Direct Solution of Sparse Nonsymmetric Linear Systems

Esmond G. Ng
Mathematical Sciences Section
Oak Ridge National Laboratory

September 26, 1995

- Research supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Lockheed Martin Energy Systems, Inc.

Scope

- Sparse nonsymmetric linear systems $Ax = b$.
- LU factorization using Gaussian elimination.
- Need pivoting for numerical stability.
- Fill occurs ...

Sparse Matrix Factorization and Fill

- The LU factors have more nonzero entries than *A*.
- Row and column permutations can affect sparsity of LU factors.
- Role of row/column permutations:
  - maintain numerical stability
  - preserve sparsity in LU factors

Implementations of Sparse Gaussian Elimination

- Pivoting for sparsity is performed either prior to or during numerical factorization.
- Pivoting for numerical stability must be performed during numerical factorization.
- Changes in sparsity structure in *A* and LU factors may be quite dynamic, and can be determined only during numerical factorization.
- Major issues:
  - Pivoting strategies.
  - Choice of data structures.
  - Symbolic manipulations.

Wish List

- Easy to use.
- Small storage requirement.
- Small execution time.
- Good accuracies.
- Known complexity.

Full Pivoting Codes

- Right-looking algorithms.
- Sparsity changes occur in active submatrix.
- Need flexible data structures to allow for dynamic changes in sparsity structure.
- Potentially more overhead in managing data structures.

- Efficient data manipulations may lead to storage fragmentation.
- Unused storage may need to be reclaimed by performing garbage collections.
- Frequency of garbage collections depends on amount of storage allocated when program starts running.
- Potentially high volume of data movement.

Pivoting Strategies in Full Pivoting Codes

- For maintaining numerical stability:
  - partial or complete pivoting
  - but may be too restrictive - does not take sparsity into consideration

- For preserving sparsity:
  - Markowitz scheme - the nonzero in the active submatrix that introduces the least fill is chosen as the pivot
  - a greedy local minimization scheme
  - both column nonzero counts and row nonzero counts have to maintained during factorization
  - can be numerically unstable

Pivoting Strategies in Full Pivoting Codes

- Threshold pivoting - compromise between numerical pivoting and sparsity pivoting (Duff & Reid)
- Relax condition on multipliers - allow multipliers to be larger than 1 in magnitude.
- From the set of candidate pivots, the one that introduces the least fill is chosen as the pivot.
- Result in row and column interchanges.
- Column and row nonzero counts still have to be updated.

- Can be potentially expensive since all nonzeros in active submatrix may have to be examined.
- Limit pivot search to a few sparsest rows (Zlatev, 1980).
- Reduce time in pivot search.
- Limiting search to a few rows has little effect on accuracy of solution.

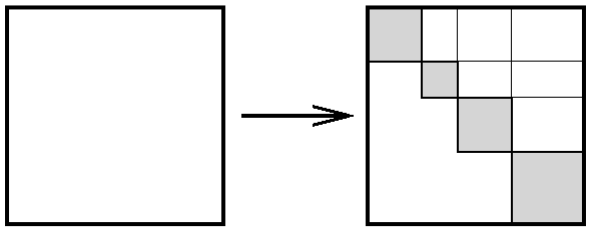
Full Pivoting Codes: Examples

- MA28 (Duff & Reid, 1983)
- MA48 (Duff & Reid, 1993)
- Y12M (Zlatev, Wasniewski, & Schaumburg, 1981)
- LUSOL (Gill, Murray, Saunders, & Wright, 1986; modified 1994)
- UMFPACK (Davis & Duff, 1993)

- Differences:
  - choice of data structures
  - implementation of pivoting strategies
  - order of computations
  - other features

Full Pivoting Codes: Parameters

- Performance controlled by parameters.
  - threshold pivoting parameter to control size of multipliers
  - number of columns to search for pivot
  - switching to dense matrix codes
  - blocking in dense matrix kernels
  - amount of elbow room
  - reducing *A* to block triangular form when it is reducible



Partial Pivoting Codes

- Use partial pivoting to maintain numerical stability.
- Tend to be left-looking.

- New column of LU factors depends only on columns of *L* that have been computed.
- Sparsity structure of new column can be determined and set up prior to computing it.
- Need relatively simple data structures - incremental by columns.

- No parameters to tune in numerical factorization.

- Fill control?

Fill Control in Partial Pivoting Codes

- Apply Gaussian elimination with partial pivoting to *A*.
- Let the LU factors be denoted by *L* and *U*.

- Sparsity structures of *L* and *U* depend on the pivotal sequence.

- Suppose *R'* is the Cholesky factor of the symmetric positive definite matrix *A'A*.

- George & Ng (1987)
  - assume *A* has a zero-free diagonal
  - sparsity structure of *L+U* is contained in the sparsity structure of *R'+R*, irrespective of the pivotal sequence

Fill Control in Partial Pivoting Codes

- Sparsity structure of $L+U$ is contained in sparsity structure of $R'+R$.
- It makes sense to keep R' sparse.
- Sparsity of R' *depends* on sparsity structure of $A'A$ and choice of a symmetric permutation (preordering).
- Good heuristic algorithms for finding good symmetric preorderings from structure of $A'A$.
- Any symmetric preordering of $A'A$ is a column preordering of A: $P'(A'A)P = (AP)'(AP)$.

Partial Pivoting Codes: Examples

- NSPFAC (Sherman, 1978)
- improved NSPFAC (Eisenstat & Liu, 1991)
- Gilbert-Peierls (Gilbert & Peierls, 1987)
- improved Gilbert-Peierls (Eisenstat & Liu ,1991)
- George-Ng, George-Liu-Ng (1987, 1988)

- Differences:
  - data structures
  - the way in which the structure of LU factors are scanned

Partial Pivoting Codes: Facts

- Gilbert & Peierls (1988):
  - sparse Gaussian elimination with partial pivoting can be implemented in time proportional to the number of floating-point operations performed

- George & Ng (1987):
  - if *A* has a zero-free diagonal, then a bound on the size of the LU factors can be computed from the sparsity structure of *A*

Other Codes

- MA32/MA42 (Duff and Reid)
- MA37/MA47 (Duff and Reid)
- sparse (Kundert & Sangiovanni-Vincentelli)
- D2 (Davis & Yew)
- Dispack (Yang)
- IBM ESSL library (Suhl, Aittoniemi, & Su)
- IMSL library (Fan, Horvath, Li, Neaderhouser, & Smith)
- PARASPAR (Zlatev, Wasniewski, Gallivan, & Yang)
- ...

Computing Environment

- IBM RS/6000 model 530 with 32 Mbytes of memory.
- Operating System: AIX 3.2.
- Compiler: AIX XL FORTRAN 2.3, with level 2 optimization (-O2).
- Testing was performed on a very lightly loaded machine.

