极大规模多对多蛋白质相似性搜索

Oguz Selvitopi*、Saliya Ekanayake†、Giulia Guidi‡、Muaaz G. Awan§、Georgios A. Pavlopoulos¶、Ariful Azad‖、Nikos Kyrpides**、Leonid Oliker*、Katherine Yelick‡*、Aydın Buluç*‡

*劳伦斯伯克利国家实验室应用数学与计算研究部，美国
†微软公司，美国
‡加州大学伯克利分校，美国§*NERSC*，劳伦斯伯克利国家实验室，美国
¶基础生物医学研究所，*BSRC*“亚历山大·弗莱明”，弗莱明街*34*号，邮编*16672*，希腊瓦里‖印第安纳大学，美国

**美国劳伦斯伯克利国家实验室联合基因组研究所roselvitopi@lbl.gov

arXiv:2303.01845v1 [cs.DC] 3 Mar 2023

摘要—**相似性搜索是不断增长的蛋白质数据集中经常执行的最基本计算之一。对于揭示超大规模下发生的新现象，可扩展性至关重要。我们利用 Summit 系统上超过 20,000 个 GPU 的强大算力，在不到 3.5 小时内，对包含 4.05 亿个蛋白质的最大公开可用数据集之一执行全对全蛋白质相似性搜索，将许多应用场景的求解时间从数周缩短至数小时。蛋白质序列长度的差异，以及成对比较空间的稀疏性，使其成为分布式内存环境中的一项挑战性问题。由于需要构建和维护一个保存指向所有其他序列的索引的数据结构，该应用具有巨大的内存占用，使问题规模难以扩展。我们通过创新的基于矩阵的分块技术克服了这一内存限制，同时没有引入额外的负载不均衡。**

一、ACM戈登·贝尔奖申报理由

我们充分利用超过 20,000 个 GPU 的强大算力，在 3.4 小时内，对一个包含 4.05 亿条蛋白质的最大型公开可用数据集之一进行多对多蛋白质相似性搜索，实现了前所未有的速度：每秒 6.91 亿次比对，将许多应用场景的求解时间从数周缩短至数小时。

二、性能属性

性能属性	**数值**
成就类别	求解时间、每秒比对次数， 每秒单元更新次数 (每秒单元更新次数)
所用方法类型	N/A
结果报告依据	整个应用程序的求解时间 和每秒比对次数。 每秒单元更新次数的内核时间
报告的精度	整数
系统规模	3364 个节点 （141,288 个 CPU 核心和 20,184 个 GPU）
测量机制	计时器

III. 问题概述

比较基因组学通过利用基因组序列之间的相似性，研究不同生物体之间的进化和生物学关系。例如，一项常见任务是确定从环境中采集的样本的功能或分类学内容，通常通过将采集到的序列与已建立的参考数据库进行搜索来完成。随着越来越多的基因组被测序，为比较基因组学构建并提供快速计算基础设施变得愈发重要。

我们的工作旨在解决在序列领域中搜索两组蛋白质之间相似性所面临的计算挑战。这项任务在计算生物学中有许多应用场景，包括功能注释 [1], 基因定位以及蛋白质进化研究 [2]。在宏基因组学中，从环境中采集的DNA序列使研究人员能够研究丰富多样的微生物基因组库，而这些基因组库往往无法通过基于培养的方法发现。此类样本包含数百万条蛋白质序列 [3] ，而许多生物学工作流程的一个主要环节是将这些序列与参考数据库进行对齐，从而找出已知基因。随着测序成本不断下降、技术日益普及，宏基因组学研究中的瓶颈正逐渐转向计算和存储 [4], [5]。

我们关注的是将一组序列与另一组序列进行对齐的问题。这一问题常见于这样的场景：利用一组功能已知的序列（参考序列集），识别另一组序列（查询序列集）中的序列。另一种场景是通过聚类，在给定的序列集中寻找相似序列。在这种变体中，会对一组序列执行多对多搜索，以找出其中的相似序列（通常随后会对序列进行聚类）。这一变体也可以看作是将给定集合与其自身进行对齐，其中查询集和参考集

相同。我们将其称为多对多蛋白质相似性搜索，并在本研究中重点关注这一搜索问题。

我们的工作表明，高性能计算是推动生命之树尺度宏基因组学研究的一种可行且快速的替代方案。我们利用适合 SIMD 型并行的加速器，而搜索中的比对操作正需要这种并行方式；据此，我们开发了新颖的并行算法和优化技术，能够同时利用节点上的所有资源并实现高性能。我们应对上述挑战的方法要点可概括如下：

- 大规模搜索操作所需的海量计算资源，通过分布式利用加速器得以满足。计算密集型比对操作构成了主要的计算瓶颈，而该领域的常用工具 [6]–[8]利用中央处理器上的向量指令实现 SIMD 并行，但并未利用更适合此类操作的加速器。

- 我们利用节点的异构架构，在加速器执行比对操作的同时，由中央处理器执行搜索中内存受限的分布式重叠检测组件，从而掩盖其开销。

- 为避免在搜索过程中进行输入输出——当搜索规模开始变得难以处理时，输入输出往往是该领域分布式工具的首选方案——我们开发了一种分布式二维分块稀疏 SUMMA 算法，可逐步执行搜索，从而有效控制整个搜索所需的最大内存量。这样，我们的方法只在开始和结束时进行输入输出，且这两个阶段都以并行方式完成，耗时最多只占整个搜索时间的 3%。

- 通过采用定制的负载均衡技术，并以分布式稀疏矩阵作为基础数据结构（其并行性能已在数值线性代数领域得到充分研究），我们实现了良好的可扩展性：强扩展并行效率超过 75%，弱扩展并行效率达到 80%。

据我们所知，2021年，DIAMOND [6]在超级计算机系统上完成了当时规模最大的已报告蛋白质序列相似性搜索。该搜索在马克斯·普朗克学会的 Cobra 超级计算机上使用了 520 个节点，将 2.81 亿条序列与 3900 万条序列进行搜索；通过在高灵敏模式下执行总计 230 亿次成对比对（每秒 120 万次比对），耗时 5.42 小时。借助我们的搜索工具 PASITS（基于稀疏矩阵的蛋白质比对），我们显著提升了这一成绩：在橡树岭领导力计算设施的 Summit 超级计算机上使用 3364 个计算节点，将 4.05 亿条序列与 4.05 亿条序列进行搜索。我们的搜索耗时 3.44 小时，以每秒 6.906 亿次比对的速度执行了总计 8.6 万亿次成对比对。总体而言，我们将已解决问题的规模提升了一个数量级（15.0 倍），并将每秒已执行的比对次数提高了两个数量级以上（575.5 倍）。

IV. 当前技术水平

文献中有许多蛋白质相似性搜索工具，各自采用不同的搜索技术，并且这些技术多年来不断得到改进。其中较受欢迎的工具包括 BLASTP [9], 、MMSeqs2 [7], 、LAST [10], 、DIAMOND [11],和 USearch [12]。在并行性方面，几乎所有上述工具都支持不同程度的并行处理，但效率各不相同。DIAMOND、LAST 和 MMSeqs2 等库对节点内并行提供了良好支持：它们可以利用向量指令、使用多个核心，并采用缓存友好算法等。其中一些工具，如 DIAMOND、MMSeqs2 或 mpiBLAST [13]，也能在分布式环境中运行。这些工具在分布式内存并行化方面的一些主要不足可概括如下：

