

Coupling MUMPS and ordering software *

Grégoire Richard[†]

January 2002

CERFACS report: WN/PA/02/24

Abstract

We present the work performed in the context of a three month internship funded by IRIT-CNRS done at CERFACS-ENSEEIH located in Toulouse (France). The first part of this work has consisted of in the integration of up-to-date sequential ordering software into MUMPS: SCOTCH from F. Pellegrini (LaBRI - Bordeaux), PORD from Y. Schulze (Padenborn University - Germany), MeTiS from G. Karypis and V. Kumar (University of Minnesota) and Approximate Min. Fill (Toulouse). In the second part, the influence of the ordering software choice on the size of the factorized matrices and on the scalability has been studied on a Cray T3E from NERSC (Lawrence Berkeley National Laboratory).

*The internship is supported by IRIT-CNRS. This internship also utilized resources of the National Energy Research Scientific Computing Center (NERSC) which is supported by the Director, Office of Advanced Scientific Computing Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

[†]richard@enseeiht.fr and gregoirerichard@yahoo.fr

Contents

1	Introduction	1
2	MUMPS New Ordering Interface	1
2.1	Previous MUMPS Ordering Interface	1
2.2	New MUMPS Ordering Options	2
3	Changes to the Code	2
3.1	MUMPS Analysis Structure	2
3.2	Sparsity Pattern Data Structure	3
3.2.1	Specification for MeTiS, SCOTCH and PORD	3
3.2.2	Specification for AMD and AMF	4
3.2.3	A new MUMPSGD	4
3.3	Assembly Tree Data Structure	5
3.4	Technical Note for Installation	5
3.4.1	MUMPS	5
3.4.2	SCOTCH	5
3.4.3	PORD	5
3.4.4	MeTiS	6
3.5	Work to Do	6
3.5.1	Tree given	6
3.5.2	Change partitioner parameters	6
3.5.3	Remaining problems	6
4	Analysis of Performance	6
4.1	Presentation of the Test	6
4.2	Real Problems	6
4.3	Grid Problems	7
4.4	Analysis of Results	7
4.4.1	Ordering Analysis	8
4.4.2	Scalability Analysis	8
5	Conclusion	8
6	Appendix	10
6.1	Analysis of Ordering Phase	10
6.1.1	Real cases	10
6.1.2	Square grids	10
6.1.3	Rectangular grids	11
6.2	Factorization and Solve Phases Analysis	11
6.2.1	Real matrices	11
6.2.2	Grid problems	13
6.2.3	Figures	15

1 Introduction

A new class of sequential hybrid ordering packages, that computes approximate solutions to the minimum fill-in problem [9], has been developed in the last few years. Those algorithms combine top-down multilevel nested dissection algorithms with bottom-up minimum degree (AMD [1] or MMD [4]) or minimum fill (MMF [5, 7]) local heuristics. MUMPS (MULTifrontal Massively Parallel Solver) has been using an approximate minimum degree ordering algorithm AMD, a nested dissection ND or a externally computed given ordering. Usually MeTiS [3] was used to compute the ordering that was given to MUMPS. The used of hybrid ordering partitioners is expected to improve significantly the fill-in of the factorized matrices, and also, the scalability of the elimination tree obtained by symbolic factorization.

The new partitioners that have been integrated into MUMPS are AMF, which stands for approximate minimum fill algorithm, SCOTCH [6] which is an hybridizing of nested dissection and AMD, PORD [8] which implements a tighter coupling between nested dissection and approximate minimum fill, and MeTiS which used multilevel nested dissection and minimum degree. The degree of coupling depends on the partitioner. AMF, SCOTCH and PORD have been tightly coupled in the sense that we obtain the assembly tree as an output of the partitioner. Whereas MeTiS has been loosely coupled in the sense that we only obtain the ordering, and then perform a symbolic factorization.

We have done a comparison of these new integrated partitioners, on both a set of symmetric and unsymmetric real problems, and on regular 3D grid problems. We present here results relative to the analysis, the factorization and the solution phases of MUMPS. Times and fill-in informations are provided in tabular, whereas megaflop rate and efficiency are provided graphically. As MUMPS may be run several successive times, using only one analysis, or/and one factorization, decrease of factorization and solution time is very interesting for the user, even if the analysis time becomes larger. Nevertheless, because of significant differences between some partitioners we also provided the partitioning time.

This paper is organized as follows. Section 2 presents interface changes in MUMPS concerning the choice of the ordering software. Section 3 mentions changes done to the code, the work it remained to do for a further release, and some issues that may be encountered during the installation of the software. Section 4 describes the test environment and the set of problems we run and gives a summary of some phenomenon which appear in the test. Numerical results are provided in an appendix.

