arXiv:2301.01145v1 [cs.MS] 1 Jan 2023

SAILFFISH：轻量级并行快速泊松求解器库

Joseph Saverin Her
mann Föttinger研究所 柏林工
业大学 柏林，德国
j.saverin@tu-berlin.de

摘要

本文介绍了一款用于一维、二维和三维规则网格泊松方程的求解器。该求解器利用卷积定理，在矩形计算域的谱空间中高效求解泊松方程。借助离散傅里叶变换，可实现频谱空间之间的转换，从而能够应用高度优化的 $\mathcal{O}(N \log\ N)$算法。其数据结构采用模块化设计，因此可替换用于谱空间之间转换及谱空间内部运算的底层接口。针对计算密集型任务，该库通过利用并行处理架构进行了优化。区域可采用多种边界条件，包括周期、狄利克雷、诺伊曼和完全无界边界条件。对于诺伊曼和狄利克雷边界条件，可以指定任意非齐次边界条件。所需解既可在正则（单元边界）网格配置上求得，也可在交错（单元中心）网格配置上求得。对于具有周期、狄利克雷或诺伊曼边界条件的问题，可以采用伪谱或二阶有限差分算子。对于无界边界条件，可使用多种格林函数。此外，还可以在谱空间中应用多种微分算子，以处理不同形式的泊松方程，或提取输入场的高精度梯度。本文首先详细介绍求解器的底层框架，随后针对每种可用的边界条件进行多项验证。最后，对该库的性能进行了考察。本代码遵循 GNU v3.0 许可证免费公开，网址为 github.com/ZeppSav/SailFFish。

***K*eywords** 泊松求解器 ·快速傅里叶变换 ·谱求解器、数值积分

1 引言

对于任意给定的向量场 $\vec{v}$，若其满足 C^2 光滑性——即二阶导数（或更高阶导数）连续——则该场可通过亥姆霍兹分解表示为：

$$\vec{v} = \nabla \times \vec{\psi} - \nabla\phi\,, \qquad \text{where} \qquad \nabla \cdot \vec{\psi} = 0\,. \tag{1}$$

这里 ψ 是一个矢量势，表示该场的无散（散度为零）部分，而 ϕ 是一个标量势，表示该场的无旋部分。矢量势的无散约束是确保分解唯一所必需的。对（1）取旋度，可得泊松方程：一个线性二阶偏微分方程（PDE）：

$$\nabla^2\vec{\psi} = \vec{f}\,, \tag{2}$$

其中 $f = -\nabla \times \vec{v}$是该向量场的旋度。对（1）取散度，可再次得到一个泊松方程，将标量势的拉普拉斯算子与向量场的散度联系起来 $\nabla \cdot \vec{v}$。这种与连续向量场的直接联系，恰好说明了泊松方程为何在物理建模中如此普遍。物理场量守恒会产生向量势和标量势，而它们可通过泊松方程或其简化形式求解。其应用包括流体动力学、静电学、电磁学、

引力、扩散和热传递等诸多领域。根据所需的求解精度以及区域的边界条件（BCs），存在多种求解泊松方程的方法。这些方法大致可分为三类：多重网格方法、快速多极子方法和谱方法 [1]。本文采用了谱方法，不过也将简要介绍其他求解方法，以作比较。

1.1《多重网格方法》

多重网格（MG）方法可用于在各种区域拓扑上的任意网格配置中，求解多种非线性和线性偏微分方程 [2, 3, 4, 5]。MG 方法的核心概念是在多个网格分辨率上求解所需的偏微分方程 $\mathcal{H}_{i=0,\ 1,\ \cdots n}$。首先以两个网格层级为例进行说明 $\mathcal{H}_0, \mathcal{H}_1$。例如，使用有限差分，在较细网格的一个不相交部分上求得初始解。通过比较源项与初步解，计算残差误差。该残差通过伴随插值从基网格$\mathcal{H}_0$ 插值到 $\mathcal{H}_1$。随后在较粗网格上求解，再插值回较细网格，以改进初始猜测。由此形成一个迭代过程，逐步平滑解，直至满足收敛准则。通常还会将这一过程扩展到更高的网格层级，以进一步提高效率。根据问题类型 [6]，有许多方案可用于在网格层级间进行最优遍历，例如 V和 W-循环。可以证明，对于包含 N 个网格点的网格，计算开销的增长规模为 $\mathcal{O}(N)$ [7]。此类方法的另一项优势是适用于不规则区域或具有可变分辨率的区域。

1.2 快速多极子法

快速多极子法（FMM）最初由 Greengard 和 Rokhlin 开发 [8, 9]。该方法利用泊松方程的格林函数解：

$$\psi(r) = \mathcal{G} * f = \int_{\mathcal{R}^n} f(r')\,\mathcal{G}(r - r')\,d^n r\,, \tag{3}$$

其中，$(*)$ 表示卷积算子。格林函数 $\mathcal{G}$ 是无界泊松方程的基本解：$\nabla^2\mathcal{G}(r) = \delta(r)$，其中 $\delta(r)$ 是狄拉克δ函数 [10]。快速多极子法利用 $\mathcal{G}$ 的性质，对远场影响进行低秩近似。$\mathcal{G}(r)$ 可表示为多极展开；在远场中，为达到所需精度，只需使用阶数较低的展开 ϵ。层次网格能够以更大的相互作用距离进行更粗略的近似。构造所需核函数的多极展开颇具难度，因此促成了核无关快速多极子方法（KIFMM）[11] 以及多层多相互作用簇（MLMIC）等类似方法的发展 [12, 13]。快速多极子法的优点之一是能够自然处理无界边界条件，不过也可以通过镜像法处理周期、齐次诺伊曼或狄利克雷边界条件 [14, 15]。快速多极子法的一项重要优势是能够处理稀疏甚至互不相连的源分布。根据问题和设置的不同，快速多极子法的复杂度通常为 $\mathcal{O}(N \log\ N)$ 或 $\mathcal{O}(N)$ [8]。

1.3 谱方法

第三类方法利用泊松方程的线性性质，将解表示为一系列基函数的和。在不失一般性的前提下，假设本文所讨论的情形中，这些基函数为三角函数。函数的正向和逆傅里叶变换定义如下：

$$\text{Forward:}\quad \hat{f}(k) = \int_{-\infty}^{\infty} f(x)e^{-2\pi ikx}\,dx, \qquad \text{Inverse:}\quad f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(k)e^{2\pi ikx}\,dk\,, \tag{4}$$

这将函数的表示从实数（x）空间转换到谱（k）空间。泊松方程可以在谱空间中写为：

$$-k^2\hat{\psi}(k) = \hat{f}\,, \qquad \text{where} \qquad k^2 = \begin{cases} k_x^2 & \text{in 1D} \\ k_x^2 + k_y^2 & \text{in 2D} \\ k_x^2 + k_y^2 + k_z^2 & \text{in 3D} \end{cases}, \tag{5}$$

和 k_i 是谱空间中的角波数。根据卷积定理，实空间中的卷积可通过谱空间中的逐点乘法来表示。这正是该方法的巨大优势所在。只需计算一次相应的格林函数（3），将其转换为谱表示并存储即可。此后，对任意给定源场进行的整个卷积运算都等价于在谱空间中进行乘法运算。在实际应用中，式（4）的连续变换采用离散傅里叶变换（DFT）处理，它在离散的截断谱空间中表示傅里叶变换。应用 DFT 后，便可利用快速

傅里叶变换（FFT），以实现 $\mathcal{O}(N \log N)$ [16]的计算开销。尽管与多重网格或快速多极子法相比，这似乎并非显著改进，但 FFT 在信号处理中的大量应用意味着，已有高度优化的 FFT 计算库。与多重网格或快速多极子法通过预设所需解的精度 ϵ 来达到目标不同，谱方法在适当条件下可以实现谱精度。这意味着所得解在实际应用中几乎精确，误差达到机器精度后便趋于饱和。该方法可表示多种边界条件，包括诺伊曼、狄利克雷、周期和无界边界条件。下一节将对此进行详细介绍。该方法的缺点是，必须针对给定分辨率生成并存储整个网格，因此会产生较大的内存开销。这使得该方法不太适用于稀疏源分布。此外，将该方法扩展到非矩形区域或非规则网格也颇具挑战 [17]。

本文采用的谱方法已在 SailFFish 中实现。SailFFish 是一个以C++编写的开源库。该库的设计注重简洁性、模块化和适应性。代码可在网上免费获取，并依据 GNU v3.0 许可证发布，网址为 github.com/ZeppSav/SailFFish。本文其余部分将概述并验证 SailFFish 求解器。第2节详细介绍求解器中实现的数值方法。第3 节介绍求解器的框架，包括面向对象编程、数据类型模块化以及输入/输出接口的应用细节。第4 节通过一系列验证案例考察求解器的准确性。第5节详细介绍库的性能及其扩展行为。最后，第6节总结全文，并阐述未来功能与应用的规划。

