

Combinatorial problems in a Parallel Hybrid Linear Solver

Ichitaro Yamazaki and Xiaoye Li

Lawrence Berkeley National Laboratory

François-Henry Rouet and Bora Uçar

ENSEEIH-IRIT and LIP, ENS-Lyon

SIAM workshop on Combinatorial Scientific Computing
Damstadt, Germany, 05/21/2011

Outline: Combinatorial problems in a parallel hybrid linear solver

1. Introduction to the hybrid solver
2. Combinatorial problems:
 - 2.1 partitioning with multiple constraints
 - 2.2 ordering sparse RHSs for parallel sparse triangular solution
3. Final remarks

Many numerical simulations require the solution of

$$A x = b,$$

where A is a large-scale highly-indefinite general sparse matrix, b is a given right-hand side, and x is the solution to be computed.

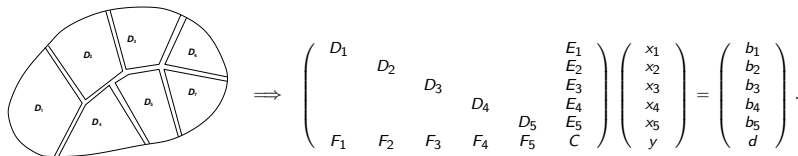
- ▶ **Omega3P** based on discretized Maxwell's equations in frequency domain for modeling accelerator cavities [SLAC, SciDAC ComPASS]
- ▶ **M3D-C¹** based on discretized extended Magnetohydrodynamics (MHD) equations for modeling fusion devices, [PPPL, SciDAC CEMM]

It is difficult to solve the system using standard techniques:

- ▶ **large-scale:** **direct methods** are robust, but scale only to hundreds of processors, and require large memory and communication costs.
- ▶ **highly-indefinite or ill-conditioning:** **preconditioned iterative methods** reduce the costs, but need an effective preconditioner, which is not readily available.

Hybrid solver based on Schur complement method (iterative substructuring):

- a framework to develop a **general-purpose algebraic hybrid linear solver**.
 - potential of combining **robustness** of direct solver with **efficiency** of iterative solver.
- ▶ Use a parallel partitioning tool to extract vertex-separator based on the sparsity structure of the matrix:



- ▶ Solve Schur complement system to compute interface solution:

$$S y = \hat{d}, \text{ where } S = C - \sum_{\ell=1}^5 F_{\ell} D_{\ell}^{-1} E_{\ell}.$$

- ▶ Solve subdomain systems to compute subdomain solution:

$$D_{\ell} x_{\ell} = b_{\ell} - E_{\ell} y.$$

- various options (direct or iterative solver) for solving subdomain/interface problems.

PDSLIn software:

Parallel domain decomposition Schur complement based linear solver

- ▶ solves real or complex general linear system with multiple RHSs.
- ▶ is implemented in C and MPI (+OpenMP) with Fortran interface.
- ▶ requires the following external packages:
 - ▶ parallel graph partitioning:
 - **PT-Scotch** (<http://labri.fr/perso/pelegrin/scotch>), or
 - **ParMetis** (<http://glaros.dtc.umn.edu/gkhome/views/metis>)
 - ▶ subdomain solver:
SuperLU (<http://crd.lbl.gov/~xiaoye/SuperLU>), default
 - **SuperLU_DIST**: two levels of parallelization
 - **SuperLU**: inexact subdomain solvesother options: **MUMPS** or **PDSLIn**
 - ▶ Schur complement solver:
PETSc (<http://mcs.anl.gov/petsc/petsc-as>).

Parallel performance (strong scaling)

Experiment 1:

p	HIPS 1.0	PDSLIn
8	284.6 (26)	79.9 (15)
32	55.4 (64)	25.3 (16)
128	-- (--)	17.1 (16)
512	-- (--)	16.1 (16)

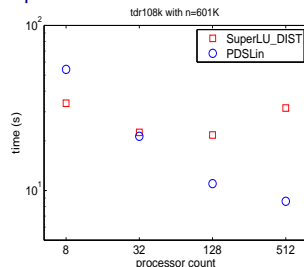
matrix211 (real unsymmetric indefinite) with $n = 801K$

- HIPS: scalable block level-1 ILU precondition.
- PDSLIn: flexible precondition. with num. threshold

Experimental setups:

- ▶ PT-SCOTCH to extract subdomains (parallel ND)
- ▶ SuperLU_DIST to factor each subdomain
- ▶ BiCGStab of PETSc to solve $Sx = b$ with $\frac{\|S\hat{y}-d\|_2}{\|d\|_2} < 10^{-12}$

Experiment 2:



Performance summary:

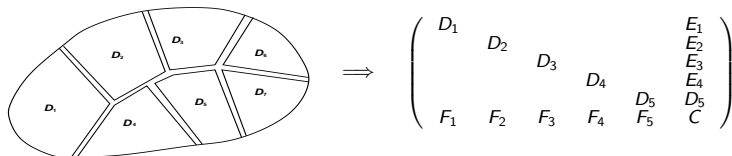
- ▶ flexible precondition for difficult problems
- ▶ improved scaling over direct solver
- ▶ potential to solve large system on thousands of processors (e.g., upto $n = 53M$ and fill-ratio of 250)

Outline 2: Combinatorial problems in a parallel hybrid linear solver

1. partitioning with multiple constraints
 - 1.1 k -way edge partitioning [Metis]
 - 1.2 recursive bisection with hypergraph partitioning [Patoh]
2. ordering sparse RHSs for parallel sparse triangular solution
 - 2.1 postordering based ordering
 - 2.2 hypergraph based ordering [Patoh]

partitioning with multiple constraints:

initial partition to extract vertex-separator impacts load-balance of solver:

Our primary **objectives** are to minimize

- ▶ number of interface vertices
→ separator size
- ▶ number of interface edges
→ nonzeros in interface

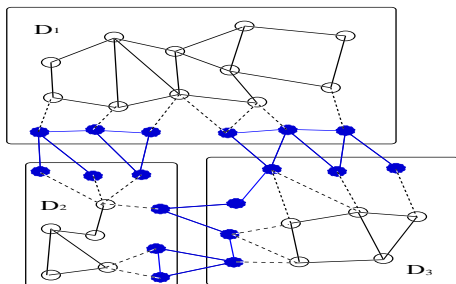
Our primary **constraints** are to balance

- ▶ numbers of interior vertices and edges
→ LU factorization of D_ℓ
- ▶ numbers of interface vertices and edges
→ local update matrix $(F_\ell U_\ell^{-1})(L_\ell^{-1} E_\ell)$

Nested dissection (our current default): obtain small separator, but may lead to imbalance.

Extract vertex-separator from k -way edge partition

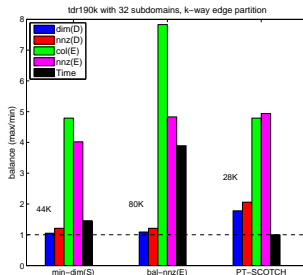
1. Compute k -way edge partition satisfying balanced subdomains D_ℓ (e.g., Metis).
2. Extract vertex-separator from edge-separator to minimize and balance interfaces (i.e., vertex cover).



Heuristics to pick the next vertex:

pick a vertex from **largest subdomain** to maintain the balance among subdomains

- ▶ pick the vertex with **largest degree** to minimize separator size, or
- ▶ pick the vertex to obtain **best balance** of interfaces (e.g., nnzs).

Performance results with k -way edge partitiontdr190k from Omega3P with $n = 1.1\text{M}$ and $k = 32$.

- ▶ improved balance of subdomains, but not of interfaces due to larger separator, where

$$\text{balance}(\text{algo.}, \text{value}) = \frac{\max \text{value}_\ell}{\min \text{value}_\ell}$$

for $\ell = 1, 2, \dots, k$ -th subdomain extracted by algo..

- ▶ challenging to balance multiple constraints by postprocessing already-computed partition.

Recursive bisection with hypergraph partitioning:

1. Compute structural decomposition $A \subseteq M^T M$, where M is an $m \times n$ matrix (e.g., using edge clique cover of $G(A)$ [Catalyurek '09]).
2. Compute 2-way partitioning of M (with multiple constraints) into a singly-bordered block diagonal form based on the recursive bisection of column-net hypergraph (e.g., Patoh):

$$\begin{pmatrix} D_1 & & E_1 \\ & D_2 & E_2 \\ F_1 & F_2 & C \end{pmatrix} = \begin{pmatrix} B_1^T & & \\ & B_2^T & \\ H_1^T & H_2^T & \end{pmatrix} \begin{pmatrix} B_1 & & H_1 \\ & B_2 & H_2 \end{pmatrix}$$

To minimize **objective** and to balance **constraints**:

cut-metrics:

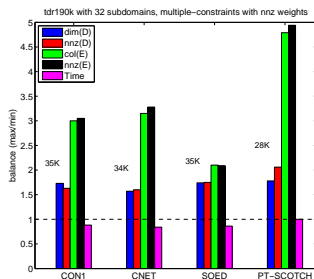
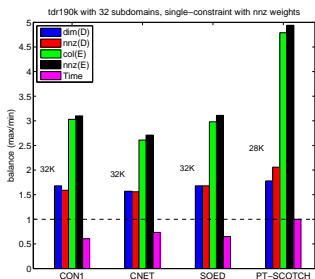
- $\sum_j 1$, (connectivity-1) \rightarrow separator size
- $\sum_j (\lambda_j - 1)$, (cut-net) \rightarrow interface nnz
- $\sum_j \lambda_j$, (soed) \rightarrow sum of above

***i*-th node weights** (based on previous partitions):

- unit weight \rightarrow subdomain sizes
- $\text{nnz}(B_k(i, :))$ \rightarrow subdomain nnzs
- $\text{nnz}(B_k(i, :)) + \text{nnz}(H_k(i, :))$ \rightarrow interface nnzs

where j -th net n_j is in the cut, and λ_j is the number of parts n_j is connected to.

Performance results with hypergraph recursive bisection: tdr190k from Omega3P with $n = 1.1M$ and $k = 32$.

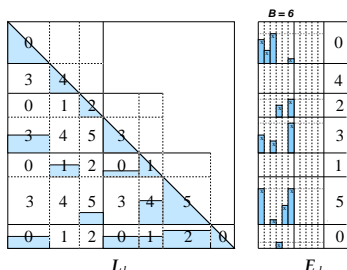


- ▶ single-constraint improved balances from those of PT-SCOTCH without significant increase in the separator size, and reduced the time.
- ▶ multi-constraints further improved balance, but not time due to larger separator.
- ▶ Current studies:
 - seeking for more effective weights
 - developing more efficient (paralle) implementation

parallel sparse triangular solver is a main computational kernel:
computing approx. Schur complement \tilde{S} requires triangular solution with U_ℓ^T and L_ℓ :

$$C - \sum_{\ell=1}^k (F_\ell U_\ell^{-1})(L_\ell^{-1} E_\ell) \xrightarrow{\text{threshold}} \tilde{S},$$

→ take advantage of **sparsity** in F_ℓ and E_ℓ and enforce **sparsity** in $F_\ell U_\ell^{-1}$ and $L_\ell^{-1} E_\ell$ with threshold.



consequences of blocking:

- ▶ improved data locality (+).
- ▶ fewer setups and messages (+).
- ▶ padded zeros (-).

optimizing for **many sparse** RHSs

- ▶ simultaneous solutions with multiple non-empty RHSs.
- ▶ static scheduling of comm. not to send empty messages.
- ▶ operations with nonzero segments of blocks.

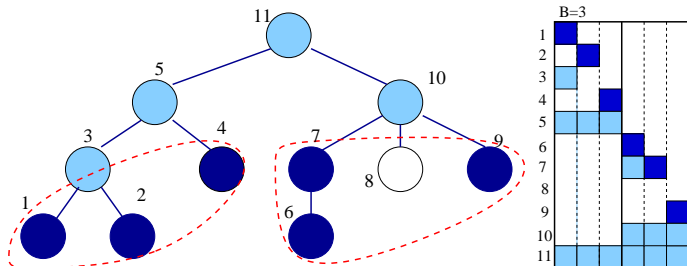
⇒ ×10 speedup.

- tradeoff between + and -

→ **reordering techniques** to reduce # of padded zeros

sparse RHS ordering based on postordering

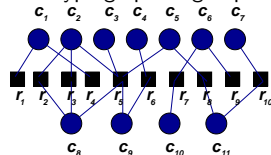
1. apply postordering to the rows of RHSs
2. permute the columns of RHSs in the ascending order of the row indexes of their first nonzeros \rightarrow **more overlap** of the paths to the root = **fewer padded zero**



- ▶ is simple, yet works great in practice (+)
- ▶ considers only the first nonzeros in each columns (-)

sparse RHS ordering based on hypergraph model

- ▶ use column-vertex/row-net hypergraph to group columns together;



where c_j belongs to r_i if corresponding element $G_\ell(i, j)$ is nonzero.

- ▶ define cost of a partition Π as the number of padded zeros in G_ℓ ;

$$\text{cost}(\Pi) = \sum_{\text{for each row } i} \lambda_i B - \text{nnz}(G_\ell(i, :))$$

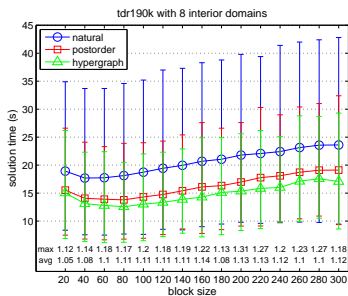
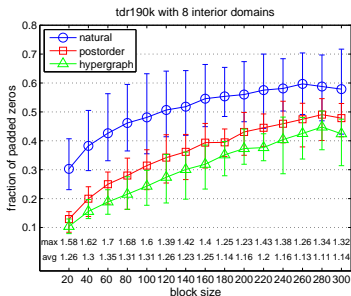
with the number λ_i of groups, to which r_i is connected, and the block size B .

