arXiv:2303.05994v1 [physics.comp-ph] 10 Mar 2023

一种适用于多种性能平台的六维半拉格朗日算法实现

NilsSchild[1], Mario R¨ath[1], Sebastian Eibl[2], Klaus Hallatschek[1], 和 Katharina Kormann[3]

[1]马克斯·普朗克等离子体物理研究所，德国[2]马克斯·普朗克计算与数据中心，德国[3]波鸿鲁尔大学，德国

摘要

本文介绍了我们开发全动理学 Vlasov方程模拟软件应用的方案，该应用将用于探索陀螺动理学模型以外的物理现象。模拟全动理学 Vlasov方程需要高效利用计算和存储能力，因为问题的维数很高。此外，实现方案需要在物理模型方面具备可扩展性，并能灵活适配用于生产运行的硬件。我们首先介绍使用半拉格朗日算法模拟六维 Vlasov方程的算法背景。随后介绍具有性能可移植性的软件栈，它支持在纯 CPU 节点以及 AMD 或 Nvidia GPU加速节点上进行生产运行。通过介绍主内核的软件架构，我们确保了实现方案的可扩展性；使用单一代码库，该方案在 Nvidia V100图形处理器上实现了近 500 GB/s 的内存带宽，在英特尔至强金牌中央处理器上则达到约 100 GB/s。我们提供了多种节点级架构上的性能数据，并讨论已使用和仍可利用的硬件能力。最后，我们量化了基于六维网格的算法所面临的网络通信瓶颈。文章最后以离子伯恩斯坦波为例，对陀螺动理学理论以外的物理现象进行了验证。

1 引言

本文探讨了我们为强磁化等离子体在六维相空间中求解全动理学 Vlasov方程所做的工作，目标是提供一个与平台无关且灵活的半拉格朗日求解器。借助该模拟软件，我们研究常用模型无法捕捉的等离子体湍流现象，例如 Raeth 等人的文章中所述的现象。 [1]。由于磁化等离子体的常用模型（例如磁约束聚变中使用的模型）通过对时间尺度或空间尺度施加严格限制来降低 Vlasov方程的维数，因此可能会遗漏有趣的物理现象。陀螺动理学理论是该领域广泛使用的一种模型，它对回旋半径进行平均，从而消除了拉莫尔频率下的现象。新一代大型超级计算机具备存储和计算基于完整 Vlasov方程的模拟的能力，能够捕捉空间尺度较小、频率较高的现象。我们的目标是开发一种等离子体模拟应用，以便研究高频区中超越陀螺动理学的现象。

我们基于 Trott 等人开发的 `Kokkos`框架进行实现 [2], ，因为该框架拥有长期稳定的开发团队和庞大的用户群。这样，代码便能受益于 `Kokkos` 内核向未来硬件的适配，而我们也无需开发和维护性能可移植性层。我们遵循 Kormann 等人 [3] 提出的优化技术，通过实现充分利用硬件的性能能力。除了性能可移植性方面的结果，我们还运用设计模式探讨了主内核的软件架构。设计模式的理念源自 Gamma 等人 [4]。通过运用设计模式，我们实现了模块化的内核，使实现具有可扩展性且易于测试。我们通过复现离子伯恩斯坦波的色散关系来验证该实现；这类波的频率量级与拉莫尔频率相当。

1.1 与既有工作的关系

近年来，多个规模不一的软件项目采用 GPU 加速数值模拟。为了编写具备性能可移植性的代码，人们采取了多种与我们的方法密切相关或仅有松散关联的方案。

无法通过全面重写进行移植的成熟软件框架，会自行实现性能可移植性层，以支持其

主要数据结构，或采用基于指令的方法在图形处理器上执行内核。GENE [5]和 AMReX [6] 是第一种方法的示例，而 ORB 采用后一种移植技术5 [7]。我们的方法与 Cabana 工具包 [8]或面向粒子应用的 Alpine 迷你应用 [9]有关。这两者都基于性能可移植性框架创建数据结构 `Kokkos` [2]，以便采用多种共享内存技术。其他选择还包括 `Kokkos` RAJA [10], ，其已在相关研究中得到探讨[11]，以及 Alpaka [12]。

Bigot 等人 [13],Umeda 和 Fukazawa [14]以及 Kormann 等人 [3] 在文章中介绍了针对五维和六维 Vlasov方程的高维半拉格朗日算法实现。这两种代码都侧重于 CPU 扩展。Asahi 等人[15]将基于 GYSELA 代码中半拉格朗日算法的性能可移植迷你应用移植到了其他平台。Einkemmer [16] 基于间断 Galerkin 方法实现了适用于四维问题的半拉格朗日算法，并使用 OpenMP 支持 CPU、使用 CUDA 支持图形处理器。我们的工作通过 HIP 编程模型扩展了共享内存概念，从而比较了 `Kokkos` 的 OpenMP、CUDA 和 HIP 后端。Einkemmer 和 Moriggl [17]给出了 Einkemmer [16] 的实现在问题维度从四维扩展到六维时的扩展结果。我们在将六维半拉格朗日算法扩展至多个节点时遇到了类似的困难，并对通信瓶颈进行了量化。

1.2 论文结构

本文其余部分安排如下：下一节将介绍问题及算法背景。第3节讨论软件栈、所需背景信息以及`Kokkos` 在 BSL6D 实现中的用法。此外，我们还将回顾六维域分解所面临的挑战。第4节介绍基于设计模式构建的主内核（即分布式拉格朗日插值）的软件架构。第5 节和6节分别分析新代码在不同平台上的性能，包括共享内存和分布式内存性能。我们通过网络通信瓶颈对扩展结果进行定量验证。第7节展示代码能够提供超越陀螺动理学模型的物理结果。结果尤其表明，在实际应用中，半拉格朗日方法中的高阶插值模板至关重要。最后，

第8 节总结了所报告实验的结论。

2 问题表述与算法背景

2.1 Vlasov方程

Lifshitz 和 Pitaevskij 的著作 [18, 第 115 页及后续页]通过 Vlasov–Maxwell 系统描述等离子体的运动：等离子体中每种带电 q_s 且质量为 m_s 的粒子，都由其在相空间中的分布函数 f_s 描述；该分布函数在自洽电磁场和外部电磁场中演化。自洽场遵循麦克斯韦方程组。本文聚焦于静电描述，忽略自洽磁场。此外，我们考虑电子在中性化离子背景中的分布函数。采用有理化静电 CGS 单位制后，电子分布遵循 Vlasov方程

$$\begin{aligned}&\partial_t f(\mathbf{x},\mathbf{v},t)+\mathbf{v}\cdot\nabla_{\mathbf{x}} f(\mathbf{x},\mathbf{v},t)\\&+\frac{q}{m}(\mathbf{E}(\mathbf{x},t)+\mathbf{v}\times\mathbf{B}_0)\cdot\nabla_{\mathbf{v}} f(\mathbf{x},\mathbf{v},t)=0,\ \text{on}\ \Omega\times\mathbb{R}^3\times(0,T)\end{aligned}\tag{1}$$

其中 Ω 为空间区域，$\mathbf{B}_0$ 表示恒定的背景磁场，$\mathbf{E(x}\ t)$ 表示由电势 ϕ 的泊松方程确定的电场，

$$\begin{aligned}\rho(\mathbf{x},t)&=q\left(\int f(\mathbf{x},\mathbf{v},t)\mathrm{d}\mathbf{v}-n_0\right)\\-\Delta\phi(\mathbf{x},t)&=\rho(\mathbf{x},t)\\\mathbf{E}(\mathbf{x},t)&=-\nabla\phi(\mathbf{x},t).\end{aligned}\tag{2}$$

泊松方程右端为电荷密度ρ ，而 n_0 是中和背景中带相反电荷粒子的密度。

Vlasov方程（1）的特征曲线是以下常微分方程的解

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t}=\mathbf{V}\qquad\qquad\frac{\mathrm{d}\mathbf{V}}{\mathrm{d}t}=\frac{q}{m}\left(\mathbf{E}\left(\mathbf{X},t\right)+\mathbf{V}\times\mathbf{B}_0\right).\tag{3}$$

作为双曲型守恒律，Vlasov方程（1）沿相空间中的特征曲线守恒粒子分布函数 f 。以时间 t时的初始条件（x, v）求解（3），将得到的特征曲线记为（X(s; x, v t), V(s; x, v t)）。我们可以沿这些曲线向后追溯到时间 0 时的初始条件 f_0 。利用特征曲线上的守恒性质，可以将 t 时 f 的取值映射为时间 0 时 f_0 的取值。对该映射求逆，即可得到方程（1）在时间 t 时的解，并将其表示为时间 0 时初始条件 f_0 的函数。

$$f(\mathbf{x},\mathbf{v},t) = f_0(\mathbf{X}(0;\mathbf{x},\mathbf{v},t),\mathbf{V}(0;\mathbf{x},\mathbf{v},t)). \tag{4}$$

