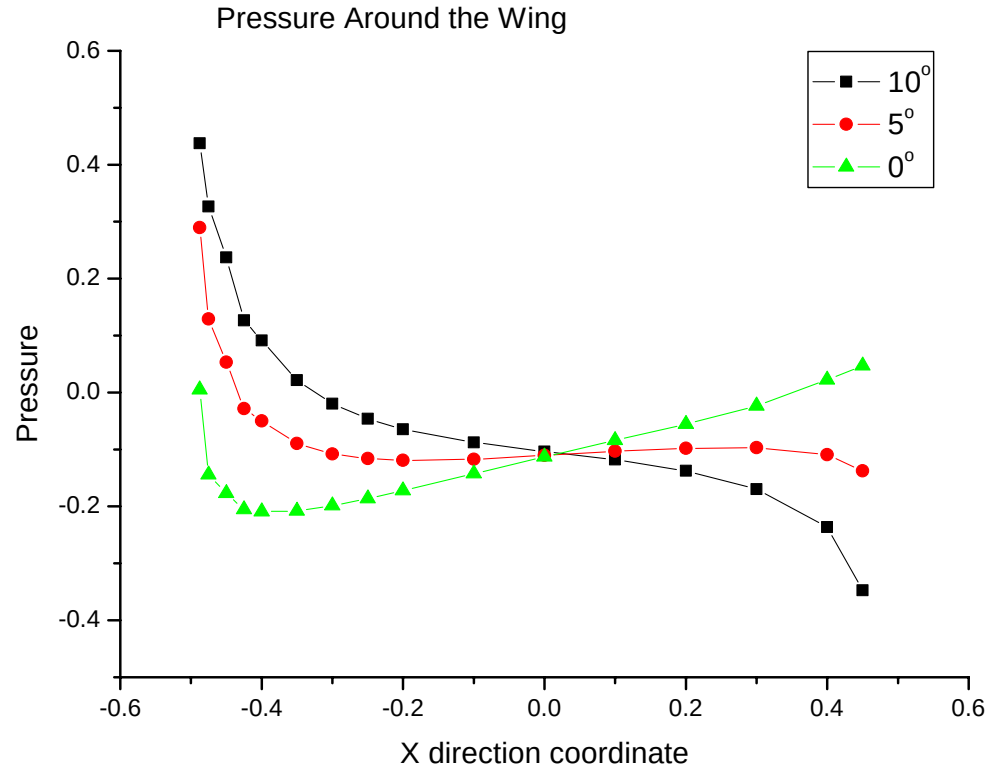

图 5-30 不同攻角来流下计算压力值比较

接着比较沿 y 轴方向的速度值。本文分别选取了从 y=0.06002 及 y=-0.06002 开始 y 坐标成等比数列（公比为 1.1）的一系列点，比较结果如图 5-31 及图 5-32 所示。从图中可以看出，不同攻角下速度变化趋势相同，速度值都是沿着远离机翼的方向逐渐减小。

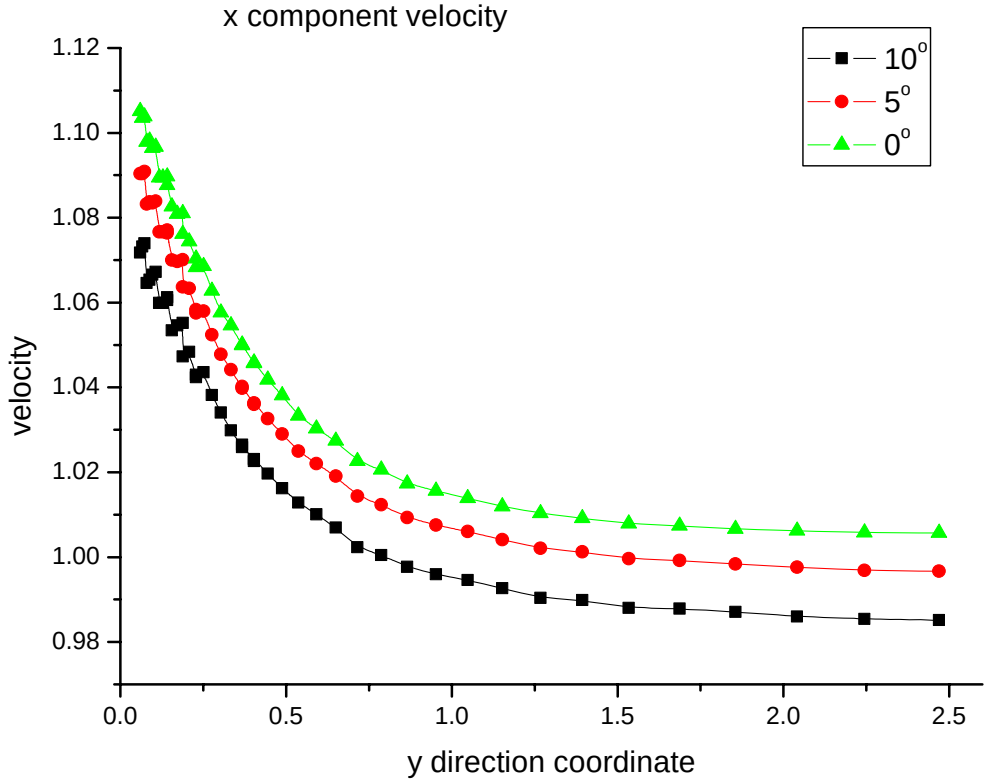

图 5-31 不同攻角来流下计算速度值比较

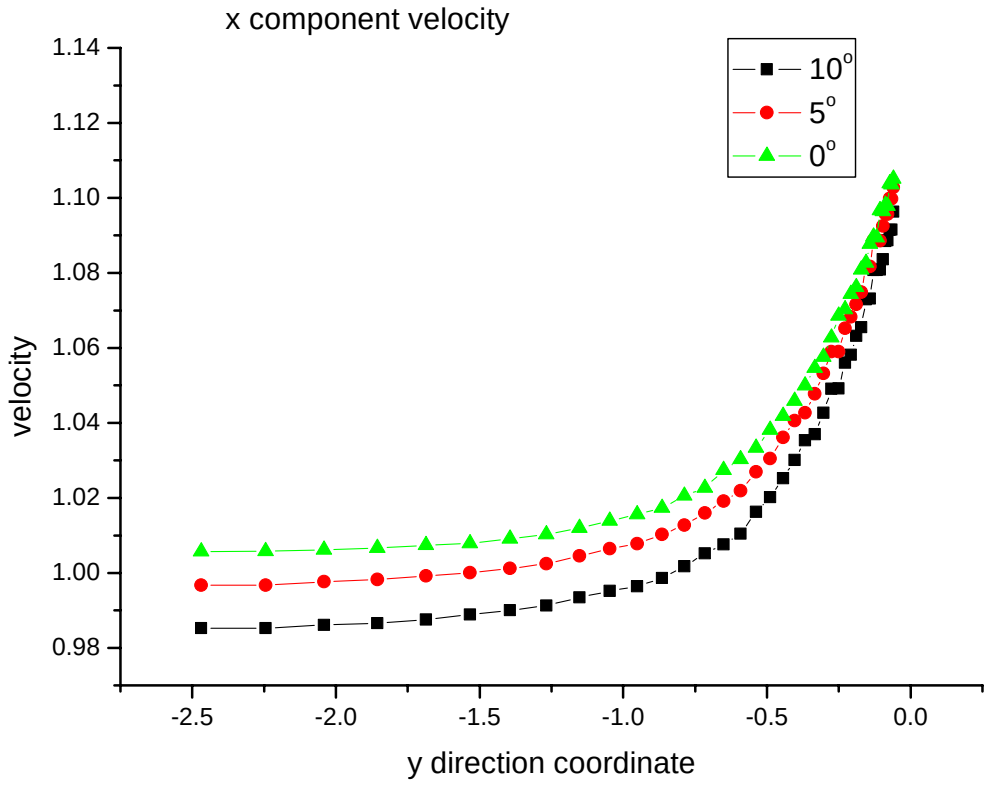

图 5-32 不同攻角来流下计算速度值比较

5.2.3 并行效率验证

本文通过在 PC-Cluster 上实现并行计算来求解翼型 NACA0012 单机翼绕流问题，下面就分别测试在不同进程数下求解的性能，以期得到最理想的并行效率。

表 5-2 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	2.732e+01	1.252e+01	8.617e+00	6.223e+00
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	5.096e+08	2.550e+08	1.700e+08	1.276e+08
Flops/sec:	1.865e+07	2.529e+07	2.767e+07	3.587e+07
MPI Message:	0.000e+00	7.015e+02	1.402e+03	2.104e+03
MPI Message Lengths:	0.000e+00	3.067e+06	5.097e+06	5.273e+06
MPI Reductions:	1.391e+03	6.990e+02	4.660e+02	3.495e+02
	5 procs	6 procs	7 procs	8 procs
Time (sec):	6.254e+00	7.267e+00	6.430e+00	5.751e+00
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	1.020e+08	8.502e+07	7.292e+07	6.382e+07
Flops/sec:	2.718e+07	1.795e+07	1.872e+07	2.074e+07
MPI Message:	2.108e+03	2.116e+03	2.814e+03	2.820e+03
MPI Message Lengths:	5.093e+06	5.111e+06	5.088e+06	5.120e+06
MPI Reductions:	2.796e+02	2.330e+02	1.997e+02	1.748e+02

从程序运行性能的结果比较中可以看出，当程序由单进程运行变为双进程运行时，运行时间缩短了一半。随着进程数的增加，进程间通信量也急剧增加，浮点运算次数减小，每秒浮点运算次数是先增大后减小又增加。这就导致了开始几个进程中（进程数从 1 到 4）随着进程数增加，由于每秒浮点运算次数在增加，计算时间缩短得比较明显；从进程数 5 开始，随着进程数增加，由于每秒浮点运算次数减小，进程数为 6 时计算时间反而增加，但是随着进程数的进一步增加，每秒浮点运算次数又逐渐增加，计算时间缩短最终在进程数为 8 时达到最小值。对于此并行求解并非进程数越大越好，应根据实际情况选取适合的进程数，以达到最小的计算时间。从图 5-33 可以明显看出，进程数为 8 时，所需时间最少。

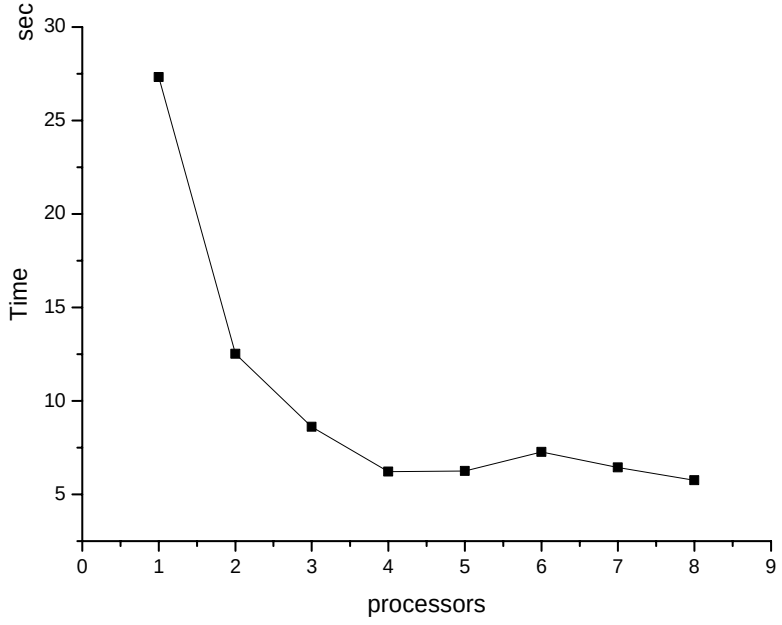

图 5-33 不同进程数下的计算时间的比较

5.3 双机翼绕流

5.3.1 数值模拟结果

双机翼绕流的物理模型类似于单机翼绕流（图 5-34），本文研究对称的并列双机翼（翼型 NACA0010-66）。其中，正方形区域的边长 H=5,来流速度 V_0=1.0，两机翼中心相距 d=0.2，坐标原点为两机翼的对称中心。

双机翼绕流问题的原理已在4.1节中详细介绍过了，需要指出的是，正方形区域的上壁面Γ_1、下壁面Γ_2及入流面Γ_3均为自然边界，出流面Γ_4为本质边界。即：

$$\begin{cases} \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_1\Gamma_2} = 0 \\ \left.\dfrac{\partial\varphi}{\partial n}\right|_{\Gamma_3} = -1 \\ \varphi|_{\Gamma_4} = 2.5 \end{cases} \tag{5-2}$$

本文计算双机翼绕流时使用的计算网格共有14802个结点，29286个三角形单元，在机翼周围的网格较密。

通过计算所得的势函数分布如图5-35所示，流函数分布如图5-36所示，压力分布如图5-37所示，速度矢量分布如图5-38所示。从图5-35和5-36中可以明显看出整个流场中的势函数和

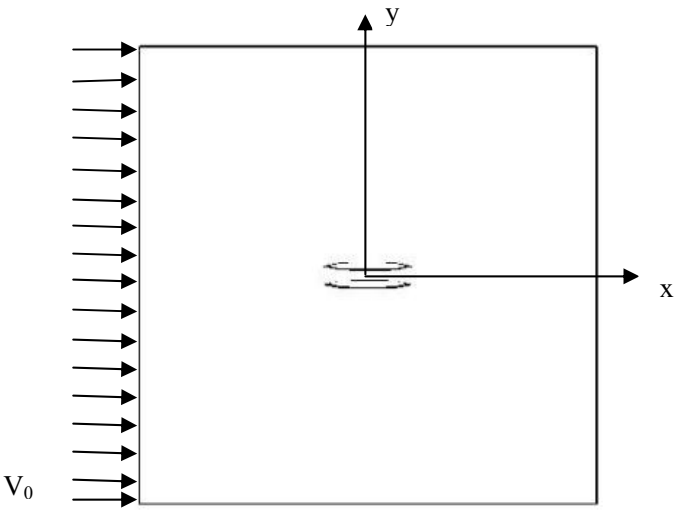

图 5-34 无限流场双机翼绕流（均匀流）简化模型

流函数的分布情况，势函数从入流面的φ=-2.5变化至出流面的φ=2.5，流函数从下壁面的ψ=-2.5变化至上壁面的ψ=2.5；而从压力分布图和速度矢量分布图中可以看出，在流场大部分未受到双机翼绕流影响的区域压力与速度分别为0和1，没有变化；而在两个机翼周围的压力和速度变化就非常明显。下面就着重研究机翼周围结点的压力值以及沿y轴的速度变化。