- 在 LAST 和 MMSeqs2 中，搜索阶段开始前，每个计算节点都会复制至少一组序列（查询序列或目标序列）的索引数据结构，这限制了可处理问题的最大规模。在 DIAMOND 中，这些数据结构会以分区块的形式写入磁盘，从而大幅增加文件系统的负担。
- 现有软件无法全局查看这些副本/数据块，且参数按副本设置，因此并行度提高或内存受限时，灵敏度会发生变化。例如，DIAMOND 指南指出：“此 [块大小] 参数会影响算法，使用不同的块大小时，结果不会完全相同。”相比之下，无论采用多少并行度或选择何种分块大小，PASTIS 算法都会给出相同的结果。

- 这些搜索工具不支持 GPU。GPU 具有更高水平的 SIMD 并行性，非常适合搜索中通常使用的成对比对，而这类比对能从这种并行性中显著受益。

在支持分布式内存的蛋白质搜索工具中，我们进一步考察 MMSeqs2 和 DIAMOND，因为这两种工具目前代表了分布式蛋白质相似性搜索领域的最先进水平。

MMSeqs2 [7]采用混合MPI/OpenMP实现分布式内存并行，并支持 SSE 和 AVX2 向量指令集。根据分布的是参考序列集还是查询序列集，该工具提供两种并行模式。第一种模式中，参考序列集被划分为数据块并分发到各个并行节点。此模式下，每个进程都使用其参考序列块搜索所有查询序列。第二种模式中，查询集被划分为数据块并分发到各个并行节点。此模式下，每个进程都使用自己的查询数据块搜索所有参考序列。我们早期关于基于CPU的PASTIS的研究发现，我们的方法比 MMSeqs2 [14] 更具可扩展性，而 MMSeqs2 则受到高输入输出开销的影响。

DIAMOND 的分布式内存并行主要旨在提供在商用集群（即云计算环境）上运行的能力。为此，它避免使用

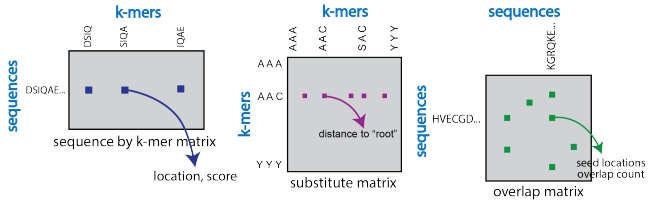


图 1：PASTIS 中使用的各种稀疏矩阵示例。每个矩阵中元素的类型各不相同，稀疏矩阵可以根据所提供的选项（例如比对类型）使用不同的元素类型。

这类集群上可能没有的 MPI，而是高度依赖 POSIX 兼容并行文件系统支持的输入输出操作。DIAMOND 将参考序列集和查询序列集都划分为数据块，这两个数据块集合的笛卡尔积中的一个元素称为一个工作包。工作进程会并行处理这些工作包。此工作流程区分节点间的并行共享文件系统与节点本地磁盘。对照所有参考数据块处理完一个查询数据块后，最后一个工作进程会汇总结果并将其写入输出文件。这些选择可能对高性能计算系统的性能产生重大影响。尽管如此，DIAMOND 的分布式内存并行重点在于提供在商用集群上运行的能力以及容错能力，而非高性能。

V. 蛋白质相似性搜索流水线

我们针对蛋白质相似性搜索问题的方法由三个主要组件构成：(i) 发现可能具有一定程度同源性的候选成对序列；(ii) 对发现的候选序列进行批量比对；(iii) 根据比对中获得的信息构建蛋白质相似性图。在发现候选成对序列时，PASTIS 可以选择引入作为某个 k-mer 的 m 个最近邻的替代 k-mer，或接入简化字母表 [15], ，这两种方式都可以提高灵敏度。PASTIS 可以使用不同的比对库及其中的算法，并能够在构建相似性图时无缝整合平均核苷酸一致性和覆盖度等常见的序列比对指标。这些选项使 PASTIS 能够覆盖整体搜索空间中的不同区域，并提高搜索的有效性。

PASTIS 的基本信息存储和处理媒介是稀疏矩阵。它们用于表示整个搜索过程中所需的不同类型信息。例如，序列中的 k-mer 信息被记录在一个稀疏矩阵中，该矩阵的行和列分别对应于序列和 k-mers，而该矩阵中的一个非零元素表示特定序列中存在特定的 k-mer。除作为并行线性代数中研究最充分且优化程度最高的结构之一外，稀疏矩阵还具有灵活性，因为任意信息都可以封装在这些元素中，例如位置、得分等。图 1 展示了部分稀疏矩阵

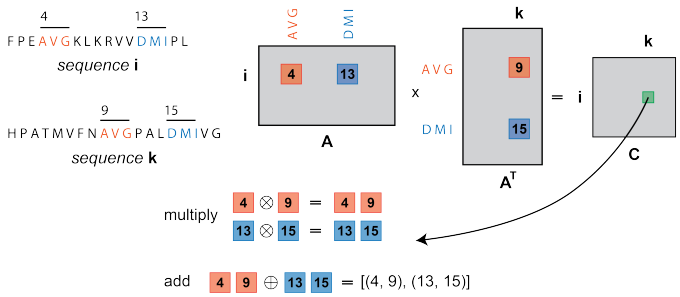


图 2：半环代数使 PASTIS 能够通过稀疏操作表达相似性搜索中的计算。此处展示了一个发现候选比对序列对的简单示例。

这些稀疏矩阵被用于 PASTIS，由此可见，这些矩阵能够轻松存储和处理我们的相似性搜索流水线所需的任何必要信息。

尽管稀疏矩阵在线性代数领域十分常见且应用广泛，但近年来，得益于 GraphBLAS 标准化工作的推动 [16]，它们在图计算中的应用和重要性开始日益凸显。其基本动机是用线性代数的语言表达图计算，从而在图计算框架中利用数十年来稀疏线性代数领域积累的算法与优化成果。与线性代数中表达的矩阵运算不同，图上的运算通常需要不同的算子来处理稀疏矩阵。例如，在 PASTIS 中，候选序列对的发现通过一种重载的稀疏矩阵“乘法”来表达；该操作涉及的元素是自定义数据类型，而传统的“乘加”运算则由自定义算子重载，这些算子称为半环。半环代数能够通过稀疏矩阵运算表达图操作，我们利用多种半环来实现不同类型的比对（图 2）。

A. 软件栈与并行性

我们的蛋白质相似性搜索流水线使用了多个库，并在分布式环境中对它们进行编排。对于分布式稀疏矩阵及其上的计算，它依赖 CombBLAS [17] ——一种分布式内存并行图形库，基于稀疏矩阵和向量上的任意用户定义半环。对于并行比对，它使用 SeqAn C++library [8] 和 ADEPT [18]。在这些库中，CombBLAS 支持 MPI/OpenMP 混合并行，SeqAn 支持具有向量化的节点级共享内存并行，而 ADEPT 支持节点级众核并行。此外，PASTIS 本身也直接使用 MPI/OpenMP 混合并行。PASTIS 的软件栈如图 3 所示。

我们方法中的一项重要设计选择，是将用于比对的并行级别与其他组件所用的并行级别分开。对于比对，我们有意优先选用能够利用节点上不同并行特性的节点内库，

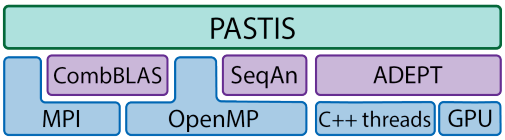


图 3：PASTIS 使用了不同的库，能够高效利用节点上的中央处理器和图形处理器资源。

例如线程、中央处理器向量指令，或 GPU 上的细粒度并行。PASTIS 通过分布式稀疏矩阵计算来处理这些节点内比对库，而这类计算已有快速且经过优化的数据结构和算法。正如我们的研究所展示的，高性能的关键在于妥善协调节点内并行与节点级并行，并采用能够克服性能瓶颈的技术。