2 求解方法

SailFFish 中采用谱方法求解泊松方程。本节将介绍其方法和求解步骤，以及不同求解器类型的选项。为便于说明，首先介绍一个简单的一维周期情形。如第 1.3 节所述，假设该函数表示为三角级数：

$$f(x) = \sum_{k=0}^{N} \hat{f}_k e^{2\pi i \frac{kx}{L_x}} = \sum_{k=0}^{N} \hat{f}_k \left(\cos 2\pi k \frac{x}{L} + i \sin 2\pi k \frac{x}{L} \right) , \tag{6}$$

其中 L 为区域的长度。由此表示可推知解在整个区域内具有周期性。三角函数表示解特征向量。将此表示代入（2），可得到网格上解的如下表示：

$$-\left(\frac{2\pi k}{L}\right)^2 \hat{\psi}_k = \hat{f}_k \, . \tag{7}$$

ˆ乘以 ψ_k 的因子表示与解特征向量对应的谱特征值 [18]。就解的精度而言，在网格上经过（反向）变换得到的值 ψ_n 将以机器精度精确对应泊松方程的解。这称为达到谱精度。可以看到，情形 $k = 0$ 对应于除以零。傅里叶模态 $k = 0$ 表示函数 ψ 的平均值；令其为零即可使平均值为零，并避免数值不稳定性。

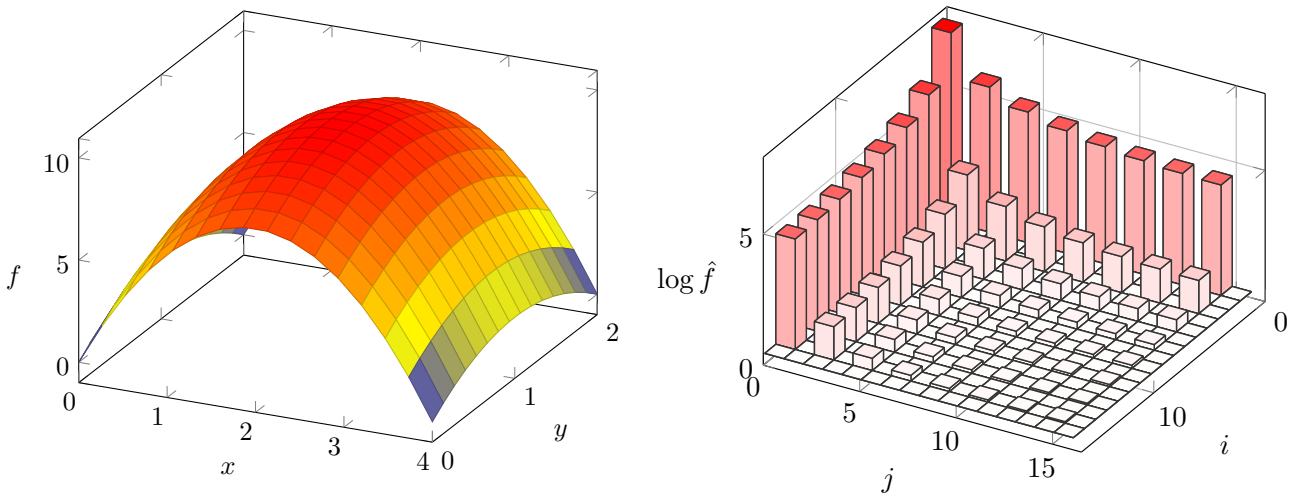
求解泊松方程的另一种方法是直接将表达式（6）与有限差分（FD）结合使用 [19]。与上述采用谱特征值所达到的精度不同，这种方法的收敛精度取决于所采用的有限差分格式。

第三种方法是在实空间中计算格林函数（3），然后将其变换，以获得谱空间表示。这种方法的优点是可以使用平滑化格林核，从而平滑输入场 [20, 21, 22]。SailFFish 已将这种方法应用于无界求解器情形，详见第2.5 节。

2.1 使用离散傅里叶变换在实空间与谱空间之间进行变换

与式（4）定义的连续正向和逆傅里叶变换不同，实际应用中，网格值是在离散、等间距的网格上指定的。通过正向和逆离散傅里叶变换，可实现与离散谱空间之间的转换，其定义如下：

$$\text{Forward:} \quad \hat{f}_d(k) = \sum_{n=0}^{N-1} f_d(x) e^{-2\pi i k \frac{n}{N}}, \qquad \text{Inverse:} \quad f(x) = \frac{1}{N} \sum_{n=0}^{N-1} \hat{f}_d(k) e^{2\pi i k \frac{n}{N}} , \tag{8}$$



ˆ 图 1：源项 f 的二维抛物面分布（左），以及其在谱空间中的表示 f ˆ（右）。f 的对数仅用于视觉展示。

根据所采用的边界条件以及输入数据的类型（纯实数或复数），可以选择不同形式的离散傅里叶变换来提升性能。例如，当输入数据为由 n 纯实数值 $f = [f_1, f_2, \ldots, f_n]$构成的数组时，可以证明离散傅里叶变换的输出具有厄米冗余。此时可采用实数到复数（R2C）离散傅里叶变换，将输出向量表示为含有 $\frac{n}{2}+1$ 个元素的复数数组。这种简化可使速度提升约一倍，并大幅降低内存开销。逆变换也可用类似方式实现。

根据所求解问题的维度，执行一维、二维或三维离散傅里叶变换。一维离散傅里叶变换应用于单个连续数据数组。更高维的离散傅里叶变换则通过沿一维笔形分解依次执行离散傅里叶变换来完成。因此，对一组数据点 $f(x_i, y_j)$ 执行的二维离散傅里叶变换计算如下：

ˆ 1. **X 正向变换**：沿 x 方向执行正向离散傅里叶变换，将 $f(x_i, y_j)$ 转换为 $f(k_x, y_j)$，ˆ
ˆ 2. **Y 正向变换**：沿 y 方向执行正向离散傅里叶变换，将 $f(k_x, y_j)$ 转换为 $f(k_x, k_y)$。

逆变换过程遵循完全相同的模板，但改为应用逆离散傅里叶变换，并按相反的顺序进行。对于前述的二维示例，逆变换的执行步骤如下：

ˆ ˆ 1. **Y 逆变换**：沿 y 方向执行反向离散傅里叶变换，将 $f(k_x,\ k_y)$ 转换为 $f(k_x,\ y_j)$，

2. **X** ˆ**逆变换**：沿 x 方向执行反向离散傅里叶变换，将 $f(k_x,\ y_j)$ 转换为 $f(x_i,\ y_j\ )$.

在二维情况下，有两种处理这些过程的方法。第一种方法是将步骤 (1) 后部分变换的数据重新排列（或存储在一个中间数组中），使其在内存中连续排列，以便执行第二次离散傅里叶变换。这样做的优点是可以更灵活地选择各空间方向上的边界条件 [20]。第二种方法是采用专用的二维离散傅里叶变换算法，根据指定的网格维度，自动对顺序正确的数据执行离散傅里叶变换。SailFFish 采用后一种方法，详情将在第 3 节介绍。扩展到三维情况也很直接。

2.2 计算并获取解

在 SailFFish 中，空间卷积在谱空间中进行，如第 1.3 节所述。这始终通过执行以下四个步骤来实现：

ˆ 1. **正向变换**：源分布 f 通过正向离散傅里叶变换转换为谱表示 f；

ˆ 2. **卷积**：将 f 与谱空间乘子（相应的特征值或格林函数的傅里叶变换 $\mathcal{G}$）逐点相乘；3. **谱运算**：如有必要，在谱空间内执行额外运算；ˆ ˆ 4. **逆变换**：将谱解 $\mathcal{G}f$ 通过逆离散傅里叶变换转换到实空间 f。

因此，整体流程相对直接。步骤 1) 的输出可视化结果如图 1 所示。尽管上述方法简单，但仍有多种求解器选项可供选择。 **DFT 类型**：这取决于待变换数据的形式（纯实数或复数）以及所施加的边界条件类型（R2C、R2R、DFT）。 **谱空间卷积核**：这些卷积核既可以通过上一节所述的谱方法或有限差分方法推导得到，也可以通过应用格林函数核的解（3）得到。 **谱空间算子**：可以在谱域中应用一系列算子，以修改所研究泊松方程的形式，或提取谱梯度。这些选项详见第 2.3.3 节。

2.3 SailFFish 中的求解器选项

SailFFish 中 求解器类型和求解器配置取决于所需的解场、边界条件以及网格属性。可用选项的范围 在本节中进行介绍。

2.3.1 求解器类型

目前，SailFFish 能够求解一维、二维或三维标量场，以及三维矢量场。对于三维矢量场，三个分量分别采用对应的三维标量求解器方法逐分量求解。考虑到这一点，现有四种求解器类：

- 泊松_周期_1维/二维/三维/三维矢量：采用周期边界条件。
- 泊松_狄利克雷_1维/二维/三维/三维矢量：采用狄利克雷（奇函数延拓）边界条件。
- 泊松_诺伊曼_1维/二维/三维/三维向量：应用诺伊曼（偶函数延拓）边界条件。
- 泊松_无界_1维/二维/三维/三维向量：该问题采用自由空间边界条件处理。