首先，通过并行计算求出机翼周围结点压力值，靠近原点机翼部分压力结果如图5-39，远离原点机翼部分压力结果如图5-40；接下来计算求出沿y轴方向的速度值，本文选取了从y=0.15开始纵坐标成等比数列（公比为1.1）的一系列点，结果如图5-41所示。最后研究两机翼之间沿y轴方向的速度分布，选取从坐标原点开始，纵坐标成等差数列（公差为0.005）的一系列点，结果如表5-3所示，从表中可明显看出，越靠近机翼速度值越大。

表 5-3 机翼间纵轴上点速度值

y	0	0.005	0.01	0.015	0.02	0.025
v	1.34136	1.34152	1.34153	1.34230	1.34232	1.34396
y	0.03	0.035	0.04	0.045	0.05	
v	1.34396	1.34394	1.34498	1.34498	1.34617	

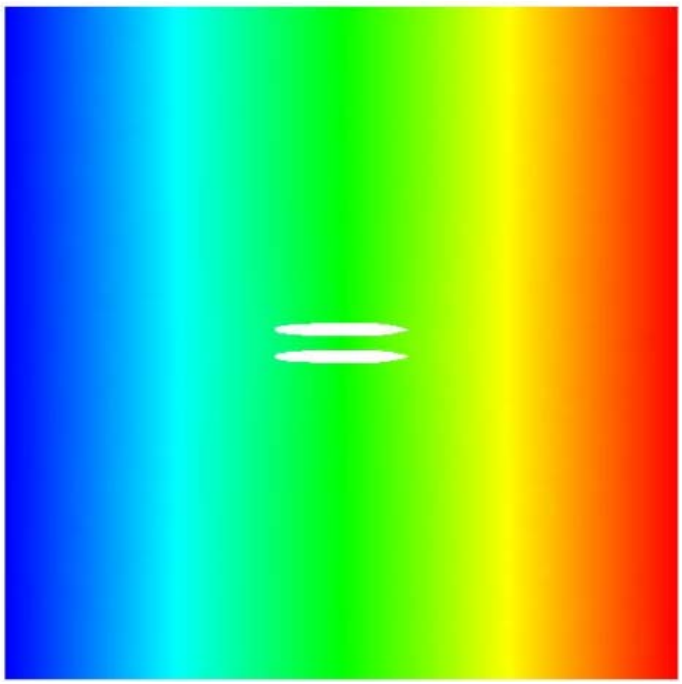

图 5-35 势函数分布图

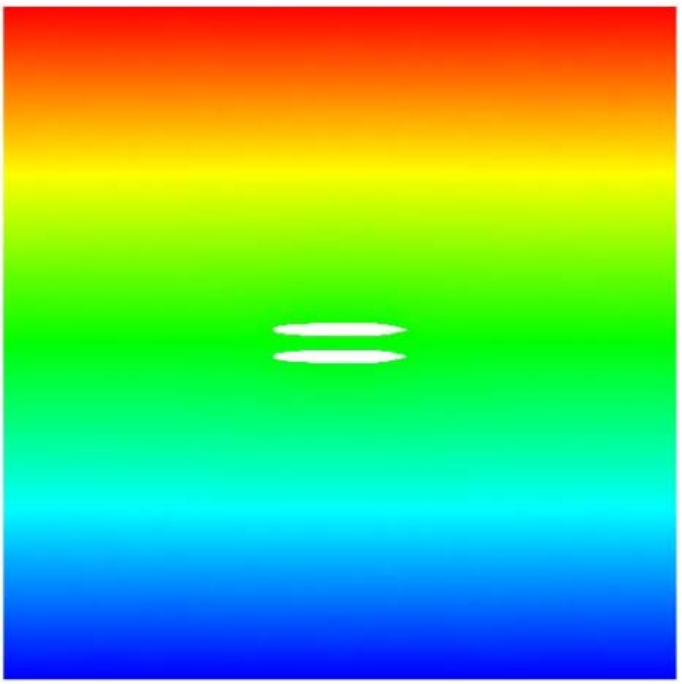

图 5-36 流函数分布图

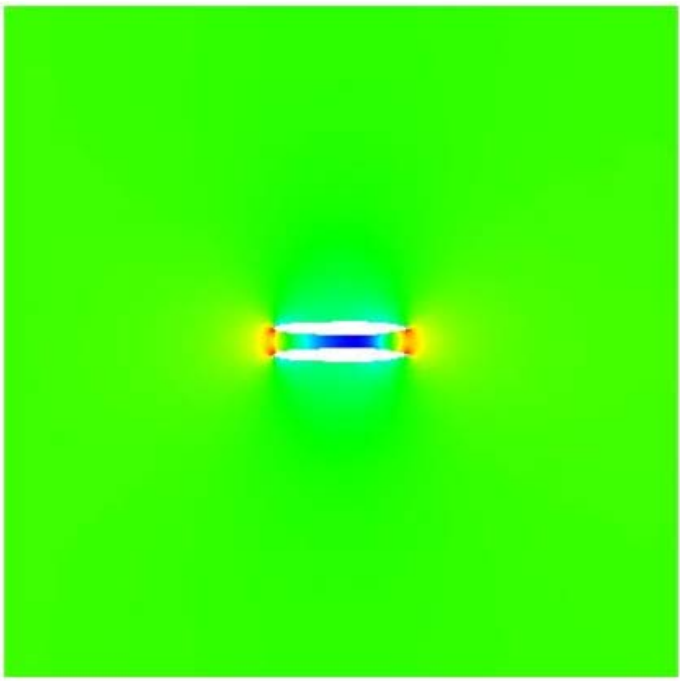

图 5-37 压力分布图

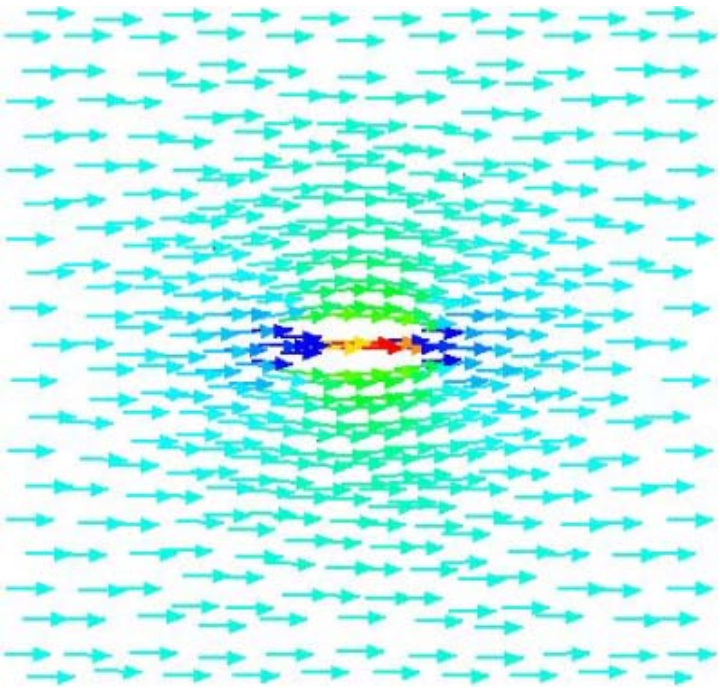

图 5-38 速度矢量分布图

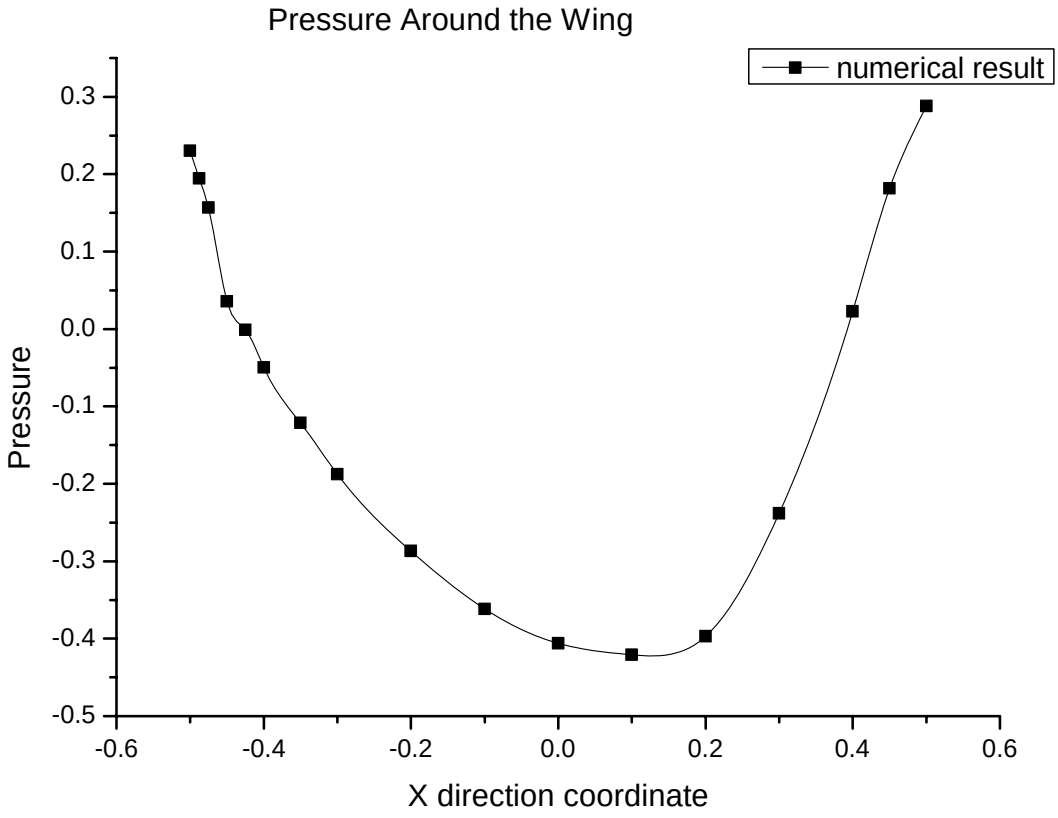

图 5-39 计算压力值

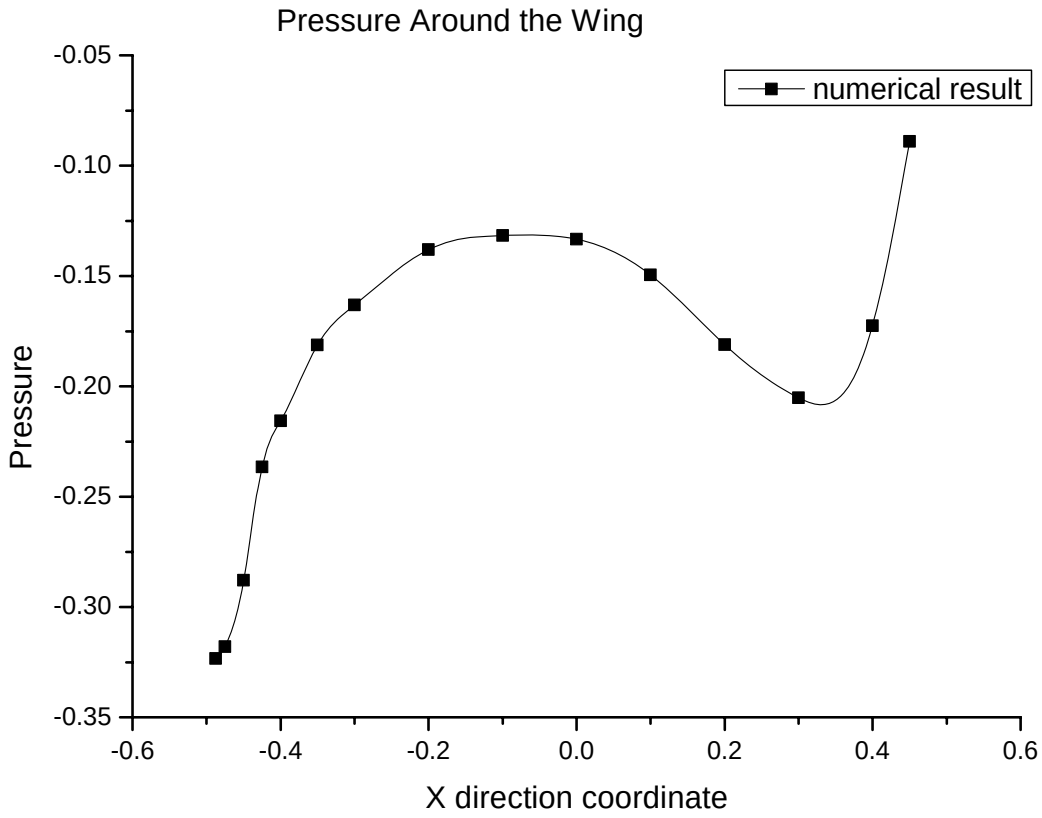

图 5-40 计算压力值

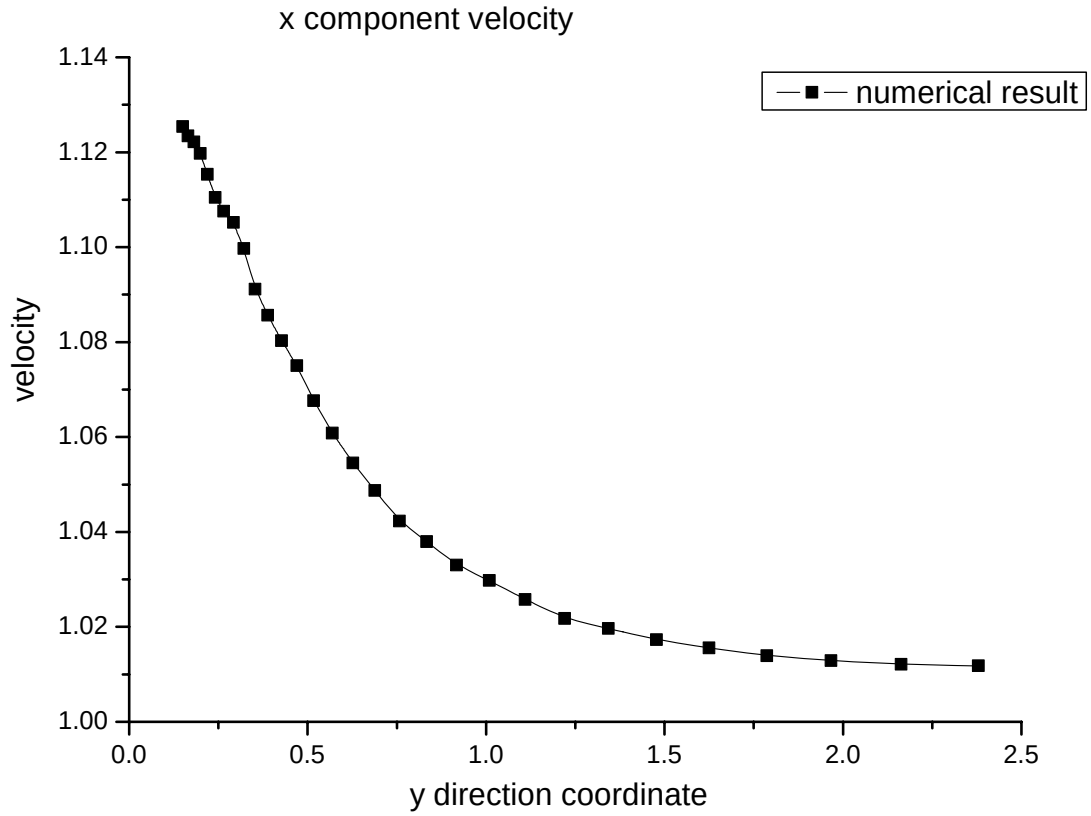

图 5-41 计算速度值

5.3.2 并行效率验证

本文通过在 PC-Cluster 上实现并行计算来求解双机翼（翼型 NACA0010-66）绕流问题，下面就分别测试在不同进程数下求解的性能，以期得到最理想的并行效率。

表 5-4 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	1.059e+02	3.820e+01	2.673e+01	2.610e+01
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	3.555e+09	1.784e+09	1.187e+09	8.891e+08
Flops/sec:	3.358e+07	5.508e+07	5.665e+07	4.402e+07
MPI Message:	0.000e+00	3.063e+03	6.112e+03	6.106e+03
MPI Message Lengths:	0.000e+00	1.523e+07	2.807e+07	3.047e+07
MPI Reductions:	6.015e+03	3.020e+03	2.009e+03	1.503e+03
	5 procs	6 procs	7 procs	8 procs
Time (sec):	1.963e+01	2.191e+01	2.274e+01	2.341e+01
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	6.758e+08	6.024e+08	5.074e+08	4.414e+08
Flops/sec:	4.879e+07	3.759e+07	2.992e+07	2.523e+07
MPI Message:	8.699e+03	6.226e+03	9.141e+03	1.213e+04
MPI Message Lengths:	2.885e+07	3.035e+07	3.052e+07	3.098e+07
MPI Reductions:	1.144e+03	1.019e+03	8.576e+02	7.471e+02

从程序运行性能的结果比较中可以看出，当程序由单进程运行变为双进程运行时，运行时间大大缩短了。随着进程数的增加，进程间通信量也急剧增加，而浮点运算次数减小，每

秒浮点运算次数是先增大后减小。这就导致了开始几个进程中（进程数从 1 到 3）随着进程数增加，由于每秒浮点运算次数在增加，计算时间缩短得比较明显；从进程数 4 开始，随着进程数增加，由于每秒浮点运算数减小，计算时间缩短的较为平缓直至计算时间反而有所增加。因此，对于此并行求解并非进程数越大越好，应根据实际情况选取适合的进程数，以达到最小的计算时间。从图 5-42 可以明显看出，进程数为 5 时，所需时间最少。

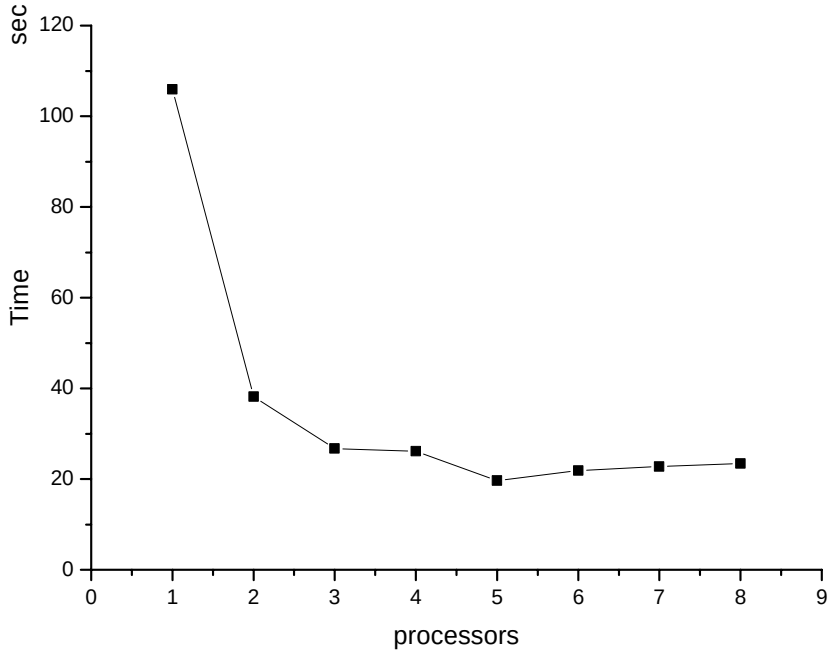

图 5-42 不同进程数下的计算时间的比较

5.4 本章小结

本章分别通过对单机翼（翼型 NACA0010-66 和翼型 NACA0012）及双机翼（翼型 NACA0010-66）的绕流流场的数值模拟，分别给出了绕流速度势分布、流线分布、速度分布以及压力分布图，并分析了进程数与并行计算效率的关系，同时给出了单机翼绕流流场速度分布和物体表面压力分布的数值结果与经验值比较，两者吻合很好。证明了本课题采用的有限元法与 PETSc 相结合能高效求解机翼（包括单双机翼）绕流问题。

第六章 圆球势流绕流的有限元并行求解

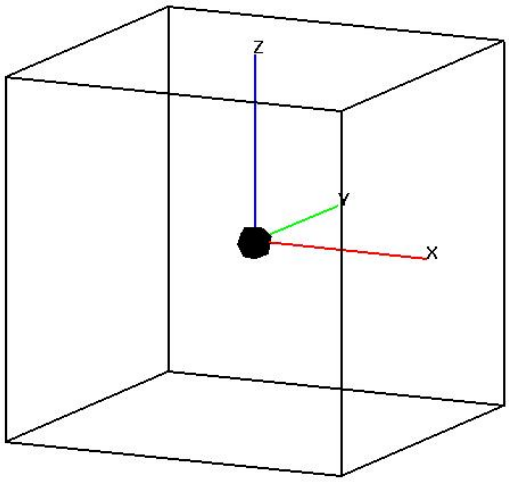

图 6-1 三维圆球绕流求解模型

6.1 原理简介

本文近似模拟无限域内圆球势流绕流，选取适当大小的计算流域，其边界条件可与无穷远处边界条件相同而对圆球绕流结果影响较小。无限流场（可视为边长H=20的正方体区域）中圆球绕流问题的物理模型如图6-1所示，其中，正方体的中心为直角坐标系o_xyz的原点，且流域在坐标系中的取值范围是：x∈[-10,10], y∈[-10,10],z∈[-10,10] 。圆球的球心为坐标系的原点，圆球半径R=1，来流速度u=1.0，v=0,w=0。

6.1.1 控制方程与边界条件

对于三维理想不可压流体的流动，最重要的简化是引进速度势函数，在给定的边界条件下求解Laplace方程。具体控制方程如下：

