A medium-grain method for fast 2D bipartitioning of sparse matrices

Daniel M. Pelt (CWI, Amsterdam)
Rob H. Bisseling (Utrecht University)

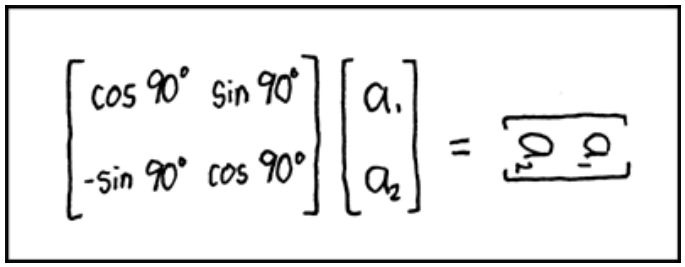
IPDPS 2014

Centrum Wiskunde & Informatica

Universiteit Utrecht

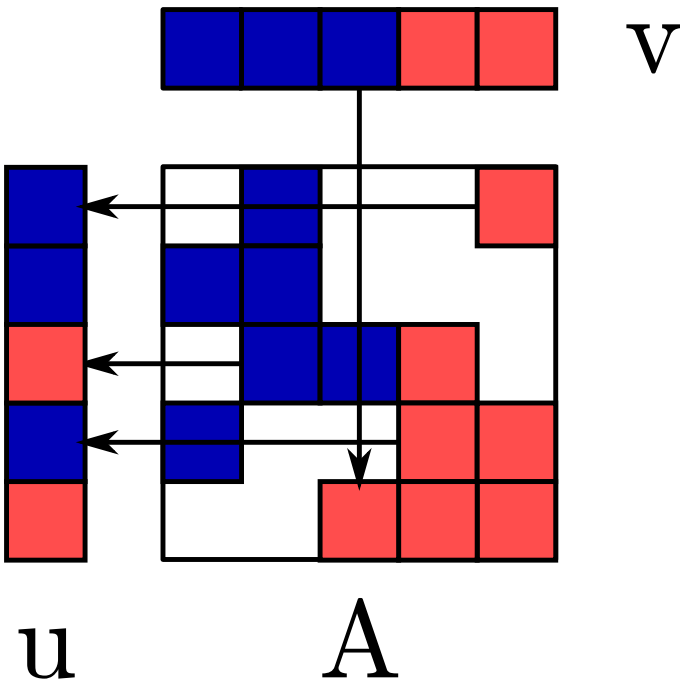
Sparse matrix-vector multiplication

- Matrix-vector multiplications are everywhere:
  - Linear systems, web search, simulations, ...
- Often, these problems are sparse
- However, they are becoming huge:
  - Millions of rows, columns, and nonzeros
- Efficient parallel sparse matrix-vector multiplication methods are needed!



Source: `http://www.xkcd.com`

Parallel sparse matrix-vector multiplication



- Approach: distribute elements of A, $\boldsymbol{u}$, and $\boldsymbol{v}$ over different processors
- Compute $\boldsymbol{u} = A\boldsymbol{v}$ in four phases:
  1. Fan-out: Communicate values of $\boldsymbol{v}$
  2. Multiplication: Perform local multiplication
  3. Fan-in: Communicate partial sums
  4. Summation: Calculate $\boldsymbol{u}$

- Distribute work evenly $\leftrightarrow$ Minimize communication

Sparse matrix partitioning

- Partition an $m \times n$ matrix A with N nonzeros into p parts A_i
- Even work distribution is usually enforced by a load balance constraint ε:

$$\max_i |A_i| \leq (1+\varepsilon)\frac{N}{p} \tag{1}$$

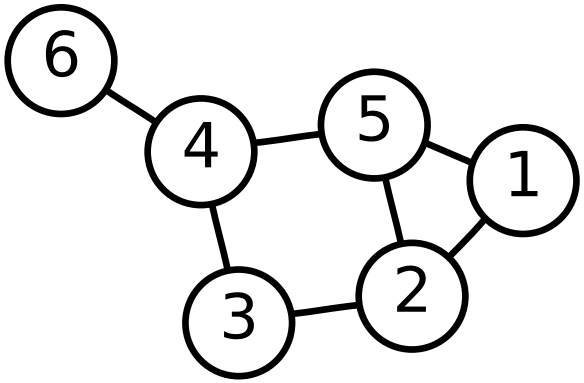
- The communication volume of a single row/column of A is:

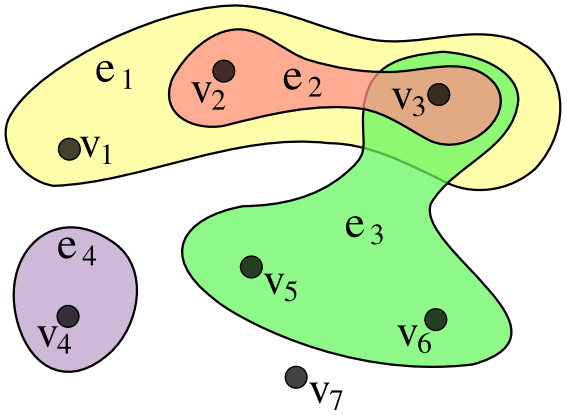
$$C_i = \lambda_i - 1 \tag{2}$$

  λ_i: number of parts that own a nonzero in that row/column
- The sum of C_i's is an attainable lower bound on the total number of elements that need to be communicated

Hypergraph partitioning

- Sparse matrix partitioning is equivalent to hypergraph partitioning
- A hypergraph is a generalization of a graph
  - Consists of vertices and hyperedges (nets)
- Partition vertices into p parts, such that the hyperedge cut is minimized



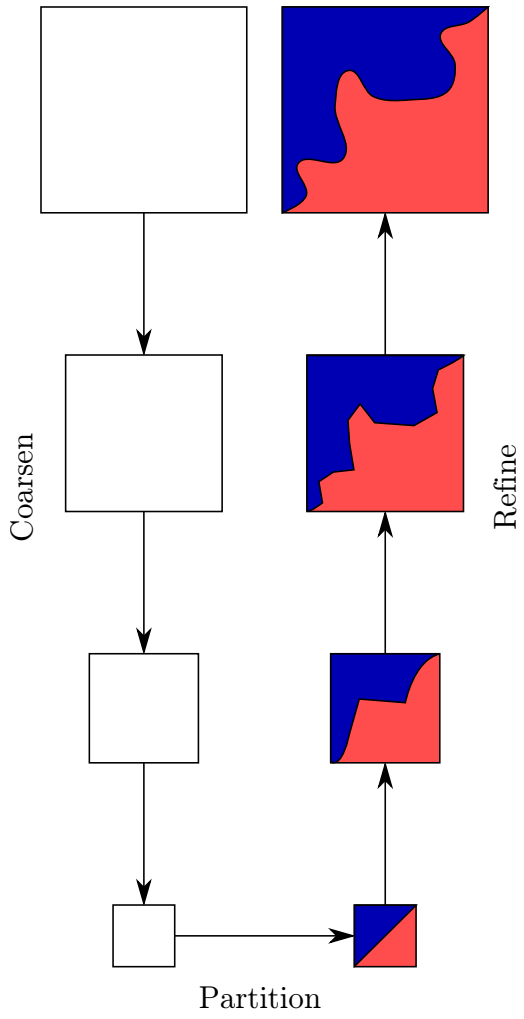


Source: `http://tinyurl.com/lls25e7`

Hypergraph partitioning

- Sparse matrix partitioning is equivalent to hypergraph partitioning
- Several standard models exist:
  - Row-net: each column of A becomes a vertex, each row a net
  - Col-net: each row of A becomes a vertex, each column a net
  - Fine-grain: each nonzero of A becomes a vertex, and each row and column a net
- 1D RN/CN models are efficient, but can be restrictive
- 2D fine-grain is general, but large

Hypergraph partitioning



- Most popular hypergraph partitioners use:
  - A multilevel coarsen & refine scheme
  - The Kernighan-Lin Fiduccia-Mattheyses method (KLFM) to refine
- During coarsening, reduce matrix size, but keep global structure
- Partition the smallest matrix using a (simple) method
- During each step of uncoarsening, refine the partitioning

Medium-grain method

- We propose a method that combines both standard models
- Groups of nonzeros in a single row or column become vertices
- The corresponding hypergraph has at most $m + n$ vertices and $m + n$ nets