CombBLAS [17]采用二维分解来处理分布式稀疏矩阵，将矩阵划分为矩形块。它使用方形进程网格，要求进程数为完全平方数。它支持压缩稀疏列和双重压缩稀疏列存储格式 [19]，并包含用于稀疏矩阵乘法等复杂操作的快速先进算法，能够在节点内 [20]和大规模场景下 [21]高效执行此类操作。

ADEPT [18] 是一个经图形处理器加速的序列比对库，支持 DNA 和蛋白质序列比对。它结合单元间通信和任务内并行的方法，在图形处理器上实现完整的 Smith-Waterman 序列比对。ADEPT 通过高效利用图形处理器的内存层次结构，并在计算动态规划矩阵时利用快速的寄存器间数据传输来实现单元间通信，从而获得出色的性能。ADEPT 提供 CUDA、HIP 和 SYCL 版本，分别可利用英伟达、AMD 和英特尔图形处理器。ADEPT 的驱动类充当调用应用程序与图形处理器内核之间的接口；驱动程序会检测节点上所有可用的图形处理器，并将比对任务分配到这些图形处理器上。每个图形处理器均由一个专用的 C++ 线程负责数据打包以及数据传输（传入和传出图形处理器）。

B. 性能特征

计算模式。 我们的蛋白质相似性搜索流水线中执行的两种基本计算类型，是比对所需的稀疏计算和编辑距离计算。前者属于内存受限型，计算强度低、内存占用高且访问模式不规则；后者属于计算受限型，计算强度高，并以统一模式计算编辑距离矩阵，而这些矩阵规模较小且密集。考虑到大多数比对库能够通过 SSE 和 AVX 向量指令集实现高性能（PASTIS 所使用的 SeqAn 便是如此），而 Summit 上的 IBM PowerPC 处理器不支持这些指令集，我们将图形处理器资源专门用于比对，并利用 CPU资源执行内存密集型稀疏计算。鉴于上述计算的特性，加速器更适合用于比对，而不适合稀疏计算。因此，我们依靠 ADEPT（见图 3）进行比对；ADEPT 支持节点内多个加速器。该库也会使用中央处理器上的资源，但这些资源仅占比对过程的一小部分。

内存需求。 多对多蛋白质相似性搜索需要大量内存，而该领域现有的库采用了各种技术来解决这一问题，从将中间文件写入磁盘到分阶段执行搜索。对于包含 2000 万条序列的中等规模数据集，通常需要存储数千亿个候选比对，并执行数百亿次比对。即使是这种相对小规模搜索所需的内存，也可能很快超过单个节点的内存容量。此外，为了高效执行搜索，还需要额外的数据结构。例如，我们的方法使用并行稀疏矩阵乘法来发现候选比对，而这种操作所需的中间内存通常远多于存储已发现候选项实际所需的空间。每个输出元素平均计算和存储的中间结果数量称为压缩因子；即使该值适中，介于 1 和 10 之间（这在基因组学数据集中很常见），也足以说明在多对多搜索中必须特别重视内存管理。最后，需要存储并进行比对的候选对数量会随着搜索中的序列数量呈二次增长，这使得海量数据集上的相似性搜索在内存需求方面更具挑战性。

输入输出与通信。 PASTIS 使用并行消息传递接口输入输出处理输入和输出文件。PASTIS 的输入是一个 FASTA 格式文件（生物信息学中用于表示核苷酸和蛋白质序列的一种非常常见的文件格式），输出则是由三元组构成的相似性图，其中各条目表示两个序列及其相似性。输出文件通常比输入文件更大。PASTIS 中的通信可分为两类：序列所需的通信，以及稀疏计算所需的通信。前者的开销通过以非阻塞方式执行通信，直到实际需要时（即进行成对比对时）才使用其结果，从而得到了有效隐藏。鉴于节点支持加速器，最稀疏计算所需的通信（以及计算）也可以被隐藏。我们将在第VI-C节中研究这一问题。与前文所述的内存和计算问题相比，在我们执行多对多蛋白质相似性搜索的方法中，输入输出与通信瓶颈通常不构成严重问题。通常，输入输出在 PASTIS 的总体执行时间中所占比例不超过 3%。

VI. 实现的创新

我们主要通过三种新颖的

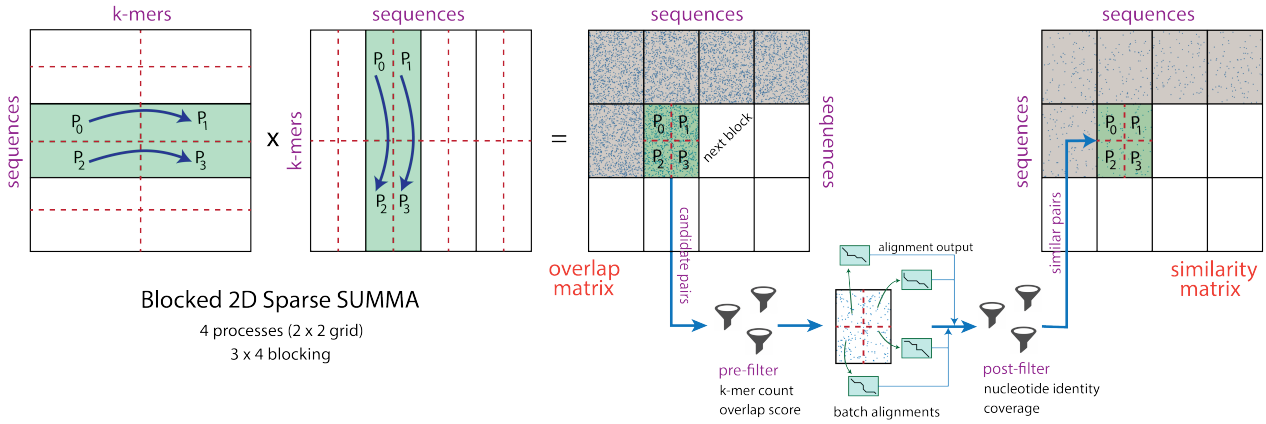


图4：通过分块二维稀疏 SUMMA 发现候选比对，以及增量相似性搜索

技术应对分布式蛋白质相似性搜索带来的计算挑战。搜索的主要性能瓶颈——巨大的内存需求——通过提出一种用于分布式构建重叠矩阵的二维稀疏 SUMMA 的分块变体加以解决（第六节A）。基于重叠矩阵具有对称性这一观察（相似性图是无向图），我们提出了避免大量稀疏计算的技术，以及两种不同的负载均衡方案。这两种方案会根据分块因子的不同表现出不同的行为，并能够实现良好的计算负载均衡（第六节B）。随后，我们介绍一种能够隐藏内存受限的稀疏计算开销以及某些通信操作开销的技术（第VI-C节）。我们在包含数千万条序列的小规模数据集上验证了所提出的创新方法。所有实验均在 Summit 系统上进行（规格见第八节）。对于报告中的所有实验，我们在每个节点上使用 1 个 MPI 任务，并利用每个节点上的全部 42 个核心和 6 个 GPU。

A. 分块二维稀疏 *SUMMA*

在我们的方法中，候选序列通过并行稀疏矩阵乘法发现，该操作生成一个重叠矩阵，其中包含需要进行比对的序列对。候选对所需的内存极大，而对候选对进行分块构造的动机基于这样一个观察：实际上只有其中一部分彼此相似。利用比对前后可获得的信息（ k-mer 的共同数量、核苷酸一致性、覆盖度等），通常最终只有不到 5% 的候选对会进入最终的相似性图。因此，增量相似性搜索可以大幅减少所使用的内存。