$$\begin{cases} \dfrac{\partial^2\varphi}{\partial x^2}+\dfrac{\partial^2\varphi}{\partial y^2}+\dfrac{\partial^2\varphi}{\partial z^2}=0 & (x,y,z)\in\Omega \\ \varphi\big|_{S_1}=\overline{\varphi} & \\ \left.\dfrac{\partial\varphi}{\partial n}\right|_{S_2}=g & \end{cases} \tag{6-1}$$

其中Ω为计算流体区域，S_1为本质边界条件，S_2为自然边界条件。

根据方程余量与权函数正交化原理建立起来的控制方程（6-1）的强解积分表达式：

$$\int_\Omega\left(\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}+\frac{\partial^2\varphi}{\partial z^2}\right)\delta\varphi d\Omega=0 \tag{6-2}$$

对上式进行分部积分，应用Green公式，并带入本质边界条件和自然边界条件，即可得到弱解积分表达式：

$$\int_{\Omega}[\frac{\partial\varphi}{\partial x}\frac{\partial(\delta\varphi)}{\partial x}+\frac{\partial\varphi}{\partial y}\frac{\partial(\delta\varphi)}{\partial y}+\frac{\partial\varphi}{\partial z}\frac{\partial(\delta\varphi)}{\partial z}]d\Omega=\int_{S_2}g\delta\varphi dS \tag{6-3}$$

6.1.2 网格生成

本文使用Gambit软件生成四面体网格，需要注意的是，为保证生成的单元网格为四面体，必须先在各个边界面上生成三角形网格单元。为了保证计算结果有较高精度，圆球附近应设置较密的网格。

Gambit输出的网格文件有其特定的数据结构，其中主要的信息包括所有网格结点的坐标值，各个界面上和整个流域内部的单元交界面（face）所包含的结点，以及这些交界面（face）所属的单元。通过读取这些信息并进行重构，就可得到有限元数值模拟中所需要的数据，如每个单元中结点的编号，所有本质边界结点号，以及所有自然边界所包含的交界面（face）信息。

6.1.3 单元分析

单元分析是有限元分析中的关键一部分。根据单元类型的不同，选取插值函数的不同，所使用的单元分析方法也不同，对结果的精度也有较大的影响。本文采用四面体单元离散计算流域，选取四结点Lagrange线性插值基函数进行单元分析。下面就进行简单介绍。

线性插值基函数：

$$\Phi_i=a_i+b_ix+c_iy+d_iz \qquad (i=1,2,3,4) \tag{6-4}$$

根据插值基函数的定义：

$$\Phi_i(x_j^{(e)},y_j^{(e)},z_j^{(e)})=\delta_{ij}=\begin{cases}1 & (i=j)\\ 0 & (i\neq j)\end{cases} \qquad (i,j=1,2,3,4) \tag{6-5}$$

在四面体单元中可通过建立体积坐标的方式得到线性插值基函数，即令：

$$\Phi_i=\frac{V_i}{V} \qquad (i=1,2,3,4) \tag{6-6}$$

其中V是四面体单元的体积，V_i是这样定义的：若单元中任一点与各个顶点的连线所构成的平面，将单元体划分为四个小四面体，则其中与序号为i的顶点不相邻的那个小四面体的体积为V_i。

通过计算表达式（6-6）即可以得到线性插值基函数（6-4）中各个系数的值，即：

$$a_i=(-1)^{i+1}\begin{vmatrix}x_j & y_j & z_j\\ x_k & y_k & z_k\\ x_l & y_l & z_l\end{vmatrix}\frac{1}{6V} \qquad b_i=(-1)^{i}\begin{vmatrix}1 & y_j & z_j\\ 1 & y_k & z_k\\ 1 & y_l & z_l\end{vmatrix}\frac{1}{6V}$$

$$c_i=(-1)^{i}\begin{vmatrix}x_j & 1 & z_j\\ x_k & 1 & z_k\\ x_l & 1 & z_l\end{vmatrix}\frac{1}{6V} \qquad d_i=(-1)^{i}\begin{vmatrix}x_j & y_j & 1\\ x_k & y_k & 1\\ x_l & y_l & 1\end{vmatrix}\frac{1}{6V} \tag{6-7}$$

其中i, j, k, l按下标1，2，3，4循环取值，四面体的体积为：

$$V=\frac{1}{6}\begin{vmatrix}1 & x_1 & y_1 & z_1\\ 1 & x_2 & y_2 & z_2\\ 1 & x_3 & y_3 & z_3\\ 1 & x_4 & y_4 & z_4\end{vmatrix} \tag{6-8}$$

显然Φ_i中只有三个独立变量，即：

$$\Phi_1+\Phi_2+\Phi_3+\Phi_4=1 \tag{6-9}$$

单元内任意一点的速度势可表示为：$\varphi=\Sigma\varphi_j\Phi_j(x,y)$，令 $\delta\varphi=\Phi_i(x,y)$，则在任一单元内弱解积分表达式可转化为：

$$[\int_{\Omega^{(e)}}(b_ib_j+c_ic_j+d_id_j)dxdydz]\varphi_j=\int_{S_2^{(e)}}g\Phi_i dS \qquad (i,j=1,2,3,4) \tag{6-10}$$