	RN/CN	**FG**	**MG**
Vertices	rows/cols	nonzeros	groups
Small	✓		✓
General		✓	✓

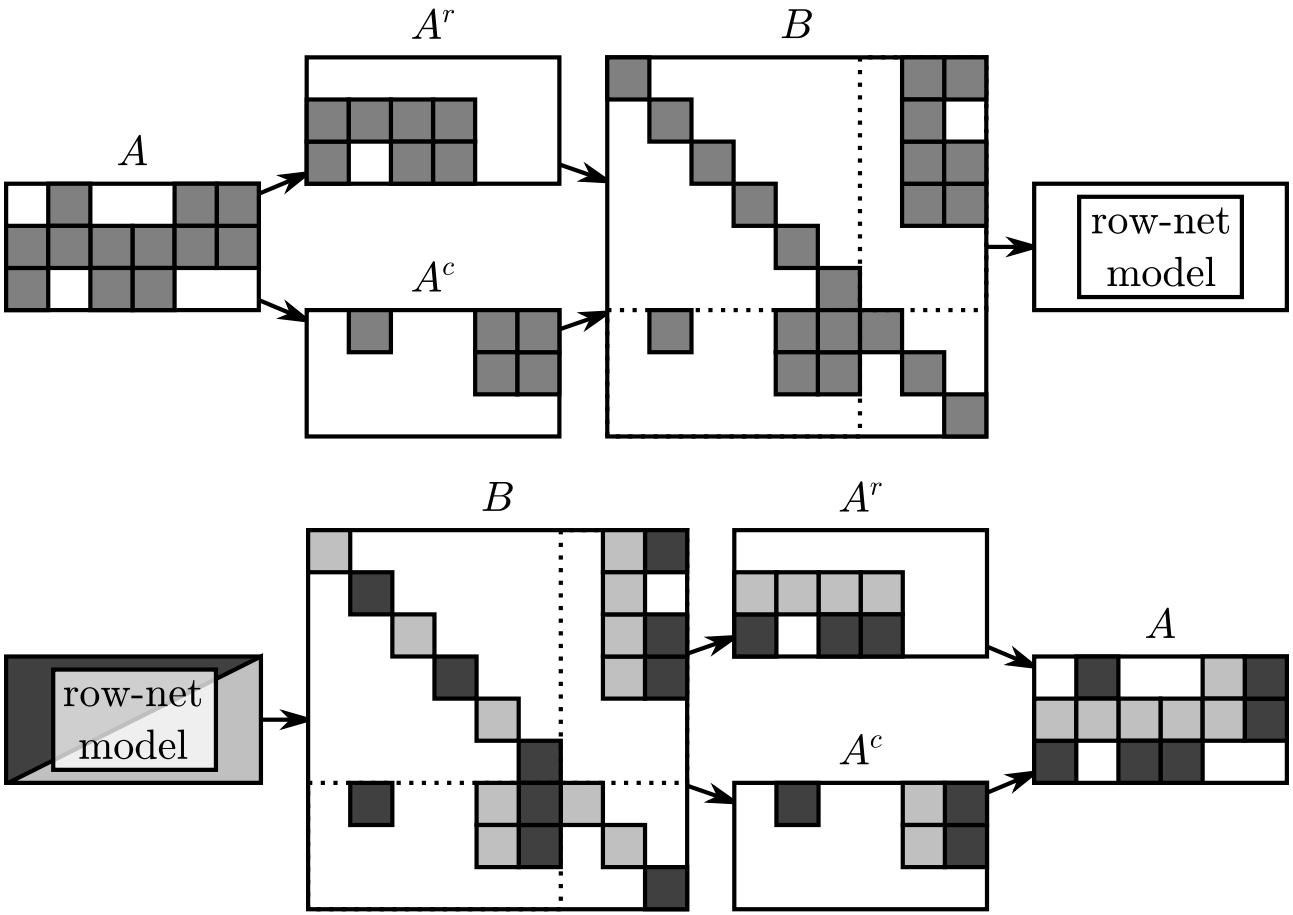
Medium-grain method

1. Split A into two parts: A^c and A^r
   - $A = A^r + A^c$
2. Form a combined $(m+n) \times (m+n)$ matrix B:

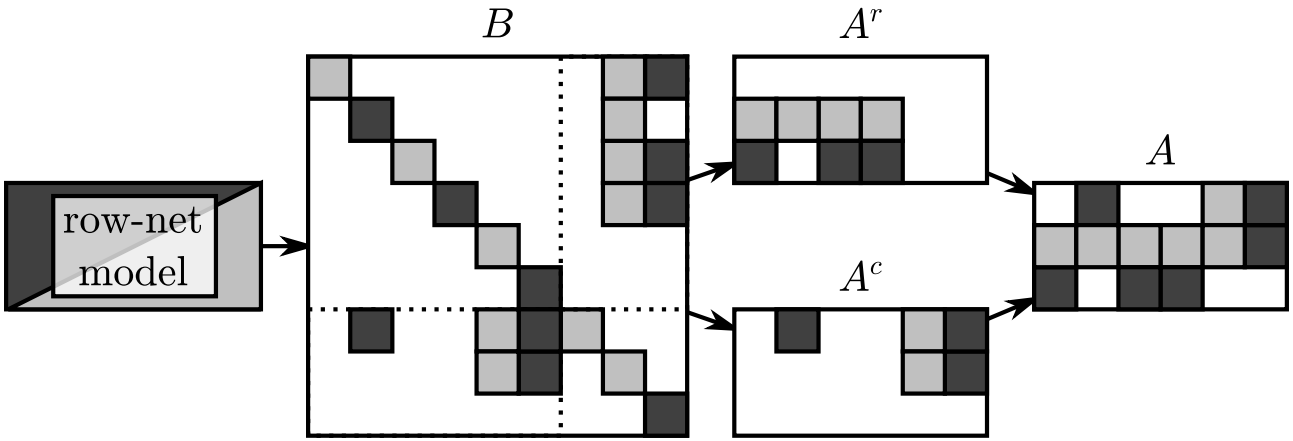
$$B = \begin{bmatrix} I_n & (A^r)^T \\ A^c & I_m \end{bmatrix} \tag{3}$$

