Partitioning & Clustering Big Graphs

George Karypis
Department of Computer Science & Engineering
Twin Cities
University of Minnesota

Overview

- Overview of graph partitioning
- The multilevel paradigm
- METIS family of partitioning tools
- Multi-threaded algorithms for partitioning & clustering
- Closing remarks

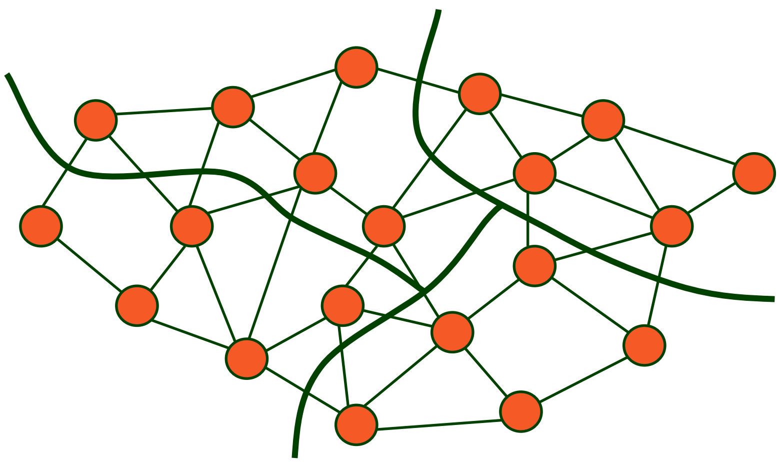
Standard graph partitioning problem

Given a graph *G=(V, E)* we want to partition it into *k* parts such that:

each part has roughly the same number of vertices

and

the edges that straddle partitions (edge-cut) is minimized

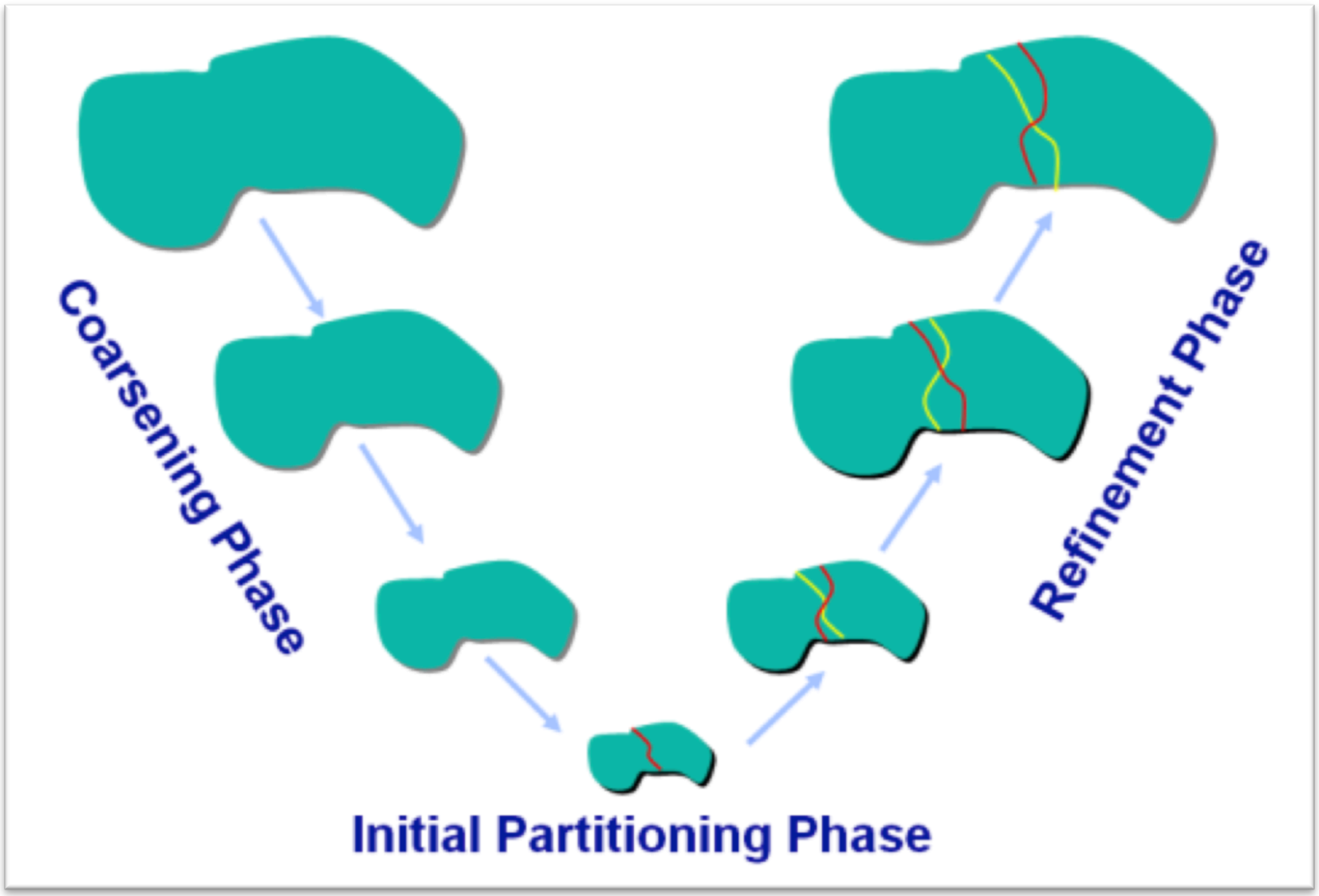


Applications

- Parallel & distributed computing
- Scientific computing
- VLSI physical design
- Data-mining
- Storage and placement
- …

