利用 HPX 运行时系统克服分布式图算法受延迟限制的局限

Karame Panagiotis Syskakis [1] Andrew Lumsdaine [2] HartmutKaiser [1]Mohammadiporshokooh[1]
psyskakis@lsu.edu al75@[illegible]v.edu hkaiser@cct.lsu.edu kmoham6@lsu.edu

[1]路易斯安那州立大学计算机科学系、计算与技术中心，美国路易斯安那州巴吞鲁日[2]华盛顿大学计算机科学与工程学院，美国西雅图

arXiv:2603.04583v1 [cs.DC] 4 Mar 2026

摘要—大规模图处理面临诸多挑战，包括图结构的不规则性、图算法受延迟限制的特性，以及分布式执行带来的开销。尽管 Spark GraphX 和并行 Boost 图形库（PBGL）等现有框架为分布式图处理引入了抽象，但它们仍难以解决负载不均衡和同步开销等固有问题。在这项工作中，我们提出了一个分布式图库原型，并使用 NW- 图库中的 C++ 机制，利用 HPX 的分布式容器和异步构造，实现了三种关键图算法：广度优先搜索（BFS）、PageRank 和三角形计数。这些算法分别属于遍历、中心性和模式匹配类别，旨在代表不同的计算特征。我们将基于 HPX 的实现与 GraphX 和 PBGL 进行比较，结果表明，HPX 这样的高性能运行时能够构建出显著优于传统框架的算法，因为它可以利用异步执行、延迟隐藏以及共享内存中的细粒度并行。原型中的所有算法均遵循统一的执行模型，使用相同的编程抽象表达本地和远程计算，并由运行时透明地管理异步操作。该设计充分利用每个局部区域内的共享内存并行，同时重叠不同局部区域之间的通信与计算，为将这一方法扩展到更广泛的分布式图算法奠定了实用基础。

索引术语—**图算法、分布式、高性能、图计算**

I. 引言

图是用于对各个领域中的关系进行建模的强大抽象，包括科学计算、数据分析以及机器学习/人工智能。随着这些应用的规模和复杂度不断增长，它们往往会超出单个计算节点的计算与内存容量，因此必须转向高性能分布式内存并行计算方案。然而，传统的高性能计算方法，尤其是适用于稠密或稀疏线性代数问题的方法，并不能直接适用于图计算。图固有的不规则结构以及图算法的数据驱动特性会导致图算法会产生细粒度且难以预测的数据访问模式，因延迟而造成显著的运行时开销。此外，许多图算法存在固有的顺序依赖关系和严格的操作顺序，这限制了并行执行的机会，并带来了同步方面的挑战。社交网络、生物分析和推荐系统等领域中的图数据快速增长，推动了分布式图处理框架的应用。然而，现有系统仍面临通信开销、负载不均衡和资源利用率低等根本性挑战，尤其是在处理大型且不规则的图时 [1]。

NWGraph 库已成为 C++ 中灵活且高效的图计算解决方案。它提供了轻量级接口，用于在各种硬件平台上表示图并执行算法。通过将 NWGraph 与 HPX（一种分布式异步运行时系统）集成，本文提出了一种新的分布式图处理库原型，以解决现有框架的局限性。HPX 的细粒度并行、自适应资源管理以及异步执行模型使其非常适合大规模图处理。此外，本文介绍了在 HPX 上使用 NWGraph 的分布式实现，对关键图算法（包括广度优先搜索（BFS）、PageRank 和三角形计数）的实现与评估。通过这一集成，我们旨在展示 HPX 在最小化通信开销、实现负载均衡以及提高资源利用率方面的优势。我们的实验将这些基于 HPX 的分布式实现的性能与成熟的分布式框架进行对比评估，从而深入了解它们的相对效率。

II. 背景

A. 分布式图处理

分布式图的本质特征在于跨越多个地址空间的细粒度数据和控制依赖，从而将计算与通信紧密耦合。图 1 展示了与之相关的一些普遍困难，

分布式图计算，其方式是展示一个被划分到不同地址空间（“局部区域”）上的图。图中的边反映了细粒度的控制和/或数据依赖。在大型图中，许多边跨越局部性边界。具体而言，对于大型随机图，跨局部性边的数量为 $O(d|V|)$，其中 d 是平均顶点度。

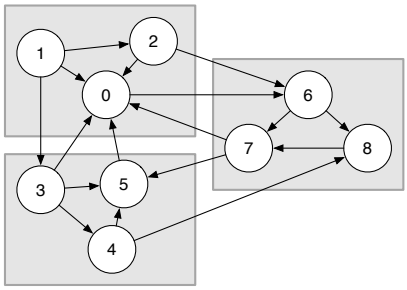


图 1：分布式图的结构。每个灰色方框表示一个独立的局部区域，拥有自己的地址空间。与许多基于分布式图的应用一样，边通常从大多数局部区域连接到其他大多数局部区域。

直接依赖这些细粒度依赖关系的算法，将承担每条跨局部区域边所带来的延迟开销。尽管延迟隐藏技术可以减少部分开销，但细粒度计算与通信相结合，再加上计算开销相较于通信开销低得多，仍然构成一项根本性挑战。此外，对科学计算应用有益的架构特性，对分布式图计算几乎没有帮助。许多图问题的局部性或数据重用很少，因此缓存和预取难以奏效。消息通常从每个进程发送到几乎所有其他进程，这会对依赖通信主要发生在本地（即仅与少数邻居顶点通信）的网络造成不利影响。

B. 图处理面临的挑战

图问题的内在特性给高效的分布式内存并行计算带来了巨大挑战。图算法通常是在理论上理想（且通常为顺序执行）的条件下，为特定问题设计的最优解法。随后，这些算法对严格数据依赖和操作顺序的规定引入了同步需求，可能会严重限制并行化的机会。此外，图算法的执行与图的结构紧密相关，而图的结构只有在运行时才已知。图通常具有不规则性；图算法的数据驱动特性意味着需要对任意位置进行细粒度数据访问，因数据访问延迟而产生显著的运行时开销。由于图算法中的计算通常非常少，通过重叠通信与计算来隐藏延迟的能力有限。