在所有情况下，三维向量 选项表示三维矢量场的情况，否则求解标量场。上文所述的前三个选项表示有界求解器类型——请参见第2.4节。最后一种求解器类型采用不同的求解方法，并在第2.5节中单独介绍。目前，SailFFish 中仅提供均匀边界条件。有关支持任意边界条件的实现细节，请参阅 FLUPS 库[20]。其中的原因将在第3节中详细说明。

2.3.2 网格类型

SailFFish 在一维、二维和三维情况下的规则网格上求解泊松方程（2）。网格的分辨率始终通过网格单元数量指定， N_c。在求解器初始化期间，必须指定每个网格维度的空间坐标上限和下限。此外，还可以指定解所在的网格位置。如果指定了 `REGULAR` 网格选项，则将在单元边界位置处计算解。已解析的网格位置数量 N_s 会根据所应用的求解器类型而变化——请参见第2.4节。如果指定了`STAGGERED` 网格选项，则在单元中心处求解。在这种情况下，解位于 N_c 位置。数值上，网格位置表示为：

$$X_i = \begin{cases} X_l + i(X_u - X_l)/N_c & \text{for } i = 0, 1, \dots N_s \quad \texttt{REGULAR} \text{ grid option} \\ X_l + (i + 0.5)(X_u - X_l)/N_c & \text{for } i = 0, 1, \dots N_c - 1 \quad \texttt{STAGGERED} \text{ grid option} \end{cases}, \tag{9}$$

其中， X_l 和 X_u 分别是网格区域的下限值和上限值。该方案应用于每个已解析的空间方向。

2.3.3 微分算子

ˆ在某些情况下，可能需要对变换后的源场进行谱微分 f。检查（4）可知：

$$\frac{d}{dx} f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{d}{dx} \left(\hat{f}(k) e^{2\pi i k x} \right) dk = \frac{1}{2\pi} \int_{-\infty}^{\infty} ik\, \hat{f}(k) e^{2\pi i k x}\, dk\,, \tag{10}$$

因此，在频域中乘以 ik，即可提取函数 f 的（谱精确）ˆ 导数。还可以结合卷积核 $\mathcal{G}(k)$ 使用，以改变所求解泊松方程的形式。SailFFish 提供多种提取空间梯度的选项：

- DIV：求解输入场的散度
- GRAD：求解输入场的梯度
- CURL：求解输入场的旋度
- NABLA：求解输入场的拉普拉斯算子

将谱微分应用于偏微分方程的逐点求解，通常称为伪谱方法 [23]。

2.4 有界求解器

如果已指定区域边界条件的类型或数值，则该区域上的解称为**有界**解。有界求解器可采用三种形式：**狄利克雷**（奇对称）、**诺伊曼**（偶对称）或**周期**。下文将介绍变换类型及其对应的特征值。

2.4.1 变换类型

狄利克雷型和诺伊曼型求解器允许使用实数到实数（R2R）离散傅里叶变换。这里采用（6）的纯实数形式，仅对正弦（离散正弦变换，DST）或余弦（离散余弦变换，DCT）展开进行处理，分别对应奇对称或偶对称边界条件。所选网格是正则网格还是交错网格，也决定了解特征向量，进而决定所采用的 DST 或 DCT 类型。表1汇总了各种边界条件和网格类型组合。此处还提供了进一步的参考资料，以详细介绍相应的特征向量函数 [18, 20, 24, 25]。由于离散傅里叶变换由周期三角函数构成，因此其本身具有周期性。对于周期边界条件，采用直接的复数到复数（C2C）标准离散傅里叶变换。虽然对于纯实数数据，可通过应用第2.1节所述的实数到复数变换来简化处理，但为适用于未来的一般应用场景，此处将其配置为处理复数输入。实数数据输入以虚部为零的复数数据输入形式存储。

表1：有界求解器类型的离散傅里叶变换类型和特征值

BC Type	Grid Type	DFT（正向）	DFT（反向）	$\mathcal{G}_i$ (PS)	$\mathcal{G}_i$ （二阶有限差分）
狄利克雷	正则	第一类离散正弦变换	第一类离散正弦变换	$-\left(\pi\frac{k+1}{L}\right)^2$	$-\frac{4}{H^2}\sin^2\frac{\pi(k+1)}{2(N+1)}$
狄利克雷	交错	第二类离散正弦变换	第三类离散正弦变换	$-\left(\pi\frac{k+1}{L}\right)^2$	$-\frac{4}{H^2}$正弦$^2\frac{\pi(k+1)}{2N}$
诺伊曼	正则	第一类离散余弦变换	第一类离散余弦变换	$-\left(\pi\frac{k}{L}\right)^2$	$-\frac{4}{H^2}$正弦$^2\ 2\ \frac{\pi k}{N-1}($)
诺伊曼	交错	第二类离散余弦变换	第三类离散正弦变换	$-\left(\pi\frac{k}{L}\right)^2$	$-\frac{4}{H^2}$正弦$^2\frac{\pi k}{2N}$
周期	Both	DFT	逆离散傅里叶变换	$-\left(2\pi\frac{k}{L}\right)^2$: k $<\frac{N}{2}$ $-\left(2\pi\frac{N-k}{L}\right)^2$: k $\geq\frac{N}{2}$	$-\frac{4}{H^2}$正弦$^2\frac{\pi k}{N}$

2.4.2 特征值的选择

对于有界问题，特征值有两种选择。第一种是伪谱法（PS）特征值。正如上文在周期情形中所示，它们直接由三角展开在实空间中求导所得到的特征值推导而来。由于特定边界条件与网格类型组合下特征向量相应的零点不同，这些特征值也会有所变化。第二种选择是使用二阶有限差分（FD2）特征值。采用这种选择的优点是，狄利克雷和诺伊曼求解器类型可以使用非齐次边界条件。这是因为有限差分已融入谱解中。为施加所需的边界值，可以对边界源项进行简单加权 [19]。SailFFish 会根据求解器和网格类型自动进行相应处理。图2展示了非齐次边界条件的示例。表1也列出了所有可用边界条件与网格类型组合对应的正确特征值。为确保谱特征值对应的目标解具有周期性，需要应用循环格林函数。周期求解器的伪谱特征值体现了这一点：它们关于点 $k = \frac{N}{2}$ 呈镜像对称。

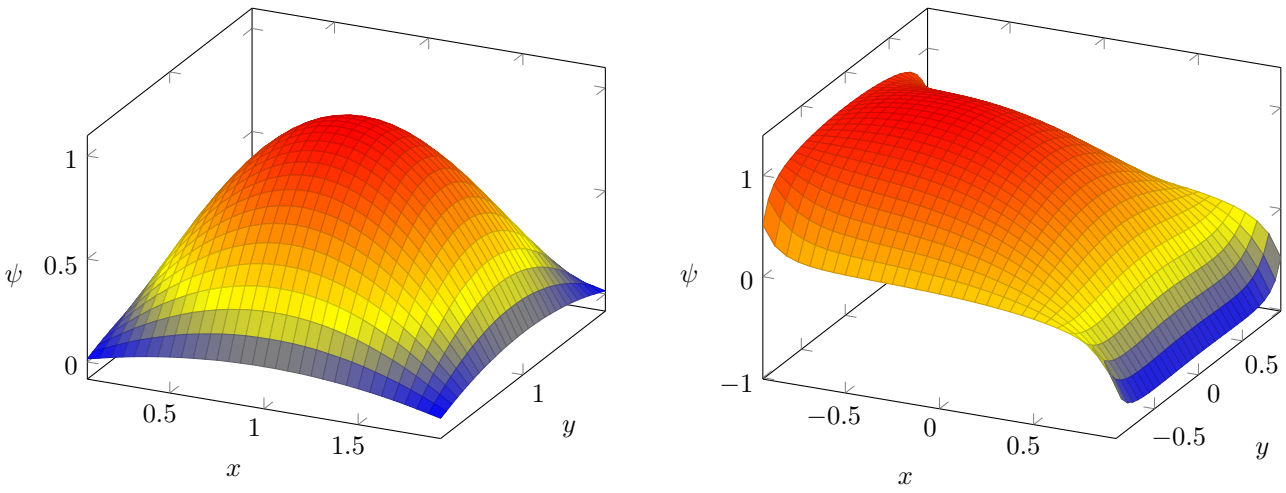


图 2：使用 FD2 型特征值求解具有齐次（左）和非齐次（右）边界条件的二维狄利克雷问题。两种情况下的源场 f 相同。

2.5 无界求解器

如果指定自由空间边界条件，则区域上的解称为**无界**解。处理这一问题有多种方法，包括 James & Lackner 提出的方法 [26, 27], 、Vico-Greengard 方法 [28],或 Hockney-Eastwood 方法 [29, 30]。SailFFish 采用了最后一种方法。为模拟自由边界条件的效果，需要将格林函数的区域沿相关维度加倍，并关于点 $N = k$作镜像处理，与周期谱特征值的做法相同。由于离散傅里叶变换具有周期性，频域中任意一点的卷积都会延伸至边界，其效果与无界解相同。为确保频域中的卷积不会发生反射，还必须将源场加倍，并对额外的求解器区域进行零填充，以避免产生任何人为影响。图3对此作了示意。得到加倍区域上的周期解后，提取目标区域内的解，并舍弃其余区域。关于这一概念在一维和混合边界条件下的简明解释，请参阅 FLUPS 文档 [20]。

