Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY

MNRAS 00, 1 (2023年) https://doi.org/10.1093/mnras/stad067

HLC2: 一种 高 效 天文星表 交叉匹配 框架，适用于 大型 天文星表 异构 计算 环境

Yajie Zhang [1,2] Ce Yu [1], [2]★Chao Sun, [1,2] Jian Xiao, [1,2] Kun Li, [1,2] Yifei Mu [1,2] 和 Chenzhou Cui [2,3]

[1] 天津大学智能与计算学院，中国天津市海河教育园区雅观路135号，邮编300350 [2] 国家天文科学数据中心技术研发创新中心，中国天津市海河教育园区雅观路135号，邮编300350 [3] 中国科学院国家天文台，中国北京市朝阳区大屯路20号，邮编100012

录用 2023年 1月 2日。 收稿 2022年 11月 28日； 原稿 修改 稿 2022年 8月 3日

摘 要 摘 要 摘 要 摘 要

交叉匹配操作旨在从多个星表中查找同一天体或同一片区域对应的数据，是天文数据分析与研究不可或缺的一环。随着正在进行和下一代大规模巡天产生大量天文星表，交叉匹配的时间复杂度急剧增加。异构计算环境为加速交叉匹配提供了理论上的可能性，但异构计算资源的性能优势尚未得到充分利用。为应对天文观测数据量大幅增长带来的交叉匹配挑战，本文提出了一种在CPU-GPU异构计算平台上基于球面位置偏差的高性能交叉匹配框架——异构计算赋能的大型星表交叉匹配器（HLC2）。该框架支持可扩展且灵活的交叉匹配，可直接应用于巡天任务和天文数据中心大型天文星表的融合。本文提出了一种性能估算模型，用于定位性能瓶颈并指导优化。本文设计了两级分区策略，根据天体的位置生成优化的数据放置方案，以提高吞吐量。为使HLC2具有更强的适应性，本文设计并集成了架构感知任务划分、线程并行化和并发调度策略。此外，针对边界问题，本文提出了一种新颖的四向策略，以有效平衡性能与完备性。我们使用公开发布的星表数据对HLC2进行了实验评估。实验表明，HLC2在不同规模的星表上均具有良好的扩展性，与先进的交叉匹配器相比，交叉匹配速度显著提升。

关键 字： 方法： 数据 分析 – 技术： 其他 – 星表 – 软件： 公开 发布 e.

1 引言

在天文学研究中，交叉匹配是一项关键操作，用于从多个来源中查找同一目标或区域对应的数据。交叉匹配本质上是一种空间匹配操作，具有高度数据并行的特点。它是多波段天文学中融合不同波段观测数据，以及时域天文学中获取同一天体不同时刻信息的先决条件。高效执行交叉匹配不仅有利于融合来自不同波长或望远镜的数据，也是时间序列数据重建、瞬变候选体发现和后续巡天部署的必要条件。随着天文观测设备的增加和观测能力的提升，天文学家努力实现对全天空的时间和空间监测，监测间隔达到数小时甚至更短。因此，持续开展的巡天项目已产生包含海量天体的天文星表，例如斯隆数字化巡天（SDSS 2015）和盖亚（Gaia et al. 2018），每个星表都包含超过 10 亿个天体（Jia, Luo & Fan 2016）。单幅观测图像可解析出近百万条星表记录（盖亚 2020）。在大数据时代，数据量天文数据量持续增长，美国国家科学院指出，到 2030 年底，一些天文台的各项任务将产生 500 PB 数据（National Academies of Sciences et al. 2021）。这给天文学家交叉匹配并从这些数据中提取信息带来了挑战。由于计算复杂度很高，逐对处理每个源并不可行。

针对两个大型星表之间的交叉匹配问题，索引分区是减少不必要计算的有效方法。以往的研究提出了多种基础空间索引，包括层次等面积等纬度像素化（HEALPix；Gorski 等，2005 年）、层次三角网格（HTM；Kunszt、Szalay 和 Thakar，2001 年）、Zones（Gray 等，2004 年；Gray、Nieto-Santisteban 和 Szalay，2007 年）等。这些方案可以减少部分计算，但也会带来边界问题。解决这些问题的方法主要包括在边界处添加冗余数据（Zhao 等，2009 年）、使用双重索引（Du 等，2014 年；Yu 等，2020）等。面对数据激增和高度分散的情况，提高效率的另一种方法是通过多种方式实现数据并行化。目前，只有少数研究针对多核 CPU（Nieto-Santisteban、Szalay 和 Thakar，2005 年；Zhao 等，2009 年；Du 等，2014 年；Riccio 等，2017 年；Soumagnac 和 Ofek，2018 年）和图形处理器（Budavari 和 Lee，2013；Jia 等人，2016；Jia 和 Luo，2016）开展了相关工作。

★ 电子邮件： yuce@tju.edu.cn

© 2023年 作者 由 牛津 大学 出版社 代表 英国 皇家 天文学 会 出版

尽管上述方法在提升整体性能方面做了大量工作，但仍存在一些问题和技术挑战。对于包含数百万个以上源的两个星表之间的交叉匹配问题，性能仍需进一步提升。同时，使用异构计算资源进行交叉匹配处理时，需要高效的分区和并行组织策略，才能充分发挥异构处理器的优势。此外，随着星表中的天体记录数量增加，现有策略为提升速度而牺牲了部分匹配完整性；而边界问题所导致的冗余数据生成与处理，也使计算更加耗时。因此，速度与完备性之间的平衡仍有进一步探索的空间。

为应对上述挑战，我们提出并实现了一种高效交叉匹配框架，名为 HLC2，旨在 CPU-GPU 异构计算环境中加快交叉匹配处理。HLC2 的主要特点可概括如下：

(i) 灵活的交叉匹配框架。HLC2 能够高效地处理包含数亿个源的星表交叉匹配计算。它能够良好地适配不同英伟达图形处理器架构及不同规模的天文星表，模块化设计也能确保可扩展性。

(ii) 边界问题的新解决方案。四方向策略旨在提供一种高效方案，在提升交叉匹配性能的同时，平衡计算完备性的损失。实验结果显示，完备性有所提高，且速度与次优方法相近。

(iii) 高性能并行处理。本文提出一种性能预测模型，用于理论分析影响性能的因素。根据性能分析结果，提出并实施了优化策略，以实现高效的 CPU-GPU 协同处理。在中央处理器上采用两级分区策略，同时在图形处理器上进行动态线程组织和输入/输出优化。与先进的交叉匹配器相比，交叉匹配速度显著提升。

本文其余部分安排如下。第 2 节简要介绍交叉匹配的细节，并介绍与天文伪球面索引方案和并行加速方法相关的现有研究。第 3 节和第 4 节重点介绍 HLC2 框架的设计和优化策略。第 5 节描述相关实验的结果。第 6 节总结全文并讨论未来工作。

2 背景与相关工作 RK

2.1 天文星表的交叉匹配

星表是一个包含赤经、赤纬、星等等天体信息的二维数据表。由于数据采集方式、校准方法和观测仪器各不相同，同一天体在不同星表中的坐标可能略有差异（Nieto-santisteban 等（2005））。因此，基于校准误差设定的距离阈值，是判断两个天体是否为同一天体的依据。通常，两个天体 O 1 和 O 2 之间的角距离可根据以下规则计算（Zhang 和 Zhao（2003））。

(1) 距离公式：
天体 O 1 和天体O 2 的坐标分别表示为(α_1, δ_1)和(α_2, δ_2)。根据球面余弦定律，球面

A 与 d 之间的角距离可以表示为：

$$d = \arccos\left(\sin\delta_1 \sin\delta_2 + \cos\delta_1 \cos\delta_2 \cos\left(\alpha_1 - \alpha_2\right)\right). \quad (1)$$