用于计算重叠矩阵的 $C = AB$ 形式并行稀疏矩阵乘法算法是二维稀疏 SUMMA 算法 [22]。在我们的分析中，假设矩阵为方阵，行/列维度为 n ，且稀疏矩阵的元素均匀分布。给定 p 个并行进程，该算法分 $\sqrt{p}$ 个阶段执行：在每个阶段中，输入矩阵的某些子矩阵会被广播，并计算输出矩阵的部分结果。假设集合广播采用树形算法 [23],的通信成本为

$$2\alpha\sqrt{p}\log\sqrt{p} + 2\beta s\sqrt{p}\log\sqrt{p},$$

其中， s 是维度为 $n/\sqrt{p} \times n/\sqrt{p}$ 的稀疏子矩阵中非零元素的数量， α 是消息启动时间，β 是每字传输时间。

我们将二维稀疏 SUMMA 算法推广到具有任意分块因子的情形，以便按块构造输出矩阵。我们在稀疏矩阵乘法中将输出矩阵构造成 $br \times bc$ 个块，其中 br 和 bc 分别为行和列分块因子。计算所需的输出块$C(r,\ c)$时，需要行条带 $A(r,\ *)$ 和列条带 $B(*,\ c)$。最初，输入矩阵分布在 $\sqrt{p} \times \sqrt{p}$ 网格中的 p 个进程之间。因此，为了能够按块计算输出矩阵，必须将 A 分割为 br 个行条带，并将 B 分割为 bc 个列条带。这些行条带和列条带中的每一个都必须分布在$\sqrt{p} \times \sqrt{p}$ 进程网格中。图4左侧展示了：对于 3×4 分块，输入矩阵如何分布在组织成 2×2 网格的四个进程之间，以及用于计算 $C(1,\ 1)$ 的子矩阵。

通信成本。与 plainSUMMA 相比，分块变体增加了通信开销，因为输入矩阵需要多次广播。如前所述，内存需求是相似性搜索的主要瓶颈之一。此外，正如第VI-C节将要介绍的，输出块计算的开销在很大程度上可以被隐藏，也就是说，广播和局部稀疏计算的开销都可以被隐藏。尽管如此，当块的数量很多且其开销无法隐藏时，通信成本的增加可能难以承受。分块变体的总通信成本为

$$2\alpha(br \cdot bc)\sqrt{p}\log\sqrt{p} + \beta s(br + bc)\sqrt{p}\log\sqrt{p}.$$

从相似性搜索的角度来看，分块二维 SparseSUMMA 的优缺点如下：

- 支持在海量数据集上进行相似性搜索，并降低多对多序列比对所需的内存消耗。

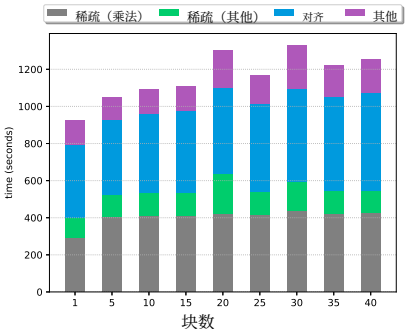


图 5：增加块数对稀疏组件和比对组件运行时间的影响。

- 为不同类型的优化开辟了道路，因此能够进一步显著缩短搜索所需的总时间。采用第六节B和第六节C中描述的技术，可将总运行时间缩短多达 30%；这些技术与分块构造相辅相成。

- 由于通信增加且稀疏计算被拆分，发现候选比对所需的时间也随之增加。

图5展示了在 Summit 的 100 个节点上，对包含 2000 万条序列的数据集进行相似性搜索时，随着块数增加，各组件的并行运行时间。与一次性执行整个搜索（即块数为 1）相比，块数增加后，比对和乘法的运行时间平均分别增加了 10%–15% 和 40%–45%，总体运行时间约增加 30%。值得注意的是，仅使用一个块时，无法在更少的节点上执行此搜索，这表明其内存需求之高。

B. 负载均衡技术

相似性搜索中的重叠矩阵 C 用于生成待比对的候选序列。该稀疏矩阵的行和列表示序列，其中每个非零元素都对应一项需要执行的两两比对。非零元素包含比对或过滤所需的自定义信息（例如序列中的种子位置、共有的 k-mer 计数等）。使用分块二维稀疏SUMMA算法进行并行计算时，重叠矩阵的每个块都会分配给$\sqrt{p} \times \sqrt{p}$ 网格中的所有 p 进程。

– –

重叠矩阵是对称的：非零元素 C_{ij} 和 C_{ji}表示需要对序列 i 和 j进行比对。这会影响我们工作中搜索的执行方式。首先，乘法过程中可能无需计算该矩阵中大约一半的元素。其次，这些元素也无需进行比对。最后，在分块构造该矩阵时，要实现良好的负载均衡，就必须采用能够考虑上述所有条件的定制方法。为此，我们提出了两种不同的方案。

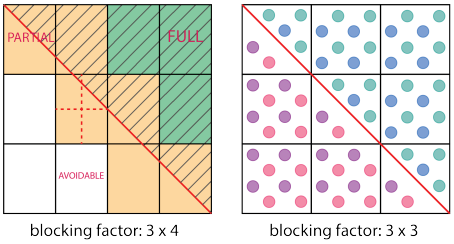


图6：基于三角性的负载均衡（左）与基于索引的负载均衡（右）。

基于三角性的负载均衡。在第一种负载均衡方案中，我们仅计算与重叠矩阵严格上三角部分有非空交集的块。如图6左侧矩阵所示，这些块可分为三类：完整块、部分块和可跳过块，其中完整块为绿色，部分块为黄色，可跳过块为白色。在分块二维稀疏 SUMMA 中，需要计算完整块和部分块，而可跳过块既不计算也不进行比对。完整块中的所有元素都需要比对，而部分块中的元素是否需要比对，则取决于它们位于重叠矩阵的下三角部分还是上三角部分。由于这些块分布在进程网格中的所有进程上，部分块可能导致负载不均衡，尤其是当它们与严格上三角部分的交集很小时。例如，对于图6左侧的重叠矩阵，假设使用 2×2 进程网格，在计算第二行与第二列交集处的块时，将有三个进程处于空闲状态。相比之下，完整块的负载均衡优于部分块；随着块数增加，完整块的数量呈二次增长，而部分块的数量呈线性增长。

基于索引的负载均衡。 在第二种负载均衡方案中，我们计算所有块，并以保留重叠矩阵原有非零元分布（通常是均匀分布）的方式对块进行剪枝。在矩阵的下三角部分，若某个非零元的行索引和列索引均为奇数或均为偶数，则保留该非零元；在矩阵的上三角部分，若某个非零元的行索引为奇数、列索引为偶数，或反之，则保留该非零元。图6右侧的矩阵展示了对 3×3blocking 进行此处理的示例。该方案会对每个块剪去约一半的非零元，同时保持矩阵的对称性，并确保每对序列只对齐一次。

比较。两种提出的负载均衡方案所需的比对计算量相同。基于三角性的负载均衡方案通过利用稀疏计算来节省开销，但代价是部分块中的负载均衡较差。不过，该方案在完整块中的负载均衡应与基于索引的负载均衡方案在块中的负载均衡一样好。基于索引的负载均衡旨在改善负载均衡，但代价是需要计算每个块。这两种方案的另一个不同之处