2.5.1 变换类型

对于无界情形，采用直接离散傅里叶变换（DFT）。因此，源函数和格林函数可能为复数。虽然 SailFFish 尚未实现此功能，但它可用于求解亥姆霍兹方程 [31, 20]。对于纯实数数据，则采用 R2C 和复数到实数变换（C2R）来提高求解器效率。

2.5.2 格林函数

不同于有界求解器（其中格林函数可直接在频率空间中应用），对于无界解则必须更加谨慎。格林函数在原点处未定义的问题无法规避。此外，Hockney & Eastwood 的原始方法仅表现出 $\mathcal{O}(h^2)$ 收敛，其中 h是网格尺寸 [30]。为克服这一问题，人们提出了一系列方案。Hejlesen 等人提出了一系列高斯平滑，通过对逆高斯正则化 [21]应用 P^{th} 阶近似，逐步逼近奇异解 $\nabla^2\mathcal{G} = \delta(r)$。同一作者推导出一种谱紧支撑核，在有限网格 [22]上实现了谱精度。在 SailFFish 中，适用于一维、二维和三维情况的谱紧支撑 Hejlesen 核函数均可使用。对于二维和三维求解器，还可以选择阶数为 $P = 2,\ 4, \ldots 10$ 的 Hejlesen 高斯近似。

2.6 半无限域求解器

可以手动将二维或三维离散傅里叶变换操作拆分为一系列连续的一维离散傅里叶变换过程。许多其他泊松求解器软件包采用了这种方法 [20, 32]。这种方法的优点是可以沿所需方向拆分拉普拉斯算子，从而得到一个二维亥姆霍兹方程，以便处理混合边界条件 [31]。通过组合使用离散正弦变换、离散余弦变换和离散傅里叶变换，可以实现

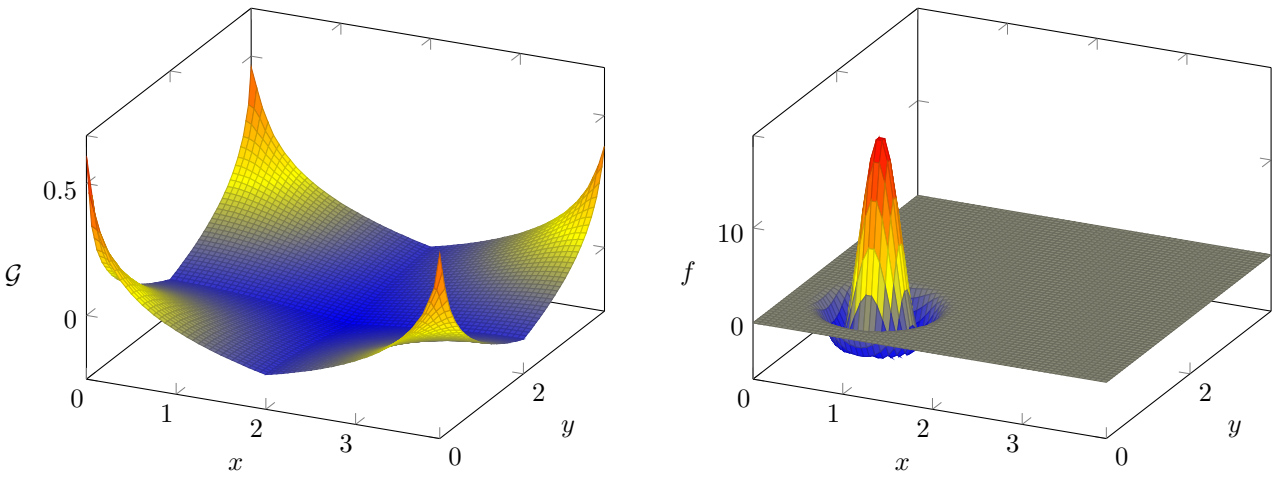


图 3：应用 Hockney-Eastwood 方法所需的操作。在实空间中对格林函数进行加倍和镜像处理（左图）。源区域也进行加倍，剩余空间进行零填充（右图）。此处模拟的是一个凸起函数——参见第 4.3 节。

完全任意的边界条件 [20]。在 SailFFish 中，经过优化的 N维离散傅里叶变换算法可用于缩短通信时间，并便于在图形处理器（GPU）上进行开发——详见第 3 节。因此，SailFFish 目前尚不支持此功能，但计划在未来的工作中实现。

3 求解器框架

SailFFish 的设计目标如下：

> 1. 简洁性、可读性和抽象性；2. 针对共享内存平台进行优化；3. 尽量减少通信时间，并最大限度地应用现有库和算法。

通过 `C++` 的面向对象结构实现抽象，以避免代码重复并提高可读性。类对象和继承将在下一节中介绍。以这种方式准备软件，可以在未来针对特定问题简化定制应用。SailFFish 的设计目标是在共享内存设备上运行，因为简单地说，并非每位用户都能使用集群计算设备，而总体理念应当是注重 **效率** 而非性能。对于所有计算密集型任务，均采用多平台共享内存并行编程框架 `OpenMP` 对计算进行并行化 [33]。

最后一点涉及利用现有的优化程序或应用程序接口，以便保持代码的可读性。以二维和三维快速傅里叶变换的执行为例。如果软件配置为处理混合边界条件，则必须分别对一维数组执行进出谱空间的变换。这要求重新组织数据，使其连续对齐，以便应用一维快速傅里叶变换算法。正如[20]中所述，这一必要性不仅使代码更加复杂（尤其是对于任意边界条件），还增加了内存通信开销。在 SailFFish 中，则通过直接利用所选快速傅里叶变换引擎中已有的二维和三维快速傅里叶变换算法来避免这一问题。这种方法的一个优点是，数据可以作为非常大的整体数据块传递。另一个优点是，可以沿单个索引对数据执行操作（例如卷积）。

这种方法对于求解无界问题尤为适用——参见第 2.5 节。Hockney-Eastwood 算法要求将每个维度上的区域加倍，因此计算开销和内存占用约增加 2^d 倍，其中 d 是问题的维数 [29]。通过依次沿各个方向执行变换，并丢弃零填充区域中的额外信息，可将这一倍数降至约 2^{d-1} [29, 21]。这必然会大幅缩小快速傅里叶变换的规模。然而，如上所述，这也要求为快速傅里叶变换算法重新排列数据。目前，为了避免这一点，我们只是

处理整个加倍后的区域，以遵循简化这一宗旨。作者希望在不久的将来，通过直接采用 James-Lackner 方法求解无界问题，免去将区域加倍的必要 [27, 26]。接下来的各节将介绍 SailFFish 库的架构。

3.1 SailFFish 的架构

SailFFish 中有两个父类包含了大部分功能： 数据类型类和 求解器类。下面将对此进行介绍。图 4展示了类数据结构的可视化图示。

3.1.1 DataType 类

DataType 父类包含一系列虚方法，用于访问（输入/输出）、处理（卷积）以及对谱空间中的数据执行变换。对于选定的 FFT 工具，将从 DataType 父类派生出相应的类，并且必须实现相应的虚方法。目前，SailFFish 中提供两种 DataType 基类：

- FFTW_数据类型使用了经过高度优化且广受欢迎的 FFTW 库（西方最快傅里叶变换） [24, 25, 34]。该库功能广泛，且始终表现出良好的性能；
- cuFFT_数据类型利用 CUDA 语言，在英伟达图形处理器（GPU）上实现高速并行执行 [35]。尽管该库的功能远不及 FFTW，但 GPU 的加速能力足以让它在桌面计算机上极快地执行。

虽然这里只介绍了两种选项，但根据目标设备、编译环境和预期的求解器类型，还有许多其他选项，可能需要实现更多数据类型类。可能的选项包括英特尔 MKL FFT 框架 [36], 、OpenCL FFT 库 [37], ，或使用 Python FFT 等其他语言实现 [38]。根据所选的数据类型，可能还需要进行额外的实现。

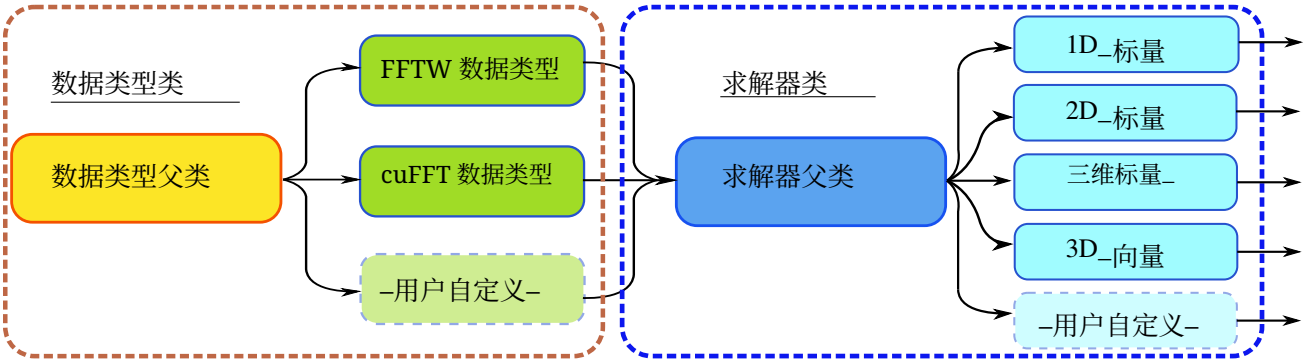


图 4：SailFFish 的架构

3.1.2 求解器类

该求解器父类直接派生自用户定义的数据类型父类，并包含快速傅里叶变换工具的全部功能和接口方法。求解器实现通过继承该类，实现了额外的抽象层。此外，该类还包含所有与网格声明和操作、求解器选项指定、格林函数准备以及卷积执行相关的虚函数。以下四种求解器派生自求解器父类：一维_标量、二维_标量、三维_标量和三维_向量，其名称取自源数据（及解数据）的类型。然后，从这些类派生出适用于特定边界条件的求解器（见第2.3.1节）。每个求解器类都必须声明并定义以下虚方法：

