# Modern Software Libraries for Graph Partitioning (Abstract)

Lars Gottesbüren
lars.gottesbueren@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Nikolai Maas
nikolai.maas@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Peter Sanders
sanders@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Daniel Seemaier
daniel.seemaier@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

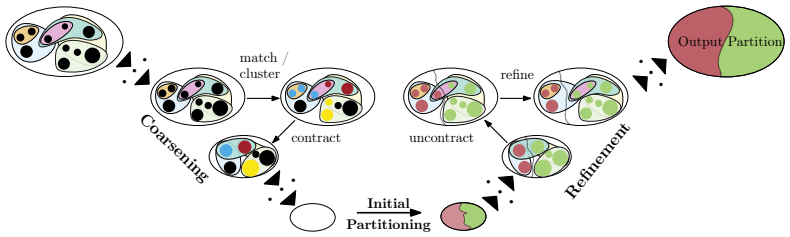


**Figure 1: The multilevel scheme.**

## Abstract

We summarize recent developments in parallel libraries for balanced graph partitioning, spanning contributions made over several research papers [2–9] that have culminated in the two state-of-the-art solvers for graphs and hypergraphs: KaMinPar and MtKaHyPar. These works have closed a significant gap in solution quality between sequential solvers and fast parallel codes.

## 1 Introduction

Balanced graph partitioning is a well-established NP-hard problem with the goal of partitioning the vertices of a graph $G = (V, E)$ into a fixed number $k \geq 2$ of disjoint blocks $V_1, \ldots, V_k$ with roughly equal

size $|V_i| \leq (1+\varepsilon)|V|/k$ (the balance constraint) while minimizing the number of cut edges $\sum_{i<j}^{k} |\{(u,v) \in E \mid u \in V_i, v \in V_j\}|$ (the objective). The term good solution quality refers to small edge cuts. An archetypical application of graph partitioning is optimizing data placement to minimize communication under load-balance.

All state-of-the-art solvers leverage the multilevel scheme (Fig. 1). To *coarsen* $G$, small vertex clusters are contracted repeatedly to obtain a hierarchy of smaller but structurally similar graphs. After computing an initial partition on the smallest graph, the contractions are undone in reverse order, projecting the current partition to the next finer graph and improving the partition by moving vertices, using refinement algorithms. This approach is both faster as we perform global optimization on the top-level graph through local moves on the coarsened graphs. In order to achieve non-trivial speedups, parallel refinement algorithms must move vertices in parallel. This introduces challenges such as guaranteeing the partition is balanced, inaccurate gain values (difference in objective function) due to adjacent vertices moving concurrently, and fine-grained decision dependencies. For example, the Fiduccia-Mattheyses (FM) algorithm repeatedly moves the vertex with *highest* gain (possibly negative), which is challenging to parallelize.

## 2 Mt-KaHyPar

Mt-KaHyPar is a shared-memory parallel multilevel hypergraph partitioning algorithm that comprises fast parallel versions of the core techniques previously used in the state-of-the-art sequential codes: flow-based refinement [4], $n$-level coarsening [5], FM refinement and pre-clustering [7]. As such, it is currently the fastest hypergraph partitioning algorithm and offers the highest solution quality. To parallelize FM, we follow a *localized* strategy, starting independent searches in different regions that gradually expand to neighbors of moved vertices. A recent improvement is *unconstrained refinement* [8], which achieves substantially better edge cuts on irregular graphs by allowing temporary balance violations.

The framework is designed for high reliability. We offer balance guarantees via a high-quality rebalancer [8] and atomic balance checks, mitigate the effects of data races during refinement via the *attributed gains* technique [7] and provide a deterministic mode for reproducible results [2]. Furthermore, Mt-KaHyPar packs a variety of ergonomic features for real-world application tasks. This includes the ability to provide fixed vertices and non-uniform maximum block weights as additional inputs for the partitioning, as well as support for many different objectives and an API for custom objectives. Specialized data structures achieve a 1.75x speedup when partitioning graphs instead of hypergraphs [3]. Library interfaces for C, C++ and Python allow easy integration into any codebase.

