大规模分布式有限体积通量计算

Ryuichi Sai*
TotalEnergies EP Research & Technology US, LLC. 休斯敦，得克萨斯州，美国
ryuichi@rice.edu

Mathias Jacquelin
Cerebras Systems 森尼韦尔，加利福尼亚州，美国

François P. Hamon
TotalEnergies EP Research & Technology US, LLC. 休斯敦，得克萨斯州，美国

Mauricio Araya-Polo
TotalEnergies EP Research & Technology US, LLC. 休斯敦，得克萨斯州，美国

Randolph R. Settgast 劳伦斯利弗莫尔国家实验室 利弗莫尔，加利福尼亚州，美国

arXiv:2304.11274v1 [cs.MS] 21 Apr 2023

摘要

设计大规模地质碳捕集与封存项目，并确保 CO_2 长期安全封存——将其作为气候变化减缓策略——需要快速且准确的数值模拟。这些模拟涉及求解描述地下流体流动的复杂偏微分方程，所采用的隐式有限体积格式广泛基于两点通量近似（TPFA）。这项任务在计算和存储器方面开销巨大，尤其是在高精度地质模型上执行时。在当前大多数高性能计算架构中，存储器层次结构和数据管理机制不足以应对大规模数值模拟的挑战。因此，设计能够利用数据流和内存计算等替代性且更加均衡的范式的算法至关重要。本文介绍了一种用于 TPFA 计算的算法，能够有效利用 Cerebras CS-2 等数据流架构，从而显著减少存储器瓶颈。与运行在 NVIDIA A100 GPU 上的多个参考实现相比，我们的实现实现了两个数量级的加速。

关键词： 有限体积、两点通量近似、高性能计算、能源、数据流架构、晶圆级引擎、分布式内存

1 引言

大气中的 CO_2 浓度正达到临界水平；仅减少化石燃料消耗或清洁其生产过程已远远不够，还需要采取减缓策略，才能符合联合国的目标。政府间气候变化专门委员会（IPCC）已将地质碳捕集与封存（碳捕集与封存）列为降低温室气体（温室气体）浓度的方法之一。要扩大碳捕集与封存的规模以实现目标，就必须启动许多新项目。设计这些大型碳捕集与封存项目需要进行数值模拟，以确保安全性并实现 CO_2 的长期封存。在监管和商业时间限制内设计安全项目，快速且准确的模拟是关键因素之一。

CCS 中涉及的物理过程的数值模拟需要求解描述地下流体流动的复杂偏微分方程。商业和科研模拟器均基于隐式有限体积（FV）格式，并广泛采用两点通量近似（TPFA）对这些方程进行离散化。这些模拟在高度精细的地质模型上进行，对计算资源和存储器的需求极高。遗憾的是，在目前大多数高性能计算系统中，复杂的存储层次结构（主机和设备）与网络通信之间的不平衡，导致整体计算性能不佳 [17], 因此限制了数值模拟的性能。因此，探索并利用能够实现快速基于有限体积法的流动模拟的替代架构及其相应算法至关重要。

HPC 系统设计的进展推动了科学和商业领域中算法的变革与优化，此前对非层次化架构的探索实现了较高的计算效率 [3, 12]。随着采用分布式内存架构的高度并行系统不断涌现，它们凭借类数据流架构设计（例如 Cerebras 、 Groq [2], SambaNova [16], 等），具备显著更高的内存带宽、更低的内存延迟和更低的内存访问能耗，正成为传统加速系统的替代方案。

在本工作中，我们探索了一种采用新型通量计算实现的数据流架构系统的能力，该实现完全基于局部通信和单级内存，并展示了突破性的性能。本文作出以下贡献：

- 它描述了一种专为数据流架构设计的新型两点通量近似实现；
- 它介绍了一种单层分布式内存架构上的数据通信模式，以支持从主方向邻居和对角邻居进行数据移动；

*该作者是莱斯大学的一名博士生。本研究是在作者任职于 TotalEnergies EP Research & Technology US, LLC. 期间完成的。

- 它介绍了多种优化方法，以提升整体性能；
- 本文既介绍了一种数据流实现，也介绍了基于不同编程模型的参考 GPU 实现；并且
- 本文在一台 Cerebras CS-2 机器上评估性能，并与配备 NVIDIA A100 GPU 的加速系统上的参考实现进行比较，展示了出色的性能特征。

本文的组织结构如下。首先在第2节回顾相关工作。第3 节概述问题领域。第4 节讨论数据流架构；第5节介绍我们将通量计算算法映射到数据流架构的方法。第6节介绍我们的 GPU 参考实现。我们在第7节评估并比较这些实现。最后，我们在第8节和第9节给出讨论与结论。

2 相关工作

用于地质碳捕集与封存模拟的有限体积方法。隐式有限体积格式 [14]广泛用于模拟地质多孔介质中的流动和输运。计算网格单元之间的通量及其对状态变量的导数，是模拟器工作流程中的关键步骤。在本研究中，我们利用数据流架构加速一个简化的有限体积通量计算内核。这是将整套离散化流动方程移植到数据流模型的前期步骤。

这些工作最终可能有益于 GEOS 等碳捕集与封存模拟器 [13]。GEOS 是一个开源的多物理场 C++ 计算平台，用于模拟碳捕集与封存及其他地下能源系统 [5–8, 10]。GEOS 采用耦合有限元–有限体积公式，模拟含断层和裂缝的热多相流与力学过程。其设计注重模块化和可扩展性，便于添加新的求解器或修改现有求解器。该代码利用 RAJA 库实现可移植的执行策略，并面向大规模并行 CPU 和 GPU 架构。

加速数据流架构上的应用。数据流架构及其在高性能计算工作负载中的实现最初在一项先前研究中进行了探索 [18], ，该研究在 Cerebras CS-1 系统上针对 $600\times595\times1536$ 网格上的 7 点模板计算，实现了 0.86 PFLOPS。虽然在 CS-2 上评估了更新版本 [20] of [18]，但该版本主要侧重于 Python 接口，以解决与底层抽象相关的编程效率问题。本研究受到这项工作的启发，但不同之处在于我们使用 Cerebras 软件语言（CSL）进行实现。该语言提供了更高层次的抽象，并支持可复用的函数和模块。此外，我们的问题领域涉及更复杂的计算和数据通信模式。此外，本研究着重于在实际应用中实现最高性能；该应用涉及复杂的数据通信，并要求进行算术强度更高的计算。

在 [11], 中，提出了一种存储器布局，以便在数据流架构上高效计算大型三维数据网格，充分利用其片上存储器和二维互连网络。此外，还设计了一种通信策略，以利用该架构快速的互连网络。与 [11]中的工作相比，我们的工作在计算物理过程时需要更多存储空间，并涉及更复杂的数据通信。此外，我们的问题将数据移动范围扩展到相邻正交邻居之外，还包括对角邻居；这一点尚未得到探索。我们还利用了多项优化来提升性能。

