arXiv:2603.14040v1 [cs.MS] 14 Mar 2026

Pyroclast：一款用于地球动力学的模块化高性能 Python 求解器

Marcel Ferrari

苏黎世联邦理工学院地球与行星科学系地球物理流体动力学组¨
苏黎世

摘要

本专著介绍了 Pyroclast 的设计、实现与评估。Pyroclast 是一个模块化、高性能的 Python 框架，面向大规模地球动力学模拟。传统地球动力学求解器通常采用单体式 Fortran、C++ 或 C 代码库实现，在图形处理器支持和可扩展性方面存在局限。Pyroclast 通过融合现代数值方法、硬件加速执行以及灵活的面向对象软件架构，解决了这些关键限制。Pyroclast 从设计之初便面向分布式和图形处理器加速环境，为使用标记点网格法和无矩阵有限差分离散模拟地幔对流和岩石圈变形，提供了易用且高效的平台。

本研究致力于实现一个稳健且可扩展的二维黏性力学求解器，为未来的黏弹塑性模型提供计算核心。该核心包括不可压缩斯托克斯方程的应力守恒交错网格离散、无矩阵几何多重网格求解器、对 Krylov 和拟牛顿求解器的支持，以及基于 MPI 的区域分解。所有求解器组件均针对 GPU 加速执行而设计

我们开展了初步基准测试，以评估性能与可扩展性。共享内存基准测试显示，斯托克斯求解器具有良好的强扩展性；与多核中央处理器基准相比，其在 NVIDIA A100 GPU 上的加速比达到 5–10× 。分布式平流基准测试验证了区域分解策略的可扩展性：在 7 个计算节点上使用 896 个 CPU核心时，弱扩展性接近理想状态。结果证实，Pyroclast 在实现高性能的同时，通过高级 Python 接口保持了灵活性，也更便于非专业用户使用。

本研究不仅提供了一个功能完善且可扩展的仿真框架，也为现代化改造旧有地球动力学代码提供了蓝图。其模块化架构和原生 Python 实现降低了研究人员的入门门槛，同时支持与现代机器学习库直接互操作。这种集成为结合基于物理的模拟与数据驱动建模的混合工作流打开了大门，加速了地球动力学及其他领域的科学发现。

目录

目录 ii

1 引言 1

1.1 地球动力学为何重要.1 1.2 数值地球动力学建模.2 1.3 地球动力学模拟面临的挑战.2 1.4 Pyroclast 的设计动机.3 1.5 目标与贡献.5 1.6 名称由来.5 1.7 论文大纲.6

2 相关工作 8

2.1 现有地球动力学求解器.8 2.1.1 长期岩石圈动力学.9 2.1.2 地震波传播.11 2.2 传统模拟框架的局限性.12 2.3 Python 生态系统的优势.13 2.4 科学计算 Python 中的领域特定语言.15 2.5 科学计算中的机器学习.16 2.6 Pyroclast 的定位.17 2.7 本工作的范围.18

3 控制方程 20

3.1 物理假设.20 3.2 质量与动量守恒.21 3.3 平流方程.22 3.4 边界条件.23 3.5 无量纲化与单位缩放.24

4 数值方法 26

4.1 标记点网格法.26 4.2 标记点与网格之间的插值.28 4.3 拉格朗日标记点平流.30 4.3.1 时间步长选取与稳定性.30 4.3.2 平流方程积分格式.30 4.4 控制方程离散化.33 4.4.1 交错网格布局.34 4.4.2 应力守恒有限差分.36 4.5 Stokes系统快速求解器.37 4.5.1 Uzawa 迭代.39 4.5.2 压力零空间消除.42 4.6 多重网格法.43 4.6.1 平滑与信息传播.44 4.6.2 多重网格循环. 45 4.6.3 几何多重网格与代数多重网格 . . . . . 47 4.6.4 限制与延拓算子.48 4.6.5 平滑器.52 4.6.6 多重网格求解斯托克斯系统 55 4.6.7 黏度重标度与初始猜测.55 4.7 加速器.56

4.7.1 Uzawa 迭代作为不动点法 . . . . . 57 4.7.2 广义共轭残差法 . . . . . 59 4.7.3 安德森加速 .61 4.8 收敛性与精度评估 .62 4.8.1 残差范数.62 4.9 数值稳定性与求解器鲁棒性.64 4.9.1 基准问题.65 4.9.2 求解器配置.66 4.9.3 平滑器配置.66 4.9.4 结果与讨论.66

5 软件架构与模拟工作流 69

5.1 设计理念.70 5.2 求解器编排与数据所有权.71 5.3 模型的组合与继承. 73 5.4 高性能 Python 软件栈 75

6 GPU 支持与设备无关执行 77

6.1 GPU 集成理念.78 6.1.1 设备无关数组编程.78 6.1.2 编译内核的动态分派 . . . . . 78 6.2 标记点平流与基于数组的 GPU 操作 . . . . . 79 6.2.1 内存限制与混合精度平流 . 80 6.3 模板运算与 Numba 内核.81

6.4 GPU 多重网格与混合执行.83

7 分布式执行与消息传递接口并行性 87

7.1 MPI 编程模型概述.88 7.1.1 进程到设备的映射.89 7.2 结构化网格的域分解.89 7.3 MPI 通信器与笛卡尔拓扑.91 7.4 分布式域中的标记点.92 7.4.1 分布式标记点插值.94 7.5 标记点平流策略.96 7.6 分布式多重网格与 RAS 预条件 99

8 基准测试与验证 101

8.1 硬件和软件配置.101 8.2 共享内存斯托克斯求解器基准测试.102 8.2.1 基准测试设置.103 8.2.2 收敛.103 8.2.3 强扩展性能.104 8.2.4 GPU 加速与 CPU 比较.106 8.3 分布式平流基准测试.107 8.3.1 基准测试设置.107 8.3.2 弱扩展结果.109 8.4 讨论与总结.111

9 结论与未来工作 113

9.1 核心贡献.113 9.1.1 面向地球动力学的现代软件架构 . . . 113 9.1.2 无矩阵多重网格求解器.114 9.1.3 现代硬件上的性能与可扩展性 . . 114 9.2 影响与科学意义.114 9.2.1 降低大规模模拟的门槛 . . . . . 115 9.2.2 与机器学习的集成.115 9.2.3 现代化遗留代码的蓝图 . . . . . 115 9.3 未来工作.116 9.3.1 完整的热力学耦合.116 9.3.2 三维模型.116 9.3.3 分布式多重网格支持 . . . . . 116 9.3.4 数值与性能改进.116 9.3.5 物理与机器学习混合方法 117 9.4 在线资源.117 9.5 最后致辞.117 致谢.118

参考文献 119

第 1 章

引言

理解地球内部的动力学是地球科学的一项根本挑战，对科学研究和社会应用都具有重要意义。地幔对流、岩石圈变形和板块构造等地球动力学过程，塑造着地球在地质时间尺度上的结构与演化。数值建模已成为探索这些过程不可或缺的工具，使科学家能够研究那些无法通过直接观测触及的复杂多尺度相互作用 [1]。本文介绍了 *Pyroclast* 的设计、实现与评估。这是一款使用 Python 编程语言构建的高性能、模块化且可扩展的求解器，适用于大规模地球动力学模拟。本研究连接了地球物理建模、数值方法、现代高性能计算技术与科学软件工程。

1.1 地球动力学为何重要

地球动力学研究支配地球内部与地表结构、演化和动力学的物理过程。它探讨广泛的地质现象，包括构造板块的运动、岩石圈的形成与变形、地幔对流和俯冲带。地球动力学过程不仅影响地球深部的动力学，也造成板块构造、地球地形演变、地震和火山活动等地表现象 [2, 3]。地球动力学还有助于我们理解地球的热演化，以及它如何影响岩石圈的长期变形模式 [4, 5, 6, 7]。因此，它在地球科学中发挥着核心作用，连接地球物理学、地质学、地球化学和行星科学。理解这些过程不仅具有科学意义，也直接关系到自然灾害评估、地下资源管理以及地球地质历史的重建。

1.2 数值地球动力学建模

尽管地球物理学和地质学研究已持续数十年，我们观测地球内部的能力在空间和时间上仍受到根本限制。直接测量基本局限于地表及地下最上方数十公里，而地球内部深达 6000 多公里。同样，地质记录仅保存了过去数十亿年间零星的信息。因此，地球历史的大部分内容及其深部结构都无法直接观测 [1]。研究其他行星天体时，这一局限尤为明显。

数值建模旨在通过提供一个基于物理原理的框架，模拟地球内部和地表动力学，从而弥合缺乏直接观测数据所造成的间隙。它不依赖有限的测量数据，而是借助数学工具来描述地球动力学过程，尤其是偏微分方程；这类方程表达了变形介质中的质量守恒、动量守恒和能量守恒等原理。通过在真实的边界条件和材料条件下求解这些基本方程，我们可以重建造成地幔对流、板块构造和岩石圈变形等大尺度地质现象的物理机制。尤其是，这种方法使研究人员能够以受控且定量的方式，探究地质系统的非线性行为和涌现行为 [1]。

在此背景下，地球动力学研究已与数值建模紧密结合。模型日益复杂，计算资源也日渐充足，这不断拓展着可模拟内容的范围。因此，建模已成为假设发现和定量研究地球过程及其他地质系统的基础组成部分 [1]。

1.3 地球动力学模拟的挑战

地球动力学模拟面临诸多根本性挑战，这些挑战既源于底层物理过程的复杂性，也源于大规模建模所需的计算需求。

首先，地球动力学系统涉及多个相互耦合的物理过程，这些过程以高度非线性的方式相互作用。其中既包括地幔对流等缓慢的长期变形过程，也包括热扩散和地震事件等更快的过程。要准确捕捉这些相互作用，就需要能够处理跨越巨大空间和时间尺度现象的先进多物理场求解器。在单个模拟循环中解析所有这些尺度，会带来显著的数值和算法困难 [8, 9]。

其次，地球动力学问题的数学表述往往会导致极度病态的方程系统。例如，将岩石建模为连续介质时便是如此。鉴于极端的黏度值，所得物理量会跨越数个数量级。例如，构造板块速度通常介于 2 至 8 cm/年之间，即约为 6×10^{-10} 至 2.5×10^{-9} m/s；而内部偏应力通常介于 200 至 300 巴（2×10^{7} 至 3×10^{7} N/m^2），在火山热点等强烈变形区域还可能超过 1 kbar（10^{8} N/m^2）。由此产生的线性方程组的条件数可能超过 10^{25}，即使对于最先进的数值求解器而言，这也是一项挑战 [10]。

第三，足够的空间和时间分辨率对于准确捕捉地球动力学系统的动态至关重要。自适应网格加密策略仅在感兴趣区域提高分辨率，已被成功用于降低计算成本 [11]。然而，高保真模拟仍然需要高昂的计算成本。为了使大规模建模成为可能，优化数值求解器并充分利用现代高性能计算架构（包括多核 CPU 和 GPU）至关重要。因此，采用高效算法和硬件加速是推动计算地球动力学科学进步的必要条件。

最后，地球科学家所具备的专业知识与实现和运行高性能数值代码所需的技术工程知识之间仍存在 substantial 的间隙。地质学和地球物理学领域的许多研究人员没有接受过数值方法、并行计算或性能优化方面的正规培训。这限制了先进建模工具的可及性，也阻碍了现代软件和硬件能力在地球动力学界的更广泛采用。

1.4 Pyroclast 的动机

尽管过去几十年来地球动力学建模取得了显著进展，但许多广泛使用的模拟工具仍然建立在使用 Fortran、C 或 C++ 等低级语言编写的遗留代码库之上。这些单体式、以 CPU 为中心的代码对于推动该领域的发展至关重要，但往往难以扩展、维护，或适应新的计算范式。在这些框架中加入新物理、利用 GPU 加速，或与现代机器学习工作流集成，通常十分困难，甚至无法实现。

与此同时，更广泛的地球系统科学界已开始采用现代化的模块化模拟软件方法。尤其是，气候和天气建模领域率先开展了相关工作，推动从传统的高性能计算（HPC）工作流程转向兼顾性能与开发者

生产力的基于 Python 的框架。将 ICON 天气模型移植到 GPU 原生 Python 工具等项目，展现了这一转变的潜力 [12]。这些举措利用了不断发展的 AI 优化的软件和硬件生态系统，而该生态系统正日益以 Python 为核心构建。

尽管 GPU4GEO [13] 等近期项目已探索使用 Julia 进行 GPU 加速地球物理建模，但现有基于 Julia 的工作主要集中于伪瞬态求解器，以及标记点网格法中的平流分量 [14]。然而，据我们所知，迄今尚无类似工作针对 Python 展开。这一区别十分重要，因为 Python 在高性能计算与机器学习的交汇处占据着独特的位置。绝大多数现代 AI 软件基础设施，包括 PyTorch [15] 和 TensorFlow [16], 等框架，都是为 Python 编写并与之深度集成的。无缝接入这一生态系统，或将为地球动力学研究开辟新方向，使物理驱动与数据驱动的方法得以结合，形成混合模拟策略。Pyroclast 正是为弥补这一间隙而开发的。它从设计之初就旨在通过模块化设计、GPU 加速，以及与基于 Python 的科学和 AI 工具原生兼容，实现地球物理建模。通过重新思考地球动力学求解器的结构与实现方式，Pyroclast 致力于弥合性能与易用性之间的差距，并帮助新一代地球科学家站上高性能计算与 AI 支持的模拟前沿。

Pyroclast 的主要目标是创建一个模块化且易于维护的仿真框架，使地球科学家能够在分布式、图形处理器加速的环境中运行成熟的地球动力学模型，而无需具备并行编程或底层硬件优化方面的深厚专业知识。同时，Pyroclast 提供了灵活的架构，使研究人员能够以最少的工作量实现新物理。该框架不强迫用户从头重写求解器，而是鼓励代码复用和可扩展性，从而更便于在稳健的仿真环境中探索新颖的物理机制。模块化是其关键设计原则；通过遵循严格的面向对象设计，Pyroclast 提高了可维护性，并允许领域科学家通过高级 Python 脚本编程与系统交互。繁重的数值计算负载将交由使用专用工具实现和编译的优化库与 GPU 内核处理。该框架并不旨在达到底层 C++或 Fortran 求解器的绝对峰值性能，而是提供一种实用的折中方案，使非专业用户能够发挥大部分性能潜力，同时享受更简单、更灵活的开发体验。通过这种方式，Pyroclast 降低了使用门槛，并为研究提供了一个实用且可扩展的平台。

1.5 目标与贡献

该项目的第一个长期目标是将 I2ELVIS 和 I3ELVIS 系列地球动力学模型移植到现代化、模块化的 Python 框架中。这些模型已广泛用于地球动力学中的热力学模拟。它们将标记点网格法与 Stokes 方程和能量方程的有限差分离散相结合，并实现了黏弹塑性流变模型 [17, 18, 19]。

鉴于本项目的时间限制，本文着重实现一个核心组件，为最终支持完整的 I2ELVIS 和 I3ELVIS 模型奠定基础。具体而言，本文开发了一个二维黏性力学模型，作为更复杂的黏弹塑性模拟的计算核心。这并非对问题的简化，而是必不可少的基础步骤。得益于 Pyroclast 的模块化设计，此处开发的组件将被完整复用于未来模型的力学核心。当前实现包含求解完整热力学问题所需的数值方法，其中最重要的是斯托克斯方程的解。该组件为支持更复杂模型所需的方法和算法奠定了基础。本文有意聚焦于力学系统：求解斯托克斯方程尤其具有挑战性，因为离散化会产生规模庞大、稀疏且通常病态程度很高的鞍点系统 [20]。高效求解这类问题需要采用基于几何多重网格法的专门方法 [21]。

本文介绍了构成 Pyroclast 基础的设计、算法和实现策略。所有求解器组件均采用无矩阵方法实现，并全面支持图形处理器，以最大限度地提升性能和可扩展性。此外，本文引入了基于 MPI 的区域分解，以支持分布式内存并行，包括用于基于标记粒子的平流方程的高效并行求解器，以及用于斯托克斯系统的分布式多重网格策略。本文为未来的开发提供了蓝图，也可作为技术参考，助力将该框架扩展至完整的黏弹塑性模型及其他领域。

1.6 名称的由来

这个名称*Pyroclast*的灵感来自火山碎屑岩：这类碎屑岩由碎片（称为火山碎屑）组成，这些碎片是在爆炸性火山喷发期间产生并喷出的。这些岩石通常形成于剧烈的火山过程，体现了地球内部动态而复杂的行为，而 Pyroclast 旨在通过计算对其进行建模。该词本身源自希腊语 *pyr* （意为“火”）和*klastos*（意为“破碎”），暗指高能量的破碎现象。这个名称也兼有

双关之意，采用了 Python 库通常使用的 Py 前缀，既表明了它所基于的编程语言，也体现了其科学领域。

1.7 论文大纲

全文分为九章，按照 Pyroclast 框架在概念和技术层面的发展脉络展开。

第1章介绍现代化地球动力学求解器的动机，概述本研究所应对的关键科学与技术挑战，并总结项目的主要目标和贡献。本章还解释了名称*Pyroclast*的由来，并以本大纲作结。

第2章回顾计算地球动力学领域的相关工作，包括用于长期岩石圈动力学和地震波传播的现有代码。本章指出传统模拟框架的局限性，并讨论 Python 生态系统的优势、领域特定语言的最新进展，以及机器学习在科学计算中日益重要的作用。本章还阐明 Pyroclast 在这一不断发展的领域中的定位。

第3章介绍模拟的物理和数学基础，涵盖质量守恒与动量守恒、平流方程、边界条件和无量纲化。

第4章介绍 Pyroclast 的数值实现，包括标记点网格法、插值格式、拉格朗日平流、交错网格离散化，以及不可压缩斯托克斯方程的无矩阵求解。本章还介绍多重网格求解器、各种平滑器和加速器，并讨论收敛行为与求解器鲁棒性。

第5章概述该框架的软件架构与模块化设计。本章阐述设计理念、求解器组件的组织方式、模拟工作流，以及如何运用高性能 Python 工具兼顾灵活性与效率。

第6章聚焦 GPU 支持与设备无关编程。本章介绍如何集成 CuPy 和 Numba 以实现 GPU 加速，讨论内存约束，并概述 GPU 上模板运算与标记点平流内核的设计。

第 7 章介绍使用 MPI 的分布式并行化策略，涵盖区域分解、MPI 通信器设置、分布式标记点交换，以及 RAS 预条件分布式多重网格方法的实现。

第8章展示共享内存和分布式执行的基准测试与验证结果，评估强扩展和弱扩展性能、收敛行为，以及相较于 CPU 基准的 GPU 加速效果，展现了该框架的效率和可扩展性。

最后，第9章总结了本研究的核心贡献，并讨论了 Pyroclast 的科学影响。本章还概述了未来的研究方向，包括支持三维模拟、热-力耦合过程、更先进的数值功能，以及与机器学习工作流集成以实现混合建模。

第2章

相关工作

在上一章中，我们指出，数值建模在地球动力学中发挥着核心作用；在这一领域，直接观测有限，而物理过程涉及广泛的空间和时间尺度。过去几十年来，人们开发了许多模拟框架来应对这一挑战，其中不少已成为该领域的基础工具。然而，这些代码往往采用底层编程语言和高性能计算范式构建。以当今的标准来看，它们在可扩展性、可维护性，以及与图形处理器加速和机器学习等新兴技术的兼容性方面存在局限。

本章概述了目前最广泛使用的地球动力学求解器，简要总结其数值方法、软件组件、硬件支持，以及在高性能计算系统上的已报告性能或扩展性结果。随后，本章考察科学软件现代化的近期趋势，尤其是气候和天气建模领域；在这些领域中，将性能与开发者生产力相结合的努力推动了模块化、基于 Python 的方法的采用。本章还探讨了机器学习在科学计算中日益重要的作用，并强调其加速科学发现、增强传统模拟工作流的潜力。这些发展凸显了对新一类模拟框架的需求：将传统的基于偏微分方程的数值模型与模块化设计相结合，并兼容高性能计算和人工智能生态系统。在这一不断演变的背景下，Pyroclast 应运而生。

2.1 现有地球动力学求解器

本节回顾了塑造当前地幔对流、岩石圈变形、地震波传播及相关大型地球物理过程研究的主要地球动力学模拟框架。

2.1.1 岩石圈长期动力学

过去几十年间，人们开发了许多高性能计算代码，用于模拟地幔对流、岩石圈变形和长期地球动力学演化。这些求解器通常采用有限差分法、有限体积法或有限元法求解热力学和力学方程，并且许多求解器支持通过消息传递接口（MPI）进行并行执行。部分求解器经过扩展或改造，以利用 GPU 加速，但目前应用仍有限。以下列出一些广泛使用的求解器及其主要特点。

ASPECT

ASPECT 是由地球动力学计算基础设施（CIG）开发的有限元代码，专为二维和三维地幔对流及行星热演化问题设计。ASPECT 支持自适应网格细化（AMR）、非线性流变和可压缩性等高级功能。它具有很强的可扩展性，并已通过消息传递接口（MPI）在数千个 CPU核心上进行了测试。不过，标准版本目前尚不支持 GPU 加速 [22, 23, 24, 25]。

CitcomS

CitcomS 是另一款由 CIG 支持的有限元代码，专注于三维地幔对流，尤其是球壳几何中的地幔对流。该代码支持通过消息传递接口（MPI）进行区域分解，并已在高性能计算环境中进行测试。尽管已有实验性的 CUDA 移植版本，CitcomS 仍主要是一款基于中央处理器的 MPI 代码 [26, 27, 28, 29, 30]。

I2ELVIS/I3ELVIS

I2ELVIS 及其三维扩展版 I3ELVIS 是基于标记点网格法的有限差分求解器。这些模型实现了黏弹塑性流变，并支持自引力和自由表面演化等其他物理过程。其数值策略将守恒型交错网格有限差分与完全非线性流变学公式相结合，并在每个时间步通过迭代求解。I2/I3ELVIS 代码目前不支持消息传递接口（MPI）或 GPU 加速，这限制了它们在现代并行架构上的可扩展性 [17, 18]。

GAIA

GAIA 是德国航空航天中心开发的一款有限体积代码，用于模拟行星地幔的热化学演化。它支持强变黏度条件下的蠕动流，并对热化学对流进行建模。GAIA 专为分布式内存并行而设计，并包含支持图形处理器的

求解器，采用 CUDA 和混合 CPU-GPU 策略，使其成为少数原生支持图形处理器的地球动力学代码之一 [31, 32, 33, 34]。

LaMEM

LaMEM（岩石圈与地幔演化模型）是美国茨大学开发的一种有限差分代码，旨在用于具有黏弹塑性流变特征的三维岩石圈—地幔模拟。它采用交错网格，并支持标记点网格技术。LaMEM 基于 PETSc 构建，可扩展至数十万个 CPU核心。近期研究探索了通过 Julia绑定实现 GPU 加速，但这些开发仍处于实验阶段，尚未纳入主版本发布。

pTatin3D

pTatin3D 是一种地球动力学建模软件包，专为长期岩石圈变形问题而设计，例如俯冲和大陆裂谷作用。它结合物质点法来追踪材料组合，并对不可压缩黏塑性斯托克斯方程进行有限元离散。该求解器采用无矩阵几何多重网格预条件器，可显著降低内存带宽需求，并提升在现代架构上的可扩展性。性能研究表明，与基于矩阵的方法相比，其加速比超过 2× ，并且在 Cray XC-30 等系统上实现了持续的弱扩展性。pTatin3D 支持基于 MPI 的并行性，并可能通过其 PETSc后端 [35] 获益于 GPU 加速。

StagYY

StagYY 是苏黎世联邦理工学院开发的有限体积代码，用于三维球壳对流，涵盖可压缩性、相变和热化学效应。虽然它不是开源软件，但在地幔动力学领域得到了广泛应用。它支持消息传递接口并行，目前正借助基于 PETSc [36, 37] 的 StagBL 抽象层进行现代化改造。

Underworld2

Underworld2 是墨尔本大学开发的粒子网格有限元代码，用于模拟岩石圈和地幔变形，包括黏弹塑性流变关系。它提供 Python 接口用于问题设置，同时依托基于 PETSc 的 C语言求解器后端。Underworld2 在基于 CPU 的 MPI 环境中具有良好的扩展性，但不包含原生 GPU 支持。基于 PETSc 的组件可能受益于外部 GPU加速库，但这并非该代码的重点 [38, 39, 40, 41, 42]。

尽管这些求解器代表了计算地球动力学发展历程中的重要里程碑，但仍存在若干常见局限。它们大多使用 Fortran、C 或 C++ 等低级语言实现，因此难以扩展，也难以与现代人工智能/机器学习软件生态系统对接。即使具备 GPU 支持，通常也仅限于特定模块，或通过 PETSc [43, 44, 45, 46] 或其他外部库间接实现，而非采用原生 GPU 实现。因此，这些框架中很少有能够充分利用当前异构高性能计算架构的。此外，单体式设计和复杂的构建系统也给模块化和快速试验带来挑战，降低了希望修改或扩展模型的研究人员的使用便利性。在所评述的代码中，LaMEM 在数值方法和物理建模方法上都与 I3ELVIS 最为相似，因此也与 Pyroclast 的目标高度一致。与 Pyroclast 一样，它采用有限差分离散与标记点网格法相结合的方法。

2.1.2 地震波传播

除了长期地球动力学和地幔对流代码外，地球物理学界还开发了许多高性能模拟框架，用于解决其他类型的物理问题。值得注意的是，许多求解器专注于地震波传播、动态地震破裂和地壳变形。这些代码通常求解双曲型方程或弹性动力学方程，并针对高分辨率、短时间尺度的模拟进行了优化。虽然它们处理的是不同类型的物理过程，但面临许多相同的计算挑战，包括实现可扩展并行计算以及高效利用加速器硬件的需求。下面简要介绍该领域一些应用最广泛的工具。

SeisSol

SeisSol 是一种用于地震波传播和动态破裂的高阶间断伽辽金代码。它专为模拟复杂的三维地震情景而设计，同时支持中央处理器和 GPU 加速。该代码采用消息传递接口（MPI）与 OpenMP 混合并行化，并已在大型高性能计算系统上展现出良好的强扩展能力 [47, 48, 49, 50]。

SPECFEM3D

SPECFEM3D 是用于全球及区域三维地震波传播的谱元代码，广泛应用于地震地面运动模拟和地震层析成像。该求解器通过消息传递接口支持分布式内存并行，并提供基于 CUDA 和 HIP 后端的 GPU 加速 [51, 52, 53]。

AWP-ODC

AWP-ODC 是用于非弹性波传播和地震破裂模拟的有限差分代码。它针对大规模地面运动建模进行了优化，并已在超级计算机上展现出卓越的可扩展性。该代码的 GPU 版本在关键内核上实现了显著的加速比，从而支持高频模拟 [54, 55]。

SW4

SW4（四阶地震波）是一种用于区域三维地震波传播的有限差分求解器，支持复杂地形和材料非均质性。它支持基于 MPI 的并行，并已使用 RAJA [56, 57] 性能可移植性框架移植到 GPU，从而在 GPU平台 [58, 59, 60]上实现了显著的加速比。

PyLith

PyLith 是一种用于准静态和动态地壳变形的有限元求解器，专为地震周期建模而开发。它构建于 PETSc 之上，并支持 MPI 并行。虽然 PyLith 目前不支持 GPU 加速，但其对 PETSc 的使用可能使未来集成支持 GPU 的求解器成为可能 [61]。

用于地震波传播和动态破裂的高性能求解器的发展，推动了地震建模的重大进步。这些代码通常针对特定使用场景进行了高度优化，并已在现代高性能计算系统上展现出卓越的可扩展性。如今，许多求解器已支持 GPU 加速，或通过原生实现，或借助性能可移植性框架实现。尽管它们所针对的物理系统不同于长期地球动力学模型，但面临许多相同的计算挑战，包括高效并行化、灵活的求解器设计，以及与不断演进的硬件架构集成的需求。结合前文讨论的地幔对流求解器，它们共同展现了地球物理模拟工具的整体图景，并有助于阐明 Pyroclast 背后的设计目标。

2.2 传统模拟框架的局限性

尽管传统模拟框架为地球动力学研究做出了重大贡献，但许多框架仍存在结构和架构方面的局限，阻碍了其开发和更广泛的应用。大多数现有代码都是为以CPU为中心的硬件架构设计的，并使用 C、C++ 或 Fortran 等底层编程语言实现。它们的结构通常是单体式的，各组件紧密耦合，因而难以

在缺乏对底层物理和技术实现的深入了解时扩展或修改软件。扩展这些求解器或引入新的物理模型，往往需要具备多个领域的专业知识：地球物理学家不仅必须理解控制方程和数值方法，还需要具备工程背景，以处理底层编程、性能调优，以及在异构架构上的并行化。这包括缓存优化、共享内存与分布式内存模型、消息传递接口（MPI）通信和 GPU 编程等方面的知识。很少有研究人员同时具备这些技能，这极大地限制了现有工具在更广泛科学界中的可及性。

这种额外的复杂性也使许多求解器难以采用现代硬件和软件技术。虽然如今已有一些地震和波传播代码支持图形处理器，但上一节介绍的长期岩石圈动力学模型中，只有少数实现了稳定且完全集成的图形处理器加速。这可能是因为标记法的实现难度更高，需要同时处理粒子和网格运算。与纯粹基于网格的方法相比，这使得它们更难在图形处理器上加速，因此图形处理器支持往往并不完善、仍处于实验阶段，或仅限于特定组件。

许多现有代码依赖 PETSc [43, 44, 45, 46] 实现分布式并行，在某些情况下还可实现部分图形处理器加速。PETSc 是一个稳健且广泛使用的库，但它仍是一个大型单体式 C 语言框架，诞生于科学计算中的 GPU 编程兴起之前，其首个稳定版本于1996 [62]发布。PETSc 中的图形处理器加速仅适用于特定运算，且仍在开发中，主要面向代数求解器和 BLAS 例程 [63]。无矩阵几何多重网格求解器所使用的模板运算等自定义内核，仍需使用 CUDA 或类似的低级语言手动实现。另一种选择是，开发者必须依赖 Kokkos [64, 65]或 RAJA [56, 57],等性能可移植性框架，或者采用 OpenACC [66]等基于指令的方法。这些模型通常使用 C 或 C++ 实现，要有效使用它们，需要具备丰富的专业技术知识。

2.3 Python 生态系统的优势

相比之下，受人工智能和机器学习需求影响的现代 Python 生态系统，将 GPU 计算视为一等要务。围绕 ndarray 或张量抽象构建的基于数组的编程模型，能够自然地将数学运算卸载到 GPU，并自动——

选择 cuBLAS [67], cuTensor [68],或相应的 hipBLAS [69]和 hipTensor [70]等经过优化的供应商库作为目标。凭借这种设计，同一段高级代码可根据可用的硬件环境，在中央处理器或 GPU 上高效运行。

对于模板内核等难以用张量运算轻松表达的自定义操作，Python 提供了多个即时编译（JIT）框架，允许用户直接用 Python 编写高性能的 CPU 和 GPU 内核。Numba [71], Numba-CUDA [72],Numba-HIP [73], 和 DaCe [74] 等项目，无需离开 Python 环境即可生成经过优化的机器码，其性能可媲美手动编写的 CUDA 或 HIP 内核。在许多情况下，同一个内核只需很少修改甚至无需修改，即可编译到多个硬件后端。

当需要直接与低级语言交互时，无论是为了访问专用外部库，还是为了实现高度优化的内核，都可以使用 nanobind [75]等工具轻松编写绑定。这些绑定可与 Python ndarray 和张量对象无缝集成，并支持与 NumPy [76], CuPy [77], 和 PyTorch [15]等库进行零拷贝数据共享。

Python 生态系统的优势远不止于 GPU 编程和性能可移植性。Python 可访问大量成熟的数据分析、可视化和数值计算库，这些库能够原生协同工作。得益于近年来兴起的人工智能驱动的软件基础设施，基于 Python 的工具也能够轻松部署。Apptainer [78,79]和 Enroot [80]等容器技术简化了在高性能计算系统上的部署，并提供了对经过优化的 GPU 就绪环境的访问，包括 NVIDIA PyTorch 容器 [81]等预构建镜像。这些环境包含厂商优化的数学库、GPU 感知型 MPI，以及 NCCL [82] 和 NVSHMEM [83], 等高级通信层，从而简化了在分布式 GPU 集群上的扩展。

除了性能和互操作性之外，Python 简洁而富有表现力的语法也在提升开发者生产力方面发挥着关键作用。这种语言能够以清晰紧凑的方式编写复杂的数值运算，其形式往往与底层数学表达式相近。这减少了所需的样板代码，也让实现更易于阅读、修改和维护。动态类型、自动内存管理和高级数据结构等特性，则通过消除底层编程带来的许多额外负担，进一步简化了开发。在这方面，Python 与 Julia 有许多相同的优势：它提供高级界面，让研究人员能够专注于模型的科学和算法层面，而不是实现的技术细节——

此外，Python 对面向对象编程的强大支持，使构建模块化、可扩展的模拟框架更加容易。这有助于缩短开发周期、改善代码组织，并提升长期可维护性。

需要注意的是，Python 并非没有取舍。作为一种解释型语言，与 C 或 Fortran 等编译型语言相比，它会带来运行时开销；结构不佳的代码也可能造成严重的性能瓶颈。不过，如果将性能关键组件交由优化库处理，或使用即时编译，这些局限大多可以得到缓解。经过精心设计，基于 Python 的模拟能够达到与底层实现相当的性能，同时保留更高的抽象层级和开发者生产力。近期大型科学计算项目的成果表明，只要设计得当，基于 Python 的框架即使在高性能计算领域的最高端，也能达到与传统底层实现相当的性能。例如，近期一项大规模量子输运模拟研究使用基于 Python、由 NumPy、CuPy、mpi4py、Numba 和 SciPy 构建的框架，实现了持续的百亿亿次级性能。研究团队在超过 37,000 个 GPU 上运行模拟，达到 1.15 Eflop/s，弱扩展效率为 82%；该项目还获提名角逐 2025 年 ACM 戈登·贝尔奖 [84, 85]。这一成果表明，借助合适的工具并经过合理设计，Python 也能扩展至极高的性能水平。

总的来说，传统框架数十年来一直服务于地球动力学界，但它们依赖低级语言、采用单体式设计，且对异构计算的支持有限，因此越来越难以扩展和维护。相比之下，Python 生态系统提供了一个灵活、模块化且能感知硬件的平台，自然而然地契合了高性能计算和人工智能加速计算的当前发展方向。采用此类现代软件范式，是弥合地球动力学领域中性能、易用性与科学创新之间差距的必要一步。

2.4 科学计算 Python 中的领域特定语言

近年来，出现了多个项目，旨在将性能优化的复杂性从领域科学家身上抽象出来，尤其面向 Python 生态系统。这些工作致力于让人们更轻松地编写高性能模拟代码，而无需具备底层编程、内存层次结构或针对特定硬件的调优方面的专业知识。其中最突出的例子之一是 ICON—— [86], 全球气候与天气预报模型；目前，ICON4Py 计划正推动其从 Fortran 移植到 Python [12]。

该项目的核心组成部分是 GT4Py—— [87], 一种直接嵌入 Python 的模板计算领域专用语言（DSL）。GT4Py 允许科学家使用高级、与后端无关的语法来表达模板运算，并将其编译为面向 CPU、图形处理器或 FPGA 的优化底层内核。它支持多个后端，包括原生内核生成后端，以及基于 DaCe 数据中心编译器的后端 [74]。尽管 GT4Py 主要为 ICON 开发，但它是开源的，可用于任何涉及基于模板的计算的科学应用。

另一个例子是 Devito [88, 89],，这是一个嵌入 Python、用于求解偏微分方程的领域专用语言。Devito 采用更高层级的方法：用户无需手动编写模板代码，而是使用 SymPy [90] 包以符号形式定义物理模型。Devito 根据这些符号表达式自动推导离散化，并生成经过优化的低层模板内核。编译器支持多种高级优化，尤为重要的是，近期的发展还实现了 MPI 并行代码的自动生成，使 Devito 能够在分布式内存系统 [91]上扩展。该框架已成功应用于波传播和地震反演等领域；与传统模拟代码相比，它展现了极高的抽象层级。

2.5 科学计算中的机器学习

近年来，机器学习已成为科学计算中的强大工具，其应用远不止数据分析或分类等传统任务。如今，机器学习正被积极用于加速数值求解器、构建代理模型，并解决众多科学领域中的逆问题。科学机器学习这一不断发展的领域涵盖了种类繁多的模型和技术，远非本文所能全面评述。因此，我们将重点介绍几种具有代表性的方法，以说明机器学习在现代基于物理的模拟工作流中日益重要的作用。

一类重要的模型是物理信息神经网络（PINN），它们通过将偏微分方程残差嵌入损失函数 [92, 93]，将微分方程约束直接纳入训练目标。这使模型无需标注数据即可近似求解偏微分方程。PINN 已成功应用于流体力学、天气与气候建模以及地球物理学等问题 [94, 95]。

神经算子采用更通用的方法，学习偏微分方程的输入场与解场等函数空间之间的映射。傅里叶神经算子（FNO）等架构在傅里叶空间中学习这些映射，并已用于模拟湍流及其他复杂的流体动力学问题 [96]。球面傅里叶神经算子等扩展模型已被

