PROCEEDINGS OF SCIENCE

QUDA 中的双链交错夸克涂抹

Steven Gottlieb,[a] **Hwancheol Jeong**[a,*] **和 Alexei Strelchenko**[b]

[a]印第安纳大学物理系，美国印第安纳州布卢明顿 *47405*[b]费米国家加速器实验室科学计算部，美国伊利诺伊州巴达维亚 *60510*

电子邮箱：sg@indiana.edu，sonchac@gmail.com，astrel@fnal.gov

基于三维晶格拉普拉斯算子的规范协变涂抹可用于构造扩展算符，使其与强子基态有更好的重叠。对于交错夸克，我们采用两链平行输运来保持味性质。我们已在 QUDA 中实现这一过程。我们展示了该代码在印第安纳大学 Big Red 200 超级计算机上的 NVIDIA A100 GPU，以及橡树岭领导力计算设施（OLCF）的 Crusher 上 AMD MI250X GPU 的性能，并讨论其可扩展性。我们还研究了在 OLCF 的 Summit 上使用 NVSHMEM 带来的性能提升。通过为所有源与汇重复使用预先计算的两链乘积，与原始 MILC代码相比，该方法将重子关联函数测量的总涂抹时间缩短了 100–120 倍，并将总耗时减少了 60–70%。

第*39*届国际格点场论研讨会（*Lattice2022*），*2022*年*8*月*8*日至*13*日，德国波恩

*报告人

© 作者依据知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议（CC BY-NC-ND 4.0）享有版权。

arXiv:2301.05518v1 [hep-lat] 13 Jan 2023

1. 引言

格点量子色动力学计算需要与特定强子态有较强重叠的算符。例如，研究低能强子谱的格点量子色动力学计算，可采用扩展算符；与单个格点上的局域算符相比，这些扩展算符与基态的重叠更好。将关联函数分解为能量本征态的形式，二点关联函数 $C(t) = \sum_{\mathbf{x}} \left\langle \mathrm{O}\,(t,\ \mathbf{x})\,\mathrm{O}^{\dagger}(0,\ \mathbf{0}) \right\rangle$ 可表示为

$$C(t) = \sum_{n} |\,\langle 0|O|n\rangle|^2\, e^{-E_n t}\,, \tag{1}$$

其中，E_n 是第 n 个能量本征态的能量。我们可以从中提取一些低能性质，包括基态贡献所对应的质量。如果规范场构型经过规范固定，就可以使用扩展算符而无需考虑平行输运；然而，通过对源进行规范协变涂抹，可以避免规范固定。

规范协变涂抹有两种常见方法：雅可比涂抹和高斯涂抹。雅可比涂抹是乌珀塔尔涂抹的一种迭代形式，它将三维标量传播子作为涂抹源 [1–4]。在自由规范场构型上，涂抹源遵循指数分布。另一种方法是对给定源迭代应用跃迁算符进行高斯涂抹。在自由规范场构型上，所得涂抹源遵循高斯分布 [2, 3, 5]。

在本文中，我们关注高斯涂抹的一种变体：以交错夸克的三维格点拉普拉斯算符取代跃迁算符 [6]。为保持交错夸克的味对称性，拉普拉斯算符应延伸至次近邻格点。我们定义连接次近邻格点的两链乘积，并将这种涂抹方法称为双链交错夸克涂抹。