即生成的单元线性方程组为：

$$A_{ij}\varphi_j=f \qquad (i,j=1,2,3,4) \tag{6-11}$$

其中$A_{ij}=(b_ib_j+c_ic_j+d_id_j)|V|$，V为四面体单元的体积，其值由式（6-8）给出。对于单元面积分 $f_i=\int_{S_2^{(e)}}g\Phi_i dS$，若假定 $S_2^{(e)}=\Delta 234$，即由单元结点序号234构成的三角形，且边函数g在Δ234内认为是线性函数，即 $g=g_2\Phi_2+g_3\Phi_3+g_4\Phi_4$，则由积分公式

$$\int_{S_2^{(e)}}\Phi_2^m\Phi_3^n\Phi_4^p dS=2A\frac{m!n!p!}{(m+n+p+2)!} \tag{6-12}$$

可求得单元线性方程组的右端项向量：

$$\begin{cases}f_1=0\\ f_2=\dfrac{A}{12}(2g_2+g_3+g_4)\\ f_3=\dfrac{A}{12}(g_2+2g_3+g_4)\\ f_4=\dfrac{A}{12}(g_2+g_3+2g_4)\end{cases} \tag{6-13}$$

其中A为单元自然边界Δ234的面积，其值可由海伦公式求得：设三角形三边长分别为a, b, c, 令t=(a+b+c)/2, 则 $A=\sqrt{t(t-a)(t-b)(t-c)}$ 。

6.1.4 总体合成

总体合成就是将单元有限元方程中的系数矩阵和右端项各个系数，按照单元结点序号与总体结点序号对应关系，累加到相应的总体有限元方程系数矩阵和右端项的元素中去。假定单元中的结点i，j对应的总体结点的序号是m，n，则进行下面所表示的累加：

$$\begin{cases}A_{mn}=A_{mn}+A_{ij}\\ f_m=f_m+f_i\end{cases} \tag{6-14}$$

6.1.5 边界条件处理

自然边界条件已经在弱解积分表达式（6-3）中得到满足，本质边界条件要通过总体有限元方程的修正得到满足。修正的方法有两种，即“消行修正法”和“对角线项扩大修正法”。

这两种方法都是强制本质边界上的函数值满足本质边界条件，本文在这采用对角线项扩大修正法对本质边界条件进行处理。

这个方法的要点是：假定本质边界S_1上的结点序号为n_1，n_2 ，…，n_r，给定的函数值分别为$\overline{\varphi_1}$，$\overline{\varphi_2}$，…，$\overline{\varphi_r}$，修正的方法是将系数矩阵$\{A_{mn}\}$中相应本质边界结点序号的对角线元素A_{pp}(p=n_1，n_2 ，…，n_r)，乘以一个大数T，其他所有元素不变动，同时把$\{f_m\}$中的相应本质边界结点序号的元素f_p改写为$TA_{pp}\overline{\varphi_p}$ (p=n_1，n_2 ，…，n_r)。

6.1.6 线性代数方程组的并行求解

线性代数方程组的并行求解是采用PETSc的KSP（Krylov Subspace）方法。由有限元方法离散得到系数矩阵为大型稀疏对称矩阵，求解这类线性系统的ksp方法有GMRES（广义最小残差法）、BCGS（双共轭梯度法）、TFQMR（拟最小残差法）、LSQR(最小二乘QR分解)以及SYMMLQ（对称LQ分解）等方法。此外在求解过程中还可以使用Jacobi、ILU以及SOR等预条件子，目前采用较多的适用于并行计算的预条件子为Jacobi。

6.1.7 速度和压力的求解

当各个结点的速度势求出后，就可以根据单元结点势函数值求解各个单元的速度。在三结点三角形单元内，任一点的速度势可表示为：

$$\varphi=\sum_{j=1}^{4}\varphi_j\Phi_j(x,y,z)=\sum_{j=1}^{3}\varphi_j(a_j+b_jx+c_jy+d_jz) \tag{6-15}$$

由于速度势插值基函数为线性函数，因此各个单元内的速度值为常数，速度分量的表达式为：

$$\begin{cases} u=\sum_{j=1}^{4}\frac{\partial\varphi}{\partial x}=\sum_{j=1}^{4}\varphi_j b_j \\ v=\sum_{j=1}^{4}\frac{\partial\varphi}{\partial y}=\sum_{j=1}^{4}\varphi_j c_j \\ w=\sum_{j=1}^{4}\frac{\partial\varphi}{\partial z}=\sum_{j=1}^{4}\varphi_j d_j \end{cases} \tag{6-16}$$

其中b_j，c_j，d_j的值由方程（6-4）给出。

当各个单元的速度值求出后，各个结点上的速度值可取包含该结点的所有单元的速度值的平均值。当速度值获得后，可通过Bernoulli方程求解压力。考虑到重力场的定常运动的Bernoulli方程为：

$$\frac{v^2}{2}+\frac{P}{\rho}=C \tag{6-17}$$

其中C为常数。

6.1.8 结果后处理

结果的后处理使用GMV（General Mesh Viewer）图形处理软件。GMV由美国Los Alomos国家实验室开发，本文采用GMV软件作势函数分布图、压力分布图以及速度大小分布图。

6.2 数值模拟结果

圆球绕流问题的原理已在上节中详细介绍过了。需要指出的是，出流边界（即为x=10的正方形边界面）为本质边界，其值为$\overline{\varphi}=10u=10$，其余各边界面均为自然边界，入流边界（即为x=-10的正方形边界面）的自然边界值为g=-u=-1，壁面边界（包括y=-10, y=10, z=-10, z=10的正方形边界面及圆球表面）的自然边界值为g=0。

本文使用的计算网格共有 27147 个结点，136415 个四面体单元，在圆球周围的网格较密。

通过计算所得流域内的速度势分布如图6-2和6-3所示，速度大小分布如图6-4所示，压力分布如图6-5所示。

为了验证结果的准确性，将所得的并行计算结果与解析解进行比较。

由于无限域中均匀流下绕流的静止圆球具有轴对称性，故其速度势在极坐标中可表示为：

$$\varphi = -U_0(r + \frac{a^3}{2r^2})\cos\theta \tag{6-18}$$

相应的速度分量为：

$$\begin{cases} v_r = \dfrac{\partial\varphi}{\partial r} = -U_0(1 - \dfrac{a^3}{r^3})\cos\theta \\ v_\theta = \dfrac{1}{r}\dfrac{\partial\varphi}{\partial\theta} = U_0(1 + \dfrac{a^3}{2r^3})\sin\theta \\ v_\lambda = 0 \end{cases} \tag{6-19}$$

利用不考虑重力场的定常运动的Bernoulli方程可求得压力的表达式为：

$$p = p_0 + \frac{1}{2}\rho U_0^2 - \frac{1}{2}\rho(v_r^2 + v_\theta^2 + v_\lambda^2) \tag{6-20}$$

其中，来流压力 p_0=0，流体介质密度 ρ =1，来流 U_0=1。

首先对圆球周围的结点压力值进行比较。圆球表面压力在 θ∈[0，90°]范围内变化较大，精确计算较为困难。在 xoz 平面内，将圆球表面压力的数值解与解析解的比较如图 6-6。从图 6-6 中的比较结果可以看出，数值计算解与近似解析解吻合较好。

接下来比较沿 z 轴方向的速度值。本文选取了从 z=1 开始的一系列点，比较结果如图 6-7 所示。从图中可以看出，速度值的数值解与近似解析解基本吻合。

6.3 并行效率验证

本文通过在 PC-Cluster 上实现并行计算来求解圆球绕流问题，下面就分别测试在不同进程数下求解的性能，以期得到最理想的并行效率。

表 6-1 不同进程数下计算性能比较

	1 proc	2 procs	3 procs	4 procs
Time (sec):	6.101e+02	3.305e+02	2.538e+02	2.081e+02
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01
Flops:	1.097e+09	5.668e+08	3.774e+08	2.865e+08
Flops/sec:	1.797e+06	1.731e+06	1.505e+06	1.396e+06
MPI Message:	0.000e+00	4.490e+02	8.980e+02	1.347e+03
MPI Message Lengths:	0.000e+00	3.395e+07	6.373e+07	5.563e+07
MPI Reductions:	8.870e+02	4.470e+02	2.980e+02	2.235e+02
	5 procs	6 procs	7 procs	8 procs
Time (sec):	1.790e+02	1.613e+02	1.535e+02	1.353e+02
Objects:	5.200e+01	5.200e+01	5.200e+01	5.200e+01