当 O_1 与 O_2 之间的角距离非常小，

$$\delta \approx \left(\delta_1 + \delta_2\right)/2, \quad (2)$$

因此球面角距离可以计算为

$$d^2 = \left(\left(\alpha_1 - \alpha_2\right) \times \cos\delta\right)^2 + \left(\delta_1 - \delta_2\right)^2. \quad (3)$$

(2) 成功交叉匹配公式：

$$d = \sqrt{\left(\left(\alpha_1 - \alpha_2\right) \times \cos\delta\right)^2 + \left(\delta_1 - \delta_2\right)^2} \leq 3\sqrt{r_1^2 + r_2^2}, \quad (4)$$

其中，r 1 和 r 2 分别为两个星表的误差半径。当两点之间的距离 d 满足该不等式时，即可认为匹配成功，这意味着这些记录代表同一个天体的信息。因此，对于大小分别为 m 和 n 的两个星表数据集，时间复杂度为 O (mn)，对于近期大规模巡天产生的大量数据而言，这是不可接受的。

2.2 天文索引方案

对于海量数据处理，通过索引减少不必要的比较是提高效率的重要手段。对每个目录进行简单的嵌套循环匹配需要进行大量不必要的成对比较，因此以往的研究提出了各种空间索引。三种主流索引方法分别是 HEALPix（Gorski et al. 2005）、HTM（Kunszt et al. 2001）和 Zones（Gray et al. 2004; Gray et al. 2007），它们分别将球面划分为菱形、三角形和矩形单元。它们的共同思路是，在给定分辨率下将天空划分为固定数量的区域，并为每个天体分配一个整数索引，指向其所在区域。彼此距离较近的天体会被分配到相同索引或相邻区域的索引。HEALPix 按分辨率层级组织索引，以整个天空为根节点，以最精细分辨率下的区域为叶节点。新建区域会在其父区域的索引后附加一个有限集合中的标签。同一父区域下细分出的区域按顺时针嵌套方式命名。研究人员提出了四叉树立方体（Q3C），将球面划分为一个立方体（Koposov & Bartunov 2006）。Landais、Ochsenbein 和 Simon（2013）受 Q3C 启发，构建了二维 PostgreSQL 库 HEALPix-Tree-C（H3C）。其功能与 Q3C 相同，但采用 HEALPix。

除了通过空间索引分区进行优化，另一种方法是创建搜索树。两者的区别在于，树索引可以将计算复杂度从 O (n 2) 降至 O (nlogn)，而空间索引则会剪枝两个星表中索引不同的记录，不直接对其进行计算。Zhao 之前的工作在相同的实验环境下，比较了针对天空区域的网格索引和针对赤纬的一维 B树搜索这两种优化方法（Zhao 等 2009年）。与赤纬一维索引的并行算法相比，网格索引的性能提升了约 60 倍。最近，Li等人（2019年）提出了一种多波段交叉匹配方案，该方案基于 K D树，采用专门设计的星表格式和数据布局；当源的数量增至 1 亿时，其查询响应速度优于 Q3C 和 H3C。树结构在检索问题中已展现出显著的实用价值。

上述主流空间索引减少了不必要的比较，但它们也带来了负面倘

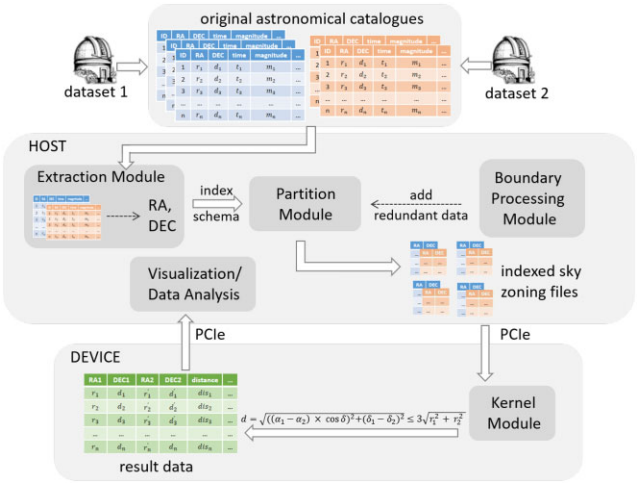

图 1. 基础HLC2架构概览。它主要由四个模块组成：提取模块，用于提取交叉匹配计算中涉及的位置信息；分区模块，用于将星表划分以进行并行计算；边界处理模块，用于平衡完备性与性能；以及内核模块，用于在图形处理器上实现星表间并行化。

影响：同一对象的两条不同记录可能被划分到不同的相邻区域中。因此，在后续基于球面位置偏差的计算中，这两条记录不会进行比较，导致该对象的相应数据丢失，进而可能遗漏部分天文发现。为了简化计算流程并提高性能，许多交叉匹配方法忽略了边界问题，从而造成一定程度的结果丢失。同时，在基于索引的分区过程中，也有针对边缘数据的额外处理策略。Zhao 等（2009年）提出了一种补救方法，通过快速位运算算法增加边界数据。然而，这会引入过多冗余数据，严重影响大型数据集上的实时性能。Du 等（2014年）提出了一种结合 HEALPix 和 HTM 的双索引方法，减少了部分数据泄漏，该方法也用于时间序列数据重建（Yu等人，2020）。尽管已取得巨大改进，但我们注意到，这些方法增加的计算负载不利于提高计算效率。因此，对于两个包含数百万个以上源的星表之间的交叉匹配，需要一种高效的边界求解算法，以平衡计算复杂度与完备性。

2.3 现有 天文 星表 交叉匹配加速方法

GPU 等现代并行处理器已广泛应用于各种场景，以提升处理性能。此外，以 Zhao 等（2009年）为代表的研究人员提出了使用 MPI 的加速方法，Du 等（2014年）利用线程池技术加速交叉匹配。Jia等人（2016年）探索了 CPU-GPU 集群上的交叉匹配策略。此外，mcatCS（Li 等，2019年）也是一种用于高效整合天体数据的分布式交叉匹配方案。然而，它侧重于复杂度较低的单记录交叉匹配，因此在大规模星表匹配中耗时较长。与此同时，Wang等人（2013）提出了两种基于圆锥搜索的并行算法，并将其部署在 GPU 上；Budavari 和 Lee（2013）则应用了它 在 多个 图形处理器上。 上述 算法 能够 在 几 秒 内 完成 数百万条 星表 数据 的 交叉匹配。 然而，进一步 提升 性能 和 资源 利用率 仍有必要， 并且 还需解决 数据 量更大的 交叉匹配 计算 任务 。

此外，还有其他基于 Hadoop（Li等人 2014年）和 Spark 框架（Jia & Luo 2016年；Ze ˇcevi ´c 等人 2019年）的加速方法。另外，已有若干用于处理海量星表的交叉匹配工具，其中包括 TOPCAT（Taylor 2011）、SIMBAD（Wenger 等人 2000）、VizieR（Ochsenbein、Bauer & Marcout 2000）、CDS-Xmatch（Boch、Pineau & Derriere 2014）和 NASA/IPAC（Helou 等人 1991）。这些工具提供观测数据查询、部分编辑和分析功能。近年来，还开发了 catsHTM（Soumagnac & Ofek 2018）等其他网络应用。然而，由于在基于网络的应用中，交叉匹配计算占据了服务器资源消耗的大部分，目前的任务数量和用户数量可能受到限制。与此同时，有人提出了一款独立的命令行 Python 工具 C3（Riccio 等人 2017），其性能可与 CDS-Xmatch 和 TOPCAT 相媲美。

3 异构计算赋能大型星表交叉匹配器

本研究聚焦于通过 CPU-GPU 协同处理加速大规模天文星表交叉匹配的方法。基于球面位置偏差的交叉匹配计算被迁移至 CPU-GPU 异构计算环境。我们提出了一种名为 HLC2 的通用交叉匹配框架，适用于高效的天文星表处理。