其他类似数据流的架构也已被提出。特别是， [9] SambaNova 从机器学习的角度展示了若干适用于科学应用的重要成果。另一种面向数据流的 [15]高性能计算科学计算方法是 Graphcore，它探索了具有高运算强度的应用，例如基于模板的格子玻尔兹曼方法和高斯滤波。

3 地质碳封存模拟的有限体积法

在本研究中，我们考虑一种可压缩单相流达西尺度模型，用于描述向地质地层注入流体（超临界 CO_2）所引起的压力变化。控制方程组由达西定律和质量守恒方程组成：

$$\boldsymbol{u} = -\frac{\kappa}{\mu}(\nabla p - \rho \boldsymbol{g}), \qquad \text{(Darcy's law)}, \tag{1a}$$

$$\frac{\partial}{\partial t}(\phi\rho) + \nabla\cdot(\rho\boldsymbol{u}) = 0, \qquad \text{（质量守恒）。} \tag{1b}$$

其中， $\boldsymbol{u}$ 为达西速度， κ 为（标量）渗透率系数， p 为压力， $\boldsymbol{g}$ 为按重力加速度加权的重力矢量。孔隙度 ϕ 和密度 ρ均随压力线性变化。假设黏度 μ为常数。

通常将低阶有限体积（FV）格式与隐式（后向欧拉）时间离散相结合，对这一近似椭圆型系统进行离散。对于由 n 个网格单元组成的网格，离散后的非线性方程组为：

$$V_K \frac{\phi_K^{n+1}\rho_K^{n+1} - \phi_K^n \rho_K^n}{\Delta t} + \sum_{L\in \text{adj}(K)} F_{KL}^{n+1} = 0, \tag{2}$$

其中， V_K 是网格单元 K 的体积， ϕ_K^{n+1}和 p_K^{n+1}分别是时间 $n+1$时网格单元 K 的孔隙度和压力， Δt 是时间步长。在本文中，我们重点计算离散界面通量，记为 F_{KL}^{n+1}，并忽略累积项。我们采用标准的两点通量近似

方法，并结合单点迎风格式，此时通量为：

$$F_{KL}^{n+1} = \Upsilon_{KL}\lambda_{\text{upw}}^{n+1}\Delta\Phi_{KL}^{n+1}, \tag{3a}$$

$$\Delta\Phi_{KL}^{n+1} = p_L^{n+1} - p_K^{n+1} + \rho_{\text{avg}}^{n+1} g(z_L - z_K), \tag{3b}$$

其中， ρ_{avg}^{n+1} 是时间 $n+1$ 时网格单元 K和 L 中流体密度的平均值， $z_L - z_K$ 是网格单元 L 和 K中心之间的高程差。传输率Υ_{KL}是一个考虑网格单元几何形状及其渗透率的系数。迎风流体流动度 $\lambda_{\text{upw}}^{n+1}$根据密度、黏度和势差计算如下：

$$\lambda_{\text{upw}}^{n+1} = \begin{cases} \dfrac{\rho_K^{n+1}}{\mu} & \text{if } \Delta\Phi_{KL}^{n+1} > 0 \\ \dfrac{\rho_L^{n+1}}{\mu} & \text{otherwise} \end{cases} \tag{4}$$

流体密度基于网格单元压力，采用微小可压缩性假设进行计算：

$$\rho_K^{n+1} = \rho_{\text{ref}} e^{c_f (p_K^{n+1} - p_{\text{ref}})}, \tag{5}$$

其中c_f表示流体压缩性， p_{ref}为恒定参考压力， ρ_{ref} 为恒定参考密度。

结果： 残差的通量部分，$\mathbf{r}_{\text{flux}}$

$\mathbf{r}_{\text{flux}} := \mathbf{0}$; 对每个网格单元 K， 执行：对K的每个相邻网格单元 $L \in$， 执行：使用公式5计算 K 和 L 中的密度；使用公式3计算通量 F_{KL} ；将残差增量累加到网格单元 K中：$(\mathbf{r}_{\text{flux}})_K = (\mathbf{r}_{\text{flux}})_K + F_{KL}$；结束循环；结束循环。算法1：本文所考虑的FV通量计算算法。

算法 1 的外层循环遍历网格单元，内层循环则遍历每个网格单元的邻居。对于每个邻居，计算一个通量，并用其累加该网格单元的局部残差。这里，我们考虑在笛卡尔网格上实现该算法的三维版本。因此，每个内部网格单元都有六个邻居（同一 X-Y 平面内的四个邻居、顶部的一个邻居和底部的一个邻居），并涉及六个通量的计算。在我们的实现中，还会计算网格单元与其对角邻居之间的四个通量，以便为精度更高的方案或更复杂的网格准备通信模式。算法1 也可以应用于非结构化网格，但需要更复杂的通信模式。算法 1 被应用 1,000 次，每次调用都使用不同的压力向量。

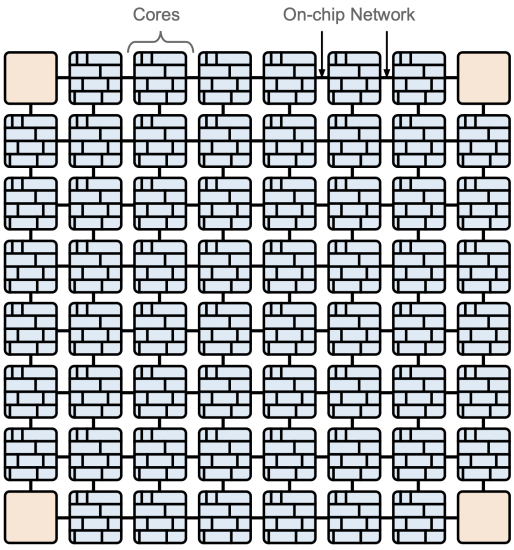


图 1。一种典型的数据流架构实现。图片来源： [2]。

4 一种替代计算方法

在本工作中，主要贡献之一是有效利用一种结合数据流计算与内存计算范式要素的架构，并能够并发访问大量计算1资源。图 1展示了此类架构的一种典型实现，并且在 [1] 其中一些主题得到了探讨。在本文其余部分，为简洁起见，我们将这类新兴架构称为数据流架构；但在后续章节中，我们会在介绍实现细节之前，先讨论一种具体的架构。其主要的颠覆性特点在于不依赖深层存储层次结构，以及可用存储器相对于计算单元的局部性。例如，所有计算和存储资源都集成在同一晶圆上，并通过低延迟、高吞吐量的互连网络彼此连接。