以下是分布式图处理面临的主要挑战：

- **通信开销：** 分布式图处理系统会产生显著的通信开销，因为节点之间必须交换数据以进行同步和状态更新 [2]。图的不规则结构进一步加剧了这一问题。例如，PageRank 等算法需要反复通信，因此延迟成为主要瓶颈 [1]。
- **负载均衡：** 由于图工作负载具有不可预测性，即使采用分区技术，有效的负载均衡仍然是一项挑战。虽然顶点分区和边分区等技术旨在均匀分配工作负载，但在实践中往往难以奏效，尤其是对于度分布不均的图（例如无标度网络） [3]。负载均衡不佳会产生拖尾任务，导致整体性能下降。
- **同步开销：** 许多算法（如广度优先搜索和三角形计数）通常需要频繁进行同步。这些屏障会造成延迟，尤其是在大规模分布式系统中[4]。HPX 利用细粒度任务并行和异步执行模型解决这一问题。
- **内存管理：** 在分布式节点间高效管理内存至关重要。由于需要存储邻接表和中间结果，图算法通常会占用大量内存 [5]。Pregel 和 GraphX 等传统框架高度依赖消息传递，这会进一步加重内存资源的负担。为避免内存耗尽，系统不得不将数据溢写到磁盘；由此产生的输入/输出开销增加会造成显著的性能损失。
- **容错：** 分布式系统容易发生节点故障，这可能会严重影响图处理 [6]。容错方法包括 Pregel 所采用的输入/输出密集型检查点机制，或记录结果派生过程的基于谱系的恢复机制（Spark/GraphX 等系统采用了后一种方法），而这两类方法都会将执行限制在一系列超步中。

C. 分布式图任务

图分析涵盖了针对不同计算挑战的广泛算法，尤其适用于大规模和分布式环境。在本文中，我们重点关注如 [2] 所示的三个基本类别：

遍历 算法是图分析的核心，能够系统地探索顶点之间的关系和子结构。其中，广度优先搜索（BFS）因其简单性以及在无权图中的有效性而得到广泛应用。BFS 是众多高阶算法的基础，包括最短路径计算和图连通性分析。它按层级运行，在每次迭代中激活一部分顶点（即当前前沿）；当前沿大小发生变化或顶点度差异较大时，会导致不规则并行和显著的负载不均衡[2]。此外，在分布式环境中，由于已访问顶点的冗余传播以及分区之间的同步开销，通信开销十分常见 [1]。

中心性算法旨在根据节点在网络中的位置和作用，量化其在图中的重要性。这是网页等应用中的关键方面

搜索排序、社交网络分析和推荐系统。虽然一些中心性度量（如介数和接近中心性）需要进行全局图遍历和最短路径计算，但 PageRank 主要依赖局部邻居信息，因此更适用于并行与分布式环境。不过，在处理高 度顶点时，第一节概述的挑战在 PageRank 中尤为突出，因为这些顶点往往占据了大部分计算和消息传递量。因此，人们提出了各种优化方法，以提升其在高性能计算架构上的可扩展性 [7]。

子图分析侧重于识别和分析图中的特定结构模式，这对于检测社区、理解网络模体以及评估图连通性至关重要。此类问题中的一个基础问题是三角形计数，即确定图中闭合三顶点环的数量。由于真实世界图具有不规则性，在分布式环境中高效地并行化三角形计数面临着平衡工作负载和减少冗余计算的挑战 [8]。为在保持准确性的同时加速大规模图上的三角形计数，人们提出了多种技术，例如基于边的划分和图稀疏化。

这三类算法共同涵盖了分布式图处理中常见的多种通信模式、同步需求和工作负载不规则性。在接下来的章节中，我们将这些算法作为代表性工作负载，评估 NWGraph 的数据结构与 HPX 的异步执行模型相结合，如何在多样化的分布式图任务中实现统一表达、有效负载均衡，并高效利用局部并行。

D. 相关工作

大规模图分析推动了各种图处理框架的发展。图计算库在若干关键方面各不相同，例如所采用的编程模型、灵活性和通用性，以及运行时和执行特性（CPU、GPU、分布式内存）。表 I 概述了各种图库解决方案所包含的功能。

分布式内存中的图库采用不同的通信/同步模型构建。例如，**Pregel**将数据交换限制在连续的计算步骤中，这种模型称为批量同步并行（BSP）。**GraphX** 构建于 Spark 的弹性分布式数据集（RDD）抽象之上，在图分析中采用类似的 BSP 接口。BSP 实现简化了算法开发和容错，但全局同步可能会加剧负载不均衡。BSP 模型还容易耗尽内存，因为局部区域在一个超步期间接收消息，却通常要等到一个周期结束后才处理这些消息或释放相关缓冲区。其他框架则利用细粒度并行和主动消息，实现完全异步的算法，并交错执行通信与计算。**并行 Boost 图形库（PBGL）**和

	Type	功能 DM	AS	HPR	GA
PBGL 2.0 [9]	通用图形库	●	●	◒	●
PBGL [10]	通用图形库库	●	○	○	◒
BGL [11]	通用图形库库	○	●	○	●
NWGraph [12]	通用图形图库	○	●	◒	●
GraphX [13]	图处理框架	●	○	◒	◒
PowerGraph [14]	图处理框架	●	●	◒	●
EAGM [15]	面向图的模型算法	●	◒	○	●
Pregel [16]	图处理框架	●	○	○	●
GraphBLAS [17]	图稀疏线性代数	○	●	●	●
Ligra [18]	图处理框架	○	○	○	●

表 I：各种图库解决方案的功能概览（DM：分布式内存，AS：异步，HPR：高性能运行时，GA：泛型算法）

其增强版本**PBGL 2.0** 支持基本图算法，后者利用主动消息提升可扩展性。

图框架之间的另一个区别在于它们提供的算法抽象。例如，有些框架采用以顶点为中心的方法，而其他框架则提供基于块或基于线性代数的抽象。**以顶点为中心的** [19] 框架旨在通过将计算集中于各个顶点，并实现与相邻顶点的直接通信，克服 MapReduce [20] 在迭代式、相互依赖任务中的局限性。**Pregel**和**PowerGraph**等框架通过可定制但预先定义的步骤来表达算法。具体而言，在 PowerGraph 中，所有算法都通过四种顶点操作实现：“收集”、“求和”、“应用”和“散射”。相比之下，所提出的**分布式 GraphBLAS API** [21] 在共享内存 GraphBLAS 的基础上进行了扩展，将图视为稀疏矩阵，并使用线性代数原语表达计算。

III. 技术方法

A. NWGraph

NWGraph [22]是一个以 C++ 编写的图算法泛型库，展示了如何利用该语言及其标准库现有的功能，有效支持图算法和数据结构。它通过函数模板实现泛型算法，并将类型要求表示为 C++概念。其设计围绕着根据各算法的具体要求，在算法接口处定义类型要求展开。借助 C++概念，NWGraph 系统地消除了不必要的约束，并将最小类型要求组织成连贯的分类体系。这些

概念定义了该库中算法的接口，使其能够与其他独立开发的组件无缝组合。NWGraph 的另一项基础设计决策是，将图所呈现的抽象接口定义为**范围的范围**：外层范围是顶点的范围，内层范围则是各顶点邻接边的范围。