3. Apply the row-net model to B
4. Translate partitioned B back to partitioned A

Medium-grain method



Diagonal of B

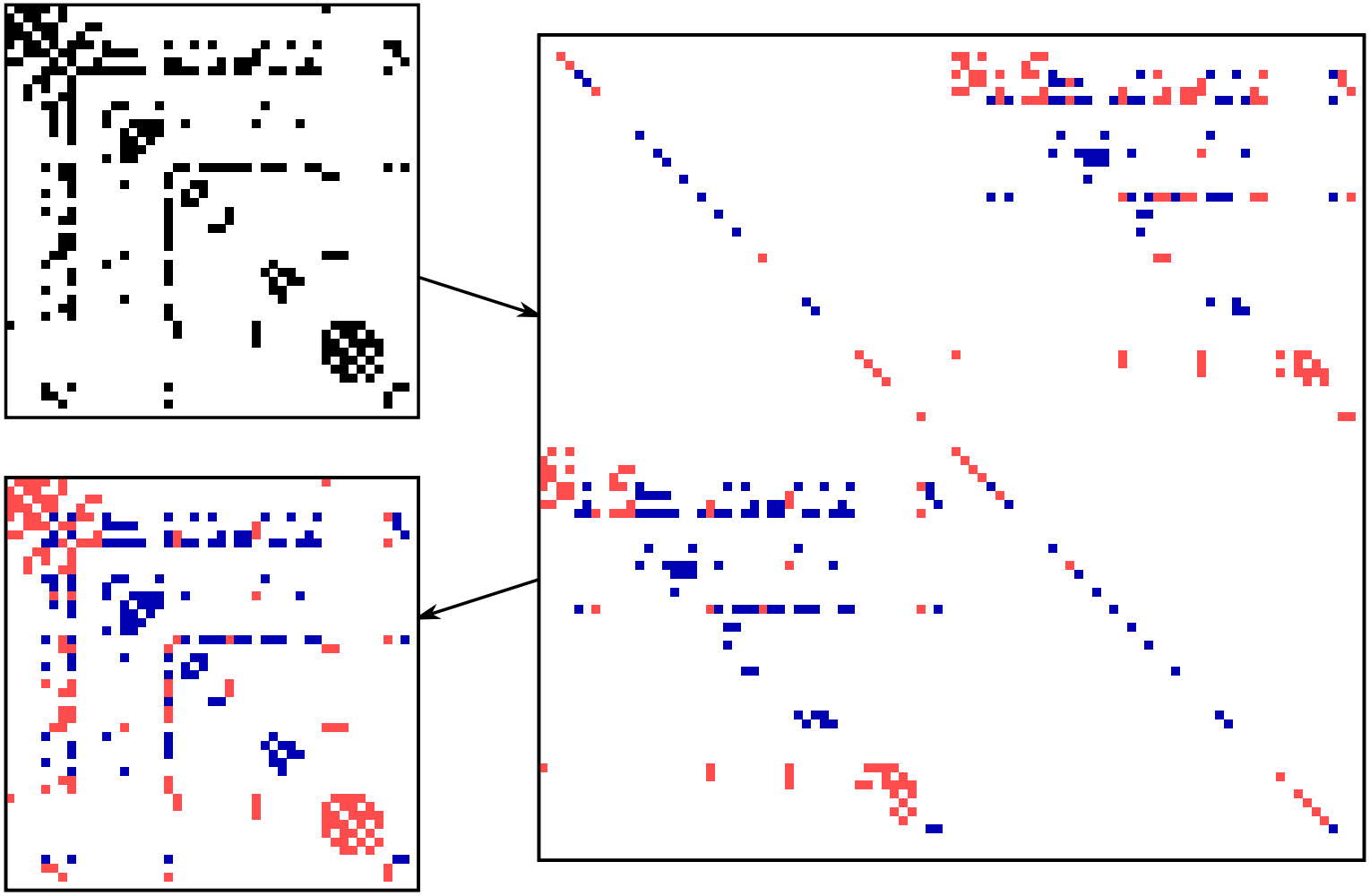


- The diagonal ensures that the comvol of A is identical to B
- Diagonal nonzeros of B are dummies
  - They are not included in the load balance equation
- If a row/column of B only includes the diagonal, it can be removed
  - This is common, so the number of vertices and nets will usually be $< m + n$