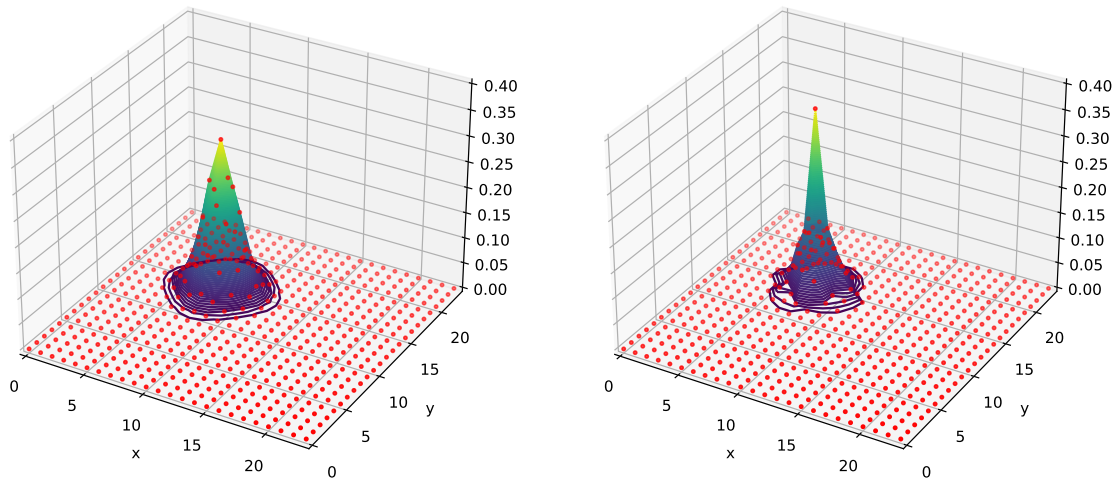
我们最初使用的 MILC代码通过在同一方向上两次应用单格点平行输运来实现两格点平行输运。每个方向上，这需要两次通信和两次矩阵-向量乘法。我们发现，在 CPU 上执行这种涂抹耗时过长。当计算的其他部分在 GPU 上运行时，情况会更糟，因为每个 GPU 只分配一个 MPI 进程，因此需要使用 OpenMP 进行多线程处理，才能让每个进程使用多个 CPU 核心。因此，我们已在 QUDA 中实现了这一流程 [7–9]。美国的百亿亿次级计算机都采用 GPU，因此，今后我们将有越来越多的项目使用这一及时加入代码库的新功能。

第 2 节简要介绍双链交错夸克涂抹。第 3 节介绍我们的 GPU 实现和算法改进。在第 4 节和 5节中，我们分别给出在一些（较新或最新的）英伟达和超威半导体 GPU 上的基准测试结果。我们还在第 6 节将 QUDA 涂抹程序应用于重子关联函数测量，以展示我们的代码在实际生产任务中的表现。我们在第 7节总结结论。

2. 双链交错夸克涂抹

考虑由 H 定义的跃迁算符

$$H(x,y) = \sum_{\mu=1}^{3} U_{\mu}(x)\,\delta_{x+\hat{\mu},\,y} + U_{\mu}^{\dagger}(x-\hat{\mu})\,\delta_{x-\hat{\mu},\,y}\,, \tag{2}$$



(a) 自由规范场（$U = 1$）　　**(b)** HISQ 规范场

图1： 在自由规范场构型（左）和 HISQ规范场构型（右）上，使用双链交错夸克涂抹对点源进行涂抹后得到的示例分布。红色数据点表示位于（x，y，z，t）=（x，y，0，0）处的涂抹后的夸克场的范数。通过连接这些点绘制等高线。

其中 x、y 是时空坐标，$U_\mu(x)$ 是规范场。对夸克场 $\psi(y)$ 进行迭代高斯涂抹，可写为

$$\widetilde{\psi} = C(1+\alpha H)^n \psi\,, \tag{3}$$

其中 C，$\alpha \in \mathbb{R}$ 和 $n \in \mathbb{Z}$ 是调节参数。对于单位规范场构型 U_μ（x）$= 1$，~涂抹后的夸克场 ψ（x）会随着迭代次数 n 增加 [2, 3, 5]而趋近于高斯分布。

三维晶格拉普拉斯算子 ∇^2 定义为

$$\begin{aligned}\nabla^2(x,y) &= \sum_{\mu=1}^{3}\left[U_\mu(x)\,\delta_{x+\hat{\mu},\,y} + U_\mu^\dagger(x-\hat{\mu})\,\delta_{x-\hat{\mu},\,y}\right] - 6\,\delta_{x,y} \\ &= H(x,y) - 6\,\delta_{x,y}\,. \end{aligned}\tag{4}$$