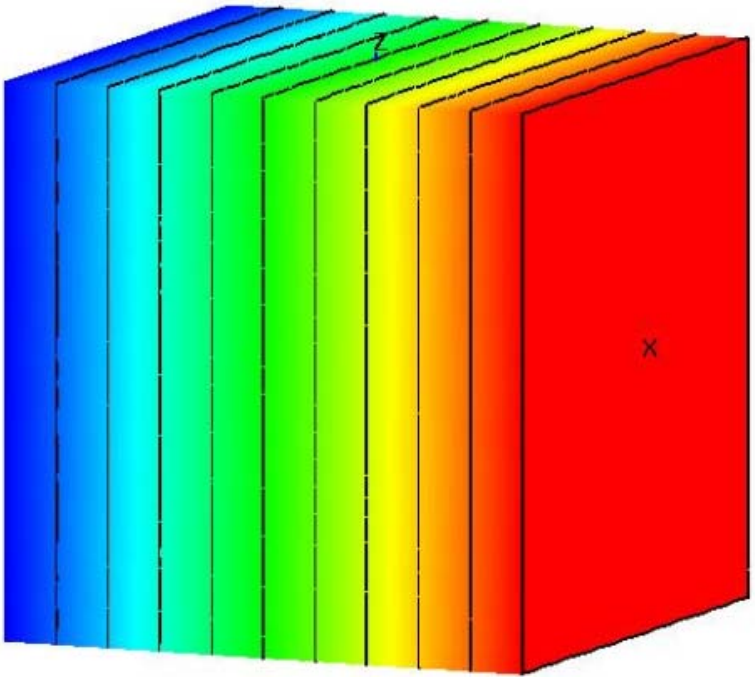

图 6-2 速度势函数及其等值线

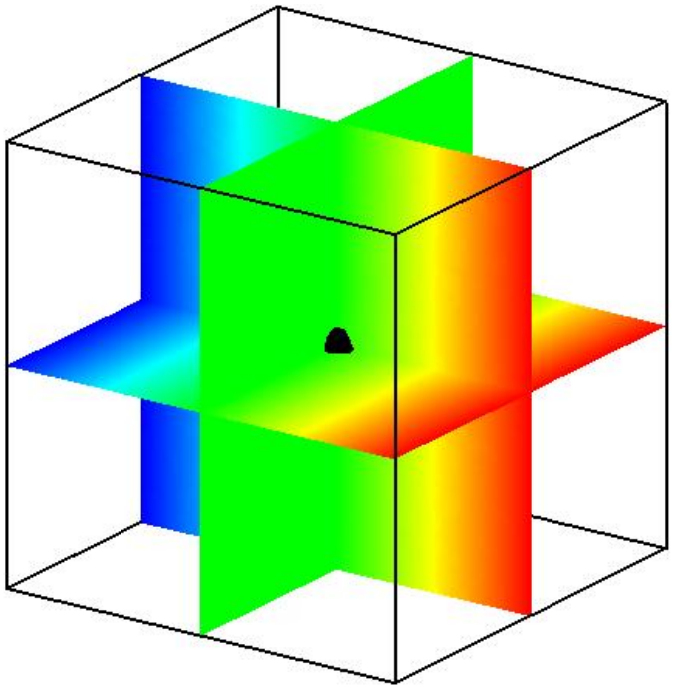

图 6-3 速度势分布剖面

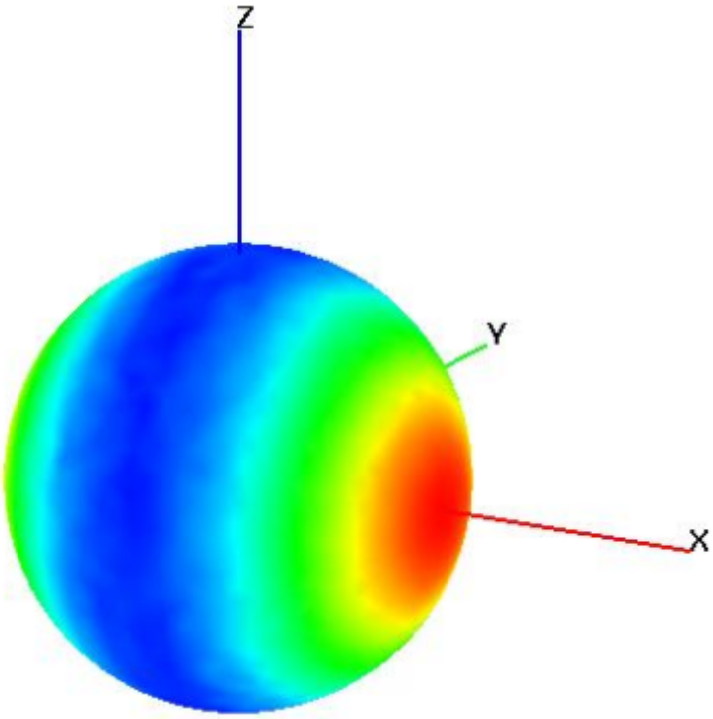

图 6-4 速度大小分布剖面

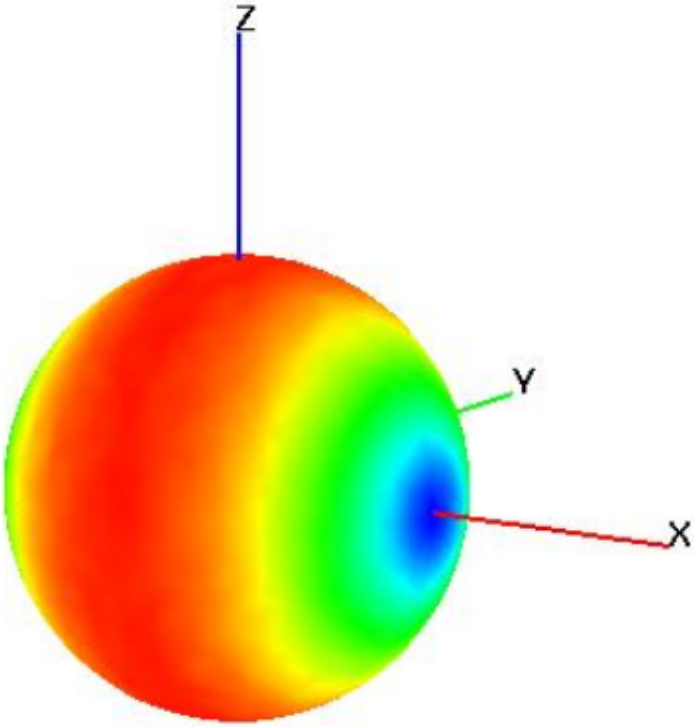

图 6-5 压力分布剖面

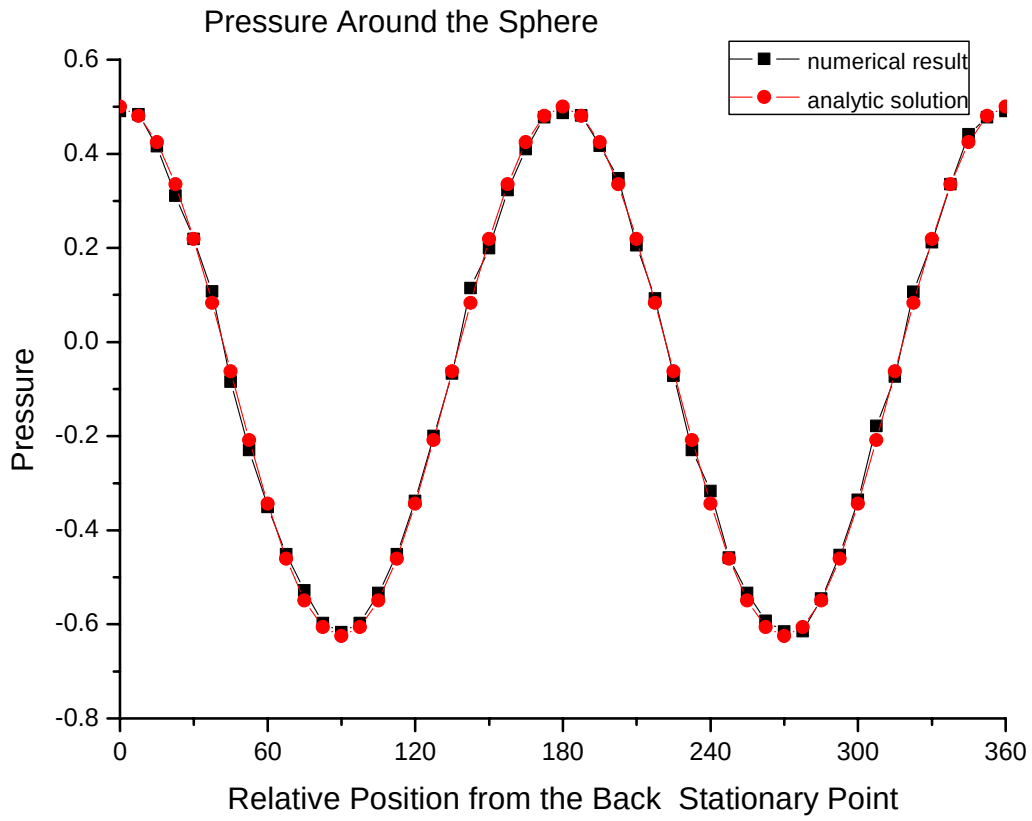

图 6-6 计算压力值与近似解析解的比较

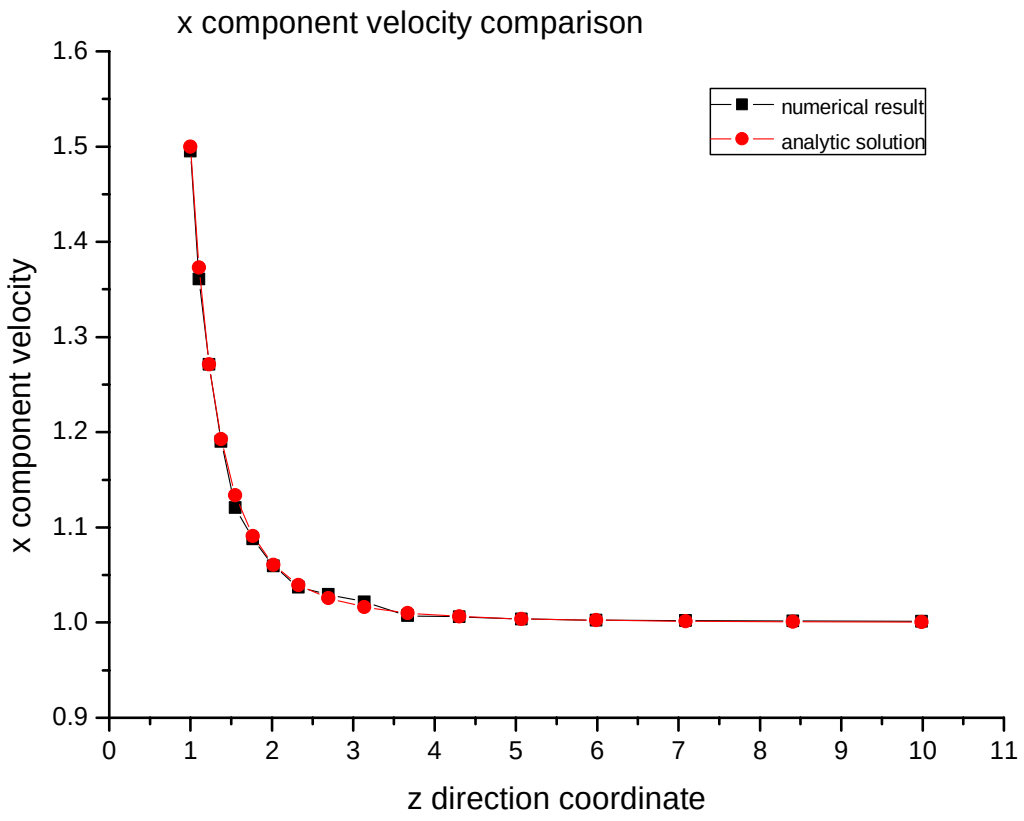

图 6-7 计算速度值与近似解析解的比较

续表 6-1

	5 procs	6 procs	7 procs	8 procs
Flops:	2.286e+08	1.912e+08	1.647e+08	1.434e+08
Flops/sec:	1.299e+06	1.208e+06	1.095e+06	1.084e+06
MPI Message:	1.794e+03	2.243e+03	2.692e+03	3.140e+03
MPI Message Lengths:	5.841e+07	5.041e+07	5.178e+07	4.637e+07
MPI Reductions:	1.788e+02	1.490e+02	1.277e+02	1.118e+02

从程序运行性能的结果比较中可以看出，当程序由单进程运行变为双进程运行时，运行时间缩短了接近一半。随着进程数的增加，浮点运算次数减小，每秒浮点运算次数也在减小，但是由于浮点运算次数减小得更快，对于此并行求解进程数越大越好。从图 6-8 可以明显看出，进程数为 8 时，所需时间最少。

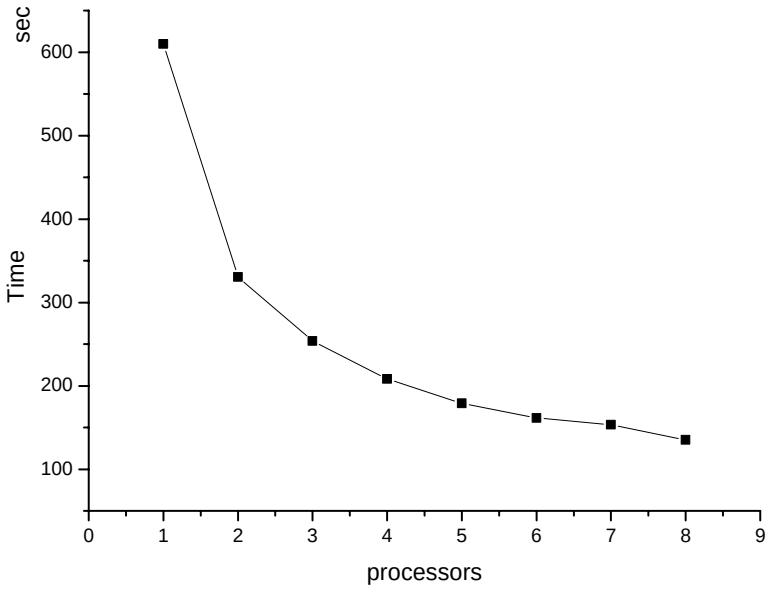

图 6-8 不同进程数下的计算时间的比较

6.4 本章小结

本章首先介绍了圆球绕流数值模拟的原理及方法，分别通过对圆球绕流流场的数值模拟，给出了绕流速度势分布、速度分布以及压力分布图，并分析了进程数与并行计算效率的关系，同时给出了圆球绕流流场速度分布和物体表面压力分布的数值结果与解析解比较，两者吻合很好。证明了本课题采用的有限元法与 PETSc 相结合能高效求解圆球绕流问题。

第七章 结论

绕流问题广泛存在于各工程领域，如海洋工程、航空航天、城市规划等众多领域，研究此类问题有着极强的实际意义。因此，近年来它一直是中外学者的研究热点。本文把有限元法与并行计算工具箱 PETSc (Portable，Extensible Toolkit for Scientific Computation) 结合，对理想不可压流体的物体势流绕流问题进行并行数值计算。分别研究了二维无限域内圆柱绕流、NACA0010-66 机翼绕流、NACA0012 机翼绕流、双圆柱绕流、双机翼绕流以及三维无限域内圆球绕流问题，通过数值模拟给出了整个无限流场内的速度势分布、流线分布、速度分布以及压力分布；将部分的绕流流场速度及物体表面压力的数值模拟的计算结果与解析解（或经验值）进行比较；此外还分别对上述各种情况下的进程数与并行效率关系进行了分析。通过对上述问题的研究分析得出以下结论：

（1）通过单圆柱、NACA0010-66 机翼绕流、NACA0012 以及圆球绕流的数值模拟计算结果与解析解或经验值的比较，不难看出两者之间的误差很小，这就说明了本次数值模拟方法正确可行，为研究双圆柱、双机翼问题奠定了基础，同样通过此数值模拟方法得出的计算结果十分可信。

（2）Gambit 网格生成的疏密程度对计算结果存在很大的影响，笔者在计算圆球绕流时开始在圆球周围分布网格不够密集，导致数值模拟计算结果与解析解之间存在较大误差，随后立即加密网格，此时计算得出的结果明显与解析解之间十分吻合。同样单元形态的选择对于计算结果的精度也有一定的影响，但是由于时间的关系，在研究二维流场中势流物体绕流问题时，本文仅选用了三角形网格。

（3）在研究各种绕流问题时，对进程数与并行计算效率的关系进行了分析。从不同进程下的比较结果，可以看出并非选取的进程数越多，所需的计算时间就越短，并行效率就越高。这是因为随着进程数增加，每秒运算量是增大的，但是通信总量也随之增加。因此，这就要求在进行并行计算时，为提高并行效率应该选取适合的进程数。

（4）从数值计算结果看出，有限元法与 PETSc 结合可以高效率实现理想不可压流体物体势流绕流问题的并行数值求解。

本文把有限元法与 PETSc 结合，对理想不可压流体的二维和三维物体势流绕流问题，进行有限元离散和并行计算。计算区域用有限单元离散，单元形式为三角形或四边形单元(二维)，以及四面体单元(三维)，单元形函数为 Lagrange 插值函数。有限元积分离散后的线性代数方程组，用 PETSc 库函数进行并行求解。

本文着重研究了均匀流下的物体势流绕流问题，由于时间的关系，笔者未对剪切流下的物体势流绕流问题进行细致研究，最初对边界条件进行了简单的处理，出流与入流边界条件为：$\frac{\partial\varphi}{\partial n}=\bar{g}$，上下壁面边界条件为：$\frac{\partial\varphi}{\partial n}=0$，通过数值模拟得出了一系列的结果，但是后来发现在剪切流下上下壁面的边界条件应是给定速度大小的，并非上述所给出的法向速度为 0，在此边界条件不是严格意义的剪切流。在实际问题中，物体受到的来流情况及其结构形状错综复杂，对物体势流绕流问题的研究还有很多问题有待深入研究探讨。而本文为今后开展复杂粘性流体流动问题的并行计算奠定了基础。

参考文献

[1] 吴建军. 绕流问题数值模拟的研究综述[J/OL]. http://www.paper.edu.cn.
[2] 张雁. 圆柱和机翼串列组合结构绕流的数值模拟[D]. 哈尔滨:哈尔滨工业大学机械电子工程专业,2004:1-5.
[3] 戴绍仕. 孤立圆柱及串列双圆柱水动力数值实验研究[D]. 哈尔滨:哈尔滨工程大学港口海岸及近海工程专业,2004:2-6.
[4] 吴建军. 机翼绕流的三维数值模拟[D]. 河海大学流体力学专业,2007:1-8.
[5] 谭维炎. 计算浅水动力学——有限体积法的运用[M]. 北京:清华大学出版社,1998.
[6] 陈景仁. 湍流模型及有限分析法[M]. 上海:上海交通大学出版社,1989.
[7] 唐建峰，段常贵. 特征线法在燃气管道动态模拟中的应用[J]. 油气运输,2001,20(8):12-17.
[8] 饶寿期. 有限元法和边界元法基础[M]. 北京:北京航空航天大学出版社,1990.
[9] 赵经文，王宏钰. 结构有限元分析[M]. 北京:科学出版社,2001.
[10] Peter Hunter,Andrew Pullan. FEM/BEM NOTES. Department of Engineering Science,The University of Auckland New Zealand:2003.2.28.
[11] 王嘉漠，沈毅. 并行计算方法[M]. 北京:国防工业出版社,1987.
[12] William Gropp,Rusty Lusk,Rob Ross,and Rajeev Thakur. Advanced MPI:I/O and One-sided Communication. The University of Chicago.
[13] Satish Balay,Kris Buschelman,Victor Eijkhout,William Gropp, etc. PETSC Users Manual[J/OL]. http://www.mcs.anl.gov/petsc.
[14] Matthew Knepley,Dmitry Karpeev. PETSc and Unstructured Finite Elements. Argonne National Laboratory.
[15] 王献孚，周树信，陈泽梁，杨驰，刘应中. 计算船舶流体力学[M]. 上海:上海交通大学出版社,1991.
[16] 章本照，印建安，张宏基. 流体力学数值方法[M]. 机械工业出版社,2003.
[17] 刘岳元，冯铁城，刘应中. 水动力学基础[M]. 上海:上海交通大学出版社,1989:144-150.
[18] Gene H Golub,Charles F Van Loan. 矩阵计算[M]. 科学出版社,2001.
[19] Mohamed HafeZ,Essam Wahba. Inviscid flows over a cylinder[D]. Department of Mechanical and Aeronautical Engineering, University of California,2003.

