Auctions for Distributed (and Possibly Parallel) Matchings

E. Jason Riedy
jason@acm.org[1]

EECS Department
University of California, Berkeley

17 December, 2008

[1] Thanks to FBF for funding this CERFACS visit.

Outline

Motivation: Ever Larger $Ax = b$

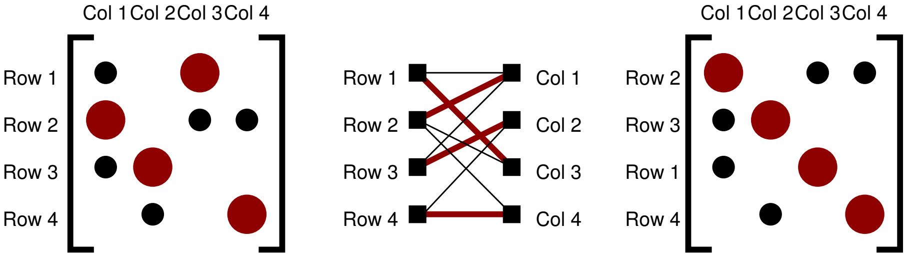
Systems $Ax = b$ are growing larger, more difficult

- Omega3P: $n = 7.5$ million with $\tau = 300$ million entries
- Quantum Mechanics: precondition with blocks of dimension 200-350 thousand
- Large barrier-based optimization problems: Many solves, similar structure, increasing condition number

- Huge systems are generated, solved, and analyzed automatically.
- Large, highly unsymmetric systems need scalable parallel solvers.
- Low-level routines: No expert in the loop!

- Use *static pivoting* to decouple symbolic, numeric phases.
- *Perturb* the factorization and *refine* the solution to recover accuracy.

Sparse Matrix to Bipartite Graph to Pivots



Bipartite model

- Each row and column is a vertex.
- Each *explicit entry* is an edge.
- Want to chose “largest” entries for pivots.
- Maximum weight complete bipartite matching:

  linear assignment problem

Mathematical Form

"Just" a linear optimization problem:

- B — $n \times n$ matrix of *benefits* in $\Re \cup \{-\infty\}$, often $c + \log_2 |A|$
- X — $n \times n$ permutation matrix: the matching
- p_r, π_c — dual variables, will be price and profit
- 1_r, 1_c — unit entry vectors corresponding to rows, cols

Lin. assignment prob.

$$\begin{aligned} \underset{X \in \Re^{n \times n}}{\text{maximize}} \quad & \operatorname{Tr} B^T X \\ \text{subject to} \quad & X 1_c = 1_r, \\ & X^T 1_r = 1_c, \text{ and} \\ & X \geq 0. \end{aligned}$$

Dual problem

$$\begin{aligned} \underset{p_r, \pi_c}{\text{minimize}} \quad & 1_r^T p_r + 1_c^T \pi_c \\ \text{subject to} \quad & p_r 1_c^T + 1_r \pi_c^T \geq B. \end{aligned}$$

Mathematical Form

"Just" a linear optimization problem:

B	$n \times n$ matrix of *benefits* in $\Re \cup \{-\infty\}$, often $c + \log_2 \lvert A \rvert$
X	$n \times n$ permutation matrix: the matching
p_r, π_c	dual variables, will be price and profit
1_r, 1_c	unit entry vectors corresponding to rows, cols

Lin. assignment prob.

$$\begin{aligned} \underset{X \in \Re^{n \times n}}{\text{maximize}} \quad & \operatorname{Tr} B^T X \\ \text{subject to} \quad & X 1_c = 1_r, \\ & X^T 1_r = 1_c, \text{ and} \\ & X \geq 0. \end{aligned}$$

Dual problem

Implicit form:

$$\begin{aligned} \underset{p_r}{\text{minimize}} \quad & 1_r^T p_r \\ & + \sum_{j \in \mathcal{C}} \max_{i \in \mathcal{R}} (B(i, j) \\ & - p_r(j)). \end{aligned}$$

Do We Need a Special Method?