- 快速傅里叶变换_数据_设置-分配数据和快速傅里叶变换类型；
- 指定_算子- 指定所需的谱空间算子类型（第 2.3.3节）；

- 正向_变换- 从实空间到谱空间的离散傅里叶变换形式；
- 反向_变换——将离散傅里叶变换从频谱空间转换到实空间；
- 卷积——卷积形式（纯实数、复数）；
- 指定_格林_函数——计算卷积步骤中网格上的格林函数。

尽管 SailFFish 已为上述求解器类型声明并定义了这些内容，但用户也可以选择
定义求解器类，并根据所需应用添加任何必要的操作。

3.2 求解器的生成与执行

这里简要概述在 SailFFish 中生成和执行求解器所需执行的例程。

3.2.1 求解器初始化

求解器初始化过程中会调用许多函数，具体如下：

1. X/Y/Z_网格_设置 指定网格位置和统计信息。 2. 数据类型_设置 执行特定于数据类型的初始化。 3. 快速傅里叶变换_数据_设置 分配快速傅里叶变换数据数组并指定快速傅里叶变换计划。 4. 指定_格林_函数 指定特征值（有界边界条件）数组或格林函数（无界边界条件）数组。 5. 准备_微分_算子 准备谱空间微分算子（如适用）。

格林函数的指定是此处计算开销最大的过程，因为必须执行许多开销较大的函数求值。该过程通过 OpenMP 并行化。初始化的计算开销并无大碍，因为初始化只在模拟开始时执行一次。此后，求解器可以任意多次执行。

3.2.2 求解器执行

执　　行过程相对简单。步骤按时间顺序列出如下　　：

1. 设置_输入 网格上 f 的值将被传输到用于变换的必要数组中。2. 正向_变换 对数据执行正向快速傅里叶变换。3. 卷积 在谱空间中进行卷积。还包括谱空间中的其他操作，例如谱微分。4. 反向_变换 对数据执行反向（逆向）快速傅里叶变换。$\vec{}$5. 获取_输出 网格上 ψ 的值将被传输到用于后处理的必要数组中。

这里最耗时的步骤是正向和反向快速傅里叶变换。第5节概述了求解器的运行时间

4 验证

为展示求解器的收敛性，我们进行了一系列验证计算。根据求解器的类型，我们选择了合适的源函数，其对应的泊松方程解析解已知。需要注意的是，所达到的收敛性取决于输入场 f。数值解的精度取决于源函数具有的连续导数阶数。因此，在收敛性研究中，选择一个具有 C^∞ 光滑性的源函数会有所帮助。

对于所展示的所有情况，SailFFish 均配置为双精度浮点精度。对于下面展示的所有结果，我们通过展示误差的无穷范数来说明求解器的收敛性：

$$E_\infty = \sup\{\psi(x,y,z) - \psi_a(x,y,z)\}\,, \tag{11}$$

其中，ψ_a 表示已知的解析解。选择该误差范数，是因为 E_∞-范数有界于 E_2-范数。误差的绝对值按网格上解的最低值 $\psi_{a,min} = \inf\{\psi_a\}$ 进行缩放。为使误差结果具有可比性，所有 E_∞ 值均已缩放，使其对应于满足$\psi_{a,min} = 1$的网格解。

4.1 有界求解器：齐次边界条件

第一类 所研究的求解器中，有界求解器如下。使用了以下一维测试函数 ：

$$S(x, L_x) = \sin\left(4\pi \frac{x}{L_x}\right), \qquad C(x, L_x) = \cos\left(4\pi \frac{x}{L_x}\right), \tag{12}$$

其中， L_x 是区域在 x 方向上的长度，坐标 x 相对于x区域（x_l）的下界取值。在更高维的情形下，沿各方向取该函数的乘积。解析解的推导十分简单。例如，对于齐次狄利克雷边界条件，使用了以下函数：

$$f_a(x, y, z) = S(x, L_x)\, S(y, L_y)\, S(z, L_z) \quad \longrightarrow \quad \psi_a(x, y, z) = -(4\pi)^2 \left(\frac{L_x + L_y + L_z}{L_x^2 L_y^2 L_z^2}\right) f_a(x, y, z) \tag{13}$$

对于齐次诺伊曼边界条件，采用与（12）类似的表达式，但将其替换为余弦表达式 C（x， L_x）。与（13）对应的解析解由此很容易得出。伪谱法（PS）和二阶有限差分（FD2）特征值的误差如图 5所示。结果展示了正则网格配置和交错网格配置。正如预期，将 PS 特征值应用于谱源分布可使解达到谱精度，因为误差可见已饱和于机器精度。应用 FD2 型特征值则产生预期的二次收敛。作为参考，图中还展示了 Hejlesen 的谱一维算子，用于一维无界问题 [21]。可以观察到，无论网格配置如何，都重现了预期的行为。二维和三维情形的相应图表也见图 5。这些情形表现出的行为与一维情形几乎完全相同。可以看到，对于所选网格尺寸，三维无界情形未能完全达到谱精度，因为其空间分辨率低于一维和二维情形（这是内存需求增加所致）。此处未展示三维向量型求解器的结果，因为根据其构造，该求解器执行的步骤与三维标量求解器类型完全相同。

4.2 有界求解器：非齐次边界条件

SailFFish 还提供了在狄利克雷和诺伊曼有界求解器中处理非齐次边界条件的选项。在这种情况下，可以利用二阶有限差分特征值的形式，通过简单地修改内部边界节点的值来满足所需的边界条件，如第 2.4.2 节所述。不过，这会排除使用伪谱特征值类型。本文将研究这些解的精度。非齐次边界条件通过应用以下输入函数来指定：

$$\psi_a(x, L_x) = a\left(1 - \frac{x}{L_x}\right)^2 + b\left(\frac{x}{L_x}\right)^2 \quad \longrightarrow \quad f_a(x, L_x) = 2.0\frac{a+b}{L_x^2}. \tag{14}$$

该函数表示一个抛物面，其值分别为 $\psi_a(0,\ L_x) = a$ 和 $\psi_b(L_x,\ L_x) = b$，对应区域的下限和上限。在三维情况下，只需为每个维度添加一个函数。由此，每个区域面上都会形成抛物型边界分布，其角点值由常数 a_x， b_x， a_y， $\cdots$ 指定。此外，该函数的另一个优点是其梯度函数很容易计算：

$$\frac{\partial \psi_a}{\partial x}(x, L_x) = 2a\left(\frac{x}{L_x} - 1\right) + 2b\frac{x}{L_x}\,. \tag{15}$$

对于采用狄利克雷边界条件的测试用例，因此使用（14）计算边界上的数值。对于采用诺伊曼边界条件的测试用例，则使用（15）。由于该函数的高阶导数为零，即使采用 FD2 型特征值，也能达到谱精度。因此，测试函数叠加了来自（12）的相应分布。对于狄利克雷边界条件，使用 S（$\cdot$），对于诺伊曼边界条件，使用 C（$\cdot$），从而确保叠加后各个面的总边界条件保持不变。图6展示了二维和三维情况下不同网格尺寸对应的结果。为简洁起见，省略了一维情况，因为其结果与二维和三维情况无法区分。可以看到，误差按预期以二阶速率衰减，这与 FD2 特征值类型相符。这里的结果与图5中齐次边界条件的结果相呼应。需要注意的是，如果所有边界均采用诺伊曼型边界条件，解只能确定到一个加法常数。可以采用使残差最小的最优解。不过，为求简洁，SailFFish 并未实现这一功能。因此，图6 中的结果是通过平移解的数值，使得 $\sup\{\psi\} = \sup\{\psi_a\}$。

4.3 无界求解器

与有界求解器一样，最好使用边界值有限且 display C^∞ 光滑的测试函数。对于无界问题，这里采用了凸起函数。其定义如下：

$$\psi_a(\zeta) = \begin{cases} e^{-cd} & \zeta < 1 \\ 0 & \zeta \geq 1 \end{cases} \rightarrow \; f_a(\zeta) = \begin{cases} -\left(c^2\zeta^2 d^4 - 2cd^2 - 8c\zeta^2 d^3\right) e^{-cd} & \zeta < 1 \\ 0 & \zeta \geq 1 \end{cases} \text{ where: } d = \frac{1}{1-\zeta^2} \tag{16}$$