由于特征曲线取决于电场，而电场又取决于分布函数，因此该表达式无法用于实际计算解。

2.2 分步后向半拉格朗日方法

半拉格朗日方法的基本思想是对相空间进行离散化，并利用（1）的守恒性质，将分布函数从 t_m 传播至 $t_{m+1} = t_m + \Delta t$。我们用（$\mathbf{x}_i$，$\mathbf{v}_j$）表示网格点。应用半拉格朗日方法包含以下两个步骤。

首先，以点（3）作为初始条件，将特征方程从t_{m+1} 到 t_m 反向求解。其次，将所得的特征曲线代入映射（4）的右端，以更新时刻 f 网格点（$\mathbf{x}_i$，$\mathbf{v}_j$处的值，并利用特征线脚点处时刻 t_{m+1} 的 f 值。 t_m

$$f(\mathbf{x}_i,\mathbf{v}_j,t_{m+1}) = f(\mathbf{X}(t_m;\mathbf{x}_i,\mathbf{v}_j,t_{m+1}),\mathbf{V}(t_m;\mathbf{x}_i,\mathbf{v}_j,t_{m+1}),t_m). \tag{5}$$

通常， f(**X**(t_m; $\mathbf{x}_i$, $\mathbf{v}_j$, t_{m+1}), **V**(t_m; $\mathbf{x}_i$, $\mathbf{v}_j$, t_{m+1}) t_m)在时间 t_m 并不位于网格点上。因此，必须使用时间 t_m 位于网格点上的值{f($\mathbf{x}_i$, $\mathbf{v}_j$, t_m)} 进行插值。

如前所述，半拉格朗日方法需要求解方程（3），并进行六维插值。另一种做法是，按照 Cheng 和 Knorr 的文章所述，利用分裂方法将 $\mathbf{x}$ 和 $\mathbf{v}$ 平流算子分离 [19]；McLachlan 和 Quispel [20]。我们采用 Strang 分裂法，将 $\mathbf{x}$ 和 $\mathbf{v}$ 平流分开，该方法在时间上具有二阶精度。分离$\mathbf{x}$和 $\mathbf{v}$ 平流的优点是，在速度平流步中，求解

速度步的特征方程时所需的电场保持不变。因此，可以解析地求解各子系统的特征方程。此外，剩余的三维插值步骤只包含可交换算子。通过李分裂，可将这些可交换算子进一步分解为三个一维插值。

Vlasov–Poisson系统还需要求解泊松方程（2），以确定电场 **E**。与平流方程所需的计算量相比，求解这个三维问题的计算量微不足道。目前仅实现了周期边界条件。泊松问题采用基于快速傅里叶变换的伪谱方法求解。

2.3 拉格朗日插值

一维插值可以采用多种插值方案。我们遵循 Kormann 等人 [3] 的文章，采用拉格朗日插值，因为它具有局部性和高精度。由于平滑性更好，样条插值在 Vlasov方程的模拟中通常更受青睐。然而，这种插值是全局的；此外，正如 Kor-mann 等人 [3], 所讨论的，即使采用局部形式，其通信开销也使样条插值在大规模模拟中的竞争力较弱。另一种选择是 Crouseilles, N.等人 [21] 所描述的不连续插值，但由于其平滑性较差，需要使用大量点后才具有竞争力。

现在考虑一维拉格朗日插值，设 x_j, j =1，.....， N 为网格点，彼此等距，间距为 Δx，而 α 是 x_j 的某个位移。我们希望根据 f 在 $f(x_j)$ 周围的取值，计算插值函数 $f(x_j+\alpha)$，并假设 $|\alpha| < \Delta x$。该插值函数可与局部模板算法进行比较。此外，我们用 ℓ_i^q 表示插值函数中含有 q 个节点的 $(q-1)$ 阶拉格朗日多项式。我们区分两种情况：

- 当 q 为奇数时，插值由下式给出

$$f(x_j+\alpha) = \sum_{i=j-(q-1)/2}^{j+(q-1)/2} \ell_i(\alpha) f(x_i) \tag{6}$$

- 对于偶数 q，我们考虑一个以待插值点 $x_j+\alpha$ 为中心的插值模板，使得插值为

由下式给出：

$$f(x_j+\alpha)=\begin{cases}\sum_{i=j-q/2}^{j+q/2-1} l_i(\alpha)f(x_i) & \alpha\le 0\\ \sum_{i=j-q/2+1}^{j+q/2} l_i(\alpha)f(x_i) & \alpha>0\end{cases} \tag{7}$$

我们注意到，第二个模板以特征线的足点为中心，因此属于迎风型格式，通常能带来更好的结果。更高阶的模板能够更好地分辨物理现象，如第7节图9所示。同时，它们也会增加计算复杂度和通信开销。

如第 α 2.2 节所述，从特征线得到的插值偏移 是一个低维子集 $D \subset \mathbb{R}^6$的函数，其中 $\dim(D) \in \{1,2,3\}$，对应的映射为 $\alpha : \mathrm{D} \to \mathbb{R}$。定义插值偏移最一般的方法是将其设为完整相空间 $\mathbb{R}^6$（而不只是 D）的函数。使用映射 $\alpha\colon \mathbb{R}^6 \to \mathbb{R}$， $\alpha = \alpha(\mathbf{x}, \mathbf{v})$ 可以简化软件架构中的抽象。在小节4.2中，我们将介绍一种软件架构，用于通过多个插值偏移 α 以及不同宽度的模板 q实现一维插值。

2.4 算法步骤的特征

在讨论我们新开发的 BSL6D 代码中上述算法的实现之前，先来总结一下在六维域被分解为块并借助消息传递接口进行分布的情况下，算法包含的各个步骤。图 1 总结了这些步骤。关于计算，我们区分对六维数据（分布函数）执行的步骤、对三维数据（场）执行的步骤，以及二者之间的映射。关于通信，我们区分点对点通信、全对全通信和全归约；后两种通信发生在六维 MPI 分布的子组中。

3 软件栈与并行化

3.1 软件栈

开发用于高维问题的可扩展应用程序，需要运用共享内存和分布式并行方面的概念。此外，还必须

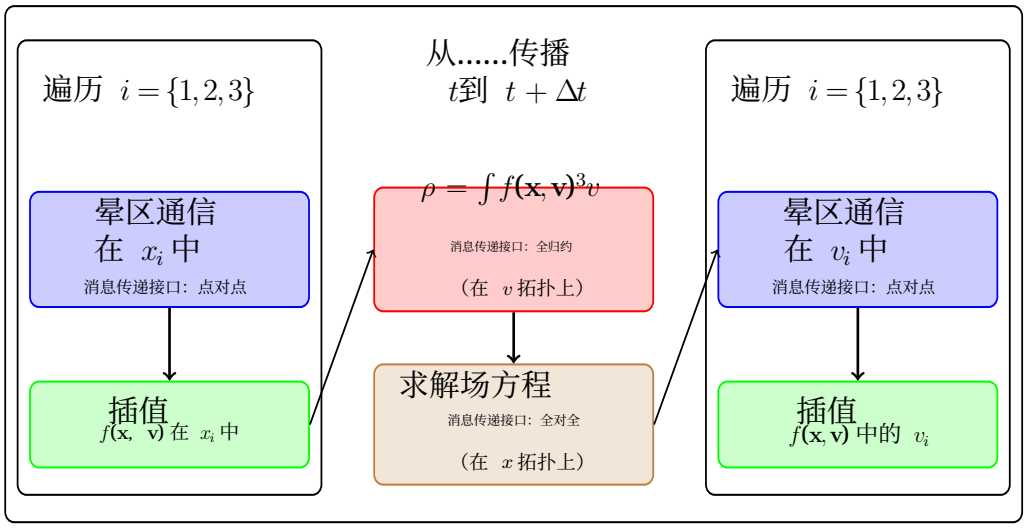

图1：通过一个时间步传播分布函数所需的算法步骤 Δt。颜色编码突出显示了在第6.1小节中单独考察的算法步骤。

注意应用程序的输入和输出。可靠的应用程序必须经过严格测试。
BSL6D 代码依赖若干第三方库来应对这些挑战。下面将介绍所需的库。

该代码的一个重要特性是实现不依赖于节点级架构。BSL6D 算法的生产运行涵盖仅配备 CPU 的节点以及 GPU 加速节点。访问这些架构需要使用不同的共享内存概念，例如`OpenMP`、 `CUDA` 或 `HIP` 。性能可移植性框架 `Kokkos` [2] 用于将节点级架构从主要实现中抽象出来。

`Kokkos` 是一种基于库的方法，用于实现性能可移植性，借助`C++`。下一小节将介绍 BSL6D 实现如何使用 `Kokkos` 。迄今为止，BSL6D 代码已在表1所列的不同共享内存架构上成功测试。

虽然 `Kokkos`为共享内存范式提供了抽象，但要扩展到大型计算集群，就需要分布式内存概念；对此，我们依赖广泛使用且定义明确的 MPI 标准 [22]。输入和输出基于 HDF5 库 [23]。求解泊松方程需要一个快速傅里叶变换库