概念分类：通过使用 C++ 概念来定义邻接表和边列表，NWGraph 能够扩展或创建新的图算法，而不必受限于特定的底层数据表示。将图定义为由前向范围构成的随机访问范围后，数据结构要求便可在多种算法间通用。这种泛化有助于设计出在内存和数据结构类型方面具有灵活性的 API，并使其能够适应异构系统的扩展需求。开发者只需遵循指定的概念要求，即可集成自定义数据结构，从而将其融入更广泛的生态系统，并支持不同图处理工具之间的交叉兼容。

可复用的数据访问模式：NWGraph 的基于范围的方法支持实现丰富的范围适配器，从而将图的一种表示形式转换为另一种。这种方法无需使用访问者对象，使其能够无缝、高效地适配特定算法。例如，提供基于遍历的视图（如广度优先搜索和深度优先搜索范围），可简化不同图结构上的遍历操作。

并行执行：NWGraph 利用现代 C++执行策略实现算法的完全并行化。它最初使用英特尔的线程构建块（TBB），之后改用 HPX，以改进线程管理并优化工作负载划分。这种方法无需手动应用加锁机制和粗粒度互斥锁，同时支持线程安全数据结构，并能更精细地控制并发粒度。

NWGraph 构建模块坚实的理论基础，使其能够全面涵盖各类必要的图操作，尤其适用于高度不规则且由数据驱动的计算。

B. HPX 运行时系统

HPX 运行时系统 [23]–[25] 具备多项特性，有助于实现第 II 节所列的设计目标：异步执行、通信与消息批处理、动态划分与负载均衡，以及代码和性能的可移植性。HPX 的首要设计目标是创建先进的异步并行多任务运行时系统（AMT），为可扩展应用提供坚实基础，同时尽可能兼顾效率、可移植性和模块化 [26]。HPX 提供以下基础设施：

主动全局地址空间（AGAS）：HPX 在应用程序运行的所有局部区域中实现一个单一的全局地址空间。AGAS 允许将所有全局对象透明地迁移到不同的局部区域，而无需更改其全局地址，从而满足分布式图数据结构进行动态负载均衡的需求。

线程与线程管理： HPX 线程管理器实现了基于工作队列的执行模型，支持工作窃取。已准备好执行的任务会封装在 HPX线程 中，并被调度运行。HPX线程 轻量且由线程管理器在用户态以协作式（非抢占式）方式调度，从而显著降低上下文切换和调度开销。

消息传输与消息管理：HPX 中任何跨局部性的消息传递都基于独立的网络消息（称为 Parcel）。它们是主动消息的扩展形式 [27]，并支持异步的消息驱动、分布式控制流以及动态资源管理。Parcel 既可用于将工作移至数据所在位置，也可用于将小块数据收集回调用方。

本地与全局同步对象：这些是对不同功能的抽象，用于事件驱动的创建、异步流控制的组织、保护数据结构免受竞态条件影响、自动进行事件驱动的即时工作调度，以及基于 Future 抽象 [28]的同步。除专用同步原语外，HPX 还提供 C++标准所规定的全部此类抽象（互斥锁、条件变量、信号量等），可用于与 HPX 线程进行协作式同步，而不会阻塞底层内核 (p)thread 的推进。

性能计数器基础设施：HPX 提供统一且可扩展的 API，用于提取有关运行时系统本身以及应用程序行为的性能相关信息。这使得预定义的性能指标以及用户或应用程序定义的性能信息均可按需提取，以便做出运行时自适应决策，从而改变算法和系统参数。

C. 我们的方法

在本文中，我们利用 NWGraph 和 HPX 中的现有功能，并将其扩展以满足分布式图计算的需求。NWGraph 的基本方法被扩展到分布式领域，目标是创建一个表达能力强、通用且高性能的分布式图计算框架。

分布式范围的范围：我们创建遵循 NWGraph 基于范围的方法的分布式图数据结构。通过将底层存储替换为分布式 HPX 容器（从 std::vector 到 hpx::partitioned_vector），抽象接口仍然是范围的范围。这种抽象保留了统一的算法接口，使为本地 NWGraph 容器编写的通用图算法能够以最少的适配在分布式表示上实例化。对于高性能算法实现所必需的局部性感知迭代，则是对范围接口的一种细化。

并行与分布式执行：在并行方面，我们利用 HPX 的并行算法和异步/Future，它们都

```
模板 <邻接_列表_图 Graph>
大小_t 三角形_计数(Graph常量&G) {
    大小_t 个三角形 = 0;
    hpx::for_each(hpx::execution::seq, G.begin(), G.end(),
        [&](自动 u_邻居顶点) {
            对于 (自动 边_uv : u_邻居顶点) {
                自动 变量 = 目标顶点(G, 边_uv);
                三角形 += 交集_大小(u_邻居顶点, G,
                    v);
            }
        });
    返回 三角形;
}
```

列表 1：朴素顺序三角形计数算法

与 C++ 标准库中的对应功能十分相似。HPX 还扩展了标准 C++，提供分布式数据结构、组件、远程动作、用于跨局部性异步操作的分布式future，以及分布式集合操作；这些能力对于分布式内存图算法至关重要。

IV. 实现

对于我们针对图的分布式数据结构开发的初始原型，我们将 NWGraph 数据结构与 HPX 的分区向量实现相结合。我们利用该原型实现了三角形计数、PageRank 和广度优先搜索（BFS）算法的分布式版本。尽管这些算法通常都可以并行化，但在分布式系统中，其性能会受到较高通信开销、负载不均衡以及数据局部性方面挑战的影响。HPX 的异步远程过程调用基础设施非常适合我们的需求。

NWGraph 除了其他格式外，还实现了图的压缩稀疏行（CSR）表示法。CSR 格式将顶点和边分别存储在不同的数组中，数组中的索引分别对应顶点或边的标识符。边数组按照每条边的源顶点排序，但只包含边的目标顶点。顶点数组存储指向边数组的偏移量，给出从每个顶点出发的第一条出边的偏移量。在我们的原型中，我们使用 hpx::partitioned_vector 作为 NWGraph 的 CSR 数据结构的数据存储。这样可以以分布式方式存储图的顶点和边，使其分布在应用运行所在的各个局部区域（计算节点）上。随后对边数组进行分区，使得每个顶点的出边列表都可以在本地遍历。

图数据在内部使用这种分区稀疏结构表示，其中每个分区对应于由某个局部区域拥有的一组顶点和边。尽管分区向量提供了图的全局逻辑视图，但底层数据分布在非连续内存区域中。这种表示方式天然契合过度分解的执行模型，使各个独立分区能够使用统一的本地计算内核并发处理。