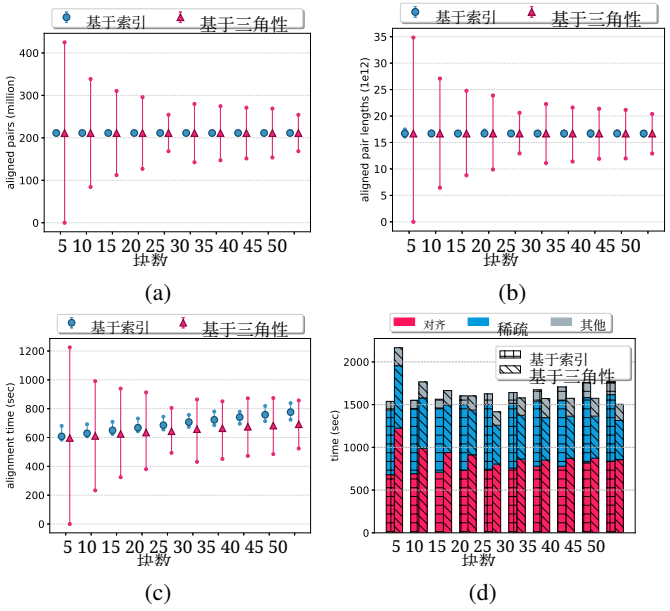


图 7：64 个进程下两种负载均衡方案的比较。左上和左下图中垂直线上的三个点分别表示相应指标的最小值、平均值和最大值，用以体现并行进程之间的负载不均衡。

在于它们改变重叠矩阵中块结构的方式。基于索引的方案在很大程度上保留了重叠矩阵中块的均匀结构，因此，与基于三角性的负载均衡方案相比，它应能带来更好的内存访问模式。

我们在图 7 中比较了 Summit 的 64 个节点上针对一个包含 2000 万条序列的数据集所采用的这两种负载均衡方案。在图 7a、7b 和 7c 中，图上的每根垂直条分别表示并行进程在相应指标上达到的平均值、最小值和最大值。如图 7a 所示，对于所有测试的块数，基于索引的方法都比基于三角性的方法实现了更好的负载均衡。随着块数增加，基于三角性的方法的负载均衡往往会有所改善。这是因为部分块的比例会随着块数增加而降低，而部分块正是该方法中导致负载不均衡的主要原因。

虽然比对时间可以说与比对数量成正比，但更好的衡量指标是编辑距离矩阵大小的总和，因为序列长度各不相同，而且本研究使用的比对算法是 Smith-Waterman算法的一个变体 [18]，它会计算整个距离矩阵。图 7b 展示了这一指标下的负载不均衡情况。在这一指标下，基于索引的方法再次比基于三角性的方法实现了更好的负载均衡。需要注意的是，这两种负载均衡方法的目标都是比对对的数量（即图 7a），而基于索引的方法在这一指标上实现了非常好的负载均衡。这反映在图 7c 所示的实际比对耗时负载不均衡情况中，在那里，基于索引的方法取得了更好的性能。

最后，基于三角性的方法的主要优势——能够减少稀疏计算——所产生的影响如图 7d 所示。由于能够避免大量此类计算，该方法在稀疏计算方面的运行时间更短。当该方法的负载不均衡影响较小时（即块数较多时），其总运行时间优于基于索引的方法。如图 7d 所示，当块数为 $\{5, 10, 15, 20\}$ 时，基于索引的方法运行时间更短；其他情况下，尽管基于三角性的方法比对时间更长，但由于它能够避免稀疏计算，因此运行时间更短。

C. 预分块

相似性图的分块构造形成了一个迭代流水线，在此过程中，图逐步构建而成（如图 4 所示）。该流水线的组件由节点上的中央处理器或图形处理器资源执行。异构节点架构，以及在图形处理器上执行计算受限的批量成对比对的能力（这类比对适合采用 SIMD 类型的并行性），使我们的方法能够利用节点上的全部计算和内存资源进行相似性搜索。我们进一步提出一种优化技术，称为预分块，该技术基于对包含候选对的稀疏块进行预计算。预分块旨在提高节点上的资源利用率，从而缩短整体搜索时间。

在逐步构建相似性图的过程中，先通过稀疏计算发现比对任务（这些计算由中央处理器执行），再由 GPU 执行比对。在以分布式方式执行比对时，大量 CPU 资源处于空闲状态，而针对下一个块或多个块的稀疏计算实际上可以同时开始。这样便能提前发现下一组比对的候选对，并为比对做好准备。请注意，这种发现过程是完整的分布式稀疏矩阵乘法，包含集合通信操作和受内存带宽限制的计算。因此，能够隐藏这些开销的能力弥足珍贵，因为它可以同时消除这两种瓶颈。

线程管理。 PASTIS 和用于稀疏计算的图形库 CombBLAS 都高度依赖 OpenMP 实现节点内并行。尽管比对库 ADEPT 在 GPU 上执行比对，但仍会使用 CPU 资源进行预处理和后处理。具体而言，它使用的线程数与节点上的 GPU 数量相同。我们分配与节点上设备数量相同的线程给 ADEPT，其余线程则用于预分块。ADEPT 所用线程的亲和性根据其所属的 CPU 插槽进行设置。

比较。所提出的预分块技术主要需要权衡的是：它会稍微增加内存使用量（计算并存储下一个块所需的内存

TA表 I：预分块对基于索引和基于三角性的负载均衡方法的影响 .

负载均衡	块	未进行预分块的时间（秒）				进行预分块后的时间（秒）				归一化			效率（%）
		对齐	稀疏	sum	总计	对齐	稀疏	sum	总计	对齐	稀疏	总计	
基于索引	10	627	582	1209	1555	722	663	740	1090	1.15	1.14	**0.70**	97.6
	20	667	582	1249	1606	765	726	793	1123	1.15	1.25	**0.70**	96.4
	30	705	586	1291	1659	804	767	842	1163	1.14	1.31	**0.70**	95.5
	40	740	590	1330	1724	836	801	873	1203	1.13	1.36	**0.70**	95.7
	50	774	596	1370	1774	871	841	919	1245	1.13	1.41	**0.70**	94.8
基于三角性	10	610	465	1076	1812	674	610	864	1468	1.10	1.31	**0.81**	78.0
	20	634	411	1045	1641	694	571	844	1320	1.09	1.39	**0.80**	82.2
	30	658	394	1052	1602	716	574	857	1287	1.09	1.46	**0.80**	83.5
	40	674	388	1062	1609	731	585	867	1286	1.08	1.51	**0.80**	84.3
	50	692	362	1053	1548	749	568	844	1243	1.08	1.57	**0.80**	88.7

或多个块），以此隐藏用于发现候选序列对的稀疏矩阵乘法开销。尽管相似性搜索需要大量内存，但只要预先计算的块数较少，预分块带来的额外内存消耗就应该不高。在 Summit 上，我们发现，在大多数运行中，对齐与稀疏计算所用时间的比例不超过 2:1。预先计算的块数应取决于这一比例，并且可以进行调整。在我们的实验中，我们始终只预先计算下一个块。预分块预计会增加对齐和稀疏计算所用的时间，因为现在这些组件是并发计算的，而中央处理器上的计算资源和内存资源必须由二者共享。不过，即使最大值略有增加，采用预分块后的总运行时间仍会从这些组件运行时间的总和降至其中的最大值。