Initial split algorithm

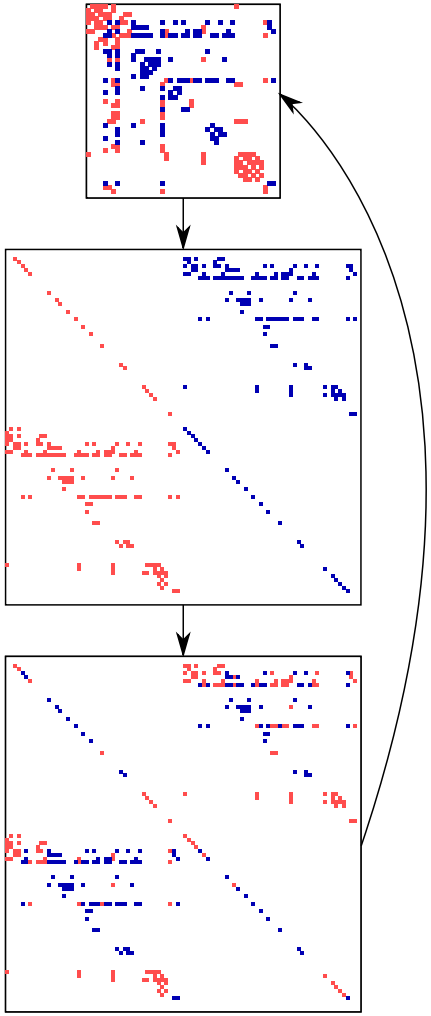
- We need an algorithm to place each nonzero a_{ij} in either A^c or A^r
- Empirically, we found that the following heuristic approach works very well:
  - Place nonzero a_{ij} in A^r if row i contains less nonzeros than column j
  - Place nonzero a_{ij} in A^c if row i contains more nonzeros than column j
  - All ties go to either A^r or A^c, depending on matrix dimensions
- Intuition: small rows/columns have a higher probability of being uncut in the best partitioning