HPX 的 hpx::partitioned_vector 是一种表示分段数组的分布式数据结构，数组的各个部分位于不同的局部区域上的分布式应用程序中。它公开的 API 与 C++std::vector 的 API 类似，因此与 NWGraph 的集成十分简单。因此，任何适用于 std::vector 的 C++算法，也都能直接用于 hpx::partitioned_vector。例如，NWGraph 的朴素顺序三角形计数算法应用于分布式图实例时仍能正常运行（见清单 1）。然而，这种直接的方法在分布式环境中会导致性能显著下降，因为逐元素访问可能需要执行同步网络操作。清单 2 所示的分布式算法通过异步启动对图中每个分区的三角形计数，缓解了这一影响。

HPX 实现了完整的 C++ 标准（并行）算法集；在分区向量上调用时，这些算法还提供了利用分段迭代器的特殊重载 [29]。这些算法会异步地在各个独立分区上调度操作，并在正确的局部性上并发执行。对于我们的原型，我们使用 HPXsegmented 算法基础设施实现了分布式三角形计数、广度优先搜索和 PageRank 算法，其性能和扩展性结果见第五节。

本文中的所有图算法都遵循统一的执行模型，在该模型中，本地和远程计算采用相同的编程抽象来表达。对本地拥有的图分区执行的操作会直接执行，而针对远程分区的操作则以异步远程动作的形式发出，并返回期值。从算法的角度来看，这两种情况都会调用相同的本地计算例程，而异步处理则由 HPX 运行时透明完成。同步会推迟到全局归约点，从而确保远程操作进行期间计算能够持续推进，而不会发生阻塞。

为缓解负载不均衡并隐藏通信延迟，每个局部性区域中的图都会被分解为比可用处理器核心更多的分区。这种有意的过度分解使 HPX 的工作窃取调度器能够在其他任务挂起并等待远程结果时，继续执行就绪任务。因此，因等待通信而无法取得进展的线程不会使系统停滞，可用核心也能持续处理彼此独立的本地工作。这种设计明确利用了每个局部性区域内的共享内存局部并行，使多个图分区能够通过 HPX 的轻量级线程和工作窃取调度器并发处理。

此外，处理资源被明确划分为用于本地计算和执行传入远程请求的部分。预留一部分核心用于处理远程动作，可确保分布式操作及时取得进展，避免纯粹的本地工作负载导致饥饿。尽管当前实现采用静态分配，但这为在持续的通信压力下保持响应能力提供了一种实用机制。

尽管未实现显式消息批处理，延迟异步远程动作的使用仍会隐式地

```
1  模板 <邻接_列表_图 Graph>
2  大小_t 分布式_三角形_计数(图常量& G, 大小_t
       首个, 大小_t 最后) {
3    大小_t 三角形 = 0;
4    std::vector<hpx::future<大小_t>> 计数;
5
6    hpx::for_每个(hpx::execution::seq,
7        G.begin() + 第一个, G.begin() + 最后一个,
8        [&](auto u_邻居顶点) {
9          for (auto edge_uv : u_邻居顶点) {
10           auto v = 目标顶点(G, edge_uv);
11           if (顶点_位于_此_局部性_(G, v))
12             三角形 += intersection_size(u_邻居顶点, G, v);
13           else
14             counts.push_back(hpx::async(
15               intersection_size,
16               顶点_局部性(G, v),
17               u_邻居顶点, hpx::ref(G), v));
18         }
19       });
20   返回 std::transform_reduce(
21
22       counts.begin(), counts.end(), 三角形,
23       [](hpx::future<size_t> f) { 返回 f.get(); },
24       std::plus<>());
25 }
```

列表 2：朴素分布式三角形计数算法

实现一种有限形式的消息合并。在较短时间窗口内发出的多个远程请求可在传输前由运行时合并，从而降低通信开销，而无需算法级缓冲或同步。

分布式三角形计数依赖一种稍微复杂一些的算法 [30]（另见列表 2）。

本质上，如果邻居顶点的数据与原始顶点位于同一局部性上，我们就在本地计算当前顶点的邻居顶点与其邻居顶点的邻居顶点之间的交集。此时，执行交集运算所需的所有信息都位于当前局部性上。否则，我们会在该邻居顶点所在的局部性上异步执行交集运算，同时传递原始顶点的邻居顶点列表。这样，另一个局部性就能获得执行完全本地交集运算所需的全部信息。请注意，函数 hpx::async 会向指定局部性执行异步远程过程调用（见列表 2，第 14 行）。它会立即返回一个 hpx::future<size_t> 实例；该对象表示远程操作识别出的三角形数量，即使结果尚未返回也是如此。只有在该 future 上调用 f.get() 时，执行线程才可能暂停，直到结果可用（见列表 2，第 23 行）。

清单 2 中所示的代码会被异步调度，在图的每个分区上并发运行（即在图的顶点范围 [first, last) 上）。这里展示的是算法的顺序版本（见清单 1，第 4 行）；不过，将 hpx::execution::par 用作 hpx::for_each 的第一个参数，就能并行执行循环。值得注意的是，将 NWGraph 的数据结构和 API 与 HPX 的分区向量及异步执行管理 API 结合使用，可以编写出一种分布式算法，它与朴素的顺序本地算法（见清单 1）非常相似，同时还能扩展

```
1  使用回复 =std::tuple<顶点_id, double>;
2  回复计算_贡献（顶点_id u, 顶点_id v
3                 分布式数组& pr, 分布式数组& 度)
4  {
5      返回 {u, pr[v]/度[v]};
6  }
7
8  // 在所有者上运行 [first,last)
9  双精度pagerank_迭代(常量分布式图& G,
10                 顶点_标识 首个, 顶点_标识 末个,
11                 分布式数组& PageRank 值, 分布式数组& 度,
12                 分布式数组& 累加值, 双精度 阻尼系数)
13 {
14     std::vector<hpx::future<空>> 待处理;
15     对于 (顶点_标识 u = 第一个; u != 最后一个; ++u) {
16         累加值[u] = 0.0;
17         对于 (自动 边_uv : G[u]) {
18             顶点_it v = 目标顶点(G, 边_uv);
19             if (is_local(v)) {
20                 累加值[u] += PageRank 值[v] / 度[v];
21             } else {
22                 // 向 v 的所有者请求贡献值
23                 pending.push_back(
24                     hpx::async(计算_贡献,
25                         顶点_局部性(G, v),
26                         u, v, pr, deg)
27                     .then([&](auto&& f){
28                         auto [u, 贡献] = f.get();
29                         原子_加(累加值[u], 贡献);
30                     });
31                 );
32             }
33         }
34     }
35     hpx::wait_all(pending);
36     std::size_t N = G.size();
37     double base = (1.0 - 阻尼系数) / N;
38     双精度 增量 = 0.0;
39     for (顶点_id u = first; u != last; ++u) {
40         double old = PageRank 值[u];
41         PageRank 值[u] = 基值 +阻尼系数 * 累加值[u];
42         差值 += abs(PageRank 值[u] - old);
43     }
44     返回 差值 ;
45 }
```