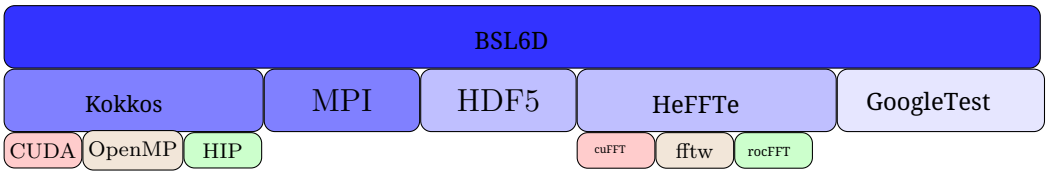

图 2：BSL6D 代码的软件栈，包括用于评估性能可移植性的已测试后端

（2）需要使用该库。为此，我们采用 HeFFTe 库 [24]，它为不同供应商专用的快速傅里叶变换库提供接口，例如 cuFFT、rocFFT 和 FFTW。此外，HeFFTe 还为前述求解器添加了分布式变换，使我们能够将分布式傅里叶变换完全卸载给 HeFFTe。完整的软件栈如图 2所示。

为确保应用程序的灵活性和可靠性，我们使用 GoogleTest C++ 库，并采用测试驱动开发方法。

3.2 Kokkos性能可移植性框架

如上一小节所述，通过使用 Kokkos [25]，该实现可在多种节点级架构或共享内存概念之间实现性能可移植性。下一节将简要回顾本文所需的 Kokkos 功能。详细说明请参见 Kokkos 文档 [26； 25]。

BSL6D 实现中的内存管理依赖于 Views，由 Kokkos 引入。它们类似于具有共享所有权的多维数组，这一概念最早由 Edwards 等人 [27]提出。存储在 View 中的数据位于 MemorySpace中。使用内存空间 Kokkos 可以抽象不同的内存资源，例如 GPU 的高带宽内存（HBM）、CPU 的 DRAM，或诸如 Cuda Unified VirtualMemory (UVM) 之类的内存概念。编译时， Kokkos 会设置一个默认内存空间；如果未显式指定内存空间， View 中的数据便存储于该空间。默认内存空间取决于 Kokkos 所针对的后端。如果 Kokkos 使用 CUDA 后端编译，则其中的数据默认存储在 Nvidia GPU 上。如果 Kokkos 使用OpenMP 后端编译，则数据存储在 CPU 内存中。

本文中使用的一种重要内存空间是 ScratchSpace，它是 GPU 上的共享内存。这种可管理的内存允许显式地将数据预取到缓存内存中，从而能够优化内存访问，例如优化 Harris [28]所述的具有非连续内存访问的转置操作。暂存空间可在TeamPolicy 中访问，它是 ExecutionPolicies 提供的 Kokkos之一。执行策略定义了如何在并行内核中遍历索引范围。支持嵌套的并行层级，例如在 CPU 上进行向量化，或使用 CUDA 块，从而能够调整块大小，在运行时启用针对特定架构的优化技术。

对于 ViewS，内存空间有一个类似的概念：执行策略与 ExecutionSpaces。执行空间定义了线程执行工作的地点。执行策略的执行空间通过 C++模板参数定义。如果未指定模板参数， Kokkos 会提供默认执行空间。在 BSL6D 实现中，内核在默认执行空间中执行。因此，不仅计算开销大的内核，所有内核都会得到加速。此外，使用兼容图形处理器的后端编译 Kokkos ，可以减少 CPU 与图形处理器之间的内存拷贝，因为内核访问的 ViewS 在整个计算过程中也可以保留在图形处理器内存中。

使用 Kokkos 实现 BSL6D 算法时，最后一个需要考虑的问题是如何将数据输入和输出到硬盘。必须谨慎处理 BSL6D 算法不同运行之间的数据交换，以及模拟数据的后处理。 View使用 MemoryLayouts将索引映射到内存地址。内存布局决定了哪些索引在内存中连续存储，哪些索引采用跨步访问。内存布局取决于编译 Kokkos 时所针对的并行后端。因此，在 BSL6D 实现的各次独立运行中，内存布局并不固定，我们必须决定如何处理存储在硬盘上的数据。我们需要记录 View的存储方式，或者固定硬盘上所存数据的布局。从用户的角度来看，使用固定布局比处理各种不同的布局更方便。Mathematica 和 NumPy 中定义的多维数组是我们使用的主要后处理工具；两者在新建数组时默认都采用 C 布局[29； 30]。因此，我们遵循这一惯例，使用 C 布局将数据存储到磁盘上。这对应于由 Kokkos 定义的 LayoutRight ，其中最右侧的索引在内存中连续存储。根据 Kokkos 后端，可能需要进行转置

步骤，才能将数据正确写入磁盘。

3.3 六维域的域分解

在六维相空间网格上进行生产运行需要大量内存。因此，若要进行高分辨率运行，就必须将网格分布到计算集群的多个节点上。分布式内存并行性通过消息传递接口实现。BSL6D 算法基于消息传递接口标准的笛卡尔拓扑 [22, p. 319] 。通过将每个维度的点数限制为对应笛卡尔拓扑维度中进程数的整数倍，确保负载均衡。

在分布式网格上需要计算拉格朗日插值多项式。没有任何一个 MPI 进程拥有完整的相空间网格。我们将处理器本地的数据称为计算域。在计算域边界处计算模板，需要用到位于其他处理器上的网格点。因此，各个独立进程必须交换网格点。我们将这些区域称为幽灵单元。实现幽灵单元有两种方式：可以扩展 `View` 包含计算域数据的 `View`，以容纳光环区域；也可以分配一个新的 [3]来存储光环区域所需的数据，Kormann 等人提出了这种做法 [31]。将幽灵单元纳入存储分布函数的缓冲区，是一种直接的实现方式，例如 Hager 和 Wellein [32]以及 Datta 等人 N 采用的方式。这种方法的缺点是，在高维六维情况下内存需求很大。假设一个六维超立方体的每个维度有 N 个点，且每个方向的光环宽度为 w ，那么分配的 View 将在每个 MPI 进程上包含 $(N + 2w)^6$ 个点。计算域与 View 大小的比值如图3 中的虚线所示。

如果将分配的内存分为一部分 `View` 用于计算域，另一部分 `View` 用于光环区域，则可以进行两项内存优化。一方面，六维幽灵域不需要分配角点。在进行一维插值时，插值过程中不会用到幽灵域中的角点。因此，在当前实现中，如果为光环区域的角点分配内存，就会造成浪费。另一方面，在一维插值过程中，只需要六个维度中某一个维度的光环区域。因此，只需为最大的幽灵域分配内存，并在各光环区域之间共享已分配的内存，以供

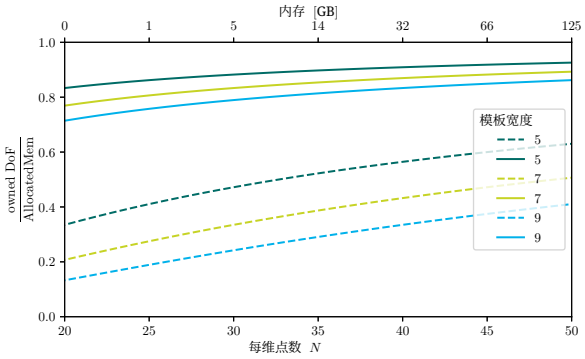

图3：拥有的自由度与内存需求之比随每维点数的变化 N （横坐标）。对于虚线，该比值为$N^6/(N+2w)^6$，未进行内存需求优化；对于实线，该比值为 $N^6/(N^6+2N^5w)$，其中包含了小节 3.3中的内存优化。第二个横坐标给出了拥有的自由度所需的存储量，以展示六维相空间网格的高内存需求。

不同维度上的插值。在每个维度有N 个网格点、边界宽度为 w 的六维超立方体中，已分配的 `View`总计为 N^6+2wN^5。图3中的直线表示计算域大小与已分配的内存大小之比，内存使用效率显著提高。

这种分离的一个缺点是实现复杂度增加。插值模板应遍历连续内存，但晕区和计算域不再位于连续内存中。因此，这种内存优化使得无法像 Hager 和 Wellein [31]、Datta 等人[32]的工作中那样，对连续内存上的模板使用单个一维循环来执行迭代。下一节将结合插值算法的软件架构，讨论这一问题的解决方案。

4 插值内核的软件架构

4.1 性能考量

在讨论主内核的软件架构（一维拉格朗日插值）之前，先考察该算法中两个对性能至关重要的组成部分。

首先，我们需要确保应用拉格朗日插值模板时能够连续访问内存。由于光环区域和计算域的数据在内存中并非连续存储，因此无法通过一组嵌套循环进行迭代。Kormann 等人已提出这一问题的解决方案。 [3]将光环区域和计算域的二维切片预取到连续内存中。然后，插值模板便可作用于存放在连续内存中的预取二维切片。二维切片 `View` 必须足够小，能够装入高级缓存，从而确保每个数据点只写入和读取一次。图4 展示了这种方法。箭头表示迭代方向。这里涉及两个维度：插值维度以及在内存中连续存储的维度。因此，每次插值都必须调整迭代模式。采用 Trott 等人提出的暂存内存 [2]由 `Kokkos` 在 `TeamPolicy` 中提供，可确保数据始终靠近缓存。