表 I 展示了用于评估预分块技术效率的各种指标。我们针对五种不同的块数$\{10, 20, 30, 40, 50\}$，比较了采用和不采用预分块的基于索引和基于三角性的负载均衡方法。我们评估了比对、稀疏乘法、两者总和以及总体执行时间（前两个大列标题下）的耗时。请注意，采用预分块时，“总和”列给出的是实际获得的时间，而不是“对齐”和“稀疏”列的简单相加。在“归一化”列标题下，采用预分块获得的值是相对于不采用预分块获得的值进行归一化的。最后一列评估了预分块技术的效率。

如表 I 所示，尽管预分块方案增加了比对和稀疏计算所耗费的时间，但它能够在很大程度上掩盖前者的开销，并使基于索引的方案运行时间减少 30%，使基于三角性的方案运行时间减少 20%。基于三角性的负载均衡方案中预分块的效率低于基于索引的方案（约为 80%，而后者为 95%），这可以用该方案中的负载不均衡会对预分块效率产生不利影响来解释。总而言之，预分块技术对于基于索引的负载均衡更为有效；在这两种负载均衡方案中，它都能显著缩短总运行时间。

VII. 性能测量方法

我们的工作采用了三种报告机制。第一种是计时器，用于测量特定组件所耗费的时间，例如比对、稀疏计算、输入输出所耗费的时间，或总运行时间等。对于第 VI 节中的部分实验，我们通过测量所有进程在相应组件上所耗费时间的最小值、平均值和最大值来计算负载不均衡。第二项报告指标是每秒比对次数。对于该指标，我们考察整个并行运行时间，记录执行的两两比对总次数，并将其除以并行运行时间。最后一项指标是每秒单元更新次数。该指标通常用于衡量比对算法的性能，表示所用算法每秒更新的单元数。对于此指标，我们只采用比对内核所耗费的时间，即 Smith-Waterman算法中的前向评分时间，并用更新的单元数除以该时间。

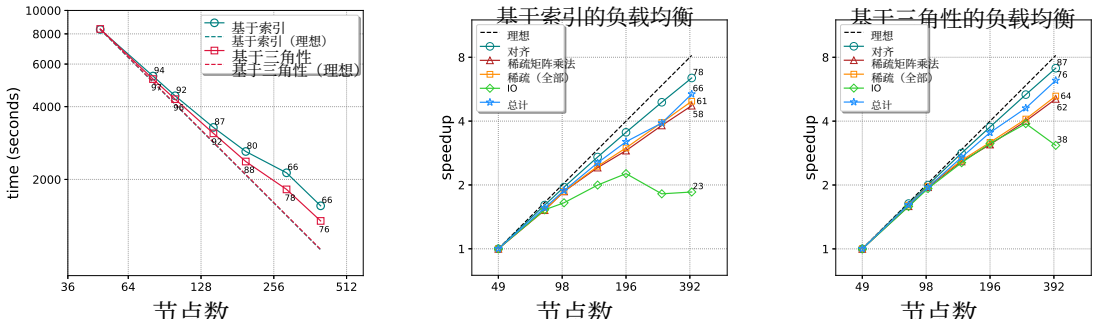
VIII. 性能结果

本节扩大了实验规模，与第 VI 节中的评估相比，以考察 PASTIS 的并行性能。我们在 OLCF 的 IBM 系统 Summit 上进行评估。该系统由 4608 个 IBM Power System AC922 节点组成，每个节点配备两个 22 核、3.8 GHz 的 IBM POWER9 处理器，以及六个 NVIDIA Tesla V100 加速器，每个加速器具有 80 个流式多处理器。每个节点配备 512 GB 的 CPU内存，并为加速器提供总计 96 GB 的 HBM2 内存。这些节点通过双轨 InfiniBand 网络，以非阻塞胖树拓扑连接。

我们分别在第 VIII-A 节和第 VIII-B 节研究强扩展和弱扩展的行为。在这些实验中，我们使用的节点数不超过 1000；强扩展实验使用的最大节点数为 400，弱扩展实验则为 784。在本节的最后一部分（第 VIII-C 节），我们展示了使用 Summit 的 3364 个节点进行的全规模运行。

A. 强扩展

我们评估基于索引和基于三角性的负载均衡方案的强扩展性能。我们使用一个包含 5000 万条序列的数据集，并在 $\{49, 81, 100, 144, 196, 289, 400\}$ 个节点上扩展我们的方法。我们使用



(a) (b) (c)

图 8：强扩展性能。图中标注的数值表示达到的并行效率（%）。表II：序列通信等待（cwait）以及输入输出时间占总运行时间的百分比。

	基于索引		基于三角性	
#节点	通信等待%	IO%	cwait%	IO%
49	0.14	0.68	0.14	1.37
81	0.17	0.70	0.17	1.39
100	0.18	0.78	0.19	1.39
144	0.21	0.87	0.22	1.45
196	0.23	0.97	0.25	1.54
289	0.23	1.48	0.27	1.62
400	0.27	1.98	0.31	2.77

每个节点运行 1 个 MPI 任务，并使用每个节点上的所有核心和加速器。我们使用 8×8 的分块因子，在启用预分块的情况下构建相似性图。对于该数据集，已执行的比对数量为 865 亿；基于索引的方案中，整个重叠矩阵（即包含所有块）包含 1.99 万亿个元素，而基于三角性的方案中则包含 1.12 万亿个元素。

图8a通过绘制并行运行时间展示了 PASTIS 的强扩展。图中的虚线表示理想情况。从49个节点扩展到400个节点时，基于索引的负载均衡方案达到66%的并行效率，而基于三角性的负载均衡方案达到76%的并行效率。后者效率更高，可归因于其避免了大量稀疏计算，尽管其负载均衡效果较差。

图8b和图8c绘制了不同组件的加速比。对于这两种方案，可以看到，计算密集型组件“对齐”（在加速器上执行）的可扩展性更好：在400个节点上，基于索引的方案和基于三角性的方案的并行效率分别为78%和87%。两种方案中，稀疏操作的效率均约为60%。借助所提出的算法创新，我们在很大程度上克服了稀疏计算带来的开销——其运行时间占总运行时间的很大一部分（图7d）——而因此仅损失11%至12%的效率。我们注意到，有些稀疏计算无法通过预分块避免。尽管输入输出可扩展性有些不稳定，但该组件在总运行时间中所占比例很小，不会成为瓶颈。

表II 展示了完成序列通信所需的输入输出时间和等待时间的总体百分比。PASTIS 会直到比对开始时才需要这些序列，并且它 uses 通过启动 IO 传输，为它们进行非阻塞通信。在读取输入序列后立即传输，此类通信操作的等待时间可以忽略不计。输入输出也不会构成瓶颈，这主要得益于高效的 MPI 输入/输出。这两个组件所占百分比的总和通常不到总运行时间的 3%。

B. 弱扩展

我们考察了采用基于索引的负载均衡方案时，相似性搜索流水线的弱扩展特性。随着节点数 $\{25, 49, 100, 196, 400, 784\}$ 增加，我们相应地调整序列数。比对数随序列数呈二次方增长。对于大多数稀疏计算，这一结论同样成立，因为稀疏矩阵乘法复杂度中的浮点运算次数与输出元素数成正比（假设压缩因子保持不变）。因此，当进程数增加 x 倍时，我们将序列数增加 $\sqrt{x}$ 倍。由此，我们从 25 个节点上的 2000 万条序列开始，并在 49、100、196、400 和 784 个节点上分别使用 2800 万、4000 万、5600 万、8000 万和 1.12 亿条序列。图 9 和表 III 展示了所得结果。

如图9所示，比对组件表现出更高的弱扩展效率。总体而言，除输入输出外，所有组件都表现出良好的弱扩展性能。如前所述，输入输出仅占并行运行时间的一小部分，这个问题似乎并未影响整体弱扩展效率，其始终高于80%。