The LAP:

$$\begin{aligned} \underset{X \in \Re^{n \times n}}{\text{maximize}} \quad & \operatorname{Tr} B^T X \\ \text{subject to} \quad & X 1_c = 1_r, \\ & X^T 1_r = 1_c, \text{ and} \\ & X \geq 0. \end{aligned}$$

Standard form:

$$\begin{aligned} \min_{\tilde{x}} \quad & \tilde{c}^T \tilde{x} \\ \text{subject to} \quad & \tilde{A}\tilde{x} = \tilde{b}, \text{ and} \\ & \tilde{x} \geq 0. \end{aligned}$$

$\tilde{A}$: $2n \times \tau$ vertex-edge matrix

- Network optimization kills simplex methods.
  - ("Smoothed analysis" does not apply.)
- Interior point needs to round the solution.
  - (And needs to solve $Ax = b$ for a *much* larger A.)
- Combinatorial methods should be faster.
  - (But unpredictable!)

Properties from Optimization

Complementary slackness

$$X \odot (p_r 1_c^T + 1_r \pi_c^T - B) = 0.$$

- If (i, j) is in the matching $(X(i, j) = 0)$, then $p_r(i) + \pi_c(j) = B(i, j)$.
- Used to chose matching edges and modify dual variables in combinatorial algorithms.

Properties from Optimization

Relaxed problem

Introduce a parameter μ, two interpretations:

- from a barrier function related to $X \geq 0$, or
- from the auction algorithm (later).

Then

$$\operatorname{Tr} B^T X^* \leq 1_r^T p_r + 1_c^T \pi_c \leq \operatorname{Tr} B^T X^* + (n-1)\mu,$$

or the computed dual value (and hence computed primal matching) is within $(n-1)\mu$ of the optimal primal.

- Very useful for finding approximately optimal matchings.

Feasibility bound

Starting from zero prices:

$$p_r(i) \leq (n-1)(\mu + \text{finite range of } B)$$

Algorithms for Solving the LAP

Goal: A parallel algorithm that justifies buying big machines.
Acceptable: A distributed algorithm; matrix is on many nodes.

Choices

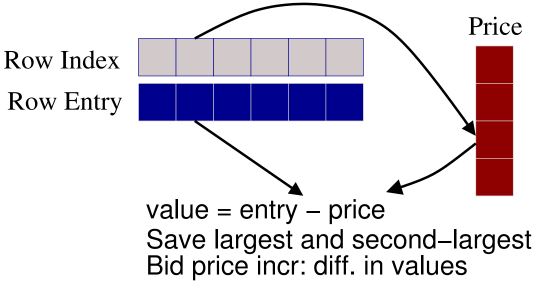
- Simplex or continuous / interior-point
  - Plain simplex blows up, network simplex difficult to parallelize.
  - Rounding for interior point often falls back on matching.
  - (*Optimal* IP algorithm: Goldberg, Plotkin, Shmoys, Tardos. Needs factorization.)
- Augmenting-path based (MC64: Duff and Koster)
  - Based on depth- or breadth-first search.
  - Both are *P*-complete, *inherently* sequential (Greenlaw, Reif).
- Auctions (Bertsekas, *et al.*)
  - Only length-1 alternating paths; global sync for duals.

Auction Algorithms

- Discussion will be column-major.
- General structure:
  1. Each unmatched column finds the "best" row, places a bid.
     - The dual variable p_r holds the prices.
     - The profit π_c is implicit. (No significant FP errors!)
     - Each entry's value: benefit $B(i,j)-$ price $p(i)$.
     - A bid maximally increases the price of the most valuable row.
  2. Bids are reconciled.
     - Highest proposed price wins, forms a match.
     - Loser needs to re-bid.
     - Some versions need tie-breaking; here least column.
  3. Repeat.
     - Eventually everyone will be matched, or
     - some price will be too high.
- Seq. implementation in ~40–50 lines, can compete with MC64
- Some corner cases to handle...

