Sparse Data Structures for Weighted Bipartite Matching

E. Jason Riedy
Dr. James Demmel
(. . . and thanks to the BeBOP group)

SIAM Workshop on Combinatorial Scientific Computing 2004

Use Sparse Matrix Optimizations...

- Take a fixed, simple algorithm: Auction alg. for matchings
  - Repeated iterations over a sparse graph.
- What's expensive, and is there anything we can do about it?
  - Take an idea from optimizing sparse matrix-vector products.
- A little speed-up in some cases, but there are more ideas available...

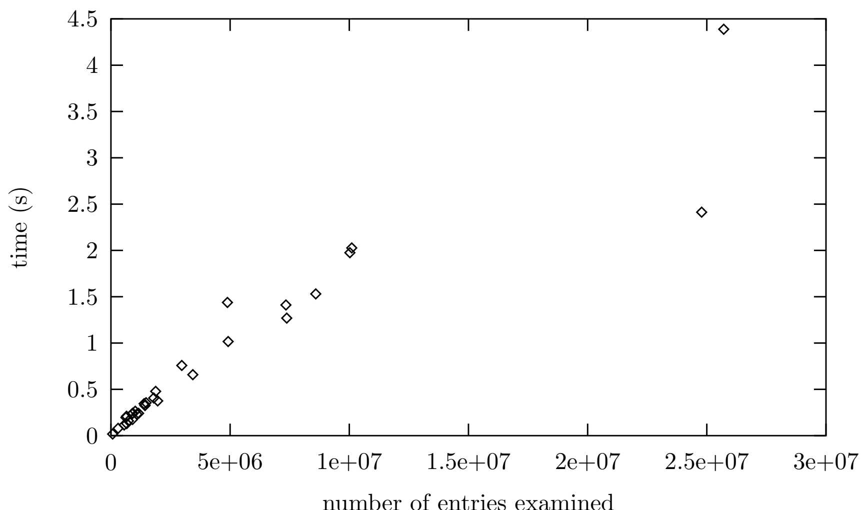
Where's the Time Going?

Auction algorithm: Iterative, greedy algorithm bipartite matching:

1. An unmatched row i finds a "most profitable" column j
   - $\pi(i) = \max_j b(i,j) - p(i)$
2. Row i places a bid for column j.
   - Bid price raised until j is no longer the best choice. (Min. increment μ)
3. Highest bid gets the matching (i,j).

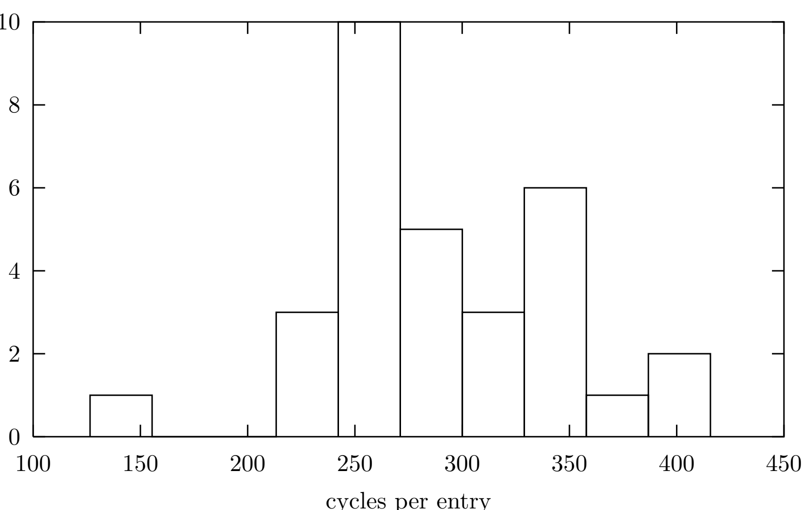
Time linear in entries examined...

Number of entries examined is problem-dependent.