3.1 HLC2 基本 架构 设计

图 1 展示了 HLC2 的基本设计。我们的算法被封装为一个端到端、基于命令行的交叉匹配工具，可在 Linux 平台上运行，并使用 C、C++ 和 CUDA 为 NVIDIA GPU 实现。GPU 通常拥有比 CPU 更多的计算核心，这些核心以流处理器的形式存在，还可进一步组成流式多处理器（SM）组。GPU 程序采用的单指令多线程方式有利于处理各种天文信息处理任务，尤其适用于交叉匹配问题中的角距离计算，因为每对星表记录都遵循相同的计算模式，且彼此之间没有数据依赖。

与此同时，采用 基于模块 的软件 架构 ，以 确保 灵活性 和 可扩展性 。它 主要 由 四 个 模块 组成 ，即 提取 、分区 、边界 处理 和 内核 。HLC2 执行 交叉匹配 的 工作流程 如下：

在中央处理器（即主机端）上，HLC2 首先调用提取模块，对原始数据进行初始化。天文学家可以输入来自不同巡天项目、规模各异的两个星表。因此，系统会从中提取天体位置数据（主要是赤经、赤纬），并过滤掉无用信息。接下来，分区模块根据 HEALPix 索引将过滤后的星表数据划分到不同的天区中；该索引也可以替换为第 2.2 节中提到的其他方案。一个星表中的记录只与另一个星表中来自同一天区的数据进行计算。同时，在数据划分期间，系统会调用边界处理模块，在避免引入过多可能影响计算性能的冗余数据的同时，确保交叉匹配计算的完备性。用于处理

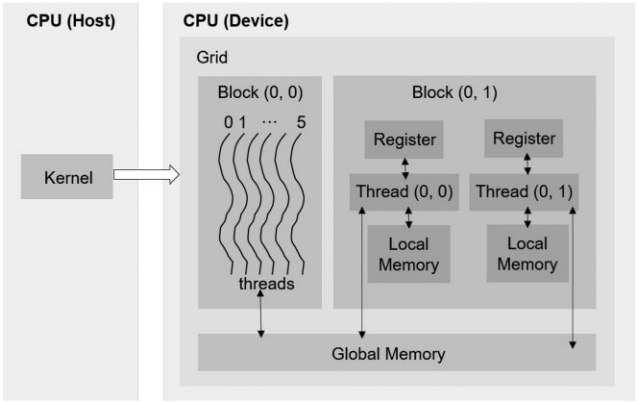

图2。CUDA线程和内存的基本层级结构。每个线程都有自己的寄存器和本地内存，每个网格都有自己的全局内存，不同线程块中的线程都可以读取和修改其中的内容。

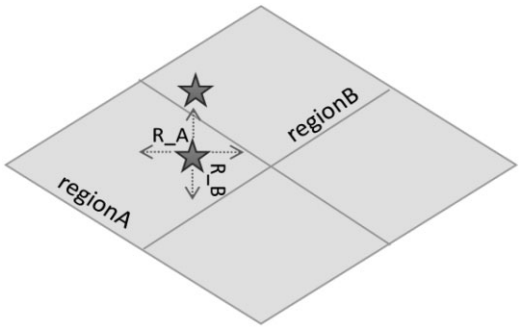

图3。四方向策略示意图。图中的实线表示索引划分的示意线。在对赤经和赤纬的误差半径进行加减后（虚线所示），区域A中的天体（以五角星形状显示）落入区域B。因此，根据我们的策略，区域A中的天体记录将被添加到区域B中。

边界 问题 详见 第 4.1.2 。 随后， 索引 天空分区文件 中的 天体 记录 对 将 被 传输 至 图形处理器。

在图形处理器（即设备端）上，内核模块采用星表间并行化，根据第2.1节所述方法计算角距离。如图2所示，英伟达提出了线程层级结构的概念，以便于线程组织（Sanders 和 Kandrot 2010年）。由于星表是一维文本数据，因此我们相应地采用一维线程层级。将索引化天空分区文件中天体记录对的距离计算任务分配给多个线程。每个线程独立执行任务，互不通信。最后，交叉匹配结果传输到中央处理器，并导出最终产品。结果也可以进行可视化，目前该功能仍在开发中。

3.2 边界问题的 四向 策略

边界问题是影响大规模星表处理过程中匹配结果完备性的主要因素。如图 3 所示，落在区块（区域A）边缘的天体，可能会落在另一星表中该区块边缘之外，并进入其相邻区域（区域B）。在边界处理模块中，

我们提出了一种名为四方向策略的高效方法，以兼顾完备性和性能。算法1展示了采用四方向策略进行边界处理的具体实现。在构建空间球面索引时，通过增加对天体是否可能位于相邻索引区域的判断，将可能处于边界的天体添加到所有对应区域中。四方向策略的示意图如图 3 所示。对于每个靠近边界的天体，我们的方法会分别判断将其赤经和赤纬误差半径（R_A 和 R_B）加到或减去其位置后得到的点是否落入相邻区域。如果是，则将该天体的信息添加到对应的相邻天空区域。区域A中的天体星表记录在赤纬加上 R_B 后会落入区域B，因此该记录也会参与区域B的计算。天体记录仍只在计算区块内参与距离计算，且索引编号、赤经和赤纬保持不变。通过这种方式，我们扩大了每个数据区块的范围；这些区块是根据用户选择的划分级别粒度划分的，从而减少部分匹配损失。

算法 1 边界 处理的 四向 策略

要求： 两个 星表 A 和 B;
确保： 索 引 ed sk y 用于并行交叉匹配的 Zones 分 区;

```
初始 映射 record_A , record_B ;
对于 所有 RA, DEC 属于 inputlines_A/B 执行
    hpid ← hp nestid(RA, DEC, order)
    ra_plus ← ra + R_A
    ra_minus ← ra − R_A
    dec_plus ← dec + R_B
    dec_minus ← dec + R_B
    hpid_up ← hp nestid(ra plus, DEC, order)
    hpid_down ← hp nestid(ra minus, DEC, order)
    hpid_left ← hp nestid (RA, dec minus, order)
    hpid_right ← hp nestid (RA, dec plus, order)
    如果 record_A/B.find(hpid) = record_A/B.end() 则
        record_A/B.insert((hpid, line));
    end if
    如果 hpid_up/down/left/right ! = hpid 那么
        如果 record_A/B.find(hpid_up/down/left/right )
          = record_A/B.end() 然后
            record_A/B.insert((hpid_up/down/left/right , line));
        结束 如果
    结束 如果
结束 对于 ;
```

4 HLC2 的 性能 优化

为提升整体性能，我们研究了使用中央处理器与图形处理器协同处理的交叉匹配性能建模，以定位性能瓶颈并估算计算开销、数据移动开销等。借助准确的性能模型，我们对基础HLC2中的交叉匹配方法进行性能分析，并开展理论分析，以指导进一步优化。

4.1 性能 模型 与 性能分析

4.1.1 模型 定义

我们的 执 行 时间 估算 模型 涵盖 主 机 主机 （中央处理器） 和 设 备设备（图形处理器） ， 并 按 轮次 执行 算法 。该模型 是 基于.....提出的

表 1. 性能 模型 中 使用 的 参数 列表。

符号	描述
R	程序 所需 的 轮次 数量
r	总 核心 数量
b	每个 多处理器 的 核心 数量
γ	多处理器 执行 一条 指令 的 开销
λ	访问 全局 内存 中的 一个 内存 区块 的 开销
q_i	所有 多处理器 在 该 轮次 访问 的 全局 内存 区块 数量
t_i	所有 多处理器 在 该 轮次 执行 的 最大 操作 数量
σ	需要执行的 固定 开销 同步 任务
α	在 中央处理器 与 图形处理器 之间 发送 第 一个 字节 时的初始 延迟 开销
β	在 中央处理器 与 图形处理器 之间 发送 后续 每个 字节 所需 的 开销
s_i	为 β 计算 而在中央处理器 与 图形处理器 之间 传输 的数据 大小
t_s	在 中央处理器 与 图形处理器 之间 传输 大小为 s_i 的数据 所需时间
I_i	本 轮 中 向内 传输 的数据 量
O_i	在 该 轮次 中 向外 传输 的 数据