## 3 KaMinPar

KaMinPar is the fastest multilevel graph partitioning algorithm available today. It supports both shared-memory and distributed-memory parallelism. The biggest advantage over prior systems is for large values of $k$ (e.g., $k \geq 1\,000$), where our *deep multilevel*

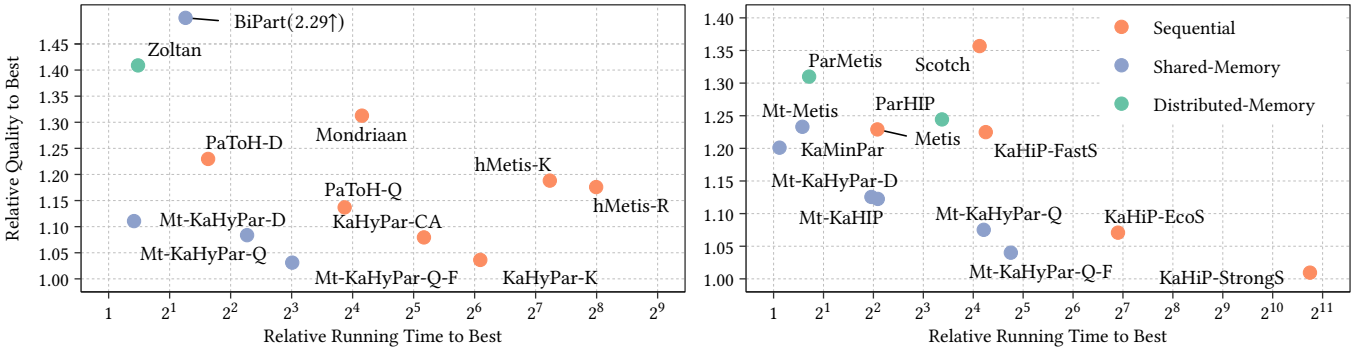


**Figure 2: Pareto plots comparing solution quality and running time relative to the best for each instance; aggregated with harmonic mean; hypergraphs on the left, graphs on the right. Parallel codes are run on 10 cores. See [3] for setup details.**

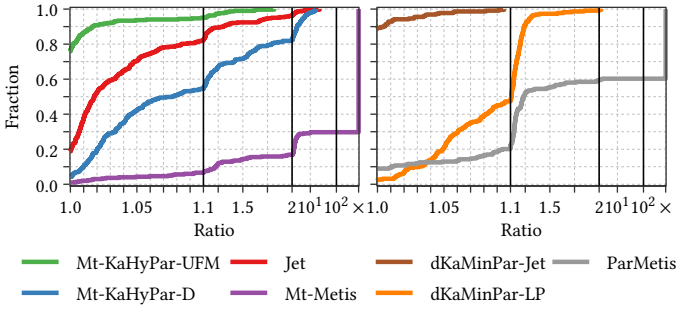


**Figure 3: Performance profiles for unconstrained FM on irregular graphs (left) and distributed Jet (right).**

*scheme* [6] eliminates a significant bottleneck in initial partitioning, leading to a larger than 5× speedup over competitors.

With deep multilevel we coarsen down to $2C$ vertices (instead of $kC$) and fuse initial partitioning into the uncoarsening stage by extending the current partition to a $k' \leq k$ appropriate for the current graph size. Overall we perform a single near-linear work cycle of coarsening and uncoarsening instead of $\log(k)$ cycles.

