Using hypergraph partitioning for iterative linear system solution techniques

Masha Sosonkina[1] Yousef Saad[2] Bora Uçar[3]

[1]Ames Laboratory/DOE

[2]University of Minnesota

[3]Emory University

Parallel Matrix Algorithms and Applications, 2006

Outline

Efficiency of parallel preconditioning

- Ideally, parallel preconditioner exhibits
  - convergence as good as in sequential case;
  - small parallel overhead (efficient parallel implementation).
- It is difficult to combine these two goals.
  Example:
  - Parallel Additive Schwarz: low parallel overheard but deteriorating convergence;
  - An incomplete LU: better convergence but not much parallelism.
- For highly-parallel preconditioners, convergence suffers due to the loss of contributions from interface unknowns.
  - May need to make potential interface contributions small.
  - Enhance the diagonal dominance of the local submatrix.

Parallel overhead in sparse linear system solution

- Graph partitioning is applied to sparse matrix to ensure a *small parallel overhead* and balanced workload.
- Small parallel overhead entails efficient communications
  - in parallel matvec multiplication;
  - in parallel preconditioning, such as incomplete LU with fill-in across the processor boundaries.
- Total communication cost increases with the number of iterations (while communication cost per iteration stays the same).
- For an iterative solver, efficient preconditioning reduces parallel overhead.

Graph partitioning for sparse matrices

"Traditional" partitioning

- Matrix representations as a graph
- Sparse matrix pattern representation using adjacency matrix.
  - Some implementations: Chaco [Hendrickson], Distributed Set Expansion (DSE) [YS], METIS [Karypis & Kumar].
  - Features: balances work per unknown, minimizes edge cut, 1D partitioning.
  - Shortcomings: may not correctly minimize communications [e.g., see Hendrickson & Kolda'00], may result in bad numerical properties of sub-domains [YS & MS 99].
- Bipartite partitioning [Hendrickson & Kolda'99]

Graph partitioning for sparse matrices

Hypergraph partitioning

Hypergraph model [Catalyurek, Aykanat, Pinar'96]

- Hypergraph $G = (V, H)$ is a graph in which V is a set of vertices and H is a set of edges that may connect more than two vertices.
- Hypergraph partitioning is $P = V_1, V_2, \ldots, V_k$ is a K-way vertex partition, $V_i \cap V_j = \emptyset$ and $\bigcap V_i = V$.
- Connectivity λ_j of a hyperedge j is the number of parts in which j has at least one vertex. A hyperedge j is said to be cut if $\lambda_j > 1$.

Graph partitioning for sparse matrices

Hypergraph partitioning

- For a partition P, the minimization cost function C_P may be defined in several ways using the cost c_j of hyperedge j.

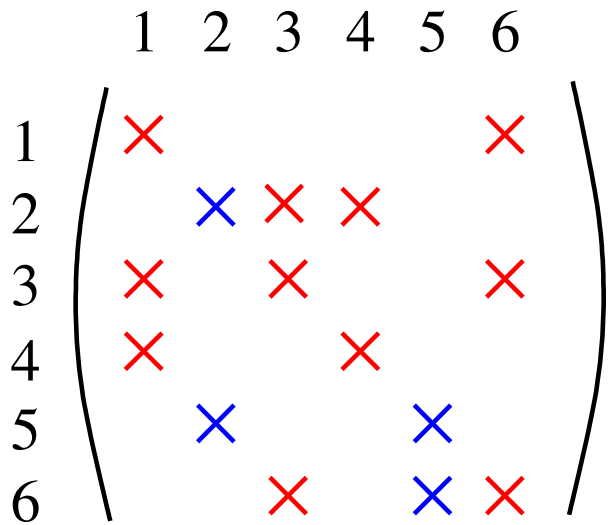
$$C_P = \sum_{j \in H} c_j(\lambda_j - 1) \quad \text{or} \quad C_P = \sum_{j \text{ cut}} c_j, \qquad \text{(c1)} \qquad \text{(c2)}$$

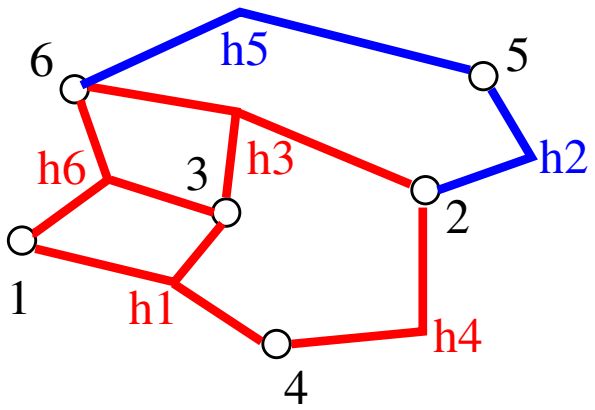
- Features: More expressive than standard graph model; correctly minimizes communications.
- Flexibility of graph representations (e.g., 2D and 1D partitioning).
- Some implementations: Mondriaan, hMETIS, SCOTCH, PaToH, Zoltan.