基于 ATGPU（Carroll & Wong 2017）模型和λ-Model（Riahi, Savadi & Naghibzadeh 2020）提出，但我们的估算更加准确、全面。表1总结了所提出模型中使用的主要符号。为构建整个架构，我们用 R 表示程序所需的轮次数，用 MP

=

{ 1, 2, }

mp mp . . . , mp k 表示多处理器（MP）集合，其核心总数为 r，每个 MP 有 b 个核心。一轮开始时，输入数据从主机传输到设备的全局内存。随后，内核函数在全部或部分多处理器及核心上运行。与此同时，指令在 MP 的核心上执行，而流式多处理器（SM）执行一条指令的成本用变量 γ 表示，该变量对应于图形处理器的时钟频率。本轮中所有多处理器访问的全局内存块数量用 q i 表示，每次访问的成本记为 λ。此外，我们将所有多处理器中的最大操作数记为 t i。执行内存访问指令时，核心会等待所有内核内存请求处理完毕。每轮结束时，输出结果从全局内存传输到主机。同步操作（例如释放和重新分配数据结构、重置设备等）的成本固定为 σ，随后开始下一轮。

在 中央处理器 与 图形处理器 之间 传输d 字节 数据 所 需 时间 可 计算 如 下:

$$T(d) = \alpha + \beta \times d. \tag{5}$$

第一个主要参数 α 是发送第一个字节时产生的固定延迟。β 表示发送后续每个字节所需的时间，对应于传输带宽的倒数（Boyer、Meng 和 Kumaran，2013 年）。根据 Riahi 等人（2020 年）提出的方法，我们测量当前实验环境下传输单个字节所需的时间，以表示 α。接着，为了确定 β，我们测量传输大小为 s i 的数据所需的时间 t s，然后变量 β 可设为 $\frac{t_s-\alpha}{s_i}$. 。令 I i（分别为 O i）表示第 i 轮中传入（分别为传出）的数据量。因此，传入数据传输的成本可表示为 T I（i）$= \alpha + \beta$I i。同样，传出数据传输的成本 T O（i）等于 $\alpha + \beta$O i。然后，假设图形处理器中有足够的资源使每个线程块都能并发运行，则该算法的开销上限可表示为:

$$\sum_{i=1}^{R} \left(T_I(i) + \frac{t_i + \lambda q_i}{\gamma} + T_O(i) + \sigma \right). \tag{6}$$

4.1.2 交叉匹配算法的性能分析

基于我们提出的模型，我们可以分析交叉匹配算法的成本函数，将其视为成本上界，并得出其与数据集大小之间的缩放关系。轮次数为 1。并行时间复杂度为 O（nb），全局内存用量为 O（n 2 ），传输复杂度为 O （$\alpha + \beta$n 2 ），输入/输出复杂度为 O （$\binom{n}{b}^2$ （$\alpha + b$））， 最终总成本为

$$O(\alpha + n^2 \frac{t_s - \alpha}{s_i} + \frac{nb + \left(\frac{n}{b}\right)^2 (n+b)\lambda}{\gamma} + \sigma). \tag{7}$$

我们假设计算使用两个大小相同、均为 n 的数据集。因此，在使用任何索引和分块调度优化之前，一个星表中的 n 条记录会与另一个星表中的 n 条记录逐一进行比较。使用索引后，星表会被划分为不同大小的区块，分别表示为$n' = n'_1, n'_2, \cdots, n'_j$和$n'' = n''_1, n''_2, \cdots, n''_i$。只有索引相同的数据区块才会彼此进行计算，表示为 $v = n'_1 n''_1 + n'_5 n''_5 + \cdots + n'_i n''_i$，这大幅减少了计算量。因此，在使用真实数据进行实验时，公式（7）中的变量 n 可设为 v，以计算特定数据集的计算量。请注意，为了进行具体的成本预测，可以根据所用实验环境确定其他变量。因此，从成本函数可以看出：（1）随着数据规模增大，数据传输成本呈平方增长；（2）当硬件环境确定时，并行复杂度和 I/O复杂度对性能有重大影响，而分区策略和并行内存访问模式会影响这些复杂度。

我们使用第 3 节介绍的基础HLC2，对大小从 n = 0 到 200 MB 的不同规模真实星表进行了预实验。每次试验重复 10 次，并对结果取平均值以降低随机误差。此外，我们还利用所提出的模型进行理论分析。请注意，与特定硬件环境相关的变量是根据实验所使用的环境确定的（如表 3 所示）。观测结果绘于图4。图5展示了实际计算时间和模型预测成本，结果均归一化至 0–1 范围。

4.1.3 讨论

从 图 4 可以 看出 图4 5 ，我们的 模型 能够 准确 预测 不同 输入 星表 的 执 行 时间 。此外 ，由 图 可 知 ，我们的 模型 还能够准确预测各种输入星表的执行时间。

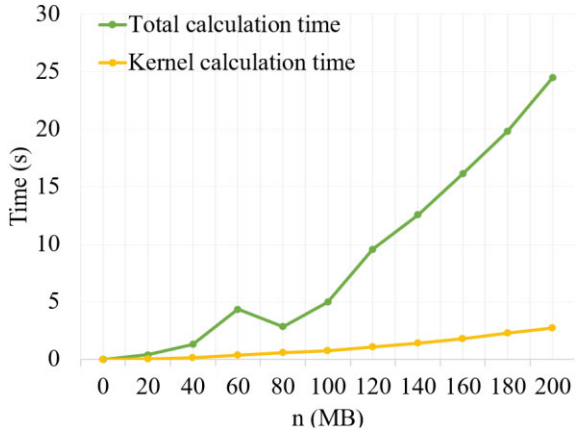

图4. 总计算时间与核函数计算时间。此初步实验所用的硬件配置如图3所示，数据集来自公开发布的星表。可以看出，数据传输占用了总时间的大部分，显著拖累了整体性能。

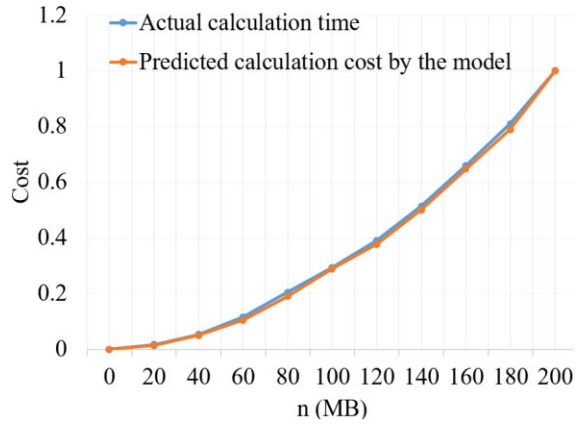

图5. 实际总运行成本与模型预测成本。此初步实验所用的硬件配置如图3所示，数据集来自公开发布的星表。结果已归一化至0–1范围。

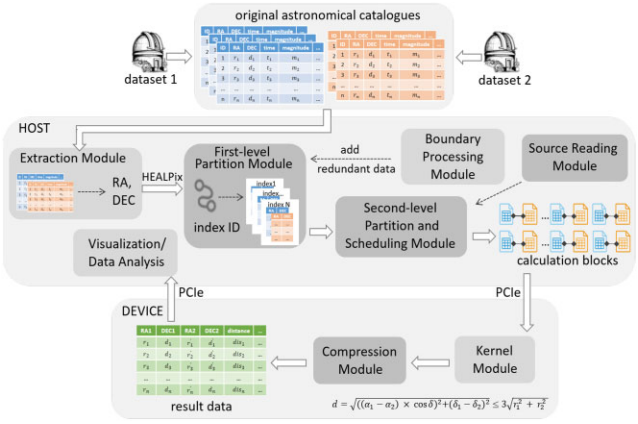

图6. HLC2 的功能扩展。我们基于基础HLC2框架集成了四个用于性能优化的新模块。经过两级分区，以便在图形处理器上存储和并行访问星表后，即可获得计算区块。源读取模块旨在获取当前中央处理器和图形处理器的计算状态，以动态调整拆分策略。最后，在压缩模块中实现输入/输出优化。