第二项优化涉及分配的 `Kokkos`暂存内存量。在 GPU 上，暂存内存对应于共享内存。GPU 共享内存容量有限，只有几 KB 的显式管理缓存内存。大量使用共享内存会减少 GPU 上的活动线程束数量。共享内存使用量增加时，活动线程束数量会减少，GPU 计算能力的利用率也会随之降低。通过 NSight Compute 分析了共享内存使用情况，发现共享内存使用量限制了 GPU 的占用率。因此，通过将插值后的网格点直接写回保存计算域的 `View` ，而不是像 [3] 中那样将数据写入另一个缓存数组，来减少暂存内存的分配量。

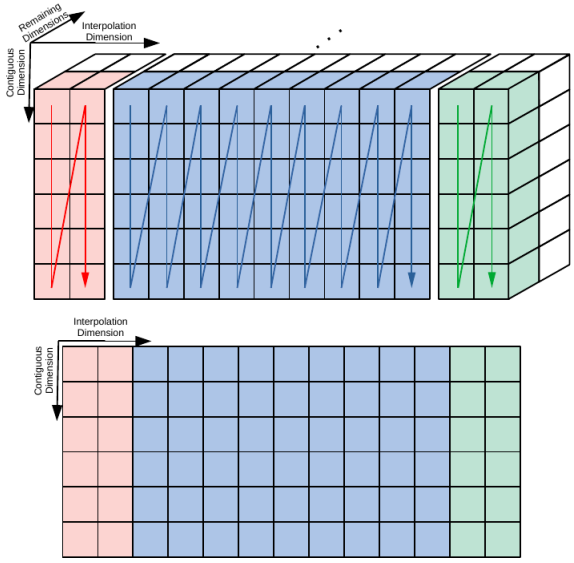

图4：插值所需预取的示意图。从六维视图中收集二维切片，沿连续维度将其排列为一个连续二维视图。在图形处理器上，这种方法可与 Harris 的 Nvidia 矩阵转置示例中共享内存的用法相比较 [28]。

4.2 拉格朗日插值的软件架构

本节介绍第2.3 小节中的拉格朗日插值软件架构。由于拉格朗日插值是 BSL6D 算法中计算开销最大的部分，因此需要高效的实现。同时，新的物理模型（通过特征方程定义插值偏移 α （3））应易于添加。此外，还需要支持不同阶数的拉格朗日多项式进行插值。最后，添加新的模板和特征时，不应要求了解第4.1小节中的迭代模式。下面将讨论算法1的各个组件，该算法以伪代码形式呈现了所实现的算法。

通用算法的结构与 Gamma 等人所著书籍 [4] 中描述的模板方法相同。模板方法定义一种骨架算法，并将原始操作委托给独立的类或函数。对于插值，迭代模式提供了骨架算法。原始操作可用于抽象 α 和模板。这些原始操作的抽象需要共享通用接口。

首先讨论用于提供骨架算法的迭代模式。第 10 至 12 行和第 13 至 15 行从 `SubView`暂存内存 `View`中预取二维切片，这些暂存内存称为 `local` ，如图 4所示。最后两个嵌套循环（第 16 至 19 行）最终执行实际的插值。这些迭代针对待插值的索引以及在内存中连续排列的索引进行。外层循环遍历其余索引，以完成相空间网格中每个点的插值。

现在需要为原始操作定义通用接口。第一个原始操作计算偏移量 α 以执行插值。如第 2.3 小节所述，偏移量通过一个通用接口定义：该接口接收全部六个索引，以确定待插值的网格点，并返回该网格点的偏移量 α 。利用 α ，即可更新二维切片中对应的网格点。由于 `View local` 在内存中连续排列，因此可以根据待插值点推导出所需的相邻点。因此，模板只接收待插值点的内存地址以及 α ，并返回插值结果。由此，我们为原始操作定义了两个接口。

现在给出了 Alg.1 的类结构。声明了一个抽象特征

类，其中包含虚函数 `shift` ，以及一个包含虚函数的抽象模板类 `interpolate` 。所有继承这些接口的类都可以用于插值类。插值类将原始操作与插值维度结合到 Alg. 1 中。因此，类 `Interpolation`会对六维分布函数进行插值。可以使用 Gamma 等人 [4] 所述的建造者模式创建具体的插值对象，客户端可通过该模式分别设置所有组件。通过使用静态多态，我们避免了抽象带来的性能损失。插值类包含 `C++` 用于具体特征类、具体模板类和插值维度的模板参数。具体插值类的实例化通过建造者模式中的模板元编程完成。为简洁起见，本文省略了这些实现细节。完整的软件架构如图 5 所示。

添加新的特征线时，无需更改迭代模式或模板。添加新模板时也是如此。我们的结构还允许为每个组件单独编写测试，从而大大简化测试工作。下一节将评估该实现的共享内存性能。

5 共享内存性能测量

本节分析前文介绍的软件架构的性能，该架构便于扩展和测试实现。测试结果通过在六维分布函数的所有六个维度上使用随机偏移执行一维拉格朗日插值获得$\alpha/\Delta x \in [-1, 1]$。插值采用宽度为三、五、七和九的非均匀模板执行。由于生产计算是在这些系统上运行的，我们选择 Nvidia GPU 和 Intel CPU 作为目标架构。AMD GPU 上的结果则展示了该实现的硬件无关性。生成数据所用的系统包括 Cobra GPU 和 CPU 分区，以及一台我们获准使用的、搭载 AMD MI250 GPU 的系统。技术规格见表 1。

我们通过硬件计数器测量实现的性能，这些计数器使用以下工具收集。在 CPU 上使用 `LIKWID`toolkit [33] ， `NSight Compute` [34]用于 Nvidia GPU，而 `rocprof`[35]用于 AMD GPU。使用这些工具，我们收集运行时间、数据量

算法 1： 基于抽象化位移和插值模板的分布函数一维插值
，假设 View 的第一个维度在内存中连续。
View 的维度在内存中是连续的。

输入： 分布函数 f(**x**，**v**，0)、下边界 low(**x**，**v**)，
上边界 up(**x**，**v**)、特征线、模板
输出： 分布函数 f(**x**，**v** Δt)

针对 $\{i,j,k,l\} < \{N_{outer,1}, N_{outer,2}, N_{outer,3}, N_{outer,4}\}$ **执行**
  本地 = Kokkos::View($N_x, N_z + 4$)
  sliceLow =获取二维_切片(low，i，j，k，l)
  sliceF = 获取_2D_切片(f，i，j，k，l)
  sliceUp = 获取二维_切片(up，i，j，k，l)
  // 考虑边界宽度
  // sizeof(sliceLow(0,:)) = sizeof(sliceUp(0,:))
  带宽 =sizeof(sliceLow(0,:))
  对于 $i_1 < bw$ **执行**
    对于 $i_0 < N_{continuous}$ **执行**
      局部(i_0，i_1) = 切片下界(i_0，i_1)
      局部(i_0，$N_{\text{continuous}}$ + 带宽 $+ i_1$) = 切片上界(i_0，i_1)
  对于 $i_1 < N_{interpolation}$ **执行**
    对于 $i_0 < N_{continuous}$ **执行**
      局部(i_0，带宽 $+ i_1$) =切片F(i_0，i_1)
  对于 $i_1 < N_{interpolation}$ **do**
    对于 $i_0 < N_{continuous}$ **do**
      α =特征偏移(i_0，i_1，i，j，k，l)
      sliceF(i_0，i_1) = 模板.插值(local(i_0，i_1),α)

以及主存与缓存之间传输的数据量和带宽。AMD 的 `ROCM` 分析器目前尚未提供用于收集带宽的派生指标。不过，使用 `LIKWID` 和 `NSight Compute` 所采用的公式计算带宽：将从主存读取和写入主存的数据量除以内核的运行时间。

所有计算均采用每维 37 个点，因此相空间网格中共有37^6 个网格点。

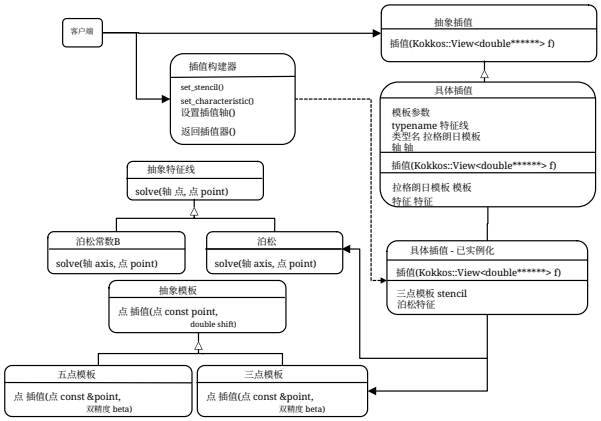

图5：使用局部模板执行插值的插值类图。所有具体插值的组合都在编译时创建。插值算法通过伪代码展示，以指出算法1中的模板方法。

5.1 平流内核运行时间测量

首先比较所分析平台上各内核的运行时间表现。结果绘制在图 6的第一行中。纵坐标表示运行时间，横坐标表示执行插值的维度。可以观察到两个显著特征。

由于 BSL6D 算法具有很高的并行化潜力，两种 GPU 的速度都显著快于 CPU。与 CPU系统相比，Nvidia GPU 的运行时间缩短了 3.5 倍至近 7.5 倍，而 AMD GPU 的运行时间缩短了 4 倍至近 8 倍。

