分布式深度多层图划分

Peter Sanders ✉卡尔斯鲁厄理工学院,
德国
DanielSeemaier ！ 卡尔斯鲁厄理工学院,
德国

摘要

我们介绍分布式内存多层图划分器 dKaMinPar 的工程实现。它可扩展至（至少）8192 个核心，同时实现了与广泛使用的串行和共享内存图划分器相当的划分质量。相比之下，以往的分布式图划分器只能在更受限的场景中扩展，而且与非分布式划分器相比，往往会造成显著的质量损失。当划分为大量块时，它们甚至会生成违反平衡约束的不可行解。dKaMinPar 通过对图划分深度多层方案进行可扩展的分布式实现，实现了稳健性。关键的是，这包括用于在细化和粗化过程中实现平衡的新算法。

2012年 ACM 学科分类 计算数学 → 图算法

关键词与短语 算法、分布式系统、图划分、多层算法、平衡

补充材料 源代码和数据已在此处公开 `https://algo2.iti.kit.edu/seemaier/ddeep_mgp/`。

致谢 本研究在 HoreKa 超级计算机上完成，该计算机由巴登-符腾堡州科学、研究与艺术部以及德国联邦教育与研究部资助。本项目获得了欧洲研究理事会（ERC）通过欧盟地平线2020研究与创新计划提供的资助（资助协议编号882500）。

arXiv:2303.01417v2 [cs.DC] 3 Mar 2023

1 引言

图是计算机科学中的核心概念，在需要对对象之间的关系进行建模时都会用到。因此，并行处理大型图至关重要。这通常需要将这些图划分为权重近似相等的块，并使大多数边位于块内（平衡图划分）。应用包括科学计算、社交网络处理、路径规划和图数据库 [4]。

原则上，多层图划分器（MGP）能够为各种输入图实现高质量划分 G，并在质量与划分成本之间取得良好权衡。其基本方法是首先通过收缩边或小簇，反复对图进行粗化 G。由此得到的小图 G' 仍能很好地表示整体输入图，并且对 G' 进行初始划分即可得到 G的良好划分。之后，通过对图进行反粗化，并在每个层级上使用细化算法改进划分，进一步提升划分质量。

然而，几十年来，多层图划分的并行化一直被证明颇具挑战性。尽管共享内存图划分器近年来已经日趋成熟，能够实现较高的质量和合理的可扩展性， [2, 11, 12, 16], 当前的分布式内存划分器[15, 23, 28] 却会导致质量严重下降，而且往往无法始终实现可行（即均衡）的划分。尤其是，高质量的图划分器无法扩展到大型超级计算机所提供的处理单元数量。由于块的数量通常 k 应随处理单元数量线性增加，这一情况变得更加严峻

处理单元数量。以往的系统无法直接处理大型 k 问题，在实现可行性方面甚至会面临更大的困难。

本文介绍了 dKaMinPar，它解决了上述所有问题。其基础是深度多层图划分概念的分布式内存改编 [11]，该改编将多层方法深入延伸至初始划分阶段。这使大规模 k 情况更加容易处理，并消除了由初始划分导致的并行化瓶颈。我们的粗化和细化算法基于标签传播方法，该方法此前已用于若干划分器 [15, 23, 28]。标签传播 [22, 24] 会在能够减少割（且不违反平衡约束）时，以贪心方式将顶点移动到其他簇/块中。即使对于复杂网络，这种方法也简单、快速、有效且稳健。我们开发了一个分布式内存版本，并通过使用改进的稀疏全到全原语等方式提升了可扩展性。或许最主要的算法创新在于新的可扩展分布式技术，这些技术能够维持平衡约束。在粗化过程中，通过反向展开导致超重簇的收缩来近似最大簇权重。在反粗化过程中，则通过查找、选择并应用全局“最佳”块移动来满足块权重约束。

第6节中介绍的实验表明，我们的实现已实现主要目标。即使对于此前无法在分布式求解器上扩展的复杂网络，它也至少可扩展至 8 192 个核心。即使对于大型 k，也能保证可行性；其质量通常与共享内存系统相差几个百分点以内。第7节总结了结果，并讨论了未来可能的改进。

贡献

- 可扩展的深度多层图划分分布式实现。使用标签传播同时进行收缩和细化，保持了简洁性。新的可扩展平衡粗化与反粗化算法。在来自 3 个输入族的大型现实世界网络和超大型合成网络上进行了广泛评估。质量可与共享内存系统相媲美。可扩展至（至少） 2^{13} 个核心和 2^{39} 条边。既适用于复杂网络，也适用于块数量较多的情况，而以往系统在这些情况下往往会失败。

2 预备知识

符号与定义。设 $G = (V, E, c, \omega)$ 是一个无向图，顶点权重c 为：$V \to \mathbb{N}_{>0}$，边权重 ω 为 $E \to \mathbb{N}_{>0}$, $n := |V|$，且 $m := |E|$。我们将 c 和 ω 扩展到集合，即 $c(V') := \sum_{v \in V'} c(v)$ 和 $\omega(E') := \sum_{e \in E'} \omega(e)$。$N(v) := \{u \mid \{u, v\} \in E\}$ 表示 v 的邻居。对于某个 $V' \subseteq V$，$G[V']$ 表示由 V' 在 G 中诱导的子图。我们要寻找节点块 $\Pi := \{V_1, \ldots, V_k\}$，将 V 划分为这些块，即 $V_1 \cup \cdots \cup V_k = V$且对于 $i \neq j$，$V_i \cap V_j = \emptyset$。平衡约束要求，对于所有 $i \in \{1, \ldots, k\}$，$c(V_i) \leq L_{\max} := \max\{(1+\varepsilon)^{c(V)}{}_k, {}^{c}(V)_k + \max_v c(v)\}$，其中不平衡参数为 ε[1]。目标函数是最小化 $\mathrm{cut}(\Pi) := \sum_{i<j} \omega(E_{ij})$（所有割边的权重），其中 $E_{ij} := \{\{u, v\} \in E \mid u \in V_i \text{ 且 } v \in V_j\}$。我们将一个在 V_j 中有邻居的顶点 $u \in V_i$ 称为

[1] 传统上，$L_k := (1+\varepsilon)\lceil {}^{c(V)}{}_k \rceil$ 被用作平衡约束。我们放宽了这一约束，因为否则要找到可行划分将是NP完全问题。

边界顶点。聚类$\mathcal{C} := \{C_1, \ldots, C_\ell\}$也是 V 的一种划分，其中块的数量 ℓ 事先并未给定（也不存在平衡约束）。

机器模型与输入格式。 本文采用的分布式内存模型包含编号为 $1..P$ 的 P 处理单元（PE），这些处理单元通过全双工、单端口通信网络相连。输入图以一维顶点划分的形式给出（通常是均衡划分）。每个处理单元都会获得输入图的一个子图（即一维划分的一个块），其中包含连续的顶点。无向边 $\{u, v\}$ 由两条有向边 (u, v)、(v, u) 表示，分别存储在拥有相应尾顶点的处理单元上。与其他处理单元所拥有的顶点相邻的顶点称为*接口顶点*，并在这些处理单元上以*幽灵顶点*的形式复制（即不包含出边）。

3 相关工作

关于图划分的研究成果极为丰富，读者可参阅综述论文 [3–5, 27]，了解大部分一般性内容。本文重点讨论用于高质量图划分的并行算法。

分布式图划分。 几乎所有高质量图划分器都基于多层次范式，例如 ParMETIS [14, 15]，ParHIP [23, 26] 以及其他方法 [6, 30]。这些算法分三个阶段对图进行划分。首先，它们构建一个层次结构，逐步生成输入图的粗化近似，通常通过收缩匹配或簇来实现。当图中剩余的顶点很少时（例如，$n \leq Ck$ 对于某个收缩限制 C），便将图划分为 k 块。最后，将此划分逐步投影到层次结构中更细的层级，并使用局部搜索算法进行细化。