代码清单 3：分布式 PageRank 迭代

良好（见第五节）。我们还想指出，所示算法具有**通用性**和**统一性**，因为它也适用于其他内部图表示（除所用的 CSR 格式之外）。

分布式PageRank依据 Brin 和 Page 提出的随机冲浪者模型的公式实现 [31],其中顶点 u的 PageRank 值计算如下：

$$PR_{\text{new}}(u) = \frac{1-d}{N} + d \cdot A(u)$$

其中，d是阻尼因子（通常设为 0.85），N 是节点总数，并且

$$A(u) = \sum_{v \in N_{\text{in}}(u)} \frac{PR(v)}{\deg(v)}.$$

其中，N_{in}（u）表示 u的入邻居集合，而deg（v）表示顶点 v 的出度。

我们的实现的简化版本如代码清单 3 所示。除分布式邻接表外，该算法还维护：i) 分布式 PageRank 向量 PR（每个顶点对应一个值）；ii) 分布式出度向量deg（每个顶点对应一个值）；以及 iii) 分布式累加器向量 A

用于在每次迭代期间收集传入贡献。每个拥有图分区的计算节点都会启动一次迭代，并执行一个 PageRank 松弛步；对于每个由本地所有的顶点 u，工作线程遍历 u的邻接表，并将邻居贡献累加到 $A(u)$ 中。如果邻居顶点 v 由本地所有，则直接根据本地状态计算贡献 $^{PR_{(v)}}{}_{\deg(v)}$，并将其加到 $A(u)$ 中。否则，会在 v的所有者上启动异步远程动作，以计算并提供根据远程顶点 v 得出的贡献（见代码清单 3，第 24 行）。这样，计算会被安排在拥有数据的节点上执行（“计算向数据移动”），从而最大限度地减少远程内存流量。最后，我们想指出 .then 的用法，它用于为异步远程操作附加延续。这样，远程贡献一到达本地节点，本地数据就会立即更新。

分布式广度优先搜索 的实现方式与一种遍历类似，该遍历会构建一棵以用户指定的源顶点为根的分布式父树。我们此前的工作 [32] 已介绍了这一分布式广度优先搜索算法的完整实现，包括代码以及在不同图数据集上的实验评估。

V. 实验测量与结果

为了测量各实现的性能，我们在 GAP数据集[33], 中的样本图以及使用 Erd¨os-R´enyi“urand”模型生成的合成图上测试了算法。urand 图的规模各不相同，平均顶点度为 32（例如，urand20 有 2^{20} 个顶点和 2^{25} 条边）。程序使用 GCC 14.2.0 编译，测量工作在两个不同的集群上进行：第一个集群由 16 个节点组成，每个节点配备两颗 20 核英特尔 (R) Xeon(R) Gold 6148 CPU 处理器和 92 GB 内存。第二个集群同样由 16 个节点组成，每个节点配备两颗 24 核 AMD EPYC 7352 24-Core 处理器和 264 GB 内存。

图 2 展示了随着计算节点数量增加，三种算法的执行时间。我们将 BFS 和 PageRank 实现的性能与并行 Boost 图形库（PBGL）提供的参考分布式实现进行比较（PBGL 未提供参考三角形计数实现）。HPX 实现的分布式广度优先搜索（蓝线）表现出良好的扩展性，在 8 个和 16 个节点上，其性能比 PBGL 实现高出一个数量级（图 2b）。请注意，从 1 个节点增加到 4 个节点时，HPX 实现呈现出超线性扩展，因为分布式图的各部分开始能够装入 CPU缓存。分布式 PageRank 的扩展性通常没那么好，HPX 实现仅在较少机器上超过 PBGL 的性能（图 2c）。在许多节点上运行时，两种实现的性能似乎趋于一致，这表明执行时间主要受通信延迟和同步开销影响。我们还展示了 HPX 分布式三角形计数实现的结果，其中我们高度异步的方法似乎在两台机器上都能扩展到 4 个节点（图 2a）。

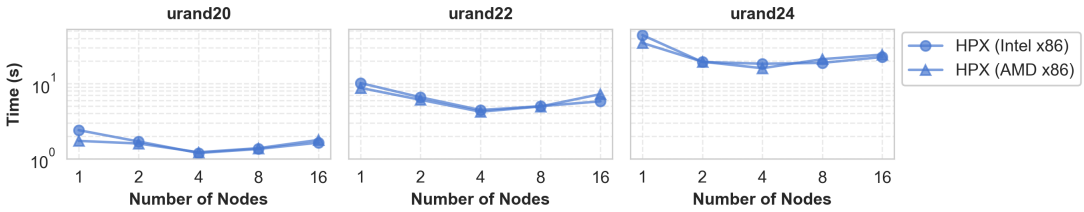
如观察到的那样，有几个因素会影响我们实现的扩展特性。首先，随着图分布到更多局部区域，远程顶点与本地顶点的比例会增加。这会导致更多计算涉及远程操作，而非在本地完成。对于较小的图，局部并行的效果也可能下降，因为每个线程的工作负载过小，无法摊销并行化开销。相反，数据分布可能对内存和缓存行为产生积极影响——每个计算节点只负责图的一个小子集，因此更有可能命中缓存。

在实验中，我们经常发现 PBGL 会遭遇内存耗尽，这解释了图 2 中较大图缺少数据的原因。图 3 对比了 PBGL 和 HPX 实现的每节点内存使用量，每个节点使用 4 个核心。在 PBGL 中，局部并行通过在独立地址空间中运行多个进程来实现，这些进程使用消息传递（MPI）进行通信。这会导致大量数据重复，其规模与并行度成正比。对远程（“幽灵”）图数据过于积极的缓存也会增加内存占用。相比之下，HPX 依赖节点内的共享内存并行，因此随着核心数增加，其内存消耗几乎保持不变。