- Size of executable file was less than 23 Mbytes (including both instruction space and data space).
- Floating-point computations were in double precision.

Test Problems

- Most problems came from the Harwell-Boeing Sparse Matrix Collection.
- Right-hand sides constructed (in extended precision) so that the solution was

  *x(i) = (-1)**(i+1).*

- Results were averages of three runs.

Test Problems

problem	order	$\|A\|$	BTF info			
			no. of diagonal blocks	no. of trivial blocks	size of largest blocks	no. of nonz in nontrivial blocks
JPWH_991	991	6027	146	145	846	5562
LNS_3937	3937	25407	351	321	3558	24118
LNSP3937	3937	25407	351	321	3558	24118
ORANI678	2529	90158	700	699	1830	47823
ORSREG_1	2205	14133	1	0	2205	14133
SAYLR4	3564	22316	1	0	3564	22316
SHERMAN2	1080	23094	211	210	870	19256
SHERMAN3	5005	20033	2111	2109	2830	17924
SHERMAN5	3312	20793	1675	1674	1638	19119
WATT_1	1856	11360	129	128	1728	11104
WATT_2	1856	11550	65	64	1792	11422
SHYY41	4720	20042	1641	1600	1560	13801
HYDR1	5308	22680	1076	1070	2314	19900
RDIST1	4134	94408	199	198	3936	90251
RDIST2	3198	56834	199	198	3000	53668
RDIST3A	2398	61896	99	98	2300	59546

Comparison of LU Codes

- MA48
- UMFPACK (version 1.0)
- Improved Gilbert-Peierls (IGP)

Ease of Use

- Codes are easy to use.

- Matrix input is simple, particularly for MA48 and UMFPACK.
- IGP requires matrix input by columns.

- UMFPACK has the longest calling sequences and the largest collection of parameters.
- Parameters in MA48 are hidden in two arrays in the calling sequence.

- IGP expects users to provide appropriate preordering.

Full Pivoting Codes: Selection of Parameters

- MA48 and UMFPACK.

- Use default values or values suggested in documentations.
- Exception: amount of elbow room.
- Elbow room: A lot.

parameters	MA48	UMFPACK
BTF	yes	yes
multipliers	10	10
no. rows/cols in search	3	4
BLAS level	3	3
block size	32	32
density to switch	0.5	

Use of BLAS in MA48 and UMFPACK

- Both MA48 and UMFPACK use level-3 BLAS.
- Use BLAS provided by IBM (/usr/lib/libblas.a).
- IBM ESSL BLAS is supposed to be much faster than /usr/lib/libblas.a, but do not have access to ESSL.

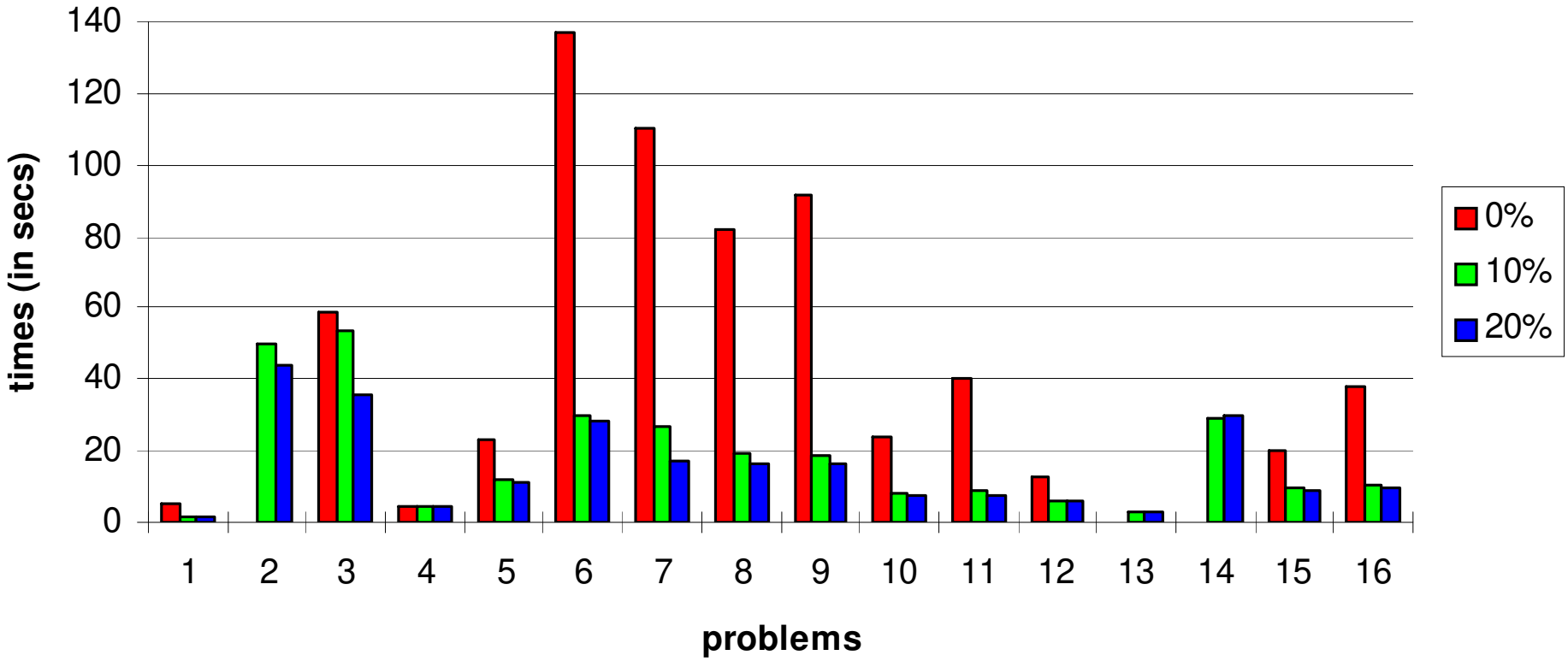
Full Pivoting Codes: Effect of Tuning Parameters

- Block triangular form:
  - good idea in general
- Threshold pivoting parameter - allowing larger multipliers:
  - favor sparsity
  - require less storage
  - decrease in factorization time
  - decrease in accuracy
- Increasing number of columns to search:
  - increase in factorization time
  - effect on storage and accuracy appears to be small

