Graph Partitioning

David Bindel

10 Nov 2015

The mountain of abstraction

- Started very low-level (caches, vector units, etc)
- Up to general ideas/kernels (tiling, matrix multiply)
- Up to parallel concepts, application ideas
- Nirvana: high-level description, performance "just happens?"

Low-level frameworks and languages

- OpenMP and MPI (of course)
- Intel Thread Building Blocks (TBB)
- Global arrays
- Newer (?) parallel languages and extensions
  - Cilk++
  - UPC
  - Titanium
  - Chapel

Libraries

One thing (or a few) done fast:

- BLAS (MKL, OpenBLAS, ATLAS, etc)
- LAPACK and successors
- FFTW
- Sparse direct solvers

Key challenge: linking (esp across languages)

Framework libraries

- Many in PDE land
  - PETSc, SLEPc, TAO, etc
  - Trilinos
  - Overture
  - deal.ii
- Interface more complicated than procedure call
- Effectively defines embedded solver language

Key challenge: learning framework + build/link

Runtime frameworks

- Lots of trendy examples
  - MapReduce / Hadoop
  - Pregel, GraphLab, PowerGraph, Ligra, etc
  - Spark
- Write code to match interface desired by framework
- Promise: "Code like this, we'll make it go fast"
  - Great when it works!
  - Sometimes not as fast as you'd hope

Key challenge: map your problem to desired form

Scripting languages and PSEs

- Matlab, Octave, R, Python, Julia
- “High productivity” vs “high performance”?
- Not necessarily slow!
  - Speed via extensions (Cython, MWrap, etc)
  - Speed via Jit (Matlab, Julia, Python Numba)
  - Speed via BLAS3 calls (all of the above)
  - Often some parallel support as well
- Performance strategies transfer
  - Model and understand data access
  - Profile and tune
- Bottlenecks may not be where you expect

Key challenge: map your problem to fit language strengths

Domain specific languages

- Classic example: SQL
- PDE domain: finite element compilers
  - Dolfin framework
  - Sundance
- Embedded languages/specializers (PyCUDA, SEJITS)

Key challenge: great opportunities from limited scope

Simulation codes

- ANSYS, ABAQUS, LS-DYNA, OpenSEES, FEAP, COMSOL, FLUENT, OpenFOAM, SPICE, Cadence, BioSPICE, ...
- Typical pattern
  - Custom language (or preprocessor) for problem input
  - Scripting language to describe analysis
  - User-defined elements/modules in compiled language
- Great for some classes of problems
- Can often be tortured into covering other types!

Key challenge: limited scope

Thinking performance

- Algorithms matter
  - But asymptotics isn't everything
- Memory matters – start with data structures
  - Compact data structures (in cache, avoid pointer-chasing)
  - Careful choice of destructive / non-destructive updates
- Model, profile, tune, repeat

And now for something completely different.

Graph partitioning

Given:

- Graph $G = (V, E)$
- Possibly weights (W_V, W_E).
- Possibly coordinates for vertices (e.g. for meshes).

We want to partition G into k pieces such that

- Node weights are balanced across partitions.
- Weight of cut edges is minimized.

Important special case: $k = 2$.

Types of separators

- *Edge* separators: remove edges to partition
- *Node* separators: remove nodes (and adjacent edges)

Can go from one to the other (easiest if graph is degree-bounded).

Why partitioning?

- Physical network design (telephone layout, VLSI layout)
- Sparse matvec
- Preconditioners for PDE solvers
- Sparse Gaussian elimination
- Data clustering
- Image segmentation

Cost

How many partitionings are there? If n is even,

$$\binom{n}{n/2} = \frac{n!}{((n/2)!)^2} \approx 2^n \sqrt{2/(\pi n)}.$$

Finding the optimal one is NP-complete.

We need heuristics!

Partitioning with coordinates

- Lots of partitioning problems from "nice" meshes
  - Planar meshes (maybe with regularity condition)
  - k-ply meshes (works for $d > 2$)
  - Nice enough $\implies$ partition with $O(n^{1-1/d})$ edge cuts (Tarjan, Lipton; Miller, Teng, Thurston, Vavasis)
  - Edges link nearby vertices
- Get useful information from vertex density
- Ignore edges (but can use them in later refinement)

Recursive coordinate bisection

Idea: Choose a cutting hyperplane parallel to a coordinate axis.

- Pro: Fast and simple
- Con: Not always great quality

Inertial bisection

Idea: Optimize cutting hyperplane based on vertex density

$$
\begin{aligned}
\bar{\mathbf{x}} &= \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \\
\bar{\mathbf{r}}_i &= \mathbf{x}_i - \bar{\mathbf{x}} \\
\mathbf{I} &= \sum_{i=1}^{n} \left[\|\mathbf{r}_i\|^2 I - \mathbf{r}_i \mathbf{r}_i^T \right]
\end{aligned}
$$

Let $(\lambda_n, \mathbf{n})$ be the minimal eigenpair for the inertia tensor $\mathbf{I}$, and choose the hyperplane through $\bar{\mathbf{x}}$ with normal $\mathbf{n}$.