用于预测全球大气动力学 [97]。其他变体，包括卷积神经算子（CNO），采用基于卷积神经网络的架构，并已在类似场景中取得成功 [98]。这些模型可实现快速的代理模型训练，尤其是快速推理，非常适合通过数据驱动方法学习未知物理。

与之密切相关的是可微物理方法，它们利用自动微分实现模拟流水线，从而支持通过物理模型进行端到端优化。例如，这使得人们能够通过对模拟步骤进行反向传播来进行逆向材料设计 [99], ，或通过可微牛顿求解器进行优化来学习控制策略 [100]。

近期，针对偏微分方程大型基础模型的研究表明，基于变换器的架构能够泛化到不同类型的方程和不同区域。Poseidon [101], 是一种在多样化偏微分方程数据集上训练的多尺度算子变换器，在多种下游任务中展现出优异表现，胜过领域专用模型，并展现了通用偏微分方程求解器的发展前景。另一个近期的大规模基础模型是 Aurora [102]。Aurora 是一种地球系统变换器模型，在超过一百万小时的异构地球物理数据上训练，涵盖大气、海洋和气候过程。在空气质量预测、海浪建模、热带气旋追踪和高分辨率天气预报等任务中，其表现已超越若干业务预报系统，同时计算成本低了数个数量级。Aurora 还可以用相对有限的计算资源针对专门应用进行微调，这使其成为迈向易用、高效、统一的地球系统预测的重要一步。它的成功凸显了机器学习在大规模环境建模中的变革潜力，也进一步说明了亟需能够与人工智能驱动的工具无缝对接的模拟框架。

总而言之，这些示例表明，机器学习正逐渐成为科学计算的核心组成部分，能够提供强大的工具来加速模拟、推断隐藏的、未知的或难以处理的物理问题，并解决通常仅靠传统数值方法无法触及的问题。

2.6 Pyroclast 的定位

地球动力学模拟工具的发展历程深受数十年高性能计算发展的影响。正如前文所述，大多数成熟的代码都依赖于以低级语言编写的单体架构。尽管这些工具推动了科学研究取得重大进展，但其结构往往限制了可扩展性，增加了维护难度，

并阻碍了图形处理器加速和机器学习等新兴技术的应用。这导致现代硬件和软件能力与领域科学家能够轻松使用的技术之间的差距日益扩大。

Pyroclast 旨在弥合这一间隙。它将现代、可扩展的高性能计算求解器的特性——包括分布式内存并行和 GPU 加速——与完全使用 Python 编写的简洁模块化架构结合起来。与将 Python 视为底层代码薄封装的框架不同，Pyroclast 将 Python 用作主要开发环境。对性能至关重要的操作会交由即时编译内核、向量化后端或 GPU 库执行，从而在不牺牲简洁性和开发者生产力的前提下，兼顾速度与可移植性。

与用于长期岩石圈动力学模拟的传统求解器不同，后者要求使用者深入了解 C/Fortran 或 PETSc 的内部机制才能进行扩展，Pyroclast 提供了简洁易用的界面，使科学家能够以极低的技术门槛实现或调整模型。其面向对象的结构鼓励模块化与复用，因此可以轻松地独立扩展或替换各个组件。这对于开发结合机理模拟与机器学习的混合模型尤其有价值；这一趋势正日益盛行于计算科学领域。

Pyroclast 还与 Python 科学计算生态系统紧密集成，其中包括用于数据分析、可视化和深度学习的库。这使得模拟工作流能够无缝部署于 AI 加速环境，并支持直接对接可微求解器、神经代理模型以及面向物理系统的基础模型。

通过顺应科学计算领域更广泛的现代化趋势，Pyroclast 开创了一类新的地球动力学模拟框架：它既具备高性能，又易于使用；既模块化，又稳健；并且原生兼容高性能计算和 AI 生态系统。它在继承以往求解器优势的同时，消除了阻碍创新与实验的关键障碍。

2.7 本文的工作范围

如第 1 章所述，本项目的长期目标是将 I2ELVIS/I3ELVIS 系列地球动力学模型移植到现代、高性能的 Python 框架中。由于完整实现整个热力学模型超出了这篇首篇论文的范围，我们转而聚焦于一个必要且可行的第一步：构建一个分布式、由图形处理器加速的黏性力学求解器。

为使问题保持可处理，我们假设变形完全为黏性变形，且材料属性固定，并由速度场携带。实现——

目前仅限于二维，这简化了开发和可视化。不过，所有算法的设计都便于自然扩展至三维，届时更大的问题规模预计将带来更好的性能。Pyroclast 的模块化设计支持这一点，使各组件能够轻松复用于不同模型和维度。

开发仿真框架面临各种数学、工程和高性能计算方面的挑战。虽然无法详细涵盖所有方面，本文将重点介绍构成当前实现基础的关键方程、数值方法、算法和优化策略。

第3章

控制方程

本章概述了 Pyroclast 当前实现中使用的控制方程。该实现对与岩石圈和地幔动力学相关的不可压缩黏性变形进行建模。模型基于连续介质力学的基本原理，描述由黏性力主导的缓慢蠕动流。我们仅关注问题的力学部分，因为这是 I2ELVIS 和 I3ELVIS 中更复杂的黏弹塑性热力学模型的计算核心。

本章内容主要基于 [103]，提出的工作，该工作详细概述了数值地球动力学建模中使用的控制原理和流变模型。这里的目的并非引入新理论，而是总结 Pyroclast 当前实现所依据的基本方程和假设。这些方程在不可压缩黏性流动的假设下定义了质量与动量守恒，并为后续章节讨论的数值模型奠定基础。

3.1 物理假设

该力学模型描述了地球岩石圈和上地幔中缓慢的浮力驱动流，并采用了一系列简化的物理假设。这些近似解保留了系统的主导动力学特征，同时使连续介质表述易于处理。

蠕动流

该流动状态的特点是雷诺数极低，因此惯性力相较于黏性力可以忽略不计。因此，控制方程可简化为稳态斯托克斯方程。对于长期岩石圈变形而言，这一假设十分合理，因为

典型的板块速度约为 10^{-9} cm/s，并在约 10^{13} s的地质时间尺度上演变。由此产生的加速度约为10^{-22} m/s^2, ，比重力小许多个数量级（$g \approx 10$ m/s^2）。因此，动量平衡体现了黏性力、压力和引力之间的准静态平衡。

不可压缩性

假设材料不可压缩，即体积变化可以忽略不计。因此，速度场 $v = (v_x, v_y)$ 满足无散条件：

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} = 0. \tag{3.1}$$

黏性流变学

材料表现为线性牛顿黏性流体。偏应力张量 σ'_{ij} 与应变率张量 $\dot{\varepsilon}_{ij}$ 之间的关系为

$$\sigma'_{ij} = 2\eta\dot{\varepsilon}_{ij} = \eta\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right), \tag{3.2}$$

其中，η 表示动力黏度。黏度和密度均视为空间可变的量，会随流动一起平流，但在一个时间步内不会发生内部演化。

应力分解

柯西应力张量 σ_{ij} 通常分解为各向同性压力分量和偏应力分量：

$$\sigma_{ij} = -p\,\delta_{ij} + \sigma'_{ij}, \tag{3.3}$$

其中，p 表示平均正应力（压力），δ_{ij} 为克罗内克δ。偏应力部分 σ'_{ij} 包含剪切变形的信息，而 p 则体现体积效应。对于不可压缩流动，$\nabla \cdot v = 0$意味着体积应变为零，因此压力项仅作为拉格朗日乘子来强制满足不可压缩性。因此，控制方程可以直接用 σ'_{ij} 和 p表示。

3.2 质量和动量守恒

在这些假设下，该系统由二维不可压缩斯托克斯方程支配，包括x方向动量方程、y方向动量方程以及不可压缩性条件。柯西

形式的斯托克斯问题将动量平衡表示为偏应力和压力的形式，如下所示：

$$\frac{\partial \sigma'_{xx}}{\partial x} + \frac{\partial \sigma'_{xy}}{\partial y} - \frac{\partial p}{\partial x} = 0, \tag{3.4}$$

$$\frac{\partial \sigma'_{yx}}{\partial x} + \frac{\partial \sigma'_{yy}}{\partial y} - \frac{\partial p}{\partial y} = \rho g_y, \tag{3.5}$$

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} = 0. \tag{3.6}$$

方程（3.4）和（3.5）分别表示x 和y方向上的动量平衡，而方程（3.6）则保证不可压缩性。本构关系（3.2）通过黏度将偏应力与速度场联系起来。

上述方程中出现的物理量如下：

- σ'_{ij}：作用于i平面、沿j方向的偏应力
- p：压力
- v_x,v_y：x 和y 方向上的速度分量
- ρ：密度
- g_y：y方向上的重力加速度
- η：动力黏度
- $\dot{\varepsilon}_{ij}$：作用于i平面、沿j方向的应变率

方程 (3.4)–(3.6) 与本构定律 (3.2) 共同定义了 Pyroclast 中使用的不可压缩 Stokes 问题。

3.3 平流方程

除了质量与动量守恒外，黏度和密度等材料属性的演化也受平流方程支配。这些量由速度场输运，但在单个时间步内不会扩散，也不会通过内部进程发生演化。

在 Pyroclast 中，平流是从拉格朗日观点处理的。使用一组离散粒子（通常称为标记点）来表示物质域。每个标记点都携带物理属性，例如 η orρ，这些属性会根据局部速度场 $v = (v_x, v_y)$进行平流。标量属性 φ 的平流遵循

$$\frac{D\varphi}{Dt} = \frac{\partial \varphi}{\partial t} + v_x \frac{\partial \varphi}{\partial x} + v_y \frac{\partial \varphi}{\partial y} = 0, \tag{3.7}$$

其中，D/Dt 表示物质导数。该方程表示，量 φ 沿着物质微团的轨迹保持不变。
从拉格朗日观点来看，我们沿着由特征常微分方程定义的物质轨迹 $\mathbf{x}(t) = (x(t),y(t))$

$$\frac{d\mathbf{x}}{dt} = v(\mathbf{x}(t),t)\,, \qquad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{3.8}$$

其中，$\mathbf{x}_0$ 表示时间 t_0时物质微团的初始位置。沿任意一条这样的轨迹，

$$\frac{d}{dt}\,\varphi(\mathbf{x}(t),t) = \frac{D\varphi}{Dt} = 0 \quad \Longrightarrow \quad \varphi(\mathbf{x}(t),t) = \varphi_0(\mathbf{x}_0), \tag{3.9}$$

这意味着 φ 在粒子路径上保持不变。因此，方程 (3.7) 的解可通过以下步骤获得：(i) 对轨迹方程 (3.8) 进行积分；(ii) 将每个物质微团的初始属性赋予该微团。

使用拉格朗日标记点可以准确追踪材料界面和显著的属性对比，同时最大限度地减少数值扩散。然而，力学方程是在固定的欧拉网格上求解的。通过在求解斯托克斯方程之前将标记点属性插值到网格上，并将由此得到的速度场映射回标记点以进行平流，从而实现拉格朗日描述与欧拉描述之间的耦合。

插值和时间积分所采用的具体数值格式，以及这些格式在 Pyroclast 中的实现，将在后续章节中详细介绍。

3.4 边界条件

系统必须补充适当的边界条件，以约束区域边界处的速度场。这些条件决定材料在计算区域边缘如何移动或变形，并在确定模型的物理行为方面发挥重要作用。

常见的机械边界条件类型包括 [104]:

- **自由滑移边界**：法向速度分量为零，且沿边界的剪切应力为零。用速度导数表示为

$$v_n = 0, \qquad \frac{\partial v_t}{\partial n} = 0, \tag{3.10}$$

其中，v_n 和v_t 分别表示法向和切向速度分量，而n 表示沿垂直于边界的方向求导。

- **无滑移边界**：法向和切向速度分量均设为零：

$$v_n = 0, \qquad v_t = 0. \tag{3.11}$$

- **周期边界**：速度场和压力场在相对的边界上呈周期性，从而确保速度和通量的连续性：

$$v_i(x_{\min}) = v_i(x_{\max}), \qquad p(x_{\min}) = p(x_{\max}). \tag{3.12}$$

- **指定速度（狄利克雷）边界**：可以施加v_i 的特定值，或指定法向速度梯度 $\partial v_i/\partial n$ ，以驱动流动或近似开放边界。

这并非全面概述，但关键在于，边界条件是斯托克斯问题的重要组成部分，必须特别谨慎地处理，尤其是在区域分布于多个进程的情况下。在并行环境中，每个子域都必须正确表示其本地边界以及与相邻区域共享的界面。确保这些边界上的一致性和连续性，对于数值稳定性和物理精度都至关重要。第 7 章将详细讨论 Pyroclast 所采用的 MPI 域分解策略中边界条件的实现。

3.5 无量纲化与单位缩放

Pyroclast 目前不支持对斯托克斯方程进行自动无量纲化。在有限的测试中，求解方程的无量纲形式并未显著提高收敛性或数值稳定性。不过，通过对模型的物理参数进行单位缩放，也能获得相同的数值优势。这种方法保留所有量的量纲，同时确保数值的大小保持均衡。

单位缩放的思路是，以一组参考值为基准对系统中的所有物理量进行重新缩放，这些参考值的选取旨在体现问题中的典型量级。这样可以在不改变物理量纲的前提下有效地对方程进行归一化，从而改善数值条件性，同时仍以物理单位表示结果。

该过程可概括如下：

1. **选择参考值**。 为问题中涉及的每个独立物理维度选择一个参考值。对于斯托克斯系统，这些维度分别是时间 T、长度 L和质量 M。这些值用作

系统的缩放基准，并定义每个基本维度的“单位”是什么。例如：

$$t_0 = 1\ \mathrm{s}, \qquad x_0 = 100\ \mathrm{km} = 10^5\ \mathrm{m}, \qquad \eta_0 = 10^{18}\ \mathrm{Pa\,s}.$$

前两个值设定了问题的时间和空间尺度，而 η_0 则定义了特征黏度（因此间接确定了质量尺度）。

2. **用基本参考维度表示参考值**。每个参考值都可以写成时间、长度和质量这三个基本维度的组合。例如，黏度的物理量纲为 $[\eta] = M/(LT)$。这一步可确保所有选定的参考量在量纲上保持一致：

$$t_0 = T_0, \qquad x_0 = L_0, \qquad \eta_0 = \frac{M_0}{L_0\, T_0}.$$

3. **求解有量纲缩放系数**。将已知参考值代入上述量纲关系，即可求得参考系数 T_0, L_0,和M_0。

$$T_0 = t_0, \qquad L_0 = x_0, \qquad M_0 = \eta_0\, x_0\, t_0.$$

在本例中，质量尺度 M_0 可直接根据黏度的定义得出，因为 η_0 已包含 M、L和T。

4. **将所有其他量表示为相对于这些系数的值**。定义好参考尺度后，任何物理变量都可以表示为无量纲量 $\tilde{}q$ 乘以其参考尺度 q_0。

$$\tilde{\eta} = \frac{\eta}{\eta_0}, \qquad \tilde{x} = \frac{x}{x_0}, \qquad \tilde{t} = \frac{t}{t_0},$$
$$\rho_0 = \frac{M_0}{L_0^3} \implies \tilde{\rho} = \frac{\rho}{\rho_0}, \qquad g_0 = \frac{L_0}{T_0^2} \implies \tilde{g} = \frac{g}{g_0}.$$

这些关系定义了斯托克斯系统中所有其余物理量的缩放方式。密度、重力、速度和压力都可以类似地推导得出。

从数值角度来看，该过程实现了与完全无量纲化相同的效果：它会重新缩放方程中的所有项，使其量级相近，从而改善数值条件性。两者的区别在于，该过程始终保留物理单位，因此更容易解读输入参数和结果。

第4章

数值方法

本章介绍构成 Pyroclast 求解器核心的数值算法。Pyroclast 基于标记点网格法（MIC），也称为物质点法；该方法结合固定的欧拉网格与拉格朗日标记点，用于表示并平流输运材料属性。本章先介绍 MIC 方法所采用的整体模拟流程，然后重点讲解 Pyroclast 中实现的具体数值组件，包括在交错网格上对控制方程进行有限差分离散、用于在标记点与网格之间交换信息的插值过程、用于材料输运的平流格式，以及用于求解变黏度不可压缩斯托克斯系统的迭代求解器。特别强调算子的无矩阵形式，以及利用几何多重网格方法高效进行速度平滑。有关并行化、GPU 加速和 MPI 支持的细节将在后续章节介绍。

4.1 标记点网格法

Pyroclast 基于标记点网格法（MIC），该方法将控制方程的欧拉离散与材料属性的拉格朗日表示相结合。控制偏微分方程在固定的直角网格上求解，而黏度、密度、温度等量则存储在随流动移动的拉格朗日标记点上。每次求解前，都会将标记值插值到网格上，再将得到的速度场插值回标记点，以更新其位置。

这种方法本质上是一种算子分裂形式，具体而言，是对平流和偏微分方程求解步骤进行李-特罗特分裂，从而得到一种一阶精度的时间积分格式 [105]。材料平流与控制方程的求解分开处理，例如不可压缩斯托克斯系统或能量方程。基于网格的求解使用从标记点插值得到的材料属性，并

利用由此得到的速度场独立更新标记物位置。原则上，可以采用二阶斯特朗分裂格式来提高时间精度 [105], ，但 Pyroclast 目前采用较简单的一阶变体。

典型的模拟循环按以下步骤进行：(1) 将标记点属性插值到网格上，以获得密度和黏度等节点场；(2) 在欧拉网格上求解离散化控制方程，计算解场，例如斯托克斯问题中的速度和压力；(3) 将速度和温度等相关场插值回标记点；(4) 使用显式时间积分格式将标记点向前平流。每个时间步都会重复这一循环，确保标记点属性与底层流场一致地演变。图4.1 概述了这一模拟流程。

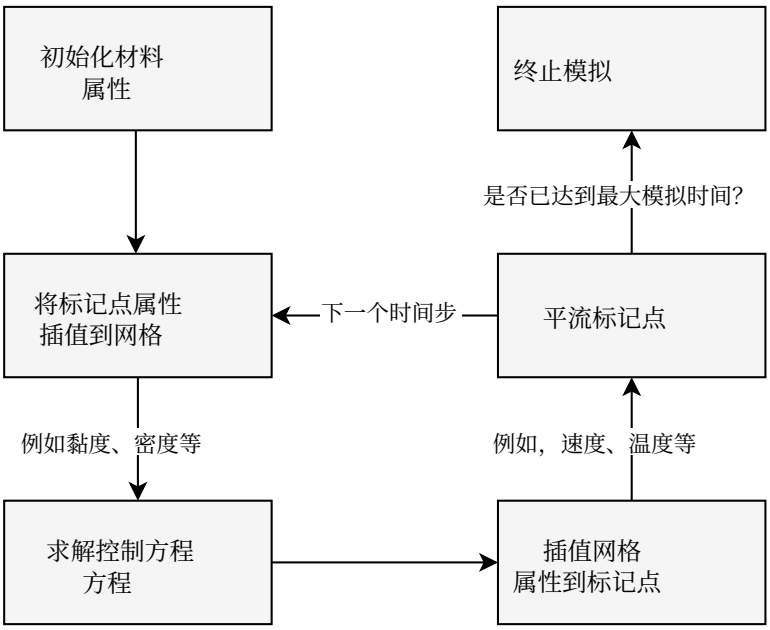

图4.1： 标记点网格法所用的高层次模拟循环。黏度和密度等材料属性在拉格朗日标记点上初始化，并插值到欧拉网格上。在网格上求解控制方程，然后将所得属性（例如速度、温度）插值回标记点。随后标记点被平流，循环持续进行，直至达到最大模拟时间。

MIC方法非常适用于地球动力学模拟，因为它能够以极小的数值扩散精确地平流复杂、非均质的材料分布 [106]。通过将材料属性存储在拉格朗日标记点上，该方法能够在较长的时间尺度内保持清晰的材料界面。将平流与控制方程的求解解耦，

也提高了模块化程度，使平流求解器和偏微分方程求解器能够独立实现和优化。这些特点使MIC成为模拟岩石圈和地幔动力学的有效方法，尤其适用于黏度、密度和组合存在强烈空间变化的情况。

4.2 标记点与网格之间的插值

图 4.2 展示了用于在标记点与网格节点之间传递物理量的插值方案。Pyroclast 使用加权线性插值核（二维时为双线性，三维时为三线性）来执行标记点到网格和网格到标记点的传递。设 φ 是存储在标记点上的标量场，例如密度、黏度或温度。

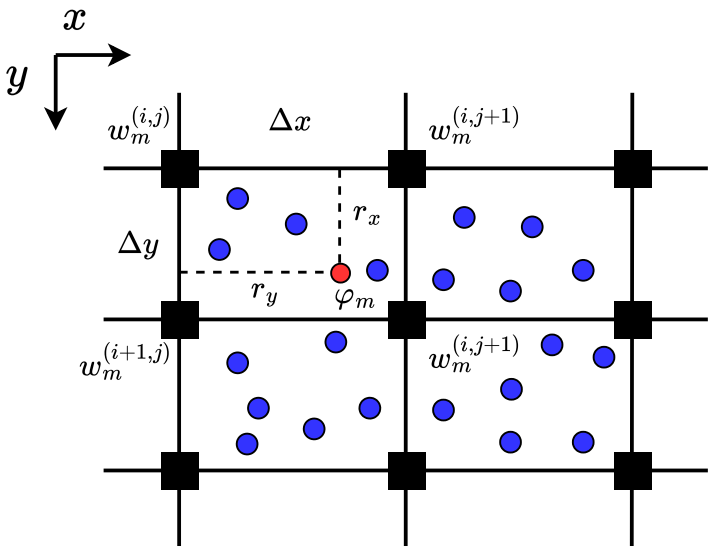

图 4.2： 二维空间中标记点（蓝色圆圈）与网格节点（黑色方块）之间线性插值的示意图。红色标记点位于一个由四个节点围成的网格单元内。使用距离 r_x 和 r_y 计算插值权重。

标记点到网格插值

将物理量从标记点传递到网格的步骤如下：

1. 对于每个标记点，确定其所在单元左上角的参考网格节点。 2. 计算标记点到参考节点的距离r_x 和r_y 。

3. 计算周围四个网格节点的插值权重：

$$w_m^{(i,j)} = (1 - r_x/\Delta x)(1 - r_y/\Delta y), \tag{4.1}$$

$$w_m^{(i+1,j)} = (r_x/\Delta x)(1 - r_y/\Delta y), \tag{4.2}$$

$$w_m^{(i,j+1)} = (1 - r_x/\Delta x)(r_y/\Delta y), \tag{4.3}$$

$$w_m^{(i+1,j+1)} = (r_x/\Delta x)(r_y/\Delta y). \tag{4.4}$$

4. 对每个相邻节点，将$w_m^{(i,j)}$ φ_m 和$w_m^{(i,j)}$累加到用于加权和与归一化的临时数组中。

5. 处理完所有标记点后，按加权平均值计算每个节点处的最终网格值：

$$\varphi^{(i,j)} = \frac{\sum_{m=1}^{n_m} w_m^{(i,j)} \varphi_m}{\sum_{m=1}^{n_m} w_m^{(i,j)}}, \tag{4.5}$$

其中，各项求和范围为所有对节点n_m (i,j)有贡献的标记点。

为避免累加过程中出现竞态条件，通常会根据目标架构，使用原子操作、散射加法原语或 MPI 并行来实现求和。尽管这种插值是标记点网格法和粒子网格法中的标准做法 [106], 但仍有必要在此详细介绍，因为它在分布式平流步骤的设计中发挥着核心作用。第 7 章将详细介绍边界和虚拟节点处理，其中涵盖 MPI 域分解和分布式标记点平流。

网格到标记点插值

将网格场插值回标记点：

1. 对于每个标记点，确定其周围的四个节点。2. 按照上文计算相同的双线性权重 $w^{(i,j)}$ 。3. 将场插值到标记点所在位置，如下所示：

$$\varphi_m = \sum_{(i,j)} w_m^{(i,j)} \varphi^{(i,j)} \tag{4.6}$$

其中，对 (i,j) 的求和涵盖了周围的四个节点。

由于权重之和自然为 1，因此无需归一化。此步骤的并行执行非常简单，因为每个标记点都可以独立处理：将其四个相邻节点上的值相加即可，且不存在竞态条件的风险。

4.3 拉格朗日标记点平流

Pyroclast 中材料属性的平流通过对由速度场定义的特征轨迹进行数值积分来实现。如第3章所述，位于$\mathbf{x}_0 = (x(t_0),y(t_0))$ 处的物质微团的运动由以下常微分方程支配

$$\frac{d\mathbf{x}}{dt} = v(\mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{4.7}$$

其中，$v = (v_x,v_y)$ 是通过求解不可压缩斯托克斯系统得到的速度场。由于 Pyroclast 采用算子分裂将平流与斯托克斯求解过程解耦，因此在每个平流步内，速度场都被视为不随时间变化。

模拟中的每个拉格朗日标记点都代表这样一团物质，并携带一组物理属性，例如黏度、密度和温度。这些量通常会随时间演变，原因可能是机械过程或热过程。然而，由于平流与其他控制方程的求解是分开进行的，在积分式 (4.7) 时，标记点属性保持不变。换言之，在平流步中，每个属性 φ 都被视为沿轨迹保持恒定，而材料本身的任何内在变化则在单独的更新步骤中处理。

4.3.1 时间步选择与稳定性

Pyroclast 使用自适应时间步进来控制每个平流步的大小。这对于多物理场模拟至关重要，因为模拟过程中相关时间尺度的范围可能会显著变化。实际上，无法选取一个固定的全局时间步来满足所有物理和数值约束。

两类主要准则用于限制平流时间步。第一类是基于物理的约束，用于确保物理过程得到准确解析。这类约束包括单个时间步内粒子位移或温度变化的最大允许值。第二类是基于网格的约束，源于数值稳定性要求和实现细节。例如，在 MPI 并行模拟中，必须限制平流步，使标记点保持在本地幽灵区域内，从而避免中间通信。这两类约束都会在运行时进行评估，并基于类 CFL 条件计算最大允许步长。

4.3.2 平流方程积分格式

为满足不同的精度和性能要求，Pyroclast 支持三种用于标记点平流的积分格式：(i)

前向欧拉法、(ii) 显式龙格-库塔方法，以及 (iii) 一种基于标记点周围速度场泰勒展开的局部多项式积分格式。这些方法在计算成本、精度和实现复杂度方面各不相同，尤其是在并行环境中。以下各节将详细介绍每种方法。

欧拉积分

Pyroclast 中最简单的积分格式是显式前向欧拉法。该方法可对轨迹方程 (4.7) 给出一阶精度的解，其形式为

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t\, v(\mathbf{x}^n), \tag{4.8}$$

其中，$\mathbf{x}^n$ 表示时间步开始时的标记点位置，Δt 表示时间增量。速度 $v(\mathbf{x}^n)$ 通过对基于网格的速度场进行三线性插值，在标记点位置处求得。

前向欧拉法计算开销较低，因为每个时间步只需计算一次速度。其精度阶数较低，因此对时间步大小较为敏感，在长时间模拟中可能导致显著的积分误差。为在保持简洁性的同时提高精度，可将欧拉法与子循环结合使用，即将全局时间步划分为若干较小的子步。这种复合更新方法在分布式内存实现中可能很有用，因为与多阶段龙格-库塔方法相比，它简化了标记点交换和速度插值的逻辑。

龙格-库塔积分

为了提高空间积分精度，Pyroclast 支持二阶和四阶显式龙格-库塔（RK）方法。这些方法通过在每个时间步 [107]内的多个中间点处计算速度场，对轨迹方程（4.7）进行积分。显式龙格-库塔方法的一般形式为

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \sum_{i=1}^{s} b_i k_i, \qquad k_i = v\left(\mathbf{x}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} k_j\right), \tag{4.9}$$

其中 s 表示阶段数，(a^{ij}, b_i) 为定义具体 RK 格式的系数 [107]。

在实际应用中，Pyroclast 实现了二阶海恩法和经典四阶龙格-库塔（RK4）法。二阶格式每步计算两次速度，一次在起始位置，另一次在预测的终点位置：

预测的终点位置：

$$\mathbf{x}^{*} = \mathbf{x}^{n} + \Delta t\, v(\mathbf{x}^{n}), \tag{4.10}$$

$$\mathbf{x}^{n+1} = \mathbf{x}^{n} + \tfrac{1}{2}\Delta t\,[v(\mathbf{x}^{n}) + v(\mathbf{x}^{*})]. \tag{4.11}$$

经典的四阶龙格－库塔格式每个时间步需要进行四次速度场评估：

$$k_1 = v(\mathbf{x}^{n}), \tag{4.12}$$

$$k_2 = v\big(\mathbf{x}^{n} + \tfrac{1}{2}\Delta t\, k_1\big)\,, \tag{4.13}$$

$$k_3 = v\big(\mathbf{x}^{n} + \tfrac{1}{2}\Delta t\, k_2\big)\,, \tag{4.14}$$

$$k_4 = v(\mathbf{x}^{n} + \Delta t\, k_3)\,, \tag{4.15}$$

$$\mathbf{x}^{n+1} = \mathbf{x}^{n} + \tfrac{\Delta t}{6}\,(k_1 + 2k_2 + 2k_3 + k_4). \tag{4.16}$$

对于光滑速度场，该方法在空间上可达到四阶精度，同时具有良好的稳定性 [107]。尽管 RK4 每个时间步需要进行四次速度场评估，但其较高的精度可减少累积的轨迹漂移和插值误差。

此处使用的所有龙格－库塔格式都是显式的，因此易于并行化，因为每条标记点轨迹都可以独立推进。积分阶数的选择体现了计算效率与粒子轨迹捕捉精度之间的权衡，尤其是在空间速度梯度较大的区域。

局部多项式积分

另一种方法是我们称之为局部多项式积分格式的方法。与龙格-库塔方法在多个中间位置评估速度场不同，该技术通过局部泰勒展开近似每个标记点周围的速度场，并在整个时间步内对其进行解析积分。将标记点位置处的速度展开至二次项，$\mathbf{x}^{n}$ 可得

$$v(\mathbf{x}) \approx v_0 + \mathbf{J}(\mathbf{x} - \mathbf{x}^{n}) + \tfrac{1}{2}(\mathbf{x} - \mathbf{x}^{n})^{\mathsf{T}}\mathbf{H}(\mathbf{x} - \mathbf{x}^{n}), \tag{4.17}$$

其中 $v_0 = v(\mathbf{x}^{n})$、$\mathbf{J} = \nabla v(\mathbf{x}^{n})$ 是速度梯度张量，而 $\mathbf{H}$ 是 v 的二阶导数张量。这种表示与下一节所讨论的斯托克斯方程的应力守恒离散化一致，从而能够稳定地计算一阶和二阶速度导数。

标记点轨迹满足特征关系

$$\mathbf{x}^{n+1} = \mathbf{x}^{n} + \int_{0}^{\Delta t} v(\mathbf{x}(t))\, dt. \tag{4.18}$$

为计算该积分，将局部展开的速度场 (4.17) 代入上式。由于假设速度场在每个平流步内保持稳定，因此只有其空间变化会影响轨迹曲率。在短时间间隔 Δt 内，标记点路径可近似为一条速度恒定的直线段，

$$\mathbf{x}(t) \approx \mathbf{x}^n + t\, v_0. \tag{4.19}$$

将这一轨迹代入泰勒展开，即可得到标记点运动过程中所经历速度的显式表达式：

$$v(\mathbf{x}(t)) \approx v_0 + t\, \mathbf{J} v_0 + \tfrac{1}{2} t^2 (\mathbf{H} : v_0 v_0), \tag{4.20}$$

其中 $(\mathbf{H} : v_0 v_0)_i = H^{ijk} v0,_j v0,_k$ 表示二阶导数张量与速度向量进行两次缩并。对上述表达式在 $t \in [0, \Delta t]$ 上积分，可得到解析更新式

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t\, v_0 + \tfrac{1}{2}\Delta t^2\, \mathbf{J} v_0 + \tfrac{1}{6}\Delta t^3\, (\mathbf{H} : v_0 v_0) + \mathcal{O}(\Delta t^4). \tag{4.21}$$

截断至第二项可得到二阶精度的更新；而加入二次项则能进一步提高精度，且当$\mathbf{H}$ 已可用时，几乎无需额外代价。

这种局部多项式方法有两大主要优势。首先，它无需多阶段格式所要求的额外速度插值，便能提高单阶段欧拉更新的精度。其次，它在分布式内存（消息传递接口）模拟中尤其有优势：由于中间速度场评估由局部解析展开取代，在积分步中暂时离开本地域的标记点仍可正确推进，而无需通信获取远程速度值。

4.4 控制方程的离散化

Pyroclast 使用全交错笛卡尔网格上的有限差分格式对不可压缩斯托克斯方程进行离散化。该离散化方案具有应力守恒性，并采用了其他 MIC 地球动力学求解器（包括 I2ELVIS 和 I3ELVIS）中使用的成熟策略 [17, 108]。尽管这种方法并非创新之举，但它为模拟强非均质介质中的缓慢黏性流动提供了稳健且精确的基础。采用交错网格可确保所有算子均使用中心差分进行离散化，从而显著提高模型的稳定性。

本节余下部分将介绍交错网格的结构，并概述控制方程的应力守恒离散化。这两个组成部分构成了 Pyroclast 中机械求解器的基础。

4.4.1 交错网格布局

斯托克斯问题中主要变量的空间排列如图4.3所示。每个压力值 $p_{i,j}$ 的周围都有四个速度分量 v_x 和 v_y，它们位于相邻的单元面上。黑色方块表示基本网格节点，用于存储黏度和密度等辅助量。边界节点（标记为 **B**）用于施加物理域边界上的条件。例如，在西侧和东侧边界施加 $v_y = 0$ ，可通过将相邻内部值取相反数后镜像设置来实现，从而使真实边界位置处的平均值为零。虚节点（标记为 **G**）添加在区域的东侧和南侧，不存储物理值。它们的作用是简化标记点到网格插值，并在应用有限差分算子时避免越界访问。

Pyroclast 采用行优先索引约定。网格索引 i 和 j 分别从 0 到 n_y 以及从 0 到 n_x。每个交错量均使用其所属右下节点的索引，如图4.3左上角所示。

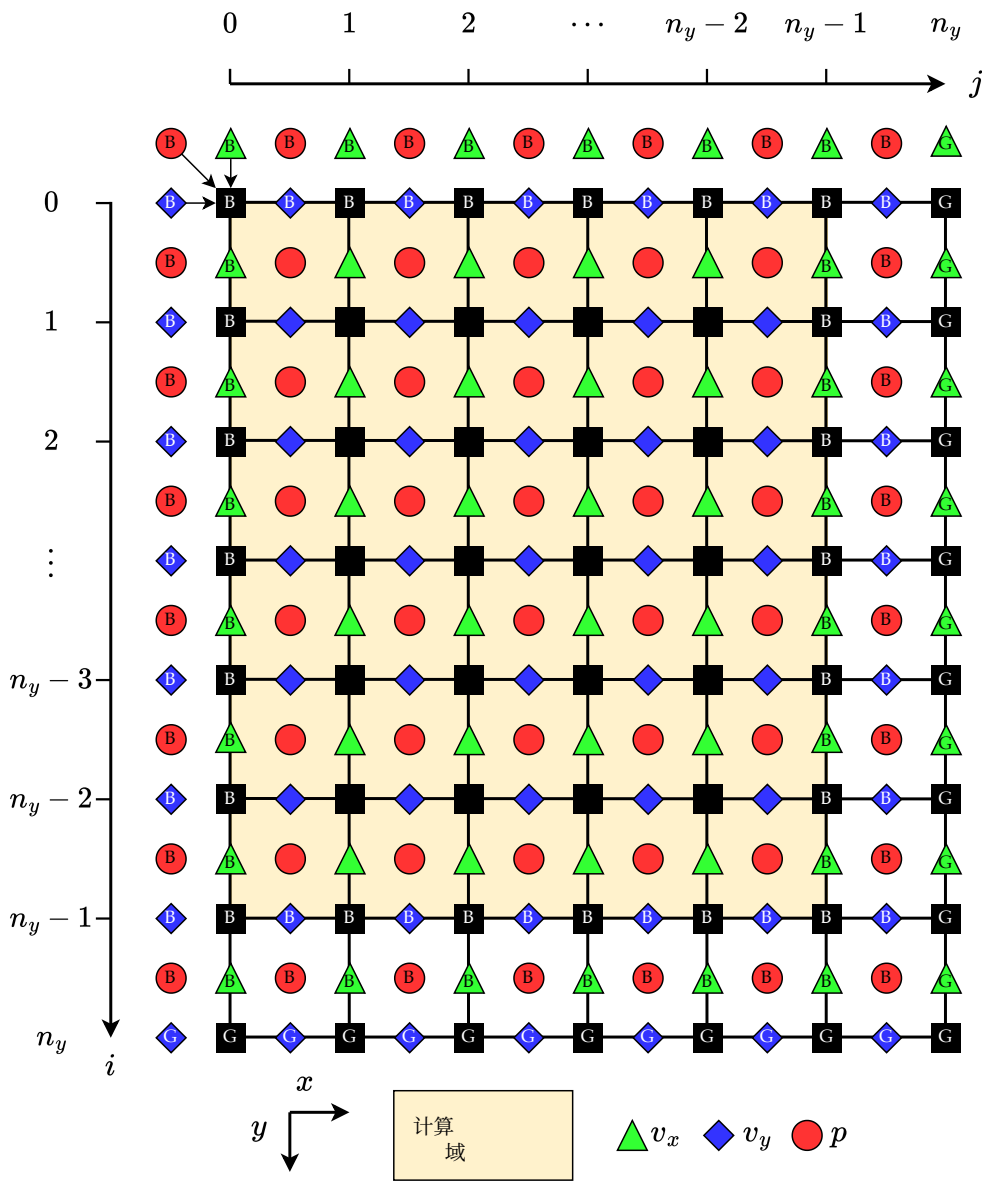

图4.3： 用于不可压缩斯托克斯方程有限差分离散的交错网格排列。速度分量 v_x （绿色三角形）和 v_y （蓝色菱形）定义在单元面上，而压力 p （红色圆形）存储在单元中心。黑色方块标示承载黏度和密度等辅助量的基本节点位置。边界节点（**B**）将计算域向外延伸，以施加边界条件。虚节点（**G**）不包含物理信息，但可确保所有量都能用大小为 $n_y+1 \times n_x+1$ 的数组表示，从而大幅简化基于模板的运算和插值。此布局确保实现应力守恒离散化，所有空间导数均使用中心差分近似。该图改编自 [109]。