The Bid-Finding Loop

For each unmatched column:

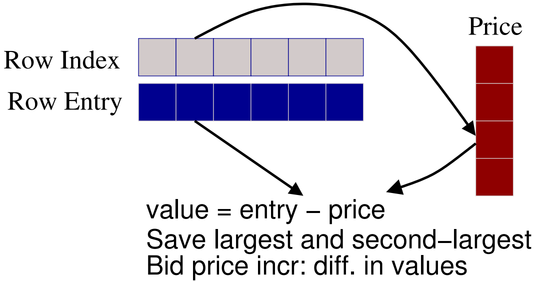


Differences from sparse matrix-vector products

- Not all columns, rows used every iteration.
- Hence output price updates are scattered.
- More local work per entry

The Bid-Finding Loop

For each unmatched column:



Little points

- Increase bid price by μ to avoid loops
  - Needs care in floating-point for small μ.
- Single adjacent row $\rightarrow \infty$ price
  - Affects feasibility test, computing dual

Termination

- Once a row is matched, it stays matched.
  - A new bid may swap it to another column.
  - The matching (primal) increases monotonically.
- Prices only increase.
  - The dual does not change when a row is newly matched.
  - But the dual may decrease when a row is taken.
  - The dual decreases monotonically.
- Subtle part: If the dual doesn't decrease...
  - It's ok. Can show the new edge begins an augmenting path that increases the matching or an alternating path that decreases the dual.

Successive Approximation (μ-scaling)

Complication #1

- Simple auctions aren't *really* competitive with MC64.
- Start with a rough approximation (large μ) and refine.
- Called ϵ-scaling in the literature, but μ-scaling is better.
- Preserve the prices p_r at each step, but clear the matching.
- **Note**: Do **not** clear matches associated with ∞ prices!
- Equivalent to finding diagonal scaling $D_r A D_c$ and matching again on the new B.
- Problem: Performance strongly depends on initial scaling.
- Also depends strongly on hidden parameters.

Performance Varies on the Same Data!

Group	Name	real^T	int^T	real	int
FEMLAB	poisson3Db	0.014	0.014	0.014	0.013
GHS_indef	ncvxqp5	0.475	0.605	0.476	0.608
Hamm	scircuit	0.058	0.018	0.058	0.031
Schenk_IBMSDS	bm_matrix_2	1.446	2.336	1.089	1.367
Schenk_IBMSDS	matrix_9	4.955	6.453	3.091	5.401
Schenk_ISEI	barrier2-4	2.915	5.678	6.363	7.699
Zhao	Zhao2	1.227	2.726	0.686	1.450
Vavasis	av41092	5.417	5.172	4.038	6.220
Hollinger	g7jac200	0.654	2.557	0.848	2.656
Hollinger	g7jac200sc	0.356	1.505	0.371	0.410

On a Core 2, 2.133GHz. Note: MC64 performance is in the same range.

Setting / Lowering Parallel Expectations

Performance scalability?

- Originally proposed (early 1990s) when

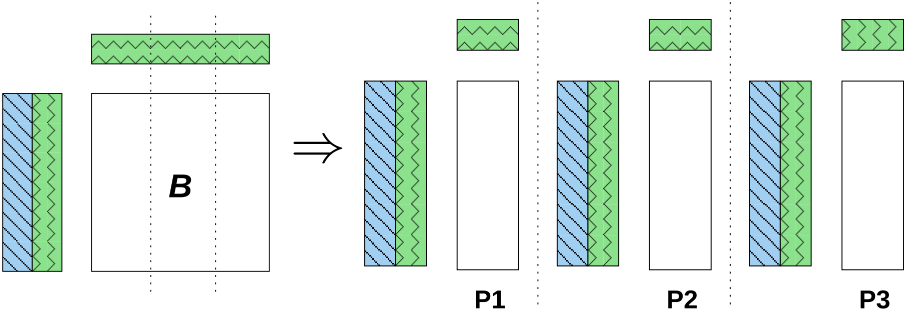
  cpu speed $\approx$ memory speed $\approx$ network speed $\approx$ slow.
- Now:

  cpu speed $\gg$ memory *latency* $>$ network *latency*.
- Latency dominates matching algorithms (auction and others).
- Communication patterns are very irregular.
- Latency (and software overhead) is not improving. . .