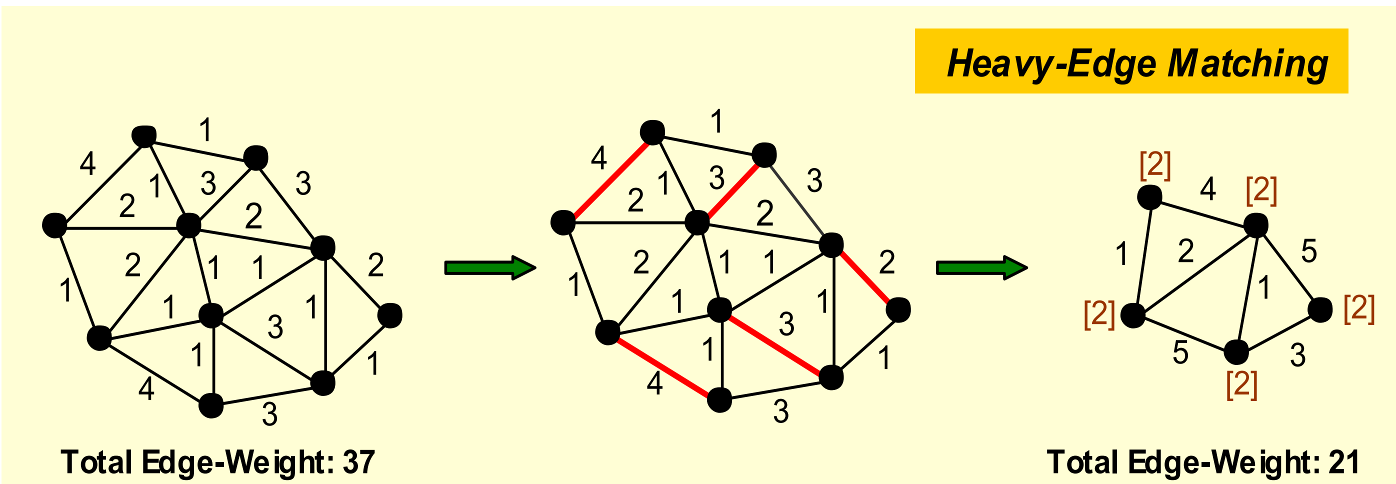
It is NP-hard. Heuristic algorithms are used!

Overview of the multilevel graph partitioning paradigm



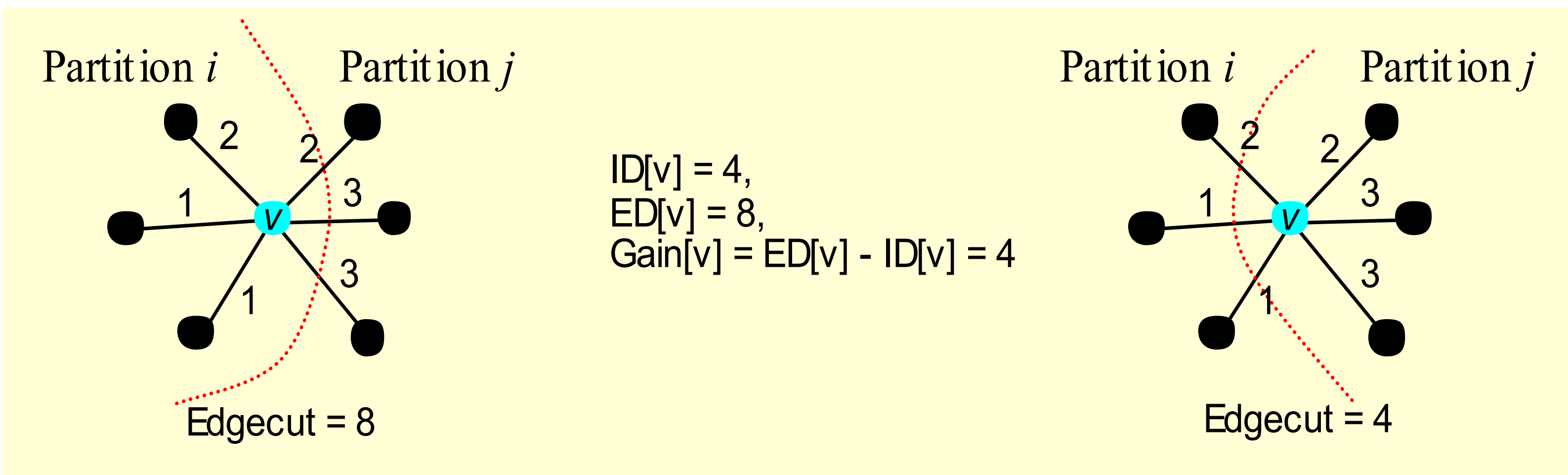
Coarsening Phase

Successive coarse graphs are constructed by computing a matching of the edges, and collapsing together the vertices incident on these edges.