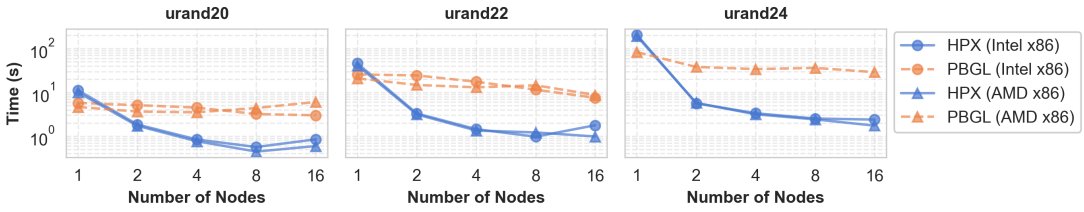
在图 4 中，我们还将分布式广度优先搜索和分布式 PageRank 算法的实现与 Spark GraphX 库进行了比较，使实验得以扩展到 GAP 数据集中的更大规模图。GAP-urand 和 GAP- kron 是合成生成的图，分别包含 2^{27}（∼128 M）个顶点和 2^{32} （∼4B）条边。与 Erd¨os-R´enyi “urand”模型不同，Kronecker 图会产生重尾度分布，更接近许多真实网络中的情况（少数高阶节点和大量低阶节点）。GraphX 也依赖独立进程实现并行，并且由于 Spark 基于 RDD 的容错计算模型，输入/输出开销更大。我们的内存异步实现速度更快，通常快出几个数量级。在较大的图上，GraphX 往往会超出允许的集群时间限制，而算法中间数据也常常耗尽内存，甚至超过磁盘存储限制。

观察到的性能提升源于 HPX 支持的三项设计选择：利用异步远程函数调用重叠通信与计算、消除全局同步屏障，以及通过工作窃取实现细粒度共享内存并行，从而高效利用 CPU 核心和内存带宽。这些结果凸显了 HPX 作为用于高性能分布式图计算的 C++运行时的优势。

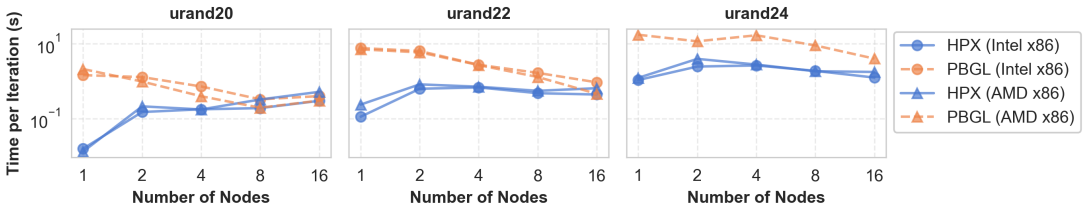
图 2 和图 4 中的结果来自一些尚未穷尽所有可能优化方案的原型。例如，可以在代码中或由运行时合并消息；可以自适应地选择分区大小，以实现更好的负载均衡；还可以在节点间重新分布分区和顶点，以改善负载均衡。对于本地任务，



(a) 分布式三角形计数



(b) 分布式广度优先搜索



(c) 分布式PageRank

图 2：分布式图算法在 Erd¨os-R´enyi 均匀随机（urand）图上的强扩展性表现 hs.

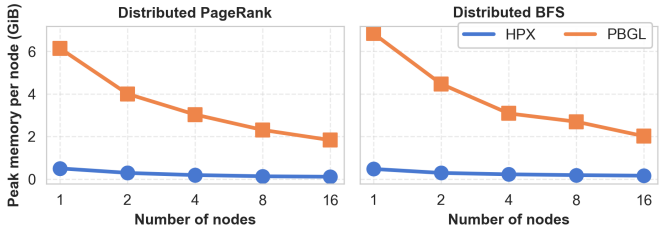


图 3：每个节点使用 4 个 MPI 进程（PBGL）/ 4 个工作线程（HPX）时的每节点内存使用量对比。

分块技术和缓存感知遍历可以显著提高缓存利用率 [34]。为简单起见，我们也省略了任何专门的图划分（例如，尽量减少跨分区边）。

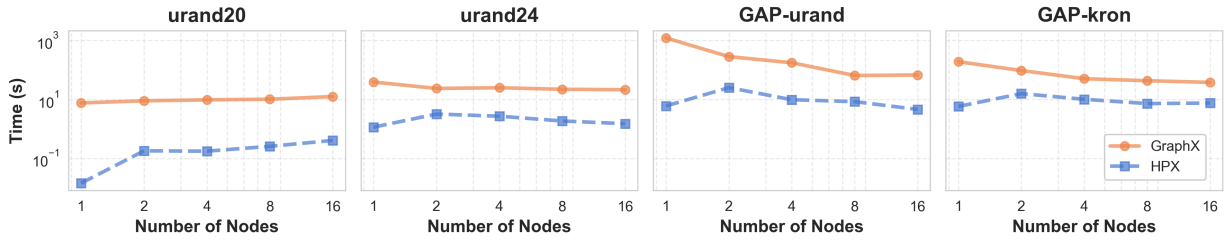
VI. 未来工作

本文介绍的原型实现和实验结果揭示了若干富有前景的未来研究方向。尤其是，尽管我们当前的设计已受益于异步执行、考虑局部性的计算以及有限形式的消息聚合，但系统地探索运行时自适应仍是重要的下一步。许多对性能至关重要的参数，例如消息合并阈值、分区大小和顶点局部性放置，目前仍是固定的或通过启发式方法选定的。动态调整这些参数——会引出若干尚待解决的研究问题，包括：在发起合并通信之前，应累加多少条消息？怎样的分区大小最能兼顾通信开销与负载灵活性？应如何分配顶点和分区，才能均衡异步远程操作的负载？此外，某些外层循环的并行化是否总有益处，仍是一个悬而未决的问题；当计算已分布在各个分区上时，顺序执行是否反而更可取？

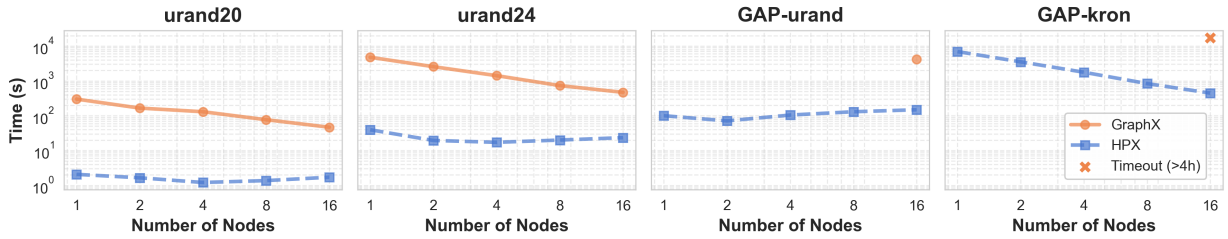
此外，我们先前的工作已证明，在 HPX 的并行算法中，运行时自适应对于处理单元选择和块大小控制是有效的 [35], [36]。在该工作中，自适应执行器会根据运行时测量结果，动态调整活跃核心的数量和工作分配的粒度，从而在各种工作负载下持续提升性能。将这些自适应机制集成到我们分布式图实现中，是对当前工作的自然且颇具前景的扩展。