多层算法的性能取决于各阶段所采用的算法组件。为网格划分设计的图划分器通常通过收缩匹配来粗化图 [6, 15, 30]。然而，这种技术不适用于划分最大匹配规模较小的复杂网络。因此，其他图划分器采用两跳匹配 [17] 或大小约束标签传播 [11, 13, 23]。由于后者简单而有效，也常在细化过程中用作局部搜索算法 [2, 8, 11, 13, 15, 23, 30]。

标签传播也用于非多层图划分算法，例如 XtraPuLP [28],，据其报告，该算法可扩展至 2^{17} 个核心；多层算法尚未达到这一水平。然而，不采用多层次范式而使用标签传播，会导致质量显著下降；参考文献 [11] 报告称，PuLP [29]（非多层）的边割平均比 KaMinPar（多层）大一倍以上。对于规模庞大且类型多样的基准测试集而言，这一差距相当大；大多数多层算法的平均边割彼此相差不超过几个百分点。

另一类具有高度可扩展性的图划分器是几何划分器，它们基于图的几何嵌入进行运算。尽管这些算法比多层算法快几个数量级 [19],，但它们通常会产生更大的边割，而且只适用于具有有意义几何嵌入的图。

深度多层图划分。由于普通多层图划分算法通常会将图缩小到仅剩 Ck 个顶点，因此较大的 k 值会打破最粗化图规模较小这一假设。这会导致其性能 [11]下降。作为替代方案，可以使用递归二分来计算具有较大 k的划分，但这会使运行时间额外增加一个 $\log k$ 因子

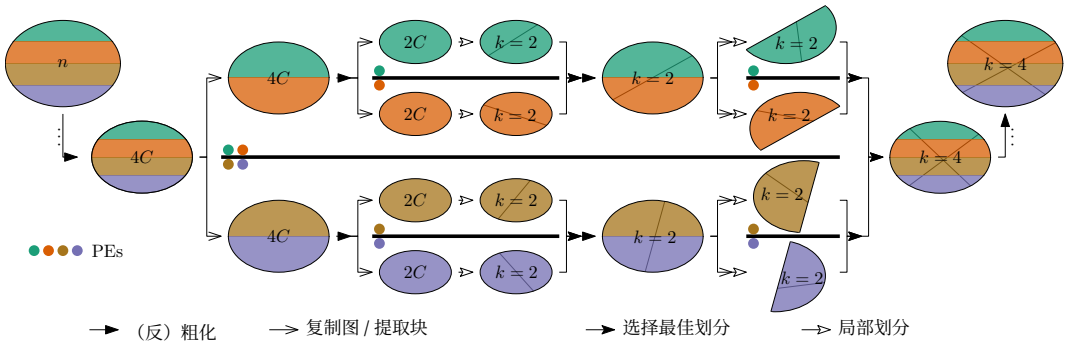

图 1 在 $P = 4$ 个处理单元上进行分布式深度多层图划分。未划分的图以其顶点数量标注，已划分的图以其块数量标注。块被细分为 $K = 2$ 个块，目标是将图划分为 $k = 4$ 个块。

，并且由于缺乏全局视图，计算平衡划分也会更加困难。深度多层图划分（深度多层图划分） [11] 通过在初始划分阶段继续进行粗化，规避了这些问题。更具体地说，深度多层图划分会将图粗化，直到只剩下 $2C$ 个顶点，而不受 k影响。为此，通过维持如下不变量来利用并行性：每个处理单元至少处理 C 个顶点；当该不变量即将被违反时，通过复制更粗化的图并拆分可用的处理单元来确保这一点。在对最粗化图进行二分划分后，它会维持如下不变量：具有 n 个顶点的（粗化）图通过在当前层级进行递归二分，被划分为 $\min\{k, n/C\}$ 个块。通过采用额外的平衡技术，基于深度多层图划分的图划分器能够获得包含大量块（例如 $k \approx 1M$个）的可行高质量划分，而且通常比基于普通多层图划分的图划分器快一个数量级。与对整个图进行递归二分相比，它将额外的 $\log k$ 因子降低为 $\log\ kC/n$。KaMinPar [11] 是深度多层图划分的一种可扩展共享内存实现，在粗化和细化过程中使用大小约束标签传播。

4 分布式深度多层图划分

我们介绍了 dKaMinPar，这是一种基于深度多层图划分的分布式图划分器。我们首先介绍分布式深度多层图划分方案本身，并通过假设 k 和 P 为 2 的幂来简化说明。然后，我们概述 dKaMinPar 中实现的用于粗化、初始划分、细化和平衡的构建模块。

分布式深度多层图划分。 我们在图 1 和算法 1 中概述了划分方案。分布式深度 MGP 首先将输入图粗化，直至剩下 $K \cdot C$ 个顶点，并构建由逐渐变粗的图组成的层次结构 $G_1, \ldots, G_\ell$（图 1 左侧及算法 1 第 6–8 行）。其中，C 是收缩限制，而 K 是一个调优参数，可将参考文献 [11] 中的二分划分步骤推广为 K 路划分。为提高粗化层级上的可扩展性，我们遵循参考文献 [30]，并维持这样一个不变量：P 个处理单元处理的图至少包含 $P \cdot C$ 个顶点；为此，我们将处理单元划分为若干组，并在否则会违反该不变量时复制当前图（第 2–5 行）。粗化收敛后，我们将最粗化图 G_ℓ 收集到所有处理单元上，并使用非分布式图划分器将其划分为 $\min\{k, K\}$ 个块（图 1 中间）。选出（每个处理单元组内的）最佳划分，并将其投影到 $G_{\ell-1}$ 上。从此处开始，我们维持两个不变量：（1）当前划分是可行的，这通过使用下文所述的

算法 1 DeepMGP(G、k、P)：在 P个处理单元上对含有 k 个块的深度多层进行划分。

输入： $G=(V、E)$，k，常量 C、K
输出： k路划分 Π 的 G

1 **如果** $|V(G)| > C \cdot \min\{k, K\}$ **则** // deep coarsening
2 **如果** $|V(G)| < C \cdot P$ **则** // duplicate graph if too small for P PEs
3 $c := P/\text{ceil}_2(|V(G)|/C)$ // number of copies
4 $G := \texttt{Replicate}(G, c)$ // replicate graph c times and group PEs
5 $P := P/c$ // number of PEs per group
   // standard multilevel graph partitioning:
6 $G_c := \texttt{Coarsen}(G)$
7 $\Pi_c := \texttt{DeepMGP}(G_c, k, P)$
8 $\Pi := \texttt{Project}(G, \Pi_c)$
9 $\Pi := \texttt{BalanceAndRefine}(G, \Pi)$
10 **否则** // base case
11 $\Pi := \{V\}$ // trivial 1-way partition
12 $k' := \text{分钟}\{k, \text{ceil}_2(|V|/C)\}$
13 **当** $|\Pi| < k'$ **时** // extend partition
14 $G'_1, \ldots, G'_{|\Pi|/P} := \texttt{DistributeBlocks}(G, \Pi)$ // G'_is are local graphs
   // partition each local graph into $\min\{k'/|\Pi|, K\}$ blocks:
15 **对于** $i := 1$ **到** $|\Pi|/P$ **执行**
16 $\Pi'_i := \texttt{LocalPartitioning}(G'_i, \text{最小}\{k'/|\Pi|, K\})$
   // combine local partitions Π'_i of G'_i to global partition Π of G:
17 $\Pi := \texttt{CollectPartitions}(\Pi'_1, \ldots\ldots, \Pi'_{|\Pi|/P})$
18 $\Pi := \texttt{BalanceAndRefine}(G, \Pi)$
19 **返回** Π

分布式平衡算法来保证；（2）含有 n_ℓ 个顶点的图被划分为 $\min\{k, \text{ceil}_2(n_\ell/C)\}$ 个块。这里，$\text{ceil}_2(x)$ 表示大于或等于 x 的最小 2 的幂。我们通过递归 K 路划分来维持第二个不变量。具体而言，每当违反该不变量时，我们便提取当前划分中由块诱导的子图，并将它们收集到各处理单元上，使每个处理单元都接收 k/P 个完全子图（注意，由于上文所述的复制过程，$k \geq P$）。随后，我们使用非分布式图划分器递归地划分这些子图，并将新划分投影回分布式图。算法 1 第 13–18 行展示了这一过程。随后，使用分布式 k 路细化算法改进所得划分，使其满足两个不变量。请注意，如果 $k > \text{ceil}_2(|V(G_1)|/C)$，则在最细图上计算出的划分块数不足。在这种情况下，我们会再次分发并划分由块诱导的子图，以补齐缺少的块（算法 1 中未列出）。