在此类架构上映射算法时，需要分层考虑两个主要方面：首先，在顶层，关键是将数据有效分配到大量处理单元之间，这些处理单元采用非集中式的相干存储器接口。其次，数据进入计算单元的本地存储器后，底层的挑战在于尽可能充分地利用数据，再与相邻计算单元共享。顶层关注的是确保数据在处理单元之间高效流动；在更传统的架构中，这一层通常由 MPI 实现。底层关注的是高效利用类似暂存区的存储器系统；而在传统架构中，对于带宽受限的算法，主要挑战是充分利用复杂的存储层次结构。除此之外，

这一点还意味着，大多数现有高性能计算系统都要求开发者了解加速计算节点主机端和设备端的完整存储层次结构。在传统架构中，后者通常通过 CUDA/HIP 或 RAJA/Kokkos、OpenMP 等高度可移植的编程模型来处理。

5 基于数据流架构的有限体积通量计算

本节详细介绍如何在数据流架构上使用两点通量近似实现有限体积通量计算。首先，我们概述数据和算法在互连网络上的映射方式。接着，我们介绍水平面上的数据移动模式（包括正交邻居之间以及对角邻居之间的数据移动）和两点通量近似计算。随后，将算法扩展到第三维，以支持多个平面。之后，将实现扩展到多次迭代。最后，详细说明用于提升性能的优化技术。

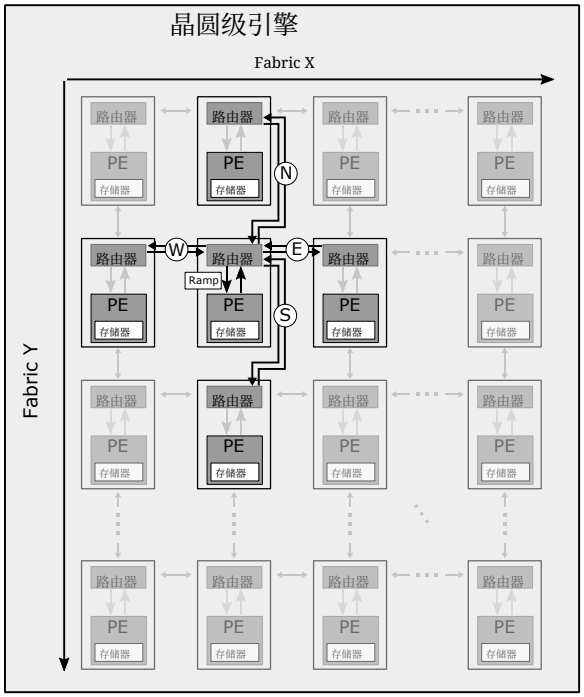


图 2. Cerebras 晶圆级引擎的简化概览

图2 简要概述了 Cerebras 晶圆级引擎（更多详情请参见 [11]）。晶圆级引擎（WSE）是数据流架构的一个示例。与大多数数据流架构实现一样，它配备了一个二维网格互连网络，用于连接执行计算的处理单元（PE）。每个处理单元都有自己的专用本地存储器，并连接到一个路由器。该路由器管理五条全双工链路：1) 北、东、南、西链路连接到相邻路由器，使数据能够在处理单元之间流动；2) Ramp 链路使数据能够在处理单元与其自身路由器之间流动。每条链路都以 32 位数据包传输数据。每个数据包都关联一个颜色或标签，用于路由并指示消息类型。

5.1 数据映射

在本研究中，物理问题由三维笛卡尔网格表示，其中每个网格单元都与 10 个邻居存在通量连接。在 X-Y 平面（水平面）上，每个网格单元有 8 个通量连接，其中 4 个沿正交坐标轴方向，另外 4 个沿对角线方向。对于每个通量连接，都会计算一个局部通量，并将其与该网格单元的所有其他通量汇总。

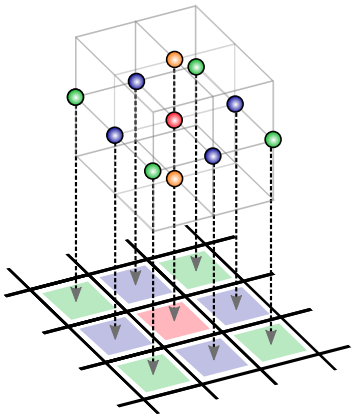


图 3. 采用基于网格单元的方法，将三维问题映射到由处理单元组成的二维互连网络。

为计算网格中所有网格单元的通量，我们考虑了两种映射技术，即基于面和基于网格单元的映射，如图3 所示。后者是映射到互连网络中最直接的方法，每个网格单元映射到一个处理单元。这使我们能够充分利用该架构的计算、存储和通信资源。

数据域是一个尺寸为 $N_x \times N_y \times N_z$ 的三维笛卡尔网格，如图 4 所示。我们对数据域进行分解，使 Z 维度上的每个网格单元都映射到同一个处理单元，而 X 和 Y 维度则映射到互连网络的两个轴上。更准确地说，我们将三维网格中坐标为 (x, y, z) 的网格单元映射到处理单元 (x, y)。整个 Z 维度都存储在处理单元的本地存储器中。正如 [11, 18], 此前所示，这一策略优化了数据流架构的利用率，并最大限度地提升了此类高性能计算应用的潜在并行性。

每次通量计算需要三类数据：当前网格单元的单元中心数据、相邻网格单元的单元中心数据，以及每对网格单元的面中心数据

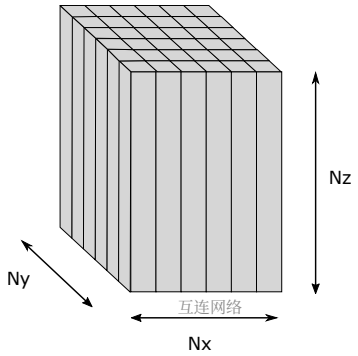


图 4.大小为 $N_x \times N_y \times N_z$ 的三维网格。X 和 Y 维映射到 WSE 的 PE 网格上，而 Z 维映射到每个 PE 的存储器上。

网格单元。3还展示了网格单元依赖关系。中心网格单元及其顶部和底部邻居被映射到标为红色的处理单元上。其四个方向的邻居网格单元及对应的处理单元标为紫色，而对角网格单元和处理单元则显示为绿色。每个处理单元为其当前残差、压力和重力系数，以及网格单元与其邻居之间通量所对应的 10 个传递率分配存储空间。每个处理单元还分配空间，以便在每次迭代中接收来自同一 X-Y 平面上全部八个相邻网格单元的压力和重力系数。下面将进一步介绍通信模式。