4.4.2 应力守恒有限差分

Pyroclast 中动量方程的离散化采用应力守恒形式。方程并非直接用速度梯度表示黏性项，而是首先用偏应力分量表示，如第3章方程（3.4）–（3.5）所示。随后，每个应力分量分别使用有限差分进行离散化，并通过本构关系表示为速度的函数：

$$\sigma'_{ij} = 2\eta\dot{\varepsilon}_{ij} = \eta\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right), \tag{4.22}$$

其中，黏度 η 在相应应力分量定义的位置进行计算。这确保了应力张量的离散散度与交错网格上速度和压力未知量的空间位置保持一致。

为说明这一方法，考虑x方向的动量方程（3.4），其离散形式可写为：

$$\frac{\partial \sigma'_{xx}}{\partial x} + \frac{\partial \sigma'_{xy}}{\partial y} - \frac{\partial p}{\partial x} = 0. \tag{4.23}$$

在交错网格上，该方程以v_x所在的位置为中心，$_x$定义在垂直单元面上。该离散化的模板如图4.4所示。第一项使用压力节点上定义的正应力σ'_{xx}的有限差分进行近似，而第二项则使用基本节点上定义的剪应力σ'_{xy}：

$$\left(\frac{\partial \sigma'_{xx}}{\partial x}\right)_{i,j} \approx \frac{\sigma'^{(i,j+1)}_{xx} - \sigma'^{(i,j)}_{xx}}{\Delta x}, \tag{4.24}$$

$$\left(\frac{\partial \sigma'_{xy}}{\partial y}\right)_{i,j} \approx \frac{\sigma'^{(i,j)}_{xy} - \sigma'^{(i-1,j)}_{xy}}{\Delta y}. \tag{4.25}$$

随后，利用相应位置处定义的黏度，将每个应力分量表示为局部速度梯度的函数：

$$\sigma'^{(i,j)}_{xx} = 2\,\eta^{(i,j)}_{P}\left(\frac{v_x^{(i,j)} - v_x^{(i,j-1)}}{\Delta x}\right), \tag{4.26}$$

$$\sigma'^{(i,j)}_{xy} = \eta^{(i,j)}_{B}\left(\frac{v_x^{(i+1,j)} - v_x^{(i,j)}}{\Delta y} + \frac{v_y^{(i,j+1)} - v_y^{(i,j)}}{\Delta x}\right). \tag{4.27}$$

这里，η_P 表示存储在压力点上的黏度，η_B 表示定义在基本网格节点上的黏度。同样的过程也适用于

y方向的动量方程，并使用适当的应力分量及其自然网格位置。

这种公式确保应力张量的每个分量都使用局部速度差以及在同一位置处计算的黏度进行离散化，从而在单元界面处保持应力守恒，并维持物理上一致的离散化。

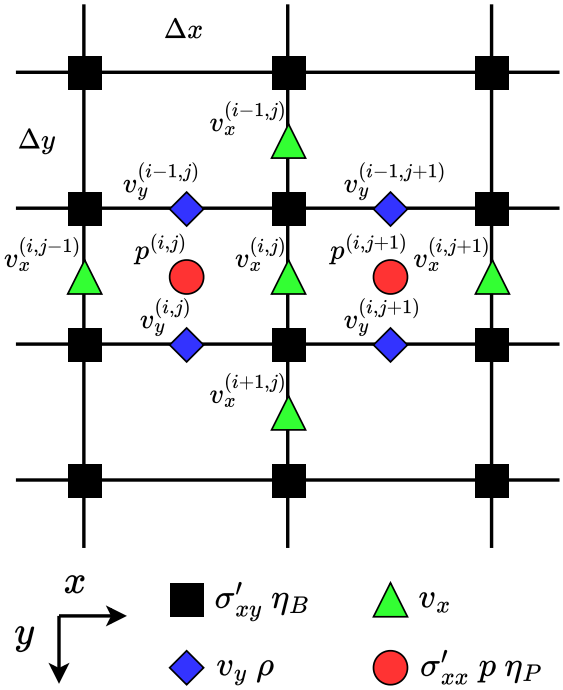

图 4.4： 交错网格上x方向动量方程离散化的模板。图中展示了主要解变量（v_x、v_y和p）的索引方式；存储在相同节点位置的所有其他量（例如，σ'_{xx}、σ'_{xy}、η_P、η_B、ρ）均遵循相同的索引约定。速度分量v_x 和v_y 定义在单元面上，压力p 定义在单元中心，偏应力分量 σ'_{xx} 和 σ'_{xy} 则位于各自的自然位置。黏度存储在与相应应力分量相同的节点上（η_P 位于压力节点，η_B 位于基本节点），从而确保应力守恒离散化的一致性。该图改编自 [108, 110]。

4.5 Stokes系统的快速求解器

上一节所述的斯托克斯方程离散化会产生一个具有鞍点结构的大型稀疏线性系统。这类系统的条件数尤其差，因此在地球动力学建模中构成了重大计算挑战。Pyroclast 的目标是提供一个可扩展的仿真框架，能够在分布式和

GPU加速架构上高效运行，因此，基于稀疏 LU 或 QR分解 [10] 的直接法并不实用。我们转而采用一种快速、完全迭代的求解器，专为斯托克斯系统设计。

鞍点系统是指一类具有一般形式 [20]的块结构线性系统：

$$\begin{bmatrix} A & B_1^\top \\ B_2 & -C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \quad \text{or compactly} \quad \mathcal{A}u = b, \tag{4.28}$$

其中 A、B_1，和 B_2 是非零子矩阵，并且满足以下一项或多项性质：

1. A 是对称的（$A = A^\top$）；2. 对称部分 $H = \frac{1}{2}(A + A^\top)$ 是半正定的；3. $B_1 = B_2 = B$；4. C 是对称半正定的；5. $C = 0$。

这类系统通常源自约束极小化问题的一阶最优性条件。例如，考虑以下二次规划：

$$\min_x \; J(x) = \frac{1}{2} x^\top A x - f^\top x \quad \text{subject to} \quad Bx = g, \tag{4.29}$$

其对应的拉格朗日函数为：

$$\mathcal{L}(x, y) = \frac{1}{2} x^\top A x - f^\top x + y^\top (Bx - g), \tag{4.30}$$

其中 y 是拉格朗日乘子向量。 $\mathcal{L}$ 的鞍点满足线性方程组 (4.28) [20]。

在不可压缩斯托克斯方程的背景下，速度场 v 和压力 p 构成未知向量 $u = [v, p]^\top$，其中压力作为拉格朗日乘子，用于强制满足不可压缩性。因此，离散 Stokes 系统可以写成标准的鞍点形式：

$$\begin{bmatrix} L & G \\ D & 0 \end{bmatrix} \begin{bmatrix} v \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{4.31}$$

其中，L 是离散黏度加权拉普拉斯算子，G 和D分别是离散梯度和散度算子。右端项f 表示体力向量。压力对角线上的零块编码了不可压缩约束，是系统不定性的主要来源。速度与压力之间的这种耦合

使系统变得极难求解，除非使用专为该问题设计的预条件子。

由于方程 (4.31) 中的系统是不可压缩斯托克斯方程的离散等价形式，系统中的每个块都对应于作用于速度场或压力场的连续微分算子的有限差分近似。用算子符号表示，离散项对应于：

$$Lv = \nabla_h \cdot \sigma' = \nabla_h \cdot \left(\eta \left(\nabla_h v + \nabla_h v^\top \right) \right), \tag{4.32}$$

$$Gp = -\nabla_h p, \tag{4.33}$$

$$Dv = \nabla_h \cdot v = 0, \tag{4.34}$$

其中 ∇_h 表示离散梯度算子。矩阵 L 对应于离散黏度加权拉普拉斯算子，此处以速度导数和变黏度的形式表示。矩阵 G 表示离散梯度，而 D 则表示离散散度。所有算子均采用前文所述交错网格上的中心有限差分实现。这种离散化在离散层面保留了连续斯托克斯系统的对称性和耦合结构。

本章余下部分将介绍 Pyroclast 中实现的数值策略，以高效求解这一鞍点系统。我们首先介绍 Uzawa 迭代，它通过舒尔补约化将速度和压力子问题解耦。随后，我们将说明用于速度更新的无矩阵几何多重网格求解器，并讨论平滑器和传递算子。最后，我们概述如何使用黑箱加速器来提高整体迭代的收敛速度。

4.5.1 Uzawa 迭代

此处给出的 Uzawa 迭代推导直接取自 [110], 的相关研究，该研究还介绍了 Pyroclast 中实现的部分数值框架。Uzawa 算法最初是在凹规划 [111], 背景下发展起来的，如今已成为求解鞍点系统的常用策略，包括求解由不可压缩斯托克斯方程 [20]离散化所产生的鞍点系统。

Uzawa 迭代的核心思想是在速度场和压力场之间交替更新。每次迭代中，使用当前的压力估计，通过求解离散动量方程来更新速度。随后，利用更新后的速度计算由系统的舒尔补导出的压力修正，从而满足不可压缩约束。速度与压力的这种解耦使该方法特别适合无矩阵实现，并且天然兼容下一节所述的几何多重网格求解器。

从分块形式的离散 Stokes 系统开始：

$$\begin{bmatrix} L & G \\ D & 0 \end{bmatrix} \begin{bmatrix} v \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{4.35}$$

我们提取第一行，它对应于离散动量平衡：

$$Lv + Gp = f. \tag{4.36}$$

对该方程求解v 可得：

$$v = L^{-1}(f - Gp). \tag{4.37}$$

将此表达式代入不可压缩约束$Dv = 0$，得到压力的舒尔补方程：

$$DL^{-1}Gp = DL^{-1}f, \tag{4.38}$$

其中，矩阵$S = DL^{-1}G$ 称为斯托克斯系统的舒尔补。

直接求解系统（4.38）并不可行，事实上，甚至无法显式构造S或L^{-1} 。因此，我们对压力方程采用 Richardson 不动点迭代：

$$p^{k+1} = p^k + \alpha \left(DL^{-1}(f - Gp^k) \right), \tag{4.39}$$

其中 α 是一个松弛参数。

接下来，我们定义：

$$v^{k+1} := L^{-1}(f - Gp^k), \tag{4.40}$$

并用它来简化压力更新：

$$p^{k+1} = p^k + \alpha\, Dv^{k+1}. \tag{4.41}$$

对式 (4.41) 中的压力更新进行预条件化，对于实现 Uzawa 迭代的快速收敛至关重要。目标是近似舒尔补的逆，

$$S = DL^{-1}G, \tag{4.42}$$

采用计算成本低廉的代理项 $\widetilde{S}^{-1} \approx S^{-1}$。为指导这一构造，我们考察式 (4.32)–(4.34) 中定义的离散算子的量纲缩放。离散梯度和散度算子的缩放分别为 $G \sim \mathcal{O}(1/\Delta x)$ 和 $D \sim \mathcal{O}(1/\Delta x)$，而离散

黏度加权拉普拉斯算子的缩放为 $L^{-1} \sim \mathcal{O}(\Delta x^2/\eta)$。结合这些关系可得

$$S = DL^{-1}G \sim \frac{1}{\Delta x} \cdot \frac{\Delta x^2}{\eta} \cdot \frac{1}{\Delta x} = \mathcal{O}\left(\frac{1}{\eta}\right). \tag{4.43}$$

量纲分析表明，舒尔补主要受黏度倒数支配，这意味着有效的预条件子应复现这种缩放关系。因此，一个简单的对角近似解

$$\widetilde{S} = \frac{1}{\eta} \quad \Longrightarrow \quad \widetilde{S}^{-1} = \eta \tag{4.44}$$

这种简单的预条件子以极低的成本捕捉了舒尔补的主导行为。尽管它忽略了压力自由度之间的耦合，但保留了正确的系数缩放关系，并且在实践中已被证明能够有效稳定 Uzawa 迭代。这一策略并非新方法，已广泛应用于变黏度斯托克斯求解器 [21, 35]。

将此预条件子应用于压力更新，可得到完整形式的 Uzawa 迭代：

$$\begin{cases} v^{k+1} := L^{-1}(f - Gp^k), \\ p^{k+1} := p^k + \alpha\, \eta\, Dv^{k+1}, \end{cases} \tag{4.45}$$

其中， η 在压力节点处逐点计算。对散度残差按局部黏度进行缩放，可确保动量方程与连续性方程之间保持一致的耦合，即使存在显著的黏度变化也是如此。

该表述定义了 Uzawa 迭代：在每一步中，利用当前压力迭代值求解离散动量方程以更新速度，随后使用更新后的速度场的散度修正压力。

Uzawa算法假设速度块L^{-1}的逆可以被精确应用。在实践中，这一点无法实现，尤其是对于大型系统而言。Pyroclast使用专门针对速度子系统的高效几何多重网格求解器来近似应用L^{-1} 。这形成了非精确 Uzawa 迭代格式，即在每一步中近似求解速度更新。尽管如此，只要速度求解器能够充分降低残差，该方法仍然有效且具有可扩展性。下一节将详细介绍这一多重网格求解器的数学原理。

4.5.2 压力零空间消除

如第3章所述，不可压缩斯托克斯方程在动量平衡中仅包含压力梯度项。因此，压力的绝对值在物理上并不受约束，只有压力差会影响流场。从数学上讲，这意味着离散 Stokes 系统具有一个一维零空间，该零空间对应于向压力场中加上任意常数：

$$p \;\to\; p + C, \quad C \in \mathbb{R}. \tag{4.46}$$

由于压力的任意常数偏移都不会改变速度场，零空间的存在会导致系数矩阵奇异。虽然物理解在相差一个常数的意义下仍然唯一，但必须显式处理这一奇异性，以确保数值稳定性和迭代求解器的收敛。

要消除这一压力零空间，必须施加一个约束来固定压力参考水平。在 Pyroclast 中，我们施加零均值压力条件：

$$\langle p \rangle = \frac{1}{V_\Omega} \int_\Omega p \, dV = 0, \tag{4.47}$$

其中 V_Ω 是总域体积，$\langle p \rangle$ 表示体积加权平均压力。

对于单元体积为 $V_i = \Delta x \Delta y$ 或 $V_i = \Delta x \Delta y \Delta z$ 的离散网格，该条件可表示为

$$\langle p \rangle_h = \frac{\sum_{i=1}^{n_p} V_i \, p_i}{\sum_{i=1}^{n_p} V_i} = 0. \tag{4.48}$$

因此，在每次 Uzawa压力校正步骤之后，都会从场中减去平均压力：

$$p^{k+1} \;\leftarrow\; p^{k+1} - \frac{\sum_{i=1}^{n_p} V_i \, p_i^{k+1}}{\sum_{i=1}^{n_p} V_i}. \tag{4.49}$$

这里，V_i 表示与压力节点i相关联的局部控制体积，而n^p 表示压力未知量的总数。

这种归一化会移除零空间分量，同时不会影响驱动流动的物理压力梯度。它确保式（4.45）中的压力校正步骤保持良态，并使迭代求解器能够可靠收敛。零均值条件在数学上等价于将压力固定在单个节点处，但它保留了系统的完全对称性，并避免引入人为的边界效应。在均匀网格上，式（4.49）可简化为简单的算术平均值减法，

$$p^{k+1} \leftarrow p^{k+1} - \frac{1}{n_p} \sum_{i=1}^{n_p} p_i^{k+1}. \tag{4.50}$$

4.6 多重网格法

如前所述，上一节引入的 Uzawa 迭代需要反复应用速度算子 L^{-1} 的逆算子，该算子对应于离散化的变黏度拉普拉斯算子。在大规模或无矩阵设置下，显式计算这个逆算子是不可行的，而雅可比或 Gauss-Seidel 方法等经典迭代方法 [112] 收敛过慢，因而不具备实用性。这些方法能够有效地阻尼高频误差，但无法有效消除平滑的低频误差模态，而在仅进行几次迭代后，低频误差模态就会主导残差。因此，仅使用标准平滑器实现全局收敛，需要进行数量大到不切实际的迭代。

多重网格（MG）方法通过在误差分量所处的自然空间尺度上进行处理，克服了这一局限。其核心思想是结合细网格上的平滑（消除高频误差）与粗网格上的修正（处理低频分量），从而加速收敛。正如 Hackbusch 在 [113], “中所指出的，多重网格迭代的显著特点是收敛速度快；随着离散化网格加密，其收敛速度不会下降”。因此，所需的计算工作量仅随未知量的数量成比例增长，不仅实现了最优复杂度，而且比例常数也极小，使多重网格方法成为求解大规模椭圆问题最高效的方法之一。

在本研究中，多重网格法可高效近似L^{-1} 的作用，并为非精确 Uzawa 迭代提供计算基础。该方法以完全无矩阵的几何形式实现，充分利用计算网格的结构化特性。因此，求解器能够直接通过模板而非组装矩阵来表示传递算子、平滑器和粗网格修正，从而提升在 CPU 和 GPU 上的性能与可扩展性。

本节余下部分将更详细地介绍多重网格法，以及 Pyroclast 如何利用它求解斯托克斯方程中的速度子系统。我们首先说明经典平滑器的局限性，以及采用多层方法的动机。随后，我们概述多重网格循环的结构，比较几何和代数多重网格算法，并介绍限制、延拓和平滑算子的实现。本节最后讨论多重网格求解器如何集成到 Uzawa 迭代中，以及如何使用与 Krylov 方法相关的全局加速技术 [112]。

4.6.1 平滑与信息传播

雅可比和 Gauss-Seidel 方法等经典迭代方法，会使用一个小型局部模板更新解网格中的每个点。这些模板运算计算效率高且易于实现，但其空间局部性固有地限制了传播能力。每次更新只会根据某个点的直接邻居修改该点，因此信息在区域内传播的速度受到限制。

为说明这一点，考虑规则笛卡尔网格上的一个简单五点模板，如图4.5所示。该模板利用中心节点的北、南、东、西邻居更新中心节点，并在每次迭代中遍历整个网格，重复这一过程。假设某个扰动或修正源于左上角的一个点（以红色菱形标出）。由于雅可比法的每次迭代只使用前一次扫描中的值，因此来自该源的信息每次迭代最多只能沿一条网格边传播。要到达区域中较远的部分，就需要进行多次迭代，其次数与源节点和目标节点之间的曼哈顿距离成正比。

理论上，高斯-赛德尔迭代通过在同一扫描中一旦有更新值可用便立即使用它们，改进了这一点。这使信息能够更快速地沿网格逐步传播，理论上一次遍历就可能到达整个区域。然而，这种链式传播过程本质上是串行的，而且传播值在每一步都会逐渐受到“污染”，导致信号质量在长距离传播时下降。此外，高斯-赛德尔方法的顺序执行特性使其不适用于现代并行架构。

为了解决这个问题，红黑高斯-赛德尔（RBGS）引入了双色排序，使并行更新成为可能。通过将网格划分为两个交错的集合，可以独立并行地更新每种颜色的节点。然而，这也重新形成了信息流动的屏障：每一步中只有交替的节点彼此相连。因此，其传播行为与雅可比方法相似，每次扫描中信息最多传播两个网格边。因而，RBGS 仍需要与网格直径成正比的迭代次数，才能传播全局修正量。

这些局限是逐点迭代法的固有特性，也说明了这类平滑器为何只能有效地减少高频误差。低频（长波长）误差在求解后期占主导地位，需要全局信息才能消除，单靠局部模板更新无法高效地修正。由此便引出了多重网格方法：它通过在不同网格分辨率上针对不同尺度的误差，加速收敛。

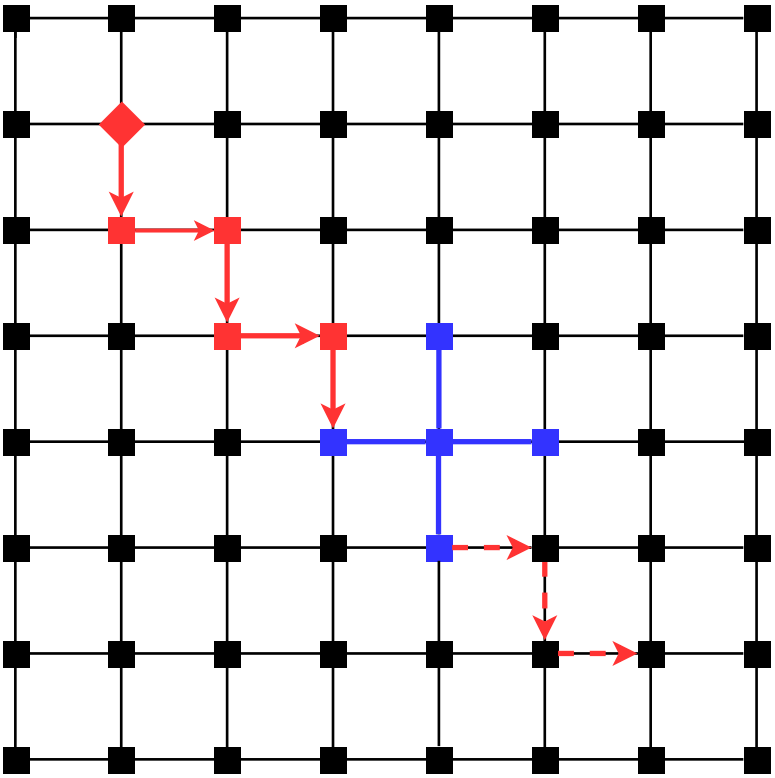

图4.5： 在结构化网格上使用局部五点模板（蓝色）进行信息传播的示意图。红色菱形标记了左上角的信息源。每次应用模板，都会根据节点紧邻的北侧、南侧、东侧和西侧邻居更新该节点。在雅可比迭代中，所有更新都使用上一次扫描的值，这意味着每次迭代中信息只能前进一条网格边（红色箭头）。因此，要使信息传播至整个区域，需要进行与网格直径成正比的扫描次数。Gauss-Seidel 迭代通过立即使用更新后的值改进了这一点，使信息能够在一次扫描中沿网格逐步传播。然而，这种逐步传播会导致信号逐渐受到污染，而且该方法本质上是串行的，因此不适用于并行执行。红黑高斯-赛德尔（RBGS）通过将网格划分为两部分并分两次交替扫描来实现并行性，但这也限制了信息流动。因此，RBGS 的表现与雅可比迭代相似，传播信息至整个网格所需的扫描次数约为其一半（即直径 / 2）。

4.6.2 多重网格循环

多重网格方法的核心思想是，在一系列网格分辨率构成的层次结构上求解误差修正量，从而加速收敛。

设 $Ax = b$ 为最细网格上离散化得到的线性方程组。假设x^ℓ 是层级 ℓ 上的近似解。我们定义误差$\delta x^\ell = x^* - x^\ell$，其中$x^*$ 是该层级上的精确解。将其代入系统可得：

$$A^\ell(x^\ell + \delta x^\ell) = b^\ell \quad \Rightarrow \quad A^\ell \delta x^\ell = b^\ell - A^\ell x^\ell = r^\ell, \tag{4.51}$$

其中，A^ℓ 是层级 ℓ 上的系统矩阵，b^ℓ 是右端项，r^ℓ 是残差。求解该残差方程可得到一个修正量，将其加到当前解上：

$$x^\ell \leftarrow x^\ell + \delta x^\ell. \tag{4.52}$$

多重网格法构建一个按层级索引的网格层次结构$\ell = 1, \ldots, N$，其中 $\ell = 1$ 表示最细网格，$\ell = N$ 表示最粗网格。残差通过限制算子R^ℓ从细层级传递到粗层级（$\ell \to \ell + 1$），而修正量则通过延拓算子P^ℓ从粗层级插值回细层级（$\ell + 1 \to \ell$）。

在每个层级，校正方程通过将残差传递到更粗层级并将修正量传播回更细层级来定义并近似求解。每个层级的操作可表示为：

(1) 预平滑：(2) 计算残差：(3) 限制到粗网格：(4) 使用多重网格法求解粗网格系统：$A^{\ell+1}\delta x^{\ell+1} = b^{\ell+1} = R^\ell r^\ell$(5) 延拓并应用修正：$x^\ell \leftarrow x^\ell + P^\ell \delta x^{\ell+1}$(6) 后平滑：

$$\begin{aligned} x^\ell &\leftarrow S^\ell(x^\ell, A^\ell, b^\ell) \\ r^\ell &= b^\ell - A^\ell x^\ell \\ b^{\ell+1} &= R^\ell r^\ell \end{aligned} \tag{4.53}$$

这里，R^ℓ 和P^ℓ 分别表示层级 ℓ 和 $\ell + 1$之间的限制算子与延拓算子。

在最粗层级N上，校正方程

$$A^N \delta x^N = b^N = R^{N-1} r^{N-1} \tag{4.54}$$

要么通过直接法 [114, 10] 求解，要么通过少量平滑迭代进行近似。这就结束了步骤 (4) 中的多重网格粗网格求解递归，并使延拓阶段得以开始。

这些步骤组合起来构成一个V循环，它递归地向下遍历更粗层级，然后返回细网格。这个过程确保所有频率的误差分量都能得到有效减少：高频误差在细网格上平滑，而低频误差则在粗网格上得到修正。图 4.6 展示了这一递归过程的结构，其中每个层级代表一种用于加速收敛的不同网格分辨率。

这种递归形式得到的求解器，其收敛率不受网格分辨率影响。多重网格方法经过高效实现后，计算复杂度可达线性 $\mathcal{O}(N)$，因此成为求解大规模椭圆系统最有效的方法之一。

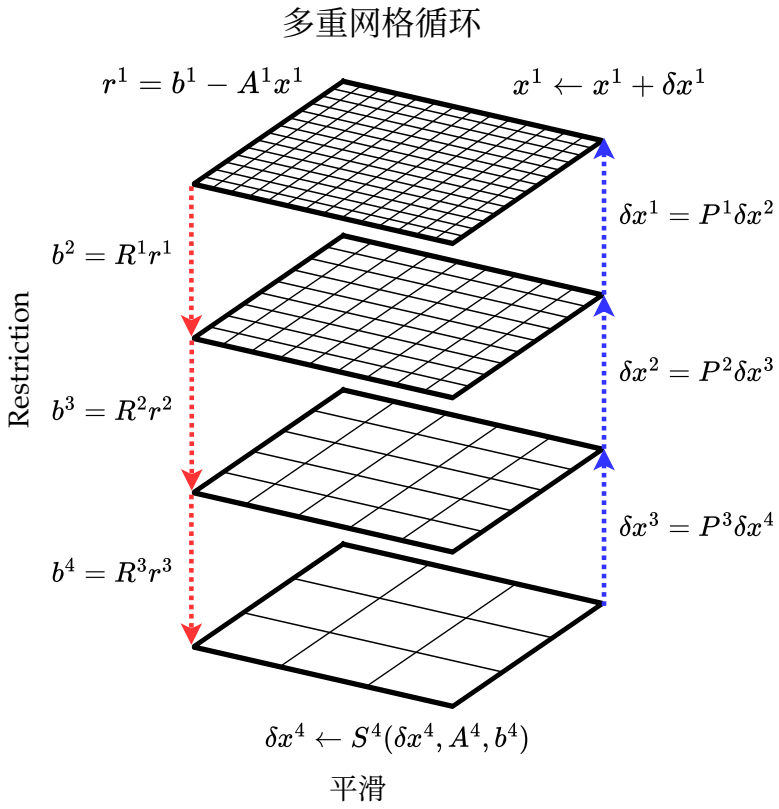

图 4.6： 多重网格 V 循环的示意图。残差依次从细网格限制到粗网格（红色箭头），以高效消除低频误差。在最粗网格上，对系统进行平滑处理或使用直接求解器求解。随后将修正量延拓回较细网格（蓝色箭头），用于修正当前解。这种层级式过程结合了细层级上对高频误差的平滑处理，以及针对低频分量的粗网格修正，从而加快收敛。

4.6.3 几何多重网格与代数多重网格

多重网格法大致可分为两类：几何多重网格（GMG）和代数多重网格（AMG）。两者的关键区别在于粗网格层次结构及相应传递算子的构建方式。

在几何多重网格中，网格结构依据区域的底层几何形状显式定义。粗网格通过以较低的空间分辨率重新离散底层偏微分方程来构建，而限制算子和延拓算子则利用插值模板或平均方案从这一结构中导出。这种方法需要了解网格布局，最适合规则或结构化网格问题，因为这类问题可以高效地生成各个网格层级。由于该方法直接基于几何层次结构进行计算，因此非常适合无矩阵实现，并且

具有高性能和低内存占用 [115]。

相比之下，代数多重网格直接根据组装后的系统矩阵构建多重网格层次结构，而不依赖任何几何信息。粗化过程和插值算子根据稀疏模式和矩阵元素推断得出。这使 AMG 更具通用性，广泛适用于不存在明显网格层级结构的非结构化网格或复杂几何形状。然而，这种通用性也带来了更长的设置时间、更高的内存用量以及更复杂的实现。AMG 通常还要求以组装形式访问矩阵，因此对于无矩阵求解器而言吸引力较低 [116]。

AMG 的主要优势在于其黑箱可用性。提供系统矩阵后，求解器便可自动构建合适的层次结构并应用多重网格循环，无需用户干预。这促成了许多不同 AMG 库的开发，例如 PyAMG [117],、AMGCL [118], 和 NVIDIA 的 AmgX [119], 。这些库通常集成到使用组装矩阵和不规则网格的有限元及有限体积代码中。

在 Pyroclast 中，我们采用几何多重网格方法，原因有三。首先，使用规则的笛卡尔网格可以轻松生成更粗层级，并能直接以模板形式定义传递算子。其次，几何多重网格与我们的无矩阵实现自然契合，无需组装或存储大型稀疏矩阵。第三，也是至关重要的一点，它能让我们全面掌控物理场在不同网格层级间的传递方式。在我们的研究背景下，这一点至关重要，因为速度和压力等变量的量级差异显著。通过对每个物理场分别处理，几何多重网格可确保粗层级表示保留底层物理的结构和尺度特征。这些优势使几何多重网格成为 Pyroclast 的最佳选择。

4.6.4 限制与延拓算子

多重网格层次结构中相邻层级之间的残差和修正量通过限制算子和延拓算子 R^ℓ 和 P^ℓ 进行传递。二者均采用第 4.2 节介绍的加权双线性插值方案。

延拓

延拓算子 P^ℓ 将粗网格修正 $\delta x^{\ell+1}$ 插值到更细的层级 ℓ，生成细网格修正场 δx^ℓ。在 Pyroclast 中，此操作的实现与网格到标记点插值例程完全相同：每个细网格节点都会接收来自周围四个粗网格节点的贡献，并根据它们的相对位置进行双线性加权。

简洁形式如下：

$$\delta x^{\ell} = P^{\ell} \delta x^{\ell+1}. \tag{4.55}$$

其逐点形式为：

$$\delta x^{\ell,(p,q)} = \sum_{(i,j)} w^{(i,j)}_{(p,q)} \, \delta x^{\ell+1,(i,j)}, \tag{4.56}$$

其中，$w^{(i,j)}{}_{(p,q)}$ 为双线性插值权重。由于这些权重之和始终为 1，因此无需归一化。该操作完全是局部的，仅依赖相邻的粗网格节点，并且很容易并行化。

限制

限制算子 R^{ℓ} 将细网格残差 r^{ℓ} 传递到更粗层级：

$$b^{\ell+1} = R^{\ell} r^{\ell}. \tag{4.57}$$

该操作在结构上类似于标记点到网格插值：它使用双线性加权，将多个细网格点的数据归约到较少的粗网格目标点。与延拓不同，限制操作会带来并发方面的挑战：多个细网格节点可能会对同一个粗网格节点作出贡献，从而在并行执行期间引发潜在的竞态条件。

为避免代价高昂的原子操作，Pyroclast 利用了网格层级结构的规则笛卡尔结构。由于所有层级都采用结构化布局，即使采用瑞士十字自适应细化， [11]，节点坐标(x^{ℓ},y^{ℓ}) 也可以表示为可分离的一维向量。这使得通过坐标二分高效计算索引边界成为可能，并且可以利用这一点并行执行此操作。

限制分两个阶段进行。首先，每个粗网格节点(i,j) 使用双线性权重累积附近细网格残差的贡献。其次，对累积值进行归一化：

$$r^{\ell+1,(i,j)} = \frac{\sum_{(p,q)} w^{(i,j)}_{(p,q)} \, r^{\ell,(p,q)}}{\sum_{(p,q)} w^{(i,j)}_{(p,q)}}. \tag{4.58}$$

为确保线程安全且无需使用原子操作，粗网格被划分为四个互不相交的颜色组，如图 4.7 所示。每组包含一组粗网格节点，这些节点彼此不共享参与计算的细网格节点，因此可以独立处理且不会发生竞争。这种四色分解通过针对每个颜色组分别进行并行扫描，实现了并行限制。

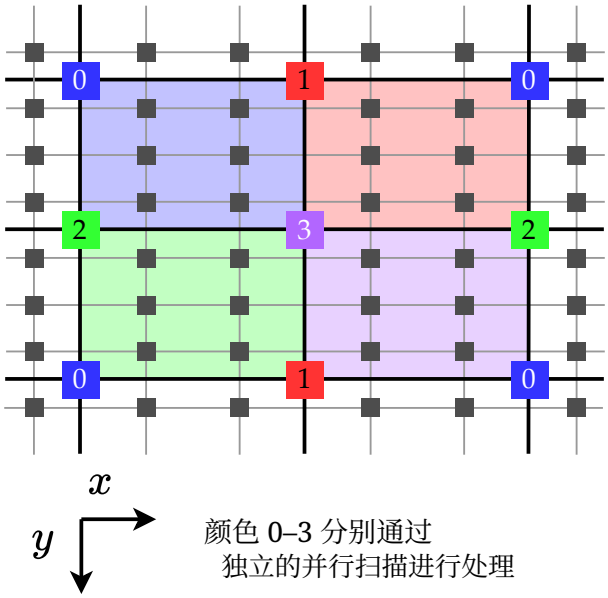

图 4.7： 用于并行限制操作的粗网格四色分解。每种颜色（0–3）代表一组可以同时处理的粗网格单元，因为同色的任意两个单元都不共享节点，因此不会产生竞态条件。每个彩色区域内的细网格值通过加权双线性插值限制到粗网格，每种颜色分别进行一次并行扫描。这种划分无需原子操作即可实现确定且线程安全的归约。

图 4.7 展示了这一策略，算法 2 则给出了完整的实现。算法首先对细网格坐标进行二分，以计算每个粗网格单元的边界索引。在每次颜色扫描中，使用各边界区域内的细网格节点计算并累加加权贡献。最后进行归一化步骤，以确保结果按正确比例缩放。

算法 1： 二分界限

输入： 单调数组 $x[0 \ldots n-1]$，目标值 x_t。
输出： 满足 $x[i] \geq x_t$ 的最小索引 i。

$l := 0, \quad r := n - 1;$
当 $l < r$ **时**
  $m := \lfloor (l + r)/2 \rfloor;$
  如果 $x[m] < x_t$，**则**
    $l := m + 1;$
  else
    $r := m;$
返回 l

算法 2： 四色并行限制

输入： 细网格场 u^h，坐标 $(x^h,\ y^h)$；
粗网格缓冲区 u^H、w^H，坐标 $(x^H,\ y^H)$。
输出： 限制后的粗网格场 u^H。
初始化： $u^H := 0,\quad w^H := 0$;
$\Delta x^H := x^H[1] - x^H[0],\quad \Delta y^H := y^H[1] - y^H[0]$;

// 计算细网格索引边界
分配 $i_{\text{bounds}}[0 \ldots n_y^H]$, $j_{\text{bounds}}[0 \ldots n_x^H]$;
并行执行 $i^H = 0$ **到** $n_y^H - 1$
  $i_{\text{bounds}}[i^H]$:=二分边界$(y^h, y^H[i^H])$;
对于$j^H = 0$ **到**$n_x^H - 1$ **，并行执行**
  $j_{\text{bounds}}[j^H]$:=二分边界$(x^h, x^H[j^H])$;
$i_{\text{bounds}}[n_y^H] := n_x^h - 1,\quad j_{\text{bounds}}[n_x^H] := n_y^h - 1$;