谢辞

经过几个月的努力和辛劳，终于完成了毕业设计，这篇论文为我四年的本科生活画上了圆满的句号。通过本课题的研究，我不仅增加了基础知识和专业知识，而且提高了独立分析问题和解决问题的能力，在学术上和做人方面也深受启迪和教益。

感谢导师万德成教授的指导。万老师为课题的研究方案提出了重要的指导性建议，适时了解研究工作的进度和方向，提出了很多宝贵的指导性建议，并细心地为我解答研究中遇到的问题。万老师谦和的品行和治学严谨的作风给予我很多做人的启示。在此我向他表示衷心的谢意。

感谢研究生学长王吉飞同学。感谢他悉心指导我的课题研究工作，并一直关注课题研究和论文撰写的进展，对论文提出了大量有价值的建议。在我研究工作遇到困难并向学长求教时，学长很耐心地与我讨论解决方案，有效地帮助我解决了一系列难题。在此对学长的帮助和支持表示衷心感谢。

感谢同一个课题组的王晴晴同学。虽然我们毕业设计是不同的方向，但在一些共同的问题上仍给予了我很多帮助，在此对她表示衷心感谢。

感谢我的父母、家人多年来对我精神上和生活上的关爱、支持和理解，使我顺利地完成本科阶段的学业。

最后再次向给予我教导、关怀和帮助的人致以深深的谢意！

译文及原文

摘要：

本手册介绍了在高性能计算机上如何使用 PETSc 数值求解偏微分方程及相关问题。PETSc 是可移植可扩展科学计算工具箱；它是一套数据结构和例程，为在并行（包含串行）的电脑上运行大规模应用代码提供了运算基础。PETSc 使用 MPI 消息传递通信标准。

PETSc 包括一套扩展的并行线性、非线性方程解法器和时间积分器，这些解法器和积分器可在 Fortran、C 和 C＋+语言编写的应用程序代码中使用。PETSc 提供了很多在并行应用代码中所需要的机制，例如并行矩阵和向量集合的规程。数据库分层次组成，这样使得用户能够抽象地运用其中的任一层次，这最适合解决一个特定问题。通过使用面向对象编程技术，PETSc 为用户提供了巨大的灵活性。

PETSc 是一套复杂的软件工具；这种工具对于一些用户来说，刚开始学习时要比学习一个简单的子程序库困难得多。特别是对于那些没有计算机学习背景和 C、C++语言编程经验的人，可能需要一个相当长的时间，才能充分掌握该软件的特点，使其得到高效的使用。然而，PETSc 的设计能力和它所包含的算法可有效执行许多应用代码，这比滚动运行简单得多。

• 对于许多简单的（或者甚至是较为复杂的）任务，一系列软件如 MATLAB 往往是最好的工具；PETSc 并非用来解决这类问题，这类问题用 MATLAB 代码就可以进行有效编译。

• 不要尝试使用 PETSc 在不同的顺序代码中提供并行的线性解法器。当然，所有先前顺序代码部分不必并行，但矩阵生成部分必须有合理的表现。不要指望能依次生成矩阵再使用 PETSc 并行求解线性系统。

因为 PETSc 是在继续发展之中，在使用情况和呼叫序列的例程会有小的变化。PETSc 具体信息详见网站 http://www.mcs.anl.gov/petsc。关于 PETSc 的一系列相关资料以及网站信息详见 http://www.unix.mcs.anl.gov/petsc/petsc-as/publications。

我们欢迎对这些网页的任何补充。

第一章　入门

可移植可扩展科学计算工具箱（PETSc）已成功地证明，现代编程范式的运用可以舒缓大型科学应用代码的发展压力，尤其是用 Fortran ， C 和 C＋+编写的代码。经过几年发展，该软件已演变成一个强大的工具集，用来在高性能计算机上数值求解偏微分方程及相关问题。

PETSc 由各种数据库构成（类似于 C ＋ +中的类），这将在用户手册的第二和第三部分详细讨论。每个数据库操纵特定的对象（例如，向量）以及对这些对象的具体操作。PETSc 中的对象和操作源于我们长久以来科学计算的经验。部分 PETSc 模块处理

- 索引，包括排列，索引到向量，重编等；
- 向量；
- 矩阵（通常是稀疏）；
- 分布式阵列（对规则的基于网格并行问题很有用）；
- Krylov 子空间方法；

- 预条件子：包括多重网格和稀疏直接解法器；
- 非线性解法器；
- 时间步进积分器：解决依赖时间的（非线性）偏微分方程。

每个模块由一个抽象的接口（简单来说是一套呼叫序列）和一个或多个算法组成，这些算法的实现是使用特定的数据结构。因此，PETSc 为各阶段解决偏微分方程提供有效的源代码，对每一类的问题提供了统一的方法。这种设计能更易进行比较和使用不同的算法（例如，尝试不同的 Krylov 子空间方法，预条件子，或拟牛顿迭代解法）。因此，PETSc 为科学建模，以、设计快速算法提供了良好的环境。

数据库能使算法和实现的定制和扩展更加容易。这种做法促进了代码的重用和灵活性，将并行问题从算法选择中分离开。PETSc 为大规模应用提供了基础。

考虑 PETSc 的不同组件之间的相互关系非常有用。图 1 是这些组件的分布图；图 2 更详细介绍其中的部分独立组件。这些图片揭示了数据库有层次的结构，这种结构使用户能够运用其中任一层，这最适合解决特定的问题。

1.1 导读

该手册分为三部分：

- 第一部分----PETSc 简介
- 第二部分----PETSc 编程
- 第三部分----其他信息

第一部分介绍了使用 PETSc 数据库的基本步骤，并用两个简单的例子说明如何用 PETSc 求解线性系统。该部分介绍了使用 PETSc 数据库的典型风格，使应用程序员能立刻使用该软件。第一部分还分别为对 PETSc 软件概览感兴趣的个人进行简介，但不包括数据库的详细使用。这个单独分布部分的读者应该注意所有内部参考文本在完整的用户手册中的位置。

第二部分详细地解释了各种 PETSc 数据库的使用，如向量，矩阵，指数集，线性和非线性解法器以及图形集。第三部分介绍了各种有用的信息，包括剖面，选项数据库，错误处理，编译和 PETSc 设计细节。

PETSc 已演变成为相当全面的方案，因此，PETSc 用户手册可能会吓到新用户。我们建议读者在认真使用 PETSc 之前先阅读整个使用说明文档，但要紧记在掌握这里介绍的所有材料之后，PETSc 可以得到有效使用。此外，任何 PETSc 功能的参考文件都在用户手册网页。

在 PETSc 分布中，目录$（PETSc_DIR）/docs 文件包含了所有文件。查看介绍 PETSc 功能的手册页面可以访问 http://www.mcs.anl.gov/petsc/petsc-as/documentation。该手册页面提供教程实例的超连结索引（由概念和常规名称组成），这样使得用户能在相关主题之间容易切换。

Emacs 的用户会发现 etags 的选项对探索 PETSc 的源代码很有帮助。此功能的具体介绍在第 14.8 节。

manual.pdf 文件包含完整的 PETSc 用户手册，文件格式为 PDF。而 intro.pdf 只包括介绍部分，第一部分：完整 PETSc 分布，用户手册，手册的页面，和其他信息；这些也可通过 PETSc 网页找到，网址 http://www.mcs.anl.gov/petsc。PETSc 主页还包括安装细节，在最新版本中 PETSc 新特点和变化，我们目前版本支持的机器，故障排除指南，以及常见问题解答。

Fortran 程序员须注意：在大多数的手册中，例子和呼叫序列由 C / C + +编程语言给出的。我们之所以遵循这一惯例，是因为我们建议用 C 或 C + +语言编写 PETSc 代码。然而，纯粹的 Fortran 程序员用 Fortran 编写可以使用 PETSc 的大部分功能，只是在用户界面中有细微的差别。第 11 章讨论了在 PETSc 中使用 Fortran 和 C 的差异，还给出了几个使用 Fortran 的实例。此外本章还介绍了一些直接支持 Fortran90 使用的例程。

1.2 PETSc 程序运行

在使用 PETSc 前，用户必须先设置环境变量 PETSc_DIR，显示完整路径的 PETSc 主目录。例如，UNIX C 下命令以这样的形式 setenv PETSc_DIR $HOME/petsc 放置在用户的.cshrc 文件中。此外，用户必须设置环境变量 PETSC_ARCH。请注意，PETSC_ARCH 只是一个安装用户选择的名称，它指向为某特定的编译器选项和机器类型编译的数据库。使用不同的 PETSC_ARCH 允许用户更容易的管理若干套不同的数据库。

所有 PETSc 程序使用的 MPI（消息传递接口）标准的消息传递通信。因此，执行 PETSc 程序，用户必须知道在自己所选定的计算机系统中运行 MPI 的操作步骤。比如，当使用 MPICH 运行 MPI，下面的命令使用 8 进程启动程序：

mpiexec –np 8 ./petsc_program_name petsc_options

PETSc 也附带一个脚本$PETSC_DIR/bin/petscmpiexec –np 8 ./petsc_program_name petsc_options，使用该信息在$ {PETSC_DIR}/ bmake / $ {PETSC_ARCH} / petscconf 中自动使用正确的 mpiexec 进行配置。

所有 PETSc 兼容的程序支持使用- h 或-help 选项，以及- V 或-version 选项。

某些选项是所有的 PETSc 程序都支持的。我们在下面列出了几个特别有用的选项；该完整列表用-help 选项运行任何 PETSc 程序都能得到。

• - log_summary —总结该程序的性能

• - fp_trap —去除浮点型的例外情况；例如除以零

• - malloc_dump —记忆体追踪；在运行结论中列出未释放记忆体的转储名单

• - malloc_debug —记忆体追踪（默认情况下启动调试版本）

• - start_in_debugger [noxterm,gdb,dbx,xxgdb] [-display name] —在调试器中开始所有的进程。

• - on_error_attach_debugger [noxterm,gdb,dbx,xxgdb] [-display name] —在遇到错误时启动调试器。

关于 PETSc 程序调试的更多信息请看第 14.4 节。

1.3 PETSc程序编写

大多数PETSc 程序开始时调用命令：

PetscInitialize(int *argc,char ***argv,char *file,char *help)

来初始化PETSc 和MPI。参数argc和argv是在C和C++程序中的命令行参数。参数文件指出了PETSc选项文件的另一个名称.petscrc，在默认情况下该文件在用户的主目录之中。第14.2节详细介绍了与该文件相关的内容和PETSc选项数据库，该数据库用于运行时的定制。最后的参数help是一个可选的特征字符串，运行程序时如果使用-help选项该参数将会被列出。在Fortran中，用命令形式call PetscInitialize(character(*) file,integer ierr)进行初始化。如果MPI开始没有进行初始化，PetscInitialize()自动回到命令MPI_Init()。在某些情况下，MPI需要直

接进行初始化（或用其他数据库进行初始化），用户一开始可以执行命令MPI_Init()（或用其它数据库调用），然后调用PetscInitialize()。在默认情况下，PetscInitialize()设置PETSc，由PETSC_COMM_WORLD到MPI_COMM_WORLD进行设定。

对于那些不熟悉MPI的用户，通信装置是一种显示进程的方式，它涉及到计算或通信。通信装置有可变的的类型MPI_Comm。在大多数情况下，用户可以运行通信装置PETSC_COMM_WORLD在给定的运行中显示所有进程，还可以运行PETSC_COMM_SELF来显示单一的进程。

MPI提供由子处理器构成的新产生的通信例程，虽然大多数用户很少需要使用这些。MPI使用方法这本由Lusk, Gropp, and Skjellum编写的书对MPI的相关概念有精辟的介绍，这些资料也可以在MPI的主页http://www.mcs.anl.gov/mpi/ 上见到。请注意，PETSc用户运行程序时无须直接用MPI进行消息传递，但他们必须熟悉消息传递和分布式记忆体计算的基本概念。

在通信中，所有PETSc例程都会返回一个整数无论是否有错误发生。如果错误已被检测到，错误码被设定为非零；否则，它是零。对于C / C + +接口，错误的变数是例程的返回值，而对于Fortran语言版本，每个PETSc 例程有一个整数的错误变数作为最终参数。错误回溯在下面一节进行讨论。

所有PETSc 程序应调用PetscFinalize()作为其最终（或接近最终的）的声明语句，下面分别给出在C / C + +和Fortran中的格式：

```
PetscFinalize();
call PetscFinalize(ierr)
```

本例程在程序总结时处理所谓的选项，如果用语句PetscInitialize()开始运行MPI须调用命令MPI_Finalize()。如果MPI是从PETSc外部初始化（由用户或另一个软件包），用户须负责调用MPI_Finalize()。