其次，我们可以观察到，在 CPU 上，从维度 v_3到维度 x_1 时，运行时间不断增加；而在 GPU 上则恰好相反，运行时间从 x_1 到 v_3时不断增加。这可以用插值维度的步幅较长来解释。在 CPU 上，由于 `View` 的默认内存布局，连续维度为 v_3 。在 GPU 上， x_1 维度在内存中连续存储。离连续维度越远，步幅就越大。

比较 ta- 的技术规格中的内存带宽

系统	COBRA	RAVEN	AMD 系统
CPU	2x 英特尔至强 Gold 6148	2 台英特尔至强 Platinum 8360Y	2 台 AMD EPYC 7763
	Skylake	Ice Lake	
内存（DDR）	96GB	512 GB	512 GB
内存带宽 - CPU（BabelStream）	100 (74) GB/s	135 (150) GB/s	—
单向网络带宽	OmniPath	InfiniBand HDR	—
CPU 节点（OSU基准测试 H-H）	12.5 GB/s	12.5 (10) GB/s	
GPU	2x Nvidia V100	4x 英伟达 A100	4x AMD MI250
内存（HBM GPU）	32 GB	40 GB	128 GB
内存带宽 - 图形处理器（BabelStream）	900 (830) GB/s	1550 (1360) GB/s	3200 (2400) TB/s
图形处理器直连（OSU 基准测试 D-D）	—	100 (80) GB/s	—
网络带宽，单向	OmniPath	InfiniBand HDR	—
GPU 节点（OSU 基准测试 D-D）	12.5 (GB/s)	25 (25) GB/s	—

表 1：用于性能评估的硬件规格。在 AMD 系统上，我们仅考虑图形处理器的能力，因此这里只给出相关配置。

ble 1 若将 Intel Gold CPU 与 Nvidia V100 GPU 的规格进行比较，或者与 AMD MI250 GPU 的半个核心相比，可实现约 9 倍的加速。因此，还需要进一步优化，才能充分发挥图形处理器的加速潜力。下一小节将进一步介绍具体的性能指标。

5.2 平流核的性能分析

本小节将考察内存带宽和缓存行为。内存带宽通过将主存与缓存之间传输的数据量除以内核运行时间来计算。缓存情况则通过将预期传输的数据量与硬件计数器测得的数据传输量进行比较来分析。若缓存达到最佳状态，这两个数值的比值应为 1。对于 NVIDIA GPU，我们不得不跳过宽度为 9 的模板，因为使用 `NSight Compute` 收集数据时，内存需求超出了可用内存。

三个不同系统的带宽见图 6 的第二行。该图的结构与上一小节使用的运行时间图相似。纵坐标表示带宽（BW），横坐标表示插值维度。黑色水平线标示两种 GPU 在技术规格中给出的峰值带宽。CPU 图中的黑线是 MPCDF 在 Cobra 集群上测得并在内部传达的带宽。未能找到 Intel CPU 供应商提供的带宽规格。灰线则通过使用 Deakin 等人开发的 BabelStream 进行简单的 Stream 基准测试得出。 [36] 使用 `Kokkos`后端构建，作为

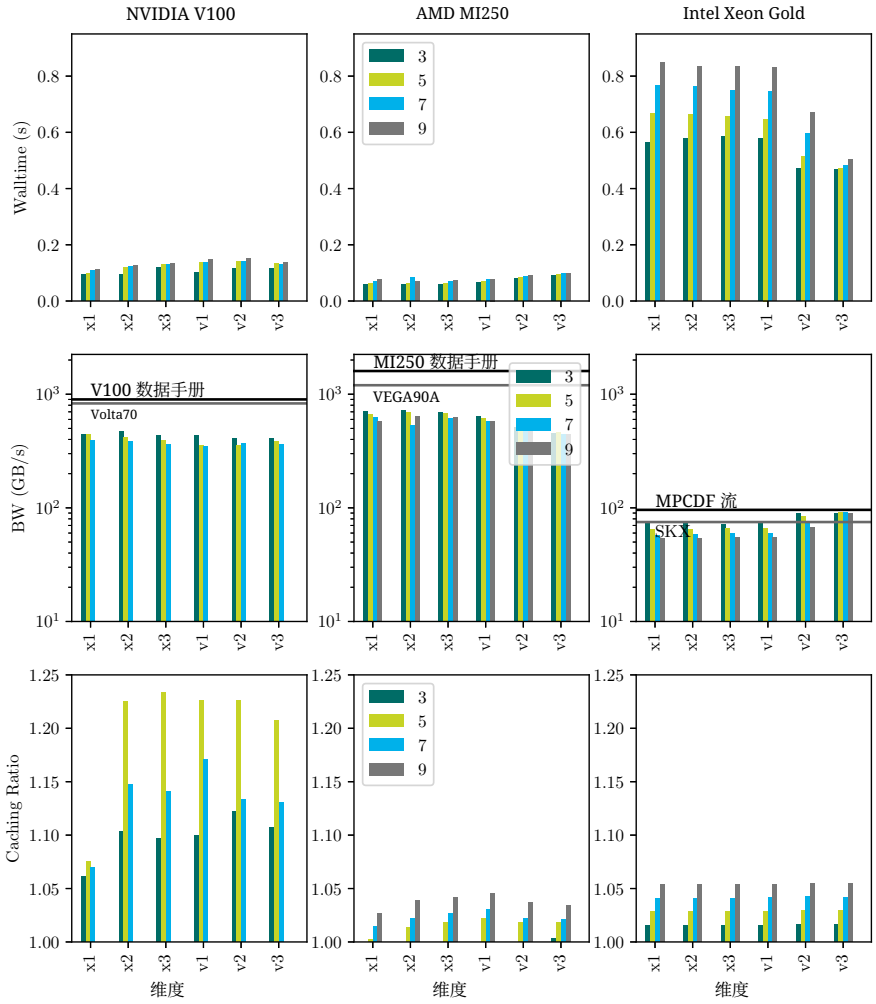

图6：用于评估 BSL6D 实现的性能可移植性的指标。我们比较了所有六个相空间维度中插值的运行时间、内存带宽和读写数据缓存比。内存带宽通过黑线与 V100 和 MI250 单个核心的技术规格进行比较。Intel Xeon Gold 的黑色参考线与 MPCDF 执行并在内部传达的 Stream 基准测试进行比较。灰线对应于使用 BabelStream 和 `Kokkos` 后端运行的结果。构建 BabelStream 时使用的 `Kokkos` 架构关键字作为灰线的标签。

可达到的带宽。根据 BabelStream 基准测试的结果，我们选取了所有系统上 Triad 流所达到的最高带宽。AMD MI250 本质上是在单个设备上包含两个独立的 GPU。由于此例中 AMD 仅使用了一个 MPI 进程，因此数据与技术规格中所列带宽的一半进行比较。

在这两种 GPU 系统上，实际带宽都比峰值带宽低约一半。BabelStream Triad 基准测试达到了制造商规格的 90% 以上。对于观测到的系统，CPU 系统取得了最佳性能。关于 GPU 性能，我们得出了两点观察结果。暂存内存（已在小节4.1中讨论）若分配量较大，可能会降低性能。此外，`Views` 和`ExecutionPolicies` 在 GPU 上的寄存器消耗较高。对于 `Views`，其寄存器使用量较高可以用整数运算来解释，因为将多个索引映射到特定内存地址需要进行这类运算。因此，暂存内存和寄存器使用量会降低 GPU 的占用率，而整数计算则需要 GPU 执行额外的指令周期。

最后，我们来考察图中第三行实现的缓存比率6。缓存比率是缓存与主存之间传输的数据量，除以预期传输的数据量所得的比值；后者假设分布函数的每个值只从内存读取一次并写入一次。AMD GPU 上的数据可能不完全可靠，因为对于宽度为 3 的模板，曾出现缓存比率小于 1 的情况，而这是不可能的。在 CPU 上，我们几乎达到了最佳缓存比率。重复读取的数据不到 5%。Nvidia GPU 的缓存效果则不那么理想。不过，仍有超过 75 % 的数据得到了有效缓存，且只读取了一次。

6 分布式内存性能测量

由于六维问题运行高分辨率模拟需要大量内存，因此必须使用多个节点。要提高网格分辨率，一个值得关注的指标是弱扩展。

接下来，我们将考察 BSL6D 的运行时间表现。

# MPI	# CPU	#GPU	每维 MPI 数 x1 到 v3					
1	18	1	1	1	1	1	1	1
2	36	2	2	1	1	1	1	1
4	72	4	2	2	1	1	1	1
8	144	8	2	2	2	1	1	1
16	288	16	2	2	2	2	1	1
32	576	32	2	2	2	2	2	1
64	1152	64	2	2	2	2	2	2