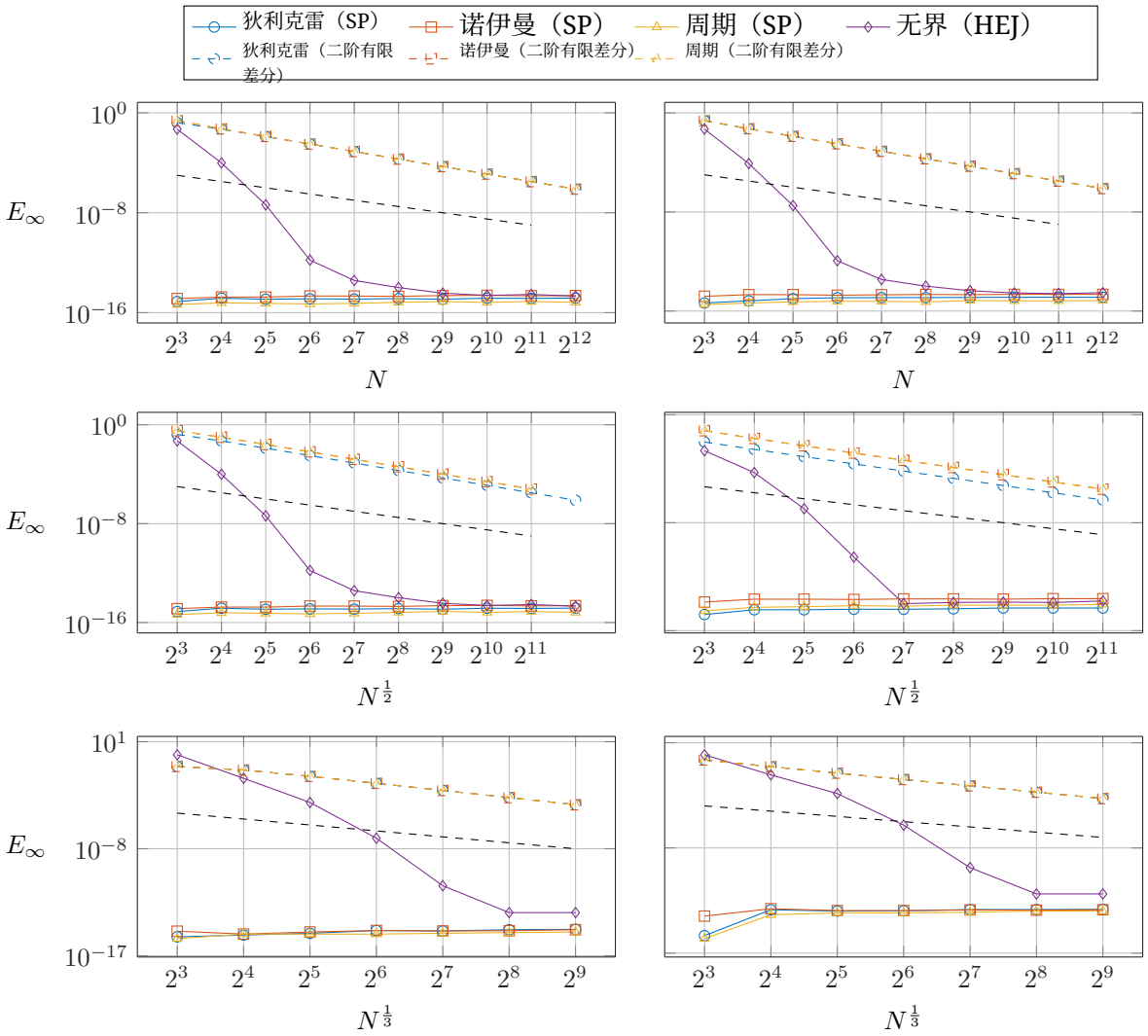


图 5：交错（左）和正则（右）网格配置下，一维（上排）、二维（中排）和三维（下排）求解器类型的误差。虚线表示二次衰减。

其中 c 是一个常数，所有情况均取 10。在一维情况下，ζ 取为到原点的绝对距离。在二维情况下，ζ 取为 x-y 平面内到原点的半径，其中：$\zeta = \sqrt{x^2 + y^2}$，且涡量矢量沿 z 轴方向。在三维情况下，指定 y-z 平面内半径为 r_0 的涡环。任意平面 $\theta = \tan^{-1}(z/y) = \text{const.}$ 内的涡量分布由式 (16) 描述，其中 ζ 是该平面内到涡环坐标的距离：$\zeta = \sqrt{(r - r_0)^2 + z^2}$，且涡量矢量沿该位置处涡环的切线方向。由于柱坐标下拉普拉斯算子含有额外的径向项，二维和三维情况下的 f_a 表达式会更加复杂。为简洁起见，此处不予列出；但可在库代码仓库中相应的无界测试函数文件中找到这些表达式。

对于无界边界条件，首先在实空间中计算格林函数，然后将其正向变换到谱空间。此过程基本上与采用交错网格类型还是规则网格类型无关（同样，这对于当前实现的均匀边界条件类型成立；对于混合边界条件，情况则有所不同 [20]）。因此，为简洁起见，以下结果仅展示交错网格的结果。对于规则网格类型，结果完全相同。如第 2.5 节所述，针对无界边界条件实现了多种核函数，包括二阶至十阶的高斯近似以及谱核表示，这些均归功于 Hejlesen [21]。对于一维情况，仅采用谱型核函数。为简洁起见，本文不展示一维结果，但在图 5 中可以看到一维情况下的谱收敛。二维和三维凸起函数的结果见图 7。可以看出，不同核函数的收敛阶完全符合给定核函数阶数所预期的行为。由于 Hockney-Eastwood 方法的内存开销以及所用设备的可用内存限制，三维中的网格尺寸仅扩展至 $[512, 512, 512]$。尽管如此，收敛

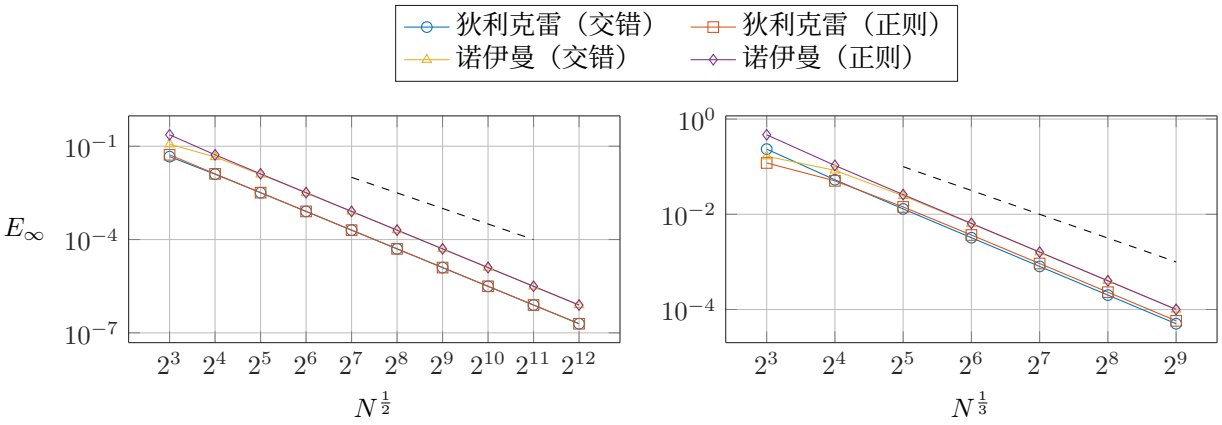


图 6：采用 FD2 型特征值、具有非齐次边界条件的二维（左）和三维（右）求解器在交错和规则网格配置下的误差。虚线表示二次衰减。

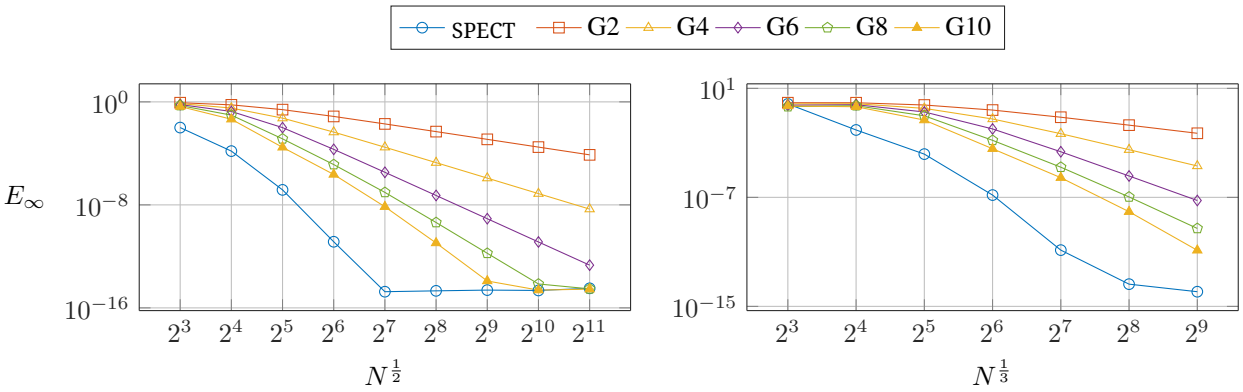


图 7：误差　二维（左）和三维（右）求解器在无界边界条件下，针对　一系列无界核类型　.