2 MUMPS New Ordering Interface

In this Section, we present to the user changes in MUMPS interface.

2.1 Previous MUMPS Ordering Interface

Previously MUMPS provided three options to perform the ordering:

- Approximate Minimum Degree (AMD)
- Nested Dissection (ND - Scalapack)

- Ordering Given

The choice of those options may be done in two different ways depending how MUMPS is called. If the driver `testhb` is used, the user only needs to change the name of the ordering options in the datafile. This should be: "AMD", "ND" or the path for the file that contain the ordering. The ordering file has to contains one integer value by line, and the value has to be between 1 and the order of the matrix, a value can appear only once. A command line for calling MUMPS on the CRAY T3E may be:

```
mpprun -n 4 testhb < dt
```

where 4 is the number of desired processors and dt is the datafile.

If the Fortran MUMPS function is called directly from another code, the user needs to set the parameter `ICNTL(7)`, in the structure `mumps_par`. The corresponding values are:

Ordering Option	AMD	ND	Ordering Given
ICNTL(7)	0	1	1

When MUMPS is called directly from an other code, the option ND is not provided. The user should use the ordering given option after having computed the ordering. In that case, the ordering is given to MUMPS as an array of size the order of the matrix: N, with values between 1 and N.

2.2 New MUMPS Ordering Options

We kept unchanged the interface of MUMPS while integrating the new ordering options. "AMF", "SCOTCH", "PORD" and "METIS" for the datafile correspond to the following values of `ICNTL(7)`:

Ordering Option	AMD	ND	Ordering Given	AMF	SCOTCH	PORD	MeTiS
ICNTL(7)	0	1	1	2	3	4	15

The loose coupling of MeTiS explains the choice of 15 instead of 5.

3 Changes to the Code

In this section we explain where changes have been made to in the source code of MUMPS, and in the other ordering packages.

3.1 MUMPS Analysis Structure

Our work was mainly concentrated in the `usedbyanalysis.F` file of the MUMPS distribution. This file contains the `MUMPS_MA41FD` routine, and its subroutines: `MUMPS_MA41GD`, `MUMPS_MA41HD`, `MUMPS_MA41JD`, `MUMPS_MA41KD` and `MUMPS_MA41LD`, whose function is to perform a symbolic factorization of the input matrices.

This symbolic factorization may be done two ways, corresponding to two branches of the code. The first branch consists of by:

- MUMPS_MA41GD: computes the sparsity pattern of $A + A^T$ from the non-zero structure of A .
- MUMPS_MA41HD: computes the ordering and assembly tree from the adjacency graph using AMD.

The second branch:

- MUMPS_MA41JD: computes the sparsity pattern of $A + A^T$ from a given ordering and the non-zero structure of A .
- MUMPS_MA41KD: performs a symbolic factorization computing an assembly tree from the adjacency graph and the given ordering.

And later on MUMPS_MA41LD performs a mapping of the assembly tree on the processors.

Our goal was to integrate in that program structure the call to the following partitioning software: SCOTCH, AMF, PORD, MeTiS.

As SCOTCH and PORD use for the local domains the same AMD algorithm as in MUMPS_MA41HD, it was possible to extract from them a forest of local assembly trees. Then, it remained to combine those trees with the global tree resulting from the Nested Dissection. In fact the complete tree structure is already constructed in a different form in PORD, so we just write a driver "MUMPSPORD" that translates the tree structure of PORD, into that of MUMPS. For SCOTCH, the driver "ESMUMPS" has been provided by Francois Pellegrini.

AMF has quite the same prototype as MUMPS_MA41HD. So the coupling was just a call to the AMF subroutine.

So these three packages have been tight coupled to MUMPS, i.e. they are called instead of MUMPS_MA41HD.

Concerning MeTiS, it was not clear that it was possible to extract an assembly tree as for SCOTCH or PORD, so we decided to perform a loose coupling of MeTiS with MUMPS. In fact, MeTiS was called before MUMPS_MA41JD to provide an ordering.

3.2 Sparsity Pattern Data Structure

The difference between the sparsity pattern data structure input of MeTiS, SCOTCH and PORD from that of AMF and AMD, leads us to develop a new function MUMPS_MA41GD "MUMPSGD", that produces a sparsity pattern with this data structure.

The sparsity pattern data structure is represented by an unweighted non-directed simple graph with no self edges, called an adjacency graph. A column (or a row, no matter because of the symmetry of $A + A^T$) is represented by a vertex, and an edge links vertex i and j if $a_{i,j}$ or $a_{j,i}$ is non-zero.