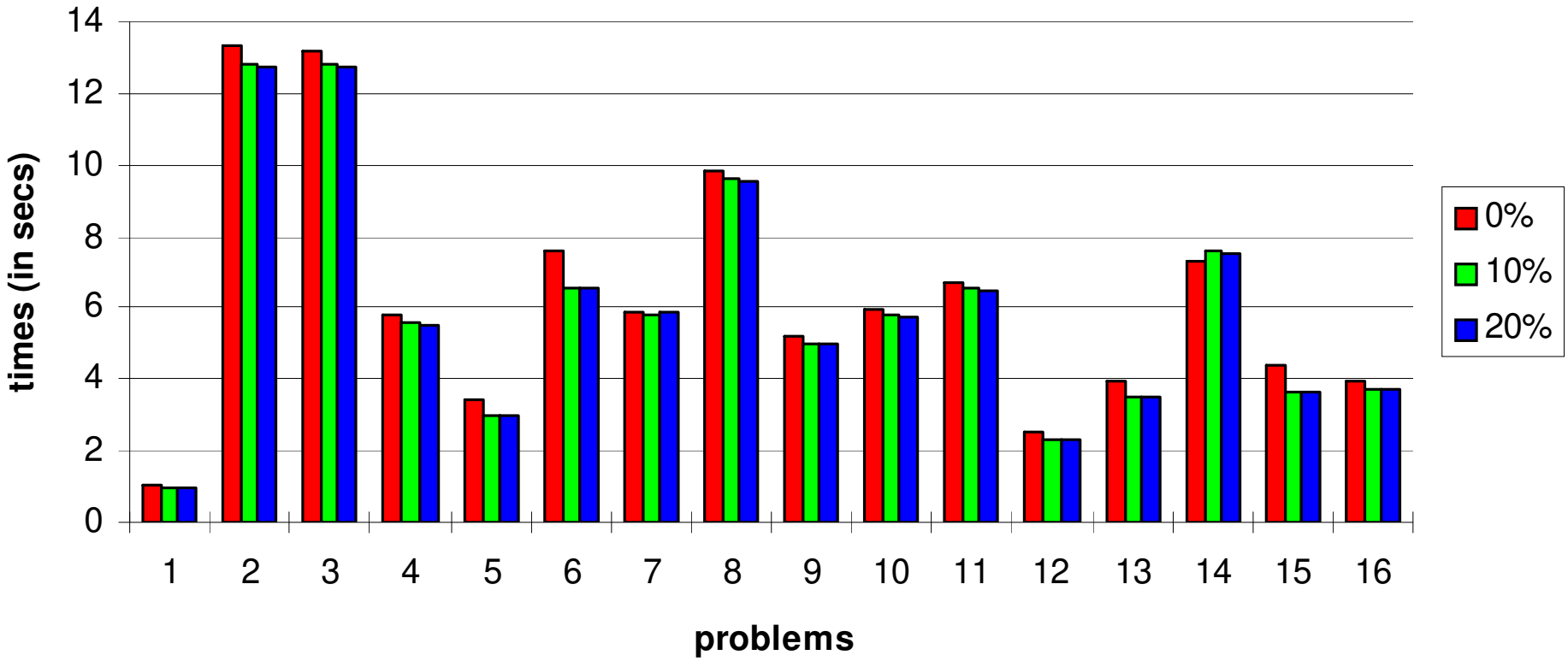
Full Pivoting Codes: Effect of Tuning Parameters

- Increasing density to switch to dense matrix codes:
  - increase in factorization time
  - decrease in storage requirement
- Varying block size in level-3 BLAS:
  - effect on factorization time appears to be very small
- Size of elbow room:
  - can affect performance of MA48 significantly
  - impact on performance of UMFPACK is typically small

MA48: Effect of Elbow Room



UMFPACK: Effect of Elbow Room



Partial Pivoting Codes: Preordering for Sparsity

- IGP requires some appropriate preordering of the columns of *A*.

- A good heuristic is a minimum degree ordering on *A'A*.
- Use multiple minimum degree from SPARSPAK.

- Each row of *A* forms a dense submatrix in *A'A*.
- Can make *A'A* sparse by ignoring dense rows in *A* in preordering.

- When should a row be treated as dense?

Improved Gilbert-Peierls: Removal of Dense Rows

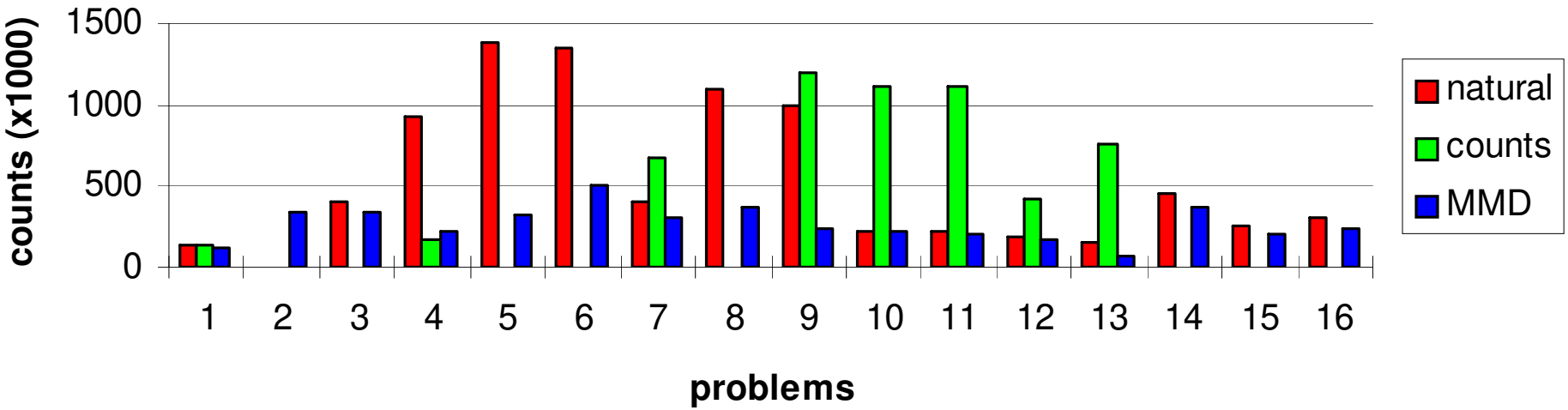
problem	none		50		100		150	
BP_1200	20,909 .142	(0)	13,386 .082	(2)	13,386 .081	(2)	15,591 .101	(1)
MAHINDAS	21,648 .176	(0)	19,018 .138	(6)	21,648 .176	(0)	21,648 .176	(0)
MCFE	61,867 .605	(0)	67,401 .762	(132)	61,867 .604	(0)	61,867 .605	(0)
ORANI678	779,016 6.615	(0)	210,753 2.696	(465)	213,383 2.821	(393)	231,574 3.907	(268)
WATT_2	213,140 3.437	(0)	202,823 3.127	(1)	202,823 3.125	(1)	213,140 3.435	(0)
RDIST1	377,151 3.446	(0)	- -	(98)	377,151 3.443	(0)	377,151 3.455	(0)
RDIST2	208,575 1.568	(0)	- -	(98)	208,575 1.573	(0)	208,575 1.573	(0)
RDIST3A	231,483 2.200	(0)	- -	(48)	231,483 2.195	(0)	231,483 2.198	(0)

Improved Gilbert-Peierls: Preordering for Sparsity

- How about other preorderings?