// 在粗网格单元上进行四次棋盘格式扫描
对于 $c_y = 0$**至**1 **执行**
  对于 $c_x = 0$**至**1 **执行**
    对于 $i^H = c^y$ **至** $n_y^H - 1$ **步长为** 2 **执行并行**
      对于 $j^H = c_x$ **至** $n_x^H - 1$ **步长为** 2 **执行并行**
        // 遍历参与计算的细网格节点
        对于 $i^h = i_{bounds}[i^H]$ **到**$i_{bounds}[i^H{+}1]$ **执行**
          对于 $j^h = j_{bounds}[j^H]$ **到** $j_{bounds}[j^H{+}1]$ **执行**
            // 计算并应用双线性权重
            对于 $(\delta_i, \delta_j) \in \{(0,0),(1,0),(0,1),(1,1)\}$ **执行**
              $r_x := |x^h[j^h] - x^H[j^H + \delta_j]| / \Delta x^H$;
              $r^y := |y^h[i^h] - y^H[i^H + \delta_i]| / \Delta y^H$;
              $w := (1 - |r_x|)\quad (1 - |r_y|)$;
              $u^H[i^H + \delta_i,\ j^H + \delta_j] \mathrel{+}= w\ u^h[i^h,\ j^h]$;
              $w^H[i^H + \delta_i j^H + \delta_j] \mathrel{+}= w$;

// 归一化累积值
对于每个粗网格节点（$i^H,\ j^H$）**并行执行**
  $u^H[i^H,\ j^H] := u^H[i^H,\ j^H] / w^H[i^H,\ j^H]$;

4.6.5 平滑器

平滑器在所有网格层级上都至关重要。在细网格上，它们减少高频误差；而在粗网格上，它们则阻尼剩余的低频分量。通过局部更新，它们无需全局通信即可改进解。Pyroclast 支持多种平滑算法，本节将对此作简要讨论。关于其数学表述以及其他稀疏线性系统迭代方法的更详细讨论，可参见 [112]。

我们考虑一个线性方程组

$$Ax = b, \tag{4.59}$$

其中

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{4.60}$$

目标是通过减小残差$r^{(k)} = b - Ax^{(k)}$，迭代改进$x^{(k)}$ 这一近似解，使其趋近真实解x^* 。

雅可比平滑器

经典的雅可比法仅使用上一次迭代中的值来更新 x 的每个元素：

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n. \tag{4.61}$$

由于每次更新完全取决于旧迭代结果，因此可以并行计算 x 的所有条目。这使得雅可比平滑器非常适合 GPU 等大规模并行架构。

在实践中，为了提高数值稳定性并防止过度修正，雅可比法采用其阻尼形式：

$$x_i^{(k+1)} = (1 - \omega) x_i^{(k)} + \omega \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad \omega \in (0, 1]. \tag{4.62}$$

松弛参数 ω 控制阻尼强度。当 $\omega = 1$ 时，我们得到标准的雅可比迭代。在 Pyroclast 中，阻尼雅可比平滑器是一种稳健且完全并行的松弛法，非常适用于无矩阵实现。

红黑高斯-赛德尔平滑器

Gauss-Seidel 方法通过在同一次迭代中使用最新更新的值来改进雅可比法：

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n. \tag{4.63}$$

由于每次更新都会立即使用最新的可用信息，高斯-赛德尔方法可能比雅可比方法收敛更快，具体取决于问题。然而，这种顺序数据依赖关系使其难以实现并行化。

为实现高效的并行执行，Pyroclast 实现了红黑高斯-赛德尔（RBGS）平滑器 [112]。在此变体中，网格被划分为两个交错的子集（“红”节点和“黑”节点），使得每个红节点上的高斯-赛德尔更新仅依赖于黑邻居，反之亦然。在每次扫描期间，所有红节点都会使用最新的黑节点值并发更新，随后并行更新黑节点。这种双色方法消除了每种颜色组内部的直接数据依赖关系，从而可以并行执行高斯-赛德尔方法，同时保留其相较于雅可比方法的大部分收敛优势。

为保持稳定性并更好地控制收敛性，我们再次采用阻尼高斯-赛德尔变体：

$$x_i^{(k+1)} = (1-\omega) x_i^{(k)} + \omega \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right), \tag{4.64}$$

其中 $\omega \in (0, 1]$ 是松弛参数。这种形式也称为逐次超松弛（SOR） [112]。

RAS 型 Jacobi 平滑器

在 Pyroclast 等无矩阵求解器中，未知向量 x 对应于存储为多维数组的一个或多个物理场，例如 $u[i,j]$。对于大型二维或三维网格，模板运算通常受内存带宽和缓存复用限制，而非算术运算成本。为提高内存局部性，我们采用分块策略，将计算域划分为可放入缓存的瓦片，并分别处理。这称为空间分块 [110]。

然而在实践中，二维纯空间分块带来的性能提升有限。为解决这一问题，Pyroclast 实现了 [110] 中提出的 RAS型时间分块方案。在这种方法中，每个瓦片会执行若干次局部雅可比迭代，然后再移至下一个瓦片，从而重复利用仍保留在缓存中的数据。瓦片之间设有小幅重叠，以

确保子域之间的连续性。这与限制性加性施瓦茨（RAS）或区域分解法 [112, 115] 的理念相似。

每次迭代都会更新一个分块的内部区域，同时将其外部边界视为固定不变；经过几次局部迭代后，再将结果写回全局场。为了避免最小化伪影，在下一次外部迭代中，平铺网格会在两个空间方向上随机移动，从而确保分块的边界随时间变化，防止误差沿固定接缝累积。该算法的伪代码如算法3所示。与标准雅可比方法或空间分块变体相比，这种 RAS 型雅可比平滑器在保持相同数学结构和数值稳定性的同时，实现了显著更高的缓存效率和可扩展性 [110]。

算法3： RAS型时间分块（改编自 [110]）

输入： 场缓冲区 u_{read}、u_{write}，分块大小 $(T_I$、$T_J)$，偶数个最大迭代次数 $N_{\max}$，偶数次内部迭代 T_{inner}。

输出： 已更新字段 u。

$N_{\text{outer}} := \lceil N_{\max}/T_{\text{inner}} \rceil$, $N_{\text{outer}} \mathrel{+}= N_{\text{outer}} \% 2$;

对于 $t := 1$ **到** N_{outer} **执行**

- 绘制随机偏移 $s_i \in [0, T_I)$、$s_j \in [0, T_J)$，并采用周期性循环回绕；
- 将内部区域划分为大小为 (T_I, T_J) 的重叠瓦片，使用 $(\text{秒}_i, \text{秒}_j)$；
- **对每个**瓦片 $\mathcal{T}$（并行）**执行**
  - **对** $\tau := 1$ **至** T_{inner} **循环**
    - **对** $(i,j) \in \mathcal{T}$ **循环**
      - $u_{\text{write}}(i,j) :=$ 雅可比模板(u_{read}, i, j);
    - // 本地缓冲区乒乓
    - 交换$u_{\text{read}} \leftrightarrow u_{\text{write}}$
  - // 每个单元格只有一个写入者，但可能被多个分块读取（良性竞争）

这三种平滑器共同构成了一套灵活的工具，可适用于不同的硬件和问题规模。阻尼雅可比和红黑高斯-赛德尔平滑器简单、稳定且易于并行化，适合一般用途；而 RAS 型雅可比平滑器则提供了一种高性能替代方案，专为现代缓存层次结构优化。

4.6.6 Stokes 系统的多重网格解法

如第 4.5.1 节所述，求解斯托克斯系统的 Uzawa算法通过交替进行速度更新和压力修正来实现：

$$\begin{cases} v^{k+1} := L^{-1}(f - Gp^k), \\ p^{k+1} := p^k + \alpha\,\eta\,Dv^{k+1}, \end{cases} \tag{4.65}$$

其中，L 表示离散变黏度拉普拉斯算子，G和D 分别表示离散梯度和散度算子。压力更新可确保满足不可压缩条件，而局部黏度场 η 则提供一致的物理尺度。

该方案的一项关键挑战在于速度算子逆 L^{-1} 的应用。由于对于大型问题，显式求逆L 在计算上不可行，因此我们转而近似求解速度子问题：

$$Lv = f - Gp^k, \tag{4.66}$$

使用少量（通常为一到两次）多重网格 V 循环。求解期间压力保持不变，右端项表示当前压力场下的动量残差。

得到近似速度v^{k+1}后，计算速度场的散度，并使用方程（4.65）中的预条件公式更新压力。由于每步只进行少量多重网格迭代，因此该方法整体计算效率较高，同时仍能实现稳健收敛。

4.6.7 黏度重标度与初始猜测

在黏度对比强烈的情况下求解斯托克斯方程，对迭代求解器而言是一项重大挑战。在这种情况下，较差的初始猜测会导致压力场收敛缓慢，甚至完全停滞。为提高稳健性，Pyroclast 实现了一种黏度重标度策略，与 [21], 中提出的方法类似：该策略分阶段逐步引入完整的黏度对比，使求解器能够逐步构建出更好的解近似。

设 $\eta_{\min} = \min \eta$ 表示区域内的最小黏度。在每个阶段，我们定义一个重标度后的计算黏度场：

$$\eta_{\text{comp}} = (1 - \theta)\,\eta_{\min} + \theta\,\eta, \tag{4.67}$$

其中，$\theta \in [0, 1]$ 是控制对比度大小的混合因子。最初，使用均匀黏度场 $\eta_{\text{comp}} = \eta_{\min}$ 求解系统，从而得到更平滑的压力解。随后以固定步长递增 θ（例如 $\theta \in \{0, 0.25、0.5、0.75、1.0\}$），逐步恢复

完整的物理对比度。在每个阶段，都会将前一次迭代得到的压力和速度解重新用作初始猜测，随着系统的病态程度不断加剧，引导求解器趋近真实解。

除了黏度重标度，Pyroclast 还支持基于物理信息的压力场初始化。对于涉及重力的问题，我们使用岩静压力分布初始化压力：

$$p_{\text{litho}}(x, y) = \int_0^y \rho(x, y')\, g_y\, dy', \tag{4.68}$$

它表示在密度为 ρ 的流体柱中平衡引力所需的静水压力。这为压力场提供了合理的初始近似解。

4.7 加速器

虽然前文所述的 Uzawa 迭代为斯托克斯系统提供了一种高效且可扩展的求解器，但仍可通过全局加速技术进一步提升其性能。在数值线性代数文献中，通常将 Uzawa 型不动点迭代视为 Krylov 子空间方法的预条件子 [120, 35]。然而，在 Pyroclast 中，我们采用不同的视角：由于 Uzawa 迭代是负责大部分计算的核心求解器，因此我们将这些外层 Krylov 或拟牛顿技术称为加速器。

这种区别源于高性能计算中的实际考量。Uzawa 等基于模板的不动点方法非常适合分布式内存并行和 GPU 执行。它们的通信模式本质上是局部的，只需要在相邻区域之间进行光环交换。相比之下，Krylov 方法需要通过全局归约计算内积，而且通常需要更复杂的正交化过程 [112]。尽管已开发出流水线 Krylov 方法和避免通信的 Krylov 方法，并将其实现于 PETSc 等库中 [43],，但它们仍易受全局通信开销影响，扩展也更加困难。

另一个限制因素是内存消耗。Krylov 方法通常需要存储多个过去的搜索方向或解向量；对于这类问题，数量通常在 10 到 30 个之间 [120, 35]。在内存是关键瓶颈的 GPU 加速系统中，内存占用的增加会严重限制能够高效求解的物理问题规模。根据我们的经验，使用标准 Krylov 求解器作为加速器时，内存用量会增加 10x–30x。

尽管如此，Krylov 方法具有更强的稳健性，在求解特别棘手的系统时很有价值。在 Pyroclast 中，我们发现

第 4.6.7 节所述的黏度重标度策略通常足以确保收敛，无需使用 Krylov 加速。不过，对于非常困难的系统或极端的黏度对比，加速方法可以大幅减少所需的 Uzawa 迭代次数。

由于 Pyroclast 实现的是非精确 Uzawa 迭代，其中L^{-1} 的作用仅通过多重网格求解器进行近似，因此底层的不动点算子会在每次迭代时发生变化。因此，标准 Krylov 方法无法直接应用。为此，我们必须使用灵活的Krylov 求解器，它们能够处理可变预条件器 [112]。

本节介绍两种与 Pyroclast 迭代框架兼容的加速技术：

- 广义共轭残差（GCR）方法，这是一种适用于线性不动点问题的灵活 Krylov 求解器。
- 安德森加速是一种拟牛顿方法，适用于线性和非线性不动点迭代，也可用于加速求解器的其他组件，例如塑性迭代 [19]。

这些方法各有优势、互为补充，并且都作为可选加速器受 Pyroclast 框架支持。

4.7.1 将 Uzawa 迭代作为不动点法

回顾式（4.31）中引入的离散 Stokes 系统，

$$\underbrace{\begin{bmatrix} L & G \\ D & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} v \\ p \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} f \\ 0 \end{bmatrix}}_{b}, \tag{4.69}$$

其中，L 是离散向量拉普拉斯算子，G 和D 分别是离散梯度和散度算子，而 $x = [v、p]^\top$ 是完整的速度–压力向量。
再次回顾第 4.5.1 节中定义的 Uzawa 迭代，该迭代依次更新速度和压力，具体如下：

$$\begin{cases} v^{k+1} := L^{-1}\big(f - Gp^k\big), \\ p^{k+1} := p^k + \alpha\,\eta\,D\,v^{k+1}, \end{cases} \tag{4.70}$$

其中，$\alpha > 0$ 是松弛参数，η 是包含局部黏度的对角矩阵。第一步近似求逆黏性算子L，第二步则根据速度散度进行压力修正。

矩阵分裂表示

按照 [121],中讨论的方法，Uzawa 迭代 (4.70) 可以改写为速度–压力空间上的线性不动点迭代。具体做法是为 Stokes 算子A 确定一种合适的矩阵分裂，使迭代可以表示为

$$Mx^{k+1} = Nx^k + b, \tag{4.71}$$

对于适当定义的矩阵 M 和N。

从（4.70）出发，我们将这两个更新方程写成线性化形式：

$$\begin{aligned} Lv^{k+1} &= f - Gp^k, \\ Dv^{k+1} - (\alpha\eta)^{-1}p^{k+1} &= -(\alpha\eta)^{-1}p^k. \end{aligned}$$

这两个关系可以合并为块系统

$$\underbrace{\begin{bmatrix} L & 0 \\ D & -(\alpha\eta)^{-1} \end{bmatrix}}_{M} \begin{bmatrix} v^{k+1} \\ p^{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & -G \\ 0 & -(\alpha\eta)^{-1} \end{bmatrix}}_{N} \begin{bmatrix} v^k \\ p^k \end{bmatrix} + \begin{bmatrix} f \\ 0 \end{bmatrix}. \tag{4.72}$$

矩阵 M 和N 定义了完整 Stokes 算子的分裂，

$$A = M - N = \begin{bmatrix} L & G \\ D & 0 \end{bmatrix}. \tag{4.73}$$

不动点表述

矩阵关系（4.71）定义了一种定常线性迭代，也可以等价地写成预条件 Richardson 迭代。这可直接由分裂形式得出：

$$Mx^{k+1} = Nx^k + b \tag{4.74}$$

$$x^{k+1} = M^{-1}Nx^k + M^{-1}b \tag{4.75}$$

$$x^{k+1} = x^k + (M^{-1}N - I)x^k + M^{-1}b \tag{4.76}$$

由于 $A = M - N$，因此有 $M^{-1}N - I = -M^{-1}A$，于是

$$x^{k+1} = x^k - M^{-1}Ax^k + M^{-1}b = x^k + M^{-1}\big(b - Ax^k\big). \tag{4.77}$$

定义残差 $r^k = b - Ax^k$，该迭代可简洁地写为

$$x^{k+1} = x^k + M^{-1}\big(b - Ax^k\big) = x^k + M^{-1}r^k, \tag{4.78}$$

这是标准的预条件 Richardson 形式。

在实践中，M^{-1} 的作用通过先对黏性算子L 应用多重网格求解器，再执行黏度加权压力修正来近似。由于多重网格求解并不精确，而且可能在不同迭代之间发生变化，有效预条件子会随时间改变，因此需要使用灵活方法 [112]。这种解释将 Uzawa算法视为作用于完整鞍点系统的线性定点迭代，并构成以下各节所介绍的 Krylov 和拟牛顿加速器的基础。

4.7.2 广义共轭残差法

广义共轭残差（GCR）方法 [122, 112] 是一种 Krylov 子空间算法，在每次迭代中最小化残差范数，同时支持柔性预条件。它将共轭残差法扩展到非对称系统，尤其适用于预条件子在不同迭代之间发生变化的情况。

在每一步中，该方法计算预条件化方向 $z^k = M^{-1}{}_k\, r^k$及其在算子 $w^k = Az^k$下的像，然后将 w^k 与之前计算出的向量进行正交化，以确保向量集合 $\{w^j\}$ 保持两两正交。更新解，使残差 r^{k+1} 在由 $\{z^j\}_{j\le k}$张成的子空间上达到最小，从而保证残差范数单调归约。由于预条件子 $M^{-1}{}_k$ 可能在每次迭代中发生变化，GCR 能够自然地适应上一节所述的非精确Uzawa预条件子。

完整流程总结于算法 4，其中展示了 Pyroclast 中实现的重启式柔性 GCR(m) 变体。
该公式直接源自第 4.7.1 节介绍的 Uzawa 迭代的不动点表示，

$$x^{k+1} = x^k + M_k^{-1}\big(b - Ax^k\big), \tag{4.79}$$

其中，$M^{-1}{}_k$ 表示一次 Uzawa 扫描的作用，该扫描由多重网格速度求解和按黏度缩放的压力修正组成。将 Uzawa 视为具有可变 $M^{-1}{}_k$ 的预条件 Richardson 迭代后，即可通过定义以下内容将其直接嵌入 GCR框架：

$$z^k = M_k^{-1} r^k, \qquad w^k = Az^k. \tag{4.80}$$

GCR更新

$$x^{k+1} = x^k + \alpha^k z^k, \tag{4.81}$$

因此，它对连续的 Uzawa 迭代值进行残差最小化组合。因此，GCR 作为全局线性加速器，建立在由矩阵分裂 $A = M - N$定义的局部不动点迭代之上，从而提高其收敛率。

算法 4: 柔性广义共轭残差法(m) for $Ax = b$

输入： 隐式算子 $y \mapsto Ay$，预条件子 $r \mapsto z = M^{-1}{}_k\, r$，
初始猜测 x^0,重启长度 m。

输出： 近似解 x。

初始化： $r^0 := b - Ax^0$, $k := 0$;

当 尚未收敛 **时**

  对 $i = 0$**至** $m-1$ **执行**

    // 应用变预条件子并计算搜索方向

    $z^{k+i} := M^{-1}{}_{k+i} r^{k+i}$;

    $w^{k+i} := Az^{k+i}$;

    // 改进的Gram-Schmidt正交化

    对于$j = 0$**到**$i-1$, **执行**

      $\gamma := \dfrac{w^{k+i} \cdot w^{k+j}}{w^{k+j} \cdot w^{k+j}}$;
      $w^{k+i} := w^{k+i} - \gamma\, w^{k+j}$;
      $z^{k+i} := z^{k+i} - \gamma\, z^{k+j}$;

    // 归一化新的基向量

    $\nu := \|w^{k+i}\|_2$;

    周$^{k+i} :=$周$^{k+i}/\nu$;
    $z^{k+i} := z^{k+i}/\nu$;

    // 更新解和残差

    $\alpha^{k+i} := r^{k+i} \cdot w^{k+i}$; // 因为 $\|w^{k+i}\| = 1$
    $x^{k+i+1} := x^{k+i} + \alpha^{k+i} z^{k+i}$;
    $r^{k+i+1} := r^{k+i} - \alpha^{k+i} w^{k+i}$;

    如果 已收敛 **则**
      跳出

  // 每m 步后重新启动

  $k := k + i + 1$;

与灵活广义最小残量法的比较

在灵活 Krylov 方法中，GCR 和 FGMRES [112] 都能够处理可变预条件器，并在每次迭代中最小化残差范数。FGMRES 在每次重启时都能实现最优残差归约，但需要存储完整的 Krylov 向量标准正交基，并在每一步求解一个小型最小二乘问题。相比之下，GCR 采用短递推公式，对于线性问题可实现类似的收敛

相关问题，同时实现起来更简单且内存效率更高。此外，文献表明，GCR 已成功应用于地球物理与流体动力学领域中非常相似的鞍点问题 [35, 120]。因此，对于大规模模拟或 GPU 加速模拟，GCR 尤具吸引力，因为这类模拟的内存和通信成本占主导地位。

4.7.3 安德森加速

安德森加速（AA） [123, 124]是一种非线性加速技术，通过构造先前迭代结果的优化线性组合来提高不动点迭代的收敛性。与需要显式访问算子 A 的 Krylov 方法不同，AA 完全基于不动点映射 $x^{k+1} = G(x^k)$ 运行，因此适用于线性系统和非线性系统。因此，可以将其视为一种拟牛顿方法，它利用近期迭代的信息隐式近似 $G(x)$ 的逆雅可比矩阵。

给定一系列迭代值及其残差 $f^k = G(x^k) - x^k$，该方法通过组合最近的 m 个残差历史来获取更新后的迭代值。系数通过在其总和等于 1 的约束下最小化残差范数来确定。这种最小二乘形式使 AA 能够加速收敛，而无需显式构造或应用系统矩阵。

算法5： 带混合的安德森加速（m），用于非精确 Uzawa 迭代

输入： 不动点映射 $G(x)$，初始猜测 $x^0 = [v^0, p^0]^\top$，深度 m，混合 $\beta \in (0, 1]$。

输出： 加速后的解 $x = [v, p]^\top$。

初始化： $x^1 := G(x^0)$; // 执行一次 Uzawa 迭代

对于 $k := 1, 2, \ldots$ **直至收敛 执行**

$m_k := \text{分钟}(m, k)$;

$r^k := G(x^k) - x^k$;

形式 $R^k := [r^{k-m^k} \ldots r^k]$;

求解 $\alpha^k := \arg\min_{\alpha \in A_k} \|R^k \alpha\|_2$;

其中 $A_k := \{\alpha \in \mathbb{R}^{m_k+1} : \mathbf{1}^\top \alpha = 1\}$;

// 计算混合安德森更新

$x^{k+1} := (1-\beta) \sum_{i=0}^{m_k} (\alpha^k)_i x^{k-m_k+i} + \beta \sum_{i=0}^{m_k} (\alpha^k)_i G(x^{k-m_k+i})$;

在线性情形下，当 $G(x) = x + M^{-1}(b - Ax)$ 时，安德森加速在数学上可证明等价于应用于预条件系统 [121] 的广义最小残差（GMRES）方法。然而，其完全基于不动点映射的形式使其更易应用于复杂或非线性情形，在这些情形下，算子 A 并不能显式获得。

应用于 Uzawa 迭代

Uzawa算法可写成如下不动点形式：

$$x^{k+1} = G(x^k) = x^k + M_k^{-1}\left(b - Ax^k\right), \tag{4.82}$$

其中，$M^{-1}{}_k$ 的隐式作用表示对非精确预条件器执行一次扫描，该预条件器由多重网格速度求解和黏度缩放压力修正组成。按照 [121], 中提出的方法，可将安德森加速直接应用于此不动点表述，以改善收敛性，而无需显式访问算子 A。

在每个 Uzawa 循环中，Pyroclast 计算下一个迭代值 $x^{k+1} = G(x^k)$，并存储最近的 m 组配对 $\{x^i, G(x^i)\}$，以构建算法5中所述的最小二乘系统。随后使用加速更新，在下一次 Uzawa 扫描前覆盖速度场和压力场。由于安德森加速独立于底层离散化，因此可以无缝应用于中央处理器和图形处理器求解器。在实践中，适中的深度（$m \approx 5-15$）和阻尼因子（$\beta \approx 0.5-0.8$）能够在各种黏度对比下提供稳健且一致的加速效果。

4.8 收敛性与准确性评估

要准确评估求解器的收敛性，需要采用与不可压缩斯托克斯方程物理尺度一致的残差范数。由于速度和压力的量级差异很大，直接使用相对 ℓ^2 范数在有量纲形式下并无意义。因此，Pyroclast 采用基于能量的替代范数，这些范数源自斯托克斯系统的连续算子。以下公式改编自 [110] 中介绍的方法。

4.8.1 残差范数

残差范数根据与离散动量方程和连续性方程相关的能量内积定义。它们提供了一种具有物理可解释性的收敛度量，用于量化当前迭代中尚未得到平衡的机械功所占的比例。

速度残差

按照我们对斯托克斯方程采用的符号约定，离散黏度加权拉普拉斯算子L 是对称且负定的。因此，为了获得正定能量范数，我们使用 $-L$ 及其对角替代算子 $\widetilde{L}^{-1} = \mathrm{diag}(-L)^{-1}$。这纯粹是偏微分方程系统书写方式上的约定问题，不会影响残差定义。按照这一约定，速度能量残差为

$$\|v - v^*\|_{-L}^2 = r_v^\top (-L)^{-1} r_v \;\approx\; r_v^\top \widetilde{L}^{-1} r_v = \|r_v\|_{\widetilde{L}^{-1}}^2, \tag{4.83}$$

其中 v^* 是精确速度解，r_v 是速度残差。

压力残差

对于压力场，自然能量范数由舒尔补 $S = DL^{-1}G$ （见公式（4.38））诱导。直接组装 S 是不可行的，因此我们通过正对角替代算子 $\widetilde{S}^{-1} \approx S^{-1}$ 来近似相应的能量，并定义

$$\|p - p^*\|_S^2 = r_p^\top S^{-1} r_p \;\approx\; r_p^\top \widetilde{S}^{-1} r_p = \|r_p\|_{\widetilde{S}^{-1}}^2. \tag{4.84}$$

在第 4.5.1 节中，我们分析了舒尔补的离散 缩放，以设计有效的预条件子。目标是近似三个离散算子的组合所产生的作用，而考虑网格因子是必要的，因为这些因子本质上属于这些离散算子。然而，能量范数源自斯托克斯问题的变分（连续）表述，不应编码特定于离散化的因子。因此，我们首先为舒尔补的连续等价形式设计替代算子，然后再在第二阶段对其进行离散化。

其连续形式等价于黏性加权拉普拉斯算子L

$$\mathcal{L}v = \nabla\cdot\big(\eta(\nabla v + \nabla v^\top)\big). \tag{4.85}$$

$\mathcal{L}$ 的主要物理作用是将速度梯度与同量级的黏性应力联系起来，而黏性应力的量级与压力相当。因此，其逆算子的作用正好相反，我们近似取 $\mathcal{L} \approx -\eta \Rightarrow \mathcal{L}^{-1} \approx -\eta^{-1}$。添加负号是为了考虑到，在狄利克雷边界条件下，$\mathcal{L}$ 是一个负定算子，因此其替代量也必须是负定的。请注意，这是一个连续近似解，并未考虑任何依赖于离散化的系数。舒尔补的连续等价形式为

$$\mathcal{S} = \nabla\cdot\,\mathcal{L}^{-1}\nabla \;\approx\; \nabla\cdot\big(-\eta^{-1}\nabla\big) \;\approx\; -\eta^{-1}\,\Delta, \tag{4.86}$$

因此，S^{-1} 的一个自然离散替代量是 $-\eta\,\Delta^{-1}{}_h$，其中下标h 表示算子的离散版本。最后，我们对其逆算子取对角近似：

$$\widetilde{S}^{-1} \;\approx\; \eta\ \mathrm{diag}(-\Delta_h)^{-1}, \quad \text{that is,} \quad \widetilde{S}_{jj}^{-1} \;\approx\; \eta_{jj}\,(\mathrm{diag}(-\Delta_h))_{jj}^{-1}, \tag{4.87}$$

在均匀二维网格上可化简为

$$\widetilde{S}_{jj}^{-1} \;\approx\; \frac{\eta_{jj}}{2/\Delta x^2 + 2/\Delta y^2}. \tag{4.88}$$

该结果是一个简单而优雅的替代量，在通过 η 保留物理意义的同时，仅引入了实现与分辨率无关的压力能量残差所需的最小网格依赖性。

全局相对能量残差

由于斯托克斯系统（4.31）中唯一非零的源项是体力 f，因此全局残差按相应的能量范数$\|f\|_{\widetilde{L}^{-1}}$进行归一化，从而得到替代的相对能量残差

$$\frac{\|x - x^*\|_{-L\oplus S}}{\|x^*\|_{-L\oplus S}} \;\approx\; \sqrt{\frac{\|r_v\|^2_{\widetilde{L}^{-1}} + \|r_p\|^2_{\widetilde{S}^{-1}}}{\|f\|^2_{\widetilde{L}^{-1}}}}, \tag{4.89}$$

其中，$x = [v,p]^{\mathsf{T}}$ 是近似完整状态解，而 $x^* = [v^*、p^*]^{\mathsf{T}}$ 是精确解。分块能量范数定义为

$$\|[v,p]\|^2_{-L\oplus S} := v^{\top}(-L)v + p^{\top}Sp, \tag{4.90}$$

在所采用的离散化和边界条件下，$-L$ 和 S 均为对称正定矩阵。

这提供了一种与分辨率无关且具有物理意义的收敛判据：速度项衡量尚未解析的黏性耗散，而压力项衡量按黏度加权的残余散度误差。残差值为 10^{-4} 表示仅有 0.01% 的机械功尚未平衡，从而为求解器精度提供了一种鲁棒且能量一致的度量。

4.9 数值稳定性与求解器鲁棒性

本节评估了 Pyroclast 中实现的斯托克斯求解器在强黏度对比条件下的数值稳定性和鲁棒性。分析采用了 [110], 中介绍的相同测试配置和讨论，并在此重现相应图示以供参考。

4.9.1 基准问题

为评估求解器的鲁棒性，我们考虑一个经典基准问题：高黏度球形包体在重力作用下沉入低黏度介质中。该问题的难度取决于包体与宿主介质之间的黏度对比，这会显著影响由此产生的斯托克斯系统的刚性。在本测试中，包体黏度设为 10^{26} Pa s，密度设为 3300 kg/m^3,；周围介质的黏度为 10^{18} Pa s，密度为 3200 kg/m^3,，由此得到的黏度对比为 10^8。在竖直方向施加均匀的重力加速度 $g_y = 10\text{m/s}^2$ 。计算域为尺寸为100×100km 的正方形盒体，离散为 $n_x = 501$ 和 $n_y = 601$ 个节点。半径为 20 km 的球形包体位于计算域中心。左、右边界采用 v_x 无滑移和 v_y 自由滑移边界条件；上、下边界则采用 v_x 自由滑移和 v_y 无滑移边界条件。由此产生的流场呈现出预期的致密、高黏度包体下沉运动，并在周围介质中形成两个对称的反向旋转涡旋（图4.8）。

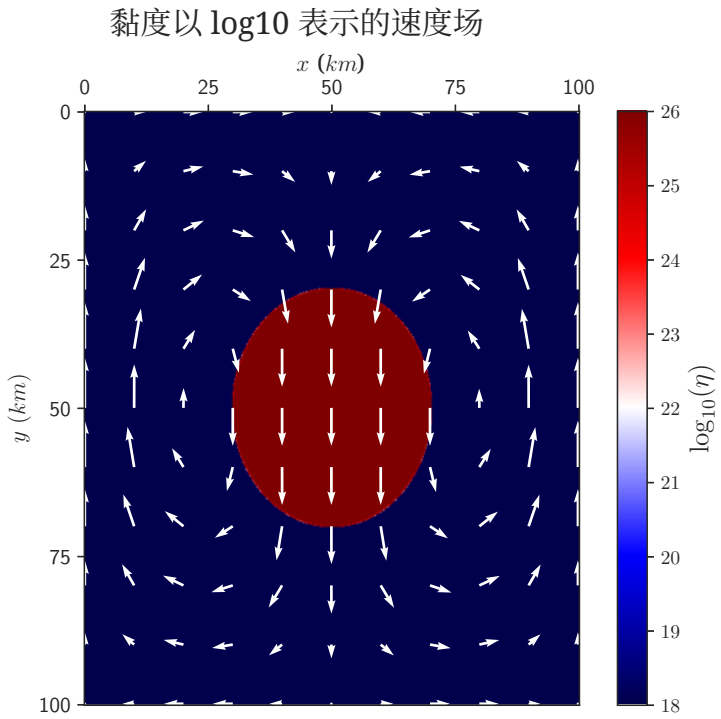

图4.8： 沉降体基准测试的速度场（白色箭头）叠加在黏度分布的对数图上，改编自 [110]。黏度为 10^{26} Pa s、半径为 20 km 的球形夹杂体在黏度为 10^{18} Pas 的周围介质中下沉。此运动在宿主材料中诱发了两个对称涡旋。

4.9.2 求解器配置

测试配置遵循 [110] 中所述的配置。共执行 1000 次 Uzawa 迭代，并禁用加速。每次 Uzawa 迭代包含一个多重网格 V 循环，该循环配置为包含四个网格层级，几何缩放因子为 2.5。在最细网格上，每个 V 循环执行五次预平滑和五次后平滑迭代；而在更粗层级上，平滑步骤数随层级递增，每层增加 2.5 倍。

为确保在如此极端的黏度对比下实现收敛，我们启用第 4.6.7 节所述的黏度重缩放，并在每 25 次 Uzawa 迭代后以 0.25 为增量，将 θ 从 0 逐步增加到 1（即 $\theta \in \{0, 0.25, 0.5, 0.75, 1.0\}$）。初始压力场设置为岩静压力分布，从而提供具有物理意义的初始猜测。

4.9.3 平滑器配置

为评估求解器鲁棒性和收敛稳定性，我们比较了四种平滑器配置：

- **红黑高斯-赛德尔：** 经典的红黑 Gauss–Seidel 方法，用作参考基线。
- **雅可比：** 简单的雅可比迭代，更易于并行化，但通常鲁棒性较差。
- **RAS：** 采用随机周期性分块移位的 RAS 型 Jacobi 平滑器（32×32 个分块，$T_{\text{内部}} = 4$）。
- **混合：** 混合设置，在最细层级使用雅可比，在更粗层级使用 RAS。

所有平滑器的速度松弛因子均为 $\omega_v = 0.3$，压力松弛因子均为 $\omega_p = 0.6$。

4.9.4 结果与讨论

图 4.9 展示了所测试平滑器的收敛历史，其依据第 4.8.1 节定义的相对能量残差进行报告。所有方法均表现出稳定收敛，残差降低了数个数量级。整体表现与 [110]中所述一致：雅可比平滑器达到最低的最终残差值，而混合配置实现了最单调、最稳定的收敛轨迹。纯 RBGS 和 RAS 变体可见振荡行为，但在整个迭代过程中均得到良好控制。

尽管 10^8, 的黏度对比非常大，所有平滑器都能收敛到 10^{-4} 或更低数量级的残差，展现出卓越的数值稳定性和

物理正确性。不同独立运行之间收敛速率的细微变化是预期之中的，因为沉降体基准测试本质上容易受到数值扰动的影响（例如线程调度、限制/延拓顺序）。为捕捉这种变异性，我们针对每个平滑器进行了十次独立运行，并在图 4.9 中报告最差情况。

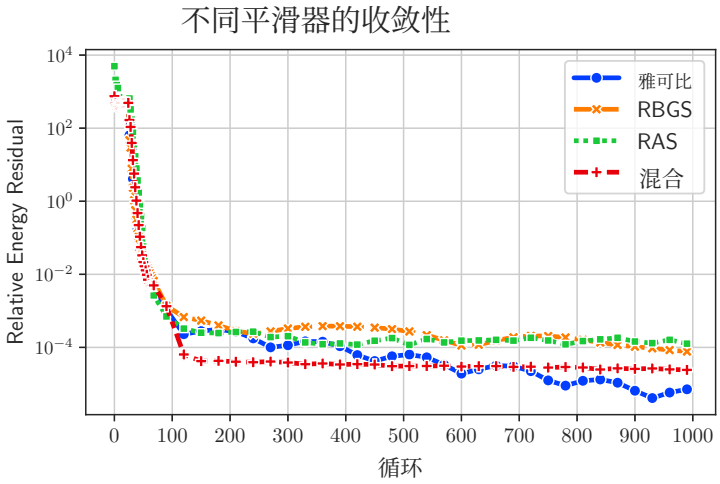

图 4.9： 沉降体基准测试中不同平滑器的收敛历史，改编自 [110]。该图展示了相对能量残差随 Uzawa 迭代次数的变化。所有平滑器均实现了数个数量级的稳健收敛。雅可比平滑器达到最低残差，而混合配置表现出最稳定的收敛行为。

所得速度场和压力场（图 4.10）在所有平滑器下均保持一致。夹杂体对称下沉，在宿主介质中形成两个方向相反的涡旋；压力场则呈现出与密度差异相关的预期垂直梯度。未观察到异常振荡或伪影。不同方法所得解的幅值和结构完全相同，证实了该实现的正确性和稳健性。

总之，基准测试结果证实，即使在黏度对比极大且没有 Krylov 加速器的情况下，Pyroclast Stokes求解器仍能保持数值稳定性并快速收敛。将黏度重标度与多重网格加速Uzawa迭代相结合，可在不牺牲精度的前提下确保稳健的性能。这些发现验证了该求解器设计，并表明其适用于大规模、高对比度的地球动力学应用。