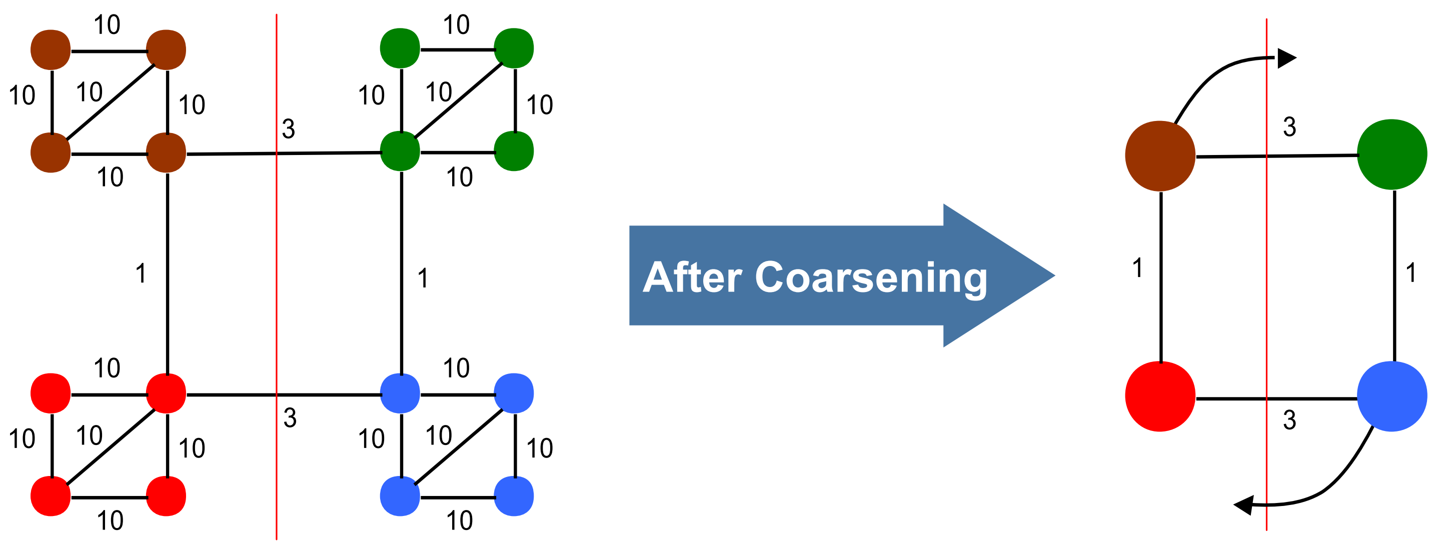
Refinement Phase

The refinement is performed by using *move-based* approaches, based on the algorithm by Fiduccia-Mattheyses (FM).



Why does the multilevel partitioning paradigm work?

- The coarsening phase by hiding a large fraction of the edges, makes the partitioning problem easier.
- Performing refinement at successive finer graphs, enhances the effectiveness of refinement algorithms.
  - Multi-scale refinement

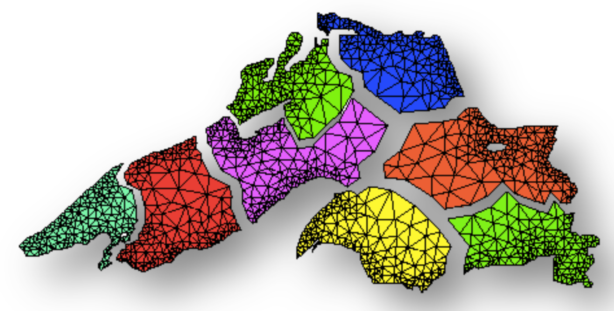


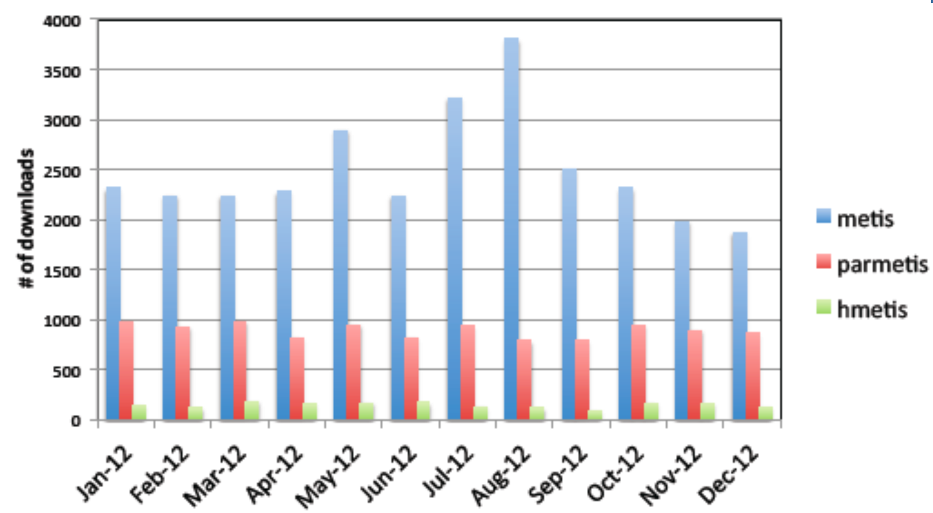
When does the multilevel paradigm have difficulties?