另一个重要方向是在我们的分布式图框架中集成 GPU加速。尽管这需要扩展 HPX 的执行和数据管理基础设施，以更好地支持异构设备，但我们预计，GPU卸载将进一步提升计算密集型图计算核的吞吐量。

最后，我们预计，更复杂的分布式图算法



（a）分布式PageRank



（b）分布式三角形计数

图 4：相对于 Spark GraphX 实现的强扩展性，包括 GAP-kron 和 GAP-urand 图（∼128M 个顶点和 ∼4B 条边）。

可以通过与教科书中的表述十分相似的方式组合起来，从而降低非专家的使用门槛。这一观察促使我们在未来探索专门的基于大语言模型的工具，以协助生成和定制分布式图代码；我们的广度优先搜索和 PageRank 实现简洁易懂，也为这一想法提供了支持。

VII. 结论

本文提出了一个构建于 HPX 运行时系统和 NWGraph 数据结构之上的分布式图分析框架。我们聚焦于三类基本的图算法——遍历、中心性和子图分析——并采用一种与其顺序表述紧密对应的统一编程模型，实现了广度优先搜索、PageRank 和三角形计数的分布式版本。

这项工作的一项关键贡献在于表明，通过将分区数据结构与异步远程任务执行相结合，分布式图算法可以在 C++ 中以通用且易于维护的方式加以表达。该方法能够在保持局部性并避免全局同步屏障的同时，将计算移动到数据所在位置。总体而言，这项工作表明，HPX 可以为可扩展的分布式图分析提供灵活而高效的基础，从而为未来关于自适应执行策略、异构架构以及分布式图算法自动生成的研究开辟道路。

参考文献

[1] A. Lumsdaine、D. Gregor、B. Hendrickson 和 J. Berry，《并行图处理的挑战》，*Parallel Processing Letters*，第 17 卷，第 01 期，第 5–20 页，2007 年。 [在线]。网址：https://doi.org/10.1142/ S0129626407002843

[2] L. Meng、Y. Shao、L. Yuan、L. Lai、P. Cheng、X. Li、W. Yu、W. Zhang、X. Lin 和 J. Zhou，“大规模图上的分布式图算法综述”，2024 年。 [在线]。可访问：https://arxiv.org/abs/ 2404 06037 。C. Gui、L. Zheng、B. He、C. Liu、X. Chen、X. Liao 和 H. Jin，“图处理加速器综述：挑战与机遇”，*CoRR*，第 abs/1902.10130 卷，2019 年。 [在线]。可访问：http://arxiv.org/abs/ 1902 10130 。V. Kalavri、V. Vlassov 和 S. Haridi，“分布式图处理的高级编程抽象”，*IEEE Transactions on Knowledge and Data Engineering*，第 30 卷，第 2 期，第 305–324 页，2018 年 2 月。 [ ] 10 1109 .2017 2762294 在线。可访问：http://dx.doi.org/ 。 /TKDE. 。S. Sahu、A. Mhedhbi、S. Salihoglu、J. Lin 和 M. T. Ozsu，“大图的普遍存在及图处理面临的意外挑战：扩展综述”，*The VLDB Journal*，第 29 卷，第 2–3 期，第 595–618 页，2019 年 6 月。[在线]。可访问：http://dx.doi.org/10.1007/s00778-019-00548-x [6] H. W. Y. Adoni、T. Nahhal、M. Krichen、B. Aghezzaf 和 A. Elbyed，“并行和分布式系统中图结构数据划分与处理的当前挑战综述”，分布式并行数据库，第 38 卷，第 2 期，第 495–530 页，2020 年 6 月。 [在线]。可访问：https://doi.org/ 10 1007 10619 019 07276 9 。 /s - - - S. Luo、X. Wu 和 B. Kao，“降低轮次复杂度的分布式 PageRank 计算”，信息科学，第 607 卷，第 109–125 页，2022 年。 [在线]。可访问：https://www.sciencedirect.com/ science/article/pii/S 0020025522005461 A. Azad、A. Buluc, 和 J. Gilbert，“使用矩阵代数进行并行三角形计数与枚举”，载于 *2015 年 IEEE 国际并行与分布式处理研讨会研讨会论文集，丛书 IPDPSW ’15*。美国：IEEE计算机学会，2015 年，第 804 811 [ ] 10 1109 2015 75 – 2 0 页。在线。可访问：https://doi.org/ 。 /IPDPSW. 。N. Edmonds 和 A. Lumsdaine，“并行 Boost 图库：作为并行图计算统一模型的主动消息”，载于大规模图分析。Chapman and Hall/CRC，2022 年，第 459–481 页。[10] D. Gregor 和 A. Lumsdaine，“并行 BGL：用于分布式图计算的通用图库”，面向对象科学并行计算（*POOSC*），2005 年 1 月。[11] J. G. Siek、L.-Q. Lee 和 A. Lumsdaine，*Boost* 图库。Pearson Education，2001 年 12 月。[12] A. Lumsdaine、L. D’ Alessandro、K. Deweese、J. S. Firoz、T. Liu、S. McMillan、J. P. Ratzloff 和 M. Zalewski，“NWGraph：C++20 中的通用图算法与数据结构库”，载于欧洲面向对象编程会议，2021 年。 [在线]。可访问：https://api.semanticscholar.org/CorpusID: 249961751 [13] R. S. Xin、J. E. Gonzalez、M. J. Franklin 和 I. Stoica，“Graphx:

一个基于 Spark 的弹性分布式图系统”，载于第一届图数据管理经验与系统国际研讨会，丛书 GRADES ’13。美国纽约州纽约市：美国计算机协会，2013 年。[在线]。可访问：https://doi.org/10.1145/2484425. 2484427[14] J. E. Gonzalez、Y. Low、H. Gu、D. Bickson 和 C. Guestrin，“Powergraph：自然图上的分布式图并行计算”，载于第 10 届 USENIX 操作系统设计与实现会议论文集，丛书 OSDI’12。美国：USENIX协会，2012 年，第 17–30 页。[15] T. Kanewala、M. Zalewski 和 A. Lumsdaine，“抽象图机”，arxiv，2016 年 4 月。[16] G. Malewicz、M. H. Austern、A. J. Bik、J. C. Dehnert、I. Horn、N. Leiser 和 G. Czajkowski，“Pregel：一种用于大规模图处理的系统”，载于 2010 年 ACM SIGMOD 数据管理国际会议论文集，丛书 SIGMOD ’10。美国纽约州纽约市：美国计算机协会，2010 年，第 135–146 页。