Example of medium-grain on gd97_b

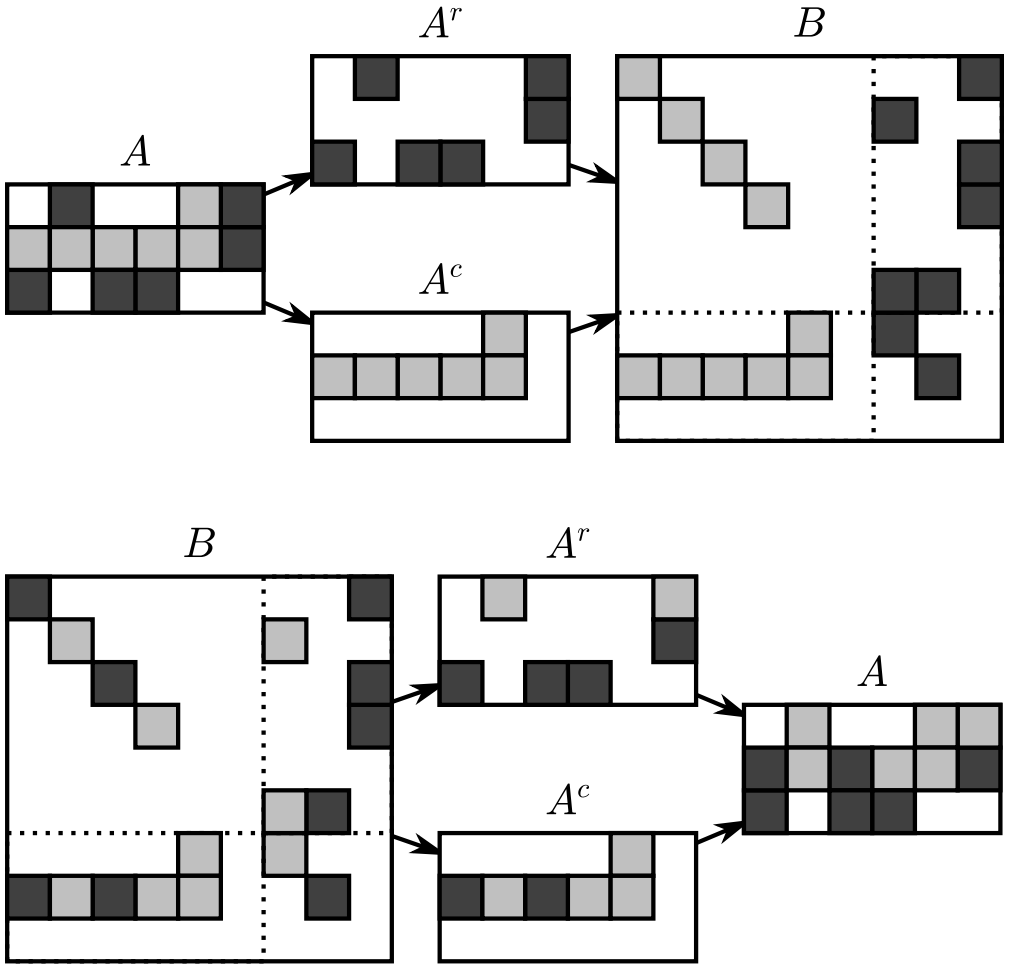


Iterative refinement

- After bipartitioning a matrix with any method, we can create a different initial split:
  - Place nonzeros assigned to part 0 in A^c, and the others in A^r
  - Or vice-versa
- Then, perform a quick KLFM refinement on B (using the row-net model), and iterate
- The result is a fast procedure for improving a given bipartitioning, based on the medium-grain method