粗化。 我们采用与 ParHIP [23] 和 KaMinPar [11]类似的大小约束标签传播并行化方法。算法首先将每个顶点分配到各自的簇中。在后续对顶点的迭代中（我们进行{3、5} 次迭代），再将顶点移至相邻簇，从而在不违反最大簇权重约束的前提下，使簇内边的权重最大化 $W := \varepsilon^{c(V)} k'$ 其中 $k' := \min\{k、 |V|/C\}$ [11]。

6 分布式深度多层图划分

如参考文献 $[2, 22]$, 所述，按度数递增顺序遍历顶点可以提高标签传播的解质量。由于这种做法对缓存不友好，且缺乏随机化带来的多样性，我们将顶点按度数划分到间隔呈指数级增长的度数桶中，即，桶 i 包含度数为 $2^i \leq d < 2^{i+1}$的所有顶点，并据此重新排列输入图。这一操作在每个处理单元上独立进行，也就是说，我们不会对顶点进行全局排序。随后，在标签传播过程中，我们将各个桶划分为较小的块，并在块间层级和块内层级随机化遍历顺序。这与 Metis $[14]$所用匹配算法的随机化方式类似。

为了传递接口顶点当前的簇分配，我们沿用 ParHIP 的做法，将每次迭代拆分为 $\max\{\alpha, \beta/P\}$ 个批次（我们使用 $\alpha = 8, \beta = 128$）。每个批次结束后，我们使用稀疏全互操作，通知相邻处理单元哪些接口顶点已移至其他簇。由于簇可能跨越多个处理单元，因此，与共享内存环境相比，控制最大簇权重更具挑战性。ParHIP 放宽了权重限制，仅在局部强制执行，也就是说，允许簇的权重最高达到 $P \cdot W$。这可能导致粗顶点的权重过大，使计算平衡划分更加困难。作为替代，我们通过在每个批次结束后将簇权重的变化发送给拥有该簇初始顶点的处理单元，来跟踪全局簇权重；该处理单元会累积这些变化，并回复簇的总权重。如果某个簇的权重超过 W，每个处理单元都会按其所占簇总权重的比例撤销顶点移动。随后，这些顶点可在下一次迭代中移至其他簇。

对图进行聚类后，我们收缩所有簇，构建层次结构中的下一层图。关于此操作的更多细节，请参见第5节。

细化。 我们还使用大小约束标签传播来改进当前图划分。与上述用于聚类的标签传播不同，顶点最初被分配到代表划分块的簇中，并将最大块权重用作权重约束。我们使用与粗化阶段相同的迭代顺序和批次数量，将顶点移动到相邻块，以在不违反平衡约束的情况下最大化块内边的权重。若出现平局，则优先选择权重较轻的块；如果两个块的权重相同，则通过掷硬币决定。

由于细化阶段的块数通常远少于粗化阶段的簇数，我们在每个顶点批次之后通过全归约操作跟踪全局块权重。需要注意的是，如果多个处理单元在同一顶点批次中将顶点移至同一个块，这并不能避免违反平衡约束。在这种情况下，我们随后使用下文所述的全局平衡算法来恢复平衡约束。与共享内存并行图划分器中通过大小约束标签传播进行细化相比，这是一个缺点；后者可以使用原子比较并交换操作严格实施平衡约束。

平衡。 如参考文献 $[11]$, 所述，在深度多层图划分过程中，平衡约束违反可能发生在初始划分之后，或将粗粒度图划分投影到图层次结构的更细层级之后。由于这些平衡约束违反受到最重顶点权重的限制，我们设计了以下平衡算法，并假设只需进行少量顶点移动即可恢复平衡，因此值得为每次顶点移动投入适度的工作量。贪心算法的步骤如下。

对于每个过载块 B，我们在每个处理单元上维护一个优先队列 P_B，其中存放该块的局部顶点。优先队列中的顶点按照其<style id='3'>相对增益排序；我们将其定义为：如果 $g \geq 0$，则为 $g \cdot c(v)$；如果 $g < 0$，则为 $g/c(v)$；其中 g 是移动 B 到不会变为过载块的块时边割的最大减少量

请注意，该评分函数通常更倾向于移动少量重顶点，而不是移动大量较轻的顶点。这源于我们的假设：只需移动少量顶点即可平衡划分。为使优先队列保持较小规模，我们维护这一不变量：优先队列 P_B 中存储的顶点数不超过移除全部超出权重 $o(B) := c(B) - L_{\max}$ 所必需的数量，且这些顶点来自 B。我们通过遍历顶点来初始化优先队列。如果某个顶点属于过载块 B，并且 $c(P_B) < o(B)$，则将其插入。否则，仅当其相对增益大于 P_B 中任意顶点的最低相对增益时，才插入该顶点；插入后，如果 $c(P_B) > o(B) + \max_v c(v)$，则移除其中最低增益的顶点。

初始化优先队列后，我们使用二叉树归约，在全局范围内反复识别每个划分块中得分最高的 ℓ 个顶点，其中 ℓ为某个常数输入参数。对于每个过载块，每个处理单元从其本地优先队列中贡献最多 ℓ 个顶点。在归约树的每一层，随后将两个大小为 $\leq \ell$ 的列表合并，并在至多 ℓ 个顶点后截断；如果较短的前缀已足以移除相应划分块的所有超额权重，则提前截断。根处理单元随后决定执行哪些移动，使任何划分块都不会过载，并将其决定广播给所有处理单元。利用这些信息，处理单元从其优先队列中移除已移动的顶点，并更新已移动顶点的邻居的相对增益。我们重复此过程，直到划分达到平衡。

5 实现细节

顶点和边标识符。 为减少通信开销，我们区分本地和全局的顶点及边标识符。这样，我们可以对全局标识符使用64 位数据类型，对本地标识符使用32 位数据类型。

图收缩。 收缩由 n_C 个簇组成的聚类并构建相应粗化图的过程如下。首先，上文所述的聚类算法为每个顶点分配一个簇标识符，该标识符对应分布式图中的某个顶点标识符。我们称某个簇由拥有其对应顶点的处理单元拥有。收缩本地子图（即，删除簇之间的重复边，并累加顶点权重和边权重）后，我们将簇映射到处理单元，使每个处理单元获得大致相同数量的粗化顶点，同时尽量减少所需的通信量。我们将每个处理单元拥有的 $\leq \delta \cdot n_C/P$ 个簇分配给同一个处理单元（在我们的实验中，$\delta = 1.1$）。如果某个处理单元拥有更多簇，我们就将剩余的簇重新分配给当前分配簇数量最少的处理单元。随后，每个处理单元通过全对全操作将粗化顶点的出边发送给相应的处理单元，然后通过删除重复边并累加顶点权重和边权重来构建粗化图。

低延迟稀疏全对全通信。 dKaMinPar 的许多步骤都需要沿着分布式图的割边进行通信，这意味着需要进行（通常非常）稀疏且不规则的全对全通信。由于 `MPI_Alltoallv` 的延迟相对较高，我们改用两级方法，将处理单元排列成网格 [25]。随后，消息先发送到相应的行，再发送到相应的列，从而将通过网络发送的消息总数从 $\mathcal{O}(P^2)$ 降至 $\mathcal{O}(P)$。

6 实验

我们用 C++ 实现了所提出的算法 dKaMinPar，并使用带有标志 `-O3 -march=native`的 g++-12.1 对其进行编译。我们使用 OpenMPI 4.0 作为并行化库，并使用 growt [21]作为哈希表。所有实验的原始数据均可在线获取[2]。