将 $U_\mu(x)$ 替换为两链乘积 $V_\mu(x) \equiv U_\mu(x)U_\mu(x+\hat{\mu})$，并调整坐标，我们定义（忽略因子 a）两链三维晶格拉普拉斯算子 ∇^2_{two}:

$$4\nabla^2_{\text{two}}(x,y) \equiv \sum_{\mu=1}^{3}\left[V_\mu(x)\,\delta_{x+2\hat{\mu},\,y} + V_\mu^\dagger(x-2\hat{\mu})\,\delta_{x-2\hat{\mu},\,y}\right] - 6\,\delta_{x,y} \tag{5}$$

$$= \sum_{\mu=1}^{3}\left[U_\mu(x)\,U_\mu(x+\hat{\mu})\,\delta_{x+2\hat{\mu},\,y} + U_\mu^\dagger(x-\hat{\mu})\,U_\mu^\dagger(x-2\hat{\mu})\,\delta_{x-2\hat{\mu},\,y}\right] - 6\,\delta_{x,y}\,. \tag{6}$$

请注意，∇^2_{two} 保留了交错夸克的味性质。

如果我们将方程（3）改写为关于拉普拉斯算子 ∇^2,

$$\widetilde{\psi} = C(1+\alpha(\nabla^2+6))^n\psi \equiv C'(1+\alpha'\nabla^2)^n\psi\,, \tag{7}$$

其中 $C' \equiv C(1+6\alpha)^n$ 且 $\alpha' \equiv \frac{\alpha}{1+6\alpha}$。现在，我们将双链交错夸克涂抹定义为

$$\widetilde{\psi} = \left(1 + \frac{\sigma}{n}\nabla^2_{\text{two}}\right)^n \psi\,, \tag{8}$$

其中 σ 和 n 是调节参数。这是一种针对交错夸克的味保持规范协变涂抹。图 1展示了点源分别在自由规范场构型（图1a）和 HISQ 规范场构型（图 1b）上应用该涂抹后的分布。尽管后者因规范场的存在而发生了畸变，但这些结果表明，选择合适的调节参数 σ 和 n [2, 4, 10]，可以使我们与强子基态获得更好的重叠。用规范场涂抹链路替代 U_μ，既能减轻畸变，也能提高与基态的重叠 [11]。此外，还有其他能够提高重叠、且具有良好性质的方法 [5, 12]。

3. GPU实现

在本项目启动之前，MILC代码库 [13] 提供了一个基于 CPU 的双链交错夸克涂抹例程。由于需要相应的 GPU 版本，我们已在 QUDA 中实现该例程。我们还在 MILC代码中为这一 QUDA 涂抹添加了接口，用于运行本文中的所有 QUDA 基准测试。[1]

我们还改进了 MILC代码中的 CPU算法。我们没有在每次涂抹迭代中执行两次连续的平行输运（如式（6）所示），而是在涂抹前预先计算两链乘积 V_μ，并在每次迭代中仅从内存中加载该乘积，然后使用式（5）。对于不同的源或汇，这一两链乘积甚至可以重复使用。在 GPU算法中，我们进行涂抹时所需的内存流量显著减少，浮点运算次数更少，MPI进程之间的通信也更少。

在图 2中，我们比较了印第安纳大学 Big Red 200 上，分别使用全部 CPU 和 GPU 时，MILC代码与我们的 QUDA代码所需的涂抹时间。Big Red 200 与 NERSC 的 Perlmutter 类似：它设有 CPU 分区，每个节点包含两颗 AMD 64 核 EPYC 7742 处理器；另设有 GPU分区，每个节点包含一颗 AMD 处理器和四块 NVIDIA A100 GPU。Big Red 200 使用 Slingshot 10 网络，而 Perlmutter 的网络已从 Slingshot 10 升级到 11。我们在两个 CPU 节点或两个 GPU 节点上，对四种不同的晶格体积进行了基准测试。对于生产任务，我们希望每块 GPU 的局部体积至少达到 24^4，而 32^4 或更大则更理想。因此，这些使用固定节点数进行的测试并非以最高效率为目标。对于 QUDA 运行，我们还测量了第二次源涂抹的时间，该过程会重复使用预先计算好的两链乘积。第一次源涂抹包括两链乘积的计算以及 n 次涂抹迭代，而第二次源涂抹只进行 n 次涂抹迭代。典型的计算会在同一规范场构型上使用多个源与汇，因此第二次源涂抹的结果颇具参考价值。结果表明，QUDA 在 GPU 上进行第一次源涂抹所需的时间为 MILC代码在 CPU 上涂抹时间的 0.79 到 0.15 倍，而 QUDA 进行第二次源涂抹所需的时间为其 0.16