- The value of the objective function in the original graph cannot be (tightly) upper bounded while operating on a coarser graph.
  - We cannot ensure improvements at a coarse graph lead to improvements in the original graph.
- Coarsening fails to make the optimization problem easier in coarser graphs.
  - It is not "in tune" with the objective.
- Coarsening fails to reduce the size of the problem (|V|+|E|).
  - Can increase the runtime/memory requirements.
- The objective function is based on global properties of the graph.
  - Can substantially increase the refinement time.

METIS, ParMETIS, & hMETIS

- Software packages for partitioning unstructured graphs and hypergraphs and computing fill reducing orderings.
  - METIS was released in 1995 (current version 5.x).
  - ParMETIS was released in 1997 (current version 4.x).
  - hMETIS was released in 1998 (current version 2.x)
- They are freely distributed and widely used.





Beyond the traditional partitioning problem (1)

- Vertex separators
  - Partition the graph by removing a minimum set of vertices.
  - Broad applications to:
    - matrix reordering for direct solvers
    - concurrency extraction by decoupling computations at each partition
    - overlapping clustering solutions

Beyond the traditional partitioning problem (2)

- Constraints
  - Multiple balancing constraints
    - balance load & memory requirements,
    - balance the different types of modules that are assigned to each chip in a multi-chip FPGA design,
    - balance incoming & outgoing messages,
    - balance iterative & direct solvers, etc.
  - Connectivity constraints
    - ensure that the graph induced by the vertices of each partition is connected.
  - Placement constraints
    - ensure that certain vertices are placed in different and/or the same partitions.
  - No constraints
    - Objective driven partitioning.

Beyond the traditional partitioning problem (3)

- Objectives
  - Communication volume
  - Subdomain connectivity
  - Redistribution overhead
  - Multiple edge-defined cost functions
  - Path-based objectives
    - timing considerations in VLSI circuits
  - Clustering objectives
    - normalized cut, ratio cut, min-max, modularity, ... [CLUTO]
  - Various combinations of the above

Some performance numbers

Table: Test graph statistics.

Graph	$\lvert V \rvert$	$\lvert E \rvert$
ljournal-2007	5.3M	49.5M
uk-2002	18.5M	261.8M
uk-2007	105.9M	3.3B

Table: Performance on some graphs.

Graph	50/%cdges/time		100/%cdges/time		200/%cdges/time	
ljournal-2007	31.2%	2.4m	35.9%	2.7m	37.0%	3.1m
uk-2002	1.1%	1.3m	1.3%	1.4m	1.4%	1.5m
uk-2007	0.4%	22.0m	0.5%	21.1m	0.5%	21.6m

Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz, 128GB

IMPROVING SINGLE NODE PERFORMANCE

Multi-threaded graph partitioning/clustering

- Opportunities:
  - Multi-core processors have become ubiquitous.
  - Their cache-coherent shared-memory architecture makes it easier to develop parallel programs
- Challenges:
  - Non-uniform access to shared memory.
  - Many applications are bound by memory bandwidth.
  - Limited memory per core.

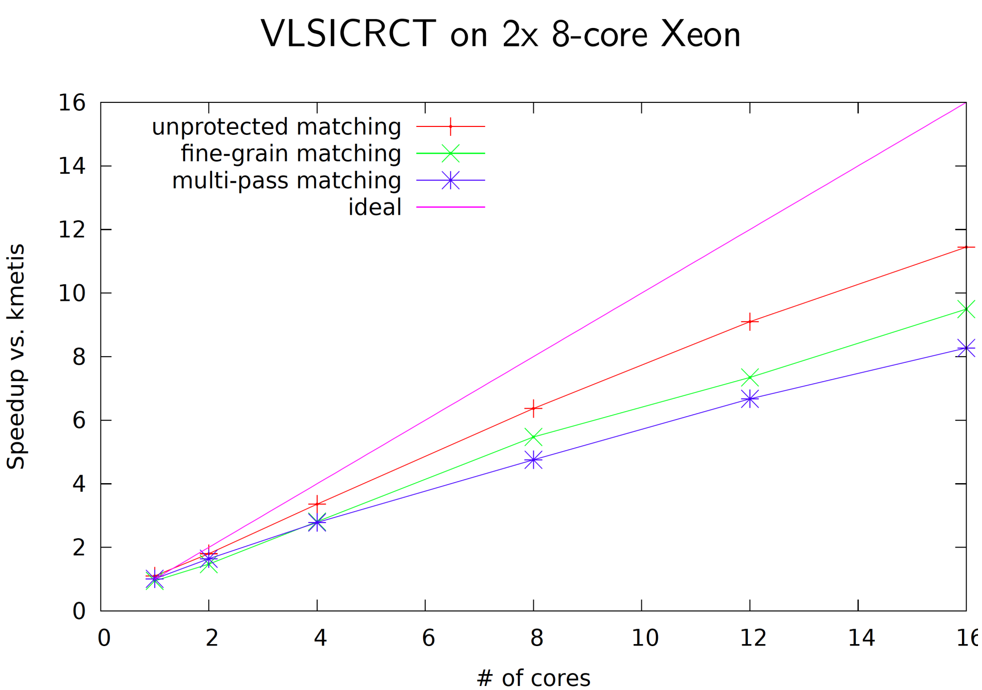
System	Cores / Node	Memory / Node	Memory / Core
Titan	16	32 GB	2 GB
Sequoia	16	16 GB	1 GB
K-Computer	8	16 GB	2 GB
Mira	16	16 GB	1 GB
JUQUEEN	16	16 GB	1 GB