实验设置。 我们在一台配备 1 TB 主存和一颗拥有 64 个核心的 AMD EPYC 7702P 处理器的共享内存机器（机器 A）上评估算法的解质量。此外，我们在一个高性能集群上进行可扩展性实验，该集群的每个计算节点均配备 256 GB 主存和两颗 Intel Xeon Platinum 8368 处理器。计算节点通过 InfiniBand 4X HDR 200 GBit/s 网络互连，延迟约为 1 µs。由于部分图生成器要求核心数为 2 的幂，我们仅使用可用 78 个核心中的 64 个。尽管我们的图划分器可以在每个 MPI 进程中使用多个线程，但我们只评估每个 MPI 进程使用一个线程的配置，因为这种配置通常能带来最佳性能。

我们将 dKaMinPar 与参考文献 [11]，中算法的分布式版本进行比较，即 ParHIP [23]（v3.14）和 ParMETIS [15]（v4.0.3）。由于其解质量无法与多层划分器相竞争，我们未将分布式版本 PuLP [29]（XtraPuLP [28]）纳入主要比较。我们在第 12 节中提供了与 XtraPuLP 的比较。我们评估算法的两种配置：dKaMinPar-Fast 使用 $C = 2000$ 作为收缩限制（与参考文献 [11] 相同），使用 KaMinPar [11] 进行初始划分，并在粗化过程中执行 3 轮标签传播；而 dKaMinPar-Strong 使用 $C = 5000$（与参考文献 [23] 相同），使用 Mt-KaHyPar [13] 进行初始划分，并在粗化过程中执行 5 轮标签传播。

实例。 我们在参考文献 [11]的基准测试集 B 中的图，以及参考文献 [23]中使用的图上评估算法。所有图的列表见第 8 节的表1。此外，我们使用图生成器 KaGen [10] 评估算法在超大规模随机生成的二维和三维几何图及双曲图上的扩展能力，这些图分别记为 $\mathsf{rgg}_{2\mathrm{D}} N\mathsf{d}D$、$\mathsf{rgg}_{3\mathrm{D}} N\mathsf{d}D$和 $\mathsf{rhg}_{3.0} N\mathsf{d}D$。这些图在每个计算节点（64 个核心）上有 2^N个顶点，平均度为 D。随机双曲图的幂律指数为 3。这些图族中最大的图有 2^{33} 个顶点和 2^{39} 条边。

方法。 我们将图与块数的组合称为一个实例。对于每个实例，我们使用不同的随机种子进行 5 次重复实验，并以算术平均值汇总边割和运行时间。汇总多个实例时，我们使用几何平均值。

为了比较不同算法的解质量，我们使用性能剖面 [9]。设$\mathcal{A}$ 为要比较的算法集合，$\mathcal{I}$ 为实例集合，$q_A(I)$ 表示算法 $A \in \mathcal{A}$ 在实例 $I \in \mathcal{I}$上的质量。对于每个算法 A，我们绘制满足 $\frac{|\{I \in \mathcal{I} \mid q_A(I) \leq \tau \cdot \min_{A' \in \mathcal{A}} q_{A'}(I)\}|}{|\mathcal{I}|}$ 的实例比例，其中 $\mathcal{I}_A(\tau) := \{I \in \mathcal{I} \mid q_A(I) \leq \tau \cdot \min_{A' \in \mathcal{A}} q_{A'}(I)\}$，而 τ 位于 x-轴上。在较低的 τ值处达到更高的比例被视为更好。对于 $\tau = 1$，y值表示算法表现最佳的实例所占百分比。

解质量和运行时间。 我们评估 dKaMin 的解质量和运行时间 Par
与使用机器 A 的全部 64 个核心的其他分布式 MGP 算法进行比较。这里，w e

[2] https://algo2.iti.kit.edu/seemaier/ddeep_mgp/

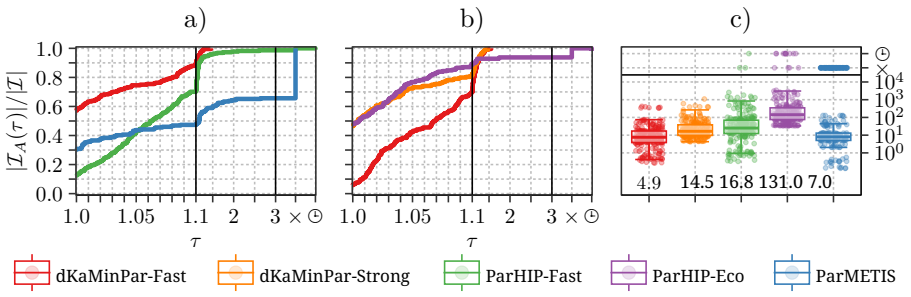

图2 $k = \{2、4、8、16、32、64、128\}$ 在机器 A 上使用 $\varepsilon = 3\%$ 时的结果。从左至右：(a) dKaMinPar-Fast、ParHIP-Fast 和 ParMETIS 的边割；(b) dKaMinPar-Fast、dKaMinPar-Strong 和 ParHIP-Eco 的边割；(c) 所有算法的运行时间。x 轴上方的数字是所有算法均得出结果的实例的几何平均运行时间 [s] 。超时以 ⏲ 标记，运行失败或结果不可行以 × 标记。

将基准测试集中的图划分为 $k \in \{2、4、8、16、32、64、128\}$ 个块，使用 $\varepsilon = 3\%$。为了深入了解 dKaMinPar 由于其分布式特性而产生的性能损失，我们还将其与共享内存划分器 KaMinPar 进行比较。此外，我们在第9节报告了 $k \in \{2^{11}, 2^{14}, 2^{17}, 2^{20}\}$（大型 k）的结果。我们将单个实例的时间限制设为一小时，约为 dKaMinPar-Fast 在大多数实例上处理小型 k 的运行时间的 10 倍[3]。

结果汇总于图2a–c。图2a显示，dKaMinPar-Fast 在约 60% 的基准测试实例上取得了最低边割，而 ParMETIS 和 ParHIP-Fast 分别仅在约 30% 和 10% 的实例上取得了更低的边割。此外，两种竞品算法经常无法计算出可行划分——尤其是 ParMETIS，无法划分大多数社交网络；在 34% 的实例上，它违反了平衡约束或发生崩溃。考察运行时间时（图 2c），因此我们仅对所有划分器均计算出可行划分或运行超时的实例取平均值（224 个实例中的 145 个）。dKaMinPar-Fast（几何平均运行时间为 4.93 秒）分别比 ParMETIS（6.98 秒）和 ParHIP-Fast（16.77 秒）快 1.4 倍和 3.4 倍。我们认为这有几个原因。与 ParMETIS 相比，运行时间更短是因为复杂网络的收缩更快；而相较于 ParHIP-Fast 的优势，则归功于我们缓存效率更高的标签传播实现、更高效的图收缩，以及第5节中介绍的低延迟稀疏全对全通信实现。

ParHIP 还提供了一种强配置 ParHIP-Eco，它通过执行更多 V循环，并在初始划分时使用进化算法，以显著改善边割，但代价是运行时间大幅增加。图2b 显示，使用上文所述的 dKaMinPar-Strong 配置时，dKaMinPar 可以达到相同的解质量，同时仍比 ParHIP-Fast 更快。

最后，我们将分布式划分器与 KaMinPar 进行比较。KaMinPar 是一种共享内存实现的深度多层图划分器，采用标签传播进行粗化和细化。结果汇总于图3。两种划分器的边割质量几乎相同，计算得出的平均边割差异不到 0.5%。然而，分布式划分器比共享内存划分器慢约 50%。这是预期之中的，因为通过消息传递进行通信通常不如共享内存通信高效。