Scaled back goal

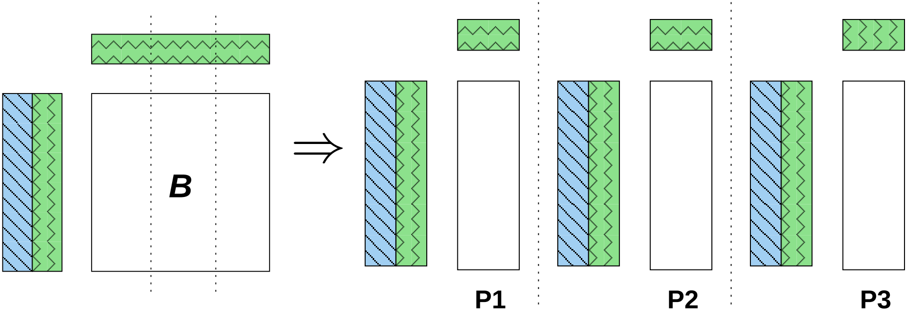
It suffices to not slow down *much* on distributed data.

Basic Idea: Run Local Auctions, Treat as Bids



- Slice the matrix into pieces, run local auctions.
- The winning local bids are the slices' bids.
- Merge... ("And then a miracle occurs...")
- Need to keep some data in sync for termination.

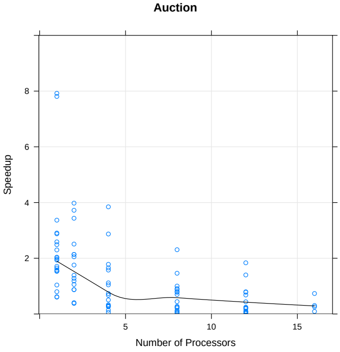
Basic Idea: Run Local Auctions, Treat as Bids

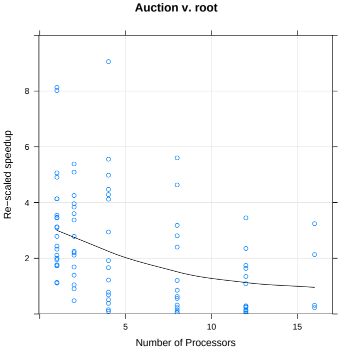


- Can be *memory scalable*: Compact the local pieces.
- Have not experimented with simple SMP version.
  - Sequential performance is limited by the memory system.
- Note: Could be useful for multicore w/local memory.

Ring Around the Auction

- Fits nodes-of-SMP architectures well (Itanium2/Myrinet).
- Needs $O($ largest # of rows $)$ data, may be *memory scalable*.
- Initial, sparse matrices cannot be spread across too many processors... (Below: At least 500 cols per proc.)





On the CITRIS Itanium2 cluster with Myrinet.

Other Communication Schemes

Blind all-to-all

- Send all the bids to everyone.
- A little slower than the ring.
- May need $O(\#\text{ local rows} \cdot p)$ intermediate data; not *memory scalable*.
- Needs $O(p)$ communication; not *processor scalable*

Tree reduction

- Reduction operation: `MPI_MAXLOC` on array of { price, column }.
- **Much** slower than everything else.
- Each node checks result, handles unmatching locally.
- Needs $O(n)$ intermediate data; not *memory scalable*.

Diving in: Lessons from Examples

- Three different matrices, four different perf. profiles.
- All are for the ring style.
- Only up to 16 processors; that's enough to cause problems.
- Performance dependencies:
  - mostly the # of comm. phases, and
  - a little on the total # of entries scanned along the critical path.
- The latter decreases with more processors, as it must.
- But the former is wildly unpredictable.

Diving in: Example That Scales

Matrix Nemeth/nemeth22

- $n = 9\,506$, $nent = 1\,358\,832$
- Entries roughly even across nodes.
- This one is fast and scales.
- But 8 processors is difficult to explain.

# Proc	Time	# Comm	# Ent. Scanned
1	0.2676	0	839 482 003
2	0.0741	55	1 636 226 414
4	0.0307	20	412 509 573
8	0.0176	27	105 229 945
16	0.0153	21	27 770 769

On `jacquard.nersc.gov`, Opteron and Infiniband.