5.2 通量计算中的分布式数据通信

如前所述，计算特定网格单元需要用到三个维度上 10 个相邻网格单元的数据。具体而言，网格单元 $(x,\ y,\ z)$ 需要：

(a) 来自 X-Y 平面正交邻居的网格单元，对应于网格单元 $(x+1,\ y,\ z)$、$(x-1,\ y,\ z)$、$(x,\ y+1,\ z)$ 和 $(x,\ y-1,\ z)$；
（b）来自 X-Y 对角邻居的网格单元 $(x+1,\ y+1,\ z)$、$(x-1,\ y-1,\ z)$、$(x-1,\ y+1,\ z)$ 和 $(x+1,\ y-1,\ z)$；以及
（c）沿 Z 维度的网格单元 $(x,\ y,\ z+1)$ 和 $(x,\ y,\ z-1)$。

5.2.1 来自 X-Y 方向邻居的数据通信。

在 X-Y 平面中，处理单元（PE）必须沿着互连网络的每个基本方向与紧邻的邻居通信。具体而言，在 X 维度上，处理单元必须与位于网格单元（$x+1,\ y,\ z$）的东侧邻居通信，并与位于网格单元（$x-1,\ y,\ z$）的西侧邻居通信。在 Y 维度上，处理单元必须与位于网格单元（$x,\ y-1,\ z$）的北侧邻居通信，并与位于网格单元（$x,\ y+1,\ z$）的南侧邻居通信。与 [11], 中采用的通信策略类似，每种通信模式都分配两个专用颜色，一个用于发送数据，另一个用于接收数据。

收到数据后，将立即以异步方式进行相应的通量计算。该策略可使通信与计算重叠进行。

在每个通信阶段，多个处理单元会同时向四个方向发送数据，每个 发送 处理单元沿各自方向发送长度为 $N_z \times 2$ 的本地数据块。图 5a显示，每个处理单元会同时从四个方向接收数据。每个 接收 处理单元从四个邻居处接收总长度为 $N_z \times 2 \times 4$ 的数据。

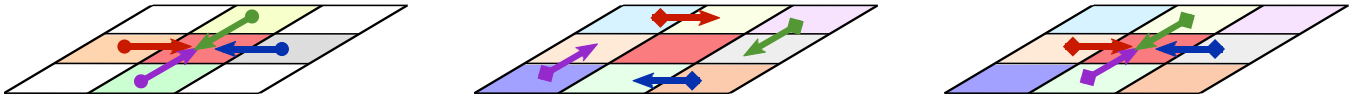
要将处理单元从 发送 切换为 接收，我们会控制处理单元路由器，并在运行时更改其配置。如图 6a所示，每个处理单元都定义了两个切换位置，分别用于发送和接收。发送完本地数据后，发送 处理单元会发出一条命令，切换自身及相邻路由器的配置，使原来的 发送 处理单元变为 接收处理单元，而原来的 接收 处理单元变为 发送 处理单元。随后，新的 发送 处理单元会向自身及其邻居发送本地数据和切换命令，以继续实验。由于数据和命令均由直接相邻的处理单元发送和接收，因此处理单元可以在发送 和 接收 状态之间交替，如图 6b所示。经过两个步骤后，所有处理单元都已发送和接收完所有数据。

5.2.2 来自 X-Y 对角邻居的数据移动。

据我们所知，我们是首个探索数据流架构中来自对角线方向的数据移动的研究者。尽管这对于评估第3节所介绍的数学方案并非必需，但我们仍然实现了这一功能，因为我们认为有必要为更复杂的通信模式做好准备。由于互连结构布局是一个规则的二维网格，沿对角线方向进行数据通信具有独特的挑战。每个处理单元的路由能力仅限于基本方向。因此，虽然沿直接基本方向邻居进行数据通信只涉及两个相邻的处理单元，但对角邻居之间的数据传输需要多个处理单元，其中一些处理单元充当中介。中介网格单元必须同时是源单元及其对角目标单元的直接邻居，并且相对于这两个网格单元分别共享一个基本方向轴。

如图5b所示，要将数据从西北角网格单元传输到中心网格单元，我们首先使用向东的通道，将数据从西北网格单元传输到北侧的中介网格单元。然后，如图5c所示，我们使用向南的通道，将数据从北侧网格单元传输到中心网格单元。

为实现与所有四个对角相邻网格单元的并发通信，采用了一种轮转协调的同步机制。该机制让每个角点处理单元顺时针或逆时针发送数据。只要所有四个网格单元都遵循相同的方向，并且每个网格单元都

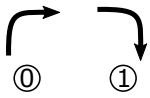


(a) 步骤 1.a - 北、南、东和西处理单元向目标处理单元发送数据。

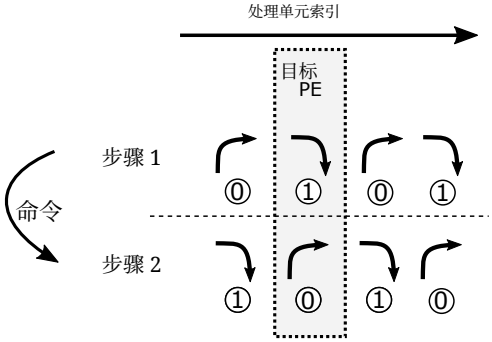
(b) 步骤 1.b - 角点处理单元向北侧发送，南侧、东侧和西侧处理单元（与步骤 1.a 并发）步骤 1.a）。

(c) 步骤 2 - 北、南、东和西处理单元将数据从角点处理单元转发到目标处理单元。

图 5. X-Y 平面上的通信模式。目标处理单元（红色）接收来自周围全部 8 个处理单元的数据。



(a) FV通量计算所用的路由器配置。配置 0 对应作为广播根节点的发送 处理单元所采用的配置，配置 1 则由接收处理单元使用。



(b) 从西侧获取目标处理单元所需全部数据需要两个通信步骤。相应的路由器配置以圆圈中的数字标出。每一步都会通过广播模式发送一条路由器命令，将配置从一种路由器配置切换为另一种。

图 6. 向东的局部　　广播操作，用于 FV通量计算中沿 X 维度交换网格单元值　　n.