表2：弱扩展所用的设置，每维包含 32^6 个网格点。

在多个节点上运行模拟时的代码。我们将在表 1 中介绍的 Raven 集群的两种节点级架构上进行扩展示例。这两种节点级架构可用于比较基于加速器的系统与基于 CPU 的系统的运行时间表现：前者包含通过 GPU 直接通信技术连接的 Nvidia A100 图形处理器，后者包含 Intel IceLake 处理器。

模拟按以下方式设置：在全部六个维度中，每个 MPI 进程使用 $N = 32$ 个点。弱扩展从单个 MPI 进程开始，且不划分分布式维度。每一步都通过再并行化一个维度，使 MPI 进程数加倍。弱扩展设置如表 2 所示。两个 MPI 进程在该维度上的间距从 v_3 增至 x_1 维度。因此，在并行运行的两个最高维度中，我们可以使用 GPU 直连通信。

所有模拟均采用七点模板，使插值维度的每个光环区域中都包含三个点。计算时间与通信时间之比将在小节 6.2 中进行分析。

6.1 扩展

我们首先考察代码在多个 CPU 和 GPU 节点上的扩展情况。前文所述实验的扩展结果见图 7左侧的图。在弱扩展过程中，我们观察第2.4小节中定义的四类行为。两幅图均展示了平流、光环通信、求解三维问题以及将分布函数归约为密度所需的运行时间。平流的运行时间

在所有运行过程中都保持不变。这是预期之中的，因为弱扩展时每个消息传递接口进程的问题规模保持不变。并行度的提高自然会增加包含消息传递接口通信在内的各个组件的开销。图1展示了三种不同类型的消息传递接口交互。首先，光环通信属于点对点通信。其次，将分布函数归约为密度时，需要对 $\mathbf{v}$ 的所有分布式维度进行归约。求解所有三维问题基于快速傅里叶变换算法。这要求在 $\mathbf{x}$ 的所有分布式维度上进行消息传递接口全对全通信。与图 ρ 和$\mathbf{E}$ 的后两种类型在所有步骤中的开销都可以忽略不计；与图7的结果相比，光环通信和平流的开销更为显著。因此，接下来我们将重点关注光环通信。可以看到两个区段中，光环通信的运行时间以固定斜率线性增长。第一个区段是从一个消息传递接口进程增加到四个，此时只需在不同消息传递接口进程之间进行节点内通信。在图形处理器上，由于采用了图形处理器直连通信，斜率仍然较小。第二个区段从节点内通信转为节点间通信时开始，即消息传递接口进程数达到八个时。在此处斜率变得更大，因为节点间带宽远低于节点内带宽。特别是从图形处理器直连技术转为节点间通信时，带宽降低导致光环通信所需时间显著增加。

尤其是在图形处理器的情况下，我们可以观察到运行时间主要由通信耗用。但对于 CPU 实现而言，如果有两个以上的维度需要执行节点间通信，通信时间也会超过插值计算所需的时间。下一小节将量化通信瓶颈。

6.2 网络与计算硬件规格的比较

在上一小节中，我们观察到，仅需少量节点，通信所耗时间就已超过平流的计算时间。因此，我们来考察 MPI 通信的硬件限制。本小节将比较处理单元的计算性能与网络容量。

平流的局部更新公式可以与模板进行比较

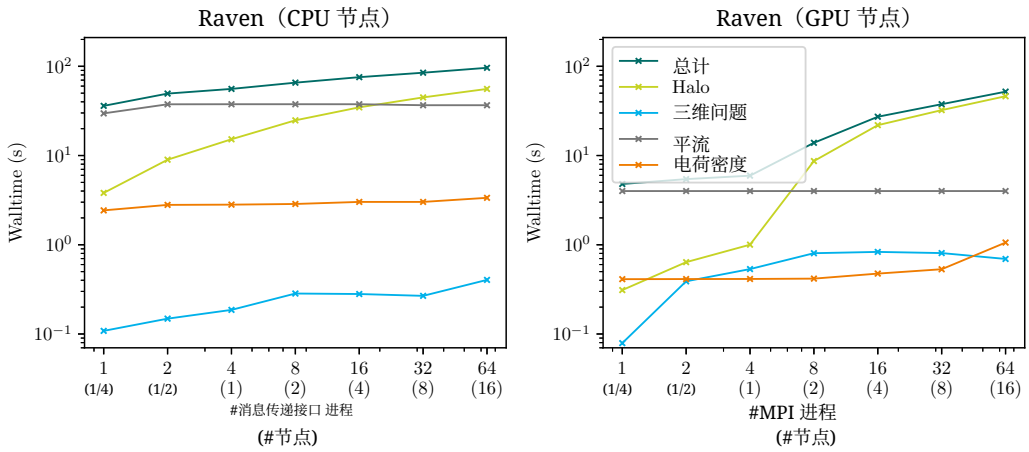

图 7：左图展示 BSL6D 代码扩展至多个 CPU 节点的情况，右图展示扩展至多个 GPU 节点的情况。

算法，如第2.3 小节所述，通常受内存限制。在 Williams 等人提出的屋顶线模型中，[37] 受内存限制的算法，其性能受主存与处理单元之间的数据交换所限制。如果算法每从内存读取一个字节所执行的浮点运算次数（也称为运算强度）低于机器平衡，则该算法受内存限制。机器平衡是指受内存限制的算法转为受计算限制时的算术强度。

如果现在使用消息传递接口（MPI）引入分布式内存概念，就必须考虑另一个屋顶线模型中未定义的瓶颈。如果交换光环区域的数据所需时间超过对所有自由度应用模板所需的时间，那么平流的运行时间就可能受光环通信限制。我们通过计算两个部分理论上可达到的最短运行时间，比较内存受限和通信受限这两种瓶颈。下面给出两个表达式，通过将传输的数据量除以带宽来计算理论运行时间。

为此，我们再次考虑一个超立方体，每维有 N 个点，光环区域的宽度为 w。执行

插值所需的最短运行时间为

$$t_{\text{interpolate}} = \frac{(N^6 \cdot (N_{\text{read}} + N_{\text{write}}) + N^5 \cdot w \cdot N_{\text{read}} \cdot N_{halo}) \cdot 8\,\text{B}}{\text{BW}_{\text{PU}}} \tag{8}$$

其中 BW_{PU} 是 $[\text{BW}_{\text{PU}}]$ =B/s 中处理单元的内存带宽， N_{read} 而 N_{write} 定义了插值过程中分布函数从内存读取和写入的频率。方程（8）分子中的第一项考虑分布函数的内存传输，第二项则考虑光环区域。 N_{halo} 计算插值函数需要单独的光环。通过 MPI 进行光环通信也可以用类似的公式表示

$$t_{\text{comm}} = \frac{N^5 \cdot w \cdot (N_{\text{read}} + N_{\text{write}}) \cdot N_{halo} \cdot 8\,\text{B}}{(BW_{\text{Network}}/N_{\text{MPI}})} \tag{9}$$

这里我们考虑与 $t_{\text{interpolate}}$ 相同的配置，区别在于节点的带宽 $\text{BW}_{\text{Network}}$ 必须由固定在该节点上的 MPI 进程共享。这两个理论运行时间中的较大值决定了该实现的运行时间。

图8 中绘出了方程（8）与（9）的比值，针对的是直线部分。比值大于 1 时，性能受插值限制；小于 1 时，问题受网络通信限制。这些直线根据上述弱扩展配置计算得出，使用宽度为 7 的模板， $w = 3$， $N = 32$ 并采用表1 中 Raven 集群的 OSU基准测试规格。BW_{PU} 的数值是在 Raven 上针对 A100 测得的，方法与第5节相同，结果也一致：我们达到 CPU 峰值带宽的 90% 和图形处理器峰值带宽的约 50%。只有在节点内通信的情况下，算法才受插值限制。一旦需要节点间通信，平流就会严重受网络通信瓶颈限制。

这使我们能够将模型与扩展结果进行比较。我们计算了时间比值的平均值和标准差 $t_{\text{interpolate}}/t_{\text{MPI}}$ ，用于绘制 $N = 32$ 时，结果见图7，并在图8的图表中添加了相应的点。我们可以看到，先前提出的通信瓶颈模型与测量数据相当吻合。这一结果表明，该算法无法在大型计算集群上实现良好扩展，因为网络通信瓶颈会导致扩展效率显著下降。