- Pro: Still simple, more flexible than coordinate planes
- Con: Still restricted to hyperplanes

Random circles (Gilbert, Miller, Teng)

- Stereographic projection
- Find centerpoint (any plane is an even partition)
  In practice, use an approximation.
- Conformally map sphere, moving centerpoint to origin
- Choose great circle (at random)
- Undo stereographic projection
- Convert circle to separator

May choose best of several random great circles.

Coordinate-free methods

- Don't always have natural coordinates
  - Example: the web graph
  - Can sometimes add coordinates (metric embedding)
- So use edge information for geometry!

Breadth-first search

- Pick a start vertex v_0
  - Might start from several different vertices
- Use BFS to label nodes by distance from v_0
  - We've seen this before – remember RCM?
  - Could use a different order – minimize edge cuts locally (Karypis, Kumar)
- Partition by distance from v_0

Greedy refinement

Start with a partition $V = A \cup B$ and refine.

- Gain from swapping (a, b) is $D(a) + D(b)$, where

$$D(a) = \sum_{b' \in B} w(a, b') - \sum_{a' \in A, a' \neq a} w(a, a')$$

$$D(b) = \sum_{a' \in A} w(b, a') - \sum_{b' \in B, b' \neq b} w(b, b')$$

- Purely greedy strategy:
  - Choose swap with most gain
  - Repeat until no positive gain
- Local minima are a problem.

Kernighan-Lin

In one sweep:

While no vertices marked
    Choose (a, b) with greatest gain
    Update $D(v)$ for all unmarked v as if (a, b) were swapped
    Mark a and b (but don't swap)
Find j such that swaps $1, \ldots, j$ yield maximal gain
Apply swaps $1, \ldots, j$

Usually converges in a few (2-6) sweeps. Each sweep is $O(N^3)$.
Can be improved to $O(|E|)$ (Fiduccia, Mattheyses).

Further improvements (Karypis, Kumar): only consider vertices on boundary, don't complete full sweep.

Spectral partitioning

Label vertex i with $x_i = \pm 1$. We want to minimize

$$\text{edges cut} = \frac{1}{4} \sum_{(i,j) \in E} (x_i - x_j)^2$$

subject to the even partition requirement

$$\sum_i x_i = 0.$$

But this is NP hard, so we need a trick.

Spectral partitioning

Write

$$\text{edges cut} = \frac{1}{4} \sum_{(i,j)\in E} (x_i - x_j)^2 = \frac{1}{4} \|Cx\|^2 = \frac{1}{4} x^T L x$$

where C is the incidence matrix and $L = C^T C$ is the graph Laplacian:

$$C_{ij} = \begin{cases} 1, & e_j = (i,k) \\ -1, & e_j = (k,i) \\ 0, & \text{otherwise,} \end{cases} \qquad L_{ij} = \begin{cases} d(i), & i = j \\ -1, & i \neq j, (i,j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

Note that $Ce = 0$ (so $Le = 0$), $e = (1, 1, 1, \ldots, 1)^T$.

Spectral partitioning

Now consider the *relaxed* problem with $x \in \mathbb{R}^n$:

$$\text{minimize } x^T L x \text{ s.t. } x^T e = 0 \text{ and } x^T x = 1.$$

Equivalent to finding the second-smallest eigenvalue λ_2 and corresponding eigenvector x, also called the *Fiedler vector*. Partition according to sign of x_i.

How to approximate x? Use a Krylov subspace method (Lanczos)! Expensive, but gives high-quality partitions.

Multilevel ideas

Basic idea (same will work in other contexts):

- Coarsen
- Solve coarse problem
- Interpolate (and possibly refine)

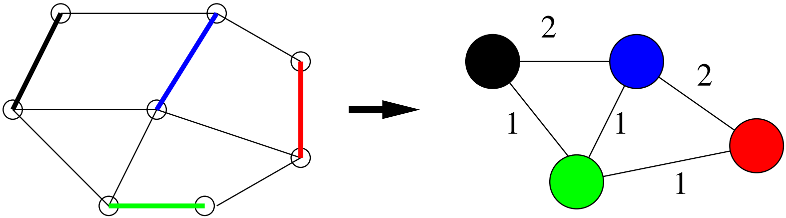
May apply recursively.

Maximal matching

One idea for coarsening: maximal matchings

- *Matching* of $G = (V, E)$ is $E_m \subset E$ with no common vertices.
- *Maximal* if no more edges can be added and remain matching.
- Constructed by an obvious greedy algorithm.
- Maximal matchings are non-unique; some may be preferable to others (e.g. choose heavy edges first).

Coarsening via maximal matching



- Collapse nodes connected in matching into coarse nodes
- Add all edge weights between connected coarse nodes

Software

All these use some flavor(s) of multilevel:

- METIS/ParMETIS (Kapyris)
- PARTY (U. Paderborn)
- Chaco (Sandia)
- Scotch (INRIA)
- Jostle (now commercialized)
- Zoltan (Sandia)

Is this it?

Consider partitioning for sparse matvec:

- Edge cuts $\neq$ communication volume
- Haven't looked at minimizing *maximum* communication volume
- Looked at communication volume – what about latencies?

Some work beyond graph partitioning (e.g. hypergraph in Zoltan).

Is this it?

Additional work on:

- Partitioning power law graphs
- Covering sets with small overlaps

Also: Classes of graphs with no small cuts (expanders)