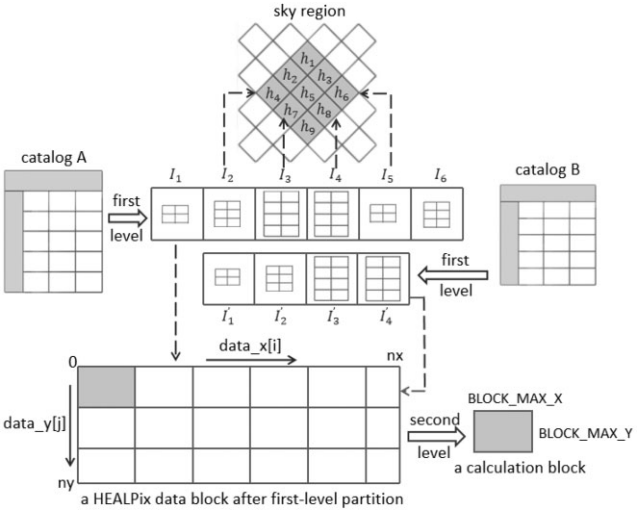

图 7. 采用 双层 分区 策略 生成 计算块 的 过程 。

如图4所示，总运行时间急剧增加，其增长幅度远大于内核时间。平均而言，数据传输约占总时间的90%，这意味着中央处理器与图形处理器之间的数据传输严重拖累了整体性能。这与数学模型一致。此外，我们发现，当星表数据密集地聚集在某一片天空区域时，可能会超出图形处理器的全局内存容量，因此无法进行传输和计算。因此，结合瓶颈分析以及交叉匹配计算面临的现有挑战，可以对HLC2进行改进，以实现更好的性能，而改进应遵循以下基本准则：

(i) 在中央处理器上，需要一种高效的预处理方案来降低计算复杂度，从而减少源端的数据传输开销。分区方法的拓扑结构应使大部分交叉匹配计算都在区块内完成，并能够支持高吞吐量的并行访问。

(ii) 在中央处理器上提取足够的并行性后，图形处理器上计算块的输入、结果和空间距离计算应尽可能相互独立，以实现计算与传输的重叠，并最大限度地提高并行核函数计算能力，从而抵消部分输入/输出开销。

4.2 中央处理器 上的 双 层 分区 策略

根据性能分析得出的基本标准，对 HLC2 进行了优化和扩展。如图 6 所示，我们集成了四个新模块。在中央处理器上，我们设计了两级分区优化，以准确限定匹配范围，并预筛选掉大量无用计算。此外，我们构建了一种名为计算区块的高效数据结构，用于高效存储和访问星表。在图形处理器上，我们提出了线程组织和输入/输出优化策略，以加快算法运行速度。图 7 展示了采用两级分区策略生成计算块的过程。值得一提的是，计算区块是一个重要概念，代表数据划分的输出。计算区块是星表中天体进行距离计算的基本单位。只有当来自不同星表的数据处于同一对计算区块中时，才会进行距离计算。它也是并行程序中任务分配的基本单位。同时，

另一个概念是 HEALPix 数据块，它在一级分区后生成。一个计算区块由多个或部分 HEALPix 数据块组成。此处，一级分区模块采用 HEALPix 索引方案，因为它效率高且受到天文学家的广泛认可。本文将 HEALPix 库原有的 *ang2pix nest* function 扩展为*hp nestid* function：给定分区级别（阶数）和天体位置（赤经、赤纬），即可获得与角度 theta 和 phi 对应的像素编号（嵌套式）。像素编号乘以一个系数后，被视为星表的索引，从而将所有索引转换为 int 格式。例如，如图 7 所示，原始星表输入被重构为地图格式，HEALPix 数据块分别按 { I 1、I 2、I 3、I 4、I 5、I6} 和$\{I'_1, I'_2, I'_3, I'_4\}$ 编制索引，每个数据块包含的天体记录数量可能不同。假设 I 2、I 3、I 4 和 I 5 分别等于 I'_1 、 I'_2 、 I'_3 和I'_4 ，那么除这四部分以外的数据都不会参与后续计算。由此可知，这些数据最初分别分布在天空区域 h 4、h 6、h 7 和 h 8 中。因此，搜索相关操作的工作负载得以降低，覆盖整个球面的星表数据分布也更加合理。

如第4.1.3节所述，实现星表间并行化的一个关键问题是中央处理器和图形处理器的内存容量有限。为了高效利用资源，在 CUDA 编程中，合理分配内存也是一项难题。因此，当硬件配置发生变化时，一级分区无法保证计算正确。为解决这些问题，源读取模块旨在获取当前中央处理器和图形处理器的计算状态。随后，二级分区与调度模块会动态生成中央处理器和图形处理器分别能够计算的最大星表记录数（N 和*block max x* and *block max y*）。提取不超过最大值的星表记录并将其组合成一个计算区块。这样，天文学家便能方便地在各种硬件环境中部署该算法；同时，可根据算法和数据类型的不同设计，调整相应参数的计算方式。例如，我们的算法中 N 和 *block max x* 的计算如下。

$$\begin{aligned} &N \times \text{sizeof(double)} \times 4 + N^2 \times \text{sizeof(double)} \\ &\quad \leqslant \text{sizeof(availableCPUMemory)} \\ &block\_max\_x \times \text{sizeof(double)} \times 4 + block\_max\_x^2 \times \text{sizeof(double)} \leqslant \text{sizeof(available图形处理器GlobalMemory)} \end{aligned} \tag{8}$$

公式中的常数‘4’表示两个输入星表中赤经和赤纬变量的数量。*block max y* 的计算方式与 *block max x* 相同。

4.3 图形处理器上的优化

4.3.1 并行化层次结构

如第 4.2 节所述，输入的原始星表被划分为计算块并分配给图形处理器。为优化每个计算块的线程管理，我们沿 x 方向划分线程，这意味着每个线程对应第一个数据集中的一条天体记录。随后，每个线程与计算块（第二个数据集）y 方向上的每条记录进行角距离计算，以实现合并访问。这里采用由线程块和网格组成的两级线程层次结构。由于计算块中的所有星表记录都存储为一维数组，因此线程层次结构由一维网格和一维区块构成，即 gridDim . y = gridDim z = 1 以及 blockDim . y = blockDim . z = 1。给定 blockDim . x 的不同取值，gridDim . x 由以下方程动态计算：

$$gridDim.x = (x\_band + blockDim.x - 1)/blockDim.x. \tag{10}$$

随后，在图形处理器上启动一个包含多个独立线程的内核。由于所有距离计算都遵循相同的模式，线程将沿着相同的执行路径运行，不会出现线程束分歧。线程束内部或线程块之间都无需同步。此外，通过分配多个 CUDA 流来执行异步内存传输，使计算与数据传输能够在时间上重叠。

4.3.2 内存管理

在图形处理器全局内存中，我们分配了以下空间：天体的位置数据数组（*d in ra*1 、*d in dec*1 、*d in ra2*和*d in dec*2）、对应于各计算区块的星表数据量（*x band*、*y band*），以及每对星表的交叉匹配结果（*d out dis*）。我们已考虑到，在图形处理器上实现星表间并行距离计算的一个关键问题是内存容量有限。因此，为了避免这一问题，在将数据传输到图形处理器之前进行数据划分时，我们按照第 4.2 节所述估算了可并行处理的数据最大量。我们还采用了页锁定内存优化，以减少数据访问延迟。同时，我们考虑将部分星表数据放入图形处理器共享内存，以加快内存访问速度。遗憾的是，当星表数据超过 100 MB 时，共享内存策略会导致频繁的数据传输，进而增加传输延迟。第 4.1 节证明，数据移动对图形处理器加速内核的性能尤为关键，而且传输到图形处理器的星表只计算一次。因此，在 HLC2 中，我们没有使用图形处理器共享内存。此外，用于存储结果的空间需要等于两个星表输入数组所占空间的乘积，但角距离仅在匹配位置存储。我们不存储角距离，而是用“1”标记匹配成功的位置，从而将传输结果的大小减半。