Diving in: Example That Does **Not** Scale

Matrix GHS_indef/ncvxqp5

- $n = 62\,500$, $nent = 424\,966$
- Entries roughly even across nodes.
- This one is superlinear in the **wrong** sense.

# Proc	Time	# Comm	# Ent. Scanned
1	0.910	0	989 373 986
2	1.128	65 370	1 934 162 133
4	2.754	63 228	1 458 840 434
8	15.924	216 178	748 628 941
16	177.282	1 353 734	96 183 742

Diving in: Example That Confuses Everything

Matrix Vavasis/av41092

- $n = 41\,092$, $nent = 16\,839\,024$
- Entries roughly even across nodes.
- Performance $\sim$ # Comm
- What?!?!

Not transposed:

# Proc	Time	# Comm	# Ent. Scanned
1	9.042	0	1 760 564 335
2	5.328	24 248	2 094 140 729
4	6.218	57 553	1 742 989 035
8	15.480	209 393	1 109 156 585
16	68.908	675 635	321 907 160

Diving in: Example That Confuses Everything

Matrix Vavasis/av41092

- $n = 41\,092$, $nent = 16\,839\,024$
- Entries roughly even across nodes.
- Performance $\sim$ # Comm
- What?!?!

Transposed:

# Proc	Time	# Comm	# Ent. Scanned
1	10.044	0	2 010 702 016
2	10.832	887 047	1 776 252 776
4	41.417	1 475 564	1 974 921 328
8	18.929	249 947	844 718 754
16	(forever)		

So What Happened?

- Matrix av41092 has one large *strongly connected component*.
  - (The square blocks in a Dulmage-Mendelsohn decomposition.)
- The SCC spans all the processors.
- Every edge in an SCC is a part of some complete matching.
- Horrible performance from:
  - starting along a non-max-weight matching,
  - making it almost complete,
  - then an edge-by-edge search for nearby matchings,
  - requiring a communication phase almost per edge.
- Conjecture: This type of performance land-mine will affect any 0-1 combinatorial algorithm.

Improvements?

- Rearranging deck chairs: few-to-few communication
  - Build a directory of which nodes share rows: collapsed BB^T.
  - Send only to/from those neighbors.
  - Minor improvement over `MPI_Allgatherv` for a huge effort.
  - Still too fragile to trust perf. results
- Improving communication may not be worth it...
  - The real problem is the number of comm. phases.
  - If diagonal is the matching, everything is overhead.
  - Or if there's a large SCC...
- Another alternative: Multiple algorithms at once.
  - Run Bora Uçar's alg. on one set of nodes, auction on another, transposed auction on another, ...
  - Requires some painful software engineering.

Forward-Reverse Auctions

Improving the algorithm

Forward-reverse auctions alternate directions.

- Start column-major.
- Once there has been some progress, but progress stops, switch to row-major.
- Switch back when stalled after making some progress.

- Much less sensitive to initial scaling.
- Does not need μ-scaling, so *trivial* cases should be faster.
- But this require the transpose.
  - Few-to-few communication very nearly requires the transpose already...
  - Later stages (symbolic factorization) also require some transpose information...

So, Could This Ever Be Parallel?

Doubtful?

- For a given matrix-processor layout, constructing a matrix requiring $O(n)$ communication is pretty easy for combinatorial algorithms.
  - Force almost every local action to be undone at every step.
  - Non-fractional combinatorial algorithms are too restricted.
- Using less-restricted optimization methods is promising, but far slower sequentially.
  - Existing algs (Goldberg, *et al.*) are PRAM with n^3 processors.
  - General purpose methods: Cutting planes, successive SDPs
  - Someone clever *might* find a parallel rounding algorithm.
  - Solving the fractional LAP quickly would become a matter of finding a magic preconditioner...
  - Maybe not a good thing for a direct method?

So, Could This Ever Be Parallel?

Another possibility?

- If we could quickly compute the dual by scaling. . .
- Use the complementary slackness condition to produce a much smaller, unweighted problem.
- Solve that on one node?
- May be a practical alternative.

Questions?

(I'm currently working on an Octave-based, parallel forw/rev auction to see if it may help...)