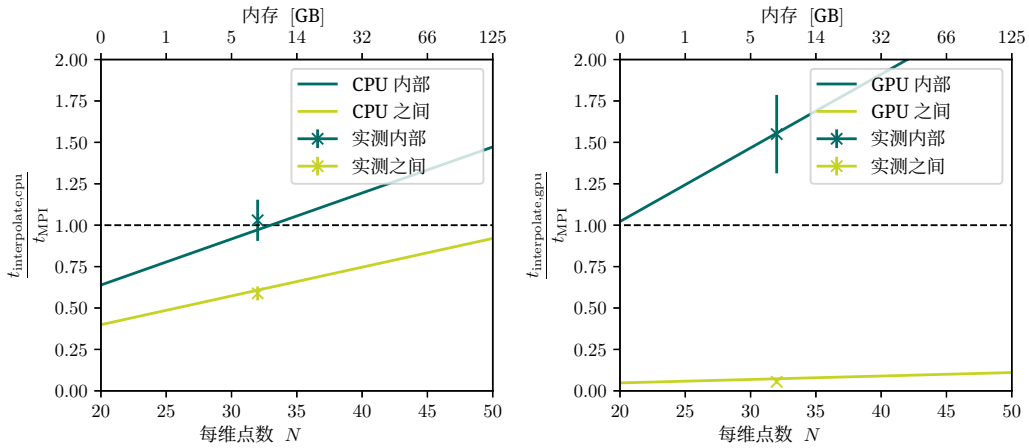

图 8：通信与内存受限的代码执行。比值低于 1 时，实现受网络通信限制；高于 1 时，则受计算限制。上方横坐标给出了六维分布函数所需的内存消耗。我们考虑 $N_{\mathrm{read}} = N_{\mathrm{write}} = 1$和 $N_{\mathrm{halo}} = 2$，其模板宽度为七，需要的边界宽度为$w = 3$。可用带宽见表 1。该图根据公式（8）和（9），展示了节点内和节点间 MPI 通信的计算时间与通信时间之比。

.

7 通过离子伯恩斯坦波进行物理验证

在本节中，我们通过重现伯恩斯坦发现的离子伯恩斯坦波色散关系 $\omega(k)$来验证该实现 [38]。作为试探解，采用一个平面波

$$\phi(\mathbf{x},t) = \hat{\phi}_{\vec{k}} \exp(i(\mathbf{k}\cdot\mathbf{x} - \omega t)) \tag{10}$$

用于求解以下 Vlasov 模型在 (ω,**k**) 中的解。

该模拟采用与 Sturdevant 等人的动力学模拟相同的设置 [39]。本测试使用简化的场方程。不同于在背景中模拟电子和静态离子，这里模拟离子（$q = 1$），并采用绝热电子，从而简化静电势的场方程（2）为

$$\frac{n_0 q^2}{T_e}\phi(\mathbf{x},t) = \rho(\mathbf{x},t), \tag{11}$$

其中， T_e 表示绝热电子的温度。分布函数采用乘积方法初始化：位形空间使用白噪声，速度空间使用麦克斯韦分布

$$f(\mathbf{x},\mathbf{v},t=0) = f_{x,0}(\mathbf{x}) f_{v,0}(\mathbf{v}) = n_0(1+\mathrm{rand}(\mathbf{x}))\left(\frac{m}{2\pi T_i}\right)^{3/2} \exp\left(-\frac{m\mathbf{v}^2}{2T_i}\right) \tag{12}$$

其中， m 对应离子质量， T对应系统温度。背景磁场选为 $\mathbf{B}_0 = B_0\hat{\mathbf{e}}_z$。通过将所有物理常数设为 **1**（$q = m = T_i = T_e = B_0 = n_0 = c = 1$），简化计算。函数 rand(**x**)生成振幅较小的白噪声 $\epsilon \ll 1$。

离子伯恩斯坦波（IBW）具有实频率，不受增长或阻尼影响。只需运行模拟，直到系统中的所有其他波都被阻尼掉，仅留下离子伯恩斯坦波，即可提取它们。我们只需研究波矢 k_x 和 $k_y = k_z = 0$，从而降低问题的维数。尽管如此，为满足最小模板宽度的要求，仍需在y 和 z 维度上设置几个网格点。因此，系统采用 $128 \times 8^2 \times 32 \times 16^2$ 网格进行设置，位形空间

盒子长度为 $L_{\mathbf{x}} = \frac{20}{3}\pi \times 2\pi \times 2\pi$，速度空间中所有方向的最大速度为 $v_{\max} = 4v_{\text{th}}$。时间步选为 $\Delta t = 0.025$。

模拟运行足够长时间后，可以通过对密度扰动进行时空傅里叶变换，提取离子伯恩斯坦波的色散关系。图9 展示了对从 $t_1 = 4000$ 到 $t_2 = 5000$的时间区间进行变换后得到的色散关系。图中清晰地显示，色散关系 $\omega(k_x)$的分支出现在拉莫尔频率的每个谐波 ω_{ci}附近。离子伯恩斯坦波的频率相较于相应的拉莫尔频率谐波$\omega_{ci,\ m} = m\omega_{ci} = m$ 略有上移，并且在波数较大时趋近于这些谐波，

$$\lim_{k_x \to \infty} \omega(k_x) = \omega_{ci,m} = m. \tag{13}$$

采用不同的插值模板运行了两次该设置，以展示数值格式不同的扩散行为。使用高阶模板对于分辨多个模态至关重要。将插值阶数从 3 提高到 7 后，可观察到的模态数量几乎增加了三倍。这是因为局部插值格式具有很强的扩散性。由于高阶模板会加剧前文所述的通信瓶颈，因此需要在高阶模板与模态分辨率之间作出折中。

该系统的色散关系可按照 Hazeltine 和 Waelbroeck 的著作中的计算方法推导得出 [40]。在$\vec{\ } \ \hat{}\ k_z = k \cdot \vec{z} = 0$ 的极限下，色散关系为

$$-\omega \sum_{m \in \mathbb{Z}} \frac{e^{-k_x^2} I_m(k_x^2)}{\omega + m} - 2 = 0 \tag{14}$$

其中 $I_m(x)$ 表示修正贝塞尔函数。对于 $\omega(k_x)$，色散关系的每一项 m在$\omega_{ci,m}$附近都有一个解。图9中绘出了解析色散关系（虚线）以及数值结果。可以看出，两种结果吻合良好。

重现离子伯恩斯坦波的色散关系，说明该模型在等离子体物理学中接近拉莫尔频率的高频区具有可信度。这些结果验证了该模型对频率接近拉莫尔频率的现象的适用性。

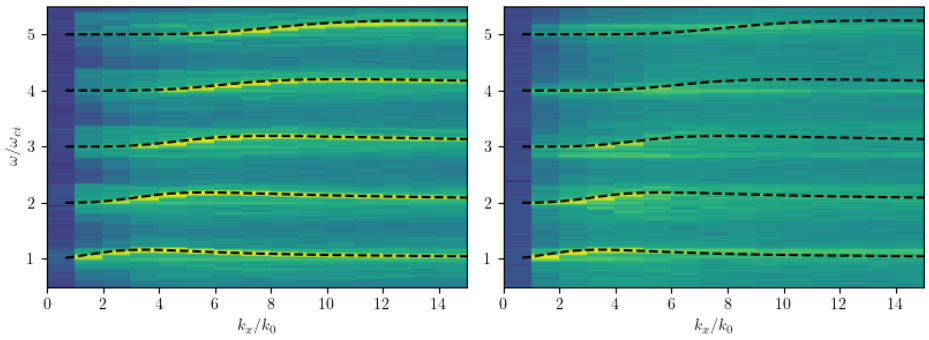

图9：弛豫后的静电势频率-波数谱，展示了插值模板宽度为 $d = 8$ （左）和 $d = 4$ （右）时的离子伯恩斯坦波色散关系（以颜色编码），以及 IBWs 的解析色散关系（虚线）。

8 结论

本文介绍了用于全动理学 Vlasov方程的高性能可移植半拉格朗日代码实现。首先，讨论了六维问题的内存需求挑战，以及针对我们算法配置所做的相应内存优化。我们采用“模板方法”设计模式实现了拉格朗日插值核，从而形成模块化的类结构。我们识别出的独立组件包括迭代、插值偏移和插值模板。在主内核中，这三个组件通过通用接口的定义进行交互。这种模块化设计便于对算法中的独立组件进行测试和扩展。

得益于性能可移植性框架 `Kokkos`，我们的软件架构在多种共享内存概念和硬件架构上均能实现良好的性能。所有经过分析的架构都可使用，同时仍能保持单一代码库。该实现充分利用了 GPU加速节点上的并行化潜力，使主内核的运行时间相比 CPU 节点可缩短至三分之一到八分之一。在节点级性能分析中，我们仍发现 GPU架构上存在进一步优化的潜力。减少整数计算、暂存内存用量和寄存器使用量，有望

提高 GPU 上的硬件利用率。

本文给出了扩展结果，并量化了采用分布式内存概念的六维实现中的通信瓶颈；这仍然是一项挑战。针对所研究的两种节点级架构开展的扩展实验表明，在大型计算集群上，不同消息传递接口进程所承担的任务彼此间独立性不足，因而无法实现良好的扩展。通过对网络带宽的分析，我们证明这一问题无法通过代码优化解决，因为它受硬件限制。因此，需要研究不同的算法，进一步解耦消息传递接口进程并获得更好的扩展结果。最后，通过复现离子伯恩斯坦波验证了该实现。未来工作还将致力于扩展物理模型，采用更精确的场方程和几何描述，并纳入碰撞效应。