3.2.1 Specification for MeTiS, SCOTCH and PORD

Let N be the size of the matrix and NE the number of edges in the adjacency graph. The graph representing the sparsity pattern of the matrix is coded by two arrays:

- $xadj$ of size $N+1$, whose values are pointers to the indices of the array $adjncy$. Thus $xadj(i+1) - xadj(i)$ refers to the number of off-diagonal non-zero in the i^{th} column of $A + A^T$. The number of non-zero in the last column is known using $xadj(N+1)$,

justifying the size of *xadj*. This implies that the location of the non-zero in *adjncy* are consecutive without loss of space between two columns.

- *adjncy* of size $2*NE$. Each column stores all the off-diagonal non-zero, so a non-zero is represented twice.

3.2.2 Specification for AMD and AMF

The sparsity graph for AMD and AMF is represented by three arrays:

- *IPE* array of size N , whose values are the pointers to the indices of *ADJ*, except that when there is no off-diagonal non-zero $LEN(I) = 0$, in that case $IPE(I) = 0$.
- *LEN* array of size N , which values are the number of extra-diagonal non-zero in the i^{th} column.
- *ADJ* Array of size at least $2*NE$, which values are the extra-diagonal non-zero. It was not specified in the specification sheet that the columns are successively stored, while it is the case.

3.2.3 A new MUMPSGD

In order to incorporate `MeTiS`, `SCOTCH` and `PORD`, a function `MUMPSGD` is used instead of `MUMPS_MA41GD`. To sum up:

- the size of the *IPE* vector is $N+1$ instead of N .
- when $IPE(I)$ was 0 now $IPE(I + 1) = IPE(I)$
- *LEN* is no longer used
- The two vectors *adjncy* and *ADJ* are the same.

This has led to changes to `psl_analysis.F`.

In the previous version of `usedbyanalysis.F`, the internal working array *IW*, containing the ordering, the sparsity graph and the elimination tree, was declared as `INTEGER*4`, while `SCOTCH` used only the `INTEGER` type. So now *IW* is declared as `INTEGER`. With this version, `MUMPS` can work in 32 or 64 bytes precision.

Another issue has occurred, the indices of arrays are not the same using `FORTRAN` and `C`. `C` starts from 0, and `FORTRAN` starts from 1. As `MeTiS`, `PORD` and `SCOTCH` are written in `C` we have to be very careful, about the values of the pointer array *xadj*, or the representation of the ordering. `MeTiS` and `SCOTCH` have provided `FORTRAN` based functions, while for `PORD` we have made one pass on those two arrays to change the pointer value of *adj* and the vertex numbering of *adjncy*. In fact it seems that internally `MeTiS` also does a pass, while `SCOTCH` uses a language dependent base-value in the call of all its subroutines.

3.3 Assembly Tree Data Structure

The data structure of the assembly tree output from PORD and SCOTCH were different from that of MUMPS_MA41LD. The translation for SCOTCH was provided by Francois Pellegrini in the function "ESMUMPSF". The one for PORD has been developed in the driver "MUMPSPORDF".

We do not describe the assembly tree data structure of PORD and SCOTCH, that can be understood by reading the driver code, but we present the one of MUMPS.

An assembly tree is a tree of super-variables, which are sets of variables that have the same symbolic factorization properties. Each super-variable has one principal variable, and secondary variables. The coding of the MUMPS assembly tree is done by two arrays of size N:

- PE,
 - $PE(I) = 0$, if I is the root of the tree.
 - $PE(I) = -J$, if I is principal and its father super-variable in the tree is J.
 - $PE(I) = -J$, if I secondary and is absorbed in the a super-variable which principal is J. If J is later on absorbed, $PE(I)$ is still -J
- NV,
 - $NV(I) = K$ greater than 0, if I is a super-variable whose parent is $-PE(I)$ and frontal size K.
 - $NV(I) = 0$, if I is a secondary variable.

3.4 Technical Note for Installation

3.4.1 MUMPS

So far no decision concerning the location of of the drivers for the partitioning code have been done. They may be included into MUMPS or let into the partitioners source code. Some minor modifications have been done to the makefile, including the new libraries.

3.4.2 SCOTCH

SCOTCH is distributed with the solver EMILIO, that needs to be installed for using the partitioner. The GNU make is needed, and bash on sun-solaris. The MUMPS version using INTEGER*4, was not supported by SCOTCH that only accepts default INTEGRER. Thus we had some problems with the 64 bits precision.

We modified the provided ESMUMPS procedure in the file "esmumps.c", to match the new specification of MUMPSGD.

A new esmumps method that integrate SCOTCH parameters driving should be provided by François Pellegrini.

3.4.3 PORD

PORD is distributed with the solver SPACE, that needs to be installed for using the partitioner. A MUMSPORD driver has been developed in the file "greg_pord.c". We decided not to output

the ordering that is not needed in the symbolic factorization. Default options are used, but it is possible to change a bit that function in order to integrate parameter driving.

