On partitioning and reordering problems in a hierarchically parallel hybrid linear solver

François-Henry Rouet

Lawrence Berkeley National Laboratory

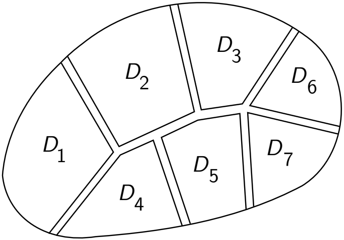
Joint work with: I. Yamazaki (U. T. Knoxville), X. S. Li (LBNL), B. Uçar (ENS Lyon)

IPDPS 2013, PDSEC Workshop, May 24th, 2013

The PDSLin solver (developers I. Yamazaki, X. S. Li)

PDSLin is a hybrid sparse linear solver:

- Schur complement method (non-overlapping domain decomposition).
- Two-level parallelism: intra- and inter-domain parallelism.
- Small number of subdomains (typically 8–64) for stability.
- Explicit approximate Schur complement (dropping).



$$A = \left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \dots & F_k & S \end{array}\right)$$

The PDSLin solver – continued

Package: http://crd-legacy.lbl.gov/FASTMath-LBNL/Software/

- C and MPI, with Fortran interface.
- Unsymmetric/symmetric, real/complex, multiple RHS.

Features

- Parallel graph partitioners:
  - PT-Scotch.
  - ParMETIS.
- Subdomains solvers:
  - SuperLU, SuperLU_MT, SuperLU_DIST.
  - MUMPS.
  - PDSLin.
  - ILU (inexact solution).
- Schur complement solvers:
  - PETSc.
  - SuperLU_DIST.

Two partitioning/reordering problems

We focus on two problems that arise when:

- Permuting the matrix into doubly-bordered form:

$$A = \left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \dots & F_k & S \end{array}\right)$$

- Updating the Schur complement (triangular solution with multiple sparse RHS):

$$\begin{aligned} S \leftarrow & S - \sum_{\ell=1}^{k} F_\ell D_\ell^{-1} E_\ell \\ = & S - \sum_{\ell=1}^{k} \left(U_\ell^{-T} F_\ell\right)^T \left(L_\ell^{-1} E_\ell\right) \end{aligned}$$

Part I

Multi-constraint partitioning

The partitioning problem

- Partitioning: we consider the graph of $A + A^T$; we want a doubly-bordered form.
- Objective: minimize the size of the Schur complement.
- Balance constraints:
  - Subdomain constraints: balance the dimension of D_ℓ and the number of nonzeros in D_ℓ.
  - Interface constraints: balance the dimension of E_ℓ and the number of nonzeros in E_ℓ.

$$\left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \dots & F_k & S \end{array}\right)$$

The partitioning problem

- Assume that we use graph partitioning and that each vertex corresponds to a row.
- Weights need to be assigned to each row for each balance objective, so that the weight of a part (row stripe) is their sum.
- Issue: one cannot know in advance which entries in a row will be in a the diagonal block or the border. The balance objective is a complex function of the partition that cannot be assessed by a looking at a priori weights.
- "Chicken-and-egg problem"[Pınar & Hendrickson '01].

$$\left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \dots & F_k & S \end{array}\right)$$

Partitioning problems with complex objectives

- Conventional methods (e.g., nested dissection) do not take these objectives into account and usually achieve bad imbalance ratios.
- Predictor-corrector approach [Moulitsas & Karypis '04, Pınar & Hendrickson '01]: refine an initial partition provided by standard tools. Improves balance but predictor step is complex.
- Some (somewhat) failed attempts: compute a (cover or edge) separator, transform into wide separator, extract a new separator (vertex cover) that improves balance. Large increase in cut...
- We use a Recursive Hypergraph Bisection with dynamic weights [Kaya, Rouet, Uçar '11].

Hypergraph partitioning

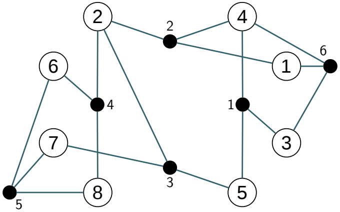
Hypergraph

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a set of vertices $\mathcal{V}$ and a set of hyperedges (nets) $\mathcal{N}$, where a net $h \in \mathcal{N}$ is a subset of vertices.