Kmetis – Serial algorithm

- Graph is stored in CSR format.
- Coarsening:
  - Matches connected pairs of vertices for contraction.
  - Vertices are matched in ascending order of degree.
  - Edges prioritized for collapsing by edge weight.
- Initial partitioning:
  - The best of several partitionings generated via recursive bisection is chosen.
- Uncoarsening:
  - Projection of the partitioning.
  - Greedy k-way refinement using only boundary vertices.

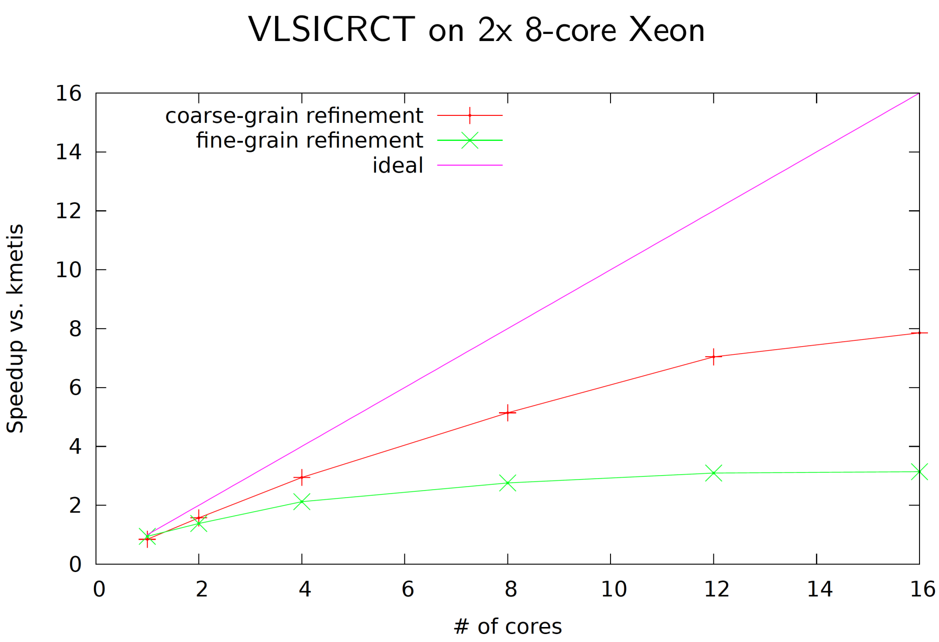
Parallel matching approaches

- Fine-grain matching
  - Lock the vertex and prospective vertices to match with.
- Multi-pass matching
  - First pass to generate match requests.
  - Second pass to grant or deny match requests.
- Unprotected matching
  - Allow threads to modify the matching without any locking.
  - Fix broken matches after matches are selected.