分配给不同的中间单元后，所有角点处理单元的数据传输都会同时进行。

从实现角度来看，X-Y 正交邻居之间的数据通信保持不变，每个发送处理单元只需向四个方向发送数据。对于一个中间处理单元，它会与相邻网格单元计算该面的部分通量，同时将接收到的数据发送给接收处理单元。如图接收处理单元所示（见图5），会两次从四个正交方向接收数据：首先，它会与直接基本方向邻居计算这四个面的部分通量；随后，对角邻居的数据到达，它再与对角邻居计算这四个面的其余通量。

5.3 优化

我们重点介绍一些在计算性能和存储器利用率方面最有效、影响最大的优化。简要提醒一下，同一垂直列中的网格单元共享一个处理单元的私有内存，因此，减少每个处理单元上的存储器消耗对于容纳尽可能大的问题至关重要。就性能而言，处理单元之间的异步通信对于将数据移动与计算重叠起来必不可少。此外，还采用向量化来充分利用互连网络的带宽。

5.3.1 处理单元私有存储器节省策略。

在本地私有内存中，处理单元需要存储指令。除了存储网格单元数据和面数据外，本地存储器还必须容纳用于数据广播和计算的额外缓冲区。在此实现中，数据缓冲区复用至关重要；这类似于寄存器分配优化，通常由编译器执行，但本工作采用手工编写的方式实现。其影响有两方面：通过最大限度地减少实现所需的存储器用量，可以解决规模更大的问题。此外，在某些情况下，覆盖或复用数据缓冲区可以避免数据复制，从而进一步提高计算效率。

5.3.2 异步通信。

通量计算算法的实现依赖于处理单元之间通过异步通信交换数据。数据沿四个方向异步发送和接收。这有两大主要优势。首先，同时进行传输可增加利用互连级并发的机会。其次，非阻塞通信能够将传输与有用的计算重叠，从而尽可能地隐藏相关开销，因为互连网络和路由器的运行完全独立于处理单元。

5.3.3 浮点运算的向量化。

大多数硬件架构都提供专用机制来处理数据数组（或向量）。在这里讨论的架构中，这通过使用特殊寄存器来实现，这些寄存器存放 数据结构描述符（*DSD*），充当向量，供给定指令对其进行操作。DSD 包含指令将要操作的数据数组的地址、长度和步幅等信息。这使得向量化指令能够在硬件层面充分利用 SIMD 单元（单精度运算时最多可利用 2 个）。用数据流术语来说，对 DSD 执行操作的指令就像一个过滤器，数据从中流过。值得注意的是，无论输入和输出数组有多长，指令的吞吐量都保持恒定，因为没有缓存可以被填满。因此，使用这些专用寄存器显著提升了性能。

6 GPU 上的有限体积通量计算

本节介绍我们在 GPU 上的参考有限体积通量实现，作为与数据流架构上的实现进行比较的基准。

参考实现采用基于 TPFA 的标准隐式有限体积通量计算方法。该数值方法与商业模拟器以及 GEOS 等开源基准模拟器采用的方法类似 [13]。为了与我们在数据流架构上的实现进行公平比较，参考实现采用基于网格单元的循环模式。针对基于网格单元的方法，我们对参考实现进行了优化，使其在 GPU 上尽可能快速。

首先，我们在主机存储器和设备存储器上分配空间。随后，将大小为 $N_x \times N_y \times N_z$ 的网格数据加载到主机存储器中，内存布局中 X 维度为最内层，Z 维度为最外层。接着，将所有数据从主机存储器复制到设备存储器。由于我们在配备了足够大设备存储器的最新硬件上评估 GPU 内核，可以一次性加载所有数据，因此无需进行数据域分解，也避免了主机存储器与设备存储器之间频繁传输数据所耗费的时间。

对于算法1的每次应用，我们都会启动一个采用 3D 线程块的 GPU 内核，并仔细地将每个网格单元映射到一个 GPU 线程。

我们的内核启动基于 RAJA [4]，这在物理模拟中广泛使用，以实现最大的可移植性。循环模式通过 RAJA 内核策略实现，如图 7所示，作用于整个数据网格大小为 $N_x \times N_y \times N_z$ 的嵌套循环空间。我们指示 RAJA 启动 GPU 内核，并将线程块大小设为 1024 ，以符合 GPU 每个线程块最多包含 1024 个线程的限制，同时最大限度地提高线程并行度和 GPU 利用率。我们还指导 RAJA 将 GPU 线程块分块为 $16 \times 8 \times 8$，其中 16 是最内层维度

```
RAJA::KernelPolicy<
  RAJA::statement::CudaKernelFixed< 16 * 8 * 8,
  RAJA::statement::Tile<1, RAJA::tile_fixed<8>, RAJA::cuda_
  block_y_direct, RAJA::statement::Tile<0, RAJA::tile_fixed<16>,
  RAJA::cuda_block_x_direct, RAJA::statement::For<2,
  RAJA::cuda_block_z_direct, RAJA::statement::For<1,
  RAJA::cuda_thread_y_direct, RAJA::statement::For<0,
  RAJA::cuda_thread_x_direct, RAJA::statement::Lambda
  <0>>>>>>>>>;
```

图 7。内核启动中使用的 RAJA 执行策略。

维度大小。最后，我们在三个相应的维度上分别应用 RAJA 的 GPU 线程循环策略：*cuda_线程_z_循环*、*cuda_线程_y_循环*、*cuda_线程_x_循环* 策略。

RAJA 为每个索引 (x, y, z) 提供一个 C++ lambda，每个索引都直接映射到数据网格中的一个网格单元。每个 GPU 块线程都会被调度为并发调用一个设备函数，为其各自映射的网格单元执行 FV通量计算。首先，每个线程并发获取自身的单元数据以及其十个相邻网格单元的所有单元数据。接下来，对于每个邻居，它使用传输率、局部网格单元值及其邻居值执行通量计算，并生成一个局部通量值。然后，它汇总所有局部通量并更新当前网格单元值。

在参考实现中执行通量计算的函数，在逻辑上与我们数据流实现中的函数完全相同。由于这两种实现都采用类 C 语言编写，通量函数的代码在语法上也很相似。唯一的区别是，数据流架构中每个处理单元的存储空间彼此隔离，而 GPU 设备存储器由所有线程共享。因此，我们无需在网格单元之间传输数据，可以直接通过简单的索引运算访问数据。内核完成算法 1的所有应用后，数据会从设备存储器复制回主机存储器。

为验证数值精度和性能的有效性，我们使用 CUDA 编程模型手动开发了第二个 GPU 内核。这个手工编写的 CUDA 版本采用相同的存储器布局、使用相同的分块尺寸，并执行相同的 FV通量计算。不过，它通过手动计算的线程块维度来启动内核，并仔细计算到网格单元的索引映射。此外，它还需要进行边界检查，以确保网格单元仍位于数据网格范围内。

7 实验评估

我们在 Cerebras CS-2 系统上评估数据流实现，并将其与运行于 NVIDIA A100 GPU 上的两种参考实现进行比较。我们讨论了它们各自的性能特征。

架构/语言	Avg.	S.D.
数据流/CSL	0.0823	0.0000014
GPU/RAJA	16.8378	0.0194403
GPU/CUDA	14.6573	0.0111278