行为完全符合预期。通过将微分算子应用于前文所述的分布，对其正确功能进行了测试；然而，对于旋度泊松方程的求解：

$$\nabla^2\vec{\psi} = \nabla \times \vec{f}\,. \tag{17}$$

在此情况下，求解过程与标准无界求解器相同，但卷积后会通过（10）对谱空间中的数据进行空间微分，以提取导数。例如，在流体动力学中，从涡量场提取速度场时，可以使用这种形式的方程[31, 21]。在此情况下，二维和三维凸起函数也有解析表达式，为简洁起见，此处同样省略。二维旋度算子在实际应用中等效于梯度算子。与此前的无界测试案例相比，应用这些梯度项会产生多个输出场，对应于提取场的速度分量。在此情况下，E_∞ 会应用于所有速度分量，以提取最大绝对误差。图 8展示了对二维和三维凸起函数应用谱微分的结果。可以看到，误差的变化规律与直接求解凸起函数的泊松方程时相同。这证明了 SailFFish 所实现的谱微分是有效的。

5 性能

现在对该库的性能进行考察。本文将检查运行时间、讨论求解器的扩展性，并最终给出一系列模拟配置下求解器运行时间的细分情况。如第 3.2.1 节所述，求解器初始化仅执行一次，因此与求解器运行时间相比，这一步的开销较小。在接下来的讨论中，运行时间统计将涵盖求解器执行的第 2 至第 4 步（见第 3.2.2 节），因为输入和输出步骤的计算开销通常较低，而且也很可能非常

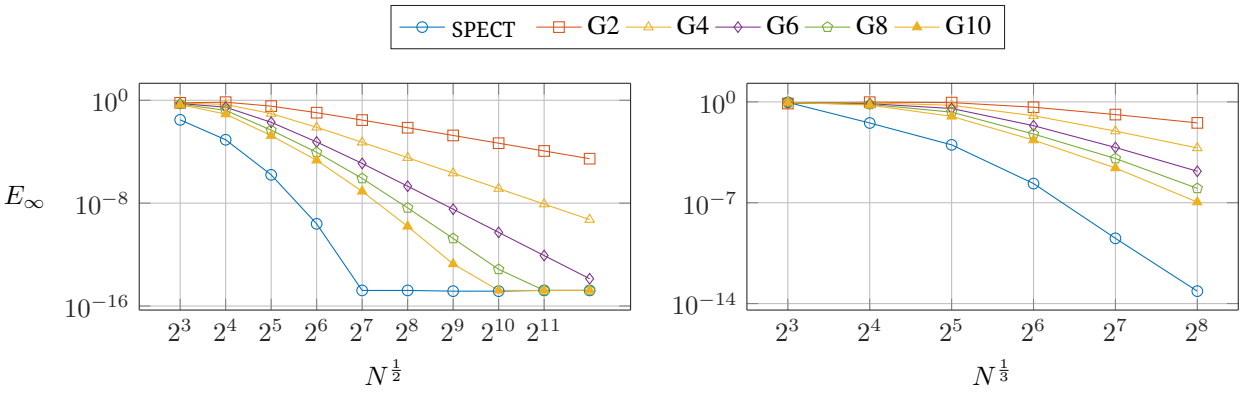


图8：采用无界边界条件时，二维（左）和三维（右）求解器在一系列无界核类型下，分别使用梯度（二维）和 CURL（三维）微分算子的误差。

a依赖于具体应用。这些函数仅作为访问底层数据数组的入口 .
此处的模拟均在一台台式计算机上进行，该计算机配备 Intel(R) 12th Gen Core™ i7-12700KF 处理器，时钟频率为 3.60 GHz，可用线程数为 20，已安装内存为 64 GB。对于使用 cuFFT_数据类型的模拟，计算在一块 NVIDIA GeForce RTW 3070 Ti 图形卡上执行，该卡具有 8GB 设备端内存。所有模拟均采用单精度浮点精度。所示的所有数据点均为十次模拟运行执行时间的平均值。

5.1 执行时间

求解器墙钟时间绘于图9中。如前所述，这些时间是正向变换、卷积和反向变换所耗时间的总和。结果展示了`FFTW_`数据类型和`cuFFT_`数据类型基类的表现。可以看出，对于较小的 N，在 FFTW 中启用多线程会因使用`OpenMP` 所带来的额外开销，以及问题规模相对较小而产生负面影响。然而，对于较大的 N，其优势便显现出来。使用八个线程时，与单线程执行相比，性能提升了一个数量级。还可以看出，使用 GPU 进行计算可显著提升性能。与中央处理器运算相比，计算时间缩短了一个至两个数量级。正如第3节所述，我们希望尽量减少求解器通信

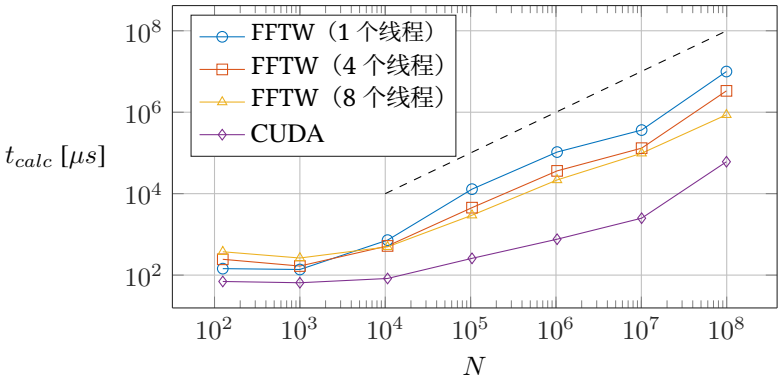


图 9：使用 FFTW 和 CUDA 数据类型时，不同问题规模对应的计算时间（微秒）。求解器类型为采用周期性边界条件的有界三维求解器。

开销。因此，理想情况下，FFT 数据数组应直接在图形处理器上分配内存，并避免中央处理器与图形处理器之间的数据传输。这使问题规模直接受限于图形处理器的可用内存。若图形处理器架构支持半精度浮点数，则可通过使用半精度浮点数来放宽这些限制。

5.2 可扩展性

由于 SailFFish 配置为共享内存执行，因此将讨论强扩展性。执行时间的三个组成部分都很容易使用 OpenMP实现多线程化。卷积过程不过是向量-向量乘法，谱微分例程也同样如此。因此，此处的扩展性与设备相关，并受到上一节所述相同开销限制的约束。对于 FFTW_数据类型，正向和反向变换步骤不过是在 FFTW 中执行正向和反向变换（例如 fftw_execute_dft 函数），因此受 FFTW 多线程实现的影响。要实现精确的线性扩展并不容易，因为单线程与多线程情况下似乎存在显著开销差异。对于某些问题规模，在 4 至 28 个线程之间观察到了线性扩展。对于其他情况，线程数超过某个值后，执行时间会出现不连续的跳变。还需要进一步研究，以确定稳健的扩展行为。不过，在大多数情况下，如图 9所示，使用多个线程能显著提升性能。对于 cuFFT_数据类型，则无法进行类似研究，因为很难指定活跃 GPU 线程的数量。

5.3 执行步骤分解

为考察执行过程中各个不同环节的相对计算开销，我们将这些环节拆分开来，并比较其相对执行时间。研究了多种问题规模和数据类型，相关内容汇总于表 2。针对旋度算子考察了两种情况，因为它与例如（17）的求解尤其相关。对于此处进行的（F 型）模拟，计算时同时使用了八个线程。不过，采用任意数量的并发线程也观察到了类似的行为。这些情况的结果

表 2：图 10 的模拟参数

模拟	数据类型	N 节点	边界条件类型	算子
F1	FFTW	64^3	周期	—
F2	FFTW	128^3	无界	CURL
F3	FFTW	512^3	周期	—
C1	CUDA	64^3	周期	—
C2	CUDA	128^3	无界	CURL
C3	CUDA	512^3	周期	—

如图10所示。可以得出若干结论。卷积过程通常只占总执行时间的一小部分，即使是在谱空间中应用微分算子的情况下也是如此。问题规模越小，这一比例越高。cuFFT数据类型的卷积耗时通常更短。这可能是因为卷积过程采用了（经过调整的）高度优化的矩阵-向量乘法算法。还观察到，cuFFT数据类型存在不对称现象。其原因有待进一步研究。