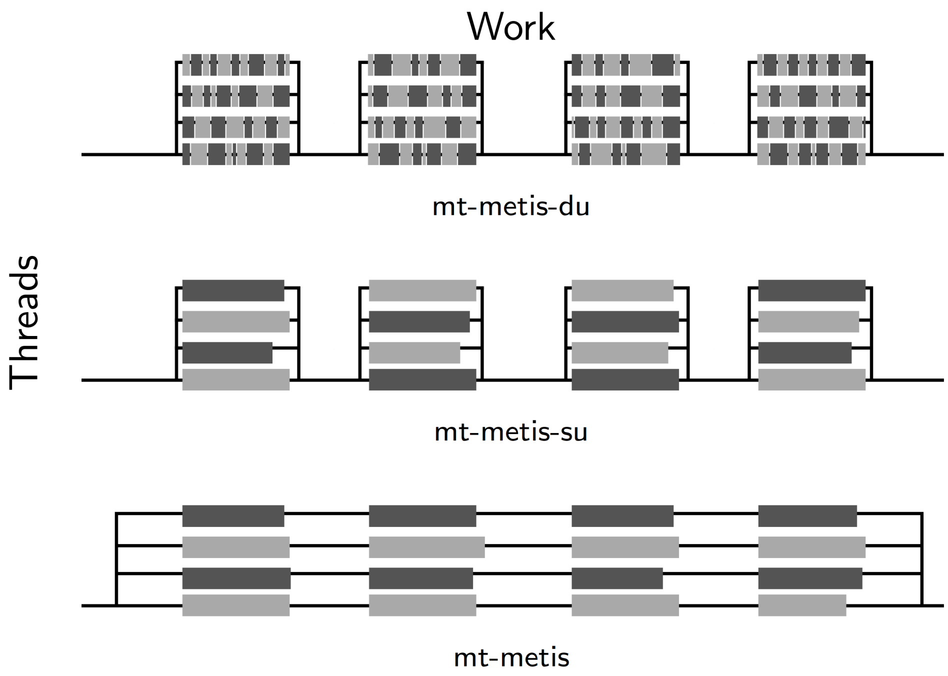


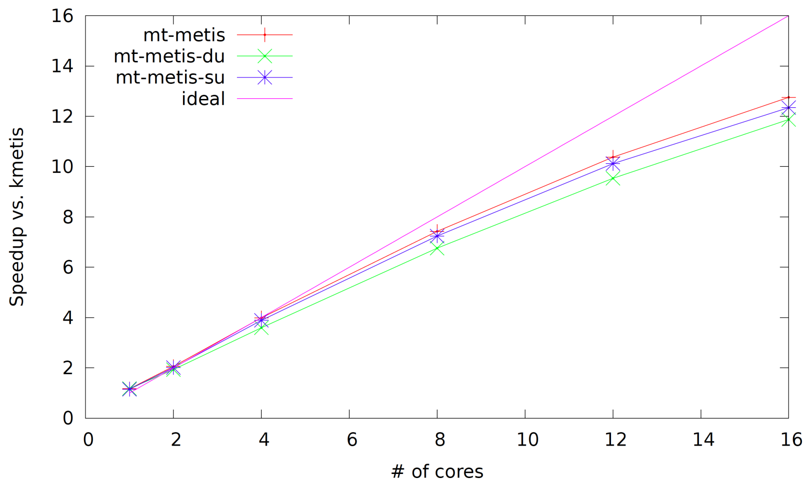
Parallel refinement approaches

- Fine-grain approach
  - Locks to and from partitions along with neighbor vertices.
- Coarse-grain approach
  - Each thread moves up to *c* vertices.
  - Potential partition weights are communicated.
  - Moves are discarded until a balanced partitioning would result.
  - Vertex book-keeping information and partition weights are updated.
  - Repeat until all of the priority queues are empty.