Example of hypergraph representation of a matrix

Column-net hypergraph model [Catalyurek, Aykanat 99]

A row number is a vertex, a column number is a hyperedge. The hyperedge contains all the vertices corresponding to the row indices of nonzeros in that column.





Incorporating numerical values into hypergraph models

- Goals:
  - Good iterative convergence.
  - Stability in parallel preconditioning operation:
    - An incomplete LU factorization is considered stable if the error of its application, $\|(\boldsymbol{LU})^{-1}\|$, is small.
- Possible solutions should consider matrix numerical properties.
  - Weighted models.
  - Separate partitioning of matrix subgraphs based on some numerical property.

Proposed weighted model

Weight definitions

- Weak Diagonal Dominance ($\mathcal{D}$) as numerical property: Column i is deemed $\mathcal{D}$ if $w_i = \frac{a_{ii}}{\|a_{:i}\|_1}$ and $w'_i = w_i / w_{max} \geq \delta_p$, where $0 < \delta_p \leq 1.0$ and **non**$\mathcal{D}$ otherwise.
- Hypergraph model with weights on hyperedges:
  - $G_{Hw} = (V, H, W_v, W_h)$ is the weighted hypergraph;
  - V is the set of vertices representing rows of the matrix;
  - H is the set of hyperedges representing the columns;
  - W_v is the set of vertex unit weights;
  - W_h is the set of hyperedge weights based on $\mathcal{D}$ and **non**$\mathcal{D}$.

Proposed weighted model

Assigning weights on hyperedges

- Based on column sum of the matrix values (W_h^1).
- Based on the relative value w' of weak diagonal dominance (W_h^2).
- Heavy vs lightweight hyperedges: Heavy hyperedges are more likely to be kept in the same partition, while lightweight are more likely to be split.
  - $\mathcal{D}$ hyperedges as heavy to make local submatrix more $\mathcal{D}$;
  - **non**$\mathcal{D}$ as heavy not to have large entries in the interface matrix.
- Need to consider weighted objective function:
  - In (c1) and (c2), c_j is $W_{h_j}^1$ or c_j is $W_{h_j}^2$.

In our experiments, the cutsize for (c1) was always larger $\longrightarrow$ always had a vertex with more than two neighbors.

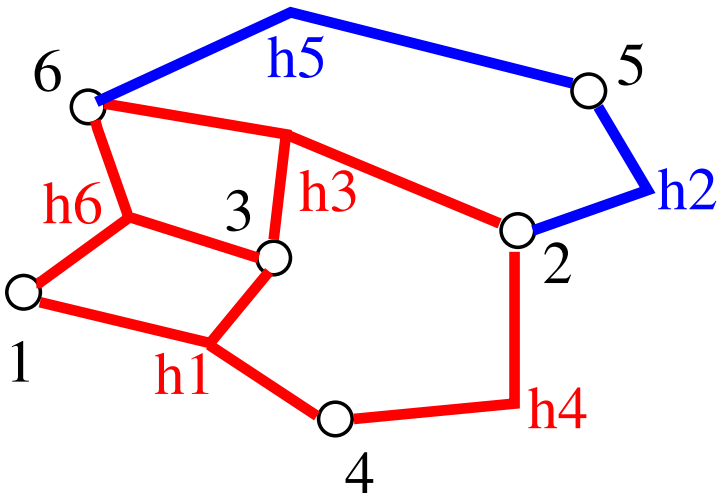
Separate partitioning of matrix subgraphs
Construction