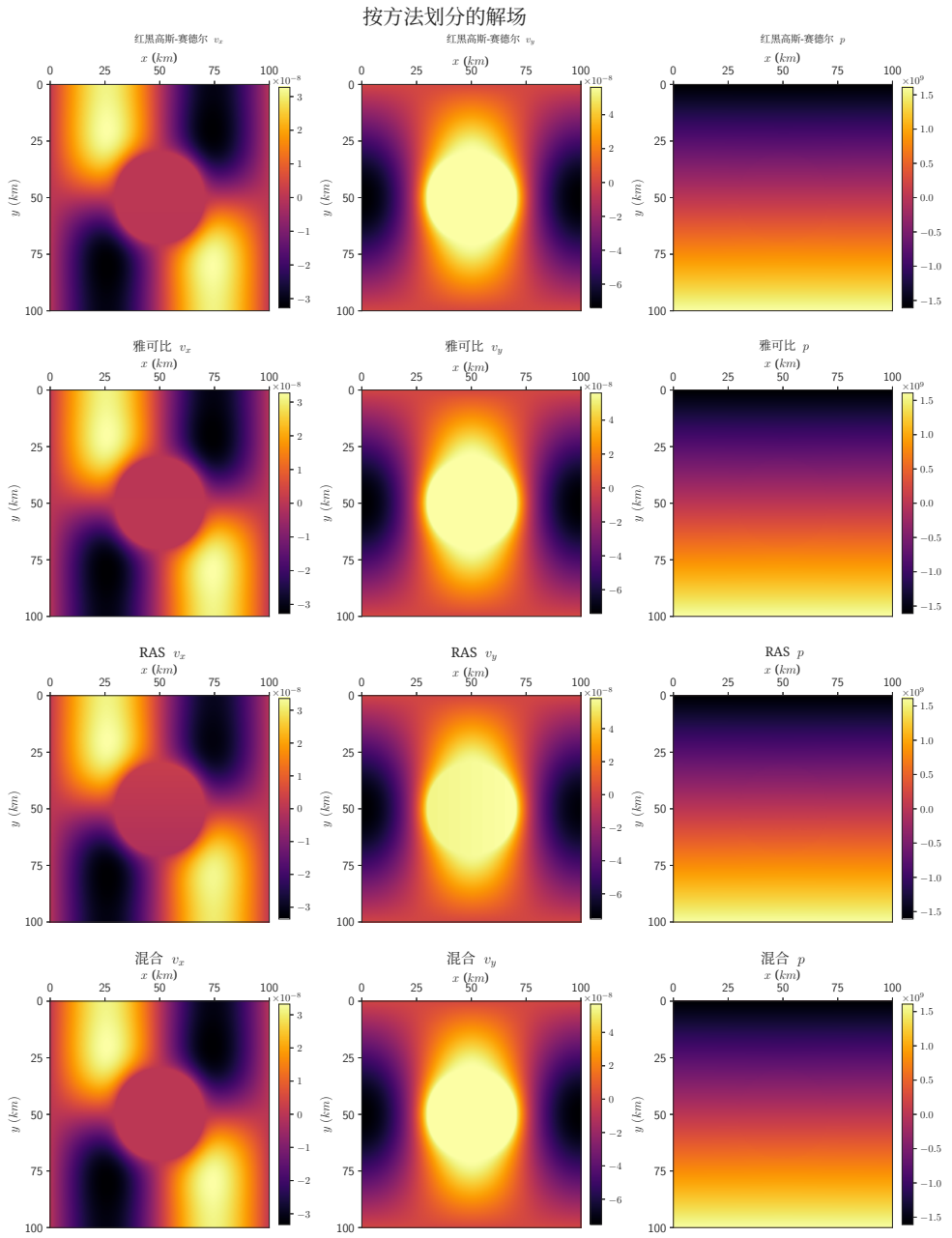

图 4.10：采用不同平滑器配置得到的速度场和压力场，转载自 [110]。每一行对应一种平滑器（红黑高斯-赛德尔、雅可比、RAS、混合），分别显示水平速度v_x（左）、垂直速度v_y（中）和压力p（右）。所有方法均得到相同的物理解，速度场光滑，且呈现出由密度差异驱动的预期压力梯度。

第5章

软件架构与模拟工作流

在上一章中，我们介绍了 Pyroclast 的基础数值方法，包括控制方程的离散化以及标记点与网格之间的耦合。这些方法定义了如何求解物理问题，但求解器的效能也取决于如何将这些算法组织到一个统一的计算框架中。本章将介绍如何通过模块化且可扩展的设计，以软件架构实现这些数值方法。

该架构的主要目标是在保持中央处理器和图形处理器上高计算性能的同时，最大限度地提高模块化程度、灵活性和可复用性。为此，Pyroclast 将模拟工作流组织为多个独立组件，这些组件可在运行时动态选择和组合。该工作流的核心协调者是主求解器，它负责维护全局模拟状态，并将操作委派给三个核心组件：网格、标记池和模型。如图5.1所示，这些组件分别封装了模拟循环中的不同阶段。这种划分体现了前文介绍的 MIC 算法的逻辑流程，也使用户只需替换组件，就能改变求解器的行为。

相同的设计原则也适用于物理模型的内部组织。Pyroclast 并非从头实现每个模型，而是采用基于继承和组合的层次结构。新模型通过扩展较简单的模型来复用现有功能，并且只添加针对新增物理过程的操作。例如，黏弹塑性模型继承热力学模型，并仅实现塑性迭代；而热力学模型本身则通过添加热耦合扩展力学模型。如图 5.2 所示，这种方法使复杂的多物理场模拟能够由少量可复用组件

组装而成。

本章其余部分将进一步阐述这些概念。首先概述组合与继承的指导原则，然后说明求解器如何协调组件之间的数据流，最后讨论高性能 Python 软件栈如何在异构计算后端上支持这种模块化设计。

5.1 设计理念

Pyroclast 的架构遵循两个核心原则：组合与继承。二者共同定义了求解器中功能的组织、扩展与复用方式。

组合 是指通过少量彼此独立、并经由定义明确的接口进行交互的组件来构建求解器。每个组件负责一项单一的概念性任务，例如管理网格场、对标记点进行平流，或求解控制方程。这种分离使各组件能够独立开发、测试和替换，从而简化实验与维护。例如，只要遵循组件接口，就可以修改插值或平流格式，而无需改动求解器的其余部分。

继承 决定了如何添加新功能。Pyroclast 不鼓励从头实现新模型，而是鼓励通过扩展基类并仅重写发生变化的方法来复用现有模型。这样便可逐步集成新的物理过程或数值策略，无需重复代码，也不会破坏现有组件。例如，新的黏弹塑性模型会继承热力学模型，并仅添加塑性 Picard 迭代所需的逻辑 [19]。

这两项原则使该架构尤其适用于大型、高性能科学代码。它们促进了物理、数值方法与流程编排之间的明确分工，从而提升可读性，并降低修改核心例程时产生副作用的可能性。它们还通过将数值算子与底层执行环境隔离，增强了后端可移植性，使相同的高层结构能够在中央处理器和图形处理器架构上高效运行。实际上，这一设计理念使 Pyroclast 能够发展为灵活的研究平台，同时保持大型地球动力学模拟所需的计算效率。

5.2 求解器编排与数据所有权

在最高层级，模拟流程由一个主求解器类负责协调，它充当整个工作流的中央控制器。它负责维护全局模拟状态、编排主要计算阶段的执行，并确保各组件基于一致的系统视图运行。所有数据、参数和时间步进信息均由主求解器持有，而各个组件则在这一共享状态上执行特定任务。

求解器将实际的数值工作委托给三个核心组件：网格、标记池和模型。每个组件都封装了 MIC方法中的一个不同步骤，同时遵循通用界面，使其能够在求解器循环中无缝交互。它们的概念性作用总结如下。

- **网格**。 网格对象定义了欧拉网格的几何形状和结构。它存储网格坐标、网格间距和网格细化等信息，并负责将材料属性从标记池插值到网格节点上。与传统的单体求解器不同，Pyroclast中的网格不会直接实现模板算子或偏微分方程求解器。它的作用纯粹是几何和组织性的，为所有基于网格的操作提供空间框架。

- **标记池**。 标记池管理携带平流材料属性的拉格朗日标记点集合。它负责将场量从网格插值到标记点，随时间执行标记点平流，并更新密度、组合或黏度等平流场。因此，标记池代表了该方法的拉格朗日部分。

- **模型**。 模型类实现了系统的控制物理。它定义了在网格上求解偏微分方程所需的模板算子，并且可能包含内部迭代循环，例如热力学—力学迭代或塑性迭代。模型使用共享模拟状态提供的网格数据进行运算，并将结果写回其中。

模拟状态类

Pyroclast 架构的核心要素之一是 模拟状态 类，它取代了单体求解器中常见的紧密耦合数据结构。在传统设计中，所有计算例程都直接操作全局定义数组。从内存管理的角度来看，这种方法效率很高，因为数据只需分配一次，便可供所有例程共享。

然而，这种做法的代价是组件之间高度耦合：每个例程都假定全局变量存在且结构固定，因此代码库僵化，难以扩展或修改，且容易影响求解器的其他部分。Pyroclast 通过将数据所有权与执行逻辑分离来解决这一局限。每个组件都基于一个共享状态对象运行，其中包含模拟不同阶段需要保留的所有数组和参数；但对这些数据的访问则通过清晰且一致的界面进行管理。

在流水线的每个步骤中，状态都会传递给负责执行特定操作的组件。该组件完成计算，并更新状态中的相关量，然后将控制权交还给主求解器对象。从这一角度看，每个组件都相当于作用于系统共享物理状态的算子。这种方法具有以下几项优势：

1. **运行时动态组合**。由于所有组件都通过同一个抽象状态界面进行交互，因此可以在运行时动态导入 Grid、Pool 和 Model 类的不同组合。这样，求解器仅根据输入文件配置，就能在不同物理模型或数值格式之间切换，无需采用通用的基类实现，也不必修改和重新编译代码。

2. **高效的内存管理**。每个组件都可以分配自己的局部临时数组用于临时计算，但只有最终结果会写回共享状态。这样，组件完成任务后便可回收其占用的内存，使求解器能够在相同的内存预算下运行规模大得多的系统。这一优势在图形处理器上尤为明显，因为可用内存有限，且内存分配会被缓存 [125]。例如，在求解热力学系统时，可以释放斯托克斯多重网格层次结构所占用的内存，并将其重新用于热求解器，从而在几乎没有额外开销的情况下，使可处理的问题规模翻倍。

3. **减少样板代码并简化可扩展性**。由于数据所有权集中管理，各组件可以专注于数值逻辑。添加新的求解器或物理模型时，只需用几行代码定义组件如何从模拟状态读取数据以及向其写入数据，无需重新定义或重新分配共享数组。

这种架构将模拟转化为一系列作用于共同演化状态的操作，其中每个组件都作为纯算子运行。它消除了对深度耦合过程的需求，使数据依赖关系显式化，并确保内存使用始终保持

高效。整体编排如图5.1所示。

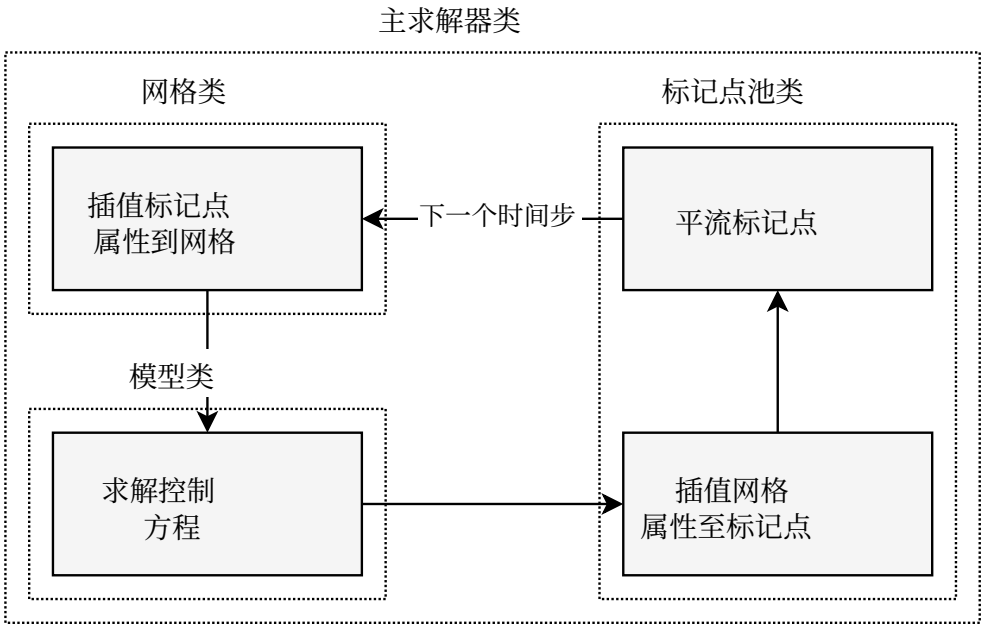

图5.1： Pyroclast 中的运行时编排与数据所有权。主求解器管理模拟状态，并将操作委托给网格、标记池和模型组件。每个组件都作为共享状态上的算子，在将状态传递到循环的下一阶段之前更新相关量。

5.3 模型的组合与继承

Pyroclast 的所有主要组件，包括网格、标记池和模型，都遵循相同的组合与继承架构原则。每一类组件都组织成由逐步专门化的子类构成的层次结构，因此能够逐步引入新功能，同时复用现有逻辑。这种设计避免了代码重复，并促进了整个求解器中一致的开发工作流。

每个组件都提供简洁且定义明确的接口，可通过扩展或与其他组件组合来增强功能。例如，网格实现可以在基础均匀网格上增加网格细化，或在其节点上纳入额外的物理量；标记池则可以实现不同的平流格式，或考虑需要输运的其他材料属性。这些层次结构彼此独立但可互操作，因为所有组件都通过共享的模拟状态以及

主求解器定义的接口进行通信。

在这三个组成部分中，继承在模型 层次结构中最直观可见，因为它直接编码了模拟的物理过程。图 5.2 以力学模型、热力学模型和黏弹塑性模型为例，展示了这一结构。**力学模型**提供基本的斯托克斯求解器，用于计算速度场和压力场。**热力学模型** 通过引入能量方程和热耦合对其进行扩展，重用力学求解器，并在两个系统之间增加迭代耦合。**黏弹塑性模型** 继承自热力学模型，并添加塑性 Picard 迭代以处理非线性屈服行为。每个后续模型仅引入新物理所需的额外功能，其余所有例程均继承自基类。

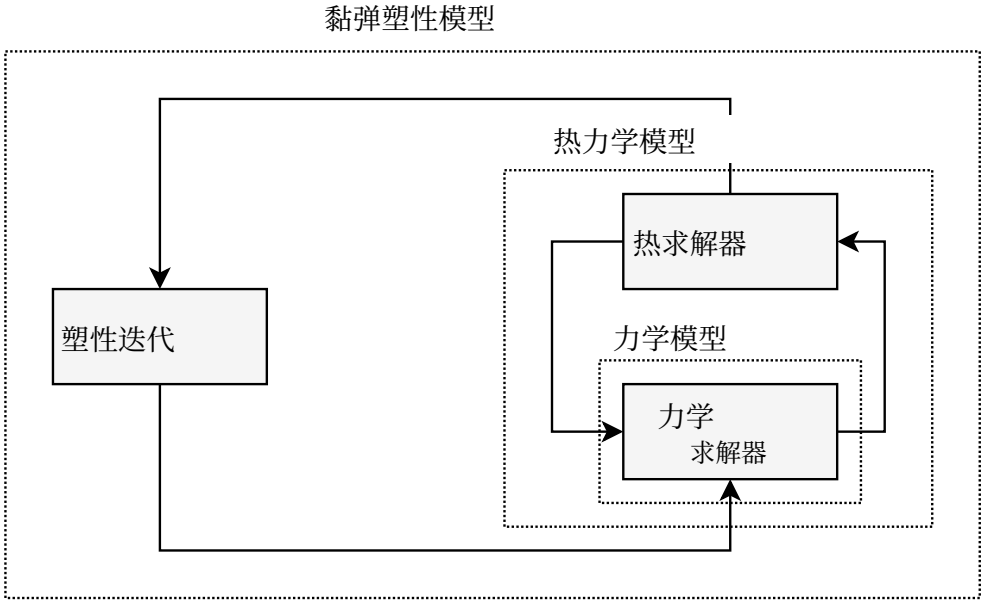

图 5.2： Pyroclast 中模型之间的组合与继承。所有主要组成部分都遵循相同的层次化设计模式。此处展示的模型层次结构说明了如何从较简单的组成部分逐步构建复杂的多物理场系统。

这种层级式组织有两个重要意义。首先，增量式设计使得开发新的物理模型变得简单，只需修改代码中少量独立的部分。其次，求解器始终具有充分的灵活性。实现新模型后，仍可通过在输入文件中选择此前定义的任何模型来实例化并运行该模型。因此，用户可以使用同一代码库运行纯力学模拟，而该代码库也支持热力学耦合或黏弹塑性

耦合。这在单体求解器中很少可行，因为添加新物理通常需要维护或重新编译代码的不同版本。

通过在所有主要组件中采用继承机制，并结合前文所述的共享状态界面，Pyroclast 兼具可扩展性和运行时灵活性。复杂的多物理场模拟可以由现有构件组装而成，而简单配置也无需额外开销即可使用。这种方法使 Pyroclast 成为一个统一且高度灵活的框架，而非一系列专用求解器的集合。

5.4 高性能 Python 软件栈

Pyroclast 的模块化架构旨在充分利用现代 Python 高性能计算生态系统。求解器不依赖手写底层内核或手动管理内存，而是基于成熟工具构建，这些工具既能在 CPU 上高效运行，也能在 GPU 上高效运行，同时保留 Python 的清晰性和灵活性。这种方法使得我们能够使用高级语言编写性能关键型科学软件，同时不牺牲数值效率或可扩展性。

计算栈的核心是 NumPy [76] 和 CuPy [77], ，它们为分别在 CPU 和 GPU 上执行稠密数组运算提供统一界面。这两个库共享相同的应用程序接口，使 Pyroclast 能够在不同硬件后端上执行相同的数值内核，避免代码重复。求解器中的所有主要数据结构均实现为基于数组的对象，因此只需选择所需后端，即可在 CPU 执行和 GPU 执行之间切换。

对于数值内核的即时编译（JIT），Pyroclast 依赖 Numba [71]。Numba 可将 Python 函数编译为优化后的机器码，适用于 CPU 和支持 CUDA 的 GPU [72]。它还支持基于 HIP 的编译，可用于 AMD 硬件 [73], ，从而确保在不同加速器架构之间具有可移植性。借助 Numba，求解器在插值、平流和无矩阵模板运算等核心计算例程中实现了接近原生的性能，同时保留了 Python 语法的可读性与灵活性。

分布式内存并行通过 mpi4py [126], 实现，它为标准 MPI 库提供了轻量且高效的界面。这使 Pyroclast 能够在多个节点上运行，并在不离开 Python 环境的情况下交换子域之间的数据。支撑单设备执行的同一模块化结构也自然扩展到了分布式环境：每个组件与共享状态的本地区域交互，同时与相邻子域交换光环数据。第 7 章将更详细地讨论这一启用 MPI 的架构。

对于需要极底层控制或与现有 C++ 和 CUDA 代码集成的操作，Pyroclast 使用 nanobind [75]。这一轻量绑定框架允许使用 C++ 编写高性能内核和外部求解器模块，并以极小的开销直接向 Python 提供这些模块。典型用例包括稀疏线性代数例程，以及难以用纯 Python 高效实现的底层优化。

通过 DLPack 标准 [127], 实现与外部框架的互操作，该标准支持数组库之间的零拷贝数据交换。这一机制使 Pyroclast 能够与 PyTorch 等深度学习库共享数据，从而集成用于参数推断、代理建模或数据驱动本构定律的机器学习模型。由于所有数据交换都在设备内存层级进行，因此不会产生中间副本，性能开销也微乎其微。

这些技术共同构成了一个协调统一且可移植的软件栈，使 Pyroclast 能够在异构系统上高效运行，同时保留本章所述的模块化面向对象结构。后续章节将重点介绍如何在这一基础上实现 GPU 加速和分布式并行。

第6章

GPU 支持与设备无关执行

上一章介绍的模块化求解器架构，为在现代 GPU 硬件上高效执行提供了天然基础。由于 Pyroclast 中的计算逻辑与数据所有权及后端特定细节完全解耦，同一套算法无需修改即可在 CPU 和 GPU 上运行。本章将说明求解器如何实现这种可移植性，并阐述现有架构中实现 GPU 加速的主要设计原则。

Pyroclast 不会用低级语言重写求解器，也不会维护多个独立实现，而是依靠设备无关的数组编程和即时编译，通过同一份高级 Python 代码面向 GPU硬件。该设计保留了模块化框架的灵活性，同时允许对性能要求较高的操作在 GPU 上原生执行。目前，Pyroclast 主要通过 CUDA运行时支持 NVIDIA GPU，并计划在未来版本中通过 HIP后端支持 AMD GPU。

本章介绍了 Pyroclast 中实现高效 GPU 执行的设计原则与实现策略。内容涵盖 GPU 实现的主要组成部分，包括提供后端可移植性的基于数组的编程模型、专为 GPU 执行设计的拉格朗日平流求解器、用于基于网格计算的模板与无矩阵内核，以及混合 GPU 多重网格实现。第8章对 Pyroclast 中启用 GPU 的组件进行了定量评估。综合运用这些技术，Pyroclast 能够充分利用 GPU 的计算能力，同时不牺牲模块化或清晰性，并为下一章介绍的分布式和多 GPU 执行模型奠定基础。

6.1 GPU 集成理念

Pyroclast 的 GPU 支持建立在两项互补的设计原则之上：设备无关数组编程和编译内核的动态分派。二者共同确保计算在适当的设备上执行，而无需重复代码或采用特定于硬件的分支。

6.1.1 设备无关的数组编程

Pyroclast 中的所有数值组件都在数组上运行，这些数组可根据所选后端存储在主机内存或图形处理器上。此功能通过基于 xp 抽象层 [128]，的统一数组界面实现，其中 xp 会动态指定为 NumPy [76] ，用于中央处理器执行；或指定为 CuPy，用于图形处理器执行 [77]。这两个库的应用程序接口几乎完全相同，因此可以在不同架构上使用相同的数组操作、广播规则和索引语义。

这种设计使数组层级的代码完全可移植。向量化运算、内存分配和逐元素计算都由后端处理，从而确保数据保留在正确的设备上。因此，求解器可以在运行时切换中央处理器和图形处理器执行，而无需修改任何数值逻辑。这种方法大大简化了数据管理，并确保模拟状态始终与当前执行设备保持一致。

6.1.2 编译内核的动态分派

虽然许多运算都可以表示为数组操作，但其他运算则需要细粒度并行化，而这难以通过高级数组语法实现。针对这类情况，Pyroclast 会动态分派即时编译内核。每个后端都会定义一组 Numba 编译函数，为不同设备实现相同的数值例程。在运行时，框架会自动导入正确的后端，并将调用分派给相应的实现。

例如，斯托克斯求解器就采用了这种模式。该求解器定义了 CPU 和 GPU 后端，为基于模板的运算（例如无矩阵内核）提供一致的接口。启用 GPU 支持时，求解器会使用 CuPy 初始化驻留于设备的数组，并将所有内核调用动态分派给 GPU 实现。在 CPU 上运行时，同一求解器实例则使用相应的 Numba 编译 CPU 内核。无论哪种情况，高层求解器代码都保持不变，所有特定于设备的细节都封装在后端层中。

这种设备无关数组编程与动态内核分发相结合，兼顾了可移植性和性能。它使 Pyroclast 能够

在维护单一统一的求解器代码库的同时，充分利用不同硬件目标的能力。

6.2 标记点平流与基于数组的 GPU 操作

标记点平流清晰地展示了 Pyroclast 如何通过结合基于数组的编程与动态分派，实现与后端无关的执行。平流方程以一系列数组操作实现，根据插值后的速度场更新标记点坐标。由于这些操作完全向量化，因此可通过统一的 xp 抽象层，使用相同的源代码在 CPU 或 GPU 上执行。后端设为 NumPy 时，计算在 CPU 上运行；设为 CuPy 时，则会根据数据布局和精度，自动将相同的操作分派至 cuBLAS 或 cuTensor 等 GPU 库。这种方法无需为平流例程分别实现 CPU 版本和 GPU 版本。

清单 6.1 展示了一个采用这种硬件无关风格编写的四阶龙格–库塔法（RK4）标记点平流积分方案示例。get xp() 函数会自动返回正确的数组后端，后续所有数组操作和插值运算都会在相应设备上执行。求解器在运行时动态选择后端时也使用同样的机制，从而确保计算过程中数据始终驻留在正确的设备上。

虽然 RK4 积分循环本身完全由数组运算编写，但插值函数展示了**动态分派**的用法。插值无法高效地表示为简单的数组表达式，因为它涉及对局部模板求值，并收集相邻网格节点的数据。为此，Pyroclast 使用 Numba 编译内核实现插值，并提供中央处理器和图形处理器版本。运行时，框架会自动导入相应的后端，并将调用分派到正确的编译函数。这样，同一个高级平流循环无需修改即可运行，而插值步骤则作为针对当前设备优化的低级并行内核执行。

这种混合方法结合了数组编程与即时内核编译的优势。高级控制流和向量化算术运算仍然可移植且易于阅读，同时通过动态调度已编译的内核实现细粒度并行性。其结果是实现了完全驻留于 GPU 的平流循环，标记点在其中进行插值和推进，无需冗余的主机与设备间数据传输。

列表 6.1： 周期域中拉格朗日标记点的硬件无关龙格–库塔平流。根据所选数组后端，相同的代码可在中央处理器或图形处理器上运行。

```
# ------- RK4          平流                    标记点
xp  =  get_xp ()      # 自动                    返回正确的数组                    后端

# 分发          将标记点坐标分发到正确的设备
xmA   =xp . 数组 ( xm )
ymA   =xp . 数组 ( ym )

# 插值例程自动                                                  在同一                  设备
vxmA   =插值_vx (xmA ,                ymA )
vymA   =插值_vy (xmA ,                ymA )

# RK4 中间阶段
xmB   =xp . mod ( xmA   +  0.5* dt * vxmA ,     xsize )
ymB   =xp . mod ( ymA   +  0.5* dt * vymA ,     ysize )
vxmB   =插值_vx (xmB ,                ymB )
vymB   =插值_vy (xmB ,                ymB )

xmC   =xp . mod ( xmA   +  0.5* dt * vxmB ,     xsize )
ymC   =xp . mod ( ymA   +  0.5* dt * vymB ,     ysize )
vxmC   =插值_vx (xmC ,                ymC )
vymC   =插值_vy (xmC ,                ymC )

xmD   =xp . mod（xmA   +dt * vxmC,     xsize )
ymD   =xp . mod ( ymA   +dt * vymC ,     ysize )
vxmD   =插值_vx (xmD ,                ymD )
vymD   =插值_vy (xmD ,                ymD )

# 计算有效速度并                                  更新标记物位置
vx_eff   =(1/6) *( vxmA   +  2* vxmB   +  2* vxmC   +vxmD )
vy_eff   =(1/6) *( vymA   +  2* vymB   +  2* vymC   +vymD )
xm   =xp . mod ( xmA   +dt * vx_eff ,   xsize )
ym   =xp . mod ( ymA   +dt * vy_eff ,   ysize )
```

6.2.1 内存约束与混合精度平流

基于 GPU 的平流面临的主要挑战之一在于标记数据的内存占用。每个欧拉网格单元通常包含 8 到 16 个标记点，这意味着基于标记点的量所消耗的内存大约比基于网格的场高一个数量级。在大型二维或三维区域中，标记点总数很容易超过可用的 GPU 内存，因此难以同时对所有标记点执行平流。

目前正在研究两种互补策略来解决这一问题。第一种是采用混合精度模拟，其中标记点的数值以单精度（FP32）存储，而网格数值仍以双精度（FP64）存储。这样可以显著减少内存使用量，同时对大多数流动状态的精度影响极小，

第二种方法是标记点分批处理，即将标记点分成适合可用 GPU 内存容量的较小组。每个批次独立进行平流和插值处理，更新后的数值再写回共享的模拟状态。分批处理会带来一些额外的内核启动开销，但可以在不超过内存限制的情况下运行规模大得多的模拟。

6.3 模板运算与 Numba 内核

基于模板的算子构成了 Pyroclast 中模型组件的计算核心。它们用于计算空间导数，并在交错网格上应用离散化控制方程，包括斯托克斯系统的动量方程和连续性方程。这些算子以无矩阵形式实现：在每个网格节点直接计算局部有限差分更新，而不是通过全局稀疏矩阵组装来完成。这种方法可将内存开销降至最低，并能在中央处理器和图形处理器上高效运行。

每个模板都表示为一个简单的逐点函数，它根据单个网格节点的局部邻域，对该节点应用离散算子。随后，使用 Numba 将此函数即时编译（JIT）为面向中央处理器和图形处理器的代码，从而为相应架构生成优化的机器码。因此，不同后端共享同一个算子定义，并根据所选执行模式在运行时导入相应的实现。

清单 6.2 展示了一个x动量算子的示例。函数 vx op point 计算在(i,j) 节点处应用与离散化x动量方程相对应的离散算子后得到的值。它使用局部黏度、速度和压力值隐式地计算结果。

清单 6.2：交错网格上x动量算子的逐点定义。该函数利用黏度、速度和压力的相邻值，计算(i,j)节点处的局部更新。

```
def vx_op_点（i,      j, dx, dy, etap,  etab,  vx, vy, p):
    # 周围的黏度            （i, j）位于交错网格上  grid
    etaA = etap [i,    j]
    etaB = etap [i,    j +1]
    eta1 = etab [i -1,  j]
    eta2 = etab [i,    j]

    # 系数
    vx1 = 2.0 *   etaA /  (dx  * dx )
    vx2 = eta1 /   (dy  * dy )
    vx3 = -( eta1    + eta2 )/( dy    * dy )  - 2.0*( etaA  +    etaB )/( dx    * dx )
    vx4 = eta2 / ( dy     * dy )
    vx5 = 2.0 *   etaB /  (dx  * dx )

    # 与.....有关的交叉项    vy
    vy1 =  eta1 / (dx  * dy )
    vy2 =-eta2 / ( dx * dy )
    vy3 = - eta1 /   (dx  * dy )
    vy4 =  eta2 / ( dx * dy)

    #  压力        梯度
    dp_右     = -p [i,  j +1] / dx
    dp_左     = +p [i,  j]   / dx

    返回    (
        vx1 * vx [i,    j -1]  +
        vx2 * vx [i -1,  j   ] +
        vx3 * vx [i,    j   ] +
        vx4 * vx [i+1,  j   ] +
        vx5 * vx [i,    j +1] +
        vy1 * vy [i -1,  j   ] +
        vy2 *  vy [i,    j   ] +
        vy3 * vy [i -1,  j +1] +
        vy4 *  vy [i,    j +1] +
        dp_右  +dp_左
    )
```

对于中央处理器执行，此函数使用 Numba 的 @njit 装饰器编译，并在标准嵌套循环中调用，该循环遍历网格的内部节点。这样便实现了向量化和多线程处理。

清单 6.3： 使用 Numba 并行循环实现的 x 动量算子。

```
vx_op_point   =nb. njit ( vx_op_point , inline =' always ', cache =True ) @nb .
```

对于图形处理器执行，使用 @cuda.jit 编译相同函数，并在 CUDA 的网格–线程块并行模型中执行。每个 CUDA 线程更新一个网格节点，而线程块组成一个覆盖整个计算域的二维网格。

清单 6.4： 使用 Numba 的 CUDA 后端实现的相同算子。每个线程根据逐点算子更新一个网格节点。

```
vx_op_point_设备        =  CUDA . 即时编译（内联=" 始终 ")( vx_操作_点）

@cuda . 即时编译
def  _vx_算子_内核 (dx ,            dy ,  etap ,  etab ,
                                    vx ,  vy ,  p ,  out ,  nx1 ,  ny1 ):
    i  =  cuda . blockIdx .y    *  cuda . 线程块维度 .y   +  cuda . 线程索引 .y
    j  =  cuda . blockIdx .x    *  cuda . 线程块维度 .x   +  CUDA . 线程索引 .x
    if  1  <=  i  <  ny1  -  1  和 1    <=  j  <  nx1  -  2:
        输出 [i,  j] =  vx_op_点_设备 (i,           j ,  dx ,  dy ,  etap,
                                              etab,   vx ,  vy ,  p)

def  vx_算子 (nx1 ,        ny1 ,  dx ,  dy ,  etap ,
                          etab ,  vx ,  vy ,  p ,  out ):
    网格 ,  块  =     启动 _2D（vx . shape）
    _vx_算子_内核 [网格,             块 ](dx,      dy ,  etap,   etab,
                                            vx ,  vy ,  p ,  输出,  nx1,   ny1 )
    返回输出
```

这种模式为在不同后端上实现微分算子提供了一种统一的方法。CPU 和图形处理器目标使用相同的算子定义和界面；两者的区别仅在于执行层。运行时，Pyroclast 会根据配置文件导入正确的后端，确保高层求解器代码保持一致。

6.4 GPU 多重网格与混合执行

实现支持 GPU 的高效多重网格求解器需要格外谨慎。模板运算受内存带宽限制，其性能很大程度上取决于局部性和数据移动。要充分利用 GPU

带宽，数据必须驻留在正确的设备上，并尽可能减少主机与设备之间的数据传输。在多重网格方法中，这颇具挑战，因为每个层级代表的问题规模都呈指数级缩小。细层级包含大部分计算工作，可受益于 GPU 并行能力；而较粗层级很快就会变得过小，无法高效利用 GPU。因此，每个层级都必须在最合适的后端上执行：细网格使用 GPU，中间层级使用多线程 CPU，最粗网格则使用串行 CPU。

简单实现通常将图形处理器视为辅助加速器。所有多重网格层级和数据都存放在中央处理器内存中；调用平滑器等计算密集型内核时，相关数组会被复制到图形处理器上。计算完成后，结果再写回中央处理器。这种卸载方法虽然简单，但效率低下，因为每次调用平滑器都会产生主机–设备传输开销。由于基于模板的平滑器受内存带宽限制，求解器的瓶颈会落在 PCIe 或 NVLink 的吞吐量上，而非图形处理器内存带宽，从而大幅削弱潜在的性能提升。

为缓解这一问题，简单实现通常会设置一个阈值，仅将足够精细、能够从图形处理器执行中受益的网格层级卸载到图形处理器上。虽然这避免了粗网格上的资源利用不足，但每个平滑步骤仍需反复传输数据，严重限制了性能。

Pyroclast 则采用 GPU 驻留式多重网格设计。层次结构分布在中央处理器内存和图形处理器内存中：选定阈值以上的细层级始终保留在图形处理器上，而粗层级则留在中央处理器上。所有驻留在图形处理器上的层级都可执行平滑、残差计算、限制、延拓和修正，无需任何主机–设备通信。只有跨设备的限制和延拓需要在中央处理器与图形处理器之间交换数据，而且这些操作针对的是规模按指数级缩小的网格，因此大幅降低了数据传输成本。

阈值决定层次结构的划分位置。所有比阈值更精细的层级都保留在图形处理器上，所有更粗的层级则留在中央处理器上。这样既能最大限度地减少通信，又可以调整划分位置，以平衡内存使用和计算负载。

图 6.1 比较了这两种策略。在朴素卸载模型（左）中，每次调用平滑器都会触发中央处理器与图形处理器之间的数据传输。在 GPU 驻留式设计（右）中，精细层级完全保留在图形处理器上，只有跨越阈值层级的限制和延拓操作会在设备之间进行，从而大幅减少通信量。