1.4 PETSc 简单实例

为了帮助用户能立刻开始使用 Petsc，我们开始以一个简单的单处理器为例（见图 3），该例用有限差分法求解一维的 Laplace 问题。此顺序码可以在${PETSC_DIR}/src/ksp/ksp/examples/tutorials/ex1.c 中找到，说明了如何利用 Petsc 的 KSP、界面预条件、Krylov 子空间法、线性解法器求解线性系统。下面的代码就是我们所强调的这个例子中最重要的一部分。

源文件

PETSc中C/C + +的源文件应使用如下的语句进行声明：#include ”petscksp.h”，petscksp.h是线性解法器数据库的源文件。每个PETSc程序必须指定一个源文件，该文件对应于程序中所需的PETSc最高层的对象，所有需要的低层对象源文件将自动列入上级文件中。例如，petscksp.h包括petscmat.h（矩阵），petscvec.h（向量），和petsc.h（petsc基础文件）。PETSc源文件位于目录${PETSC_DIR}/include中。Fortran语言的PETSc源文件在11.1.1节进行讨论。

选项数据库

如图3所示，用户可以在运行时使用选项数据库输入控制数据。在这个例子中，命令PetscOptionsGetInt(PETSC_NULL,"-n",&n,&flg)检查用户是否提供了设置n值的命令行选项。如果是的话，变量n相应设置；否则，n仍保持不变。一个完整的选择数据库说明详见第14.2节。

向量

用户创建新的平行或连续向量，x，M维用以下命令执行：

VecCreate(MPI Comm comm,Vec *x);
VecSetSizes(Vec x, int m, int M);

这里comm代表MPI通信装置，m选择大小。存储向量空间类型设置调用命令VecSetType()或者VecSetFromOptions()。同一类型的其余向量以这样的形式定义：

VecDuplicate(Vec old,Vec *new);

命令

VecSet(Vec x,PetscScalar value);
VecSetValues(Vec x,int n,int *indices,PetscScalar *values,INSERT VALUES);

分别为向量的所有部分设置某一标值，并给每个组成部分赋不同的值。更详细的关于PETSc向量（包括基本操作、散射/收集、指数集，分发阵列）的信息在第2章讨论。

注意到在该例中变量类型PetscScalar的使用。PetscScala在C/C++编写的PETSc版本中简单定义为双精度型（相应在Fortran语言中的双精度型），该版本不能使用复杂的数字进行编译。当PETSc数据库用复杂数字进行编译时，PetscScalar数据类型能够使用相同的代码。14.7节讨论了PETSc程序中复杂数字的使用。

矩阵

PETSc中矩阵和向量的使用是类似的。用户可创建新的平行或连续矩阵，A，M行N列，使用以下命令：
MatCreate(MPI Comm comm,Mat *A);
MatSetSizes(Mat A,int m,int n,int M,int N);
这里矩阵格式在运行时可以指定。用户可以分别使用m和n指定每一个进程的行和列大小。它们的值可以使用如下命令进行设置：
MatSetValues(Mat A,int m,int *im,int n,int *in,PetscScalar *values,INSERT VALUES)
在所有元素被赋入矩阵之后，必须执行这两个命令：
MatAssemblyBegin(Mat A,MAT FINAL ASSEMBLY);

MatAssemblyEnd(Mat A,MAT FINAL ASSEMBLY);
第3章讨论了各种各样的矩阵格式以及一些基本矩阵操纵例程。

线性方程求解器

在创建矩阵和向量之后就需要定义线性系统，AX = B，接着用户可以使用线性方程求解器用以下连续的命令求解该线性系统。
KSPCreate(MPI Comm comm,KSP *ksp);
KSPSetOperators(KSP ksp,Mat A,Mat PrecA,MatStructure flag);
KSPSetFromOptions(KSP ksp);
KSPSolve(KSP ksp,Vec b,Vec x);
KSPDestroy(KSP ksp);

用户首先创建KSP，为其设定特定的求解环境（线性系统矩阵和选择不同的预处理矩阵）。然后用户选择各种不同的自定义求解方案来求解该线性系统，最终注销KSP对象。我们强调命令KSPSetFromOptions()，该命令可让使用者在运行时使用选项数据库自定义求解线性系统的解决方案，这在第14.2节进行介绍。通过这个数据库，用户不仅可以选择迭代方法和预

条件子，还可以设置各种监测例程等。

第四章详细介绍了KSP，包括PC和KSP的预条件子以及Krylov子空间方法。

非线性方程求解器

大部分偏微分方程问题在本质上是非线性的，PETSc提供了一个直接处理非线性问题的接口叫做SNES。第5章详细介绍了非线性方程求解器。我们建议大部分的PETSc用户直接使用SNES进行求解，而不是在非线性方程求解器中使用PETSc来求解非线性问题。

错误检查

在通信中，所有PETSc例程都会返回一个整数无论是否有错误发生。PETSc的宏指令CHKERRQ（ierr）检查ierr的值，并调用PETSc错误处理器进行错误检测。CHKERRQ（ierr）在所有子程序中使用时能进行完整的错误追踪。在图4我们看到在PETSc样例中由错误检测生成的追踪。发生错误的行在文件${PETSC_DIR}/src/mat/impls/aij/seq/aij.c的第1673行，该错误是由于数组在记忆体中分配过大而产生的。例行在程序ex3.c中的第71行被调用。关于使用PETSc Fortran接口的错误检查内容介绍详见第11.1.2节。

运行PETSc数据库的调试版本会进行大量记忆体损毁情况的检测工作（阵列范围外编写）。宏指令CHKMEMQ可在代码中的任何位置被调用来检查记忆体损毁的现状。把部分（或很多）宏写进您的代码之中，这样就可以轻松地跟踪出哪些部分您的代码已被损毁。

并行编程

PETSc使用消息传递模型进行并行编程使用MPI进行处理机间的通信，因此，用户通过应用代码免费使用所需的MPI例程。但是，默认情况下在PETSc中用户受到很多细节的消息传递保护，这是因为它们都是隐藏在并行对象中的，如向量、矩阵、解法器。此外，PETSc提供诸如广义向量分散/搜集和分布式阵列的工具来协助管理并行数据。

用户谨记在建立PETSc对象（如向量，矩阵，或解法器）时要指定一个通信器来显示分发对象的处理器。例如，如上所述，一些创建矩阵、向量、线性解法器的命令：

```
MatCreate(MPI Comm comm,Mat *A);
VecCreate(MPI Comm comm,Vec *x);
KSPCreate(MPI Comm comm,KSP *ksp);
```

建立的例程集合了通信器上所有的进程；因此，这所有进程都要调用该例程。此外，如果使用了连续的例程集合，用户必须在每个进程中以同样的顺序调用它们。

下面的例子（如图5）说明如何并行求解线性系统。此代码用有限差分法处理二维拉普拉斯离散问题，在这里线性系统借助于KSP求解。在图3可看出，作为连续版本该代码执行相同的任务。请注意，用户启动程序，创建向量和矩阵以及求解线性系统在单进程和多进程例子中是完全一样的。图3和图5两个例子主要差异在于，在并行条件下每个进程形成部分矩阵和向量。

程序的编译和运行

图6说明了使用MPICH编译和运行PETSc程序。请注意不同的网站上的数据库编译器的名称可能会稍有不同。第15章讨论了PETSc编译的相关内容。用户遇到与PETSc程序相关的困难时应参阅PETSc主页的故障排除指南或者文件$PETSC DIR/docs/troubleshooting.html。PETSc主页网址为http://www.mcs.anl.gov/petsc。

如图7所示，选项-log_summary详尽列出了运行性能表现，包括时间、浮点运算（失败）率，以及消息传递活动。第12章详细介绍该选项的概况，包括在图7中所示的输出数据的具

体说明。这个例子涉及到使用GMRES方法和ILU方法在单进程中求解线性系统。该例中的低浮点运算（失败）率取决于代码求解微小系统。我们举这个例子只是为了证明提取运行性能的信息是很方便的。

使用PETSc编写应用代码

该例演示了软件的使用情况，它可以作为发展自定义应用程式的范本。我们建议新用户在目录${PETSC_DIR}/src/<library>/examples/tutorials中检测程序。这里的<library>代表任一PETSc数据库（在下面一节列出），例如snes、ksp。手册页位于$PETSC DIR/docs/index.html或http://www.mcs.anl.gov/petsc/petsc-as/documentation，为范例提供索引（由例程的名称和概念组成）。

我们建议按照以下步骤使用PETSc编写应用程序：

1、根据PETSc网站上的指令安装并测试PETSc。

2、在目录中复制一个PETSc例子，该例与你所感兴趣的问题同属一类。（例如，求解线性问题时复制例${PETSC_DIR}/src/ksp/ksp/examples/tutorials。

3、在例子所在目录下复制相应的编译文件，编译并运行该程序。

4、使用例子程序作为起点，编写自定义代码程序。

Abstract:

This manual describes the use of PETSc for the numerical solution of partial differential equations and related problems on high-performance computers. The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers. PETSc uses the MPI standard for all message-passing communication.

PETSc includes an expanding suite of parallel linear, nonlinear equation solvers and time integrators that may be used in application codes written in Fortran, C, and C++. PETSc provides many of the mechanisms needed within parallel application codes, such as parallel matrix and vector assembly routines. The library is organized hierarchically, enabling users to employ the level of abstraction that is most appropriate for a particular problem. By using techniques of object-oriented programming, PETSc provides enormous flexibility for users.

PETSc is a sophisticated set of software tools; as such, for some users it initially has a much steeper learning curve than a simple subroutine library. In particular, for individuals without some computer science background or experience programming in C or C++, it may require a significant amount of time to take full advantage of the features that enable efficient software use. However, the power of the PETSc design and the algorithms it incorporates may make the efficient implementation of many application codes simpler than “rolling them” yourself.

• For many simple (or even relatively complicated) tasks a package such as Matlab is often the best tool; PETSc is not intended for the classes of problems for which effective Matlab code can be written.

• PETSc should not be used to attempt to provide a “parallel linear solver” in an otherwise

sequential code. Certainly all parts of a previously sequential code need not be parallelized but the matrix generation portion must be to expect any kind of reasonable performance. Do not expect to generate your matrix sequentially and then "use PETSc" to solve the linear system in parallel.

Since PETSc is under continued development, small changes in usage and calling sequences of routines may occur. PETSc is supported; see the web site http://www.mcs.anl.gov/petsc for information on contacting support.

A list of publications and web sites that feature work involving PETSc may be found at http://www.unix.mcs.anl.gov/petsc/petsc-as/publications.

We welcome any additions to these pages.

Chapter1 Getting Started

The Portable, Extensible Toolkit for Scientific Computation (PETSc) has successfully demonstrated that the use of modern programming paradigms can ease the development of large-scale scientific application codes in Fortran, C, and C++. Begun several years ago, the software has evolved into a powerful set of tools for the numerical solution of partial differential equations and related problems on high-performance computers.

PETSc consists of a variety of libraries (similar to classes in C++), which are discussed in detail in Parts II and III of the users manual. Each library manipulates a particular family of objects (for instance, vectors) and the operations one would like to perform on the objects. The objects and operations in PETSc are derived from our long experiences with scientific computation. Some of the PETSc modules deal with

• index sets, including permutations, for indexing into vectors, renumbering, etc;

• vectors;

• matrices (generally sparse);

• distributed arrays (useful for parallelizing regular grid-based problems);

• Krylov subspace methods;

• preconditioners, including multigrid and sparse direct solvers;

• nonlinear solvers; and

• timesteppers for solving time-dependent (nonlinear) PDEs.

Each consists of an abstract interface (simply a set of calling sequences) and one or more implementations using particular data structures. Thus, PETSc provides clean and effective codes for the various phases of solving PDEs, with a uniform approach for each class of problems. This design enables easy comparison and use of different algorithms (for example, to experiment with different Krylov subspace methods, preconditioners, or truncated Newton methods). Hence, PETSc provides a rich environment for modeling scientific applications as well as for rapid algorithm design and prototyping.

The libraries enable easy customization and extension of both algorithms and implementations. This approach promotes code reuse and flexibility, and separates the issues of

parallelism from the choice of algorithms. The PETSc infrastructure creates a foundation for building large-scale applications.

It is useful to consider the interrelationships among different pieces of PETSc. Figure 1 is a diagram of some of these pieces; Figure 2 presents several of the individual parts in more detail. These figures illustrate the library's hierarchical organization, which enables users to employ the level of abstraction that is most appropriate for a particular problem.

1.1 Suggested Reading

The manual is divided into three parts:

- Part I - Introduction to PETSc
- Part II - Programming with PETSc
- Part III - Additional Information

Part I describes the basic procedure for using the PETSc library and presents two simple examples of solving linear systems with PETSc. This section conveys the typical style used throughout the library and enables the application programmer to begin using the software immediately. Part I is also distributed separately for individuals interested in an overview of the PETSc software, excluding the details of library usage. Readers of this separate distribution of Part I should note that all references within the text to particular chapters and sections indicate locations in the complete users manual.

Part II explains in detail the use of the various PETSc libraries, such as vectors, matrices, index sets, linear and nonlinear solvers, and graphics. Part III describes a variety of useful information, including profiling, the options database, viewers, error handling, makefiles, and some details of PETSc design.

PETSc has evolved to become quite a comprehensive package, and therefore the PETSc Users Manual can be rather intimidating for new users. We recommend that one initially read the entire document before proceeding with serious use of PETSc, but bear in mind that PETSc can be used efficiently before one understands all of the material presented here. Furthermore, the definitive reference for any PETSc function is always the online manual page.

Within the PETSc distribution, the directory ${PETSC_DIR}/docs contains all documentation. Manual pages for all PETSc functions can be accessed on line at

http://www.mcs.anl.gov/petsc/petsc-as/documentation

The manual pages provide hyperlinked indices (organized by both concepts and routine names) to the tutorial examples and enable easy movement among related topics.

Emacs users may find the etags option to be extremely useful for exploring the PETSc source code.Details of this feature are provided in Section 14.8.