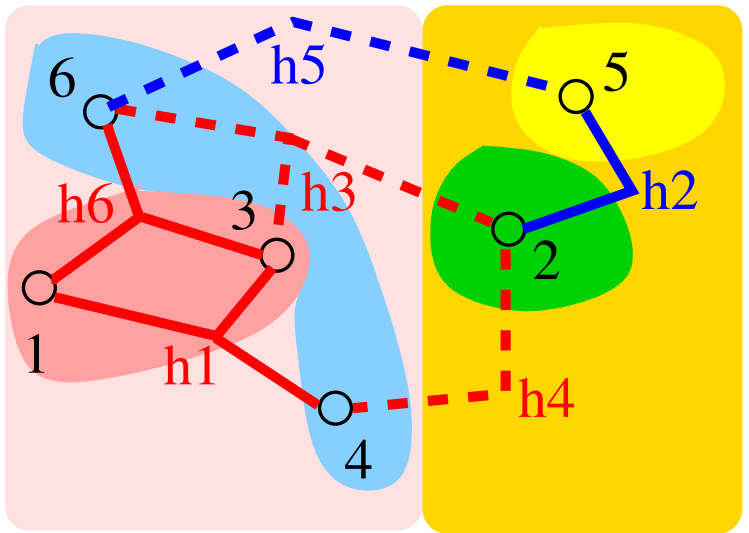
To balance the number of $\mathcal{D}$ and **non**$\mathcal{D}$ edges in each partition.

1. Consider two classes of hyperedges: $\mathcal{D}$ and **non**$\mathcal{D}$;
2. Form two corresponding subgraphs:
   - a $\mathcal{D}$ (**non**$\mathcal{D}$) subgraph contains hyperedge $\boldsymbol{h_i}$ if all the nodes in $\boldsymbol{h_i}$ and the corresponding vertex $\boldsymbol{v_i}$ are $\mathcal{D}$ (**non**$\mathcal{D}$).
   - Otherwise hyperedge $\boldsymbol{h_i}$ is in *residual* subgraph.
3. Partition each subgraph separately into the total given number $\boldsymbol{p}$ of parts.
4. Obtain a $\boldsymbol{2p}$ bipartite graph $\boldsymbol{G_B}$.
5. Use a bipartite matching algorithm to match $\mathcal{D}$ and **non**$\mathcal{D}$ parts based on the connectivity in $\boldsymbol{G_B}$.

Forming and partitioning of two subgraphs

An illustration



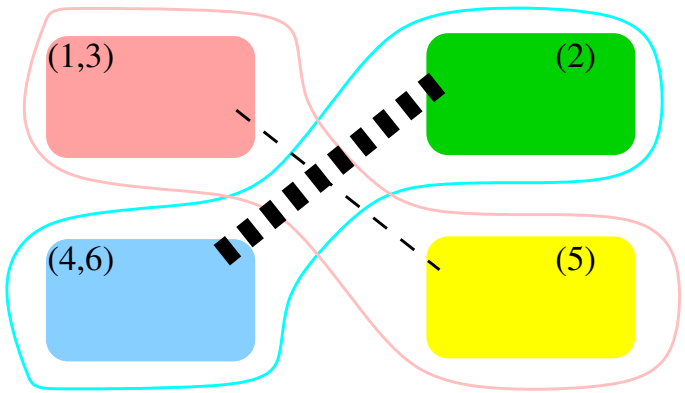


Separate partitioning of matrix subgraphs

Matching subgraphs

To pair $\mathcal{D}$ and **non**$\mathcal{D}$ parts based on their mutual connectivity

- Use dropped hyperedges to establish connections between G_B parts.
- Count the number of connections between any two opposite parts and make it as the weight (cost) for that connection.



- Use existing codes for solving the assignment problem
  - ACM TOMS algorithm 548 (using Hungarian algorithm).
  - Different expressions of the cost measure are possible.

Test problems

Matrix characteristics

Matrix	n	nnz	condest	% $\mathcal{D}$ row; $\mathcal{D}$ col
igbt3	10,938	234,006	4.74e19	26 ; 1
bjtcai	27,628	442,898	6.46e19	43 ; 49
mat9	103,430	2,121,550	3.08e23	35 ; 38
new3	125,329	2,678,750	3.48e22	55 ; 63

- Semiconductor device simulation problems contributed by O. Schenk.
- No convergence using Metis without MC64 scaling.

Experiments
Preconditioners tested

- add_ilut is Additive Schwarz procedure, with local ILUT on sub-domains, without overlap, with and without inner iterations
- lsch_ilut is Schur complement technique, with ILUT as factorization used for local submatrix [YS and MS 97]. Equivalent to Additive Schwarz preconditioner on Schur complement.