5 实验与结果

本节对 HLC2 进行详细评估。我们评估 HEALPix 层级和线程组织策略对性能的影响。接下来，评估所提出的用于解决边界问题的四向策略。最后，全面分析 HLC2 的整体性能，并将其与其他最先进的方法进行比较。

所有实验数据最初均取自各类天文项目公开发布的星表。为展示所提方法的可扩展性和适用性，我们从真实巡天数据中提取行数相同或不同的星表，并设计了 D1–D6 数据集（见表2）。例如，数据集 D3 是从公开可用的盖亚数据集中生成的：从赤经范围 [22.5125°、90.0009°] 和赤纬范围 [0.0056°、9.9999°] 中分别选取数量不同的记录；数据集 D4 则用于自匹配场景。除了真实星表数据，我们还基于真实数据构建了模拟数据集，用于边界问题实验。我们调整了数据分布，使其更多地集中在经过 HEALPix 索引的天空区域边缘。因此，这有助于检测边界

表 2. HLC2 实验 中 各 数据集 的 信息

Name	巡 天 测 量	文件 大小	描述	天体 数量
D1	斯隆数字化巡天 × 2MASS	13.6 M × 40 K	赤经 ∈ [340.0001°, 360.0000°], 赤纬 ∈ [1.0000°, 1.2000°]	221 126 × 784
D2	斯隆数字化巡天 × 斯隆数字化巡天	80 M × 80 M	赤经 ∈ [340.0001°, 360.0000°], 赤纬 ∈ [1.0000°, 1.2000°]	1105 627 × 1105 627
D3	盖亚 × 盖亚	1 G × 400 M	赤经 ∈ [22.5125°, 90.0009°], 赤纬 ∈ [0.0056°, 9.9999°]	12 935 566 × 5420 000
D4	2MASS ×2MASS	3.7 G × 3.7 G	赤经 ∈ [0.0000°, 360.0000°], 赤纬 ∈ [−89.9928°, 89.9956°]	61 734 000 × 61 734 000
D5	2MASS ×2MASS	9 G × 1.5 G	赤经 ∈ [0.0000°, 360.0000°],赤纬∈ [−89.9928°, 89.9956°]	144 046 000 × 25 722 500
D6	2MASS ×2MASS	28.5 G × 6.2 G	赤经 ∈ [0.0000°, 360.0000°],赤纬∈ [−89.9928°, 89.9956°]	467 555 800 × 102 890 000

表 3. 实验 的硬件 配置 。

模型	处理器	架构	时钟频率 （GHz）	内存 大小 （GB）	流式多处理器 数量 流式多处理器	核心	CUDA 计算能力	编译器
Intel Xeon E5-2682	CPU	Broadwell	2.5	60	-	32	-	gcc 7.5
英伟达 Tesla P100	GPU	Pascal	0.715	16	56	3584	6.0	nvcc 10.2
英特尔 至强 Silver4114	CPU	Skylake-EP	2.2	32	-	16	-	gcc 7.5
英伟达 Tesla V100	GPU	Volta	1.246	16	80	5120	7.0	nvcc 10.2

处理策略。请注意，模拟数据集仅用于测试边界处理策略。此外，我们在两台具有不同中央处理器和图形处理器架构的服务器上开展实验，下文分别简称为 P100 和 V100。详细的硬件配置列于表 3。HLC2 的数据样本和源代码已发布于 https://github.com/Melody888Evan/HLC2。

5.1 HEALPix 层级 的 影响

为确保执行两级划分后实现适当的负载均衡，必须避免将很大比例的数据聚集到少数几个 HEALPix数据块中。因此，为选择适中的“HEALPix 层级”，我们衡量了不同选择在完备性和性能之间的平衡。我们使用数据集 D2、D3 和 D5，并生成基于“HEALPix 层级”的索引，层级范围从 5 到 13。默认设置为每块 64 个线程。图 10 展示了随着分区粒度变化，总计算时间和匹配源数量的结果。由于实验中不同大小的数据集所匹配的数量各不相同，我们使用每个数据集各项结果与其最大结果的比值[1] 来表示匹配数量的绝对值，以便更清楚地展示匹配结果随划分级别变化的趋势。可以看出，对于这三个数据集，随着分区级别的提高，完备性呈下降趋势。这与更细的分区粒度会增加匹配丢失可能性的事实相符。至于执行时间，D2 在层级 10 时达到最低点。D3 和 D5 分别在层级 8 和 9 时达到最佳性能。对于 D3，层级 8 的设置比层级 13 快至 12.1 倍，且从层级 13 到层级 8，完备性提高了 1.4%。由于天体记录只与同一区块中的记录进行计算，较小的分区在一定程度上会降低计算复杂度。但当数据量增至 1G 时，过细的分区会增加数据预处理和生成冗余数据的开销。因此，达到最佳索引层级后，总交叉匹配时间预计会继续增加。

[1] $x^* = \frac{x}{x\_\max}$，x^*表示图中使用的比例结果，x 表示每个原始数据，x max 指不同分区级别输入下的最大结果值。由于大型星表在无匹配损失的情况下获取结果耗时较长，因此我们使用各结果与 x max 的比值来表示匹配数量。

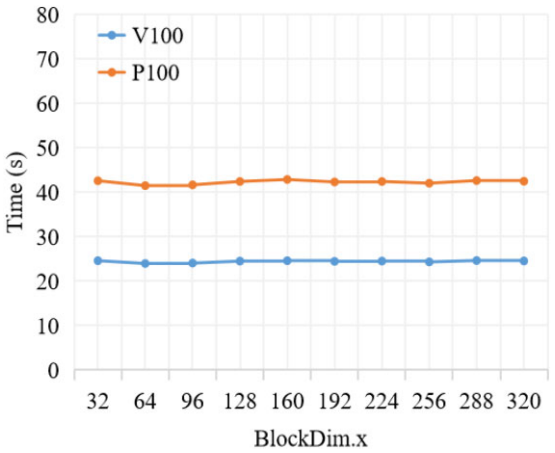

图 8. 线程 组织 方式 变化时的 性能 。对于 数据 集 D3 和 HEALPix 等级 9， 曲线 显示 了 HLC2 在 P100 和 V100 上的 处理 时间。

根据我们的实验结果，通常将“HEALPix 级别”设为 10、8 和 9 较为合适。

5.2 线程 组织 策略 的 影响

在两个不同的 GPU 环境中，使用数据集 D3 并固定 HEALPix 层级为 9，评估线程组织策略的影响。根据第 4.3 节所述的计算策略，可根据每个区块中的线程数动态确定一个网格中的区块数。对于 CUDA 计算能力为 6.0 的 P100 服务器，其流处理器数量为 3584，少于 V100 的 5120。部分由于这一原因，HLC2 在 P100 上的整体交叉匹配时间长于在 V100 上的时间。由于 V100 和 P100 的每个流式多处理器中都有 64 个流处理器，因此一个流式多处理器一次只能执行 64 个线程。因此，图 8 显示，BlockDim.x 在 V 和 P 上均可获得峰值性能。在此配置下，每个流式多处理器都能实现最大线程并行化，同时将线程调度开销降至最低。然而，GPU 线程组织参数也可能受到内存带宽和 GPU时钟频率等其他多种因素的影响，因此应根据具体硬件配置进行调整，以获得最佳结果。此外，实验结果表明，GPU 的输入/输出问题