The file manual.pdf contains the complete PETSc Users Manual in the portable document format(PDF), while intro.pdf includes only the introductory segment, Part I. The complete PETSc distribution, users manual, manual pages, and additional information are also available via the PETSc home page at http://www.mcs.anl.gov/petsc. The PETSc home page also contains details regarding installation, new features and changes in recent versions of PETSc, machines that we

currently support, a troubleshooting guide, and a FAQ list for frequently asked questions.

Note to Fortran Programmers: In most of the manual, the examples and calling sequences are given for the C/C++ family of programming languages. We follow this convention because we recommend that PETSc applications be coded in C or C++. However, pure Fortran programmers can use most of the functionality of PETSc from Fortran, with only minor differences in the user interface. Chapter 11 provides a discussion of the differences between using PETSc from Fortran and C, as well as several complete Fortran examples. This chapter also introduces some routines that support direct use of Fortran90 pointers.

1.2 Running PETSc Programs

Before using PETSc, the user must first set the environmental variable PETSC_DIR, indicating the full path of the PETSc home directory. For example, under the UNIX C shell a command of the form

setenv PETSC DIR $HOME/petsc

can be placed in the user's .cshrc file. In addition, the user must set the environmental variable PETSC_ARCH to specify the architecture. Note that PETSC_ARCH is just a name selected by the installer to refer to the libraries compiled for a particular set of compiler options and machine type. Using different PETSC_ARCH allows one to manage several different sets of libraries easily.

All PETSc programs use the MPI (Message Passing Interface) standard for message-passing communication[15]. Thus, to execute PETSc programs, users must know the procedure for beginning MPI jobs on their selected computer system(s). For instance, when using the MPICH implementation of MPI [9] and many others, the following command initiates a program that uses eight processors:

mpiexec -np 8 ./petsc program name petsc options

PETSc also comes with a script

$PETSC DIR/bin/petscmpiexec -np 8 ./petsc program name petsc options

that uses the information set in ${PETSC_DIR}/bmake/${PETSC_ARCH}/petscconf to automatically use the correct mpiexec for your configuration.

All PETSc-compliant programs support the use of the -h or -help option as well as the -v or -version option.

Certain options are supported by all PETSc programs. We list a few particularly useful ones below; a complete list can be obtained by running any PETSc program with the option -help.

- -log_summary - summarize the program's performance
- -fp_trap - stop on floating-point exceptions; for example divide by zero
- -malloc_dump - enable memory tracing; dump list of unfreed memory at conclusion of the run
- -malloc_debug - enable memory tracing (by default this is activated for debugging versions)
- -start_in_debugger [noxterm,gdb,dbx,xxgdb] [-display name] - start all processes in

debugger

• -on_error_attach_debugger [noxterm,gdb,dbx,xxgdb] [-display name] – start debugger only on encountering an error

See Section 14.4 for more information on debugging PETSc programs.

1.3 Writing PETSc Programs

Most PETSc programs begin with a call to

PetscInitialize(int *argc,char ***argv,char *file,char *help);

which initializes PETSc and MPI. The arguments argc and argv are the command line arguments delivered in all C and C++ programs. The argument file optionally indicates an alternative name for the PETSc options file, .petscrc, which resides by default in the user's home directory. Section 14.2 provides details regarding this file and the PETSc options database, which can be used for runtime customization. The final argument, help, is an optional character string that will be printed if the program is run with the –help option. In Fortran the initialization command has the form

call PetscInitialize(character(*) file,integer ierr);

PetscInitialize() automatically calls MPI_Init() if MPI has not been not previously initialized. In certain circumstances in which MPI needs to be initialized directly (or is initialized by some other library), the user can first call MPI_Init() (or have the other library do it), and then call PetscInitialize(). By default, PetscInitialize() sets the PETSc "world" communicator, given by PETSC_COMM_WORLD, to MPI_COMM_WORLD.

For those not familar with MPI, a communicator is a way of indicating a collection of processes that will be involved together in a calculation or communication. Communicators have the variable type MPI_Comm.In most cases users can employ the communicator PETSC_COMM_WORLD to indicate all processes in a given run and PETSC_COMM_SELF to indicate a single process.

MPI provides routines for generating new communicators consisting of subsets of processors, though most users rarely need to use these. The book Using MPI, by Lusk, Gropp, and Skjellum [10] provides an excellent introduction to the concepts in MPI, see also the MPI homepage http://www.mcs.anl.gov/mpi/. Note that PETSc users need not program much message passing directly with MPI, but they must be familiar with the basic concepts of message passing and distributed memory computing.

All PETSc routines return an integer indicating whether an error has occurred during the call. The error code is set to be nonzero if an error has been detected; otherwise, it is zero. For the C/C++ interface, the error variable is the routine's return value, while for the Fortran version, each PETSc routine has as its final argument an integer error variable. Error tracebacks are discussed in the following section.

All PETSc programs should call PetscFinalize() as their final (or nearly final) statement, as given below in the C/C++ and Fortran formats, respectively:

PetscFinalize();
call PetscFinalize(ierr)

This routine handles options to be called at the conclusion of the program, and calls MPI_Finalize() if PetscInitialize() began MPI. If MPI was initiated externally from PETSc (by either the user or another software package), the user is responsible for calling MPI_Finalize().

1.4 Simple PETSc Examples

To help the user start using PETSc immediately, we begin with a simple uniprocessor example in Figure 3 that solves the one-dimensional Laplacian problem with finite differences. This sequential code, which can be found in ${PETSC_DIR}/src/ksp/ksp/examples/tutorials/ex1.c, illustrates the solution of a linear system with KSP, the interface to the preconditioners, Krylov subspace methods, and direct linear solvers of PETSc. Following the code we highlight a few of the most important parts of this example.

Include Files

The C/C++ include files for PETSc should be used via statements such as

#include “petscksp.h”

where petscksp.h is the include file for the linear solver library. Each PETSc program must specify an include file that corresponds to the highest level PETSc objects needed within the program; all of the required lower level include files are automatically included within the higher level files. For example, petscksp.h includes petscmat.h (matrices), petscvec.h (vectors), and petsc.h (base PETSc file). The PETSc include files are located in the directory ${PETSC_DIR}/include. See Section 11.1.1

for a discussion of PETSc include files in Fortran programs.

The Options Database

As shown in Figure 3, the user can input control data at run time using the options database. In this example the command PetscOptionsGetInt(PETSC_NULL,"-n",&n,&flg); checks whether the user has provided a command line option to set the value of n, the problem dimension. If so, the variable n is set accordingly; otherwise, n remains unchanged. A complete description of the options database may be found in Section 14.2.

Vectors

One creates a new parallel or sequential vector, x, of global dimension M with the commands

VecCreate(MPI Comm comm,Vec *x);
VecSetSizes(Vec x, int m, int M);

where comm denotes the MPI communicator and m is the optional local size which may be PETSC_ DECIDE. The type of storage for the vector may be set with either calls to VecSetType() or VecSetFrom Options(). Additional vectors of the same type can be formed with

VecDuplicate(Vec old,Vec *new);

The commands

VecSet(Vec x,PetscScalar value);
VecSetValues(Vec x,int n,int *indices,PetscScalar *values,INSERT VALUES);

respectively set all the components of a vector to a particular scalar value and assign a different value to each component. More detailed information about PETSc vectors, including their basic

operations, scattering/gathering, index sets, and distributed arrays, is discussed in Chapter 2.

Note the use of the PETSc variable type PetscScalar in this example. The PetscScalar is simply defined to be double in C/C++ (or correspondingly double precision in Fortran) for versions of PETSc that have not been compiled for use with complex numbers. The PetscScalar data type enables identical code to be used when the PETSc libraries have been compiled for use with complex numbers.Section 14.7 discusses the use of complex numbers in PETSc programs.

Matrices

Usage of PETSc matrices and vectors is similar. The user can create a new parallel or sequential matrix, A,which has M global rows and N global columns, with the routines and

MatCreate(MPI Comm comm,Mat *A);
MatSetSizes(Mat A,int m,int n,int M,int N);

where the matrix format can be specified at runtime. The user could alternatively specify each processes' number of local rows and columns using m and n. Values can then be set with the command

MatSetValues(Mat A,int m,int *im,int n,int *in,PetscScalar *values,INSERT VALUES);

After all elements have been inserted into the matrix, it must be processed with the pair of commands

MatAssemblyBegin(Mat A,MAT FINAL ASSEMBLY);
MatAssemblyEnd(Mat A,MAT FINAL ASSEMBLY);

Chapter 3 discusses various matrix formats as well as the details of some basic matrix manipulation routines.

Linear Solvers

After creating the matrix and vectors that define a linear system, Ax = b, the user can then use KSP to solve the system with the following sequence of commands:

KSPCreate(MPI Comm comm,KSP *ksp);
KSPSetOperators(KSP ksp,Mat A,Mat PrecA,MatStructure flag);
KSPSetFromOptions(KSP ksp);
KSPSolve(KSP ksp,Vec b,Vec x);
KSPDestroy(KSP ksp);

The user first creates the KSP context and sets the operators associated with the system (linear system matrix and optionally different preconditioning matrix). The user then sets various options for customized solution, solves the linear system, and finally destroys the KSP context. We emphasize the command KSPSetFromOptions(), which enables the user to customize the linear solution method at runtime by using the options database, which is discussed in Section 14.2. Through this database, the user not only can select an iterative method and preconditioner, but also can prescribe the convergence tolerance, set various monitoring routines, etc. (see, e.g., Figure 7).

Chapter 4 describes in detail the KSP package, including the PC and KSP packages for

preconditioners and Krylov subspace methods.

Nonlinear Solvers

Most PDE problems of interest are inherently nonlinear. PETSc provides an interface to tackle the nonlinear problems directly called SNES. Chapter 5 describes the nonlinear solvers in detail. We recommend most PETSc users work directly with SNES, rather than using PETSc for the linear problem within a nonlinear solver.

Error Checking

All PETSc routines return an integer indicating whether an error has occurred during the call. The PETSc macro CHKERRQ(ierr) checks the value of ierr and calls the PETSc error handler upon error detection. CHKERRQ(ierr) should be used in all subroutines to enable a complete error traceback. In Figure 4 we indicate a traceback generated by error detection within a sample PETSc program. The error occurred on line 1673 of the file ${PETSC_DIR}/src/mat/impls/aij/seq/aij.c and was caused by trying to allocate too large an array in memory. The routine was called in the program ex3.c on line 71. See Section 11.1.2 for details regarding error checking when using the PETSc Fortran interface.

When running the debug version of the PETSc libraries, it does a great deal of checking for memory corruption (writing outside of array bounds etc). The macros CHKMEMQ can be called anywhere in the code to check the current status of the memory for corruption. By putting several (or many) of these macros into your code you can usually easily track down in what small segment of your code the corruption has occured.

Parallel Programming

Since PETSc uses the message-passing model for parallel programming and employs MPI for all interprocessor communication, the user is free to employ MPI routines as needed throughout an application code. However, by default the user is shielded from many of the details of message passing within PETSc, since these are hidden within parallel objects, such as vectors, matrices, and solvers. In addition, PETSc provides tools such as generalized vector scatters/gathers and distributed arrays to assist in the management of parallel data.

Recall that the user must specify a communicator upon creation of any PETSc object (such as a vector,matrix, or solver) to indicate the processors over which the object is to be distributed. For example, as mentioned above, some commands for matrix, vector, and linear solver creation are:

```
MatCreate(MPI Comm comm,Mat *A);
VecCreate(MPI Comm comm,Vec *x);
KSPCreate(MPI Comm comm,KSP *ksp);
```

The creation routines are collective over all processors in the communicator; thus, all processors in the communicator must call the creation routine. In addition, if a sequence of collective routines is being used, they must be called in the same order on each processor.

The next example, given in Figure 5, illustrates the solution of a linear system in parallel. This code, corresponding to ${PETSC_DIR}/src/ksp/ksp/examples/tutorials/ex2.c, handles the twodimensional Laplacian discretized with finite differences, where the linear system is again solved with KSP. The code performs the same tasks as the sequential version within Figure 3. Note that the user interface for initiating the program, creating vectors and matrices, and solving

the linear system is exactly the same for the uniprocessor and multiprocessor examples. The primary difference between the examples in Figures 3 and 5 is that each processor forms only its local part of the matrix and vectors in the parallel case.

Compiling and Running Programs

Figure 6 illustrates compiling and running a PETSc program using MPICH. Note that different sites may have slightly different library and compiler names. See Chapter 15 for a discussion about compiling PETSc programs. Users who are experiencing difficulties linking PETSc programs should refer to the troubleshooting guide via the PETSc WWW home page http://www.mcs.anl.gov/petsc or given in the file $PETSC DIR/docs/troubleshooting.html.

As shown in Figure 7, the option -log_summary activates printing of a performance summary, including times, floating point operation (flop) rates, and message-passing activity. Chapter 12 provides details about profiling, including interpretation of the output data within Figure 7. This particular example involves the solution of a linear system on one processor using GMRES and ILU. The low floating point operation (flop) rates in this example are due to the fact that the code solved a tiny system. We include this example merely to demonstrate the ease of extracting performance information.

Writing Application Codes with PETSc

The examples throughout the library demonstrate the software usage and can serve as templates for developing custom applications. We suggest that new PETSc users examine programs in the directories

${PETSC_DIR}/src/<library>/examples/tutorials,

where <library> denotes any of the PETSc libraries (listed in the following section), such as snes or ksp. The manual pages located at

$PETSC DIR/docs/index.html or
http://www.mcs.anl.gov/petsc/petsc-as/documentation
provide indices (organized by both routine names and concepts) to the tutorial examples.

To write a new application program using PETSc, we suggest the following procedure:

1. Install and test PETSc according to the instructions at the PETSc web site.

2. Copy one of the many PETSc examples in the directory that corresponds to the class of problem of interest (e.g., for linear solvers, see ${PETSC_DIR}/src/ksp/ksp/examples/tutorials).

3. Copy the corresponding makefile within the example directory; compile and run the example program.

4. Use the example program as a starting point for developing a custom code.