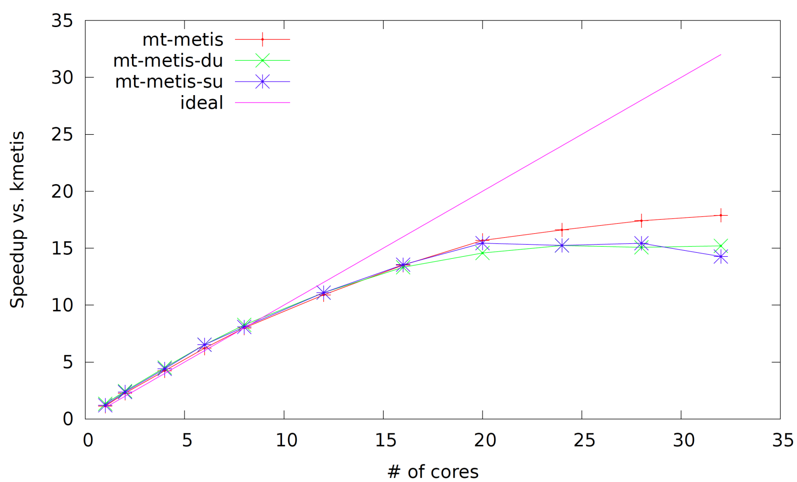


Thread lifetimes and data ownership



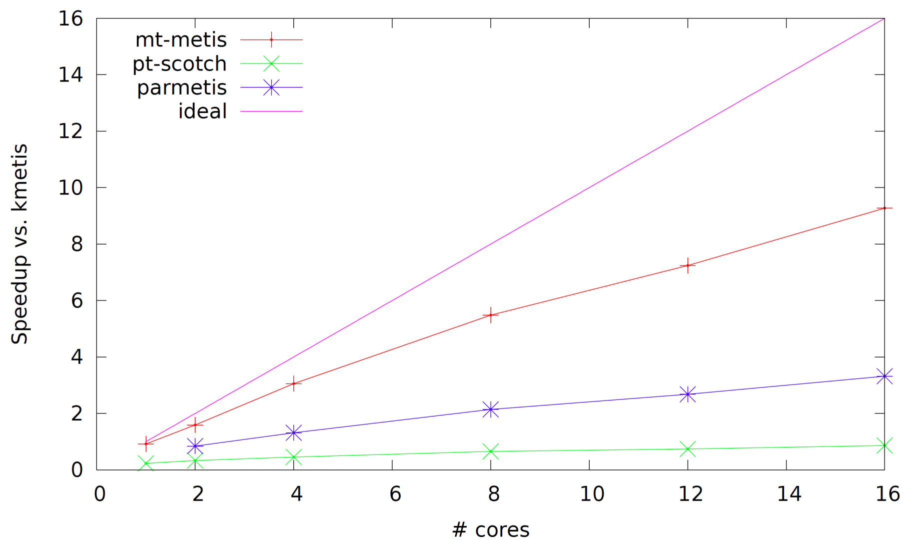


(a) 2x 8-core Xeon

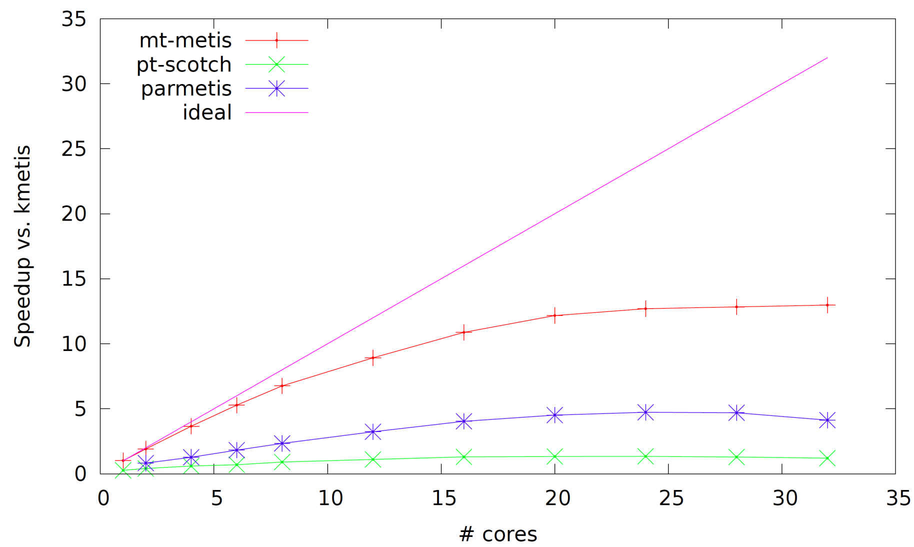


(b) 8x 4-core Opteron

Overall performance – Mean speedup over kmetis

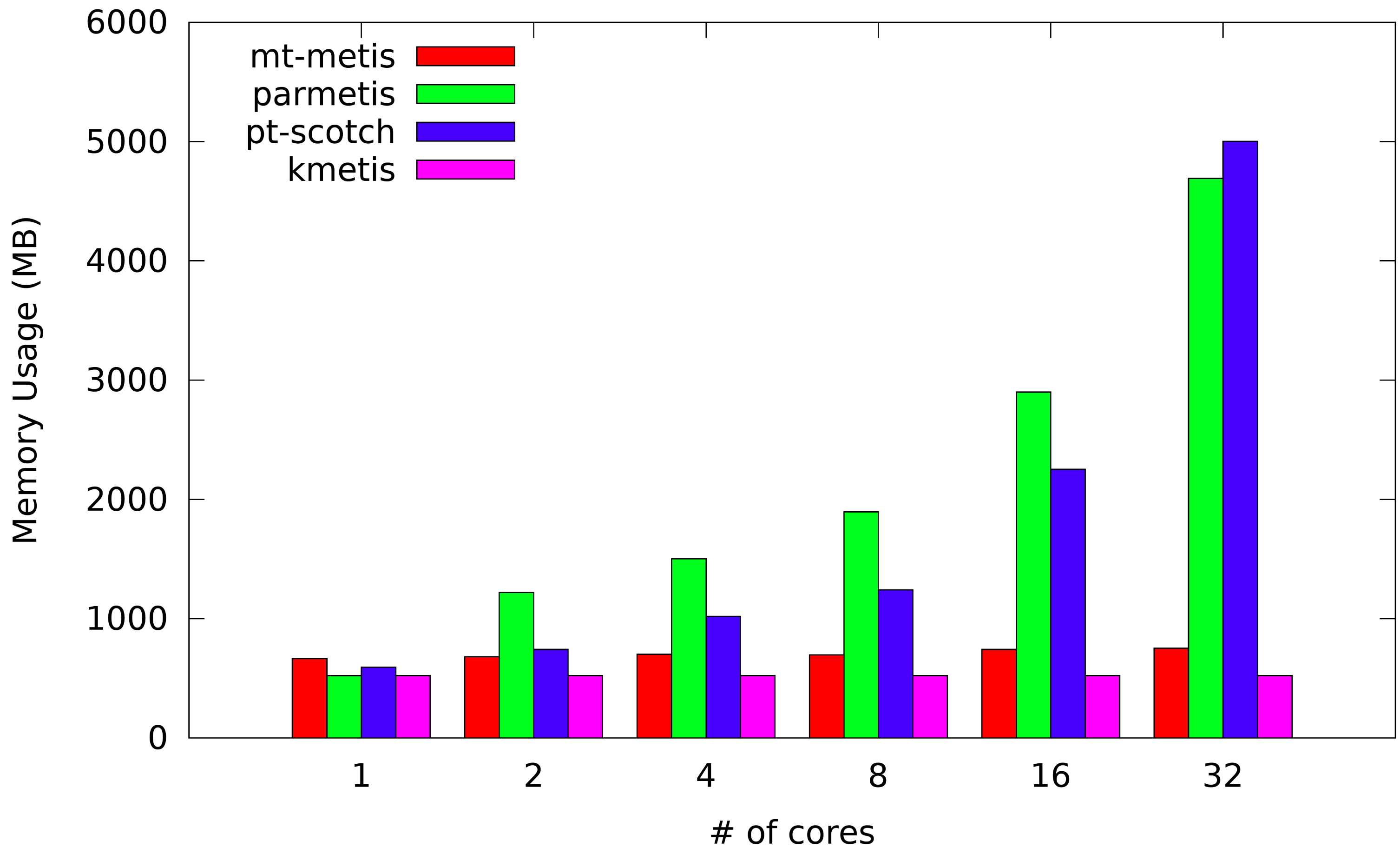


(c) 2x 8-core Xeon



(d) 8x 4-core Opteron

Memory usage



Graph clustering

- Goal:
  - Develop a multi-threaded multi-level modularity maximization graph clustering algorithm.
- Key differences/contributions
  - No need to worry about balance constraints.
  - Focus on coarsening heuristics that are suited for “big data” types of graphs.
    - First-choice & two-hop matching.
  - The objective is not locally defined.
    - Focus on “cautiously” optimistic heuristics with frequent global updates.
  - Automatically determine the right number of clusters during initial clustering.