表 1. CS-2 和 A100 上的时间测量

7.1 实验配置

CS-2 是 Cerebras 开发的第二代机箱，采用 7 纳米技术，并在 WSE-2 互连网络上搭载了总计 850,000 个处理单元。我们的实验使用互连网络的最大尺寸 750×994，以充分利用大规模分布式计算能力。由于互连网络边界周围一层较薄的处理单元预留作系统自身用途，因此 SDK 允许使用的最大尺寸就是该尺寸。虽然 CS-2 连接到一台 Linux 服务器，但服务器仅用于调度工作负载；实验期间不会在 Linux 机器上执行任何计算。我们使用了 Cerebras SDK 0.6.0，该版本包含更快的编译器和额外的实用程序库，例如 内存复制，与 [11] 中报告的先前版本相比，性能有所提升。

作为参考的基于 GPU 的平台是 TotalEnergies 的 *Cypress*。主机配备一颗 16 核 AMD EPYC 7F52 CPU 和 256 GB RAM，搭载四块 NVIDIA A100 GPU，每块 GPU 具有 40 GB 的设备存储器。主机运行 CentOS 8，我们的工具链包括 NVIDIA 驱动程序 460.32.03、CUDA 11.2、GCC 8.3.1 和 RAJA 2022.03。

我们将 CS-2 生成的数值结果与参考实现生成的结果进行比较和验证。

7.2 性能结果

下面给出 FV通量计算的性能结果。我们讨论了各种观测结果，以深入了解 CS-2 的性能特征。

测量与比较。 表 1 给出了执行算法 1000 次的墙钟时间测量结果1，针对 $750 \times 994 \times 246$ 网格。我们的数据流实现的性能显示在 Dataflow/CSL 行中，而在 NVIDIA A100 GPU 上运行的参考实现显示在 GPU/RAJA 行中。此外，在同一 GPU 上运行的手工编写 CUDA 版本显示在 GPU/CUDA 行中。对于每种实现，我们报告多次运行所得的平均内核时间和标准差。

我们基于 RAJA 的实现性能与基于 CUDA 的版本相当。它平均每个流式多处理器（SM）使用 30.79 个线程束，理论上限为 32 个线程束。其占用率为 48.11%，理论占用率为 50%。其算术强度为 2.11 浮点运算/字节，性能达到 6012 GFLOPS，即峰值性能的 76%。这些结果表明，我们的参考实现具有高性能特征，并能高效利用硬件。与基于 RAJA 的参考实现相比，我们的数据流实现实现了 204 倍加速。

如 [11], 所报告，在实验过程中达到稳态时，CS-2 的平均功耗为23 kW。就FV通量模拟而言，这相当于 13.67 GFLOP/W。相比之下，在相同工作负载下，A100的峰值功耗为250 W。总体而言，数据流实现的能源效率比参考实现高2.2倍，且未计入主机或网络通信设备。

扩展性比较。我们进行了弱扩展性实验，类似于 [11], 所示的实验，以展示FV通量计算的可扩展性。我们调整X和Y方向的网格维度，同时保持Z方向的维度不变。X和Y方向的维度最大扩展至 750×994。

表2 展示了每次实验的网格大小、CS-2上实现的吞吐量（单位为Gigacell/s）、计算1，000 次算法1 所需的墙钟时间，以及在A100上运行的RAJA参考实现的内核时间。此处报告的计时仅对应于设备上执行的计算。在CS-2上，时间通过SDK <time> 库记录，该库使用硬件时间戳计数器。在A100上，我们使用CUDA提供的运行时函数（例如 *cudaEventRecord*、*cudaEventSynchronize*和 *cudaEventElapsedTime*）测量GPU内核运行的耗时。结果表明，FV通量计算在CS-2上也实现了近乎完美的扩展性。

数据通信成本。 为了进一步了解在 CS-2 上进行 FV通量计算的性能特征，我们开展了一项实验，修改了数据流实现，移除所有通量计算，仅专注于数据通信。我们在大小为 $750 \times 994 \times 246$ 的网格上运行了此修改版代码，执行了 1000 次算法 1，得到平均数据通信时间为 0.0199 秒。总设备时间为 0.0823 秒，由此可推算所有通量计算共耗时 0.0624 秒。

表 3 展示了时间分布分析结果。我们发现，数据通信成本占设备总时间的 24。18% ，而其余的75。82% 用于浮点计算。该分析进一步揭示了 CS-2 上 FV通量计算的性能特征，可为优化该平台上此算法的未来实现提供参考。

Nx	Ny	Nz	总计数量 网格单元数	吞吐量 [十亿网格单元/秒]	CS-2 时间 [秒]	A100 时间 [s]
200	200	246	9,840,000	121.01	0.0813	0.9040
400	400	246	39,360,000	481.43	0.0817	3.2649
600	600	246	88,560,000	1078.79	0.0821	7.2440
750	600	246	110,700,000	1347.21	0.0821	9.6825
750	800	246	147,600,000	1794.01	0.0822	13.2407
750	950	246	183,393,000	2227.38	0.0823	16.8378

表 2. 不同网格尺寸下的可扩展性实验结果。

	时间 [s]	百分比 [%]
数据移动	0.0199	24.18
计算	0.0624	75.82
总计	0.0823	100.00

表 3. 在 CS-2 上采用最大可用网格大小时的时间分布。

操作	FLOP	内存流量	互连网络流量
60 次浮点乘法	1	2 次加载，1 次存储	0
40 浮点减法	1	2 个加载，1 个存储	0
10 FNEG	1	1 次加载，1 次存储	0
10 次浮点加法	1	2 次加载，1 次存储	0
10 次 FMA	2	3 次加载，1 次存储	0
16 次 FMOV	0	1 次存储	1 次加载

表 4. CS-2 上单个网格单元的指令和内存访问次数。

7.3 屋顶线模型

屋顶线 [19, 22]模型直观地展示代码相对于机器峰值性能的表现。该模型综合考虑代码的运算强度、内存带宽和整体性能，并将它们呈现在一张图表中。通过将代码的性能与平台的性能上限进行比较，屋顶线模型还能为代码的潜在优化提供有价值的见解。

为了计算将我们的数据流实现置于屋顶线图上时的性能特征，我们采用了 [11]中描述的相同屋顶线模型。表4详细列出了有限体积通量计算在网格单元级别所需的指令汇总，包括浮点运算次数、存储器流量和互连网络流量。每个处理单元执行 N_z 个网格单元的计算，每个网格单元有 10 个面，每个面执行一次两点通量近似计算。每次通量计算包含 6 次浮点乘法、4 次浮点减法、1 次浮点加法、1 次融合乘加和 1 次浮点取负；融合乘加需要两次浮点运算，其他操作各需要一次。因此，每次通量计算需要 14 次浮点运算，每个网格单元总计执行 140 次浮点运算。