[3] 只有 twitter-2010 需要 6 分钟（$k = 64$）或 7 分钟（$k = 128$）。

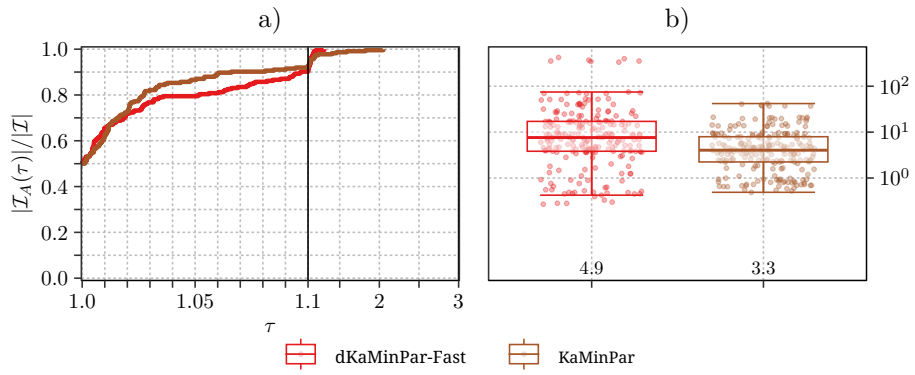

图3 $k = \{2、4、8、16、32、64、128\}$ 在机器 A 上使用 $\varepsilon = 3\%$。从左到右：(a) dKaMinPar-Fast 和 KaMinPar 的边割；(b) 相应的运行时间。x 轴上方的数字为几何平均运行时间 [s]。

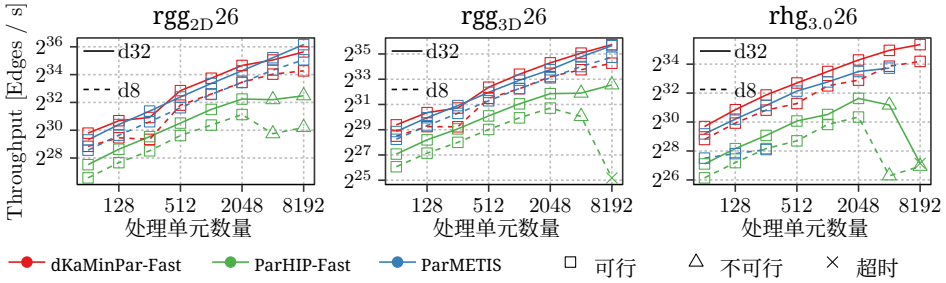

图 4 rgg_{2D}、rgg_{3D} 和 rhg 图的吞吐量；每个计算节点有 2^{26} 个顶点，平均度为 $\in \{8、32\}$，在机器 B 的 64–8 192 个核心上使用 $k = 16$ 和 $\varepsilon = 3\%$。

dKaMinPar 的弱可扩展性。我们使用高性能集群机器 B 上的 64–8 192 个核心（即 1–128 个计算节点）评估 dKaMinPar 的弱可扩展性。基准测试实例采用随机生成的几何图（二维和三维）及双曲图，每个计算节点包含 2^{26} 个顶点，平均度为 $\in \{8,32\}$。

在图 4 中，我们将这些图划分为 $k = 16$ 个块，并观察到 dKaMinPar-Fast 在所有三个图族上一直到 8 192 个核心都具有弱可扩展性。在几何图上，我们实现了与 ParMETIS 相近的吞吐量，而 ParHIP-Fast 在超过 2 048 个核心后可扩展性有所下降。这很可能是由于 ParHIP-Fast 在图收缩过程中进行了大量低效通信。此外，我们注意到，ParHIP-Fast 最初的设计目标是在标签传播过程中通过使用非阻塞 MPI 操作来重叠本地工作与全局通信。该实现依赖 MPI 进度线程，而这些线程似乎在现代 OpenMPI 版本中不可用。ParMETIS 在随机双曲图上的吞吐量明显更差，并且无法在 8 192 个核心上计算出划分。如前所述，这很可能是因为它在遵循幂律度分布的图上采用了低效的粗化策略。

观察边割（第 10 节中的表 3，上半部分）可见，在稠密的 $\text{rgg}_{2D}26$d32 图和两个 rgg_{3D} 图上，ParMETIS 的边割比 dKaMinPar-Fast 小 5%–13%。然而，在较稀疏的 $\text{rgg}_{2D}26$d8 图上，dKaMinPar-Fast 的割比 ParMETIS 小 19%，这已经是相当大的改进。对于双曲图，差距要大得多：ParMETIS 得到的割仅比 dKaMinPar-Fast 大约 5.5–6.1 倍。这样的解不适用于许多应用。

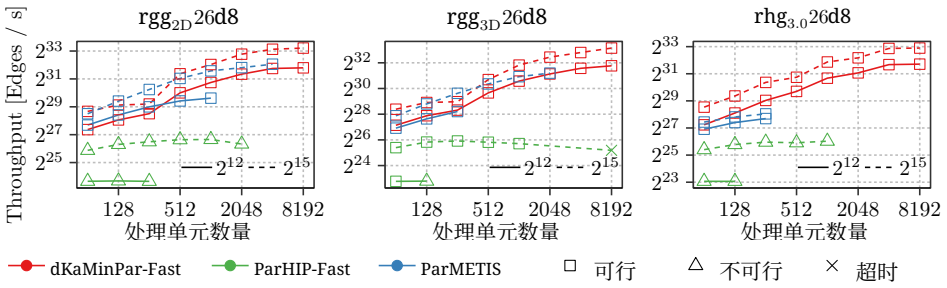

图 5 在机器 B 的 64–8192 个核心上，rgg2D、rgg3D 和 rhg 图的吞吐量；每个计算节点有 2^{26} 个顶点，平均度为 8，且 $\varepsilon = 3\%$。块的数量随图的规模进行调整，使每个块包含 2^{12} 或 2^{15} 个顶点。

现在，我们通过随所用计算节点数量增加而扩展图规模和 块数，评估弱可扩展性。在这种情况下，使用大量核心时，块数也会很大。图 5 总结了各算法在此设置下的吞吐量。请注意，本实验仅使用较稀疏的图，因为即使计算节点数量较少，ParMETIS 和 ParHIP 也无法划分这些图的稠密版本。

ParHIP-Fast 除了 6 个实例外，无法获得可行划分；而这 6 个实例使用的核心数均不超过 1 024 个，且其吞吐量仅在最多 256 个核心时呈上升趋势。虽然 ParMETIS 的弱可扩展性尚可，并且能为网格型图计算出可行解，但它无法在 8 192 个核心上划分任何图，并且在较少核心上也经常崩溃（例如，对于每块包含 2^{12} 个顶点的 rgg_{2D}，它只能在最多 1 024 个核心上运行）。对于随机双曲图，它仅能在 64 个核心上计算出可行解。与此同时，dKaMinPar-Fast 在每个图族上均展现出最高达 8 192 个核心的弱可扩展性，不过需要指出的是，从 4 096 个核心增加到 8 192 个核心时，吞吐量的增幅相当小。

就割边数而言，dKaMinPar 找到的边割平均分别比 ParMETIS 和 ParHIP-Fast 少 19.3% 和 2.8%（仅对相应图划分器计算出可行划分的实例取平均），改进幅度从 $\text{rgg}_{3D}26$ d8 上的 0% 到 $\text{rhg}_{8}3.0\text{d}26$（每块 2^{15} 个顶点）上的约 60% 不等。各实例的详细边割结果见表 3（第 10 节）。

dKaMinPar 的强可扩展性。现在我们来考察 dKaMinPar 的强可扩展性。我们将基准测试集中的三个最大低阶图和高阶图划分为$k = 16$ 块，在机器 B 上使用 64–8 192 个核心，时间限制为 15 分钟。结果汇总于图 6，可以看到，高阶图在最多 1 024–2 048 个核心下具有良好的可扩展性。无论使用多少个核心，ParMETIS 都无法划分这些图。尽管 ParHIP-Fast 在 uk-2007-05 上可扩展至 2 048 个核心，但值得注意的是，它的运行时间仍然超过 dKaMinPar 仅使用 64 个核心时的运行时间。由于 Twitter 图的度分布高度偏斜，因此很难对其进行粗化；在这种情况下，我们观察到只有 dKaMinPar 能在时间限制内划分该图。