- Try IGP with 3 different preorderings:
  - natural ordering
  - increasing nonzero counts
  - multiple minimum-degree ordering

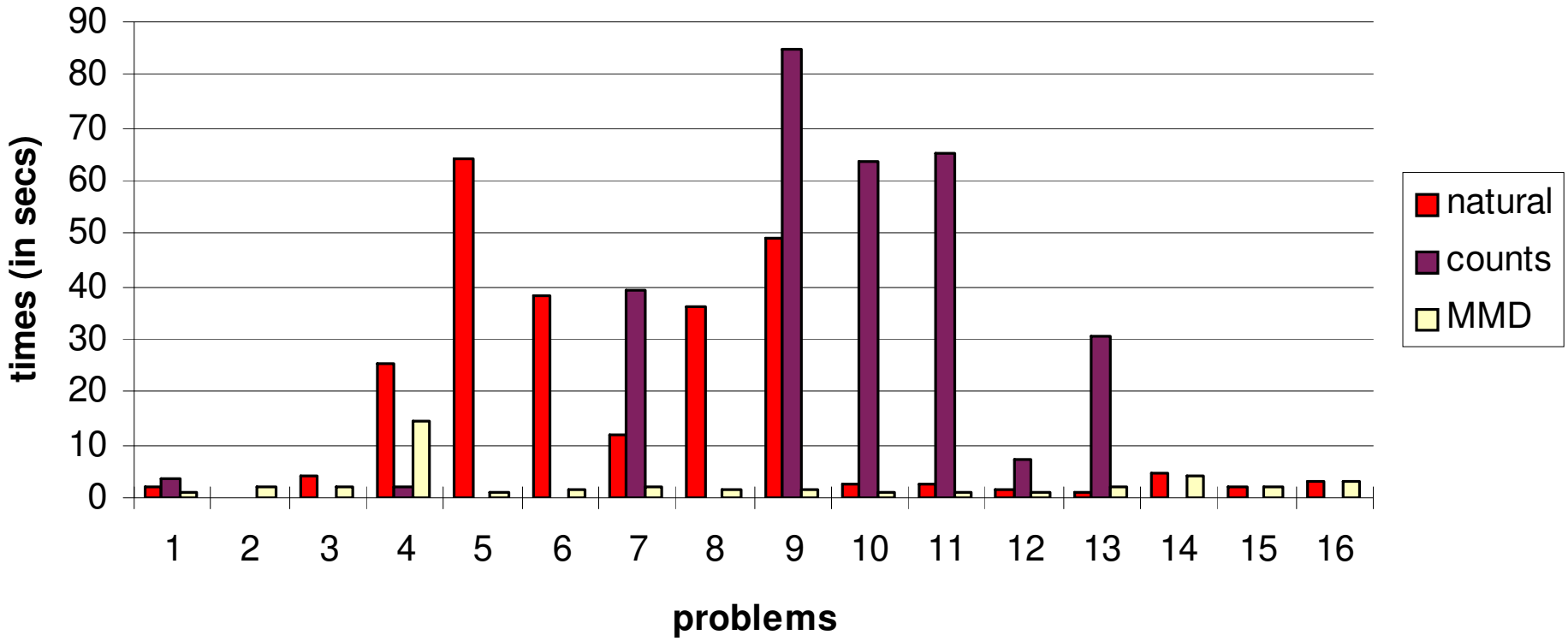
Improved Gilbert-Peierls: Preordering for Sparsity



Improved Gilbert-Peierls: Preordering for Sparsity

nonzero counts in LU factors			
problem	natural	no. of nonzeros	MMD
JPWH_991	136,010	142,852	119,928
LNS_3937	-	-	341,623
LNSP3937	399,335	-	338,267
ORANI678	921,131	160,505	213,383
ORSREG_1	1,383,373	-	314,842
SAYLR4	1,344,966	-	499,030
SHERMAN2	403,551	674,812	302,735
SHERMAN3	1,091,961	-	369,795
SHERMAN5	998,542	1,204,396	232,436
WATT_1	217,902	1,109,018	213,006
WATT_2	225,183	1,11,8783	202,823
SHYY41	193,063	428,754	161,774
EXTR1	45,183	203,489	25,184
RDIST1	450,518	-	377,151
RDIST2	258,163	-	208,575
RDIST3A	303,162	-	231,483