Some results

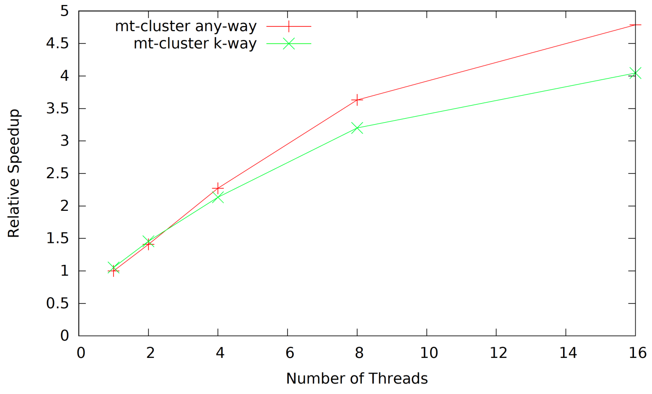
Table: Graphs Used in Scaling Experiments

Graph	# Vertices	# Edges
road_usa	23,947,347	28,854,312
soc-LiveJournal1	4,846,609	42,851,237
europe.osm	50,912,018	54,054,660
uk-2002	18,520,486	261,787,258

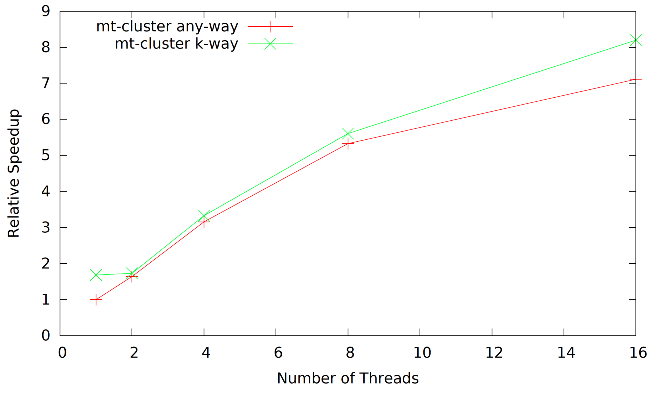
Table: Clustering Times (s)

Graph	1 Thread	8 Threads	16 Threads
road_usa	20.66	5.69	4.32
soc-LiveJournal1	39.77	7.47	5.59
europe.osm	42.89	12.52	10.33
uk-2002	202.43	44.76	35.87

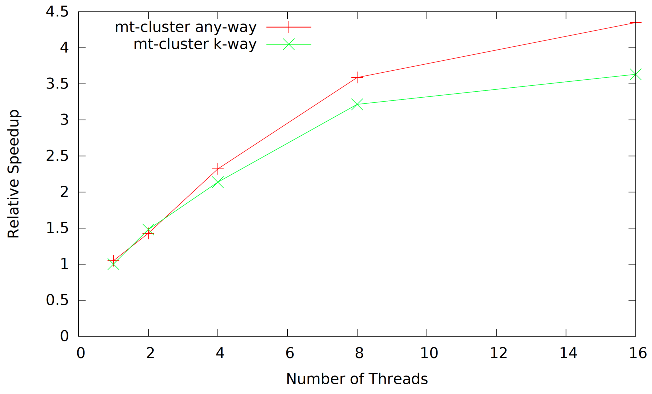
Scaling results



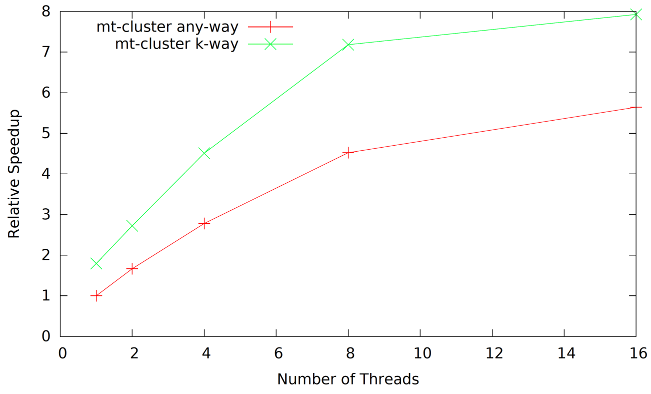
road_usa



soc-LiveJournal1



europe.osm



uk-2002

Final words

- Lessons from mt-metis/mt-cluster
  - The shared memory opportunity
    - Explicit shared memory programing leads to reduced overheads.
    - Reduced memory foot-print.
  - Data locality is still important.
  - Synchronization should be avoided.
  - Lessons learned from MP algorithms are still valid!

- Thank you.
  - http://www.cs.umn.edu/~metis