[1] 我们正在努力将 QUDA代码和 MILC代码接口分别合并到 QUDA 和 MILC代码 GitHub 仓库的主（develop）分支中[7, 13]。

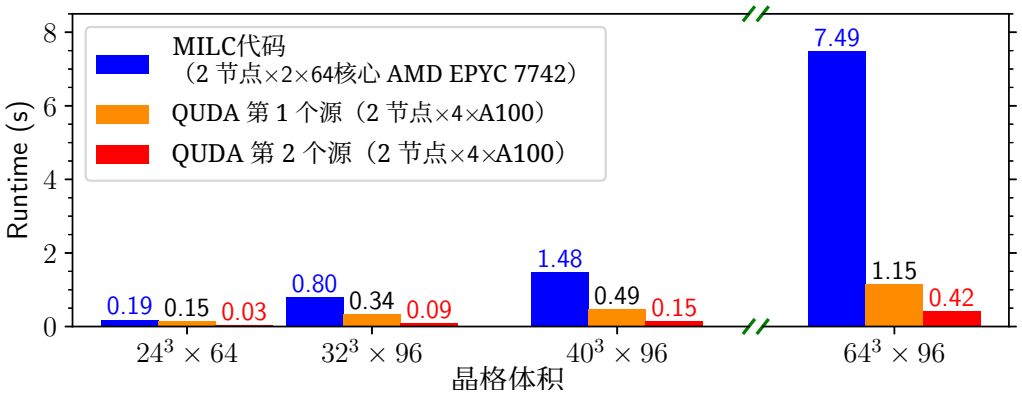


图2： 使用 MILC 代码和 QUDA 进行双链交错夸克涂抹，对源夸克场进行涂抹所需的时间。测量是在 Big Red 200 的两个节点上进行的。MILC 代码以每个 CPU 核心一个 MPI 进程运行（双精度浮点运算次数峰值总计为 ≈ 14 太拉浮点运算次数/秒），QUDA 以每个 GPU 一个 MPI 进程运行（双精度浮点运算次数峰值总计为 ≈ 80 太拉浮点运算次数/秒）。MILC 代码运行时，格点按 $(x, y, z, t) = (8, 8, 4, 1)$ 划分；QUDA 运行时，格点按 $(x, y, z, t) = (2, 2, 2, 1)$ 划分。请注意，沿 t维划分并不能提升应用于固定时间源（如点源或墙源）的三维高斯涂抹性能。