Improved Gilbert-Peierls: Preordering for Sparsity



Improved Gilbert-Peierls: Preordering for Sparsity

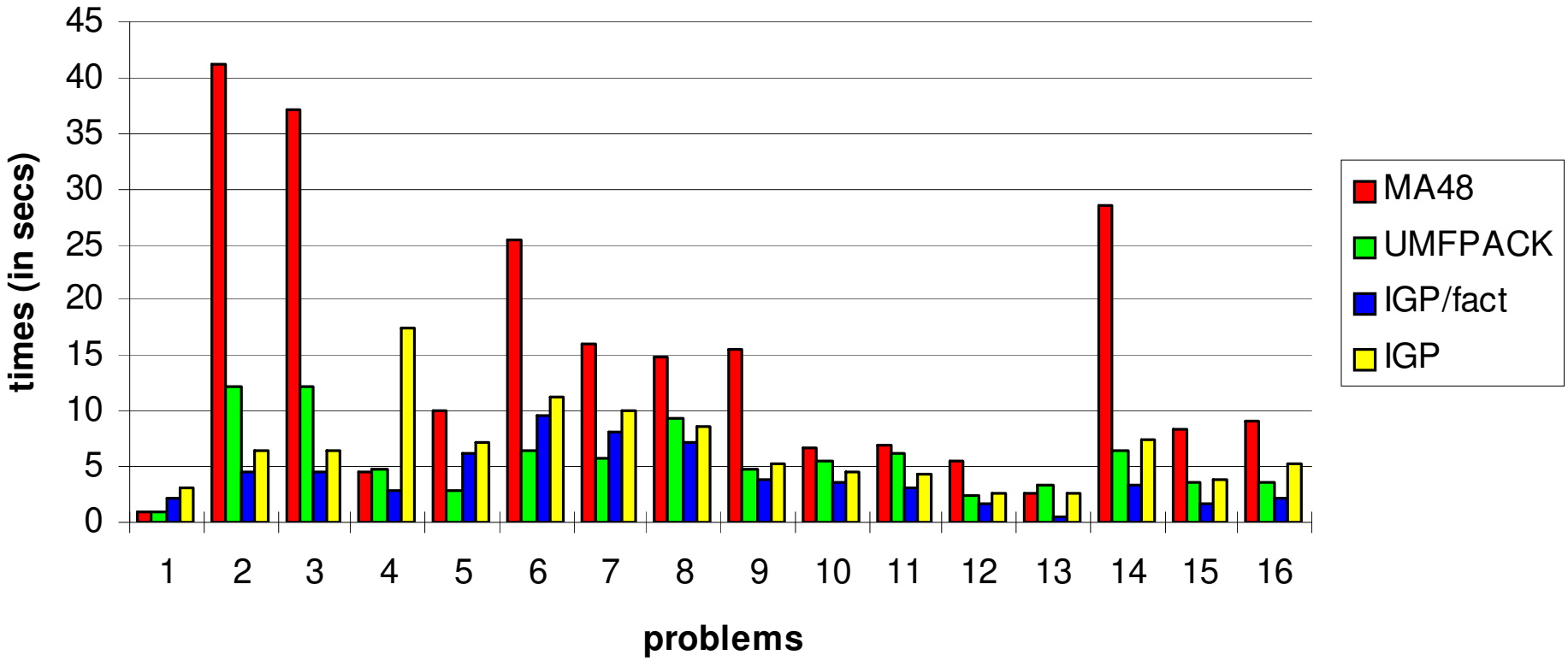
factorization times in seconds				
problem	natural	no. of nonz.	MMD	
JPWH_991	1.974	3.659	2.182	.912
LNS_3937	-	-	4.646	1.914
LNSP3937	4.150	-	4.660	1.922
ORANI678	25.268	1.917	2.821	14.629
ORSREG_1	64.082	-	6.109	1.161
SAYLR4	38.115	-	9.522	1.797
SHERMAN2	11.790	39.306	8.255	1.837
SHERMAN3	36.161	-	7.146	1.487
SHERMAN5	49.189	84.852	3.855	1.345
WATT_1	2.507	63.859	3.568	1.058
WATT_2	2.548	65.199	3.124	1.067
SHYY41	1.608	7.048	1.636	.938
HYDR1	1.021	30.525	.394	2.169
RDIST1	4.450	-	3.443	3.962
RDIST2	2.072	-	1.573	2.218
RDIST3A	3.313	-	2.194	3.076

Comparison of LU Codes

- Full pivoting codes (MA48 and UMFPACK)
  - default parameters

- Partial pivoting code (Improved Gilbert-Peierls)
  - rows with more than 100 nonzeros are treated as dense rows when computing column preordering
  - multiple minimum degree column ordering

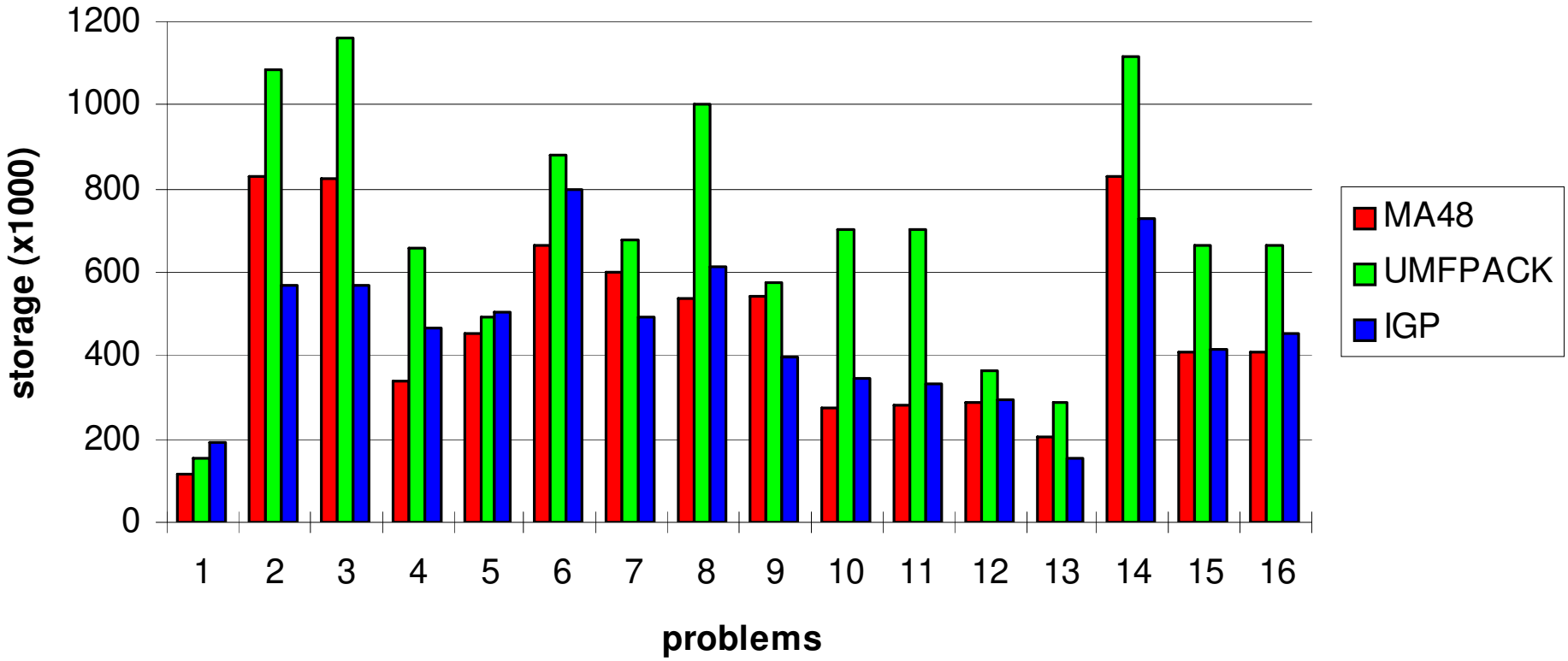
Factorization Times



Factorization Times (in seconds)

problem	MA48		UMFPACK	Improved Gilbert-Peierls	
JPWH_991	.677	.377	.926	.912	2.182
LNS_3937	33.047	8.090	12.268	1.914	4.646
LNSP3937	29.517	7.573	12.292	1.922	4.660
ORANI678	3.854	.655	4.877	14.629	2.821
ORSREG_1	7.241	2.769	2.865	1.161	6.109
SAYLR4	20.289	4.975	6.378	1.797	9.522
SHERMAN2	11.533	4.527	5.713	1.837	8.255
SHERMAN3	11.480	3.420	9.331	1.487	7.146
SHERMAN5	11.741	3.812	4.851	1.345	3.855
WATT_1	5.202	1.457	5.584	1.058	3.568
WATT_2	5.346	1.513	6.310	1.067	3.124
SHYY41	4.224	1.243	2.303	.938	1.636
HYDR1	2.044	.575	3.253	2.169	.394
RDIST1	23.833	4.704	6.570	3.962	3.443
RDIST2	6.859	1.474	3.519	2.218	1.573
RDIST3A	7.525	1.619	3.555	3.076	2.194