这些浮点运算总共需要在存储器中加载和存储 406 个单精度 32 位浮点数，并从互连网络加载 16 次。这些互连网络加载来自与八个相邻处理单元之间的数据通信。网格中顶部和底部网格单元的数据访问只需访问存储器，因为它们位于同一处理单元的存储器中，无需访问互连网络。因此，相对于存储器访问，算术强度为 0.0862 次浮点运算 / 字节；相对于互连网络传输，算术强度为 2.1875 次浮点运算/字节。网格大小为 $750 \times 994 \times 246$，计算耗时 0.0823 秒，性能达到 311.85 太浮点运算每秒，如图 5（上）所示。我们的数据流实现对于存储器访问是带宽受限的，对于互连网络访问则是计算受限的。不过，相对于存储器访问的算术强度已接近计算受限（0.0892 次浮点运算/字节）。

对于 GPU，我们使用 Empirical Roofline Toolkit 测量机器性能特征 [21]，并使用 NVIDIA Nsight Compute 在 A100 上分析 RAJA 参考实现。GPU 屋顶线图见图 8 （下图）。我们的 RAJA 实现受存储器带宽限制，相对于其算术强度达到了峰值性能的 76%。这些结果也表明，我们的参考实现可作为可靠的比较基准。

8 讨论

本研究聚焦于在分布式内存系统上使用两点通量近似进行 FV通量计算，展示了这一方法的可行性及其能够带来的显著加速。FV通量计算可自然扩展为无矩阵的有限体积算子，用于迭代 Krylov 方法以求解方程（2）。结合对稀疏矩阵向量乘法算子的评估，在 Cerebras 架构上实现高性能的无矩阵有限体积算子，将是评估其对有限体积模板适用性的重要一步。

此外，据我们所知，本研究提出的算法是首个利用对角处理单元间数据通信的算法，这使得其他

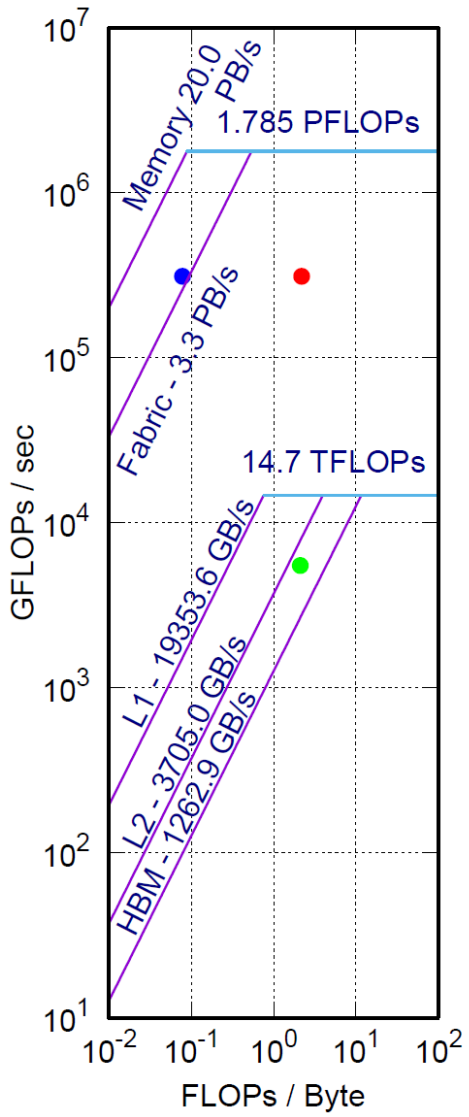


图8. 针对 $750 \times 994 \times 246$ 网格的 CS-2 和 A100 屋顶线模型（对数-对数刻度）。圆点表示有限体积通量计算的实现。对于 CS-2（上图），有两种不同的资源——存储器和互连网络。左侧的蓝点表示内存访问，右侧的红点表示互连网络访问。该内核的内存访问受带宽限制，而互连网络访问受计算限制。A100（下图）以绿色表示。该内核受存储器限制，达到峰值性能的 76%。

类型的应用也得以实现，例如在分块横向各向同性介质上求解声波方程，这类应用同样需要从对角邻居处获取数据。

此外，由于每个处理单元都拥有分布式隔离内存空间，本文讨论的内存优化技术对于逆时迁移工作流等应用至关重要，因为这类应用需要处理大量中间结果。尽管优化过程需要手动完成，本研究仍展示了复用中间结果存储空间的潜力。

9 结论与未来工作

本研究提出了一种新颖的有限体积通量算法实现，利用低延迟的局部通信和单级内存架构。该方法扩展了局部通信模式，可与基数方向和对角方向的处理单元交换数据。我们通过复用中间结果的存储空间，展示了节省存储器的优化方法。我们介绍了通过浮点运算向量化以及将数据移动与计算重叠来加速计算的策略。

我们的实验表明，我们的数据流实现可在单节点 CS-2 机器上达到 311.85 太浮点运算每秒，相较于运行在 NVIDIA A100 GPU 上的参考实现，性能显著提升 204 倍。我们还表明，FV通量计算在数据流架构上也能实现近乎完美的弱扩展性。我们进一步展示了在采用数据流架构的系统上进行 FV通量计算的性能和能耗特性，并证明该架构可高效用于这类吞吐量巨大且能耗节省显著的应用。我们还讨论了本研究的意义。

未来的工作包括支持任意网格拓扑结构，并将其高效映射到数据流架构上，以便移植更广泛的有限体积应用。我们还需要提出数据广播策略，以支持任意形状网格中任意网格单元之间的数据移动。此外，在数据流架构上开发非线性和线性求解器，可以扩展有限体积应用的范围，使其适用于更多常见的现实场景。

致谢

作者感谢 TotalEnergies 和 Cerebras Systems 允许我们分享相关材料。感谢 TotalEnergies 的 Shelton Ma 和 Harry Zong 提供的所有支持。我们还要感谢 Cerebras 的 Leighton Wilson 和 Michael James 为我们牵线并提供指令。