到 0.06 倍。晶格体积越大，性能提升越明显，因为较小的测试规模不足以让 GPU代码高效运行。

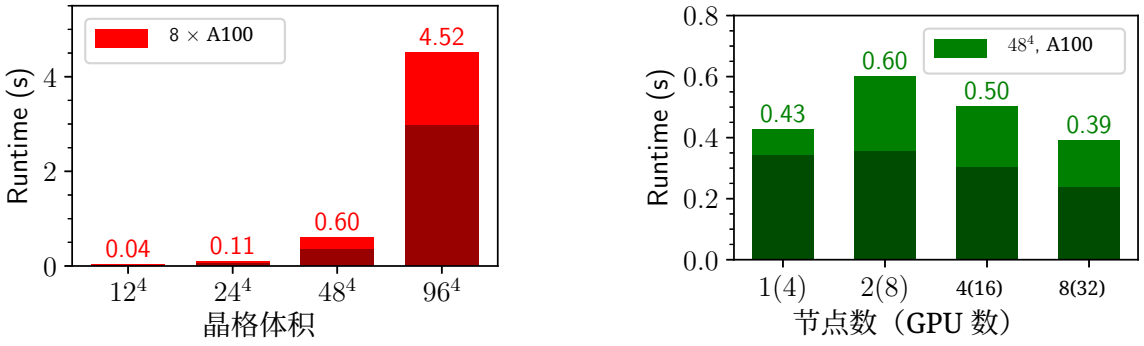
本节从充分利用所有可用 CPU/GPU 资源的角度，比较了在一台同时配备 CPU 和 GPU 节点的计算机的两个节点上，MILC代码涂抹与 QUDA 涂抹的最高性能。不过，实际应用中，我们的主要目标是在 GPU 节点上运行的测量中实现 QUDA 涂抹。在这种情况下，在 CPU 上运行的涂抹可能会成为严重的瓶颈。第 6 节讨论了一个示例。

4. 英伟达 GPU 上的性能

本节报告 QUDA 中双链交错夸克涂抹（QUDA 双链涂抹）在两个基于 NVIDIA GPU 的系统上的性能。我们依次对三种不同的颜色墙源进行涂抹。只有第一个颜色源涂抹会计算两链乘积，其余涂抹则重复使用该乘积。

在图 3中，我们测量了 Big Red 200 上 NVIDIA A100 GPU 的 QUDA 双链涂抹性能和可扩展性。总运行时间包括一次双链计算（未着色部分）和三次 $n = 50$涂抹迭代（着色部分）。我们发现，单次双链计算耗时约占10–40%。这表明重复使用两链乘积具有优势。

图 3a 展示了格点体积变化时的涂抹时间。格点体积按几何级数扩大 16 倍时，总涂抹时间分别扩大了2.75、5.45和 7.53倍。这表明，即使格点体积达到最大值，计算仍未达到饱和，因此在更大的格点上，其性能（每秒浮点运算次数）会更好。不过，对于较小的格点，其耗时仅占典型格点模拟规模下耗时的一小部分。图 3b展示了节点数和 GPU 数量变化时的涂抹时间。请注意，两节点运行



(a)体积可扩展性 （**b**）强可扩展性

图 3： 在 Big Red 200 中，使用英伟达 A100 GPU 执行 QUDA 双链涂抹，对三个色源进行涂抹所需的时间。未着色区域表示双链计算所需的时间，着色区域表示进行 $3 \times n$ 次涂抹迭代所需的时间。格点按 $(x,\ y,\ z,\ t) = (2,\ 2,\ 1,\ 1)$ 划分（使用 4 个 GPU 时），按 $(2,\ 2,\ 2,\ 1)$ 划分（使用 8 个 GPU 时），按 $(4,\ 2,\ 2,\ 1)$ 划分（使用 16 个 GPU 时），按 $(4,\ 4,\ 2,\ 1)$ 划分（使用 32 个 GPU 时）。