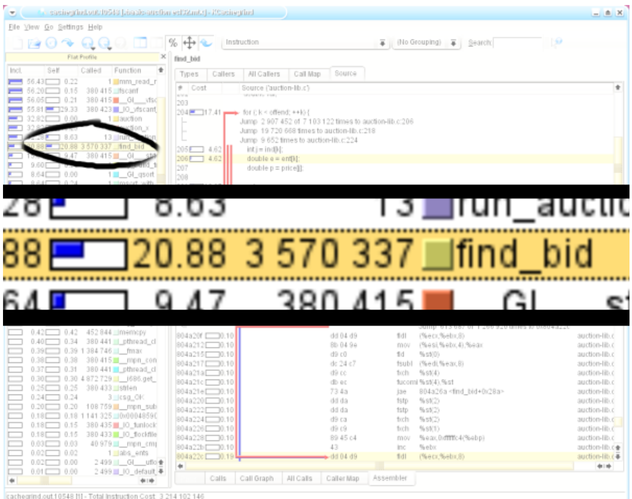
Expensive Inner Loop!

1.3 GHz Itanium 2



Verifying...

Using kcachegrind (N.Nethercote and J.Weidendorfer) and valgrind (J. Seward).



And Locating...

No obvious culprits in the instructions...

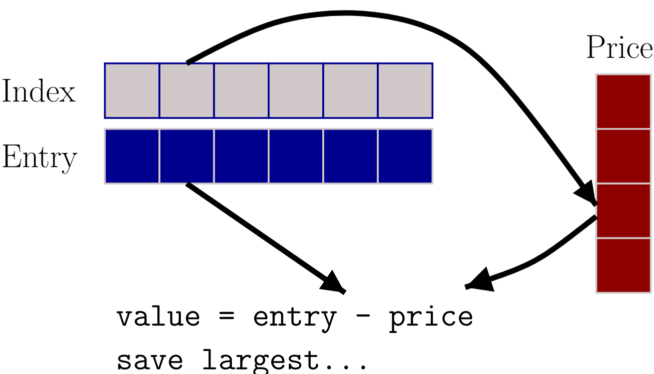
Line	Time	Code
5	4.62	int j = ind[k];
6	4.62	double e = ent[k];
7		double p = price[j];
8		
9	13.87	val = e - p;
0		
1	9.25	if (val <= best_val_1) continue;

And Locating...

But considering cache effects!

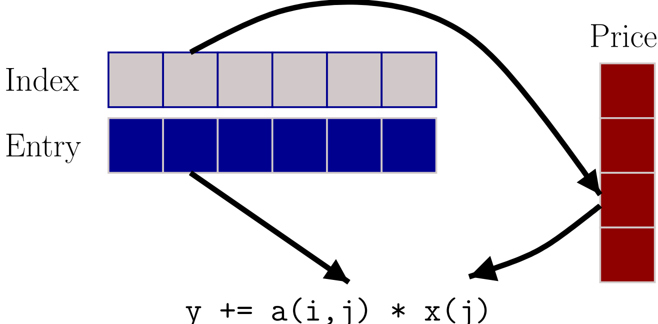
31.81		int j = ind[k];
51.57		double e = ent[k];
		double p = price[j];
16.62		val = e - p;

Auction's Inner Loop

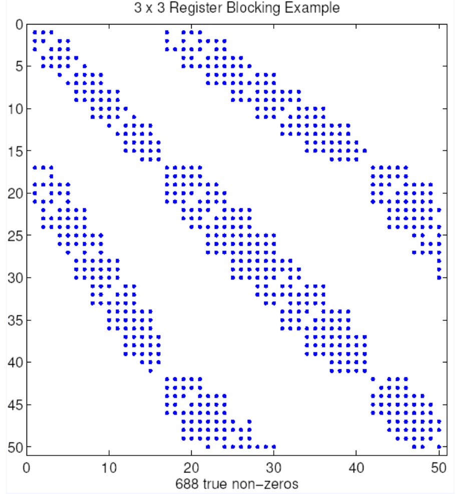


Auction's Inner Loop

Same accesses as sparse matrix-vector multiplication!

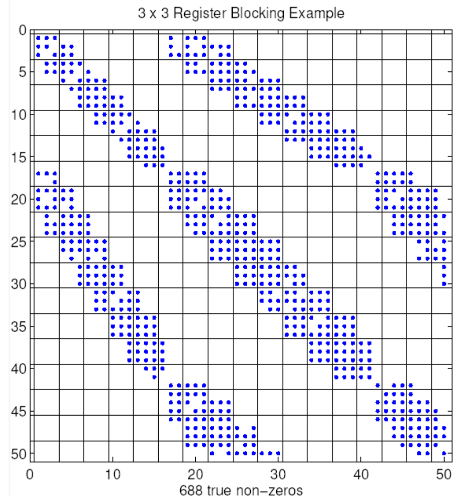


Performance Through Blocking?