转向最大度较小的图时，我们观察到 kmer_V1r、nlpkkt240 和 $\text{rgg}_{2D}27$分别在最多 2 048、2 048 和 1 024 个核心上具有强扩展性。其他算法似乎无法划分 kmer_V1r；ParMETIS 也只能在 512–4 096 个核心上划分该图，尽管该图相对较小，能够装入单个计算节点的内存中。ParHIP-Fast 完全无法划分该图。与我们的弱扩展实验类似，ParMETIS 在

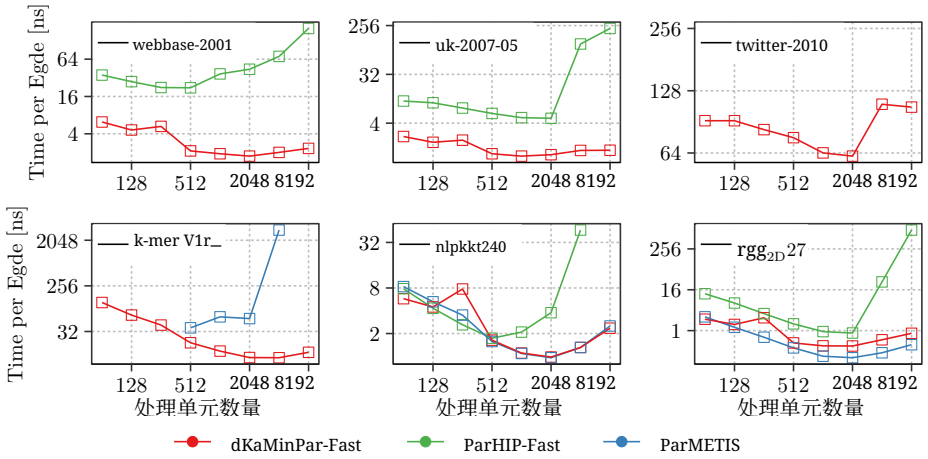

图 6 基准测试集中最大低阶图和高阶图在 $k = 16$, $\varepsilon = 3\%$ 下使用机器 B 的 64–8192 个核心时的强扩展运行时间。

网格型图 rgg_{2D} 以及 nlpkkt240 上表现出更好的可扩展性和吞吐量。

扩展到大量核心时，得到的边割保持相对稳定（第 11 节中的表 4）。令人惊讶的是，8192 个核心上的几何平均边割略优于 64 个核心上的结果（好 2.0%）。

7 结论与未来工作

我们的分布式内存图划分器 dKaMinPar 能够成功地使用数千个核心对各种输入图进行划分，兼具高速度和良好质量。实现仍有进一步改进的空间，例如更好地利用每个计算节点上的共享内存。除此之外，还可以探索质量与时间之间的权衡。通过分布式实现更强大的局部改进算法（如局部搜索或基于流的技术），可以以更长的执行时间为代价获得更好的质量。此时，考虑采用多种不同版本的图划分器并行运行、针对部分输入实现良好划分质量，也是有意义的。例如，ParMETIS 中采用的基于匹配的粗化可能有助于处理网格状网络。另一方面，更激进地处理高阶节点的方法可能有助于处理某些社交网络。

参考文献

1 图聚类与划分基准测试。《社会网络分析与挖掘百科全书》，施普林格（2014年）2 Akhremtsev, Y., Sanders, P., Schulz, C.：高质量共享内存图划分。《IEEE 并行与分布式系统汇刊》**31**(11), 2710–2722（2020年）3 Bichot, C., Siarry, P.（编）：图划分。威利（2011年）4 Buluç, A., 等：图划分的最新进展。载于：《算法工程：精选成果与综述》，《计算机科学讲义》，第9220卷，第117–158页（2016）5 Çatalyürek, U.V., 等：（超）图划分的更多最新进展。《ACM 计算调查》（2022）6 Chevalier, C., Pellegrini, F.：PT-Scotch：一种用于高效并行图排序的工具。《并行计算》**34**(6–8), 318–331（2008）

7 Davis, T.A., Hu, Y.：佛罗里达大学稀疏矩阵集合。《ACM 数学软件汇刊》38(1), 1:1–1:25（2011年） 8 Devine, K.D., 等：面向科学计算的并行超图划分。载于：第20届国际并行与分布式处理研讨会（IPDPS 2006） 9 Dolan, E.D., Moré, J.J.：利用性能剖面对优化软件进行基准测试。《数学规划》91(2), 201–213（2002年） 10 Funke, D., 等：无通信的大规模分布式图生成。《并行与分布式计算杂志》131, 200–217（2019年） 11 Gottesbüren, L., 等：深度多层图划分。载于：第29届欧洲算法研讨会（ESA）2021。《LIPIcs》，第204卷，第48:1–48:17页。Dagstuhl城堡莱布尼茨信息学中心 12 Gottesbüren, L., Heuer, T., Sanders, P.：并行基于流的超图划分。载于：第20届实验算法国际研讨会（SEA 2022）。第233卷，第5:1–5:21页。LIPICS 13 Gottesbüren, L., Heuer, T., Sanders, P., Schlag, S.：可扩展的共享内存超图划分。载于：第23届算法工程与实验研讨会（ALENEX 2021）。第16–30页。SIAM 14 Karypis, G., Kumar, V.：用于划分非规则图的快速高质量多层次方案。《SIAM 科学计算杂志》20(1), 359–392（1998年） 15 Karypis, G., Kumar, V.：用于非规则图的多层 k路划分方案。《并行与分布式计算杂志》48(1), 96–129（1998年） 16 LaSalle, D., Karypis, G.：多线程图划分。载于：第27届IEEE国际并行与分布式处理研讨会（IPDPS）。第225–236页（2013年） 17 LaSalle, D., 等：改进现代图与体系结构的图划分。载于：第5届非规则应用、体系结构与算法研讨会（IA3）2015。第14:1–14:4页。ACM 18 Leskovec, J., Krevl, A.：SNAP数据集：斯坦福大型网络数据集集合。 `http://snap.stanford.edu/data` （2014） 19 von Looz, M., Tzovas, C., Meyerhenke, H.：用于并行几何划分的平衡k均值。载于：第47届国际并行处理会议（ICPP）2018。第52:1–52:10页。ACM 20 o. M., U.：网络算法实验室。数据集。 `http://law.di.unimi.it/datasets.php`21 Maier, T., Sanders, P., Dementiev, R.：并发哈希表：快速且通用（？）！《ACM 并行计算汇刊》5(4), 16:1–16:32（2019年） 22 Meyerhenke, H., Sanders, P., Schulz, C.：通过规模约束聚类划分复杂网络。载于：实验算法——第13届国际研讨会（SEA）2014。《计算机科学讲义》，第8504卷，第351–363页。施普林格 23 Meyerhenke, H., Sanders, P., Schulz, C.：面向复杂网络的并行图划分。《IEEE 并行与分布式系统汇刊》28(9), 2625–2638（2017年） 24 Raghavan, N., Albert, R., Kumara, S.：一种近线性时间算法，用于检测大规模网络中的社区结构。《物理评论E辑：统计、非线性与软物质物理》76, 36–106（2007年） 25 Sanders, P., Schimek, M.：大规模并行最小生成树算法的工程实现。载于：第27届IEEE国际并行与分布式处理研讨会（IPDPS）（2023年） 26 Sanders, P., Schulz, C.：立足局部，着眼全局：高度平衡图划分。载于：第12届实验算法研讨会（SEA）2013 27 Schulz, C., Strash, D.：图划分：面向大数据的公式化方法与应用。载于：《大数据技术百科全书》。施普林格（2019年） 28 Slota, G.M., 等：可扩展、多约束、复杂目标图划分。《IEEE 并行与分布式系统汇刊》31(12), 2789–2801（2020年）

29 Slota, G.M., Madduri, K., Rajamanickam, S.：PuLP：面向小世界网络的可扩展多目标多约束划分。载于：2014年 IEEE 国际大数据会议（IEEE BigData 2014）。第 481–490 页30 Walshaw, C., Cross, M.：JOSTLE：并行多层图划分软件概述。网格划分技术与区域分解技术，第 27–58 页（2007 年）