3.4.4 MeTiS

MeTiS provided FORTRAN callable functions. We used METIS_NODEND, which is the procedure called in the onmetis program.

3.5 Work to Do

Some minor modifications that have not been done are proposed here.

3.5.1 Tree given

For comparison purpose with ordering given, it would be useful to have a tree-given option.

3.5.2 Change partitioner parameters

The sensitivity of the partitioning as a function of its parameters is not known. Although users should not be concerned by these parameters, an analysis may lead to some advise about the depth to stop the nested dissection.

3.5.3 Remaining problems

PORD is fails on some unassembled matrices used by Jean-Cristophe Rioual at CERFACS. SCOTCH and PORD fail on ECL32.

4 Analysis of Performance

4.1 Presentation of the Test

In our study, we have used a set of test problems to illustrate the performance of our algorithms. Most results presented in this paper have been obtained on the Cray T3E-900 (512 DEC EV-5 processors, 256 Mbytes of memory per processor, 900 peak Megaflop rate per processor) from NERSC at Lawrence Berkeley National Laboratory, although there have been a few experiments on a 32 processor SGI Origin 2000 at CERFACS in Toulouse, France.

4.2 Real Problems

Our test matrices come from the forthcoming Rutherford-Boeing Sparse Matrix Collection [2]¹ and the industrial partners of the PARASOL Project². The PARASOL test matrices are available from Parallab (Bergen, Norway)³. There are larger matrices available from this Web site but it is difficult to process them symbolically on a single processor.

¹Web page <http://www.cse.clrc.ac.uk/Activity/SparseMatrices/>

²EU ESPRIT IV LTR Project 20160

³Web page <http://www.parallab.uib.no/parasol/>

<i>Real Unsymmetric Assembled (RUA)</i>				
Matrix name	Order	No. of entries	StrSym ^(*)	Origin
BBMAT	38744	1771722	0.54	Rutherford-Boeing (CFD)
INVEXTR1	30412	1793881	0.97	PARASOL (Polyflow S.A.)
MIXTANK	29957	1995041	1.00	PARASOL (Polyflow S.A.)
TWOTONE	120750	1224224	0.28	Rutherford-Boeing (circuit sim)
WANG4	26068	177196	1.00	Rutherford-Boeing (semiconductor)
<i>Real Symmetric Assembled (RSA)</i>				
Matrix name	Order	No. of entries	Origin	
SHIP_003.RSA	121728	4103881	PARASOL (Det Norske Veritas)	

Table 1: Test matrices. ^(*) StrSym is the number of nonzero matched by nonzero in symmetric locations divided by the total number of entries (that is, a symmetric matrix has value 1.0).

Note that matrices MIXTANK and INVEXTR1 have been modified because of out-of-range values (causing underflow). To maintain, as much as possible, the same numerical behavior during the factorization all entries with an exponent smaller than -300 have been set to numbers with the same mantissa but with an exponent of -300. (In fact the out-of-range underflow values in the initial matrix had an exponent equal to either -308 or -309.)

For each linear system, the right-hand side vector is generated so that the true solution is a vector of all ones.

4.3 Grid Problems

To further analyze and understand the scalability of our solvers, we also report results obtained for the 11-point discretization of the Laplacian operator on three-dimensional (NX, NY, NZ) grid problems.

We consider a set of 3D cubic (NX=NY=NZ) and rectangular (NX, NX/4, NX/8) grids on which a nested dissection ordering is used. When increasing the number of processors, we have tried as much as possible to maintain a constant number of operations per processor while keeping as much as possible the same shape of grids. It was not possible to satisfy all these constraints, thus the number of operations per processor is not completely constant.

Since all our test matrices are symmetric, we can use MUMPS to compute either an \mathbf{LDL}^T or a \mathbf{LU} factorization. For a given matrix, the unsymmetric solver perform roughly twice as many operations the symmetric one.

4.4 Analysis of Results

The comparison of the partitioners has been divided into two parts:

- the analysis properties, including fill-in reduction with the number of entries in factor and the number of operations in the factorization, and the time for the ordering.
- the scalability analysis, including time for factorization and solve, and graph representing Megaflop rate and efficiency.

Analysis results are presented for all the partitioners on the grid problems, and all the partitioner except ND on the real ones.

For the sake of clarity, Megaflop rate and efficiency graphs are presented for the three better partitioners MeTiS, SCOTCH and PORD, and ND which is a good reference for regular grid problems.

The Megaflop rate is defined as the number of floating point operations by processor. The efficiency is defined as the Megaflop rate scaled to the Megaflop rate with only one processors.