部分工作由美国能源部授权劳伦斯利弗莫尔国家实验室依据合同 DE-AC52-07NA27344 完成。发布编号 LLNL-CONF-847874

参考文献

[1] Dennis Abts、Ibrahim Ahmed、Andrew Bitar、Matt Boyd、John Kim、Garrin Kimmell 和 Andrew Ling。2022 年。《借助 Groq 确定性张量流处理器实现可靠扩展系统的挑战与机遇》。载于《2022 年第 52 届 IEEE/IFIP 国际可靠系统与网络会议——增刊》（DSN-S）。IEEE，19–22 页。https://doi.org/10.1109/DSN-S54099.2022.00016[2] Dennis Abts、John Kim、Garrin Kimmell、Matthew Boyd、Kris Kang、Sahil Parmar、Andrew Ling、Andrew Bitar、Ibrahim Ahmed 和 Jonathan Ross。2022 年。《Groq 软件定义的扩展型张量流多处理器：从芯片到系统的架构概述》。载于《2022 年 IEEE Hot Chips 第 34 届研讨会》（HCS）。1–69 页。https: //doi.org/ 10 1109 55958 2022 9895630 . / HCS . . Mauricio Araya-Polo[3]、Félix Rubio、Raúl de la Cruz、Mauricio Hanzich、José María Cela 和 Daniele Paolo Scarpazza。2009 年。《在同构与异构多核处理器上通过逆时偏移进行三维地震成像》。《科学程序》，17 卷，1–2 期（2009 年 1 月），185–198 页。https://doi.org/ 10 1155 2009 382638 . / / David A. Beckingsale、Jason Burmark[4]、Rich Hornung、Holger Jones、William Killian、Adam J. Kunen、Olga Pearce、Peter Robinson、Brian S. Ryujin 和 Thomas RW Scogland。2019 年。《RAJA：面向大规模科学应用的可移植高性能计算》。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， 71 81 10 1109 3 49587 2019 00012 – 。[5] https://doi.org/ . /P HPC . . Q.M. Bui、F.P. Hamon、N. Castelletto、D. Osei-Kuffuor、R.R. Settgast 和 J.A. White。2021 年。《多孔与裂隙介质耦合过程的多重网格约简预条件框架》。《计算方法与应用力学工程》，387 卷（2021 年），114111。https://doi.org/10.1016/j.cma.2021. 114111[6] J. Camargo、F.P. Hamon、A. Mazuyer、T. Meckel、N. Castelletto 和 J.A. White。2022 年。《墨西哥湾海上碳封存变形监测的可行性》。第 16 届温室气体控制技术会议论文集（GHGT-16），2022 年 10 月。[7] A. Costa、M. Cusini、T. Jin、R.R. Settgast 和 J.E. Dolbow。2022 年。《用于水力压裂模拟的多分辨率方法》。《国际断裂杂志》，237 卷，1 期（2022 年），165–188 页。https://doi.org/10.1007/s10704-022-00662-y [8] M. Cusini、A. Franceschini、L. Gazzola、T. Gazzola、J. Huang、F.P. Hamon、R.R. Settgast、N. Castelletto 和 J.A. White。2022 年。《断层与裂隙天然地层中地质碳封存的现场尺度模拟》。技术报告。劳伦斯利弗莫尔国家实验室（LLNL），美国加利福尼亚州利弗莫尔。[9] Murali Emani、Venkatram Vishwanath、Corey Adams、Michael E. Papka、Rick Stevens、Laura Florescu、Sumti Jairath、William Liu、Tejas Nama 和 Arvind S ujeeth。2021 年。《利用 SambaNova 可重构数据流架构加速科学应用》。《计算科学与工程》，23 卷，2 期（2021 年），114–119 页。https: // doi.org/ 10 1109 2021 3057203 . /MCSE. . J. Huang、Y. Hao、R.R.[10] Settgast、J.A. White、K. Mateen 和 H. Gross。2022 年。《用于模拟支撑剂运移与裂缝导流能力的三维模型的验证与应用》。《岩石力学与岩石工程》（ 2022 1 23 10 1016 2021 103488 ），– 。[11] https://doi.org/ . /j.ijggc. . Mathias Jacquelin、Mauricio Araya-Polo 和 Jie Meng。 。《可扩展的分布式高阶模板计算》。载于《国际高性能计算、网络、存储与分析会议论文集》（得克萨斯州达拉斯）（SC '22 30, 13）。IEEE 出版社，文章页码。[12] B. Kahle 和 W.D. Hillis。1989 年。《连接机模型 CM-1 架构》。Thinking Machines Corporation。https://books.google. com/books?id=PCq7uAAACAAJ [13] 劳伦斯利弗莫尔国家实验室、斯坦福大学、道达尔能源公司和 GEOS 贡献者。2023 年。《GEOS：用于地质碳封存的新一代模拟》。网站。检索日期：2023 年 3 月 19 日，网址：https://www.geos.dev/

[14] Knut-Andreas Lie。2019。使用 MATLAB/GNU Octave 进行储层模拟简介：MATLAB 储层模拟工具箱（MRST）用户指南。剑桥大学出版社。[15] Thorben Louw 和 Simon McIntosh-Smith。2021。使用 Graph-core IPU 处理传统 HPC 应用。在第 3 届加速机器学习研讨会（AccML）上，与 HIPEAC21 同期举行。https: //workshops.inf.ed.ac.uk/accml [16] Raghu Prabhakar 和 Sumti Jairath。2021。SambaNova SN10 RDU：借助数据流加速 Software 2.0。在 2021 年 IEEE Hot Chips 第 33 届研讨会（HCS）上。IEEE，1–37。https://doi.org/10.1109/HCS52781. 2021.9567250[17] Daniel Reed、Dennis Gannon 和 Jack Dongarra。2022。重塑高性能计算：挑战与机遇。arXiv:2203.02544 [cs.DC][18] Kamil Rocki、Dirk Van Essendelft、Ilya Sharapov、Robert Schreiber、Michael Morrison、Vladimir Kibardin、Andrey Portnoy、Jean Francois Dietiker、Madhava Syamlal 和 Michael James。2020。在晶圆级处理器上快速进行模板代码计算。在国际高性能计算、网络通信、存储与分析会议论文集（佐治亚州亚特兰大）（SC '20 58, 14）中。IEEE Press，文章页码。[19] Samuel Williams、Andrew Waterman 和 David Patterson。2009。Roofline：多核架构的直观性能模型。《ACM 通讯》52，4（2009 年 4 月），65–76。https://doi.org/ 10.1145/1498765.1498785[20] Mino Woo、Terry Jordan、Robert Schreiber、Ilya Sharapov、Shaheer Muhammad、Abhishek Koneru、Michael James 和 Dirk Van Essendelft。2022。场方程建模的颠覆性变化：晶圆级引擎的简单接口。arXiv: 2209 13768 [ ]. cs.DC Charlene Yang[21]。 2021。《经验 Roofline 工具包》。网址：https://bitbucket. org/berkeleylab/cs-roofline-toolkit。[22] Charlene Yang、Thorsten Kurth 和 Samuel Williams。2020。面向 GPU 的分层 Roofline 分析：加速 NERSC-9 Perlmutter 系统的性能优化。《并发与计算：实践与经验》32，20（2020），e5547。https://doi.org/10.1002/cpe. 5547 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5547 e5547 cpe.5547。