GPU 驻留式多重网格求解器的实现更加复杂，因为所有多重网格运算，包括平滑器、残差评估、限制和延拓，都必须在 GPU 上执行。它还会带来少量内存

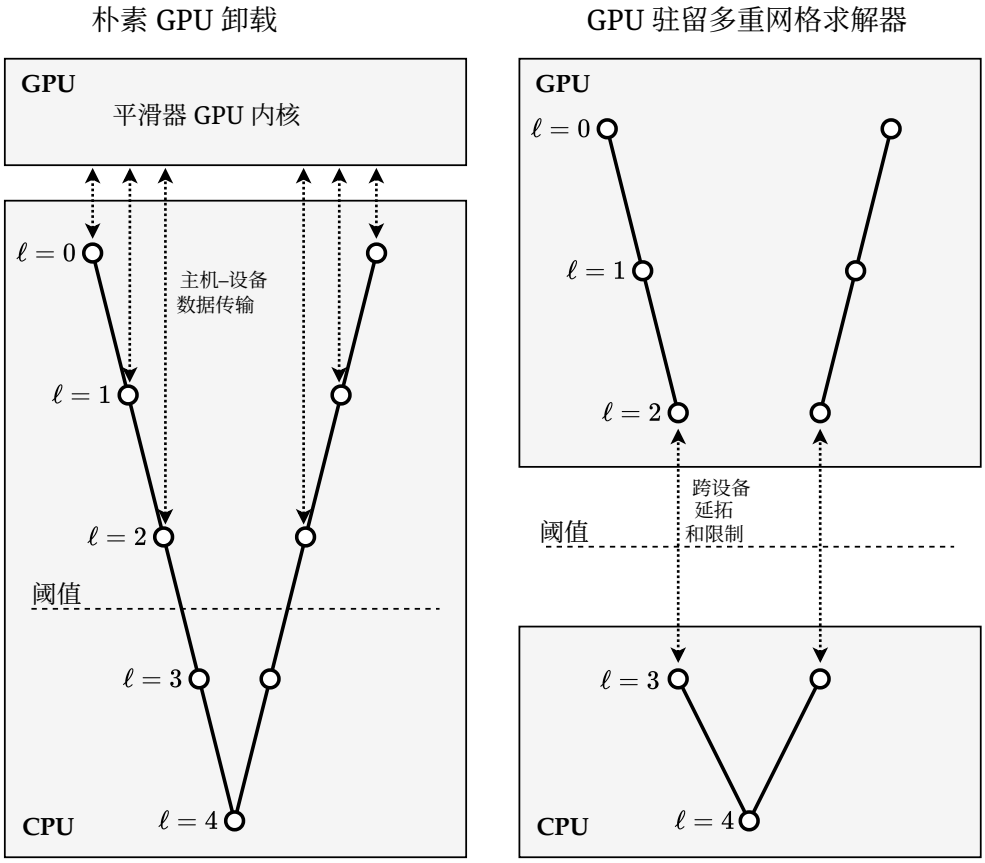

图 6.1： 朴素 GPU 卸载（左）与 GPU 驻留式多重网格设计（右）的对比。在朴素方案中，所有层级都驻留在中央处理器上，每次调用平滑器都需要进行主机与设备间的数据传输。在 Pyroclast 的 GPU 驻留式实现中，细网格层驻留在图形处理器上，而粗层级由中央处理器处理。虚线标示了区分图形处理器驻留层级和中央处理器驻留层级的阈值。跨设备数据移动仅在越过此边界进行限制和延拓时发生。

开销，因为层次结构的部分内容会同时存储在两个设备上。假设网格加密比为 2，这通常意味着二维情况下 GPU 内存额外增加约 30%，三维情况下增加约 15%。尽管有这项开销，其优势仍然显著，因为大部分计算都发生在能够充分利用 GPU 带宽的细层级上。

Pyroclast 的模块化架构使这种设计易于集成。多重网格循环只需以与后端无关的形式实现一次，并在初始化期间为每个层级分配设备。在运行时，求解器会自动导入正确的后端，并在相应设备上分配数据。只有用于平滑、限制和延拓的特定设备内核需要分别实现，并采用相同的

上一节所述的基于模板的技术。

这种混合执行模型使 Pyroclast 在实现高 GPU 效率的同时，保持设计简洁且模块化。细网格完全在 GPU 上运行，以最大限度地提高带宽利用率；粗层级则留在 CPU 上，以节省内存并避免 GPU 资源闲置。由此形成的多重网格实现灵活且可扩展，能够高效适应异构计算环境。

第 7 章

分布式执行与 MPI 并行

上一章讨论的 GPU 实现使 Pyroclast 能够利用单个节点上单个 GPU 的细粒度并行性。为了扩展到多个节点并利用由多个 GPU 组成的大型计算集群，Pyroclast 将这一模型扩展为使用消息传递接口（MPI）标准进行分布式内存执行 [129]。

MPI 提供了一种基于进程的并行编程模型，其中每个进程（或 进程）都作为一个独立的求解器实例运行，拥有自己的内存空间和计算域的本地子集。通过消息传递在各进程之间进行显式通信，即可实现并行，并在子域之间交换数据。该模型尤其适用于大规模科学模拟，因为它能够显式控制同步和数据移动，从而实现跨多个 CPU 和 GPU 的可扩展性。

在 Pyroclast 中，MPI 功能通过 mpi4py 包 [126], 实现。该包为 MPI 标准提供 Python 绑定，并公开了几乎所有相关功能，包括点对点通信、集合操作以及各种通信器拓扑。这使求解器能够保持纯 Python 接口，同时与高性能分布式系统无缝集成。

本章介绍 Pyroclast 如何通过 MPI 将其混合中央处理器–图形处理器架构扩展至分布式内存并行。讨论重点包括区域分解、光环交换以及分布式标记点平流的设计。尽管目前只有拉格朗日平流求解器完整实现了 MPI 支持，但相同的算法和设计原则也正用于将分布式执行扩展到 Pyroclast 的欧拉组件，包括插值、平滑和多重网格预条件。

7.1 MPI 编程模型概述

消息传递接口（MPI）定义了一种标准化模型，用于
分布式内存并行计算。在 MPI 中，每个进程被称为
一个 进程，作为程序的独立实例运行，并拥有自己的
本地内存空间。并行性通过显式通信来实现
通过消息传递原语在进程之间通信，而非使用共享内存
同步。该模型能够全面控制数据局部性和通信，
因此非常适用于大规模科学模拟。
最新的 MPI 5.0 标准的完整技术规范见 [129];
此处我们仅对最相关的概念进行高层次概述。

MPI 支持两种主要的通信范式。第一种是**点对点操作**，各进程通过显式的发送和接收调用直接交换数据。第二种是**集合操作**，用于协调通信器中所有进程之间的数据交换，例如将某个变量从一个进程广播给所有其他对等进程，或在整个进程组中归约数值。通信器定义了参与特定通信上下文的进程集合，而 Pyroclast 利用这一机制将进程组织成与计算域结构相对应的逻辑拓扑。

Pyroclast 中同时采用**阻塞** 和**非阻塞**通信模式。阻塞通信会确保数据传输完成后才继续执行，从而使进程之间实现严格同步。另一方面，非阻塞操作会立即返回控制权，使计算能够在通信进行期间继续执行。这实现了**异步通信**，使数据传输和计算能够在时间上重叠，从而提高可扩展性和资源利用率。非阻塞通信还支持流水线等关键优化：将一个操作拆分为多个阶段，使通信与计算重叠并交错进行。在接下来的章节中，这种模式对于在大量进程下高效扩展平流求解器尤为重要。

对于 GPU 执行，Pyroclast 依赖**支持 GPU 的消息传递接口** 实现，这些实现支持直接从设备内存进行通信。它们利用 NVIDIA GPUDirect [130] 和远程直接内存访问（RDMA） [131], 等技术，使 GPU 之间或网络接口之间能够传输数据，而无需经由主机内存中转。基于 CPU 的 RDMA 通信也使用相同机制，使 CPU 和 GPU 进程能够异步交换光晕数据或标记数据，同时继续执行其他计算。这种设计使 Pyroclast 能够在多个节点之间实现低延迟通信，并有效重叠计算与数据交换。

7.1.1 进程到设备的映射

在支持 GPU 的分布式配置中，Pyroclast 采用每个 GPU 对应一个进程的执行模式。每个 MPI 进程都绑定到特定的 GPU 设备，并负责管理所有相关数据和计算。这避免了资源争用，并使求解器能够利用支持 CUDA 的消息传递接口进行设备间直接数据传输。虽然理论上可以采用每个进程多 GPU 配置，但目前尚不支持。

7.2 结构化网格的区域分解

Pyroclast 中的分布式内存并行基于结构化域分解：全局计算网格被划分为多个子域，每个子域由一个独立的 MPI 进程管理。每个进程负责更新其所对应的解，并与相邻进程交换边界数据，以维持全局一致性。

为说明基本概念，图 7.1 展示了均匀网格的一种简单一维分解。每个进程拥有区域中一个连续的片段，其中包括边界处额外的一些网格点，这些网格点称为光晕节点。光晕中存储着相邻子域数据的副本，用于计算超出本地边界的模板运算。每个计算步骤结束后，相邻进程会交换更新后的边界数据，这一过程称为光环交换。这样可以确保在执行下一次模板运算之前，每个子域的光晕区域都包含最新值。

带光晕节点的一维分布式网格

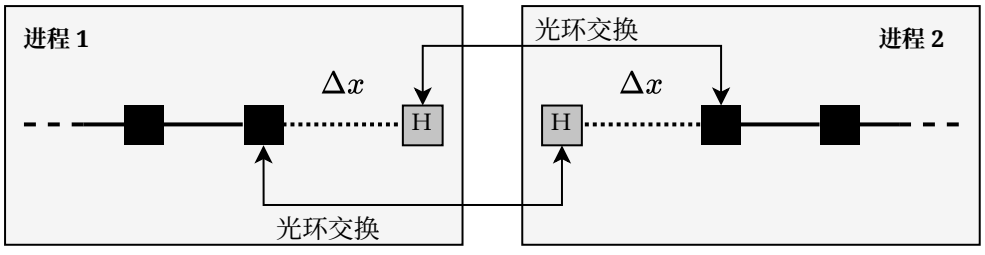

图 7.1：将结构化网格一维域分解为两个 MPI 进程。每个子域拥有其内部网格点，以及一组存储相邻进程边界值副本的光晕节点（H）。每次迭代后，数据都会通过光环交换跨界面交换。

对于交错网格，这一概念可以自然地扩展到多维。图 7.2 展示了二维区域分解，并标出了子域的不同区

域。内部区域（绿色）表示本地计算域，光晕区域（黄色）存储引用相邻进程的数据，最外层（红色）包含虚节点。虽然光晕和虚节点都是物理域之外的附加单元，但它们的用途截然不同。

光晕节点表示驻留在其他进程上、但在物理上与域内部相邻且必须在执行期间保持同步的量。读取光晕节点时，会访问属于相邻子域的值；写入光晕节点则表示更新远程数据。根据具体情况，这些更新可能是替换某个值、添加增量，或执行累加、减法等归约操作。相比之下，虚节点的存在只是为了确保所有交错量的数组形状一致。它们的用途与第 4.4.1 节所述单节点交错网格设置中的虚节点相同。虚节点使所有字段能够共用相同的数组维度，而无需在模板或插值核中引入条件分支。为方便起见，可以向虚节点写入数据，但它们从不存储有意义的物理数据，也不会参与光环交换。

在此域分解框架中，边界条件的处理方式保持一致。周期边界的实现很直接：当位于区域边缘的子域与一个越界邻居进行光环交换时，通信会被重定向到另一侧边缘上对应的子域，从而形成周期性回绕。这确保所有量在边界处保持周期连续性，无需在求解器中进行特殊处理。

无滑移和自由滑移边界条件需要格外谨慎。在初始化时，每个进程都会检查其子域是否位于物理边界上。如果是，则将子域内部向物理域外延伸一行和/或一列网格点。这样便可将光晕节点重新用作物理边界节点，从而直接在用于存储光晕数据的同一位置应用边界条件。此方法确保边界子域除了重新利用其光晕区域外，无需进行特殊处理，并使相同的模板运算能够与非周期性边界条件无缝配合

这种分解策略清晰地映射了局部数据和全局数据，有助于在边界附近高效进行模板计算，并构成 Pyroclast 中所有分布式操作的基础，包括光环交换、标记点平流和分布式多重网格。

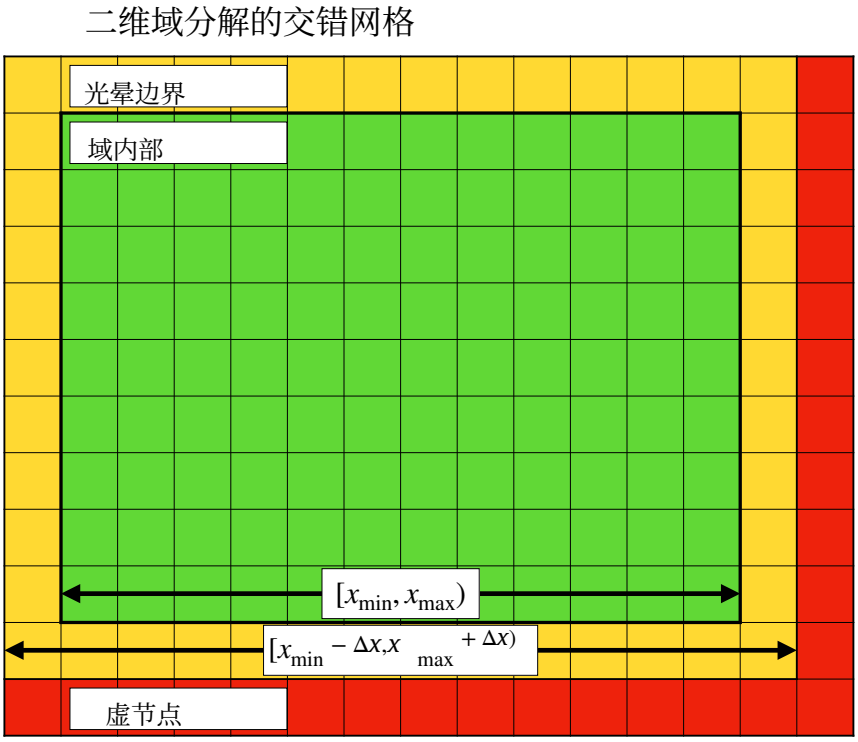

图 7.2： 交错网格的二维区域分解。每个子域都包含一个内部区域（绿色）、光晕边界（黄色）和虚节点（红色）。光晕节点存储与相邻进程交换的边界数据，而虚节点则确保所有交错场的数组形状一致，并简化网格运算。

7.3 MPI 通信器和笛卡尔拓扑

在 MPI 中，所有通信都在通信器的上下文中进行。通信器定义了一组可以彼此交换消息的进程，并为集体操作和点对点操作提供通信范围。每个 MPI 程序都从全局通信器 MPI_COMM_WORLD 开始，该通信器包含模拟中的所有进程，但也可以创建其他通信器来表示与问题结构相匹配的特定子组或拓扑。通过将进程组织到与数据布局相对应的通信器中，MPI 可以用直接对应底层虚拟逻辑域的方式来表达通信模式。

MPI 标准提供了多种通信器，用于以反映工作负载拓扑的方式组织进程。除了通用图拓扑外，MPI 还提供了一种专用于笛卡尔区域的通信器。由于 Pyroclast 处理的是矩形或长方体网格，自然的选择是使用笛卡尔通信器。

使用笛卡尔通信器有两个主要优势。首先，它能将进程直接映射到逻辑二维或三维进程网格上，使求解器可以通过子域索引 (p_x, p_y, p_z) 进行推理，而不必依赖特定于硬件的进程秩标识符。这种抽象消除了手动映射域坐标与进程秩 ID 的需要，并将计算域的虚拟布局与机器上的实际进程绑定解耦。因此，通信模式完全根据模拟域定义，不受底层硬件配置影响。

其次，由于通信器能够获知各进程在逻辑上的邻接关系，MPI 可以在初始化时自动重新排列或“重排”进程，以尽量缩短它们在硬件互连上的通信距离 [129]。因此，交换光环数据的相邻子域会在网络拓扑中尽可能靠近，从而显著降低延迟和数据传输成本。通信器还支持沿各个轴分别设置周期性，使相对边界上的子域能够自动连接，以实现周期性边界条件。

在 Pyroclast 中，这种笛卡尔拓扑构成了所有光环数据和标记点交换操作的基础。它提供了一个简洁且感知几何结构的通信层，能够自然地扩展到二维和三维问题。

7.4 分布式域中的标记点

将域分解模型扩展到包含拉格朗日标记点，需要仔细定义所有权、参考系和通信规则。与定义在固定欧拉节点上的基于网格的量不同，标记点可以在区域内自由移动，并且在平流过程中可能穿过子域边界。因此，每个 MPI 进程都必须确定哪些标记点属于其本地域，并能够将离开本地域的标记点传输给相邻进程。

为理解这一问题，不妨回顾此前图 7.1 所示的一维分布式示例。在两个子域的界面处，存在一个小间隙，对应于一个进程的最后一个节点与下一个进程的第一个节点之间的空间。从欧拉视角看，该区域不属于任一进程的内部域，但其中仍必须包含标记点，因为标记点位于网格节点之间。为避免歧义，Pyroclast 采用如下约定：每个进程负责管理其内部域以及与相邻子域共享的北侧和西侧界面间隙中的所有标记点。这样可确保全局计算域中的每个点都唯一归属于一个进程。

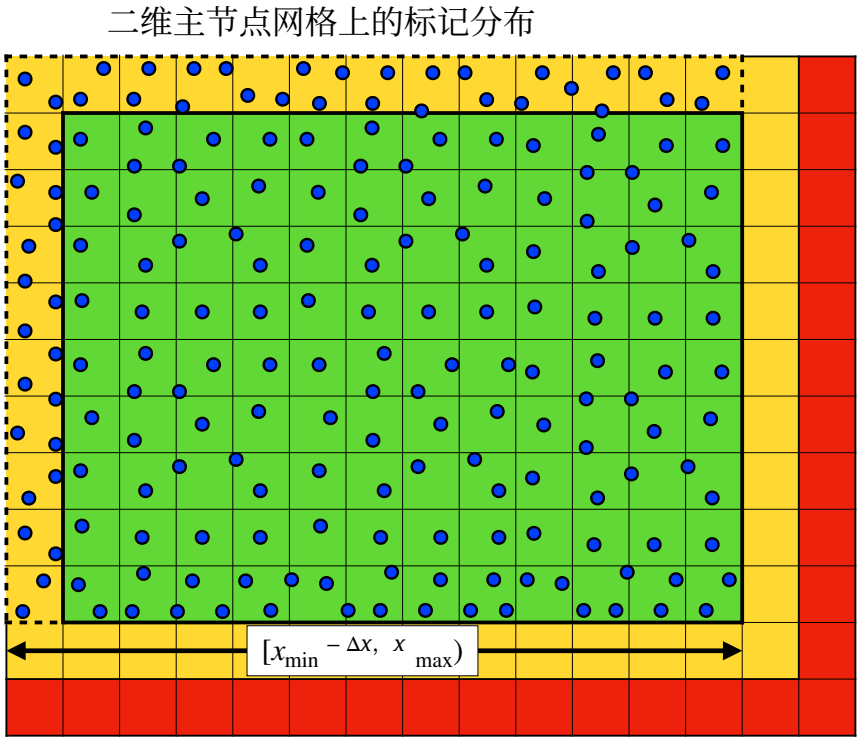

图 7.3： 主节点网格二维子域中的拉格朗日标记点分布。内部区域（绿色）包含本地标记点，此外还有标记点分布在北侧和西侧的晕区间隙（黄色）中，这些区域由此进程管理，如虚线边界所示。幽灵单元（红色）不存放标记点，仅用作填充。

图 7.3 展示了标记点相对于子域主节点网格的分布情况。标记点分布在内部域以及北侧和西侧的晕区间隙中，从而完整覆盖物理域，同时避免相邻进程之间出现重叠或冗余。当边界条件要求扩展区域时，如上一节所述，新增的网格单元将成为内部域的一部分，并相应地处理这些区域中的标记点，以确保欧拉参考系与拉格朗日参考系之间的一致性。

每个进程都在全局参考系中存储标记点坐标，与网格所用的坐标系一致。虽然也可以相对于子域原点，在局部坐标系中存储位置，但只要所有运算都使用一致的参考系，这样做并无计算上的益处。参数$x_{\min}$ 和$x_{\max}$ （以及类似地，$y_{\min}$、$y_{\max}$）表示给定进程所处理的内部域两端的坐标。在平流过程中，标记点可能会移出扩展区域 $[x_{\min} - \Delta x,\ x_{\max}) \times [y_{\min} - \Delta y,\ y_{\max})$ ，该区域包含分配给该进程的晕区间隙。发生这种情况时，该标记点的所有权

将转移给其内部域覆盖标记点新位置的相邻进程。这一迁移步骤可确保标记点在进程网格中始终得到正确分配，并确保材料属性能够跨子域边界一致地进行平流。

7.4.1 分布式标记插值

将标记点-网格插值扩展到分布式内存环境时，必须格外谨慎，以确保所有贡献都能在子域边界处正确累加。在 Pyroclast 中，分布式环境使用的插值例程与单节点求解器中使用的相同，后者已在第 4.2 节讨论。唯一的区别在于，每当数据依赖关系跨越子域边界时，都需要进行光环交换。

根据第 7.2 节提出的原则，对光晕节点的读取属于远程读取，而对光晕节点的写入则属于远程写入。这意味着，任何访问光晕区域中数据的插值操作，都隐含地需要进行光环交换，以便在相邻进程之间传播信息。

乍看之下，图 7.3 所示的标记分布似乎表明，在标记点到网格插值过程中，只有北侧和西侧的光晕区域会被修改。因此，人们可能会认为，只需将更新信息传递给北侧和西侧的相邻进程，就足以完成该操作。然而，这种推理忽略了 Pyroclast 使用交错网格布局这一事实：每个物理量都定义在不同的参考网格上，而这些网格可能沿 $-\Delta x$、$-\Delta y$ 方向或两个方向同时发生偏移。因此，每个子域实际上都包含多个重叠网格，与某个物理量相关联的标记点可能会将其贡献投射到最多四个相邻子域所属的网格点上。

如图 7.4 所示，可以通过观察压力网格上标记点的分布来理解这种情况。由于压力网格相对于主节点网格沿 $-\Delta x$ 和 $-\Delta y$ 方向错开，其关联的标记点也分别沿 $+\Delta x$ 和 $+\Delta y$方向发生偏移。因此，其中一些标记点会对局部子域边界之外的节点产生贡献，这就需要将插值结果与所有四个相邻进程交换。

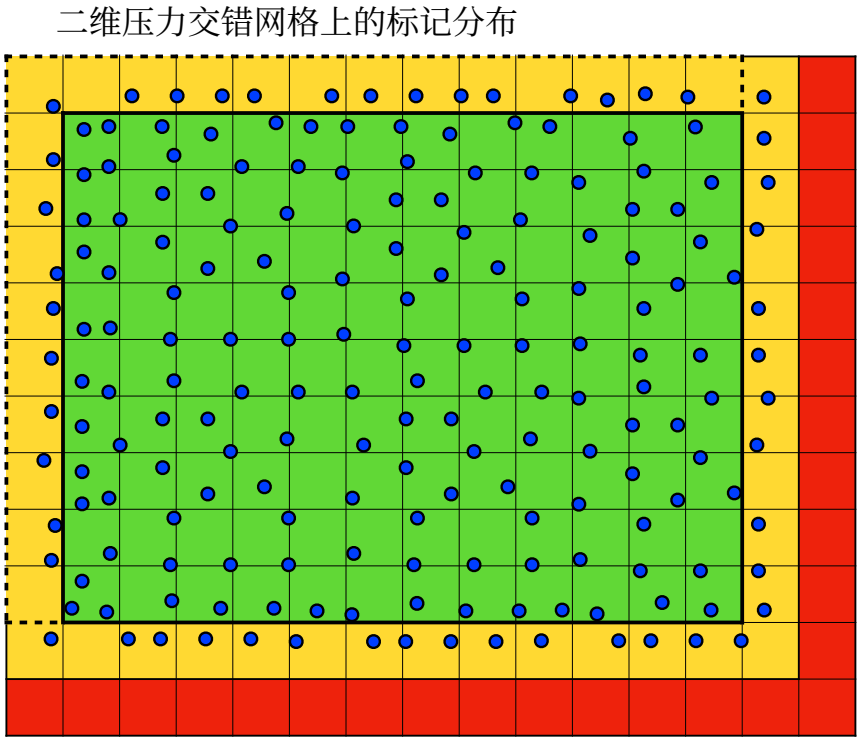

图 7.4： 标记点在二维压力交错网格上的分布。相对于主节点网格，压力网格在 $-\Delta x$ 方向偏移，在 $-\Delta y$ 方向偏移，因此标记物位置分别在 $+\Delta x$ 方向和 $+\Delta y$方向出现偏移。因此，子域边界附近的标记点会对属于所有四个相邻进程的网格节点产生贡献。所有包含标记点的幽灵单元（黄色）都需要进行光环交换，以便在子域间正确传递插值数据。

标记到网格的分布式插值

与往常一样，将标记值插值到网格上，要比将网格值插值回标记点复杂得多。每个标记点都会对其所在位置周围的网格节点产生贡献；如图 7.4 所示，其中一些节点可能属于相邻子域。因此，要确保所有贡献都能跨进程边界正确累积，就需要采用分布式归约过程。

在 Pyroclast 中，此操作通过一个两阶段流程实现，该流程确保子域间的一致性，同时与单节点插值核保持完全兼容。

阶段 1：局部累加和首次光环交换。 每个进程首先计算其标记点对局部网格上定义的加权值和权重数组的局部贡献。经过此阶段后，内部网格节点已经包含完整数据，但子域边界上的节点仅包含部分和，因为它们缺少来自相邻进程中

标记点的贡献。同时，光晕区域包含本地标记点对属于相邻子域的网格节点所产生的贡献。为协调这些部分值，将执行首次光环交换：每个进程向其相邻进程发送光晕值，并接收相应的贡献。接收到的数据会加到本地网格的内部边缘节点上，从而确保正确计入远程标记点的有效贡献。

阶段 2：归一化和第二次光环交换。收到并累积所有贡献后，每个进程都会将加权值除以累积权重进行归一化，从而得到最终的插值量。此时，光晕区域中仍然只有对远程网格节点的本地贡献。因此，需要进行第二次光环交换来同步归一化后的数据，确保所有光晕区域都反映正确的边界值。完成这一步后，每个进程都持有一份一致且完全更新的分布式网格场副本。

整体算法总结于算法 6。此设计确保分布式插值的行为与单节点实现完全相同，同时最大限度地减少冗余通信。

网格到标记点的分布式插值

网格到标记点插值比标记点到网格的情况简单得多。每个标记点根据周围的四个网格节点计算其值；只要网格场的光晕区域保持最新，这些节点就一定可在本地获取。如果场的光晕值已过时，则在执行插值前，只需对相关网格量进行一次光环交换，即可从相邻进程获取最新数据。光晕同步后，每个进程都可以使用与单域实现相同的内核，独立地插值标记值。无需进一步通信。

7.5 标记物平流策略

建立分布式插值后，我们现在可以解决跨子域边界平流标记点的问题。每个 MPI 进程负责追踪位于其区域本地部分的标记点。当标记点在平流过程中离开该区域时，必须将其转移到拥有其进入的子域的进程。此过程通过在每个平流步执行一次压缩 和迁移操作来完成。

平流后，所有标记点都会被分类为活动标记点（仍留在本地域内）或出域标记点（离开区域）。出域标记点

算法 6： 分布式标记点到网格插值
使用两阶段归约

输入： 标记物位置 (x_m, y_m)、标记值 v_m，
网格坐标 (x, y)。
输出： 分布式网格数组：网格值和网格权重。

阶段 1：局部累加
遍历 局部子域中的标记点 m **执行**
  遍历 围绕标记点 m 的节点$(i,\ j)$ **执行**
    计算插值权重 $w_m^{(i,j)}$;
    网格值$[i、\ j]$ $+=$ $v_m w_m^{(i,j)}$;
    网格权重$[i、\ j]$ $+=$ $w_m^{(i,j)}$;

首次幽灵区交换与归约
每个进程将其光晕区域（对远程网格的本地贡献）发送至
 相邻进程;
每个进程接收相应的边界区域（远程
 对其自身内部网格节点的贡献）;
将接收到的数据累加到网格值的内部边缘节点中
 和网格权重;

阶段 2：归一化与同步
归一化内部节点：网格值：$=$ 网格值/网格权重
 （逐元素）;

第二次光环交换
每个进程发送其内部区域已归一化的边界层
 区域发送给相邻进程，以便它们更新自己的光晕区域;
每个进程从邻居接收归一化后的值，并将
 这些值写入其本地光晕区域;
返回网格值;

会被打包到发送缓冲区中，并传送到相应的进程。随后，将来自相邻进程的传入标记点追加到本地数组中，并将所有活动标记点紧凑排列到连续内存中。这确保不会遗留任何非活动标记点，后续的平流和插值步骤也能在密集的数据布局上执行。

目前，Pyroclast 会在每个平流步重新分配内存，使其恰好容纳活动标记点和新流入的标记点。原则上，采用指数分配策略可以减少这项开销。在这种方案中，只有必要时才扩展标记点数组，例如扩展至原来的 1.3×–1.5× 倍；紧缩步骤则确保所有活动标记点都集中在数组的起始位置。这样可以摊销内存重新分配

成本，并提高缓存效率。

基于这一数据管理模型，可以实现两种不同的平流策略，每种策略在性能和可扩展性方面各有取舍。

受晕区约束的平流

在这种方法中，时间步受到限制，使得标记点在单个平流步内无法移出相邻子域的光晕区域。在二维情况下，这意味着标记点只能移动到八个相邻位置之一（N、S、E、W、NE、NW、SE、SW）；在三维情况下，则只能移动到二十六个相邻子域之一。通过检查标记点跨越了哪些边界来确定目标位置：例如，如果$x > x_{\max}$但$y < y_{\max}$，标记点向东移动；而同时跨越这两个条件则意味着向东北方向转移。

这种策略的优点是通信严格保持在局部范围内，因为每个进程只与其直接相邻的进程交换标记点。然而，它也对时间步施加了非物理限制。随着分辨率提高，即使稳定性或物理因素允许采用更大的时间步，最大允许时间步仍会减小。一种可行的解决方法是采用子循环，即在每个全局时间步内执行多次较小的平流更新。目前，Pyroclast 实现了这种受光晕区域约束的平流模型。

自由标记物平流

自由平流策略取消了对标记点位移的限制。每个进程都会存储所有子域的空间边界，这些边界在初始化时交换一次。平流完成后，每个标记点通过对域边界进行二分搜索，确定其新的所属进程。随后，各进程执行 MPI Alltoall 操作，交换待发送和待接收的标记点数量。此步骤只需在每个进程之间传递一个整数，因此开销很小。

确定通信模式后，每个进程都会发布非阻塞点对点传输，仅与必要的进程发送和接收标记数据。这样，标记点无需受到时间步限制，即可在单个步骤中跨越多个子域迁移。其缺点是，通信可能变为非本地通信，具体取决于进程的空间分布和硬件映射。这可能会增加延迟和数据传输成本，并需要额外的同步和簿记。虽然这种方法更加灵活，但实现起来也更复杂，并且在大规模运行中可能引发通信瓶颈。

关于分布式龙格-库塔平流的说明

在分布式环境中实现多阶段龙格–库塔（RK）平流格式，会带来单节点执行中不会遇到的若干挑战。如第4章所述，RK 格式通过在每个时间步内的多个中间位置计算速度场，提高轨迹精度。然而，在分布式环境中，这些中间位置的计算会使平流过程更加复杂，因为标记点可能会在连续阶段之间暂时离开局部子域。

在每个龙格-库塔子步中，都必须在标记点的预测位置对速度进行插值。如果该位置位于本地域之外，所需的网格值就存储在其他 MPI 进程上，因此需要额外进行光环交换或临时迁移标记点。若在每个子阶段都执行此类数据传输，将产生大量通信开销和同步操作。

因此，Pyroclast 目前在分布式平流中采用简单的前向欧拉格式，因为每个时间步只需计算一次速度。正在研究的另一种方案是第4章介绍的局部多项式积分（LPI）格式。由于 LPI 对速度场的局部泰勒展开进行解析积分，因此无需多次进行速度插值，就能近似高阶龙格-库塔格式的效果。这使其特别适用于分布式平流，因为即使标记点的轨迹暂时越过子域边界，也能正确推进标记点。

7.6 分布式多重网格与 RAS 预处理

将多重网格求解器扩展到分布式内存架构，需要平衡局部计算与全局通信。在 Pyroclast 中，这通过区域分解方法实现：每个 MPI 进程在其子域上执行局部多重网格求解，并使用与相邻进程交换的边界数据。该策略遵循限制性加性施瓦茨（RAS）预处理方法的原则 [115, 112]。

在 RAS 框架中，全局计算域被划分为相互重叠的子域，每个子域都包含一个充当缓冲区的扩展光环区域。在每个子域内独立执行局部多重网格循环，并将重叠区域边缘处的值视为固定的狄利克雷边界。经过几次局部平滑和限制-延拓步骤后，通过光环交换在各进程间合并子域的解，以维持全局一致性。受限的重叠区域可确保狄利克雷边界不会引入人工不连续性，同时让通信局限于局部并以异步方式进行。

在 Pyroclast 中，分布式多重网格实现采用混合模型：每个进程都为其本地域维护完整的多重网格层次结构，细层级可选择卸载到 GPU 上执行，粗层级则在 CPU 上执行。进程间仅在层次结构的顶层进行通信，或根据收敛率每隔几个 V 循环定期通信。这样既能最大限度地减少同步频率，又能使多重网格预处理器有效捕捉远程耦合。

为进一步减少边界伪影，可将子域间的重叠范围扩大到超过单个光环宽度。这些冗余更新将平滑区域延伸至相邻区域，形成缓冲区，从而减轻人工狄利克雷约束的影响。由此得到的预条件子高效且可扩展，兼具子域内几何多重网格和进程间加性施瓦茨耦合的优势。

Pyroclast 中的分布式多重网格与 RAS 预条件框架目前仍在积极开发中。虽然设计和通信模型已经完成开发，但尚未开展定量的性能与可扩展性测试。随着分布式求解器组件逐渐成熟，这些实验将作为未来工作的一部分。

第8章

基准测试与验证

本章介绍了一系列旨在验证 Pyroclast 正确性、稳定性和并行性能的数值基准测试。这些基准测试涵盖了在中央处理器和图形处理器架构上对斯托克斯求解器进行的共享内存执行，以及使用消息传递接口对拉格朗日平流求解器进行的分布式内存执行。所有测试均在相同的硬件和软件环境下进行，以确保条件一致；除非另有说明，所有运行均采用双精度。

此处展示的结果代表了当前 Pyroclast 实现的初步性能评估。尽管目前仍处于开发周期的早期阶段，但基准测试已展现出极佳的数值稳健性，以及在中央处理器和图形处理器架构上的良好可扩展性。仍需进行大量优化和调优，尤其是针对 GPU 内核、多重网格层级管理和分布式内存组件。未来的工作将着重提升性能可移植性，并扩展基准测试套件，以涵盖完全耦合的热力学-力学模拟。

8.1 硬件和软件设置

所有基准测试均在瑞士国家超级计算中心（CSCS）的*Piora* 集群上执行。该集群由七个通过 InfiniBand 网络连接的计算节点组成。该系统采用异构架构，包括两个配备新一代图形处理器的节点和五个搭载较早一代中央处理器的节点。

两台新节点配备双路 AMD EPYC 7773X 处理器（每个插槽 64 个核心，共 128 个核心），每台搭配四块 NVIDIA A100 GPU 和 1 TB 系统内存。其余五台节点采用双路 AMD EPYC 7742 CPU，共 128 个核心，内存为 512 GB。

InfiniBand 互连速度因节点而异：较旧节点为 100 Gbit/s，较新节点为 200 Gbit/s，分别对应 Mellanox MT4115 和 MT4123 主机适配器。

所有计算均在统一的软件环境中进行，该环境基于 NVIDIA PyTorch 容器 [81]，，并通过 Apptainer [78, 79]运行。容器包含 CUDA 12.9 和现代化的 Python 高性能计算软件栈。Pyroclast 使用的关键软件包包括 NumPy 1.26.4、CuPy 13.2、Numba 0.62、Numba-Cuda 0.20.0、mpi4py 4.0.3 和 nanobind 2.2.0。MPI 通信由 Open MPI v4.1.7 提供，通过 PMIx 初始化，并由 Slurm 管理。Numba 中的多线程由 OpenMP 后端处理。

所有运行均采用显式进程绑定，以确保对进程放置和内存局部性的全面控制。CPU 亲和性和 NUMA 绑定通过 Slurm 以及相应的 OpenMP 环境变量强制设置。每项基准测试在计时执行前都包含专门的预热阶段。基准测试运行期间，求解器执行多次迭代，并记录所有迭代的总运行时间。这种方法最大限度地减少了瞬态效应的影响，并能稳定测量持续性能。所有测试在硬件配置相同的不同节点上均产生了确定且可复现的结果。

该环境提供了一个具有代表性的 CPU–GPU 混合平台，配备现代编译器、通信库和运行时管理，确保基准测试能够准确反映 Pyroclast 在当前高性能计算硬件上的性能特征。

8.2 共享内存 Stokes 求解器基准测试

本节评估斯托克斯求解器在共享内存环境中的性能，重点考察单个计算节点内的 CPU 和 GPU 执行。目标是评估 Pyroclast 对大型二维系统中的线程级并行性和硬件加速的利用效率。所有测试均在第 4.9 节数值稳定性研究所采用的相同配置下进行，并使用沉降体基准测试作为具有代表性的黏弹力学问题。

斯托克斯求解器基准测试主要针对三个方面：网格分辨率不断提高时的收敛行为、跨 CPU核心 的强扩展性能，以及与多核 CPU 相比，在图形处理器上运行可实现的加速比。这些实验共同初步评估了 Pyroclast 的无矩阵多重网格实现的计算效率，以及其是否适用于高分辨率地球动力学模拟。

8.2.1 基准测试设置

基准测试配置遵循第 4.9 节所述的沉降体问题。所有材料参数、边界条件和求解器设置均与数值稳定性测试中使用的设置相同，以确保验证与性能分析的一致性。

测试了四种网格分辨率： 2500×2500、 5000×5000、 10000×10000 和 15000×15000。对于每种分辨率，多重网格层次结构均配置为包含六个网格层级，并在运行时确定几何缩放因子，使最粗网格约包含 30×30 个节点。所有层级均使用标准雅可比平滑器，最细网格上进行五次预平滑和五次后平滑。更粗层级上的平滑迭代次数采用与第 4.9 节相同的缩放策略。求解器运行 500 次 Uzawa 迭代，每次迭代包含一个多重网格 V 循环和一次压力修正，并记录总运行时间。