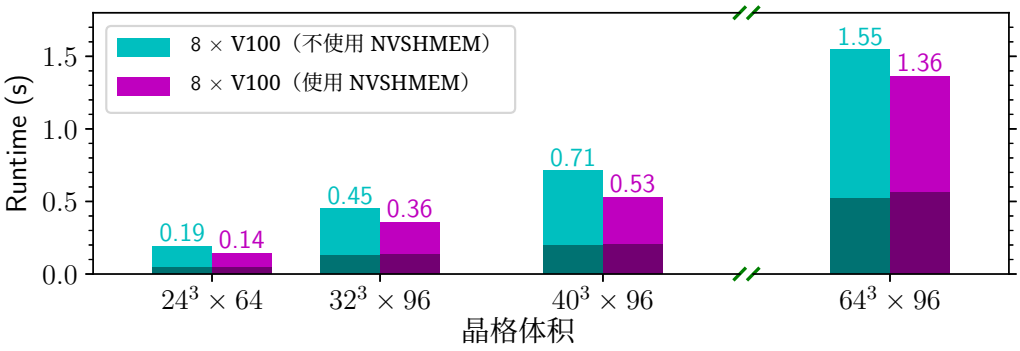
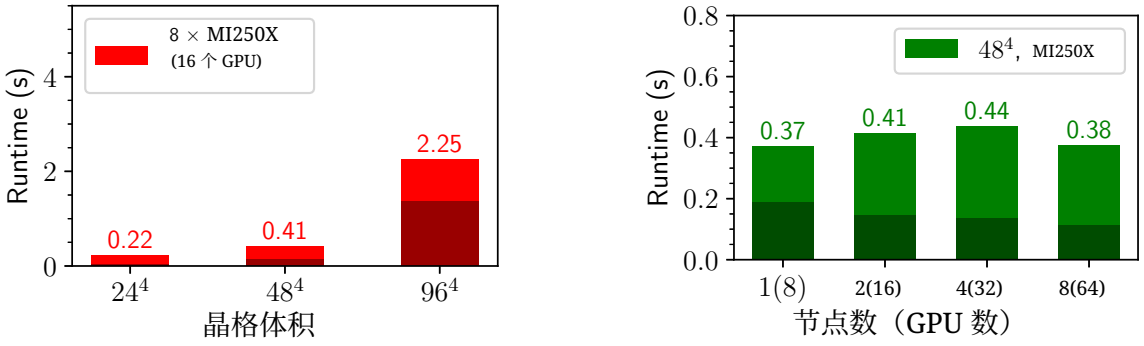


图 4： 在 Summit 中启用或未启用 QUDA 的 NVSHMEM 支持时，使用 QUDA 双链涂抹在英伟达 V100 GPU 上对三个色源进行涂抹所需的时间。未着色（着色）区域表示双链计算（涂抹迭代）所需的时间。这里使用两个 Summit 节点，但每个节点的六个 V100 GPU 中只使用四个，因为六个 GPU 不适合用于划分某些代表性格点的空间维度。所有格点均按 $(x,\ y,\ z,\ t) = (2,\ 2,\ 2,\ 1)$ 划分。

比单节点运行更慢。即使双链计算也要多花三倍时间。这说明跨节点通信会显著影响性能，且对双链计算的影响更为明显。排除单节点结果后，随着节点数增加，性能有所提升，但扩展性很差。图 3a 和 3b 表明，该例程属于通信密集型计算。

上述观察表明，在更快速的通信环境下，QUDA 双链涂抹的性能可能更好。橡树岭领导力计算设施的 Summit 支持英伟达 NVSHMEM 技术。NVSHMEM 通过实现共享内存空间中的 GPU 之间直接通信，提升 GPU 操作的强扩展性 [14]。QUDA 支持 NVSHMEM [15]。图4 展示了启用 NVSHMEM 后，QUDA 双链涂抹在 Summit 上的性能提升。在这里，双链计算耗时超过总运行时间的一半。我们发现，NVSHMEM 将双链计算时间缩短了约30–50%。



(a)体积可扩展性 **(b)**强可扩展性

图5： 在 Crusher 上使用 $n = 50$ QUDA 双链涂抹，涂抹三个色源所需的时间。请注意，每个 MI250X 包含两个图形计算芯片，因此实际使用的 GPU 数量是 MI250X 数量的两倍。未着色（着色）区域表示双链计算（涂抹迭代）所需的时间。格点划分为 $(x, y, z, t) = (2, 2, 2, 1)$，对应 8 个 GPU；$(4, 2, 2, 1)$，对应 16 个 GPU；$(4, 4, 2, 1)$，对应 32 个 GPU；以及 $(4, 4, 4, 1)$，对应 64 个 GPU。

	Big Red 200	Crusher
GPU	4 ×NVIDIA A100 （≈ 40 太拉浮点运算次数/秒）	4 × AMD MI250X（=8GPU） （≈ 210 太拉浮点运算次数/秒）
GPU 内存带宽	1555 GB/s	3277 GB/s
NIC	2 慧与 × Slingshot-10 (200 Gbps)	4 × HPE Slingshot-11 (800 Gbps)