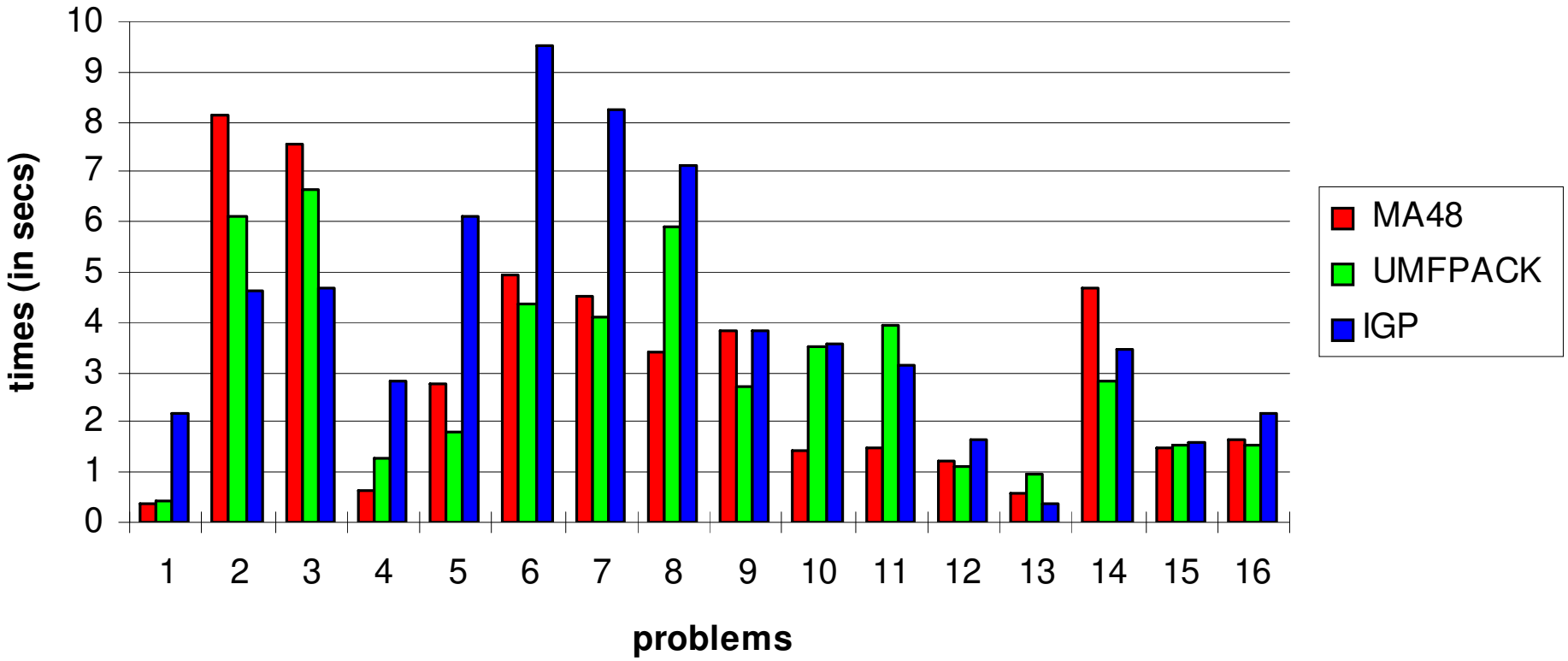
Storage Requirements



Storage Requirements

problem	MA48	UMFPACK	Improved Gilbert-Peierls
JPWH_991	115,039	152,520	193,889
LNS_3937	830,168	1,083,334	570,231
LNSP3937	824,588	1,163,346	565,197
ORANI678	337,068	659,948	467,958
ORSREG_1	451,370	494,323	504,489
SAYLR4	661,397	879,860	799,840
SHERMAN2	597,002	675,122	494,145
SHERMAN3	539,069	999,817	609,768
SHERMAN5	544,243	572,512	396,405
WATT_1	273,923	702,078	345,830
WATT_2	278,335	703,249	330,841
SHYY41	286,197	362,343	296,325
HYDR1	201,968	285,050	155,249
RDIST1	827,640	1,115,441	728,010
RDIST2	411,188	663,382	414,105
RDIST3A	408,638	666,846	452,060

Repeated Factorization Times (Same Pivot Sequence)



Repeated Factorization Times (Same Pivot Sequence)

problem	MA48	UMFPACK	Improved Gilbert-Peierls
JPWH_991	.375	.423	2.182
LNS_3937	8.132	6.122	4.646
LNSP3937	7.551	6.654	4.660
ORANI678	.653	1.256	2.821
ORSREG_1	2.755	1.828	6.109
SAYLR4	4.971	4.385	9.522
SHERMAN2	4.534	4.100	8.255
SHERMAN3	3.399	5.899	7.146
SHERMAN5	3.804	2.711	3.855
WATT_1	1.453	3.504	3.568
WATT_2	1.469	3.910	3.124
SHYY41	1.241	1.096	1.636
HYDR1	.560	.936	.394
RDIST1	4.684	2.811	3.443
RDIST2	1.472	1.549	1.573
RDIST3A	1.628	1.544	2.194