所有测试均使用标准雅可比平滑器。图形处理器基准测试在较新的 AMD EPYC 7773X 节点上运行，使用 32 个 CPU 线程。仅使用中央处理器的基准测试则在较早的 AMD EPYC 7742 节点上进行。硬件差异对性能的影响可以忽略不计，因为斯托克斯求解器主要受内存带宽限制，并且在两种处理器上表现出相似的扩展特性。图形处理器阈值设为 2000×2000。

8.2.2 收敛

在分析性能之前，我们首先验证斯托克斯求解器在所有基准配置下都表现出与分辨率无关的收敛性。观察到的行为与第 4.9 节所呈现的结果高度一致，其中展示了在极端黏度对比下的稳健性。

图 8.1 报告了所有测试网格分辨率下，经过 500 次 Uzawa 迭代后的相对能量残差。在每种情况下，求解器都能达到约为 10^{-4} 数量级的最终残差，对应的相对能量误差约为 0.01%。四种分辨率下的收敛轨迹几乎完全一致：衰减速率和最终误差水平实际上都与网格间距无关。这证实了 Pyroclast 所采用的替代残差度量能够按预期工作，并且多重网格层次结构和平滑器配置能够产生一致的、与分辨率无关的收敛。

这些结果表明，所有测试案例都收敛到相同的物理解，且数值稳定性没有下降。

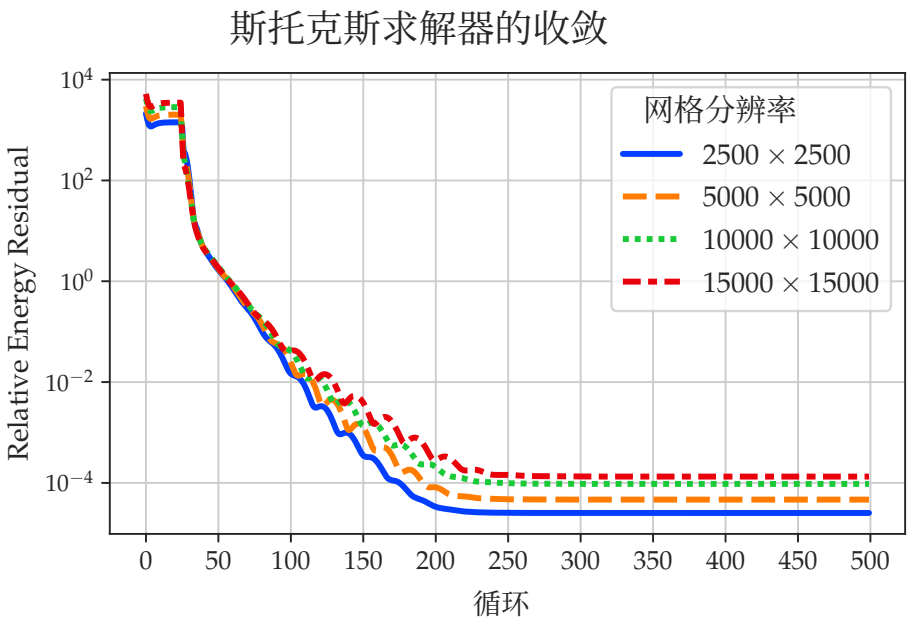

图 8.1：四种网格分辨率下，500 次 Uzawa 迭代期间的相对能量残差。所有运行均收敛至 $\sim 10^{-4}$，且轨迹几乎完全相同，表明求解器性能与分辨率无关。

8.2.3 强扩展性能

为评估斯托克斯求解器的共享内存并行效率，我们使用 1 至 128 个 OpenMP 线程，在四种网格分辨率（2500×2500、5000×5000、10000×10000 和 15000×15000）下测量强扩展。图 8.2 展示了每种配置下 500 次 Uzawa 迭代的总运行时，图 8.3 则展示了相应的并行加速比，其定义为$S^p = T_1/T_N$，其中 T_1 表示使用 1 个线程时的运行时，T_N 表示使用 N 个线程时的运行时。

结果表明，OpenMP 在所有分辨率下的扩展性都较为有限。这一表现符合预期：多重网格 V 循环主要受内存受限的模板运算支配，其算术强度太低，无法充分利用大量 CPU 核心。随着线程数增加，性能很快会受到内存带宽以及缓存和非统一内存访问（NUMA）相关影响的限制。因此，加速比在约 32–64 个线程时有所提升，但到了 128 个线程时便停滞不前，甚至略有下降，这与双路系统的特性一致 [110]。

尽管存在这些限制，绝对运行时间仍具有竞争力，而且求解器在经过测试的最高核心数下仍能保持稳定的性能。我们预计，三维问题的强扩展效率会显著提高，因为每个网格点增加的计算工作量会带来更高的算术强度，并能更充分地利用共享内存并行性。

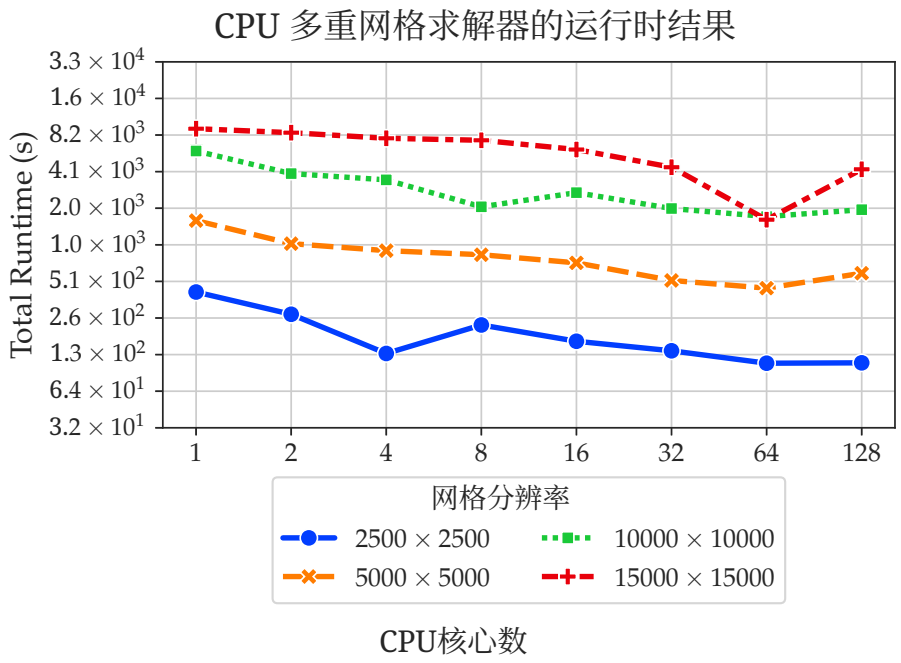

图 8.2： 共享内存斯托克斯求解器在四种网格分辨率下执行 500 次 Uzawa 迭代的总运行时。性能随线程数增加而提升，但由于多重网格内核受内存限制，加速比有限。

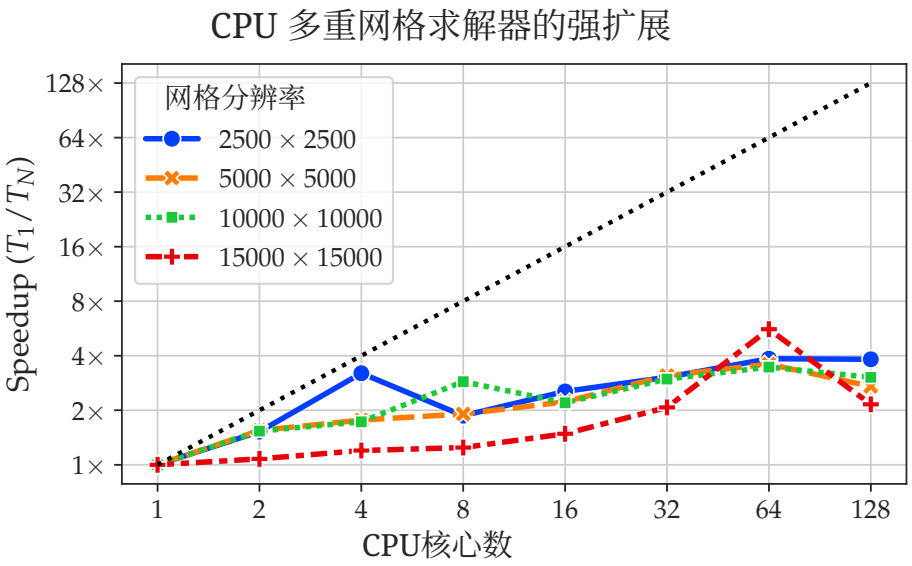

图 8.3： 共享内存斯托克斯求解器的强扩展。加速比随线程数增加而提升，达到 32–64 个线程后，由于内存带宽限制而趋于饱和。虚线表示理想线性加速比。

在 64 个 CPU 核心时， 15000×15000 配置出现了一个有趣的异常值，其加速比显著高于相邻线程数或较小问题规模下的表现。目前尚未完全理解这一现象的根本原因。不过，多次运行以及在硬件相同的不同节点上均稳定复现了这一结果，这表明它确实是性能特征，而非测量误差。还需进一步分析，以确定这一峰值是否由有利的非统一内存访问效应、缓存行为或其他架构交互所致。

8.2.4 GPU 加速与中央处理器比较

为评估 GPU 加速带来的收益，我们比较了单个 GPU 实现（使用一块 NVIDIA A100）的运行时与每种网格分辨率下性能最佳的多线程中央处理器配置。图 8.4 展示了 GPU 的绝对运行时间（右轴），以及 GPU 相较于中央处理器最快运行结果的加速比（左轴）。

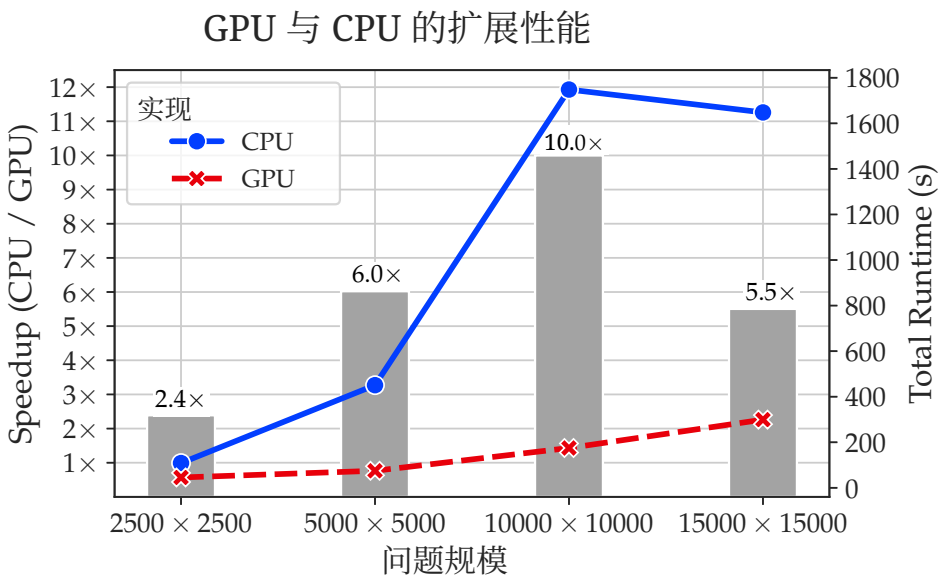

图 8.4： 针对每种网格规模，GPU 加速与性能最佳的中央处理器配置的对比。折线表示绝对运行时间（右轴），柱状图表示 A100 相较于中央处理器最佳运行结果的加速比（左轴）。GPU 效率随问题规模增大而显著提升，在 10000×10000 网格上达到 $10\times$ 的加速比峰值，这体现了设备驻留型模板运算和高内存带宽的优势。

对于较小到中等的网格尺寸，图形处理器未得到充分利用： 2500×2500案例仅实现了约 2.4×的温和加速比。随着问题规模增大，图形处理器的性能显著提升。加速比峰值

达到 10×（对应 10000×10000 网格），这体现了 A100 高内存带宽和设备驻留式多重网格执行的出色表现。15000×15000 案例仍显示出 5.5×的可观加速比，不过，由于此特定配置下观察到的中央处理器运行时异常快，该数值被人为压低。图形处理器运行时随问题规模持续平稳扩展，这表明加速比下降是由中央处理器的异常值所致，而非图形处理器方面的任何限制。

总体而言，结果证实，GPU 实现对于大规模斯托克斯问题具有很高的效率。完全驻留于 GPU 的多重网格层次结构避免了 PCIe/NVLink传输，并能够充分利用 GPU 的内存带宽，加速占据运行时开销主要部分的模板运算。这些发现凸显了问题规模的重要性：小网格可能无法充分利用 GPU，但大网格则恰好处于 GPU 加速能够带来显著性能提升的范围内。

8.3 分布式平流基准测试

本节评估 Pyroclast 基于消息传递接口实现的分布式拉格朗日函数平流求解器的性能与可扩展性。尽管欧拉斯托克斯组件的完整 MPI支持仍在开发中，平流求解器已具备完整的分布式实现，因此是评估多个节点间并行效率的理想测试案例。

我们的基准测试侧重于弱扩展行为，即全局问题规模随 MPI 进程数量成比例增加，从而使每个进程的计算负载保持不变。这一选择反映了基于标记点的平流在大规模地球动力学模拟中的预期用法：标记点总数会随区域大小增长。

这些测试考察了在并行性不断提高且集群硬件异构程度不断增加的情况下，光环交换、标记点迁移和基于模板的插值的表现。所有测量均采用第 8.1 节所述的相同预热流程和绑核策略，以确保结果一致且可复现。

8.3.1 基准测试设置

为评估 Pyroclast 的分布式标记点平流能力，我们进行了弱扩展性基准测试，为每个计算节点分配固定的本地问题规模，即 5000×5000 个网格单元。因此，全局计算域的规模会随着节点数成比例增加：

$$5000 \times 5000, \quad 7100 \times 7100, \quad 10000 \times 10000, \quad 13300 \times 13300$$

分别在 1、2、4 和 7 个节点上运行。这样可以确保每个进程的计算负载保持不变，从而测量通信、光环交换和标记点迁移的可扩展性。

该基准问题由一块在给定无散速度场作用下发生平流的黏性材料旋转板状体构成。速度场在区域中心周围诱发平滑的旋转运动，背景材料中嵌有一个高黏度矩形夹杂体（图 8.5）。虽然处理恒定速度场时并不需要标记点到网格的分布式插值，但我们仍然像在平流步骤之间求解斯托克斯系统那样执行标记点插值。这一配置提供了一个受控测试，其中标记点轨迹不断穿越子域边界，确保基准测试同时考验光环交换和标记点压缩。

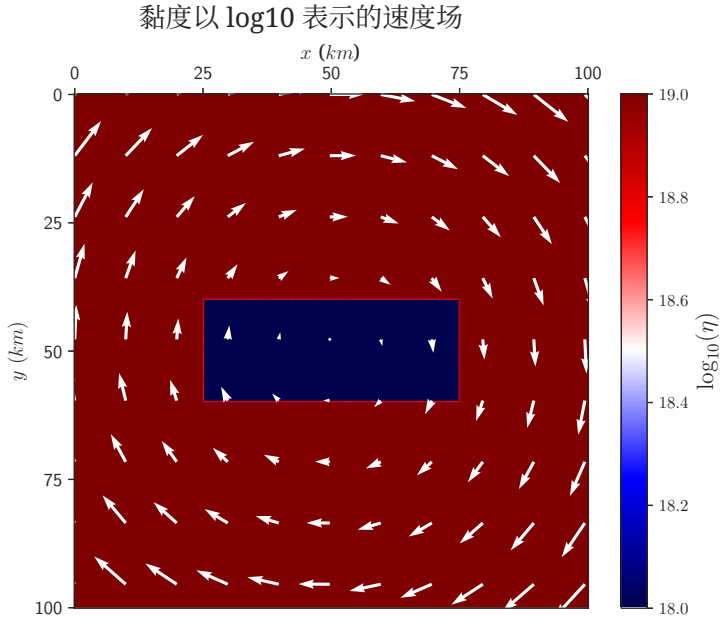

图 8.5： 分布式平流测试的基准配置：叠加在含有高黏度矩形夹杂体的黏度图上的旋转速度场。平滑的旋转流动使标记点迁移并穿越子域边界，从而为测试分布式平流、光环交换和标记点迁移提供了受控条件。

模拟在最多七个节点上进行（总计896个CPU核心），采用混合MPI+OpenMP并行性。对于每种节点数量，我们测试了三种不同的分解策略：

$$8 \times 16, \qquad 16 \times 8, \qquad 32 \times 4,$$

其中该记法表示每节点 MPI 进程数 $\times$每进程 OpenMP 线程数。每个 MPI 进程恰好拥有一个具有非周期边界的子域，

标记点平流则在中央处理器上使用显式欧拉法执行。

所有运行均采用第 8.1 节所述的相同预热和绑核策略。每项基准测试包含连续进行的 20 次平流迭代，我们报告这 20 步的总运行时间。所有测试均具有确定性且可复现。

8.3.2 弱扩展结果

图 8.6 报告了节点数增加时，三种 MPI–OpenMP 配置下分布式平流基准测试的总运行时间。总体而言，平流求解器的弱扩展行为表现出色：对于性能最佳的配置（32×4），模拟从单个节点上的 5000×5000 扩展到七个节点上的 13300×13300 时，运行时间几乎保持不变。这表明，基于 MPI 的光环交换和标记点迁移机制能够在集群中高效扩展。

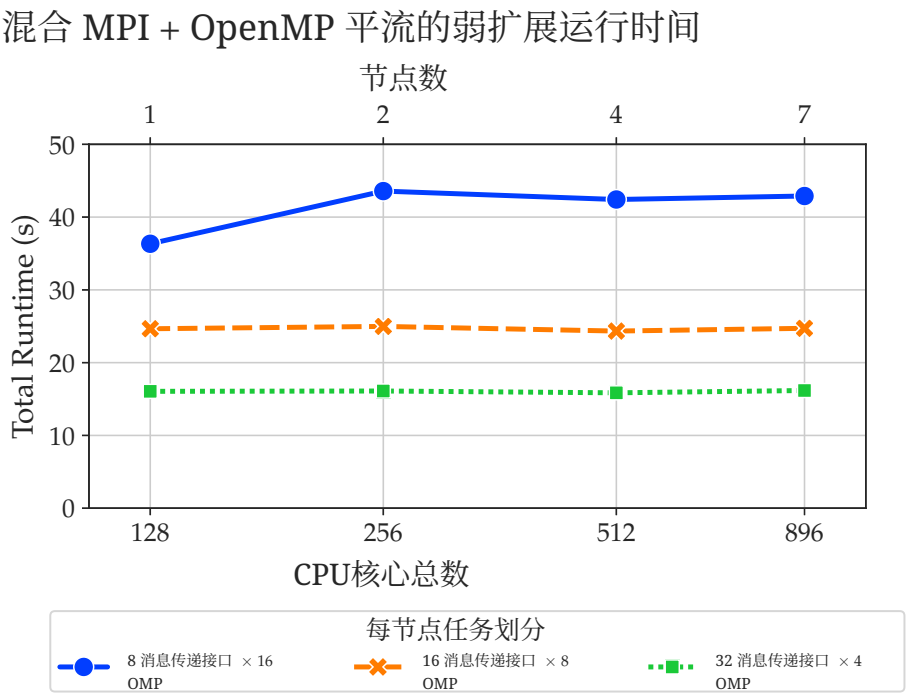

图 8.6： 分布式平流基准测试的弱扩展运行时间。每个节点处理固定的 5000×5000 子域，而全局计算域随节点数增加而扩大。每个 MPI 进程使用较少 OpenMP 线程的配置性能最佳，在最多七个节点时运行时间几乎保持不变。

结果的一个显著特点是性能对 OpenMP

并行性的敏感性。与共享内存 Stokes 基准测试类似，平流求解器无法从增加 OpenMP 线程数中显著获益。相反，增加线程数往往会使本地工作速度以近乎恒定的比例下降。主要原因是标记点到网格的插值例程依赖原子归约来累积节点权重。这些归约会引发争用，尤其在线程数增加时，从而限制 OpenMP 可扩展性。相比之下，MPI 级分布式插值不会受到这种争用的影响，因此在各节点间具有出色的可扩展性。

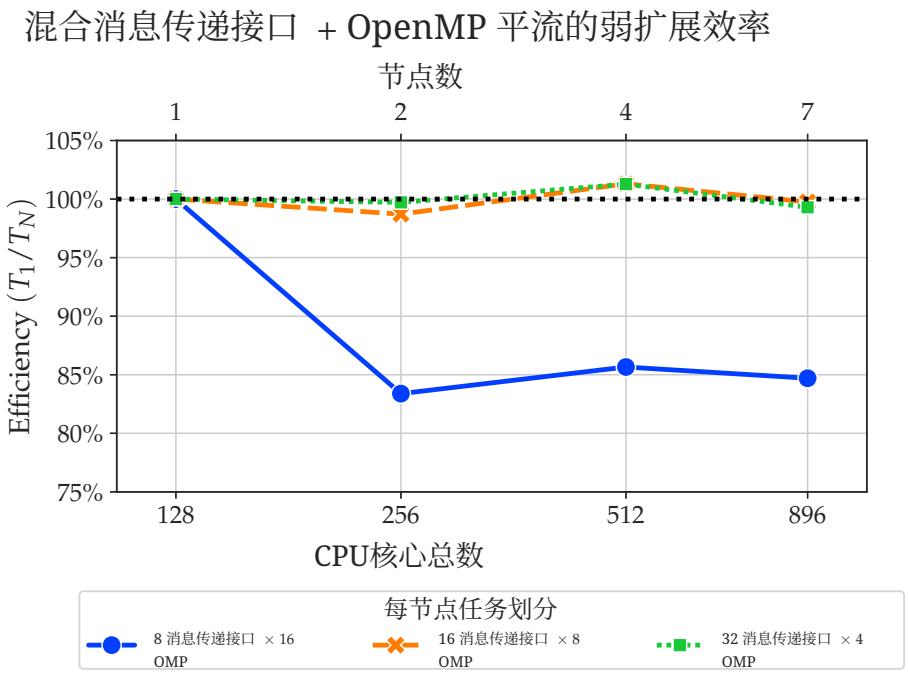

图 8.7： 分布式平流基准测试的弱扩展效率。 16×8 和32×4配置在所有节点数量下都保持接近 100% 的效率。中等规模下略微出现的超线性扩展，可能反映了缓存效应或轻微的运行时间波动。

图 8.7 展示了归一化的弱扩展效率，计算方式为 T_1/T_N，其中 T_1 是单节点运行时间，T_N 是运行在N 个节点上的运行时间。 16×8 和 32×4 配置的效率均接近 100%，在四个节点时略微呈现超线性表现。这一小幅提升可能源于缓存效应或运行时间测量中的统计波动；这两种情况在分布式内存扩展测试中都很常见。 8×16 配置的效率较低，这反映了上文提到的同样与 OpenMP 相关的局限性，但总体扩展效率仍保持在 80% 以上。

综合来看，这些结果表明，只要

MPI/OpenMP 配比避免过度使用 OpenMP 线程，Pyroclast 中的分布式平流框架就能在现代高性能计算集群上实现高效扩展。未来改进插值核，或可进一步提升共享内存扩展能力，并缩小不同混合配置之间的性能差距。

8.4 讨论与总结

本章展示的基准测试初步评估了 Pyroclast 在多种架构和问题规模下的数值稳健性与计算性能。尽管目前仍处于开发初期，结果已展现出该框架的若干重要优势，同时也明确指出了未来优化的空间。

斯托克斯求解器性能

共享内存基准测试表明，Pyroclast 的无矩阵多重网格斯托克斯求解器在所有测试的网格尺寸下均能实现稳定且与分辨率无关的收敛。该求解器始终达到相对能量残差 $\mathcal{O}(10^{-4})$，并在不同分辨率下保持完全一致的收敛轨迹，证实了多重网格层次结构和 Uzawa 迭代的正确性与稳健性。

CPU 上的强扩展结果体现了模板主导的多重网格内核具有内存受限的特性：随着线程数增至 32–64，性能加速比有所提升，随后因带宽限制和非统一内存访问效应而趋于饱和。这种表现符合二维结构化网格求解器的预期，也与文献中的发现一致 [110]。在 15000×15000 情况下，64 个线程时观察到一个有趣的性能异常：多次测试均获得异常高的加速比。还需进一步分析，才能确定这种表现是否源于有利的内存放置、缓存效应或其他架构因素。

GPU 加速对于大型系统效果显著。由于阈值以上的整个多重网格层次结构都驻留在设备上，A100 GPU 能够充分利用高内存带宽，并避免主机与设备之间的数据传输。随着问题规模增大，GPU 利用率显著提升，相较于速度最快的 CPU 配置，加速比峰值达到 $10\times$。即使是最大的测试用例也表现出明显的加速；观测到的加速比下降并非 GPU 的局限所致，而是因为 CPU 测量结果异常快。这些结果证实，Pyroclast 的 GPU 后端已能为大型斯托克斯系统带来显著的计算优势。

分布式平流性能

分布式内存基准测试表明，在混合 MPI+OpenMP 配置下，Pyroclast 的标记点平流基础设施能够在多个节点间高效扩展。对于性能最佳的分解方案，弱扩展性表现出色：运行时在最多七个节点、896 个 CPU核心的情况下仍几乎保持不变。即使全局计算域规模不断增大，MPI 级光环交换和标记点迁移机制仍展现出稳定且可预测的性能。

OpenMP 可扩展性较为有限，这反映出共享内存的标记点到网格插值例程存在瓶颈。当许多线程同时操作相邻网格节点时，原子归约会引发争用，导致线程数增加时性能几乎持续保持在较低水平。尽管如此，求解器的 MPI 组件几乎实现了理想的扩展性；对于每进程 OpenMP 线程数较少的分解方式，整体效率仍接近 100%。

展望

综合来看，这些基准测试凸显了 Pyroclast 初始求解器基础设施出色的数值稳健性和良好的并行性能。GPU 后端已在大规模斯托克斯问题上实现了显著加速，而分布式平流求解器则能够在多节点 CPU 配置中有效扩展。

需要强调的是，这些结果仍属初步成果。许多组件，包括 GPU 内核、多重网格层次结构、插值例程和通信模式，尚未经过系统优化。因此，随着开发工作的推进，绝对性能和可扩展性预计都将得到显著提升。

总体而言，此处介绍的基准测试为 Pyroclast 的长期目标奠定了坚实基础。随着持续优化，并逐步扩展至完全耦合的热力学-力学模拟，Pyroclast 有望成为用于大规模地球动力学建模的高性能、灵活且可扩展的框架。

第9章

结论与未来工作

本工作的目标是开发一个现代化、模块化且可扩展的高性能地球动力学模拟框架。最终成果是*Pyroclast*，这是一个基于 Python 的求解器，展示了如何有效结合灵活性、模块化和高性能。Pyroclast 填补了地球动力学计算领域长期存在的间隙：许多广泛使用的求解器依赖单体式且面向中央处理器的代码库，而这些代码库越来越难以维护、扩展或适应异构硬件和机器学习等新兴计算范式。

本文确立了新一代地球动力学建模框架的力学核心。Pyroclast 通过为不可压缩斯托克斯系统提供无矩阵、经图形处理器加速的求解器，展现了 Python 与现代科学计算生态系统相结合，能够成为构建大规模、研究级模拟软件的实用基础。该框架兼具数值稳健性、架构灵活性与高性能，同时显著降低了研究人员面临的技术门槛。

9.1 核心贡献

9.1.1 面向地球动力学的现代软件架构

Pyroclast 采用清晰的模块化架构，其中 MIC 流程的每个主要组件都彼此独立，易于测试和扩展。该设计基于严格的职责分离和设备无关编程，确保同一段代码能够在 CPU 或 GPU 上高效运行。这一架构转变是一项重要贡献。通过避免遗留代码中十分常见的单体式结构，Pyroclast 使研究人员能够快速试验新的物理模型和数值方法。研究人员无需

浏览复杂的构建系统或重写底层内核。这种易用性在面向 HPC 的地球动力学代码中并不常见，也使 Pyroclast 既成为实用工具，又成为现代参考实现。

9.1.2 无矩阵多重网格求解器

本项目的核心技术成果，是为不可压缩黏性流实现了一种无矩阵几何多重网格求解器。所有算子，包括模板内核、限制和延拓，均在不构建全局稀疏矩阵的情况下应用。这种方法最大限度地减少了内存占用并提高了吞吐量，在 GPU 上尤其如此。即使存在显著的黏度对比，该求解器仍能保持稳定且与分辨率无关的收敛。在经典的沉降体基准测试中，它表现可靠，黏度对比最高可达 10^8, ，并且始终能达到$\mathcal{O}$（10^{-4}）或更低的相对能量残差，无需手动调整参数。考虑到实际地球动力学问题可能存在极严重的病态条件，该方法的稳健性尤为突出。这些结果表明，Pyroclast 的求解器不仅正确，而且能够与成熟的底层代码相媲美，尽管它是用 Python 编写并依赖即时编译。

9.1.3 现代硬件上的性能与可扩展性

本工作的一个主要目标是证明，基于 Python 的高性能计算无需以牺牲性能和效率为代价。结果证实了这一目标，并凸显了 Pyroclast 所采用方法的有效性。GPU 驻留式多重网格求解器相较于经过优化的多核 CPU 配置，实现了整整一个数量级的加速比。所有经过测试的二维基准测试，规模最高达 15000×15000 个网格点，均能高效运行，并在这一规模下保持稳定收敛。

与此同时，标记点平流求解器在分布式环境中展现出近乎理想的弱扩展性，这得益于高效的基于 MPI 的光环交换和幽灵层管理。这些发现验证了 CuPy、Numba 和 mpi4py 可作为在 Python 中开展大规模科学计算的可靠基础。Pyroclast 是首批在单一且完全基于 Python 的界面中结合 GPU 加速与分布式内存并行的地球动力学求解器之一。

9.2 影响与科学意义

Pyroclast 的意义不仅体现在本文展示的数值结果和架构创新上，还为以下方向提供了发展路径：

现代而灵活的地球动力学建模，并与科学计算的主要发展趋势相契合。这些趋势包括异构硬件的使用日益普遍、对可重复研究的重视程度不断提高，以及机器学习迅速融入科学工作流程。

9.2.1 降低大规模模拟的门槛

Pyroclast 的一个显著特点是易于使用。传统的地球动力学求解器通常要求使用者精通 C 或 Fortran 等语言，同时还要熟悉消息传递接口、GPU 编程和复杂的构建系统。Pyroclast 凭借高级 Python 接口、模块化结构以及与设备无关的数值内核，消除了许多此类障碍。研究人员可以直接使用清晰简洁的组件，只需进行少量修改即可纳入新的物理模型，也不必再费力摸索遗留代码库。这种易用性让更广泛的地球科学家群体能够开展高分辨率、富含物理过程的模拟，并通过支持快速验证新想法，激发科学创造力。

9.2.2 与机器学习的集成

Pyroclast 天然兼容 PyTorch 和 TensorFlow 等机器学习框架。其基于数组的设计和驻留于 GPU 的数据结构，使集成神经代理模型、数据驱动的闭合定律或可微算子变得轻而易举。这些能力为多种混合方法开辟了道路，将物理洞见与数据驱动的方法结合起来。例如，基于神经网络的加速求解器、通过可微模拟进行反演建模，或由学习所得模型引导的自适应策略。几乎没有地球动力学代码能够在不进行大量修改的情况下与这类方法对接。Pyroclast 填补了这一间隙，使自身立足于高性能计算与人工智能的交汇处，而这一交汇点在物理科学各领域正日益重要。

9.2.3 现代化改造遗留代码的蓝图

Pyroclast 还展示了一种现代化现有地球动力学求解器的实用策略。许多广泛使用的代码依赖于在创建时适用、但如今难以适配新硬件和不断变化的研究需求的软件架构。Pyroclast 表明，可以设计出兼具模块化、易于维护和高性能特点，同时满足地球动力学数值计算要求的模拟工具。本文提出的方法为重构旧工具提供了范例，使其能够支持 CPU 与 GPU 混合执行、无矩阵算法以及更现代的软件工程实践。

9.3 未来工作

尽管这项工作奠定了坚实的基础，Pyroclast 的长期愿景还不止于此。本文所述工作直接引出了若干重要的后续发展方向。

9.3.1 完整的热力学耦合

下一项重要工作是整合能量方程和温度相关流变学。此项扩展将支持对黏弹塑性变形进行模拟，并使模型能够重现 I2ELVIS 和 I3ELVIS 等模型所处理的完整过程范围。

9.3.2 三维模型

将 Pyroclast 扩展到三维是一项重大挑战，也蕴含着巨大的机遇。本文介绍的核心算法自然可以推广，但在三维计算域中实际实现时，需要仔细优化数据移动、缓存优化和 GPU 内核调优。Pyroclast 的三维版本将有助于研究更真实的地质场景，并为 CPU 与 GPU 混合执行提供更具挑战性的基准。它还将支持扩展至更大型的多 GPU 环境。

9.3.3 分布式多重网格支持

要充分发挥 Pyroclast 的分布式潜力，必须在 Stokes 多重网格求解器中实现 MPI 支持。限制性加性 Schwarz 预条件等区域分解方法在这方面尤其有前景。完成这一组件后，便能在大型集群上实现完整的多层级扩展，并使高分辨率三维模拟成为可能。

9.3.4 数值与性能改进

数值组件仍有若干值得探索的改进方向。提升 GPU 内核占用率并减少内存带宽瓶颈，可立即带来性能提升。优化平滑策略、改进多重网格层级的构建方式，以及完善标记点到网格的传递，将增强该方法的稳健性，尤其是在处理非均质材料时。探索局部多项式积分等先进的平流方法，或许还能在不牺牲性能的前提下进一步提高精度，尤其是在分布式环境中。

9.3.5 混合物理与机器学习方法

鉴于 Pyroclast 与机器学习框架兼容，未来研究的一个令人振奋的方向是整合神经代理模型、可微算子和学习型物理。这些方法可以加速模拟、改进参数估计，并支持由数据驱动的自适应策略。传统物理建模与现代数据驱动方法相结合，或将为地球动力学研究开辟新的方向。

9.4 在线资源

Pyroclast 框架以开放在线代码库的形式提供。源代码、文档和示例问题可在 GitHub 上找到：https://github.com/MarcelFerrari/Pyroclast。

9.5 最后致辞

Pyroclast 展示了高性能地球动力学模拟既可以功能强大，也可以易于使用。它弥合了传统高性能计算实践与现代科学工作流之间的间隙。该框架灵活、GPU 就绪，旨在支持实验、可扩展性以及与机器学习的集成。

本研究中介绍的进展为新一代地球动力学求解器奠定了基础。该框架具有模块化、可扩展的特点，并顺应新兴的计算趋势。Pyroclast既是实用的研究工具，也代表着一种愿景：新一代地球动力学建模软件可以如何发展。

致谢

我谨向**Taras Gerya 教授（苏黎世联邦理工学院）**致以诚挚的谢意，感谢他始终如一的支持与指导。这不仅贯穿于本研究的开发过程，也贯穿于我在苏黎世联邦理工学院从事地球动力学研究的整个历程。他的指导深刻影响了我对科学、研究乃至地球本身的思考方式。

特别感谢**Alexander Sotoudeh（苏黎世联邦理工学院）**在这个项目期间与我并肩工作。我们无数次的讨论、共同经历的挫折以及发现新知的时刻，让这段经历不仅效率更高，也真正充满乐趣。他的贡献帮助塑造了这项工作的许多方面，我也很感激这段合作，以及由此建立的友谊。

我也非常感谢**Hussein Harake（瑞士国家超级计算中心）**，感谢他进行的诸多富有见地的讨论、提供的专业技术知识，以及在基准测试关键阶段给予的最后关头帮助。他乐于及时伸出援手，解决硬件问题，这对我们顺利完成所有工作至关重要。

参考文献

[1] T. Gerya，引言。剑桥大学出版社，2019年，第 1–11 页。[2] D. Turcotte 和 G. Schubert，板块构造。剑桥大学出版社，2014年，第 1–91 页。[3] G. Schubert、D. L. Turcotte 和 P. Olson，地球与行星的地幔对流。剑桥大学出版社，2001年。[4] D. Turcotte 和 G. Schubert，热传递。剑桥大学出版社，2014年，第 160–229 页。[5] T. Gerya，热量守恒方程。剑桥大学出版社，2019年，第 128–138 页。[6] D. Turcotte 和 G. Schubert，岩石流变学。剑桥大学出版社，2014年，第 336–385 页。[7] T. Gerya，岩石黏性流变。剑桥大学出版社，2019年，第 73–81 页。[8] ——，地震-热-力学建模。剑桥大学出版社，2019年，第 224–239 页。[9] ——，水-热-力学建模。剑桥大学出版社，2019年，第 240–276 页。[10] M. Ferrari，“地球物理标记点网格模拟中严重病态线性系统稀疏求解器的比较”，2024年。[在线]。链接：https://arxiv.org/abs/2409.11515[11] T. Gerya，自适应网格加密。剑桥大学出版社，2019年，第 277–291 页。

[12] C2SM，“ICON4Py：气候和天气模型 ICON 的 Python 实现”，https://github.com/C2SM/icon4py，2024年，GitHub 代码库，访问日期：2025-10-30。