8 基准测试实例

表1 基准测试集的基本属性。根据最大度 Δ，将图大致分为低度图和高度图。

	图	n	m	Δ	Ref.
Low-degree graphs	packing	2 145 839	34 976 486	18	[1]
	channel	4 802 000	85 362 744	18	[1]
	hugebubbles	19 458 087	58 359 528	3	[1]
	nlpkkt240	27 993 600	746 478 752	27	[1]
	europe.osm	50 912 018	108 109 320	13	[1]
	kmerU1a	64 678 340	132 787 258	35	[7]
	$\text{rgg}_{3D}26$	67 106 449	755 904 090	34	[10]
	$\text{rgg}_{2D}26$	67 108 858	1 149 107 290	45	[10]
	$\text{del}_{3D}26$	67 108 864	1 042 545 824	40	[10]
	$\text{del}_{2D}26$	67 108 864	402 653 086	26	[10]
	$\text{rgg}_{3D}27$	134 214 672	1 575 628 350	36	[10]
	$\text{rgg}_{2D}27$	134 217 728	2 386 714 970	46	[10]
	$\text{del}_{2D}27$	134 217 728	606 413 354	14	[10]
	$\text{del}_{3D}27$	134 217 728	2 085 147 648	40	[10]
	kmerP1a	138 896 082	296 930 692	40	[7]
	kmerA2a	170 372 459	359 883 478	40	[7]
	kmerV1r	214 004 392	465 409 664	8	[7]
High-degree graphs	亚马逊	400 727	4 699 738	2 747	[18]
	欧盟-2005	862 664	32 276 936	68 963	[1]
	YouTube	1 134 890	5 975 246	28 754	[18]
	2004年	1 382 867	27 182 946	21 869	[1]
	Orkut	3 072 441	234 370 166	33 313	[18]
	enwiki-2013	4 203 323	183 879 456	432 260	[20]
	enwiki-2018	5 608 705	234 488 590	248 444	[20]
	uk-2002	18 483 186	523 574 516	194 955	[1]
	阿拉伯语-2005	22 743 881	1 107 806 146	575 628	[20]
	乌克兰语-2005	39 454 463	1 566 054 250	1 776 858	[20]
	意大利语-2004	41 290 648	2 054 949 894	1 326 744	[20]
	twitter-2010	41 652 230	2 405 026 092	2 997 487	[20]
	sk-2005	50 636 059	3 620 126 660	8 563 816	[20]
	uk-2007-05	105 153 952	6 603 753 128	975 419	[1]
	webbase-2001	115 554 441	1 709 619 522	816 127	[20]

9 大 k 的详细结果

表 2 $k \in \{2^{10}, 2^{14}, 2^{17}, 2^{20}\}$ 和 $\varepsilon = 3\%$ 在机器 A 上的边割和运行时间结果。不可行解的几何平均不平衡度显示在不可行解数量旁边。最后两列 相对运行时间 和 相对边割 显示了所有相应算法未发生崩溃的实例相对于 dKaMinPar-Fast 的几何平均运行时间和边割（比较边割时还会排除超时实例）。

算法	# 超时	# 崩溃	# 不可行	# 可行	相对运行时间	相对边割
dKaMinPar-Fast	**0**	**0**	**0**	**128**	1.00	1.00
ParHIP-Fast	6	48	57 (1.19)	17	22.21	1.06
ParHIP-Eco	39	40	37 (1.10)	12	88.35	0.96
ParMETIS	**0**	44	54 (1.18)	30	2.38	1.01
KaMinPar	**0**	**0**	**0**	**128**	**0.46**	0.98

如表 2 所示，只有基于（分布式）深度多层图划分的图划分器，才能在块数较多时稳定地计算出可行划分。ParHIP-Fast 和 ParMETIS 分别仅能在 128 个实例中的 17 个和 30 个实例上做到这一点。我们还注意到，ParHIP 的速度相当慢，即使采用快速配置，速度也比 dKaMinPar-Fast 慢一个数量级以上。这是因为 ParHIP 在最粗化图中为每个块保留的顶点相对较多（$C = 5000$），因此即使 k的值适中，也无法充分缩小图。

10 弱扩展的边割结果

表 3 机器 B 使用 64–8 192 个核心、$\varepsilon = 3\%$ 的随机生成图弱扩展实验的边割结果。由于篇幅限制，未显示部分列。边割和几何平均值均以 dKaMinPar-Fast 为基准进行报告。超时（15 分钟）和崩溃分别标记为 ◷ 和 ×，不可行解标记为 △。

	图	算法	边割与处理单元数量 / 1 000 128	512	2 048	8 192	几何平均值
$k = 16$	$\text{rgg}_{2D}26d8$	dKaMinPar-Fast	280	572	1 177	2 428	1.00
		ParHIP-Fast	1.10	1.13	1.14	△	1.11
		ParMETIS	1.22	1.19	1.18	1.17	1.19
		XtraPuLP	87.12	131.52	333.21	637.60	190.28
	$\text{rgg}_{2D}26d32$	dKaMinPar-Fast	4 156	8 544	17 508	36 050	1.00
		ParHIP-Fast	1.12	1.14	1.16	△	1.15
		ParMETIS	0.95	0.94	0.94	0.94	0.94
		XtraPuLP	24.39	41.98	78.05	174.62	53.72
	$\text{rgg}_{3D}26d8$	dKaMinPar-Fast	4 668	12 037	31 526	81 200	1.00
		ParHIP-Fast	1.01	1.01	1.00	◷	1.01
		ParMETIS	0.95	0.96	0.92	0.92	0.94
		XtraPuLP	12.48	19.92	25.31	37.47	21.13
	$\text{rgg}_{3D}26d32$	dKaMinPar-Fast	40 411	103 491	267 885	701 865	1.00
		ParHIP-Fast	1.02	1.02	1.01	△	1.02
		ParMETIS	0.88	0.87	0.86	0.84	0.87
		XtraPuLP	6.25	10.03	11.41	15.71	9.94
	$\text{rhg}_{3.0}26d8$	dKaMinPar-Fast	3	11	6	3	1.00
		ParHIP-Fast	1.04	1.06	1.34	△	1.20
		ParMETIS	4.85	×	×	×	5.51
		XtraPuLP	9 830.42	12 480.89	77 483.14	608 259.52	48892.85
	$\text{rhg}_{3.0}26d32$	dKaMinPar-Fast	76	66	70	90	1.00
		ParHIP-Fast	1.06	1.13	1.18	◷	1.12
		ParMETIS	4.22	6.11	8.95	×	6.07
		XtraPuLP	1 074.49	4 613.94	17 322.39	59 519.97	6125.31
2^{15} vertices per block	$\text{rgg}_{2D}26d8$	dKaMinPar-Fast	5 543	22 517	90 764	364 279	1.00
		ParHIP-Fast	△	△	△	×	–
		ParMETIS	1.18	1.17	1.16	×	1.17
		XtraPuLP	△	△	×	×	–
	$\text{rgg}_{3D}26d8$	dKaMinPar-Fast	38 348	156 298	635 175	2 560 517	1.00
		ParHIP-Fast	1.03	1.03	×	◷	1.03
		ParMETIS	0.99	1.00	0.99	×	1.00
		XtraPuLP	△	△	×	×	–
	$\text{rhg}_{3.0}26d8$	dKaMinPar-Fast	1 028	5 157	19 457	77 084	1.00
		ParHIP-Fast	△	△	×	×	–
		ParMETIS	2.45	×	×	×	2.52
		XtraPuLP	117.45	×	×	×	115.92
2^{12} vertices per block	$\text{rgg}_{2D}26d8$	dKaMinPar-Fast	16 751	68 202	277 126	1 109 307	1.00
		ParHIP-Fast	△	×	×	×	–
		ParMETIS	1.09	1.07	×	×	1.08
		XtraPuLP	×	×	×	×	–
	$\text{rgg}_{3D}26d8$	dKaMinPar-Fast	74 560	300 409	1 213 176	4 876 567	1.00
		ParHIP-Fast	△	×	×	×	1.03
		ParMETIS	1.00	×	×	×	1.01
		XtraPuLP	×	×	×	×	–
	$\text{rhg}_{3.0}26d8$	dKaMinPar-Fast	7 255	30 901	123 132	495 273	1.00
		ParHIP-Fast	△	×	×	×	–
		ParMETIS	1.67	×	×	×	1.67
		XtraPuLP	×	×	×	×	–