A Note on Accuracies

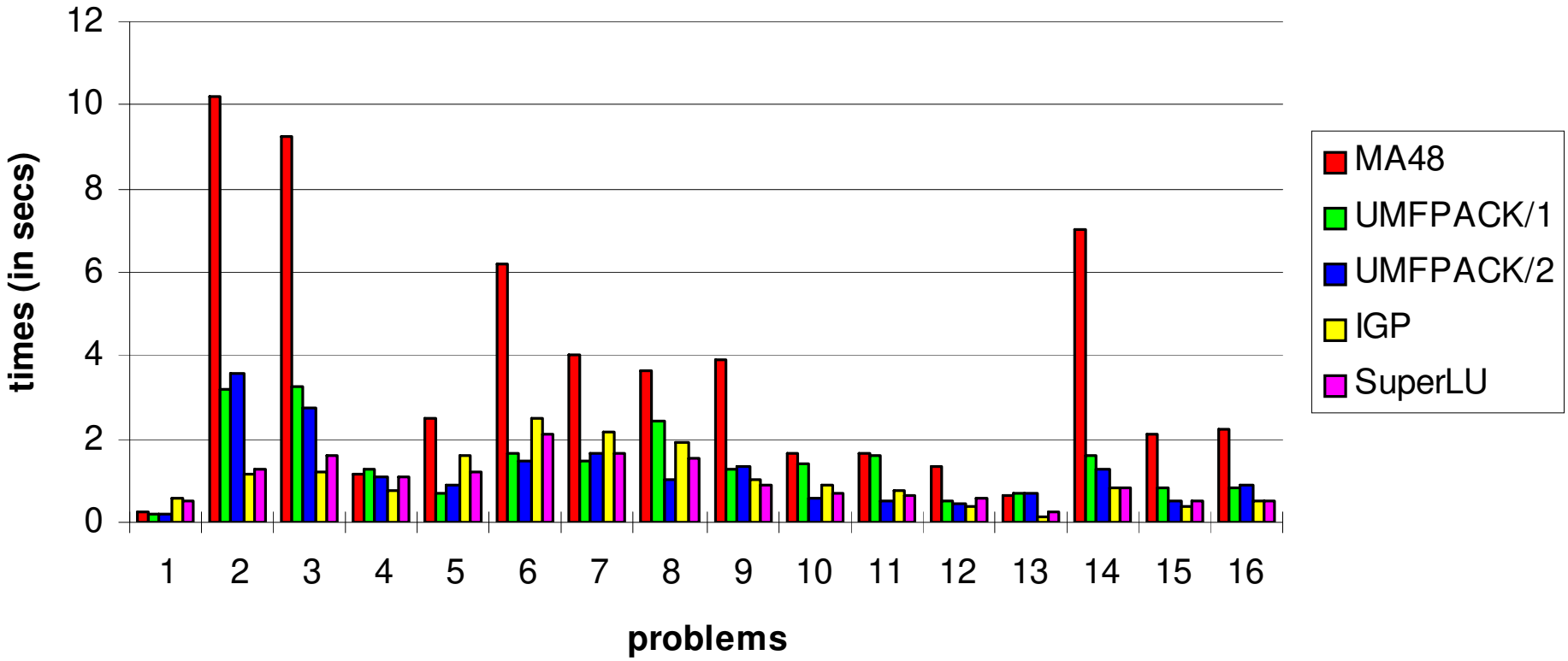
- All codes gave consistent accuracies, but ...
- Doing LU on A' occasionally computed more accurate solutions.

Recent Additions

- Version 2 of UMFPACK
- SuperLU (Demmel, etc.)
  - blocked version of Improved Gilbert-Peierls
  - use BLAS operations

- Preliminary experiments
  - performed on IBM RS/6000 model 590
  - include
    - MA48
    - versions 1 & 2 of UMFPACK
    - Improved Gilbert-Peierls (IGP)
    - SuperLU
  - warning: SuperLU is written in C

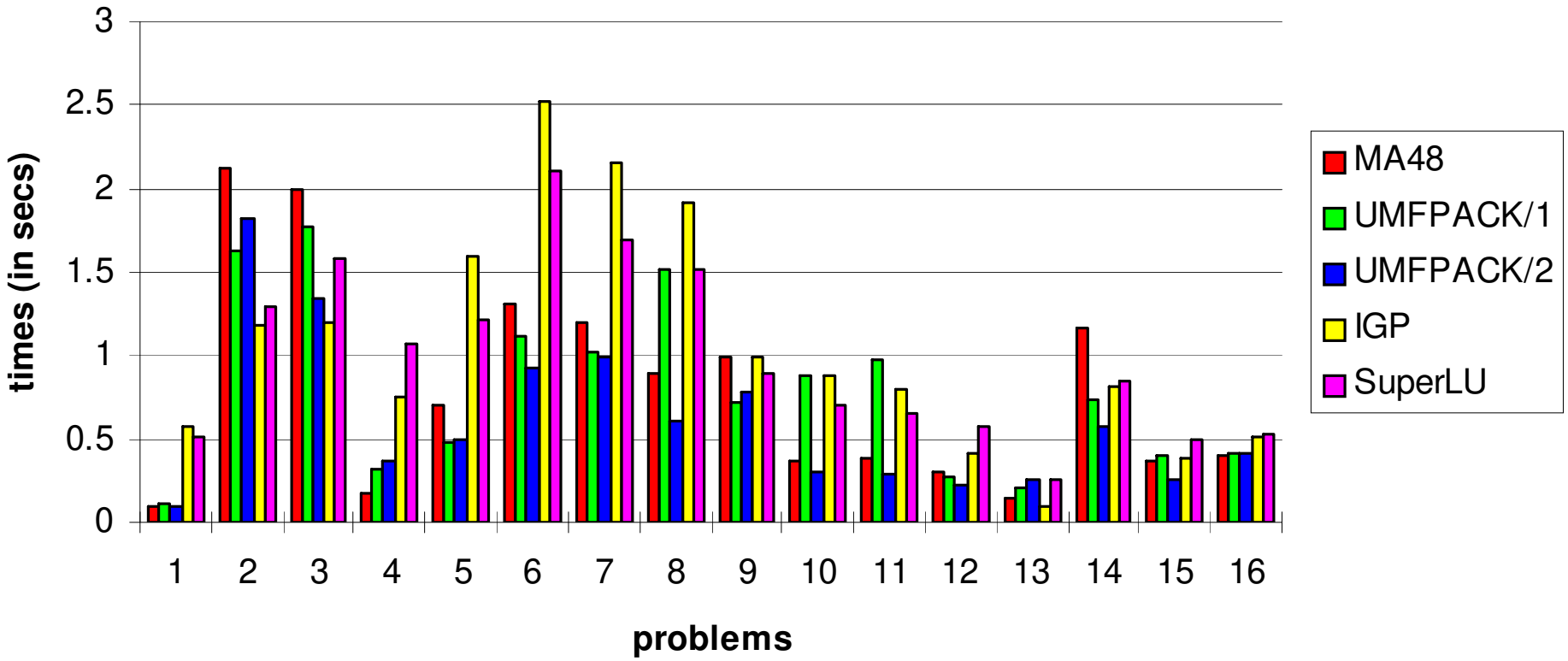
Factorization Times



Factorization Times (in seconds)

problem	MA48	UMFPACK/1	UMFPACK/2	IGP	SuperLU
JPWH_991	.28	.22	.19	.57	.51
LNS_3937	10.23	3.17	3.58	1.18	1.29
LNSP3937	9.23	3.28	2.75	1.20	1.58
ORANI678	1.14	1.26	1.08	.75	1.07
ORSREG_1	2.49	.71	.90	1.60	1.22
SAYLR4	6.20	1.64	1.48	2.52	2.11
SHERMAN2	4.04	1.45	1.66	2.16	1.69
SHERMAN3	3.65	2.41	1.00	1.91	1.51
SHERMAN5	3.90	1.25	1.37	.99	.90
WATT_1	1.64	1.41	.57	.88	.70
WATT_2	1.68	1.62	.54	.79	.66
SHYY41	1.37	.53	.45	.41	.57
HYDR1	.67	.72	.68	.10	.26
RDIST1	7.00	1.61	1.26	.81	.85
RDIST2	2.09	.86	.54	.38	.49
RDIST3A	2.22	.86	.92	.51	.53