[13] GPU4GEO 研究团队，“GPU4GEO：前沿图形处理器多物理场求解器”，https://gpu4geo.org，2025年。 [在线]。可访问：https://gpu4geo.org[14] ——，“GPU4GEO：软件”，https://gpu4geo.org/software/，2025年。[在线]。可访问：https://gpu4geo.org/software/

[15] J. Ansel、E. Yang、H. He、N. Gimelshein、A. Jain、M. Voznesensky、B. Bao、P. Bell、D. Berard、E. Burovski、G. Chauhan、A. Chourdia、W. Constable、A. Desmaison、Z. DeVito、E. Ellison、W. Feng、J. Gong、M. Gschwind、B. Hirsh、S. Huang、K. Kalambarkar、L.Kirsch、M. Lazos、M. Lezcano、Y. Liang、J. Liang、Y. Lu、C. Luk、B. Maher、Y. Pan、C. Puhrsch、M. Reso、M. Saroufim、M. Y. Siraichi、H. Suk、M. Suo、P. Tillet、E. Wang、X. Wang、W. Wen、S. Zhang、X. Zhao、K. Zhou、R. Zou、A. Mathews、G. Chanan、P. Wu 和 S. Chintala，《PyTorch 2：通过动态 Python 字节码转换和图编译加速机器学习》，载于第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议，第 *2* 卷（*ASPLOS '24*）。ACM，2024年4月。 [在线]。可访问：https://docs.pytorch.org/assets/pytorch2-2.pdf

[16] M. Abadi、A. Agarwal、P. Barham、E. Brevdo、Z. Chen、C. Citro、G. S. Corrado、A. Davis、J. Dean、M. Devin、S. Ghemawat、I. Goodfellow、A. Harp、G. Irving、M. Isard、Y. Jia、R. Jozefowicz、L. Kaiser、M. Kudlur、J. Levenberg、D. Man´e、R. Monga、S. Moore、D. Murray、C. Olah、M. Schuster、J. Shlens、B. Steiner、I. Sutskever、K. Talwar、P. Tucker、V. Vanhoucke、V. Vasudevan、F. Vi´egas、O. Vinyals、P. Warden、M. Wattenberg、M. Wicke、Y. Yu 和 X. Zheng，《TensorFlow：面向异构系统的大规模机器学习》，2015年，软件可从 tensorflow.org 获取。 [在线]。网址：https://www.tensorflow.org/

[17] T. V. Gerya 和 D. A. Yuen，《用于模拟多相黏弹塑性热力学问题的鲁棒特征线方法》，《地球与行星内部物理学》，第 163 卷，第 1 期，第 83–105 页，2007年，地球科学中的计算挑战。 [在线]。可访问：https://www.sciencedirect.com/science/article/pii/S0031920107000969

[18] ——，“基于特征的标记点-网格法，采用守恒有限差分格式模拟具有强

可变输运性质的地质流动”，《地球与行星内部物理学》，第140卷，第4期，第293–318页，2003年。 [在线]。可访问：https: // www.sciencedirect.com/science/article/pii/S0031920103001900[19] T. Gerya，“黏弹塑性的二维实现”，载于《数值地球动力学建模导论》。剑桥大学出版社，2019年，第188–208页。[20] M. Benzi、G. H. Golub 和 J. Liesen，“鞍点问题的数值解”，《数值学报》，第14卷，第1–137页，2005年。[21] T. Gerya，“多重网格法”，载于《数值地球动力学建模导论》。剑桥大学出版社，2019年，第292–318页。[22] T. Heister、J. Dannberg、R. Gassm¨oller 和 W. Bangerth，“通过现代数值方法实现高精度地幔对流模拟。II：现实模型与问题”，《国际地球物理学杂志》，第210卷，第2期，第833–851页，2017年。 [在线]。可访问：https://doi.org/10.1093/gji/ggx195[23] M. Kronbichler、T. Heister 和 W. Bangerth，“通过现代数值方法实现高精度地幔对流模拟”，《国际地球物理学杂志》，第191卷，第12–29页，2012年。 [在线]。可访问：http://dx.doi.org/10.1111/j.1365-246X.2012.05609.x[24] W. Bangerth、J. Dannberg、M. Fraters、R. Gassmoeller、A. Glerum、T. Heister、R. Myhill 和 J. Naliboff，“ASPECT v3.0.0”，2024年12月。[在线]。可访问：https://doi.org/10.5281/zenodo.14371679[25] ——，“ASPECT：行星演化、对流与构造高级求解器，用户手册”，2024年12月。 [在线]。可访问：https://doi.org/10.6084/m9.figshare.4865333[26] L. Moresi、S. Zhong、L. Han、C. Conrad、E. Tan、M. Gurnis、E. Choi、P. Thoutireddy、V. Manea、A. McNamara、T. Becker、W. Leng 和 L. Armendariz，“Citcoms v3.3.1”，2014年11月。 [在线]。可访问：https://doi.org/10.5281/zenodo.7271920[27] E. Tan、E. Choi、P. Thoutireddy、M. Gurnis 和 M. Aivazis，“Geoframework：在计算框架内耦合多个地幔对流模型”，《地球化学、地球物理学、地球系统》，第7卷，第6期，文章编号Q06001，2006年。[28] S. Zhong、M. Zuber、L. Moresi 和 M. Gurnis，“温度依赖性黏度和地表板块在地幔对流球壳模型中的作用”，《地球物理研究杂志：固体地球》，第105卷，第B5期，第11 063–11 082页，2000年。

[29] A. McNamara 和 S. Zhong，“球形地幔中的热化学结构：超级地幔柱还是地幔堆积体？”《地球物理研究杂志：固体地球》，第 109 卷，B7 期，B07402 页，2004年。[30] B. Holt 和 D. Ernst，“使用 CUDA 加速地球物理模拟”，《计算科学教育杂志》，第 2 卷，第 21–27 页，2011年12月。[31] C. H¨uttig、A.-C. Plesa 和 N. Tosi，“GAIA：研究岩质和冰质天体热化学演化的框架”，https://helmholt z.software/software/gaia，2023年9月，访问日期：2025-10-31。[32] C. H¨uttig 和 K. Stemmer，“螺旋网格：一种球面离散化的新方法及其在地幔对流中的应用”，《地球化学、地球物理学、地球系统》，第 9 卷，第 2 期，2008年。 [在线]。可访问：https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2007GC001581[33] ——，“沃罗诺伊网格上动力黏度的有限体积离散化”，《地球与行星内部物理学》，第 171 卷，第 1 期，第 137–146 页，2008年，《计算地球动力学新进展：理论、数值方法与应用》。 [在线]。可访问：https: //www.sciencedirect.com/science/article/pii/S0031920108001659[34] C. H¨uttig、N. Tosi 和 W. B. Moore，“变黏度不可压缩纳维–斯托克斯方程的改进公式”，《地球与行星内部物理学》，第 220 卷，第 11–18 页，2013年。[在线]。可访问：https://www.sciencedirect.com/science/article/pii/S0031920113000459[35] D. A. May、J. Brown 和 L. Le Pourhiet，“pTatin3D：用于长期岩石圈动力学的高性能方法”，载于《国际高性能计算、网络、存储与分析会议论文集》，丛书：SC ’14。IEEE出版社，2014年，第 274–284 页。 [在线]。可访问：https://doi.org/10.1109/SC.2014.28[36] P. J. Tackley，“使用阴阳网格，在三维球壳中模拟具有巨大黏度对比的可压缩地幔对流”，《地球与行星内部物理学》，第 171 卷，第 1 期，第 7–18 页，2008年，《计算地球动力学新进展：理论、数值方法与应用》。 [在线]。可访问：https: //www.sciencedirect.com/science/article/pii/S0031920108002276[37] S. 贡献者，“StagBL：用于地球动力学模拟的可扩展、可移植、高性能离散化与求解器层”，https://gith ub.com/stagbl/stagbl，2025年10月，访问日期：2025-10-31。

[38] J. Mansour、J. Giordani、L. Moresi、R. Beucher、O. Kaluza、M. Velic、R. Farrington、S. Quenette 和 A. Beall，“Underworld2：面向桌面、高性能计算和云端的 Python 地球动力学建模”，《开放源代码软件期刊》，第 5 卷，第 47 期，第 1797 页，2020年。 [在线]。可访问：https://doi.org/10.21105/joss.01797

[39] R. Beucher、L. Moresi、J. Giordani、J. Mansour、D. Sandiford、R. Farrington、L. Mondy、C. Mallard、P. Rey、G. Duclaux、O. Kaluza、A. Laik 和 S. Mor´on，《Uwgeodynamics：用于数值地球动力学建模的教学与研究工具》，《开放源代码软件期刊》，第 4 卷，第 36 期，第 1136 页，2019年。 [在线]。网址：https://doi.org/10.21105/joss.01136

[40] L. Moresi、F. Dufour 和 H. B. M¨uhlhaus，“采用黏弹性/脆性岩石圈的地幔对流建模：数值方法与板块构造建模”，《纯粹与应用地球物理学》，第 159 卷，第 10 期，第 2335–2356 页，2002年。 [在线]。获取地址：https: //doi.org/10.1007/s00024-002-8738-3[41] L. Moresi、F. Dufour 和 H.-B. M¨uhlhaus，“用于黏弹性地质材料大变形建模的拉格朗日积分点有限元法”，《计算物理学杂志》，第 184 卷，第 2 期，第 476–497 页，2003年。 [在线]。获取地址：https: //www.sciencedirect.com/science/article/pii/S0021999102000311[42] L. Moresi、S. Quenette、V. Lemiale、C. M´eriaux、B. Appelbe 和 H.-B. M¨uhlhaus，“研究地壳和地幔非线性动力学的计算方法”，《地球与行星内部物理学》，第 163 卷，第 1 期，第 69–82 页，2007年，地球科学中的计算挑战。 [在线]。获取地址：https: //www.sciencedirect.com/science/article/pii/S0031920107001446[43] S. Balay、S. Abhyankar、M. F. Adams、S. Benson、J. Brown、P. Brune、K. Buschelman、E. M. Constantinescu、L. Dalcin、A. Dener、V. Eijkhout、J. Faibussowitsch、W. D. Gropp、V. Hapla、T. Isaac、P. Jolivet、D. Karpeev、D. Kaushik、M. G. Knepley、F. Kong、S. Kruger、D. A. May、L. C. McInnes、R. T. Mills、L. Mitchell、T. Munson、J. E. Roman、K. Rupp、P. Sanan、J. Sarich、B. F. Smith、S. Zampini、H. Zhang、H. Zhang 和 J. Zhang，“PETSc 网页”，https://petsc.org/，2024年。 [在线]。获取地址：https://petsc.org/[44] S. Balay、S. Abhyankar、M. F. Adams、S. Benson、J. Brown、P. Brune、K. Buschelman、E. Constantinescu、L. Dalcin、A. Dener、V. Eijkhout、J. Faibusso witsch、W. D. Gropp、V. Hapla、T. Isaac、P. Jolivet、D. Karpeev、D. Kaushik、M. G. Knepley、F. Kong、S. Kruger、D. A. May、L. C. McInnes,

R. T. Mills、L. Mitchell、T. Munson、J. E. Roman、K. Rupp、P. Sanan、J. Sarich、B. F. Smith、H. Suh、S. Zampini、H. Zhang、H. Zhang 和 J. Zhang，“PETSc/TAO 用户手册”，阿贡国家实验室，技术报告 ANL-21/39 - Revision3.24，2025年。

[45] S. Balay、W. D. Gropp、L. C. McInnes 和 B. F. Smith，《面向对象数值软件库中并行性的高效管理》，载于《科学计算中的现代软件工具》，E. Arge、A. M. Bruaset 和 H. P. Langtangen 编。Birkh¨auser Press，1997年，第 163–202 页。[46] R. T. Mills、M. F. Adams、S. Balay、J. Brown 和 A. Dener，《面向基于 GPU 的百亿亿次级系统的性能可移植 PETSc》，《并行计算》，第 108 卷，文章编号 102831，2021年。 [在线]。链接：https: // www.sciencedirect.com/science/article/pii/S016781912100079X[47] M. K¨aser 和 M. Dumbser，《非结构化网格上弹性波的任意高阶间断伽辽金方法——I. 含外部源项的二维各向同性情形》，《国际地球物理学杂志》，第 166 卷，第 2 期，第 855–877 页，2006年 08 月。 [在线]。链接：https://doi.org/10.1111/j.1365-246X.2006.03051.x[48] M. Dumbser 和 M. K¨aser，《非结构化网格上弹性波的任意高阶间断伽辽金方法——II. 三维各向同性情形》，《国际地球物理学杂志》，第 167 卷，第 1 期，第 319–336 页，2006年 10 月。[在线]。链接：https://doi.org/10.1111/j.1365-246X.2006.03120.x[49] R. Dorozhinskii、G. B. Gadeschi 和 M. Bader，《通过融合通用矩阵乘法实现 SeisSol 中 ADER-DG 方法的高效 GPU 实现》，《并发与计算：实践与经验》，第 36 卷，第 12 期，文章编号 e8037，2024年。 [在线]。链接：https: // onlinelibrary.wiley.com/doi/abs/10.1002/cpe.8037[50] A. Heinecke、A. Breuer、S. Rettenberger、M. Bader、A.-A. Gabriel、C. Pelties、A. Bode、W. Barth、X.-K. Liao、K. Vaidyanathan、M. Smelyanskiy 和 P. Dubey，《异构超级计算机上的千万亿次级高阶动态破裂地震模拟》，载于《*SC ’14*：国际高性能计算、网络、存储与分析会议论文集》，2014年，第 3–14 页。

[51] D. Komatitsch、J. Tromp、R. Garg、H. N. Gharti、M. Nagaso、E. Oral、D. Peter、M. Afanasiev、R. Almada、J.-P. Ampuero、E. Bachmann、K. Bai、P. Basini、S. Beller、J. Bishop、F. Bissey、C. Blitz、A. Bottero、E. Bozdag、E. Casarotti、J. Charles、M. Chen、P. Cristini、C. Durochat、P. Galvez Barron、R. Gassmoeller、D. Goeddeke、L. Grinberg,

A. Gupta、E. Heien、V. Hjoerleifsdottir、F. Karakostas、S. Kientz、J. Labarta、N. Le Goff、P. Le Loher、M. Lefebvre、Q. Liu、Y. Liu、D. Luet、Y. Luo、A. Maggi、F. Magnoni、R. Martin、R. Matzen、G. D. McBain、D. McRitchie、M. Meschede、P. Messmer、D. Michea、D. Miller、R. Modrak、V. Monteiller、C. Morency、S. Nadh Somala、T. Nissen-Meyer、K. Pouget、M. Rietmann、E. Sales de Andrade、B. Savage、B. Schuberth、A. Sie minski、J. Smith、L. Strand、C. Tape、E. Valero Cano、B. Videau、J.-P. Vilotte、H. Weng、Z. Xie、C.-H. Zhang和H. Zhu，《Specfem/specfem3d：Specfem3d v4.1.1》，2024年3月。[在线]。链接：https://doi.org/10.5281/zenodo.10823181

[52] D. Komatitsch 和 J. Tromp，《全球地震波传播的谱元模拟——I. 验证》，《国际地球物理学杂志》，第149卷，第2期，第390–412页，2002年。 [在线]。网址：https ：//onlinelibrary.wiley.com/doi/abs/10.1046/j.1365-246X.2002年.01653.x

[53] D. Komatitsch，《用于地震波传播的 GPU 显卡集群上的流固耦合》，《*Comptes Rendus M´ecanique*，第339卷，第2期，第125–135页，2011年，高性能计算。 [在线]。网址：https://www.sciencedirect.com/science/article/pii/S163 1072110002081

[54] Y. Cui、K. B. Olsen、T. H. Jordan、K. Lee、J. Zhou、P. Small、D. Roten、G. Ely、D. K. Panda、A. Chourasia、J. Levesque、S.M. Day 和 P. Maech- ling，《千万亿次级超级计算机上的可扩展地震模拟》，载于*SC ’10*：*2010*年*ACM/IEEE*高性能计算、网络、存储与分析国际会议论文集，2010年，第1–20页。

[55] J. Zhou、D. Unat、D. J. Choi、C. C. Guest 和 Y. Cui，《在 GPU Fermi 芯片组上对三维有限差分地震模拟进行动手性能调优》，《计算机科学 *Procedia*》，第 9 卷，第 976–985 页，2012年，国际计算科学会议 ICCS 2012 论文集。 [在线]。可访问：https: // www.sciencedirect.com/science/article/pii/S1877050912002256

[56] D. A. Beckingsale、J. Burmark、R. Hornung、H. Jones、W. Killian、A. J. Kunen、O. Pearce、P. Robinson、B. S. Ryujin 和 T. R. Scogland，《RAJA：面向大规模科学应用的可移植性能》，发表于*2019 IEEE/ACM* 国际性能、可移植性与生产力研讨会 - 高性能计算（*P3HPC*），2019年，第71–81页。

[57] D. Beckingsale、R. Hornung、T. Scogland 和 A. Vargas，“使用 RAJA 进行性能可移植的 C++ 编程”，载于第*24*届并行原理与实践研讨会论文集

编程，丛书：PPoPP ’19。美国纽约：计算机协会，2019年，第455–456页。 [在线]。链接：https://doi.org/10.1145/3293883.3302577[58] N. A. Petersson、B. Sjogreen、H. Tang 和 R. Pankajakshan，“geodynamics/sw4：Sw4，3.0版”，2023年9月。 [在线]。链接：https://doi.org/10.5281/zenodo.8322590[59] B. Sj¨ogreen 和 N. A. Petersson，“弹性波动方程二阶形式的四阶精度有限差分格式”，《科学计算杂志》，第52卷，第1期，第17–48页，2012年。 [在线]。链接：https://doi.org/10.1007/s10915-011-9531-1[60] R. Pankajakshan、P.-H. Lin 和 B. Sj¨ogreen，“将三维地震模拟代码（sw4）移植到 Coral 机器”，《*IBM*研究与开发杂志》，第64卷，第3/4期，第17:1–17:11页，2020年。[61] B. T. Aagaard、M. G. Knepley 和 C. A. Williams，“在准静态与动态地壳变形有限元模型中实现断层滑移的区域分解方法”，《地球物理研究杂志：固体地球》，第118卷，第6期，第3059–3079页，2013年。 [在线]。链接：https: //agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/jgrb.50217 [62] P. D. Team，“各版本的变更”，https://petsc.org/release/ch anges/，2025年，访问日期：2025-10-31。[63] PETSc 开发团队，“GPU 支持路线图”，https://petsc.or g/release/overview/gpu roadmap/，2025年，最后更新于2025-10-31（v3.24.1）。[64] C. R. Trott、D. Lebrun-Grandi´e、D. Arndt、J. Ciesko、V. Dang、N. Elling- wood、R. Gayatri、E. Harvey、D. S. Hollman、D. Ibanez、N. Liber、J. Mad- sen、J. Miles、D. Poliakoff、A. Powell、S. Rajamanickam、M. Simberg、D. Sunderland、B. Turcksin 和 J. Wilke，“Kokkos 3：面向百亿亿次时代的编程模型扩展”，《*IEEE*并行与分布式系统汇刊》，第33卷，第4期，第805–817页，2022年。[65] H. C. Edwards、C. R. Trott 和 D. Sunderland，“Kokkos：通过多态内存访问模式实现众核性能可移植性”，《并行与分布式计算杂志》，第74卷，第12期，第3202–3216页，2014年，面向高性能计算的领域特定语言与高级框架。 [在线]。链接：http://www.sciencedirect.com/science/article/pii/S07437 31514001257

[66] OpenACC 组织，《*OpenACC* 应用程序编程接口》，2025年6月。 [在线]。获取地址：https://www.openacc.org/sp ecification[67] 英伟达公司，《*cuBLAS*：*NVIDIA CUDA* 基本线性代数子程序库》，2025年，面向 NVIDIA GPU 的 GPU 加速 BLAS 库。 [在线]。获取地址：https://developer.nvidia.com/cublas[68] ——，《*cuTENSOR*：*NVIDIA* 张量线性代数库》，2025年，面向 NVIDIA GPU 的 GPU 加速张量原语库。 [在线]。获取地址：https://developer.nvidia.com/cutensor[69] AMD 公司，《*hipBLAS*：*ROCm/HIP* 基本线性代数子程序库》，2025年，支持多个后端（rocBLAS、cuBLAS）的 BLAS 编组库。 [在线]。获取地址：https://rocm.docs.amd.com/ projects/hipBLAS/en/latest/[70] ——，《*hipTensor*：*ROCm/HIP* 张量原语库》，2025年，适用于 AMD GPU 的张量原语 C 库。 [在线]。获取地址：https://rocm.docs.amd.com/projects/hipTensor/en/latest/[71] S. K. La m、A. Pitrou 和 S. Seibert，“Numba：基于 LLVM 的 Python JIT 编译器”，载于高性能计算领域 *LLVM* 编译器基础设施第二届研讨会论文集，LLVM ’15 系列。美国纽约州纽约市：计算机协会，2015年。 [在线]。获取地址：https://doi.org/10.1145/2833157.2833162[72] 英伟达公司，“Numba-CUDA：适用于 Numba JIT 编译器的 CUDA 目标”，2024年，通过 Numba 实现的 NVIDIA GPU 内核 Python JIT 编译器。 [在线]。获取地址：https://nvidia.github.io/numba-cuda/[73] AMD 公司，“Numba-HIP：适用于 Numba JIT 编译器的 ROCm/HIP 后端”，2024年，面向 Numba 的实验性 AMD HIP/ROCm GPU 后端。 [在线]。获取地址：https://github.com/ROCm/numba-hip[74] T. Ben-Nun、J. de Fine Licht、A. N. Ziogas、T. Schneider 和 T. Hoefler，“有状态数据流多重图：一种面向异构架构性能可移植性的以数据为中心的模型”，载于国际高性能计算、网络、存储与分析会议论文集，SC ’19 系列。美国纽约州纽约市：计算机协会，2019年。 [在线]。获取地址：https://doi.org/10.1145/3295500.3356173 [75] W. Jakob，“nanobind：小巧高效的 C++/Python 绑定”，2022年，https://github.com/wjakob/nanobind。

[76] C. R. Harris、K. J. Millman、S. J. van der Walt、R. Gommers、P. Virtanen、D. Cournapeau、E. Wieser、J. Taylor、S. Berg、N. J. Smith、R. Kern、M. Picus、S. Hoyer、M. H. van Kerkwijk、M. Brett、A. Haldane、J. F. del R´ıo、M. Wiebe、P. Peterson、P. G´erard-Marchant、K. Sheppard、T. Reddy、W. Weckesser、H. Abbasi、C. Gohlke 和 T. E. Oliphant，“使用 NumPy 进行数组编程”，《自然》，第585卷，第7825期，第357–362页，2020年9月。 [在线]。获取地址：https://doi.org/10.1038/s41586-020-2649-2

[77] R. Okuta、Y. Unno、D. Nishino、S. Hido 和 C. Loomis，“CuPy：用于 NVIDIA GPU 计算的兼容 NumPy 的库”，载于第三十一届神经信息处理系统年会（*NIPS*）机器学习系统研讨会（*LearningSys*）论文集，2017年。[在线]。链接：http://learningsys.org/nips17/assets/papers/pap er 16.pdf

[78] G. M. Kurtzer、cclerget、M. Bauer、I. Kaneshiro、D. Trudgian 和 D. Godlove，《hpcng/singularity：Singularity 3.7.3》，2021年4月。 [在线]。可访问：https://doi.org/10.5281/zenodo.4667718[79] G. M. Kurtzer、V. Sochat 和 M. W. Bauer，《Singularity：用于计算迁移的科学容器》，《公共科学图书馆·综合》，第 12 卷，第 5 期，第 1–20 页，2017年5月。 [在线]。可访问：https: //doi.org/10.1371/journal.pone.0177459[80] NVIDIA，《Enroot：将传统容器/操作系统映像转换为非特权沙箱的简单而强大的工具》，2025年，访问日期：2025-10-31。 [在线]。可访问：https://github.com/NVIDIA/enroot[81] 英伟达公司，《NVIDIA PyTorch 容器》，英伟达公司，2025年，访问日期：2025年10月31日。[在线]。可访问：https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch[82] ——，《NVIDIA 集合通信库（NCCL）》，英伟达公司，2025年，访问日期：2025年10月31日。 [在线]。可访问：https://developer.nvidia.com/nccl[83] ——，《NVIDIA NVSHMEM：适用于 NVIDIA GPU 的 OpenSHMEM 库》，英伟达公司，2025年，访问日期：2025年10月31日。 [在线]。可访问：https://docs.nvidia.com/nvshmem[84] N. Vetsch、A. Maeder、V. Maillou、A. Winka、J. Cao、G. Kwasniewski、L. Deuschle、T. Hoefler、A. N. Ziogas 和 M. Luisier，《采用 GW 近似的从头算量子输运：42,240 个原子与持续百亿亿次级性能》，2025年。 [在线]。可访问：https://arxiv.org/abs/2508.19138

[85] S. Russo。（2025年10月）苏黎世联邦理工学院团队模拟完整纳米带晶体管，并获戈登·贝尔奖提名。瑞士国家超级计算中心（CSCS）。访问日期：2025年11月1日。¨在线[。可访问：https://www.cscs.ch/science/computer-science-hpc/2025/eth-zuerich-team-simulates-a-full-nanoribbon-transistor-and-earns-a-gordon-bell-prize-nomination

[86] ICON合作伙伴联盟（MPI-M；DWD；DKRZ；KIT；C2SM），《ICON 2025.04 版本发布》，https://doi.org/10.35089/WDCC/IconRelease2025.04，2024年，DKRZ 世界气候数据中心（WDCC）。

[87] E. G. Paredes、L. Groner、S. Ubbiali、H. Vogt、A. Madonna、K. Mariotti、F. Cruz、L. Benedicic、M. Bianco、J. VandeVondele 和 T. C. Schulthess，《GT4Py：使用 Python 为天气与气候应用提供高性能模板》，2023年。 [在线]。可访问：https://arxiv.org/abs/2311.08322

[88] M. Louboutin、M. Lange、F. Luporini、N. Kukreja、P. A. Witte、F. J. Herrmann、P. Velesko 和 G. J. Gorman，“Devito（v3.1.0）：一种用于有限差分和地球物理勘探的嵌入式领域专用语言”，《地球科学模型开发》，第 12 卷，第 3 期，第 1165–1187 页，2019年。 [在线]。可访问：https://www.geosci-model- dev.net/12/1165/2019/[89] F. Luporini、M. Louboutin、M. Lange、N. Kukreja、P. Witte、J. H¨uckelheim、C. Yount、P. H. J. Kelly、F. J. Herrmann 和 G. J. Gorman，“Devito 的架构与性能：一个自动化模板计算系统”，《*ACM* 数学软件汇刊》，第 46 卷，第 1 期，4，2020 年。 [在线]。可访问：https://doi.org/10.1145/3374916
[90] A. Meurer、C. P. Smith、M. Paprocki、O. ˇCert´ık、S. B. Kirpichev、M. Rocklin、A. Kumar、S. Ivanov、J. K. Moore、S. Singh、T. Rathnayake、S. Vig、B. E. Granger、R. P. Muller、F. Bonazzi、H. Gupta、S. Vats、F. Johansson、F. Pedregosa、M. J. Curry、A. R. Terrel、v. Rouˇcka、A. Saboo、I. Fernando、S. Kulal、R. Cimrman 和 A. Scopatz，“Sympy：Python 符号计算”，《*PeerJ*计算机科学》，第 3 卷，文章 e103，2017年 1 月。 [在线]。可访问：https://doi.org/10.7717/peerj-cs.103[91] G. Bisbas、R. Nelson、M. Louboutin、F. Luporini、P. H. J. Kelly 和 G. Gorman，“面向可扩展有限差分求解器的自动化 MPI-X 代码生成”，2024年。 [在线]。可访问：https://arxiv.org/abs/2312.13094[92] M. Raissi、P. Perdikaris 和 G. E. Karniadakis，“物理信息深度学习（第一部分）：数据驱动的

非线性偏微分方程解”，2017年。 [在线]。可访问：
https://arxiv.org/abs/1711.10561

[93] ——，“物理信息深度学习（第二部分）：非线性偏微分方程的数据驱动发现”，2017年。 [在线]。可访问：https://arxiv.org/abs/1711.10566

[94] S. Cai、Z. Mao、Z. Wang、M. Yin 和 G. E. Karniadakis，“用于流体力学的物理信息神经网络（PINN）：综述”，力学学报，第37卷，第12期，第1727–1738页，2021年。[在线]。可访问：https://doi.org/10.1007/s10409-021-01148-1

[95] M. Habib、A. Habib 和 B. Alibrahim，“物理信息神经网络在地球科学中的应用：从基础地震学到综合环境研究”，开放地球科学，第17卷，第1期，第20250853页，2025年。 [在线]。可访问：https://doi.org/10.1515/geo-2025-0853

[96] Z. Li、N. Kovachki、K. Azizzadenesheli、B. Liu、K. Bhattacharya、A. Stuart 和 A. Anandkumar，《用于参数化偏微分方程的傅里叶神经算子》，2021年。 [在线]。可访问：https://arxiv.org/abs/2010.08895

[97] B. Bonev、T. Kurth、C. Hundt、J. Pathak、M. Baust、K. Kashinath 和 A. Anandkumar，《球面傅里叶神经算子：学习球面上的稳定动力学》，2023年。[在线]。可访问：https://arxiv.org/abs/2306.03838

[98] B. Raoni´c、R. Molinaro、T. De Ryck、T. Rohner、F. Bartolucci、R. Alaifari、S. Mishra 和 E. de B´ezenac，《用于稳健且准确学习偏微分方程的卷积神经算子》，载于第*37*届神经信息处理系统国际会议论文集，NIPS ’23 系列。美国纽约州雷德胡克：Curran Associates Inc.，2023年。

[99] H. Liu、Y. Liu、K. Li、Z. Zhao、S. S. Schoenholz、E. D. Cubuk、P. Gupta 和 M. Bauchy，“利用端到端可微性和张量处理单元计算加速材料逆向设计”，*npj* 计算材料，第 9 卷，第 1 期，第 121 页，2023年。 [在线]。获取地址：https://doi.org/10.1038/s41524-023-01080-x[100]Y. Hu、L. Anderson、T.-M. Li、Q. Sun、N. Carr、J. Ragan- Kelley 和 F. Durand，“DiffTaichi：用于物理模拟的可微编程”，2020年。[在线]。获取地址：https: //arxiv.org/abs/1910.00935

[101] M. Herde、B. Raoni´c、T. Rohner、R. K¨appeli、R. Molinaro、E. de B´ezenac 和 S. Mishra，“Poseidon：用于偏微分方程的高效基础模型”，2024年。[在线]。获取地址：https://arxiv.org/abs/2405.19101[102]C. Bodnar、W. P. Bruinsma、A. Lucic、M. Stanley、A. Allen、J. Brandstetter、P. Garvan、M. Riechert、J. A. Weyn、H. Dong、J. K. Gupta、K. Thambiratnam、A. T. Archibald、C.-C. Wu、E. Heider、M. Welling、R. E. Turner 和 P. Perdikaris，“地球系统基础模型”，《自然》，第 641 卷，第 8065 期，第 1180–1187 页，2025年。[在线]。获取地址：https://doi.org/10.1038/s41586-025-09005-y[103] T. Gerya，《数值地球动力学建模导论》，第 2 版。剑桥大学出版社，2019年。[104] ——，《动量方程和连续性方程的数值解》。剑桥大学出版社，2019年，第 82–104 页。[105] S. MacNamara 和 G. Strang，算子分裂。Cham：施普林格国际出版，2016年，第 95–114 页。 [在线]。获取地址：https://doi.org/10.1007/978-3-319-41589-5 3[106] T. Gerya，平流方程与标记点网格法。剑桥大学出版社，2019年，第 105–127 页。[107]龙格–库塔方法。约翰·威立父子公司，2016年，第 3 章，第 143–331 页。 [在线]。获取地址：https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119121534.ch3[108] T. Gerya，《动量方程和连续性方程的数值解》。剑桥大学出版社，2019年，第 82–104 页。[109] ——，“动量方程”，载于《数值地球动力学建模导论》。剑桥大学出版社，2019年，第 60–72 页。[110] M. Ferrari、C. P¨untener、A. Sotoudeh 和 N. Viebig，“二维变黏度斯托克斯方程中用于无矩阵平滑器的三维分块及其在地球动力学中的应用”，2025年。 [在线]。获取地址：https://arxiv.org/abs/2509.19061[111] A. Duguid，“K. J. Arrow、L. Hurwicz 和 H. Uzawa 所著《线性与非线性规划研究》。斯坦福大学出版社，1958年。229 页。$7.50。”《加拿大数学通报》，第 3 卷，第 3 期，第 196–198 页，1960年。[112] Y. Saad，《稀疏线性系统的迭代方法》，第 2 版。工业与应用数学学会，2003年。 [在线]。获取地址：https://epubs.siam.org/doi/abs/10.1137/1.9780898718003

[113] W. Hackbusch，《预备知识》。柏林，海德堡：Springer Berlin Heidelberg，1985年，第1–16页。 [在线]。获取地址：https://doi.org/10.100 7/978-3-662-02427-0 1[114]T. A. Davis，《稀疏线性系统的直接方法》。工业与应用数学学会，2006年。 [在线]。获取地址：https://epubs.siam.org/doi/abs/10.1137/1.9780898718881[115]W. Hackbusch，《多重网格方法及其应用》。柏林，海德堡：Springer Berlin Heidelberg，1985年。 [在线]。获取地址：https://doi.org/10.1007/978-3-662-02427-0[116] J. W. Ruge 和 K. St¨uben，《4. 代数多重网格》，第73–130页。 [在线]。获取地址：https://epubs.siam.org/doi/abs/10.1137/1.9781611971057 .ch4[117]N. Bell、L. N. Olson、J. Schroder 和 B. Southworth，“PyAMG：Python 中的代数多重网格求解器”，《开放源代码软件期刊》，第8卷，第87期，第5495页，2023年。[在线]。获取地址：https://doi.org/10.21105/joss.05495 [118] D. Demidov，“Amgcl——用于高效求解大型稀疏线性系统的 C++ 库”，《软件影响》，第6卷，第100037页，2020年。[在线]。获取地址：https://doi.org/10.1016/j.simpa.2020.100037[119]N. Corporation，“AmgX：GPU 加速的代数多重网格求解器”，2025年，访问日期：2025年11月8日。[在线]。获取地址：https://developer.nvidia.com/amgx[120] M. Furuichi、D. A. May 和 P. J. Tackley，“采用混合精度算术和舒尔补方法开发可处理大幅黏度跃变的斯托克斯流求解器”，《计算物理学杂志》，第230卷，第24期，第8835–8851页，2011年。 [在线]。获取地址：https: // www.sciencedirect.com/science/article/pii/S0021999111005274[121]N. Ho、S. D. Olson 和 H. F. Walker，“加速 Uzawa 算法”，2016年。 [在线]。获取地址：https://arxiv.org/abs/1510.042 46[122] S. C. Eisenstat、H. C. Elman 和 M. H. Schultz，“用于非对称线性方程组的变分迭代方法”，《SIAM 数值分析杂志》，第20卷，第2期，第345–357页，1983年。 [在线]。获取地址：https://doi.org/10.1137/0720023[123] D. G. Anderson，“非线性积分方程的迭代过程”，《ACM 杂志》，第12卷，第4期，第547–560页，1965年10月。 [在线]。获取地址：https://doi.org/10.1145/321296.321305

[124]H. F. Walker 和 P. Ni，“不动点迭代的安德森加速”，《*SIAM* 数值分析杂志》，第49卷，第4期，第1715–1735页，2011年。 [在线]。获取地址：https://doi.org/10.1137/10078356X[125]CuPy 开发团队，内存管理 — *CuPy* 用户指南，2025年，访问日期：2025年11月10日。 [在线]。获取地址：https: //docs.cupy.dev/en/stable/user guide/memory.html[126] L. Dalcin 和 Y.-L. L. Fang，“mpi4py：开发12年后的进展更新”，《科学与工程计算》，第23卷，第4期，第47–54页，2021年。[127] DMLC 项目，*DLPack*：通用内存中张量结构，*1.2*版，DMLC，2025年，1.2版，于2025年10月11日发布。 [在线]。获取地址：https://dmlc.github.io/dlpack/latest/[128] CuPy 开发团队，*CuPy* 用户指南：如何编写与 *CPU/GPU* 无关的代码，2025年，访问日期：2025年11月10日。 [在线]。获取地址：https://docs.cupy.dev/en/stable/user guide/basic.html#how-to-write-cpu-gpu-agnostic-code[129]消息传递接口论坛，“MPI：消息传递接口标准，5.0版”，消息传递接口论坛，技术报告，2025年6月，访问日期：2025年11月12日。 [在线]。获取地址：https://www.mpi-forum.org/docs/mpi-5.0/mpi50-report.pdf[130] 英伟达公司，“NVIDIA GPUDirect：提升 GPU 的数据移动与访问能力”，https://developer.nvidia.com/gpudirect，2025年，访问日期：2025年11月12日。[131] K. Taranov、F. Fischer 和 T. Hoefler，“高效的 RDMA 通信协议”，2022年。 [在线]。获取地址：https://arxiv.org/abs/2212.09134