表 4. 使用 模拟 数据 集 比较 不同 索引 方案 的 匹配 记录 数量。 随着 数据 集 规模 增大， 边界 问题 通常 会 变得 更加 严重。 与 其他 索引 相比， 我们的 四方向 策略 准确度 最高。

数据 大小	无 索引	HTM	HEALPix	双 索引	四向 策略
10 M	665 039	662 539 (−2500)	663 914 (−1125)	665 039 (−0)	665 039 (−0)
50 M	14 366 740	14 337 115 (−29 625)	14 366 740 (−0)	14 366 740 (−0)	14 366 740 (−0)
100 M	58 553 930	58 548 430 (−5500)	58 553 680 (−250)	58 553 712 (−218)	58 553 798 (−132)
200 M	181 467 425	181 449 425 (−18 000)	181 465 925 (−1500)	181 466 248 (−1177)	181 466 947 (−478)
500 M	680 502 844	680 452 444 (−50 400)	680 501 594 (−1250)	680 501 829 (−1015)	680 502 079 (−765)

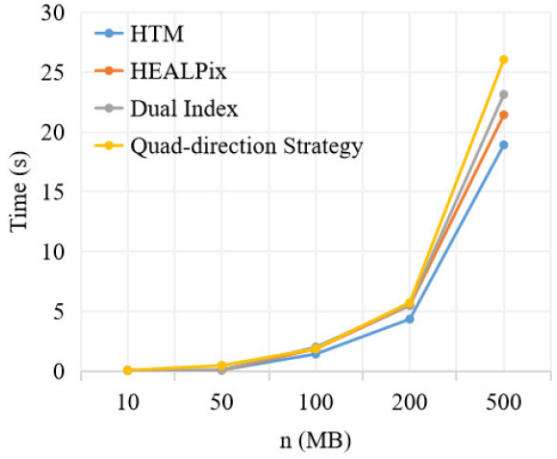

图9。不同索引下的性能。对于数据更多分布在已索引天空区域边缘的模拟数据集，BlockDim . x = 64，曲线展示了 HLC 在 V 上的处理时间。我们的四方向策略速度与次优方法相当，同时完备性有所提升，如表4所示。

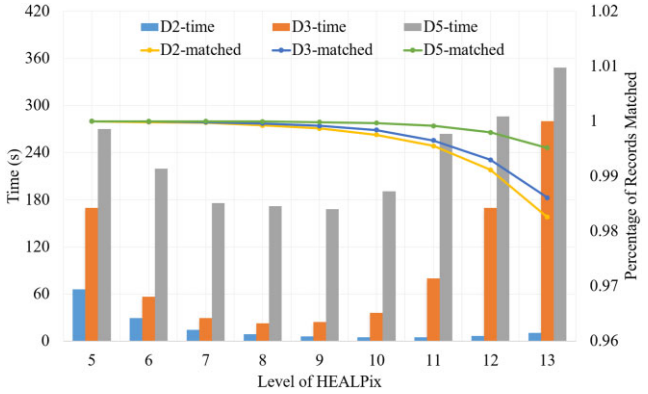

图 10. HEALPix 层级变化时的性能。在数据集 D2、D3 和 D5，以及 V100 上的块维度 . x = 64 条件下，HLC2 的处理时间和完备性随不同 HEALPix 层级而变化。每个数据集的各项结果占该数据集最大结果的百分比，用于表示匹配到的星表记录数。

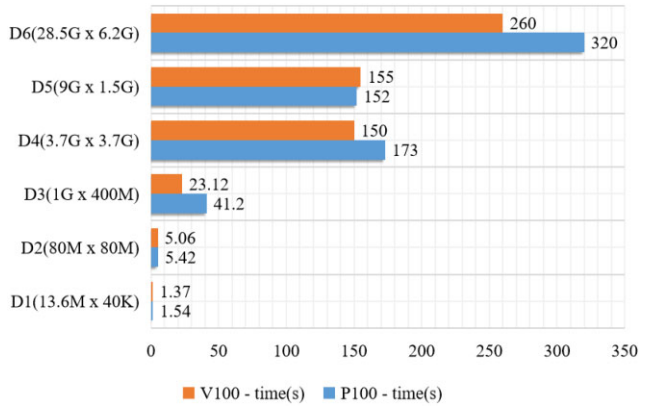

图 11. 星表规模变化时的性能。可以看出，HLC2 能够高效地获得不同规模数据集的交叉匹配结果，并能在几分钟内完成数亿规模星表的计算。

也 限制 了 计算 资源 的 利用 ， 并 阻碍 了进一步 提升 性能 ， 在 两台 服务器 上 都是 如此 ， 这一 问题 仍 需 进一步 解决 。

5.3 对 四方向 策略 的 评估

在 V100 上使用 BlockDim . x = 64 ， 对不同规模模拟数据集采用各种索引方案进行交叉匹配完备性比较。表4和图9分别展示了不同索引方案成功匹配的记录数量及所需的交叉匹配时间。表4中括号内的结果表示与不使用索引得到的准确串行结果不同的结果数量。由于串行计算需要数天时间，因此我们未绘制不使用索引时的执行时间。可以看出，当数据量小于 50M 时，基于这些索引的总计算时间相近；HTM 会带来更多边界泄漏（0.4%），而我们的四方向策略与双重索引的完备性相当。随着数据量增加，使用 HTM 的时间最短，但 HTM 带来的误差会对天文发现产生重大影响。与双重索引相比，采用四方向策略的执行时间大致相同，但数据损失减少约一半。因此，综合考虑性能和完备性，四方向策略在这四种索引中表现最佳。

5.4 总体 性能 评估 与 分析

为了 说明 HLC2 如何 随 数据 规模 扩展， 图 11 展示了表2展示了 HLC2 在数据集 D1–D6 上的运行时间。可以看出，通常每个星表包含的天体越多，耗时就越长，这表明我们的方法能够很好地适用于数据量各异的星表。同时，数据集中天体的相对分布也会影响交叉匹配的复杂度。星表中的天体较少时，并不能获得最佳加速效果，因为输入/输出和初始化的开销占了相当大的比例。随着天体数量增加，匹配计算占据了大部分时间，进而带来更好的加速效果。我们的实验表明，与图4中给出的计算结果相比，

表 5. 交叉匹配器 性能 比较 。 可以 看出 ， 我们的 HLC2 在 各种 大小 的 星表 中 处理 速度 更高 ， 尤其 是在 大型 数据 集 中 。

交叉匹配器	天体数量较少时的耗时	天体数量适中时的 耗 时 情 况	时间 对于 大量 天体 的 _久久爱
Zhao 等 人 (2009年)	–	–	32 分钟 (470 992 970 × 100 106 811)
Du 等 人 (2014年)	–	7 分钟 (946 464 × 470 992 970)	23 分钟 (470 992 970 × 100 106 811)
catsHTM （Soumagnac & Ofek 2018 ）	–	160 秒 (55 395 532 × 55 395 532)	53 分钟 (470 992 608 × 563 908 224)
C3 （Riccio 等 2017 ）	90 秒 (1000 000 × 1000 000)	800 秒 (1000 000 × 10 000 000)	–
HLC2	5 秒 (1105 627 × 1105 627)	150 秒 (61 734 000 × 61 734 000)	260 s (467 555 800 × 102 890 000)

优化后，数据移动时间减少了 38%。此外，HLC2 在 P100 和 V100 上的性能呈现出相似的趋势，这证实了两级分区策略能够很好地适应不同的环境。

为了将 HLC2 与其他先进的交叉匹配器进行比较，我们主要选择了四种具有代表性的球面位置偏差交叉匹配器，即 Zhao 等人的算法、Du 等人的算法和 C3。此外，我们还讨论了基于其他交叉匹配算法的算法，例如 catsHTM 和 Jia 等人的算法，以进行比较。它们都是运行在多 CPU 和 GPU 处理器上的先进交叉匹配算法。在这些竞争方案中，我们从性能和完备性两方面比较了 Du 等人的算法和 Zhao 等人的算法，而其他方案仅通过端到端执行时间进行测试，因为它们没有处理边界问题。