When results were not available for one processor we scaled to the lowest number of processor giving a result. When results were not available for one of the partitioning, we scaled the the missing value to the mean of the two others (this is the case with WANG4 and MeTiS).

When the computation failed (due generally to memory problems) the value has been replaced by a blank.

4.4.1 Ordering Analysis

A better fill-in reduction is obtained with the hybrid partitioner (INVEXTR1), but AMF also performed quite well on some cases, and is generally better than AMD.

Hybrids are slower for the ordering time, this may due to the coupling time (recopy of some arrays) or some software engineering choices. Note that SCOTCH is very slow for the ordering time.

4.4.2 Scalability Analysis

AMF and AMD have some surprising behavior: they have a very large factorization time, whereas they have a good fill-in reduction. That may be explained by some problems in the mapping of the assembly tree over processors.

Although, SCOTCH has not the best fill-in reduction, it appears to have the best scalability, leading to better factorization and solution time, followed by MeTiS and PORD.

5 Conclusion

It appears from the set of tests that we have run that the new partitioners that have been coupled with MUMPS lead to significant improvements in fill-in reduction and in scalability. SCOTCH is, in general, the best concerning factorization and solution times, which are the measures that interest users. Nevertheless, the time for ordering with SCOTCH is a lot larger than for other partitioners. This drawback should be eliminated with a better internal coupling between the local minimum degree algorithms and nested dissection. Moreover, SCOTCH and PORD are memory consuming preventing them from running on very large problems on distributed memory systems, and are not stable on some graph (for example ECL32, or some unassembled matrices for PORD).

On the other hand, AMF performs quite well on most the tests we ran, sometimes as well as the hybrid partitioners, and is a lot less memory consuming. Some software problems on the mapping of the assembly tree on the processors have lead, in some cases, to results for AMF not as good as expected. Nevertheless AMF is better than AMD on one processor.

In the future the default ordering options that will be integrated with MUMPS should be a combination of AMF for problems that may lead to memory issues, and SCOTCH for the others. AMF heuristics may be integrated with SCOTCH for local problems with larger sub-domains than presently.

Acknowledgments

I would like to thank Michel Dayde that make possible my come back at the ENSEEIHT for this internship, and Patrick Amestoy for being my advisor and teacher of all the moment.

References

- [1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.
- [2] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services and Report TR/PA/97/36 from CERFACS, Toulouse.
- [3] G. Karypis and V. Kumar. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0*. University of Minnesota, September 1998.
- [4] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, 1985.
- [5] E. Ng and P. Raghavan. Performance of greedy heuristics for sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 20:902–914, 1998.
- [6] F. Pellegrini, J. Roman, and P. R. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Practice and Experience*, 12:69–84, 2000.
- [7] Edward Rothberg and Stanley C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM Journal on Matrix Analysis and Applications*, 19(3):682–695, 1998.
- [8] J. Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *to appear in BIT*, 1999.
- [9] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2:77–79, 1981.

6 Appendix

6.1 Analysis of Ordering Phase

6.1.1 Real cases

<i>NZ in factors</i> (10^6)	<i>BBMAT</i>	<i>INVEXTR1</i>	<i>MIX</i>	<i>SHIP003</i>	<i>TWOTONE</i>	<i>WANG4</i>
<i>SCOTCH</i>	36.99	16.23	21.02	64.34	24.75	9.94
<i>PORD</i>	33.79	14.99	19.32	67.76	28.37	7.64
<i>METIS</i>	37.24	15.98	19.01	61.28	25.52	8.33
<i>AMF</i>	37.59	22.77	26.23	62.94	22.65	9.38
<i>AMD</i>	45.99	30.34	38.53	77.08	22.11	11.61

<i>nb of operations</i> (10^9)	<i>BBMAT</i>	<i>INVEXTR1</i>	<i>MIX</i>	<i>SHIP003</i>	<i>TWOTONE</i>	<i>WANG4</i>
<i>SCOTCH</i>	25.15	7.96	15.80	85.64	25.47	5.95
<i>PORD</i>	23.74	7.64	15.18	111.9	36.95	4.61
<i>METIS</i>	28.20	8.44	13.55	83.25	28.84	5.12
<i>AMF</i>	28.45	20.17	30.98	95.79	29.41	6.86
<i>AMD</i>	41.35	35.78	64.38	155.2	29.34	10.51

<i>time ordering</i> (s)	<i>BBMAT</i>	<i>INVEXTR1</i>	<i>MIX</i>	<i>SHIP003</i>	<i>TWOTONE</i>	<i>WANG4</i>
<i>SCOTCH</i>	47.03	42.28	42.33	14.55	63.73	10.63
<i>PORD</i>	29.57	10.70	6.83	8.66	64.27	9.85
<i>METIS</i>	12.98	7.39	6.52	6.76	23.31	2.53
<i>AMF</i>	15.68	1.56	0.93	1.84	5.93	1.18
<i>AMD</i>	3.38	1.38	0.93	1.65	5.33	0.88

