Multilevel $k$-way Partitioning Scheme for Irregular Graphs

G. Karypis and V. Kumar

Trevor Steil

University of Minnesota

March 7, 2017
Table of Contents

1 Introduction

2 Algorithm
   • Coarsening Phase
   • Initial Partitioning
   • Uncoarsening Phase

3 Results
Graph Partitioning

Given a graph $G = (V, E)$ with $|V| = n$, we want to partition $V$ into subsets $V_1, \ldots, V_k$ such that

- $V_i \cap V_j = \emptyset$ for $i \neq j$
- $\bigcup_{i=1}^{k} V_i = V$
- $|V_i| = \frac{n}{k}$ for all $i$ (approximately at least)
- edge-cut is minimized

where edge-cut is the number of edges between vertices of different subsets of partition
Uses of Graph Partitioning

- Assigning tasks to processors in parallel computation
- Sparse matrix-vector multiplication

![Graph Partitioning Diagram](image)

**Figure 3.6** A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task $i$ computes $\sum_{0 \leq j \leq 11, A[i,j] \neq 0} A[i,j] \cdot b[j]$.

From *Introduction to Parallel Computing* by A. Grama, A. Gupta, G. Karypis, and V. Kumar
Can repeatedly partition graph in two

- Recursive Bisection
- Recursive nature leads to logarithmic term in running time ($O(|E| \log k)$)
- Partitioning graph in two is still difficult to do.
Multilevel k-Way Partitioning

Idea:
It is easier to partition smaller graphs

Instead of trying to partition original graph, try reducing to a smaller graph.
Basic Algorithm Steps

1. Successively coarsen graph
2. Partition coarsest graph
3. Uncoarsen graph and refine partition
Let $G_0 = (V_0, E_0) = (V, E)$ be the original graph. Want a sequence $G_i = (V_i, E_i)$ with $|V_i| < |V_{i-1}|$.

Need to add weights to vertices and edges to keep track of number of original vertices and edges are in each coarsened vertex and edge.
Use of Matchings

Collapse along edges of a matching to coarsen graph

Randomized Matching Algorithm:

- Visit vertices in random order
- At each unmatched vertex, choose a random unmatched neighbor (if any exist)
- Add corresponding edge to matching

Visit Order: 1, 2, 3, 4, 5
Use of Matchings

Collapse along edges of a matching to coarsen graph

Randomized Matching Algorithm:

- Visit vertices in random order
- At each unmatched vertex, choose a random unmatched neighbor (if any exist)
- Add corresponding edge to matching

Visit Order: 1, 2, 3, 4, 5
Use of Matchings

Collapse along edges of a matching to coarsen graph

Randomized Matching Algorithm:

- Visit vertices in random order
- At each unmatched vertex, choose a random unmatched neighbor (if any exist)
- Add corresponding edge to matching

Visit Order: 1, 2, 3, 4, 5
Use of Matchings

Collapse along edges of a matching to coarsen graph

Randomized Matching Algorithm:
- Visit vertices in random order
- At each unmatched vertex, choose a random unmatched neighbor (if any exist)
- Add corresponding edge to matching

Visit Order: 1, 2, 3, 4, 5
Heavy Edge Matching

Coarsening step reduces total edge weight by edge weights of matching
  • To minimize edge cut, choose edges that maximize weight of matching

Algorithm:
  • Visit vertices in random order
  • At each unmatched vertex, choose unmatched neighbor along heaviest edge
  • Add corresponding edge to matching

Has same computational complexity as randomized matching \( O(|E|) \)
Partition $G_n = (V_n, E_n)$ into $k$ subsets after several coarsening steps using any method

- i.e. recursive bisection, spectral bisection, etc.

Want vertex weights to be approximately equal among partitions
Uncoarsening Phase

Basic Steps:

- Take partitioning of $G_i$ to a partitioning of $G_{i-1}$ by reversing the matching and collapse of coarsening that sent $G_{i-1}$ to $G_i$.
- Refine partition to have lower edge cut

Several refining algorithms exist, based on various heuristics.
Kernighan-Lin (KL) Algorithm

Used for bisection (not $k$-way partitioning)

For a vertex $v$, let $gain(v)$ be the decrease in edge-cut when $v$ is moved to other partition.

**KL Algorithm**

1: Compute $gain$ for all vertices
2: Create priority queue for each partition based on $gain$
3: for $v$ with highest gain do
4: Move $v$ to other partition
5: Update $gain$ of neighbors of $v$
6: Record current edge-cut
7: Lock $v$ so it cannot be moved again
8: end for
9: Choose configuration that had lowest edge-cut
Let $W_i[j] =$ weight of partition $j$ in $G_i$, $W^{min} = 0.9|V_0|/k$ and $W^{max} = C|V_0|/k$ for some $C$. Only allow vertex $v$ to move from partition $a$ to partition $b$ if

$$W_i[b] + w(v) \leq W^{max}$$

and

$$W_i[a] - w(v) \geq W^{min}$$

where $w(v)$ is the weight of vertex $v$. 

Using the straightforward generalization is not practical for large $k$

- Each vertex is in one of $k$ subsets and can move to any of $k - 1$ other subsets
- Requires $k(k - 1)$ queues
Lookahead in KL for Bisections

By maintaining queues and moving all vertices, groups of vertices can cross boundary when some individual moves will lead to negative gain

Edge-Cut: 2
Lookahead in KL for Bisections

By maintaining queues and moving all vertices, groups of vertices can cross boundary when some individual moves will lead to negative gain.

Edge-Cut: 3
Lookahead in KL for Bisections

By maintaining queues and moving all vertices, groups of vertices can cross boundary when some individual moves will lead to negative gain

Edge-Cut: 2
Lookahead in KL for Bisections

By maintaining queues and moving all vertices, groups of vertices can cross boundary when some individual moves will lead to negative gain.

Edge-Cut:1
Lookahead for \( k \)-way Partitioning

Heuristically, moving groups of vertices can be accomplished by moving a single vertex in a coarser graph.

A coarser partitioning of the previous example:

![Diagram showing coarser partitioning]

Edge-Cut: 2
Lookahead for $k$-way Partitioning

Heuristically, moving groups of vertices can be accomplished by moving a single vertex in a coarser graph.

A coarser partitioning of the previous example:

Edge-Cut: 1
Greedy Refinement

Greedy Refinement pseudocode

1: for vertices \( v \) on boundary of partitions do
2: \hspace{1em} Move \( v \) to the subset that minimizes edge-cut while maintaining balance condition
3: end for

This refinement step is performed iteratively.
Completed Algorithm

Multilevel $k$-way Partitioning (ML$k$P) Algorithm

1. Coarsen graph using Heavy Edge Matching
2. Partition coarse graph
3. Uncoarsen graph and refine partition using Greedy Refinement
Performance

MLkP:

- produces partitions of the same quality as Multilevel Recursive Bisection (MLRB)
- runs approximately 3x faster than MLRB