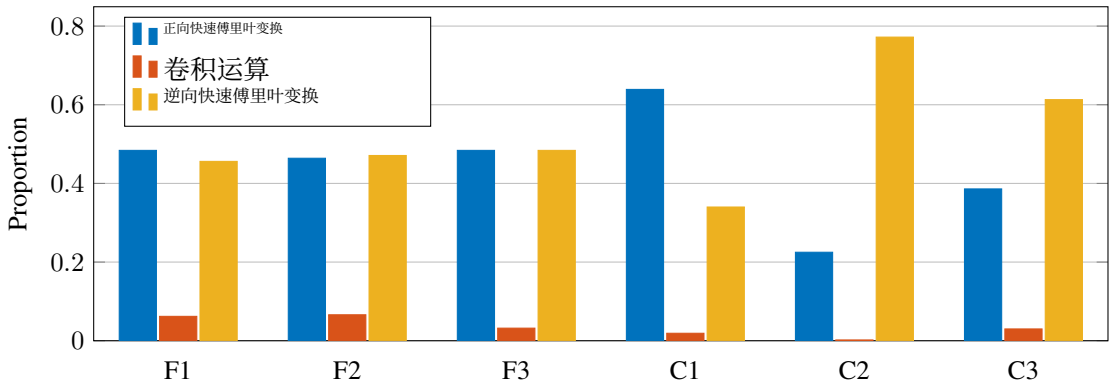


图 10：不同问题规模、数据类型和微分算子的计算时间占比。求解器参数见表2。

6 结论与展望

SailFFish 是一个轻量级并行库，用于求解具有规则网格间距的矩形网格上的泊松方程。该求解器以简洁性和适应性为设计目标，并针对在共享内存应用中执行进行了优化。该求解器能够处理有界泊松问题，其中区域边界可指定狄利克雷、诺伊曼或周期边界条件。通过指定谱特征值，可以在齐次边界条件下以谱精度求解泊松方程。此外，采用二阶有限差分特征值还可以处理狄利克雷或诺伊曼类型的任意非齐次边界条件。SailFFish 还可通过应用 Hockney-Eastwood 方法进行区域加倍，以处理无界边界条件。在这种情况下，可使用多种格林函数。目前仅支持均匀边界条件，但计划在不久的将来实现混合边界条件。希望未来能够应用经过优化的 James-Lacker 算法，以减少无界边界条件情况下的内存开销。所有求解器类型均可采用正则或交错网格配置执行。所有求解器配置均已通过泊松方程的解析解验证，结果表明其收敛速率符合预期。

本文介绍了求解器框架，以便为今后对求解器进行适配提供参考。目前，求解器采用两种 FFT 框架来执行谱空间中的变换和运算：

- FFTW_数据类型，利用 FFTW 库在中央处理器上进行优化计算；以及
- cuFFT_数据类型，利用 cuFFT 库在图形处理器上进行优化计算。

我们希望，在开源环境下，随着新的优化库不断推出，未来能够实现更多数据类型格式。SailFFish 的开发是实现涡量输运方程 GPU 加速求解器的第一步。

致谢

作者谨此感谢 Denis-Gabriel Caprace 多次就快速求解器的话题及其应用展开了许多有益、有趣且愉快的交流。s

参考文献

[1] A. Gholami、D. Malhotra、H. Sundar 和 G. Biros。快速傅里叶变换、快速多极子法还是多重网格？单位立方体均匀网格和非均匀网格上最先进泊松求解器的比较研究。《*SIAM*科学计算杂志》，38(3)：C280–C306，2016年1月。[2] A. Brandt。多重网格开发指南。载于《多重网格方法》，第220–312页，科隆-波尔茨，1982年11月。[3] A Brandt 和 A.A Lubrecht。多层矩阵乘法与积分方程的快速求解。《计算物理学杂志》，90(2)：348–370，1990年。[4] Van Emden Henson。非线性问题的多重网格方法：概述。载于 Charles A. Bouman 和 Robert L. Stevenson 编，《计算成像》，第5016卷，第36–48页。国际光学与光子学学会、SPIE，2003年。[5] C.H. Venner 和 A.A. Lubrecht。《润滑中的多层方法》，第37卷。爱思唯尔科学，1993年。[6] J.C. Adams。Mudpack：用于高效求解线性椭圆偏微分方程的多重网格便携式 Fortran 软件。《应用数学与计算》，34(2)：113–146，1989年。[7] Brandt. A. 多尺度科学计算；2001年综述。载于 T.J. Barth、T. Chan 和 R. Haimes 编，《多尺度与多分辨率方法：理论与应用》，第3–95页。施普林格柏林-海德堡，第1版，2002年。[8] J. Carrier、L. Greengard 和 V. Rokhlin。用于粒子模拟的快速自适应多极子算法。《*SIAM*计算与统计计算杂志》，9(4)：669–686，1988年。[9] L. Greengard 和 V. Rokhlin。三维拉普拉斯方程快速多极子方法的新版本。《数值学报》，6：229–269，1997年。[10] E. Kreyszig。《高等工程数学》。约翰·威立父子公司，第10版，2010年7月。[11] Y. Ying、G. Biros 和 D. Zorin。二维和三维中的核无关自适应快速多极子算法。《计算物理学杂志》，196(2)：591–626，2004年。

[12] A. van Garrel、C.H. Venner 和 H.W. Hoeijmakers。用于风力涡轮机转子流动模拟的快速多层面元法。载于 AIAA SciTech 论坛论文集，美国得克萨斯州 Grapevine，2017年1月。[13] J. Saverin、D. Marten、G. Pechlivanoglou、C.N. Nayeri 和 C.O. Paschereit。空气动力学奇异元的多层模拟。载于 AIAA SciTech 论坛论文集，美国佛罗里达州 Kissimee，2018年1月。[14] R. Cocle、G. Winckelmans 和 G. Daeninck。结合涡格法与并行快速多极子方法，实现高效区域分解模拟。《计算物理学杂志》，227(21)：9091–9120，2008年。纪念 Tony Leonard 70岁生日特刊。[15] J. Saverin。用于空气动力学模拟的多层涡粒子法。博士论文，柏林工业大学，德国柏林，2021年11月。[16] James W. Cooley 和 John W. Tukey。复傅里叶级数机器计算的一种算法。《计算数学》，19：297–301，1965年。[17] G. Pöplau 和 Daniel Potts。非等距网格上的快速泊松求解器：多重网格与傅里叶方法的比较。《SPIE—国际光学工程学会会议论文集》，2003年12月。[18] V. Fuka。PoisFFT——一个免费的并行快速泊松求解器。《应用数学与计算》，267：356–364，2015年。[19] A. Wiegmann。长方体上的快速泊松、快速亥姆霍兹及快速线性弹性静力求解器。技术报告，劳伦斯伯克利国家实验室，1999年6月。LBNL-43565。[20] Denis-Gabriel Caprace、Thomas Gillis 和 Philippe Chatelain。FLUPS：一个基于傅里叶的无界区域泊松求解器库。《SIAM科学计算杂志》，43(1)：C31–C60，2021年1月。[21] M.M. Hejlesen。用于求解泊松方程的高阶正则化方法及涡方法的若干应用。博士论文，丹麦技术大学机械工程系，2016年。[22] M.M. Hejlesen、G. Winckelmans 和 J.H. Walther。一维、二维和三维无界泊松方程的非奇异格林函数。《应用数学快报》，89：28–34，2019年。[23] Steven A. Orszag。伪谱近似与谱近似的比较。《应用数学研究》，51(3)：253–259，1972年。[24] M. Frigo 和 S. G. Johnson。FFTW3 的设计与实现。《IEEE论文集》，93(2)：216–231，2005年。特刊主题为“程序生成、优化与平台适配”。[25] M. Frigo 和 S. G. Johnson。西方最快的傅里叶变换。技术报告 MIT-LCS-TR-728，麻省理工学院，1997年9月。[26] K. Lackner。理想磁流体动力学平衡的计算。《计算物理通信》，12：33–44，1976年。[27] R.A James。孤立源分布的泊松方程求解。《计算物理学杂志》，25(2)：71–93，1977年。[28] F. Vico、Greengard L. 和 M. Ferrando。利用自由空间格林函数进行快速卷积。《计算物理学杂志》，323：191–203，2016 年 10 月。[29] R.W. Hockney 和 J.W. Eastwood。《基于粒子的计算机模拟》。CRC出版社，1988年。[30] J.W. Eastwood 和 D.R.K. Brownrigg。关于孤立系统泊松方程求解的评述。《计算物理学杂志》，32(1)：24–38，1979年。[31] Chatelain C. 和 P. Koumoutsakos。用于具有周期与无界方向的涡流的基于傅里叶的椭圆方程求解器。《计算物理学杂志》，229(7)：2425–2431，2010年。[32] M.M. Hejlesen。泊松求解器。https://github.com/mmhej/poissonsolver，2019年。[33] OpenMP架构审查委员会。OpenMP。https://www.openmp.org/，2021年。[34] M. Frigo 和 S. G. Johnson。FFTW，2021年。[35] 英伟达公司。cuFFT，2022年。[36] 英特尔公司。英特尔数学核心函数库。https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top/fourier-transform-functions.html，2022年。[37] B. Natarajan 和 C. Robeck。clFFT。https://github.com/clMathLibraries/clFFT，2022年。[38] NumPy。NumPy快速傅里叶变换。https://numpy.org/doc/stable/reference/routines.fft.html，2022年。