6.1.2 Square grids

<i>NZ in factors</i> (10^6)	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	5.42	9.12	13.87	25.12	39.08	62.66	76.44	86.23
<i>PORD</i>	6.57	12.22	18.83		54.05	95.33	111.62	
<i>METIS</i>	5.68	10.04	15.86	24.31	36.35	63.49	94.67	105.84
<i>AMF</i>	5.72	10.70	15.89	29.66	46.84	73.38	91.28	
<i>AMD</i>	6.70	12.29	18.95	34.79	60.32	96.00		
<i>ND</i>	5.70	9.85	14.07	24.57	38.99	60.65	83.03	96.96

<i>nb of operations</i> (10^9)	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	3.03	6.47	12.28	30.44	61.50	116.0	158.4	183.6
<i>PORD</i>	4.74	10.93	26.03		106.9	255.3		
<i>HAMD</i>	4.30	10.66	19.03	51.24	95.38	188.2	240.6	
<i>AMD</i>	4.417	10.49	19.47	50.62	111.1			
<i>METIS</i>	4.09	9.51		33.61	52.05	141.6	278.7	301.5
<i>ND</i>	3.60	8.00	13.45	30.09	59.13	112.7	178.1	

<i>time ordering(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	10.69	16.59	23.33	42.25	61.56	90.13	116.57	145.73
<i>PORD</i>	5.03	7.31	11.34		26.20	34.05	41.16	
<i>AMF</i>	0.63	0.96	1.30	1.81	3.00	3.54	4.64	
<i>AMD</i>	0.57	0.83	1.13	1.61	2.69	3.18		
<i>METIS</i>	2.71	4.35	5.63	9.12	13.10	18.68	23.79	26.48
<i>ND</i>	0.66	1.10	1.48	2.36	3.81	5.01	6.80	7.47

6.1.3 Rectangular grids

<i>NZ in factors(10⁶)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	4.70	7.97	12.07	21.18	34.63	51.81	59.80	78.40
<i>PORD</i>	4.69	8.16	12.97	22.51	37.08	54.47	65.45	81.81
<i>METIS</i>	4.79	8.22	12.04	22.21	35.38	56.30	66.42	79.05
<i>AMF</i>	4.45	7.28	11.07	18.43	32.47	49.41	69.16	74.13
<i>AMD</i>	5.83	8.59	13.29	22.87	37.98	72.45		
<i>ND</i>	5.10	8.63	13.30	22.01	35.27	53.91	64.73	79.28

<i>nb of operations(10⁹)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	1.67	3.52	6.60	15.00	32.42	56.66	69.68	109.5
<i>PORD</i>	1.72		6.81	15.54	32.01	53.55		
<i>METIS</i>	2.02	4.47	7.63	19.49	39.53	81.93	101.6	
<i>AMF</i>	1.55		5.19	11.06	26.84	50.97	92.79	89.23
<i>AMD</i>	2.54		7.01	13.83	28.85	101.7		
<i>ND</i>	2.24	4.76	8.97	18.41	36.47	67.81	89.60	118.2

<i>time ordering(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	11.61	17.14	26.41	38.91	61.62	88.07	99.09	129.53
<i>PORD</i>	6.17	8.43	10.42	17.14	25.07	32.35	43.06	42.22
<i>AMF</i>	0.84	1.04	1.37	1.99	2.84	3.77	4.58	5.01
<i>AMD</i>	0.65	0.82	1.08	1.56	2.26	3.06		
<i>METIS</i>	3.27	4.98	6.93	10.68	15.12	21.30	24.13	28.53
<i>ND</i>	0.81	1.31	1.82	2.92	4.30	6.21	7.28	8.67

6.2 Factorization and Solve Phases Analysis

6.2.1 Real matrices

BBMAT

<i>facto(s)</i>	2	4	8	16	32	48	64
<i>SCOTCH</i>	36.84	19.83	13.88	8.34	5.69	6.57	6.39
<i>PORD</i>	50.23	27.28	18.02	12.18	9.78	9.00	8.58
<i>METIS</i>		39.66	24.18	15.31	10.92	10.73	9.01
<i>AMF</i>		103.87	70.06	51.13	45.74	42.23	41.73
<i>AMD</i>		43.76	23.39	15.18	12.29	11.08	10.32
<i>ND</i>		48.22	23.93	17.94	13.75	14.03	13.69