Hypergraph partitioning (NP-complete)

Partition the vertices into a given number of parts of (almost) same size, so that some cutsize metric is minimized; e.g.

$$\text{con1} = \sum_{n \in \mathcal{N}} c(n)(\lambda(n) - 1) \text{ , or } \text{cnet} = \sum_{n \in \mathcal{N}} c(n) \text{ , or } \text{soed} = \sum_{n \in \mathcal{N}} c(n)\lambda(n)$$



Hypergraph partitioning

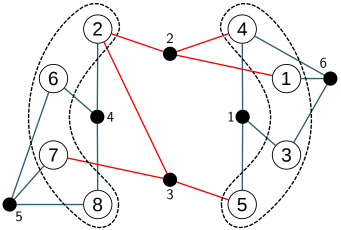
Hypergraph

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a set of vertices $\mathcal{V}$ and a set of hyperedges (nets) $\mathcal{N}$, where a net $h \in \mathcal{N}$ is a subset of vertices.

Hypergraph partitioning (NP-complete)

Partition the vertices into a given number of parts of (almost) same size, so that some cutsize metric is minimized; e.g.

$$\text{con1} = \sum_{n \in \mathcal{N}} c(n)(\lambda(n) - 1) \text{ , or } \text{cnet} = \sum_{n \in \mathcal{N}} c(n) \text{ , or } \text{soed} = \sum_{n \in \mathcal{N}} c(n)\lambda(n)$$



Framework

Recursive bisection paradigm:

1. The first bisection is performed as for the single constraint case.
2. For the subsequent steps: use the partial/coarse information gathered during the previous step to set secondary constraints (complex objectives) and use multi-constraint bisection (we use PaToH [Çatalyürek & Aykanat, '99]): modify vertex-weights.

Algorithm 1 RB

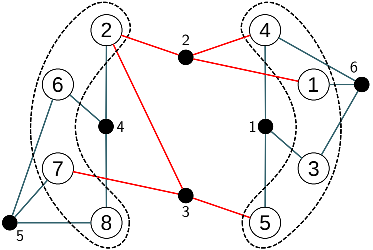
---

if not first bisection step **then**
    Use previous bisection information: set secondary constraints.
end if
Bisect with standard tools.
Discard or split nets according to the objective function and create the two columns sets.
call RB on the first set.
call RB on the second set.

Applying RHB to our problem

Algorithm:

1. Decompose A patternwise as $A = M^T M$ [Çatalyürek, Aykanat, Kayaaslan '09] (M "short and wide" matrix).
2. Permute M into singly-bordered form using RHB and a column-net model:



	4	5	1	6	3	2
6	●	●				
7		●			●	
2	●				●	●
8	●	●				
5			●		●	
1				●		●
3			●	●		
4			●	●		●

Weights:

$w(v_i, 1) = \ |\{j : m_{ij} \neq 0\}|^2 \Rightarrow$ balance on the row stripes of A.

$w(v_i, 2) = \ |\{j : m_{ij} \neq 0$ and column j is not cut yet$\}|^2 \Rightarrow$ balance on the diagonal blocks of A.

Results with PDSLin

We compared NGD with PT-Scotch and our RHB approach:

Matrix	Alg.	Time (s)	Iter.	n_S $\times 10^2$		n_{D_ℓ} $\times 10^3$	nz_{D_ℓ} $\times 10^3$	$nzcol_{E_\ell}$ $\times 10^0$	nz_{E_ℓ} $\times 10^0$
dds.quad	NGD	98.3+5.5	18	95	min	35	1408	980	18792
					max	58	2372	3292	61880
	RHB	90.4+5.3	19	99	min	37	1504	956	17548
					max	58	2162	3614	66416
dds.linear	NGD	108.7+7.5	11	44	min	87	1355	305	1695
					max	114	1792	2593	14622
	RHB	100.7+6.7	10	38	min	87	1346	305	1685
					max	112	1762	2267	12566
matrix211	NGD	89.8+8.9	17	121	min	80	3328	1290	15480
					max	106	8782	5580	133056
	RHB	73.3+9.9	18	130	min	78	6290	1428	17136
					max	173	7223	4380	104256
G3_circuit	NGD	26.3+6.9	11	66	min	192	925	975	1718
					max	205	985	2493	3944
	RHB	22.9+5.3	8	51	min	193	933	899	1749
					max	201	969	1750	3300

Part II

Reordering sparse RHS for triangular solution

Triangular solution with sparse RHS

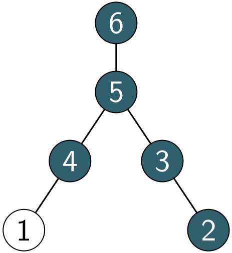
Updating the Schur complement consists of triangular solutions (L_ℓ, U_ℓ) with multiple sparse RHS (F_ℓ, E_ℓ).
We rely on the elimination tree of D_ℓ:

Theorem [Gilbert '86, Gilbert & Liu '93]

The structure of $L^{-1}b$ is the union of paths in the tree for the nodes in $struct(b)$ to the root node.