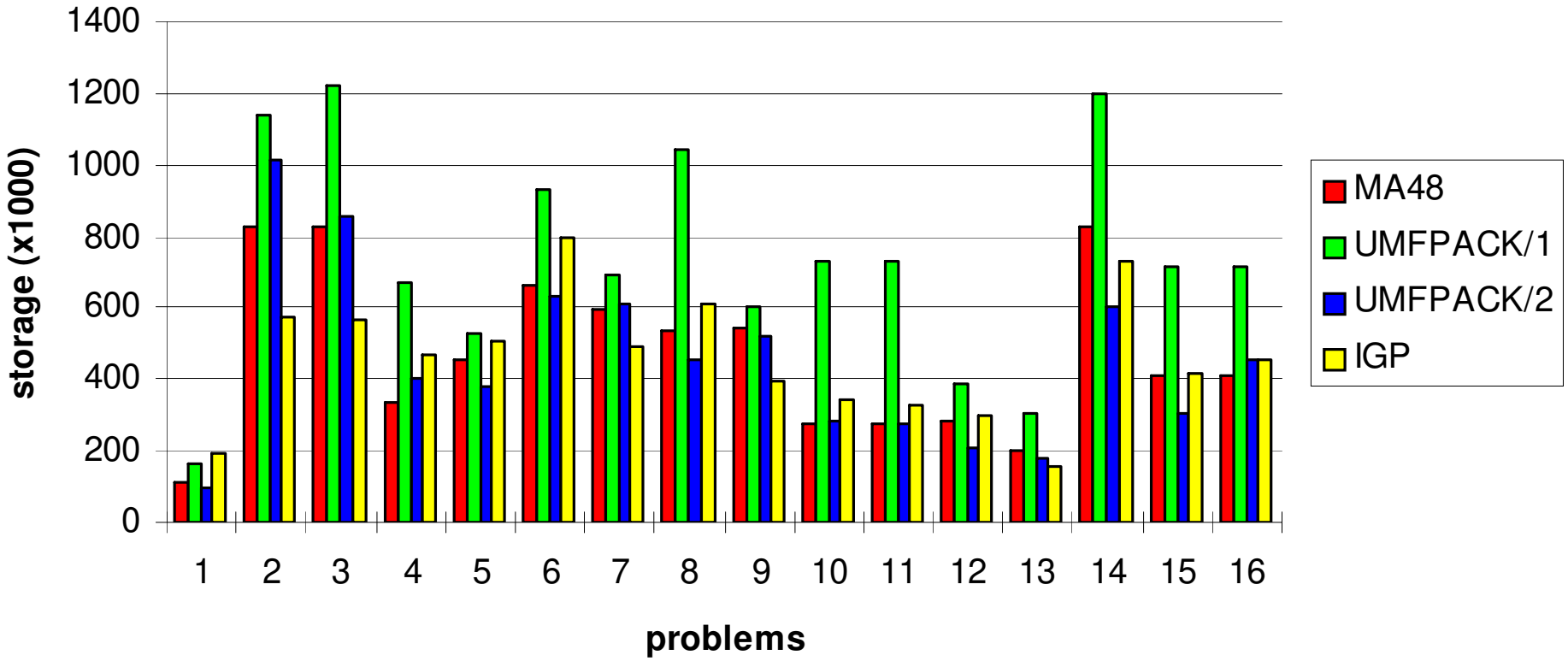
Repeated Factorization Times (Same Pivot Sequence)



Repeated Factorization Times (Same Pivot Sequence)

problem	MA48	UMFPACK/1	UMFPACK/2	IGP	SuperLU
JPWH_991	.10	.11	.09	.57	.51
LNS_3937	2.12	1.63	1.82	1.18	1.29
LNSP3937	1.99	1.77	1.34	1.20	1.58
ORANI678	.18	.32	.37	.75	1.07
ORSREG_1	.71	.48	.50	1.60	1.22
SAYLR4	1.31	1.11	.93	2.52	2.11
SHERMAN2	1.19	1.02	.99	2.16	1.69
SHERMAN3	.89	1.51	.60	1.91	1.51
SHERMAN5	.99	.72	.78	.99	.90
WATT_1	.37	.87	.31	.88	.70
WATT_2	.38	.97	.28	.79	.66
SHYY41	.31	.27	.23	.41	.57
HYDR1	.15	.21	.25	.10	.26
RDIST1	1.16	.74	.57	.81	.85
RDIST2	.37	.40	.25	.38	.49
RDIST3A	.40	.41	.42	.51	.53

Storage Requirements



Storage Requirements

problem	MA48	UMFPACK/1	UMFPACK/2	IGP
JPWH_991	115,039	164,460	100,329	193,889
LNS_3937	830,168	1,142,334	1,013,918	570,231
LNSP3937	824,588	1,221,638	856,906	565,197
ORANI678	337,068	667,652	403,672	467,958
ORSREG_1	451,370	526,821	380,909	504,489
SAYLR4	661,397	933,434	634,254	799,840
SHERMAN2	597,002	690,484	610,608	494,145
SHERMAN3	539,069	1,045,884	456,463	609,768
SHERMAN5	544,243	600,107	518,375	396,405
WATT_1	273,923	729,113	283,842	345,830
WATT_2	278,335	730,837	272,221	330,841
SHYY41	286,197	386,507	211,602	296,325
HYDR1	201,968	308,195	181,570	155,249
RDIST1	827,640	1,199,301	604,172	728,010
RDIST2	411,188	717,980	305,350	414,105
RDIST3A	408,638	712,320	452,470	452,060

Concluding Remarks

- Personal (and philosophical) view.
- Single machine data.

- Did not examine all options available.
  - drop tolerance, iterative refinement, ...
  - tuning parameters in full pivoting codes

- Future work:
  - preordering for partial pivoting codes
  - exploiting memory hierarchy
  - codes for parallel architectures