For coarsening, we use label propagation clustering with additional two-hop matching to ensure coarsening progress on irregular graphs. Starting from each vertex in its own cluster, label propagation visits vertices in random order in parallel; a vertex joins a cluster with the plurality of its neighbors subject to a size constraint. In our contraction algorithm we use a two-level buffering data structure to avoid generating edges twice for the coarse graph's CSR. For initial bipartitioning, we implement a portfolio of 7 randomized sequential greedy graph growing heuristics and 2-way FM. In the refinement stage, we also utilize size-constrained label propagation, starting with the given partition. As optional components, we offer parallel localized $k$-way FM refinement in shared-memory mode and a distributed version of Jet refinement [1].

## 4 Evaluation

In Fig. 2 we summarize the landscape of multilevel algorithms with a Pareto plot comparing running time versus solution quality, lower and to the left is better. For hypergraphs (left), Mt-KaHyPar configurations occupy all spots on the Pareto front, with -D (default, $\log(n)$ coarsening levels, parallel FM) being the fastest and -Q-F ($\approx n$ coarsening levels, flow-based refinement) having the best quality. For graphs (right), Mt-KaHyPar, KaMinPar and KaHiP (strong-social) occupy the spots on the Pareto front, with KaMinPar opting for a faster, but lower quality configuration. Respectively, Mt-KaHyPar and KaMinPar achieve self-relative speedups of 22.3x and 27.9x on 64 cores. Distributed KaMinPar scales to at least 8192 cores [9].

Fig. 3 shows substantial quality improvements on irregular graphs due to recent advances in unconstrained refinement. These plots relate the fraction of instances ($y$) for which a specific algorithm achieves a quality within factor $x$ of the best found solution for that instance, with $x = 1$ showing the fraction of best solutions contributed. Mt-KaHyPar with U(nconstrained) FM beats Jet [1], which beats Mt-KaHyPar-D [8]. Similarly, the distributed Jet refinement integrated in dKaMinPar significantly outperforms the previous LP-based dKaMinPar. The partitions computed by Mt-Metis and ParMetis are largely imbalanced (marked with ×) and thus invalid.

## 5 Outlook

We highlight two upcoming improvements. Leveraging compressed data structures, KaMinPar will soon be able to partition graphs with several hundred billion edges on a single machine, without using external memory or streaming. Moreover, we are improving the deterministic partitioning mode in Mt-KaHyPar to match the state-of-the-art solution quality of the non-deterministic mode.

*Acknowledgements.* Parts of this work were performed on the HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882500).

## References

[1] Michael S. Gilbert, Kamesh Madduri, Erik G. Boman, and Sivasankaran Rajamanickam. 2023. Jet: Multilevel Graph Partitioning on GPUs. *arxiv* (2023).
[2] Lars Gottesbüren and Michael Hamann. 2022. Deterministic Parallel Hypergraph Partitioning. In *Euro-Par*.
[3] L. Gottesbüren, T. Heuer, N. Maas, P. Sanders, and S. Schlag. 2024. Scalable High-Quality Hypergraph Partitioning. *ACM Transactions on Algorithms* (2024).
[4] Lars Gottesbüren, Tobias Heuer, and Peter Sanders. 2022. Parallel Flow-Based Hypergraph Partitioning. In *SEA*.
[5] Lars Gottesbüren, Tobias Heuer, Peter Sanders, and Sebastian Schlag. 2022. Shared-Memory *n*-level Hypergraph Partitioning. In *ALENEX*.
[6] Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, and Daniel Seemaier. 2021. Deep Multilevel Graph Partitioning. In *ESA*.
[7] Lars Gottesbüren, Tobias Heuer, Peter Sanders, and Sebastian Schlag. 2021. Scalable Shared-Memory Hypergraph Partitioning. In *ALENEX*.
[8] Nikolai Maas, Lars Gottesbüren, and Daniel Seemaier. 2024. Parallel Unconstrained Local Search for Partitioning Irregular Graphs. In *ALENEX*.
[9] Peter Sanders and Daniel Seemaier. 2023. Distributed Deep Multilevel Graph Partitioning. In *Euro-Par*.