Example:
Solution of $L\,x = [0\ 1\ 0\ 1\ 0\ 0]^T$

Node 1 is not accessed.



Multiple RHS

Right-hand sides are processed by blocks of size B. Within a block, operations are performed on the union of the different solution vectors. Some padded zeros are introduced.

Ordering/partitioning matters; example with 4 RHS and $B = 2$:

1	2	3	4
X	0	X	0
0	X		
0	X	X	X

1	3	2	4
X	X		
		X	0
0	X	X	X

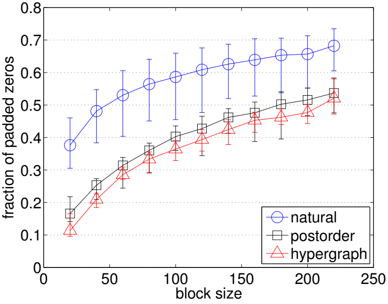
- We have a simple heuristic and a hypergraph model.
- We tackled a similar (but actually quite different) problem in an out-of-core context (cf. [Amestoy et al. '12]).

Two approaches

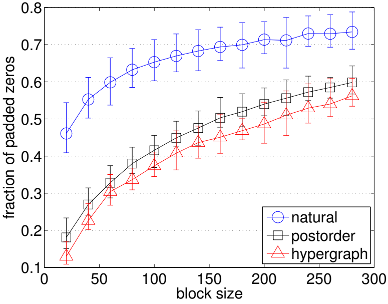
1. Simple heuristic: ordering RHS according to their first nonzero, following the postordering of the elimination tree. This is inexpensive and increases similarities between consecutive columns but only one path is taken into account.
2. Hypergraph model: partitioning the row-net model of the RHS matrix (interface) with the con1 metric minimizes the number of padded zeros (con1 and padded zeros differ by a constant). This hypergraph can be easily sparsified by removing quasi-dense rows.

Results

Padded zeros vs block size B:



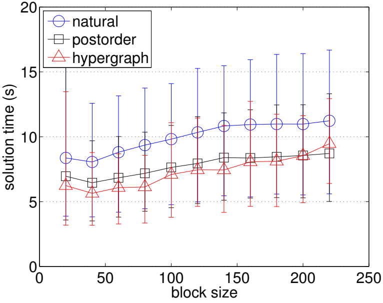
Matrix tdr190k
$N = 1.1$ M, $NZ = 43.3$ M
Accelerator cavity design.



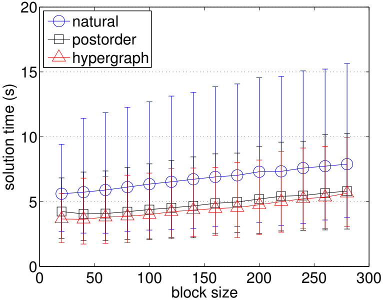
Matrix matrix211
$N = 0.8$ M, $NZ = 55.8$ M
Fusion (M3D-C^1).

Results

Time for updating the Schur complement vs block size B:



Matrix tdr190k
$N = 1.1$ M, $NZ = 43.3$ M
Accelerator cavity design.



Matrix matrix211
$N = 0.8$ M, $NZ = 55.8$ M
Fusion (M3D-C^1).

Conclusion

- Multi-constraint partitioning:
  - Using Recursive Hypergraph Bisection improves load balance, usually at the price of a moderate increase in the size of the Schur complement.
  - Total run time of PDSLin decreases ($\sim 10-50\%$ for our applications of interest, accelerator modeling and fusion).
  - Parallel algorithms?
- Reordering sparse right-hand sides:
  - Using the row-net hypergraph model or the postordering heuristic decreases the amount of padded zeros.
  - Practical gains in PDSLin: Schur complement update time decreased by $\sim 30\%$.