The quality of preconditioner is determined by the size of $\|(LU)^{-1}\|$ as estimated in [Chow and YS 98]
$\mathcal{E} = \log\left(\|(LU)^{-1}e\|_1\right)$

- FGMRES(20) with relative $tol_1 = 10^{-7}$ and $tol_2 = 10^{-4}$.
- Scaling by 2-norm; columns after rows.

Experiments: Comparison of hypergraph partitioning

add_ilut without overlap and no inner iteration, $\delta_p = 0.8$.

Matrix	nW		$\mathcal{D}$		non$\mathcal{D}$		SPM	
	$Iter_O$	$\mathcal{E}$	$Iter_O$	$\mathcal{E}$	$Iter_O$	$\mathcal{E}$	$Iter_O$	$\mathcal{E}$
Four processors								
igbt3	151	4.7	157	4.7	153	4.7	180	4.7
bjtcai	339	4.24	413	4.28	328	4.24	192	4.17
mat9	167	3.21	154	3.28	177	3.16	175	3.32
new3	328	3.86	603	3.45	211	3.38	222	3.86
Eight processors								
igbt3	408	4.5	322	4.51	400	4.52	395	4.51
bjtcai	466	4.24	655	4.12	359	4.23	325	4.13
mat9	227	3.13	180	3.25	237	3.16	180	3.29
new3	379	3.41	675	3.33	344	3.84	371	3.84
Twelve processors								
igbt3	196	4.43	200	4.43	241	4.48	379	4.43
bjtcai	507	3.78	579	3.77	502	4.17	700	3.98
mat9	216	3.21	191	3.23	207	3.16	185	2.93
new3	700	2.82	636	3.3	489	3.82	535	3.83

Experiments: Different partitioning tolerance

Matrix	$\mathcal{D}$ add_ilut		non$\mathcal{D}$ add_ilut		$\mathcal{D}$ lsch_ilut	nW add_ilut
	$\delta_p = .5$	$\delta_p = .8$	$\delta_p = .5$	$\delta_p = .8$	$\delta_p = .8$	$\delta_p = 1.0$
Four processors						
igbt3	40	30	28	33	20	29
bjtcai	77	59	37	32	11	40
mat9	54	48	44	73	17	50
new3	80	78	54	60	16	56
Eight processors						
igbt3	65	36	55	44	14	36
bjtcai	72	59	80	75	13	77
mat9	67	70	80	58	17	58
new3	95	93	92	64	18	98
Twelve processors						
igbt3	94	53	58	53	13	53
bjtcai	76	77	76	64	12	70
mat9	73	67	89	81	17	86
new3	99	94	91	92	19	75

- add_ilut without overlap and with five inner iterations.
- lsch_ilut with five inner iterations.
- lower convergence tolerance.

Experiments: Options for subgraph partitioning

Balanced matching in a bipartite graph: Standard deviations of $\mathcal{D}$ rows and of subdomain sizes

Matrix	max_match			min_match			max_match_norm		
	$Iter_O$	loc	$\mathcal{D}$	$Iter_O$	loc	$\mathcal{D}$	$Iter_O$	loc	$\mathcal{D}$
igbt3	193	0.007	0.072	304	0.11	0.102	171	0.008	0.083
bjtcai	110	0.017	0.029	179	0.017	0.058	109	0.017	0.094
mat9	73	0.015	0.032	91	0.015	0.073	73	0.015	0.071
new3	130	0.009	0.004	141	0.006	0.052	130	0.009	0.043

Balancing $\mathcal{D}$ rows: Standard deviations for different weight assignments.

Matrix	nW	$\mathcal{D}$		non$\mathcal{D}$		SPM
		W_h^1	W_h^2	W_h^1	W_h^2	
igbt3	0.14	0.12	0.13	0.1	0.06	0.07
bjtcai	0.28	0.29	0.22	0.25	0.33	0.03
mat9	0.04	0.06	0.07	0.03	0.02	0.03
new3	0.02	0.04	0.12	0.02	0.03	0.04

- 8-way partitioning is used.
- user-defined $\mathcal{D}$ tolerance $\delta_p = \mathbf{0.5}$.

Summary

- We have presented several options to make the hypergraph partitioning preconditioner-aware:
  - by means of weights on hyperedges;
  - by separate partitioning of matrix subgraphs.
- The algorithms may lead to better iterative convergence.

Future work

- Analysis of the communication volume of the proposed methods.