Iterative refinement

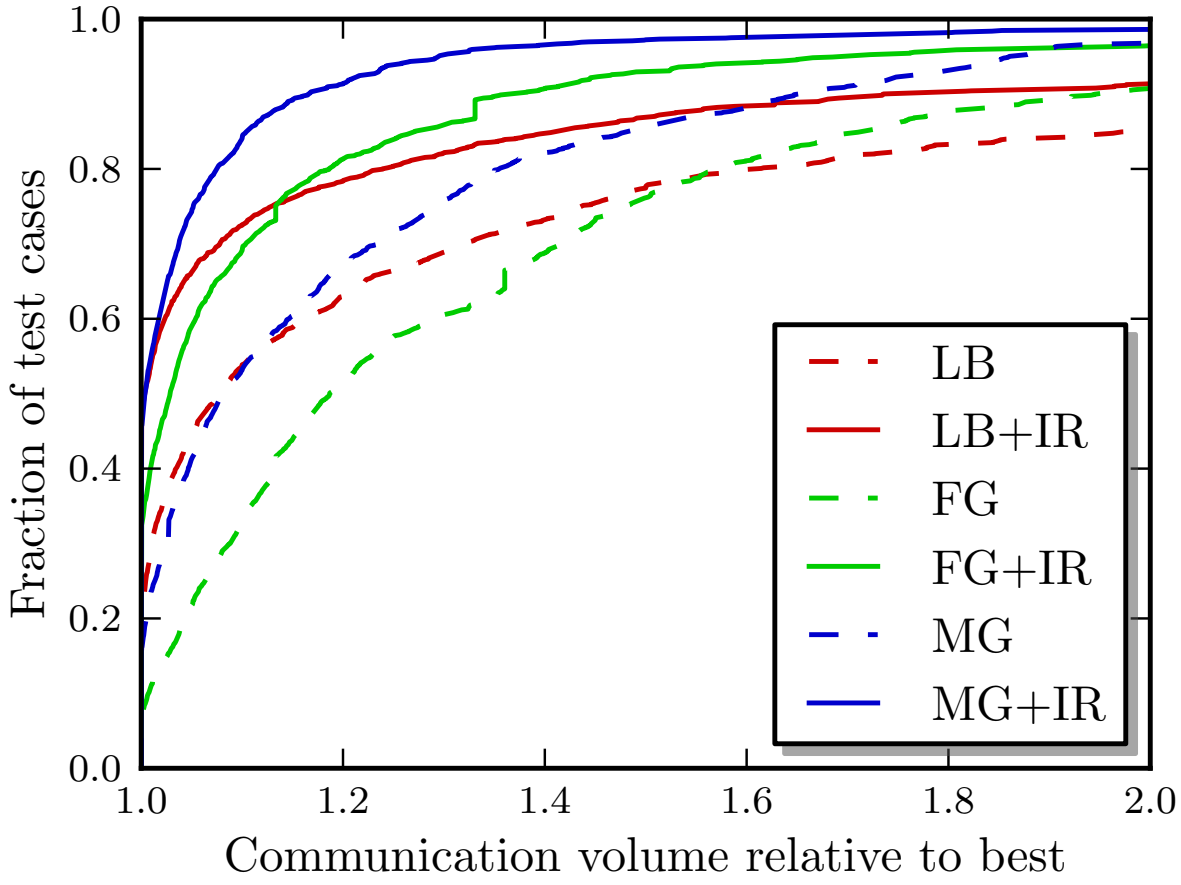


Experiment

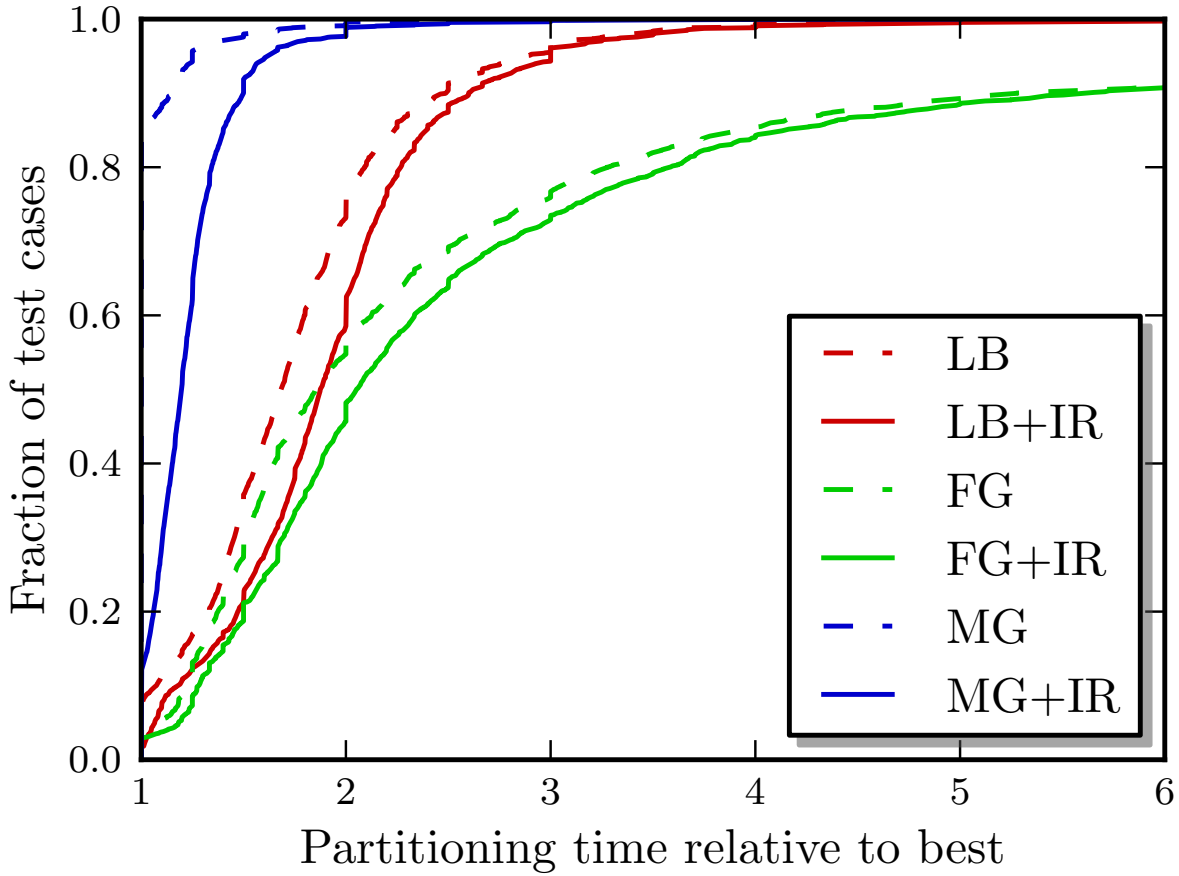
- All matrices of the University of Florida sparse matrix collection[1] with $500 \leq N \leq 5,000,000$ nonzeros:
  - 582 rectangular
  - 1007 structurally symmetric
  - 675 square unsymmetric
- Compare average comvol and partitioning time of 10 runs of:
  - Localbest: try row-net and col-net, take the best
  - Fine-grain
  - Medium-grain
  - All with and without iterative refinement
- Using the `Mondriaan` software package[2]
  - With internal and `PaToH`[3] as hypergraph partitioner

[1] `http://www.cise.ufl.edu/research/sparse/matrices/`
[2] `http://www.staff.science.uu.nl/~bisse101/Mondriaan/`
[3] `http://bmi.osu.edu/~umit/software.html#patoh`