C. 大规模相似性搜索

本节对一个包含 4.05 亿条蛋白质序列的数据集执行多对多相似性搜索。该数据集通过对来自两千多个宏基因组和宏转录组数据集的 15.9 亿个蛋白质序列片段进行聚类和组装而创建 [24]。我们使用非冗余变体的一个子集，其中可与更长序列按其 99% 的残基进行对齐且序列一致性达到 95% 的子片段均被删除[1]。

[1]https://metaclust.mmseqs.org/current release/

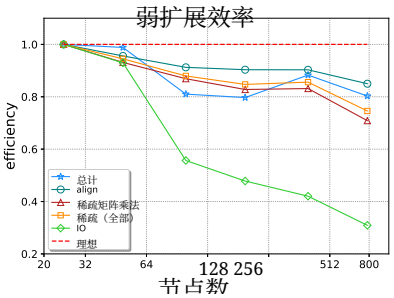


图 9：不同组件的弱扩展。

#节点数	#序列数	#对齐数
25	20M	135亿
49	28M	26.7B
100	40M	55.1B
196	56M	108.9B
400	80M	225.4B
784	112M	452.4B

表 III：数量 of 序列和比对。

表 IV：大规模生产运行的参数、结果和统计数据。

实验参数	
系统	OLCF 的 Summit
节点数	3364
进程网格（二维）	58×58
每进程核心数	42
每进程 GPU 数	6
编译器（中央处理器）	GNU gcc 9.1.0
编译器（图形处理器）	CUDA nvcc 11.0.3
MPI	Spectrum MPI 10.4
程序参数	
输入序列数量	404,999,880
k-mer 长度	6
空位开放罚分	11
空位延伸罚分	2
Common *k*-mer 阈值	2
ANI阈值	0.30
覆盖度阈值	0.70
分块因子	20×20
负载均衡	基于三角性
预分块	已启用
结果	
发现的候选项	95,855,955,765,012
已执行的比对	8,552,623,259,518（8.9%）
相似对（输出元素）	1,048,288,620,764（12.3%）
搜索空间	1.6e17
比对空间	5.2e-5
输出（文件大小）	27 TB
运行时间	3.44 小时
每秒比对次数	690,609,577
每秒单元更新次数	176.3 万亿细胞更新每秒
细分及其他	
Time	
对齐	2.62 小时
稀疏矩阵乘法	2.06 小时
稀疏（全部）	2.22 小时
预分块	2.62 小时
IO	12.0 分钟
通信等待	0.2 分钟
负载不均衡（%）	
比对	7.1
稀疏	3.1
序列×k-mer 矩阵	
维度	$404{,}988{,}624 \times 244{,}140{,}625$
元素	48,824,292,733

表 IV 展示了实验中使用的参数和程序，并详细介绍了所获得的结果。

我们的生产运行使用了 3364 个节点（占整个 Summit 系统的 73%），并在 3.44 小时内完成了整个搜索。总计发现了 95.9 万亿个候选成对比对，其中执行了 8.6 万亿个，而其中 1.1 万亿个通过了平均核苷酸一致性和覆盖度阈值，最终纳入搜索结果。持续达到每秒 6.906 亿次比对，峰值达到 176.3 万亿细胞更新每秒。

我们在每个节点上使用 1 个 MPI 任务，每个节点配备 42 个核心和 6 个 GPU。对于分块二维稀疏 SUMMA，我们总共使用了 400 个块，分块因子为 20×20。对于本文所述的性能相关参数，我们启用了预分块，并采用了基于三角性的负载均衡方案，因为它在块数较多时性能更好。表 IV 底部进一步细分了整体执行时间。

我们尝试在包含 5000 万、1 亿和 2 亿条序列的大型数据集上运行 DIAMOND 和 MMSeqs2，以执行多对多搜索。这两种搜索工具都依赖 SSE 和 AVX 向量指令来快速进行比对。Summit 系统的处理器不支持这些指令集。因此，我们尝试使用 NERSC 的 Cori 系统。该系统基于中央处理器，采用英特尔处理器，并支持这类向量指令。对于 MMSeqs2，我们先从较少的节点数开始，即使用 64 个节点处理 5000 万条序列子集，但 6 小时内未能完成运行。我们还尝试使用 256 个节点处理 5000 万和 1 亿条序列子集，但同样未能在 12 小时内完成。测试中，我们将 MMSeqs2 的敏感度值设为 5.7。对于 DIAMOND，我们尝试使用 150 个节点处理 1 亿条序列子集，并使用 400 个节点处理 2 亿条序列子集，但两次运行都因错误而失败。我们尝试了 DIAMOND 的非常灵敏模式和超高灵敏模式。我们成功使用默认模式（快速模式）在 4 个节点上完成了 DIAMOND 对 5000 万条序列子集的处理。此次运行耗时 22 分钟，每秒比对次数稳定在约 6 万次。该数值远低于作者在更高灵敏度模式和更多节点上运行时得到的结果。因此，我们将本次运行结果与 DIAMOND 的结果进行比较这些数据最近在马克斯·普朗克学会的另一台超级计算机系统 Cobra 上报告 [6]。

我们将本次运行的性能结果与 DIAMOND 报告的结果进行比较 [6]。根据其研究报告，DIAMOND 在 520 个节点上，使用高灵敏模式对 2.81 亿条查询序列与包含 3900 万条序列的参考数据库进行搜索，耗时 5.42 小时，完成 230 亿次比对；使用超高灵敏模式则耗时 17.77 小时，完成 231 亿次比对。以高灵敏模式为例，这相当于在由 $281 \times 39 \times 10^{12}$ 条序列构成的多对多搜索空间中，每秒处理 120 万条序列。我们的运行在由 $405 \times 405 \times 10^{12}$条序列构成的搜索空间中，达到了每秒 6.906 亿条序列的处理速度。我们的实验在大 15.0 倍的搜索空间中进行，同时将每秒比对次数提高了两个数量级，即达到 575.5 倍。

至于执行的比对总数，如果将报告中的 DIAMOND 运行规模扩展到我们实验所用的搜索空间规模，预计比对次数将达到 3457 亿次。我们的方法能够进行更高灵敏度的搜索，每

单位搜索空间的比对次数为 5.2e-5，而 DIAMOND 为 2.1e-6，相差 24.8 倍。假设 DIAMOND 随节点数线性扩展，将其扩展到 2025 个节点后，预计求解时间为 12.53 小时，而我们的实验耗时为 3.44 小时。这表明，尽管我们的比对次数多了一个数量级，搜索速度仍快 3.6 倍。

IX. 启示

多对多蛋白质序列搜索是一种稀疏且不规则的全对全比较。稀疏性取决于数据，因此只有在运行时才能确定哪些成对比较值得执行。因此，这给网络互连带来了巨大压力。直接执行这项任务，相当于进行一次庞大的 MPI_AlltoAllv 调用，也称为个性化全对全广播。PASTIS 通过规范化计算，显著减轻了互连网络的压力。它通过将问题转化为稀疏矩阵运算来实现这一点。这项技术也可应用于其他不规则应用；它已成功用于图问题和组合问题 [17]。

节点内计算涉及稀疏矩阵-矩阵乘法，以及每个节点上大量的成对比对。后者非常适合映射到 GPU 的宽向量单元；英伟达在新近发布的 Hopper 架构中引入了动态规划（DPX）指令，有望使蛋白质序列搜索中开销最大的部分提速达 40 倍。在配备 DPX 指令的此类架构上运行的 PASTIS 将显著提速，但随着计算速度大幅提升，在大规模运行时也会受到通信成本的限制。因此，采用 Hopper 等加速器的未来超级计算机需要配置更高的二分带宽和网络注入带宽。