表 1： Big Red 200 和 Crusher 的 GPU 与网络接口卡规格 [17–20]。每秒浮点运算次数的数据表示双精度峰值性能。

5. 超威半导体 GPU 的性能

QUDA 还支持 AMD ROCm 平台上的 HIP [16], ，因此可以在 AMD 图形处理器上运行。本节报告 QUDA 双链涂抹在橡树岭领导力计算设施的 Crusher 上的性能。Crusher 是一套基于 AMD 图形处理器的系统，其硬件与 Frontier 上的硬件相同。与第 4 节一样，我们依次涂抹三个不同的颜色墙源，其中两链乘积仅在第一次涂抹时计算，并在后续涂抹中重复使用。

在图 5 中，我们采用与图 3 相同的方法，测量 Crusher 中 AMD MI250X GPU 上 QUDA 双链涂抹的性能和可扩展性。[2] 图 5a 和 5b 分别绘出了体积可扩展性和强可扩展性。为便于比较，纵轴范围与图 3a 和 3b 相同。与 A100 相比，使用 MI250X 时涂抹迭代的速度约快一倍。这一结果符合我们的预期：该计算的瓶颈在于 GPU内存带宽，因为其计算通信比与典型的 dslash例程相近（见公式（

[2]这些图表是根据我们在海报中展示的图表更新的。当时我们观察到超威半导体 GPU 上的双链计算出现了异常减速。后来发现，问题源于一个与双链计算没有直接关系的 HIP API。我们找到了避免这一问题的方法，并重新运行了基准测试。目前我们也在努力解决这个问题。

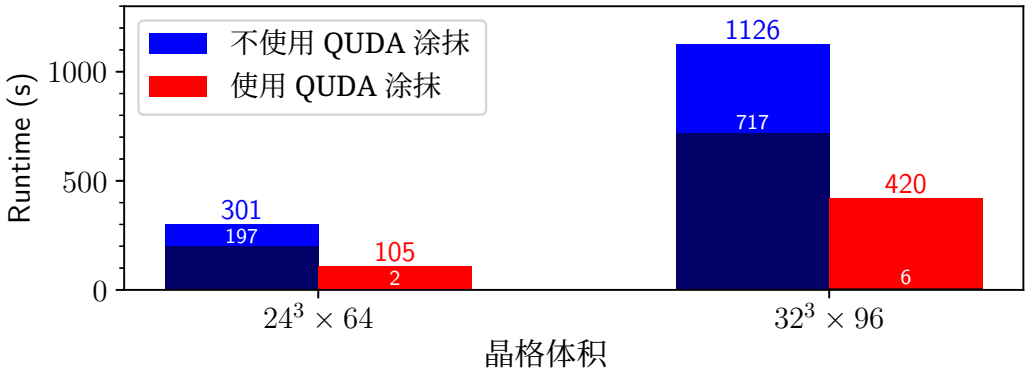


图6： 采用 $n = 30$QUDA 双链涂抹对 72 个源/汇进行涂抹的重子关联函数测量所需的总时间。该测量在 Big Red 200 的 GPU 分区中的两个节点上进行，每个节点使用四个 CPU 核心和 GPU。无论是否启用 QUDA 双链涂抹，所有其他受 QUDA 支持的计算均在 GPU 上执行。阴影区域表示涂抹所用的总时间。

5））， 而 MI250X 的内存带宽是 A100 的两倍（见表 1）。另一方面，MI250X 上的双链计算性能与 A100 相近或更低。唯一的例外是 96^4 格点结果，在该结果中我们观察到预期的两倍性能提升。关于可扩展性，我们观察到扩展性较差，这与在 A100 GPU 上的观察结果一致。