HLC2 与其他先进交叉匹配器的性能比较见表 5。我们将 HLC2 的实验结果与其论文中报告的结果进行了比较。根据星表大小，将星表分为小型、中型和大型三组，并对所有大小的星表都测试了 HLC2。在大型星表上，HLC2 的速度比 Zhao 等人的位运算算法快 7.4 倍以上；在中型和大型数据集上，分别比 Du 等人的算法快 28 倍和 5.3 倍以上，同时完备性没有明显损失。mcatCS 仅关注星表中一个记录（例如，天文学家感兴趣的 n 个天体）的匹配，其复杂度仅为 O (n)。因此，HLC2 的处理速度远快于 mcatCS。与 C3 相比，HLC2 在小型和中型星表上的速度分别约快 18 倍和 2033 倍，并且对不同数据规模的适应性更强。对于 catsHTM 等基于圆锥搜索的交叉匹配方法，HLC2 在中等数据规模下略快，并且随着数据量增长，性能优势更加明显。这些性能提升证明了我们的分区和并行化优化的有效性。与我们复杂度为 O (mn) 的方法相比，Jia 等人的算法基于圆锥搜索，适用于不同的天文场景，且没有给出端到端耗时，因此我们未将其列入表中。

6 结 论 与 未 来 工 作

天文观测数据量的持续大幅增长，使高效交叉匹配方法和工具成为必需。本文提出了一种适用于中央处理器与图形处理器异构计算环境的高效、可扩展交叉匹配框架 HLC2。用户只需输入两个待匹配的星表，即可自动生成交叉匹配结果。此外，HLC2 灵活支持用户在采用不同架构的 NVIDIA GPU 上处理不同规模的星表。该框架可直接用于融合巡天任务和天文数据中心的大型天文星表，并能在几分钟内完成包含数亿个源的星表交叉匹配。

此外，我们在基础框架上扩展了功能并优化了性能。针对中央处理器，我们提出了一种用于生成计算区块的两级分区策略，以及一种用于解决边界问题的四方向策略。我们还通过动态线程组织和输入/输出优化，利用图形处理器加速交叉匹配处理。实验结果表明，在总体执行时间方面，HLC2 优于最先进的方法，尤其适用于包含数百万条天体记录的大型星表。新颖的四方向策略在计算时间相当的情况下，可将结果损失减少约一半。HEALPix 层级、线程组织和天体分布都可能对性能产生显著影响。

未来的研究中，我们将继续探索大规模星表交叉匹配中的边界问题，以便在可接受的时间内返回所有源的完整匹配结果。同时，我们将把 HLC2 扩展到配备中央处理器和图形处理器的多计算机集群，从而提供更强的计算能力。在这种情况下，需要考虑更高层次的并行性。将 HLC2 作为核心实现的一部分后，多物理节点上的任务迁移、负载均衡和输入/输出问题也面临更多挑战，值得进一步探索。我们还计划将 HLC2 集成到中国虚拟天文台平台中，以提供可靠的交叉匹配服务。

致谢

本研究得到中国国家重点研发计划（2022YFF0711502）和国家自然科学基金（NSFC）（12273025，12133010）的资助。数据资源由中国国家天文数据中心（NADC）、中国科学院天文数据中心和中国虚拟天文台（中国虚拟天文台）提供支持。

数据 可用性

代码 和 数据 样本 可 从 https://github.com/M elody888Evan/HLC2 获取 。 本 研究 分析 所 用 数据 可 向 通讯 作者 提出 请求 获取 。

参 考 文 献

Boch T., Pineau F., Derriere S., 2014, Cds xmatch 服务文档。Boyer M., Meng J., Kumaran K., 2013，载于《2013年 IEEE 并行与分布式处理国际研讨会、研讨会及博士生论坛》。第 1097 页 Budavari T., Lee M. A., 2013，天体物理学源代码库，记录编号 ascl:1303.021 Carroll T. C., Wong P. W., 2017，载于《2017年第四十六届国际并行处理会议研讨会（ICPPW）》。第 113 页 Du P., Ren J., Pan J., Luo A., 2014，《中国科学：物理学、力学与天文学》，57，577 GAIA, 2020, edr3. https:// 159.226.170.25/mirror/ Gaia/edr3/ gaia sources/ Gaia C. et al., 2018，《天文学与天体物理学》，616

Gorski K. M., Hivon E., Banday A. J., Wandelt B. D., Hansen F. K., Reinecke M., Bartelmann M., 2005, ApJ，622，759 Gray J., Szalay A. S., Thakar A. R., Fekete G., O'Mullane W., Nieto-Santisteban M. A., Heber G., Rots A. H., 2004，预印本（ cs/0408031）Gray J., Nieto-Santisteban M. A., Szalay A. S., 2007，预印本（cs/0701171）Helou G., Madore B. F., Schmitz M., Bicay M. D., Wu X., Bennett J., 1991，载于 Albrecht M. A., Egret D. 编，《天体物理学与空间科学图书馆》第 171 卷，《天文学中的数据库与在线数据》。第 89 页 Jia X., Luo Q., 2016，载于《第二十八届国际科学与统计数据库管理会议论文集》。第 1 页 Jia X., Luo Q., Fan D., 2016，《国际并行与分布式系统会议论文集 - ICPADS》，2016-Janua，617。Koposo v S., Bartuno v O., 2006，载于《第十五届天文数据分析软件与系统会议》。第 735 页 Kunszt P., Szalay A., Thakar A., 2001，载于《MPA/ESO/MPE研讨会论文集》。加兴、柏林、海德堡，第 p631 页 Landais G., Ochsenbein F., Simon A., 2013，《天文数据分析软件与系统 XXII》，475，227 Li L., Tang D., Liu T., Liu H., Li W., Cui C., 2014，《国际并行与分布式处理研讨会论文集》，IPDPS。第 1735 页 Li B., Yu C., Li C., Hu X., Xiao J., Tang S., Cui C., Fan D., 2019, PASP，131，054501 美国国家科学、工程与医学院等，2021 Nieto-santisteban M. A., Szalay A. S., Thakar A. R., 2005, p. 1 Ochsenbein F., Bauer P., Marcout J., 2000, A&AS , 143, 23 Riahi A., Savadi A., Naghibzadeh M., 2020, Computing, 102, 2099 Riccio G., Brescia M., Cavuoti S., Mercurio A., di Giorgio A. M., Molinari S., 2017, PASP, 129, 1 Sanders J., Kandrot E., 2010, CUDA 示例：通用图形处理器编程入门。Addison-Wesley Professional。SDSS, 2015, 第13次数据发布。https:// www.sdss.org/dr13/ 数据访问/ volume/ Soumagnac M. T., Ofek E. O., 2018, PASP, 130, 1 Taylor M., 2011, 天体物理学源代码库, 1(S 01010) Wang S., Zhao Y., Luo Q., Wu C., Xv Y., 2013, 载于《2013 IEEE第九届电子科学国际会议》。p. 326 Wenger M. et al., 2000, A&AS , 143, 9 Yu C., Li K., Tang S., Sun C., Ma B., Zhao Q., 2020, MNRAS , 496, 629 Ze ˇcevi ´c P., Slater C. T., Juri ´c M., Connolly A. J., Lon ˇcari ´c S., Bellm E. C., Golkhou V. Z., Suberlak K., 2019, AJ , 158, 37 Zhang Y.-X., Zhao Y.-H., 2003, Chin. J. Astron. Astrophys. , 3, 183 Zhao Q., Sun J., Yu C., Cui C., Lv L., Xiao J., 2009, 载于《并行处理算法与体系结构国际会议》。p. 604

本文由作者准备的 T E X/L A T E X 文件排版而成。