由于 PASTIS 在稀疏矩阵乘法中使用半环，而所有高性能稀疏矩阵乘法 GPU 实现都针对浮点运算进行了硬编码，因此我们在中央处理器上执行了这些步骤。得益于数十年来对中央处理器上高性能稀疏矩阵乘法实现的研究，这并未成为性能瓶颈。然而，随着前述变化应用于加速器，对其他代数执行稀疏矩阵运算将会受到极大欢迎。英伟达的 cuSPARSE 库朝这一方向迈出了一大步：其最新版本提供了一种优化实现，可以使用自定义运算符执行稀疏矩阵与稠密矩阵的乘法。我们建议 GPU 厂商提供的稀疏矩阵库也将这一能力扩展到其他函数。

致谢

本项工作由美国能源部科学办公室下属的先进科学计算研究（ASCR）计划资助，合同编号为 DE-AC02-05CH11231。本研究还得到百亿亿次计算项目（17-SC-20-SC）的资助，该项目由美国能源部科学办公室与国家核安全管理局合作开展。

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。

参考文献

[1] N. Glover、C. Dessimoz、I. Ebersberger、S. K. Forslund、T. Gabald´on、J. Huerta-Cepas、M.-J. Martin、M. Muffato、M. Patricio、C. Pereira、A. S. da Silva、Y. Wang、E. Sonnhammer 和 Q. f. O. C. Thomas、Paul D，《直系同源物探索的进展与应用》，《分子生物学与进化》，第 36 卷，第 10 期，页码 2157–2164，2019 年 6 月。[2] G. Caetano-Anoll´es 和 D. Caetano-Anoll´es，《蛋白质结构架构的进化结构化宇宙》，《基因组研究》，第 13 卷，第 7 期，页码 1563–1571，2003 年。[3] A. Godzik，《宏基因组学与蛋白质宇宙》，《结构生物学最新观点》，第 21 卷，第 3 期，页码 398–403，2011 年。[4] M. B. Scholz、C.-C. Lo 和 P. S. Chain，《新一代测序与生物信息学瓶颈：宏基因组数据分析的现状》，《生物技术最新观点》，第 23 卷，15，2012，1，9 期，页码 – 分析生物技术。[5] T. Prakash 和 T. D. Taylor，《宏基因组数据的功能注释：挑战与应用》，《生物信息学简报》，第 13 卷，第 6 期，页码 711–727，2012 年 7 月。[6] B. Buchfink、K. Reuter 和 H.-G. Drost，《利用 DIAMOND 实现生命之树尺度的灵敏蛋白质比对》，《自然·方法》，第 18 卷，第 4 期，页码 366–368，2021 年 4 月。[7] M. Steinegger 和 J. S¨oding，《MMseqs2 支持灵敏的蛋白质序列搜索，用于海量数据集分析》，《自然·生物技术》，第 35 卷，第 11 期，页码 1026–1028，2017 年 11 月。[8] A. D¨oring、D. Weese、T. Rausch 和 K. Reinert，《SeqAn：用于序列分析的高效通用 C++库》，《BMC生物信息学》，第 9 卷，第 1 期，页码 11，2008 年 1 月。[9] S. F. Altschul、T. L. Madden、A. A. Sch¨affer、J. Zhang、Z. Zhang、W. Miller 和 D. J. Lipman，《带缺口的 BLAST 和 PSI-BLAST：新一代蛋白质数据库搜索程序》，《核酸研究》，第 25 卷，第 17 期，页码 3389–3402，1997 年 9 月。[10] S. M. Kiełbasa、R. Wan、K. Sato、P. Horton 和 M. C. Frith，《自适应种子驯服基因组序列比较》，《基因组研究》，第 21 卷，第 3 期，页码 487–93，2011 年 3 月。[11] B. Buchfink、C. Xie 和 D. H. Huson，《使用 DIAMOND 进行快速而灵敏的蛋白质比对》，《自然·方法》，第 12 卷，59，2015 年 1，页码。[12] R. C. Edgar，《搜索与聚类速度比 BLAST 快几个数量级》，《生物信息学》，第 26 卷，第 19 期，页码 2460–2461，2010 年 8 月。[13] A. E. Darling、L. Carey 和 W. C. Feng，《mpiBLAST 的设计、实现与评估》，2003 年 1 月。 [在线]。网址：https://www.osti.gov/biblio/976625

[14] O. Selvitopi、S. Ekanayake、G. Guidi、G. A. Pavlopoulos、A. Azad 和 A. Buluc.，《使用稀疏矩阵进行分布式多对多蛋白质序列比对》，载于 SC20：国际高性能计算、网络、存储与分析会议，2020 年，页码 1–14。[15] L. R. Murphy、A. Wallqvist 和 R. M. Levy，《用于蛋白质折叠识别的简化氨基酸字母表及其对折叠的影响》，《蛋白质工程、设计与筛选》，第 13 卷，152，2000 年 3 月 3，149 期，页码 –。[16] J. Kepner、P. Aaltonen、D. Bader、A. Buluc.、F. Franchetti、J. Gilbert、D. Hutchison、M. Kumar、A. Lumsdaine、H. Meyerhenke、S. McMillan、C. Yang、J. D. Owens、M. Zalewski、T. Mattson 和 J. Moreira，《GraphBLAS 的数学基础》，载于 2016年IEEE 高性能极限计算会议（HPEC），2016 年，页码 1–9。[17] A. Azad、O. Selvitopi、M. T. Hussain、J. R. Gilbert 和 A. Buluc.，《组合 BLAS 2.0：在分布式内存系统上扩展组合算法》，IEEE 并行与分布式系统汇刊，第 33 卷，第 4 期，页码 989–1001，2022 年。[18] M. G. Awan、J. Deslippe、A. Buluc、O. Selvitopi、S. Hofmeyr、L. Oliker 和 K. Yelick，《ADEPT：面向 GPU架构的领域无关序列比对策略》，《BMC生物信息学》，第 21 卷，第 1 期，页码 406，2020 年 9 月。

[19] A. Buluc. 和 J. R. Gilbert，《超稀疏矩阵的表示与乘法》，载于 *IPDPS*。IEEE，2008 年。[20] Y. Nagasaka、S. Matsuoka、A. Azad 和 A. Buluc，《英特尔 KNL 和多核架构上的高性能稀疏矩阵-矩阵乘积》，载于 *ICPP 工作坊*。美国纽约州纽约市：ACM，2018 年，页码 34:1–34:10。[21] M. T. Hussain、O. Selvitopi、A. Buluc.和 A. Azad，《极端规模下避免通信且内存受限的稀疏矩阵-矩阵乘法》，载于 *2021 年 IEEE 国际并行与分布式处理研讨会（IPDPS）*，2021 年，页码 90–100。[22] A. Buluc. 和 J. R. Gilbert，《并行稀疏矩阵-矩阵乘法与索引：实现与实验》，《SIAM 科学计算杂志》，第 34 卷，第 4 期，页码 C170–C191，2012 年。[23] E. Chan、M. Heimlich、A. Purkayastha 和 R. van de Geijn，《集合通信：理论、实践与经验：研究论文》，《并发计算：实践与经验》，第 19 卷，第 13 期，页码 1749–1783，2007 年 9 月。[24] M. Steinegger 和 J. S¨oding，《线性时间内对海量蛋白质序列集进行聚类》，《自然·通讯》，第 9 卷，第 1 期，页码 2542，2018 年 6 月。