6. 应用：重子关联函数测量

我们的重子关联函数计算包含许多源与汇。图6 展示了一项重子关联函数测量示例所需的总运行时间。计算的大部分环节此前已在 QUDA 中实现。禁用 QUDA 涂抹时，涂抹由 CPU 上的 MILC代码涂抹例程执行，而其他主要计算则由 GPU 上的 QUDA 执行。请注意，这里使用的 CPU 数量与 GPU 数量相同。结果显示，在 CPU 上运行时，源/汇涂抹约占总测量时间的60–70%。然而，启用 QUDA 涂抹后，涂抹时间缩短了100–120倍，仅占总测量时间的约1–2%。因此，作业总耗时降至原来的约 30–40%。

7. 结论

我们已在 QUDA 中实现双链交错夸克涂抹。在八块 NVIDIA A100 GPU 上运行时，其速度最高可达 MILC代码的六倍；后者运行于四颗 AMD EPYC 7742 CPU 的共 256 个核心上。在同一规范场上为另一个源重复使用预计算的两链乘积，可将这一性能差距扩大至 18 倍。

该例程在 NVIDIA A100 和 AMD MI250X 上的扩展性都相当差，尤其是双链计算部分。NVSHMEM 可以提升

双链计算的性能 30–50%。对于重子关联函数测量，所有 72 个源/汇重复使用两链乘积，可将涂抹时间占总测量时间的比例从 60–70% 降至 1–2%。因此，即使不进一步优化，这段代码对于需要对源或汇进行大量涂抹的 GPU 作业也很有用。

致谢

本研究由百亿亿次计算项目（17-SC-20-SC）提供支持。该项目由美国能源部科学办公室和国家核安全管理局合作开展。我们衷心感谢美国能源部科学办公室通过项目编号 DE-SC0010120 提供的支持。本研究还部分得到了 Lilly Endowment, Inc. 通过支持印第安纳大学普适技术研究所提供的资助。本研究使用了橡树岭国家实验室橡树岭领导力计算设施的资源。该设施由美国能源部科学办公室根据合同编号 DE-AC05-00OR22725 提供支持。我们感谢 QUDA [8, 9]开发人员，其姓名可在其网站上找到[7] 。

参考文献

[1] S. Gusken、U. Low、K.H. Mutter、R. Sommer、A. Patel 和 K. Schilling, Phys. Lett. B 227 (1989) 266。[2] S. Gusken，Nucl. Phys. B Proc. Suppl. 17 (1990) 361。[3] C. Alexandrou、F. Jegerlehner、S. Gusken、K. Schilling 和 R. Sommer, Phys. Lett. B 256 (1991) 60。[4] UKQCD合作组，Phys. Rev. D 47 (1993) 5128 [hep-lat/9303009]。[5] G.M. von Hippel、B. Jäger、T.D. Rae 和 H. Wittig，JHEP 09 (2013) 014 [1306。1440]。[6] 强子谱合作组，Phys. Rev. D 80 (2009) 054506 [0905。2160]。[7] https://github.com/lattice/quda。[8] M.A. Clark、R. Babich、K. Barros、R.C. Brower 和 C. Rebbi，Comput. Phys. Commun. 181 (2010) 1517 [0911。3191]。[9] R. Babich、M.A. Clark、B. Joo、G. Shi、R.C. Brower 和 S. Gottlieb，2011年9月，DOI[1109。2935]。[10] T.A. DeGrand 和 R.D. Loft，Comput. Phys. Commun. 65 (1991) 84。[11] S.N. Syritsyn 等，Phys. Rev. D 81 (2010) 034507 [0907。4194]。[12] G.S. Bali、B. Lang、B.U. Musch 和 A. Schäfer，Phys. Rev. D 93 (2016) 094515[1602。05525]。

[13] https://github.com/milc-qcd/milc_qcd/tree/develop。[14] https://developer.nvidia.com/nvshmem。[15]
https://github.com/lattice/quda/wiki/Multi-GPU-with-NVSHMEM。[16]
https://github.com/lattice/quda/wiki/Building-QUDA-With-HIP。[17] https://kb.iu.edu/d/brcc。[18]
h
t
t
p
s://
w
w
w.
n
vidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf。
[19] https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html。[20]
https://www.amd.com/en/products/server-accelerators/instinct-mi250x。