<i>solve(s)</i>	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.45	0.27	0.22	0.17	0.16	0.20	0.16
<i>PORD</i>	0.58	0.41	0.35	0.29	0.27	0.28	0.25
<i>METIS</i>		0.39	0.27	0.25	0.23	0.31	0.26
<i>AMF</i>		1.83	2.00	2.08	2.12	2.16	2.13
<i>AMD</i>		0.46	0.31	0.27	0.27	0.36	0.27
<i>ND</i>		0.48	0.30	0.26	0.29	0.42	0.31

TWOTONE

<i>facto(s)</i>	8	16	32	48	64
<i>SCOTCH</i>	13.59	9.65	7.14	6.50	6.05
<i>PORD</i>	27.60	22.80	18.99	18.11	18.19
<i>METIS</i>	26.89	17.31		13.26	11.00
<i>AMF</i>	46.08	28.46		19.69	19.54
<i>AMD</i>	21.23	17.11		13.15	11.94

<i>solve(s)</i>	8	16	32	48	64
<i>SCOTCH</i>	0.70	0.59	0.49	0.51	0.44
<i>PORD</i>	1.04	1.10	0.95	1.01	0.97
<i>METIS</i>	0.96	0.68		0.69	0.60
<i>AMF</i>	1.70	1.60		1.57	1.57
<i>AMD</i>	0.80	0.81		1.02	0.85

WANG4

<i>facto(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	20.71	11.07	6.00	3.67	2.70	2.53	2.30	2.28
<i>PORD</i>	15.65	11.11	6.28	5.69	3.43	3.04	3.06	2.91
<i>METIS</i>	20.32	12.40	6.53	3.97	3.00	2.51	2.49	2.64
<i>AMF</i>	21.85	17.63	8.56	5.92	4.91	4.40	4.21	4.18
<i>AMD</i>	34.98	18.63	11.01	6.42	4.51	3.24	3.63	3.29
<i>ND</i>	13.57	7.72	4.41	2.61	2.03	1.81	2.44	1.94

<i>solve(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.44	0.27	0.16	0.12	0.11	0.11	0.19	0.11
<i>PORD</i>	0.50	0.36	0.24	0.18	0.15	0.15	0.15	0.15
<i>METIS</i>	0.52	0.32	0.19	0.14	0.12	0.13	0.18	0.13
<i>AMF</i>	0.53	0.40	0.26	0.23	0.19	0.18	0.19	0.19
<i>AMD</i>	0.62	0.39	0.27	0.18	0.15	0.14	0.27	0.14
<i>ND</i>	0.46	0.30	0.19	0.14	0.12	0.13	0.18	0.13

MIXTANK

<i>facto(s)</i>	4	8	16	32	48	64
<i>SCOTCH</i>	15.42	8.97	5.71	4.59	3.95	3.90
<i>PORD</i>	15.83	9.83	7.28	6.94	6.54	6.52
<i>METIS</i>	13.48	7.93	5.38	4.31	4.04	3.62
<i>AMF</i>	31.78	16.88	14.56	13.61	12.82	13.43
<i>AMD</i>		31.86	20.91	15.26	15.30	13.90
<i>ND</i>		28.51	18.66	15.50	15.54	14.13

<i>solve(s)</i>	4	8	16	32	48	64
<i>SCOTCH</i>	0.25	0.17	0.14	0.14	0.23	0.13
<i>PORD</i>	0.30	0.23	0.21	0.19	0.22	0.19
<i>METIS</i>	0.26	0.18	0.15	0.15	0.22	0.14
<i>AMF</i>	0.36	0.31	0.28	0.28	0.30	0.28
<i>AMD</i>		0.33	0.31	0.32	0.54	0.31
<i>ND</i>		0.32	0.29	0.31	0.49	0.30

SHIP_003.RSA

<i>facto(s)</i>	16	32	48	64
<i>SCOTCH</i>	23.74	14.94	12.72	12.51
<i>PORD</i>		25.76	25.77	24.36
<i>METIS</i>		18.33	16.65	14.00
<i>AMF</i>	33.36	24.40	23.24	22.07
<i>AMD</i>		34.50	30.31	24.47

<i>solve(s)</i>	16	32	48	64
<i>SCOTCH</i>	0.52	0.44	0.42	0.43
<i>PORD</i>		1.19	1.10	1.03
<i>METIS</i>		0.66	0.69	0.54
<i>AMF</i>	1.31	1.22	1.09	1.07
<i>AMD</i>		0.83	0.87	0.72

INVEXTR1

<i>facto(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	28.48	14.28	8.38	4.77	3.42	3.00	2.45	2.73
<i>PORD</i>	29.15	17.53	11.51	7.17	5.96	5.45	5.22	5.18
<i>METIS</i>	34.28	19.82	11.43	6.24	4.62	3.50	3.52	3.47
<i>AMF</i>			30.30	18.76	15.72	13.51	13.07	13.06
<i>AMD</i>			35.18	23.66	19.03	16.43	16.20	16.10