- ▶ **approximate** by assuming $\text{nnz}(G_\ell(i, :)) \approx B$;

$$\text{cost}(\Pi) \approx \sum_{\text{for each net } r_i} B (\lambda_i - 1)$$

this is in the form of the so-called **connectivity-1** cutsizes metric for hypergraph partitioning algorithms (e.g., PaToH).

Performance results: sparse RHSs ordering



- ▶ 20% - 40% reduction in the number of padded zeros and in the solution time from natural ordering (i.e., nested dissection ordering of global matrix).
- ▶ Our default setup uses postordering.

Concluding remarks:

- ▶ Many interesting and important combinatorial problems in a parallel hybrid linear solver.

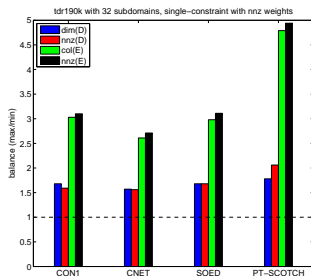
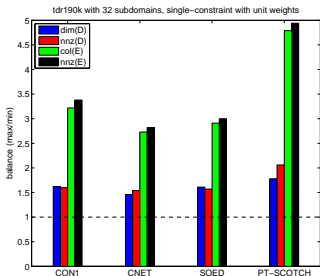
Current research:

- ▶ partitioning and ordering techniques
- ▶ scheduling and MPI+OpenMP to enhance the performance of subdomain solver
- ▶ preconditioning techniques (e.g., FETI, HSS,..) for the Schur complement system
- ▶ integration of the solver into the simulation codes
- ▶ performance comparison of different hybrid solvers

Thank you!!

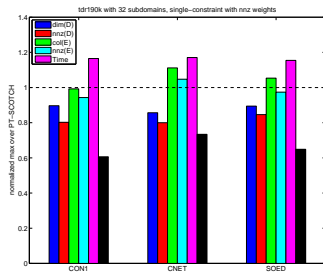
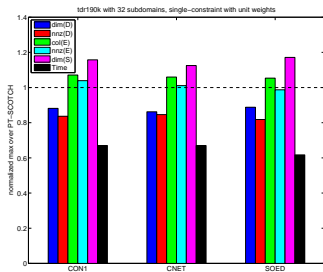
Extra slides

Performance results with single-constraint: balancing partitions



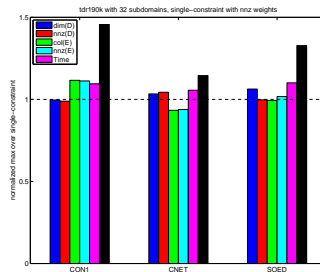
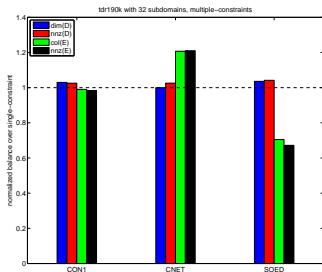
- ▶ balances were improved over PT-SCOTCH.

Performance results with single constraint: normalized maxs over PT-SCOTCH

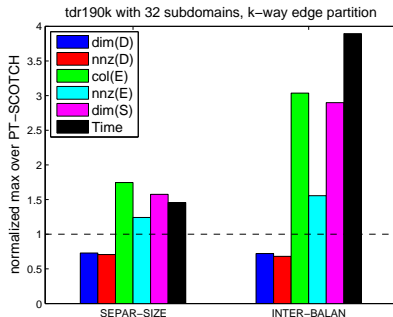
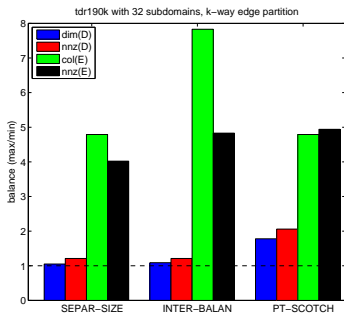


- ▶ timing was improved because

Performance results with multiple constraints:



- ▶ multi-constraints further improve the balance, but not the time due to larger separator.

Performance results: k -way edge partition

- ▶ improved balance of subdomains, but not of interfaces due to large separator.
- ▶ challenging to postprocess partitioning and to obtain multiple constraints.