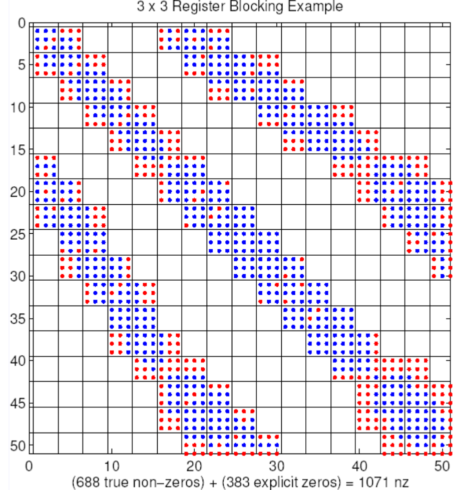
(Images swiped from Berkeley's BeBOP group.)

Performance Through Blocking?



(Images swiped from Berkeley's BeBOP group.)

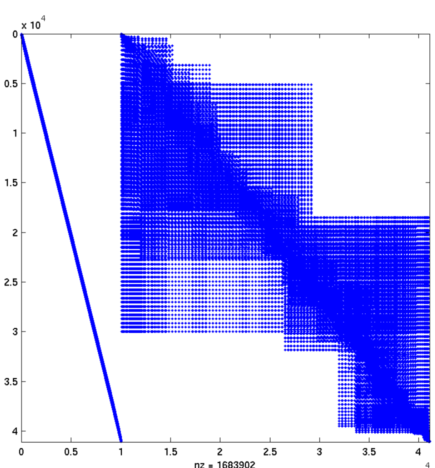
Performance Through Blocking?



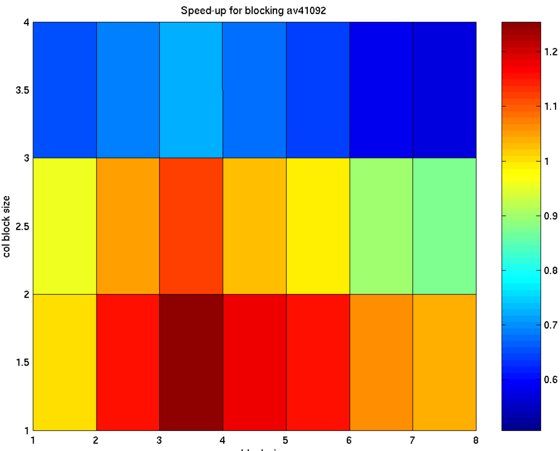
More entries, but 1.5× performance on Pentium 3!
(Images swiped from Berkeley's BeBOP group.)

Blocking Speeds Some Matches

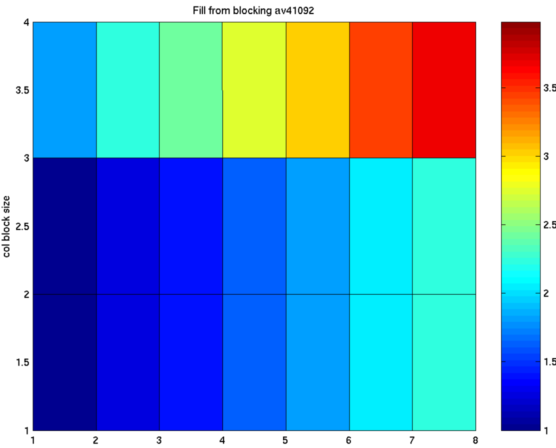
Finite element matrix from Vavasis (in UF collection):



Blocking Speeds Some Matches



Blocking Speeds Some Matches



Observations

A blocked graph data structure may provide additional performance if:

- you iterate over whole rows,
- the graph / matrix has runs of columns, and
- you're willing to use an automated tuning system.

Maximizing the runs: linear arrangement. Hard, but there may be cheap heuristics. Only worth-while if you're performing **many** iterations. (For mat-vec, often > 50 computations of Ax.)