Comvol, all matrices, $p = 2$, Mondriaan



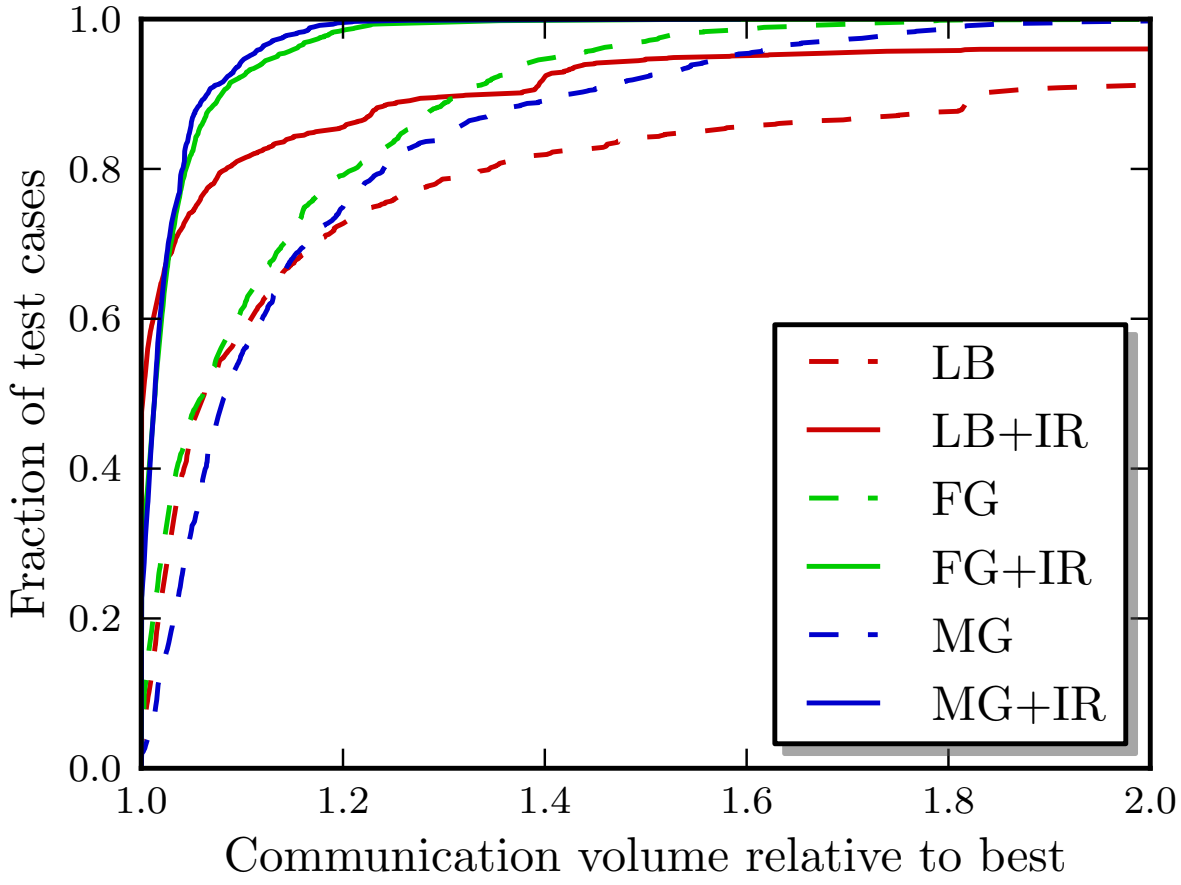
Partitioning time, all matrices, $p = 2$, Mondriaan



Partitioning time and comvol, $p = 2$, Mondriaan

		LB	LB+IR	MG	MG+IR	FG	FG+IR
Comvol	Rec	1.00	**0.94**	1.02	0.96	1.28	1.11
	Sym	1.00	0.75	0.80	**0.67**	0.88	0.69
	Sqr	1.00	0.77	0.68	**0.62**	0.76	0.66
	All	1.00	0.80	0.81	**0.73**	0.93	0.77
Time	Rec	1.00	1.05	**0.53**	0.60	1.08	1.18
	Sym	1.00	1.14	**0.64**	0.79	1.55	1.70
	Sqr	1.00	1.08	**0.66**	0.75	1.23	1.32
	All	1.00	1.10	**0.62**	0.72	1.32	1.43

Comvol, all matrices, $p = 64$, PaToH



Conclusions

- The medium-grain method creates 2D bipartitionings of sparse matrices
- Iterative refinement can be applied after any bipartitioning
- Results show that, compared to popular methods, MG+IR:
  - produces partitionings with the lowest communication volume
  - is the fastest at producing them
- Iterative refinement significantly improves the quality of all methods, at a small increase of partitioning time
- Open source implementation in `version 4.0` of `Mondriaan`[4]
- For more information and/or questions: `D.M.Pelt@cwi.nl`

[4] `http://www.staff.science.uu.nl/~bisse101/Mondriaan/`