11 强扩展的边割结果

表 4 机器 B 上使用 64–8 192 个核心进行强扩展实验的边割结果，$k = 16, \varepsilon = 3\%$。受篇幅限制，部分列未予显示。边割和几何平均值均相对于 dKaMinPar-Fast 报告。超时（15 分钟）和崩溃分别用 ◷ 和 ×标记，不可行解用 △ 标记。

图	算法	边割数（按处理单元数量计）/ 1 000				几何平均值
		128	512	2 048	8 192	
高阶图						
webbase-2001	dKaMinPar-Fast	9 634	9 618	9 602	9 524	1.00
	ParHIP-Fast	1.09	1.10	1.14	1.12	1.11
	ParMETIS	×	×	×	×	–
	XtraPuLP	2.62	3.30	6.13	13.79	5.20
uk-2007-05	dKaMinPar-Fast	4 093	4 138	4 176	4 064	1.00
	ParHIP-Fast	1.07	1.03	1.01	1.06	1.04
	ParMETIS	×	×	×	×	–
	XtraPuLP	42.94	116.57	185.19	264.43	125.13
twitter-2010	dKaMinPar-Fast	616 530	601 073	604 954	588 380	1.00
	ParHIP-Fast	◷	◷	◷	◷	–
	ParMETIS	◷	×	×	×	–
	XtraPuLP	1.36	1.48	1.51	1.58	1.48
低度图						
k-mer V1r	dKaMinPar-Fast	10 936	10 897	10 880	10 836	1.00
	ParHIP-Fast	×	×	×	×	–
	ParMETIS	×	0.84	0.84	◷	0.84
	XtraPuLP	13.19	13.12	12.56	12.56	12.85
nlpkkt240	dKaMinPar-Fast	5 658	5 641	5 623	5 547	1.00
	ParHIP-Fast	0.99	1.00	1.00	×	1.00
	ParMETIS	0.93	0.94	0.95	0.97	0.95
	XtraPuLP	3.44	3.88	2.80	2.43	3.09
$\mathrm{rgg}_{2\mathrm{D}}27$	dKaMinPar-Fast	349	349	350	347	1.00
	ParHIP-Fast	1.14	1.15	1.13	1.15	1.14
	ParMETIS	1.19	1.18	1.17	1.15	1.17
	XtraPuLP	72.87	150.46	259.72	194.77	153.46

12 与 XtraPuLP 的比较

我们将 dKaMinPar 与单层图划分器 XtraPuLP [28] (v0.3) 进行比较。XtraPuLP 是单层标签传播的混合（OpenMPI+OpenMP）实现。为避免图生成带来的运行时间开销过大，我们仅让 XtraPuLP 在每个 MPI 进程中使用一个线程。

图 7 展示了强扩展运行时间，而图 8 展示了弱扩展运行时间（$k = 16$）；图 9 展示了弱扩展运行时间（每个块有$\{2^{15}, 2^{18}\}$ 个顶点）。相应的边割数据列于表 4（第 11 节）和表 3 （第 10 节）。令人意外的是，在大多数测试的真实世界图和人工图上，dKaMinPar 的吞吐量都更高，不过需要指出的是，如果为 XtraPuLP 的每个 MPI 进程配置更多线程，其性能很可能会有所提升。此外，我们注意到，计算出的边割通常显著更差（在使用 8 192 个核心对 $\text{rhg}_8$3.0d26 进行划分时，最多相差 5 个数量级），不及多层划分器的结果。

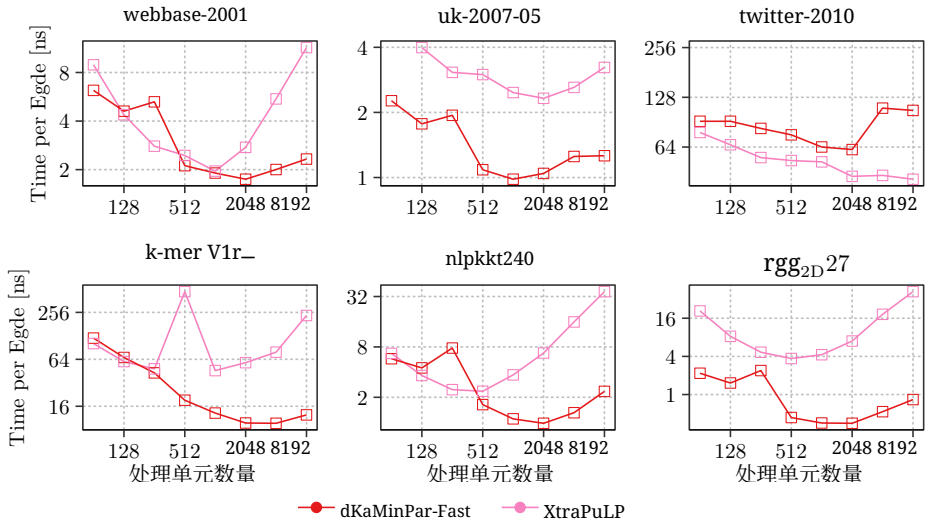

图 7 在机器 B 的 64–8192 个核心上，对基准测试集中最大规模的低阶图和高阶图进行强扩展时的运行时间，$k = 16, \varepsilon = 3\%$。

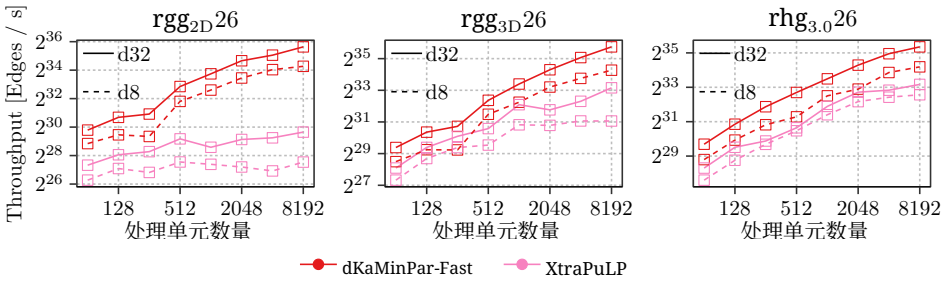

图 8 在机器 B 的 64–8 192 个核心上，每个计算节点包含 2^{26} 个顶点、平均度为 $\in \{8、32\}$时，rgg_{2D}、rgg_{3D} 和 rhg 图的吞吐量，以及 $k = 16$ 和 $\varepsilon = 3\%$

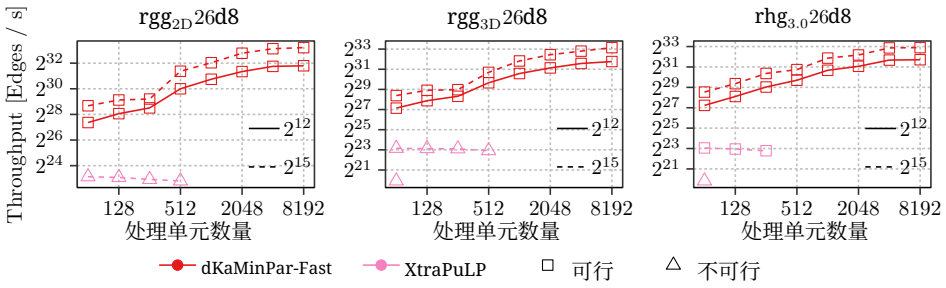

图 9 在机器 B 的 64–8192 个核心上，每个计算节点具有 2^{26} 个顶点、平均度为 8 且 $\varepsilon = 3\%$ 的 rgg2D、rgg3D 和 rhg 图的吞吐量。块的数量随图的规模进行缩放，使每个块包含 2^{12} 或 2^{15} 个顶点。