[ ] 10 1145 1807167 1807184 在线。可访问：https://doi.org/ 。 / 。[17] J. Kepner、D. Bader、A. Buluc、J. Gilbert、T. Mattson 和 H. Mey-erhenke，“图、矩阵与 GraphBLAS：七个充分理由”，Procedia Computer Science，第 51 卷，第 2453–2462 页，2015 年。[18] J. Shun 和 G. E. Blelloch，“Ligra：用于共享内存的轻量级图处理框架”，载于 ACM SIGPLAN 并行编程原理与实践研讨会，PPoPP ’13，中国深圳，2013 年 2 月 23–27 日，A. Nicolau、X. Shen、S. P. Amarasinghe 和 R. W. Vuduc 编。ACM，2013 年，第 135–146 页。

[ ] 10 1145 2442516 2442530 在线。可访问：https://doi.org/ 。 / 。[19] R. R. McCune、T. Weninger 和 G. Madey，“像顶点一样思考：大规模分布式图处理中以顶点为中心的框架综述”，ACM计算综述（CSUR），第 48 卷，39，2015 年 2 月 1 日，第 – 页。[20] J. Dean 和 S. Ghemawat，“MapReduce：大型集群上的简化数据处理”，Commun. ACM，第 51 卷，第 1 期，第 107–113 页，2008 年 1 月。

[ ] 10 1145 1327452 1327492 在线。可访问：https://doi.org/ 。 / 。[21] B. Brock、A. Buluc、T. G. Mattson、S. McMillan、J. E. Moreira、R. Pearce、O. Selvitopi 和 T. Steil，“分布式 GraphBLAS API 的若干考量”，载于 2020 年 IEEE 国际并行与分布式处理研讨会（IPDPSW），2020 年，第 215–218 页。[22] “NWGraph：西北图库”，2024 年。 [在线]。可访问：https://github.com/pnnl/NWGraph [23] H. Kaiser、P. Diehl、A. S. Lemoine、B. A. Lelbach、P. Amini、A. Berge、J. Biddiscombe、S. R. Brandt、N. Gupta、T. Heller 等，“HPX：用于并行与并发的 C++标准库”，Journal of Open Source Software，第 5 卷，第 53 期，第 2352 页，2020 年。[24] “HPX：用于并行与并发的 C++ 标准库”，2024 年，https://hpx.stellar-group.org。 [在线]。可访问：https://hpx.stellar-group.org [25] H. Kaiser、M. Simberg、B. Adelstein Lelbach、T. Heller、A. Berge、J. Biddiscombe、A. Reverdell、A. Bikineev、G. Mercer、A. Schaefer、K. Huck、A. Lemoine、T. Kwon、J. Habraken、M. Anderson、S. Brandt、M. Copik、S. Yadav、M. Stumpf、D. Bourgeois、A. Nair、D. Blank、G. Gonidelis、R. Stobaugh、N. Gupta、S. Jakobovits、V. Amatya、L. Viklund、P. Diehl 和 Z. Khatami，“STEllAR-GROUP/hpx：HPX V1.10.0：用于并行与并发的 C++ 标准库”，2024 年 8 月。在线。可访问：https://doi.org/ 。 /zenodo。N. Wu、I. Gonidelis、S. Liu、Z. Fink、N. Gupta、K. Mohammadipor-shokooh、P. Diehl、H. Kaiser 和 L. V. Kale，“使用任务基准测试量化 Charm++ 和 HPX 的开销”，载于《Euro-Par 2022：并行处理研讨会》，J. Singer、Y. Elkhatib、D. Blanco Heras、P. Diehl、N. Brown 和 A. Ilic 编。Cham：Springer Nature Switzerland，2023 年，第 5–16 页。[27] T. von Eicken、D. E. Culler、S. C. Goldstein 和 K. E. Schauser，“主动消息：一种集成通信与计算的机制”，载于第 19 届年度国际计算机体系结构研讨会论文集，丛书 ISCA ’92。美国纽约州纽约市：美国计算机协会，1992 年，第 256–266 页。

[ ] 10 1145 139669 140382 在线。可访问：https://doi.org/ 。 / 。[28] D. P. Friedman 和 D. S. Wise，“Cons 不应计算其参数”，载于自动机、语言与编程国际研讨会，1976 年。 [在线]。可访问：https://api.semanticscholar.org/CorpusID: 40986838[29] M. H. Austern，“分段迭代器与层次算法”，载于《泛型编程国际研讨会精选论文》。德国柏林、海德堡：Springer-Verlag，1998 年，第 80–90 页。

[30]T. A. Kanewala、M. Zalewski 和 A. Lumsdaine，《分布式共享内存并行三角形计数》，载于高性能科学计算平台会议（*PASC 2018*）论文集，瑞士巴塞尔，*2018*年*7*月*2*日至*4*日。ACM，2018年，第 5:1–5:12 页。

[ ] 10 1145 3218176 3218229 在线。可访问：https://doi.org/ 。 / 。[31] S. Brin 和 L. Page，《大规模超文本网络搜索引擎的剖析》，计算机网络与 *ISDN* 系统，第 30 卷，第 1–7 期，第 107–117 页，1998年。[32] K. Mohammadiporshokooh、P. Syskakis 和 H. Kaiser，《使用 NWGraph 和 HPX 对分布式图算法的初步评估》，

2026 [ ] 2601 18158 。在线。可访问：https://arxiv.org/abs/ 。[33] S. Beamer、K. Asanovic 和 D. Patterson，《GAP 基准测试套件》，2017 [ ] 1508 03619 。在线。可访问：https://arxiv.org/abs/ 。[34] Y. Zhang、V. Kiriansky、C. Mendis、S. Amarasinghe 和 M. Zaharia，《让缓存服务于图分析》，载于*2017*年 *IEEE* 国际大数据会议（大数据），2017年，第 293–302 页。[35] K. Mohammadiporshokooh、S. R. Brandt 和 H. Kaiser，《使用 HPX 运行时系统自适应优化并行算法性能的新执行模型和执行器》，*SNComputer Science*，第 6 卷，第 8 期，第 911 页，2025年。[36] K. Mohammadiporshokooh、S. R. Brandt、R. Tohid 和 H. Kaiser，《自适应优化 HPX 并行算法的性能》，载于异步多任务系统与应用，P. Diehl、Q. Cao、T. Herault 和 G. Bosilca 编。Cham：Springer Nature Switzerland，2026年，第 70–82 页。