9 致谢

计算工作使用了马克斯·普朗克计算与数据中心的高性能计算系统 Raven 和 COBRA，以及通过 LoGy项目使用的 Cineca 的 Marconi100。此外，我们感谢 Klaus Reuter 就 Raven 的网络性能进行了富有成效的讨论。

本工作部分在 EUROfusion 联盟框架内开展，由欧盟通过欧拉原子能研究与培训计划（资助协议编号 101052200 – EUROfusion）资助。此外，本工作还获得了 EUROfusion 高性能计算机（Marconi-Fusion）的支持。但本文所表达的观点和意见仅代表作者本人，并不一定反映欧盟或欧盟委员会的立场。欧盟和欧盟委员会均不对这些观点和意见承担责任。

参考文献

[1] M. Raeth、K. Hallatschek、K. Kormann，使用六维全动理学半拉格朗日代码模拟板状离子温度梯度，《等离子体物理》 [正在准备中的手稿]（2023年）。

[2] C.R. Trott、D. Lebrun-Grandie、D. Arndt、J. Ciesko、V. Dang、N. Ellingwood、R. Gayatri、E. Harvey、D. Hollman、D. Ibanez、N. Liber、J. Mad-

sen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, J. Wilke, Kokkos 3：百亿亿次时代的编程模型扩展，《IEEE 并行与分布式系统汇刊》33（2022）805–817。[3]K. Kormann, K. Reuter, M. Rampp，六维 Vlasov–Poisson 方程的大规模并行半拉格朗日求解器，《国际高性能计算应用期刊》（2019）。[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides，《设计模式》，Addison Wesley 专业计算丛书，第 35 版，Addison-Wesley，波士顿，2007。[5] K. Germaschewski, B. Allen, T. Dannert, M. Hrywniak, J. Donaghy, G. Merlo, S. Ethier, E. D'Azevedo, F. Jenko, A. Bhattacharjee，迈向聚变装置的百亿亿次级全装置建模：将 GENE 陀螺动理学微湍流代码移植到图形处理器，《等离子体物理》28（2021）。[6] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, M. Zingale，AMReX：块结构自适应网格细化框架，《开放源代码软件期刊》4（2019）1370。[7] N. Ohana, C. Gheller, E. Lanti, A. Jocksch, S. Brunner, L. Villard，基于全局电磁粒子网格代码 ORB5，在众核与多核架构上进行陀螺动理学模拟，《计算物理通讯》262（2021）107208。[8] ECP-CoPa，Copa Cabana——百亿亿次级粒子应用工具包协同设计中心，2023年。网址：https://github.com/ECP-copa/Cabana。[9] S. Muralikrishnan, M. Frey, A. Vinciguerra, M. Ligotino, A. J. Cerfon, M. K. Stoyanov, R. Gayatri, A. Adelmann，面向百亿亿次架构的迷你应用：粒子网格方案在等离子体物理应用中的扩展性与性能可移植性，2022。doi：https://doi.org/10.48550/arXiv.2205.11052。[10] D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, B. S. Ryujin, T. R. Scogland，RAJA：

面向大规模科学应用的可移植性能，载于：2019年IEEE/ACM高性能计算性能、可移植性与生产力国际研讨会（P3HPC），IEEE，2019年，第71–81页。[11] V. Artigues、K. Kormann、M. Rampp、K. Reuter，《用于实现粒子网格代码的性能可移植性框架评估》，《连接与计算：实践与经验》32（2020年）1–23。[12] A. Matthes、R. Widera、E. Zenker、B. Worpitz、A. Huebl、M. Bussmann，《使用alpaka库，无需更改一行实现代码即可针对多种众核架构进行调优和优化》，载于：《高性能计算》，施普林格国际出版公司，2017年。[13] J. Bigot、V. Grandgirard、G. Latu、C. Passeron、F. Rozar、O. Thomine，《gysela代码在Blue Gene上扩展至32k核以上》，ESAIM：会议录43（2013年）117–135。[14] T. Umeda、K. Fukazawa，《京计算机上空间等离子体Vlasov代码的性能调优》，载于：AsiaSim 2014，施普林格·柏林·海德堡，柏林、海德堡，2014年，第127–138页。[15] Y. Asahi、G. Latu、V. Grandgirard、J. Bigot，《动力学等离子体模拟迷你应用的性能可移植实现》，载于：WACCPD 2019：使用指令进行加速器编程，施普林格·查姆，美国科罗拉多州丹佛，2019年，第117–139页。[16] L. Einkemmer，《GPU上的半拉格朗日Vlasov模拟》，《计算物理通讯》254（2020年）107351。[17] L. Einkemmer、A. Moriggl，《配备GPU的大规模超级计算机上的半拉格朗日4D、5D和6D动力学等离子体模拟》，《国际高性能计算应用期刊》（2022年）。[18] E. Lifshitz、L. Pitaevskij（编），《第10卷：理论物理学教程》，第1版，培格曼出版社，牛津，1981年。[19] C. Cheng、G. Knorr，《Vlasov方程在构型空间中的积分》，《计算物理学杂志》22（1976年）330–351。[20] R. I. McLachlan、G. R. W. Quispel，《分裂方法》，《数值学报》11（2002年）341–434。

[21] Crouseilles、N.、Mehrenberger、M.、Vecil、F.，《用于Vlasov-Poisson方程的不连续伽辽金半拉格朗日方法》，ESAIM: Proc. 32（2011年）211–230。[22] M. Snir、S. Otto、S. Huss-Lederman、D. Walker，《消息传递接口》，第2版，麻省理工学院出版社，马萨诸塞州剑桥。[等]，1998。[23]HDF Group，《分层数据格式》第5版，1997-NNNN。Https://www.hdfgroup.org/HDF5/。[24] A. Ayala、S. Tomov、A. Haidar、J. Dongarra，《heffte：面向百亿亿次级计算的高效快速傅里叶变换》，载于：计算科学——ICCS 2020，施普林格国际出版公司，查姆，2020年，第262–275页。[25] C. R. Trott、L. Berger-Vergiat、D. Poliakoff、Sivasankaran、D. Lebrun-Grandie、J. Madsen、N. A. Awar、M. Gligoric、G. Shipman、G. Womeldorff，《Kokkos生态系统：面向高性能计算的全面性能可移植性》，Comp. Sc. Eng. 23（2021年）10–18。[26]K. Team，《Kokkos 文档》，2023年。网址：https://kokkos.github.io/kokkos-core-wiki/index.html。[27] H. C. Edwards、C. R. Trott、D. Sunderland，《Kokkos：通过多态内存访问模式实现众核性能可移植性》，J. Par. and Dist. Comp. 74（2014年）3202–3216。[28] M. Harris，《CUDA C/C中的高效矩阵转置++，2023年。网址：https://developer.nvidia.com/blog/efficient-matrix-transpose-cuda-cc/。[29]W. Language，《系统数组》，2023年。网址：https://reference.wolfram.com/language/ref/Array.html。[30]NumPy，numpy.array，2023年。网址：https://numpy.org/doc/stable/reference/generated/numpy.array.html。[31] G. Hager、G. Wellein，《科学家与工程师的高性能计算导论》，Chapman & Hall，CRC 计算科学丛书，CRC 出版社，博卡拉顿，2011年。

[32] K. Datta、S. Kamil、S. Williams、L. Oliker、J. Shalf、K. Yelick，《现代微处理器上模板计算的优化与性能建模》，SIAM Review 51（2009）129–159。[33] J. Treibig、G. Hager、G. Wellein，《Likwid：面向 x86 多核环境的轻量级性能分析工具套件》，载于：PSTI2010，即第一届并行软件工具与工具基础设施国际研讨会论文集，IEEE，加利福尼亚州圣迭戈，2010年。[34] NVIDIA，《Nsight Compute 文档》，2023年。网址：https://docs.nvidia.com/nsight-compute/。[35]AMD，《ROCm 性能分析工具用户指南》，2023年。网址：https://docs.amd.com/bundle/ROCm-Profiling-Tools-User-Guide-v5.3/page/Introduction_to_ROCm_Profiling_Tools_User_Guide.html。[36] T. Deakin、J. Price、M. Martineau、S. McIntosh-Smith，《通过 babel-stream 评估并行编程模型可达到的内存带宽》，《国际计算机科学与工程杂志》17（2018年）247–262。[37] S. Williams、A. Waterman、D. Patterson，《Roofline：一种直观的多核架构可视化性能模型》，《ACM通讯》52（2009）65–76。[38] I. B. Bernstein，《磁场中等离子体的波》，《物理评论》109（1958）10–21。[39] B. J. Sturdevant、Y. Chen、S. E. Parker，《环形离子温度梯度不稳定性的低频全动理学模拟》，《等离子体物理》24（2017）081207。[40] R. D. Hazeltine、F. L. Waelbroeck，《等离子体物理学框架》，CRC 出版社，2018年。