<i>solve(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.51	0.30	0.19	0.15	0.12	0.12	0.16	0.12
<i>PORD</i>	0.54	0.36	0.33	0.21	0.19	0.18	0.18	0.18
<i>METIS</i>	0.61	0.37	0.24	0.19	0.15	0.14	0.20	0.14
<i>AMF</i>			0.49	0.43	0.42	0.43	0.43	0.44
<i>AMD</i>			0.46	0.35	0.34	0.36	0.48	0.35

6.2.2 Grid problems

square symmetric grids

<i>facto(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	14.95	15.22	13.48	16.03	16.18	18.41	19.81	21.52
<i>PORD</i>	22.81	29.87	32.15		31.26	44.81		
<i>METIS</i>	24.99	30.73		29.59	26.12	37.16	57.97	65.68
<i>AMF</i>	19.60	33.38	24.07	27.91	27.04	31.33	35.41	
<i>AMD</i>	21.52	29.53	24.39	26.88	30.25			
<i>ND</i>	19.32	20.76	17.30	18.88	19.00	21.66	29.23	

<i>solve(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.40	0.38	0.38	0.39	0.45	0.49	0.60	0.55
<i>PORD</i>	0.50	0.57	0.70		0.98	1.19		
<i>METIS</i>	0.50	0.46		0.74	0.88	1.16	1.73	1.50
<i>AMF</i>	0.47	0.64	0.69	0.66	0.83	0.97	1.25	
<i>AMD</i>	0.55	0.66	0.66	0.63	0.78			
<i>ND</i>	0.47	0.46	0.47	0.65	0.57	0.70	0.83	

square unsymmetric grids

<i>facto(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	20.45	21.33	19.50	25.93	29.09	37.05	41.40	
<i>PORD</i>	30.58	41.83	49.45		75.91	135.99		
<i>METIS</i>	31.63	39.82			38.27	65.16	109.43	
<i>AMF</i>	25.95	48.46	34.79	54.55	56.31	94.78	115.95	
<i>AMD</i>	28.94		35.51	43.18	58.30	77.33	91.31	91.68
<i>ND</i>	25.14	29.41	24.41	30.71	31.67	39.19	51.71	57.76

<i>solve(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.39	0.37	0.31	0.35	0.42	0.62	0.74	
<i>PORD</i>	0.51	0.52	0.53		0.94	1.68		
<i>METIS</i>	0.49	0.46			0.53	0.95	1.36	
<i>AMF</i>	0.47	0.57	0.53	0.65	0.85	1.37	1.86	
<i>AMD</i>	0.55		0.48	0.49	0.76	1.08	1.11	1.41
<i>ND</i>	0.46	0.47	0.42	0.45	0.52	0.77	0.98	1.06

rectangular symmetric grids

<i>facto(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	9.88	9.91	8.49	9.51	11.39	11.40	11.43	15.55
<i>PORD</i>	11.34		12.63	12.99	14.24	15.62		
<i>METIS</i>	13.38	13.66	12.01	17.31	19.81	20.46	28.86	
<i>AMF</i>	11.19		19.75	14.30	15.58	20.46	25.14	22.44
<i>AMD</i>	14.46		12.90	12.68	14.26	29.27		
<i>ND</i>	13.34	12.90	12.14	13.29	13.24	14.55	15.14	18.04

<i>solve(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.38	0.35	0.37	0.49	0.55	0.56	0.55	0.66
<i>PORD</i>	0.43		0.66	0.90	0.98	0.89		
<i>METIS</i>	0.46	0.42	0.45	0.61	1.03	2.33	1.40	
<i>AMF</i>	0.45		1.27	1.50	1.55	1.96	1.58	1.88
<i>AMD</i>	0.51		0.76	0.97	1.11	2.18		

rectangular unsymmetric grids

<i>facto(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	12.76	13.05	11.82	14.21	18.40	20.54	23.89	29.49
<i>PORD</i>	13.81	21.41	17.32	23.64	29.05	33.60		
<i>METIS</i>	16.73	17.53	16.40	23.35	32.07	37.90	36.59	47.83
<i>AMF</i>	13.39	21.62	23.54	26.56	33.83	50.11	71.22	45.01
<i>AMD</i>	18.74	18.20	18.43	21.93	27.96	69.44	83.32	95.51
<i>ND</i>	17.16	17.17	16.68	19.70	20.44	26.90	27.27	30.72

<i>solve(s)</i>	1	2	4	8	16	32	48	64
<i>SCOTCH</i>	0.37	0.34	0.33	0.32	0.43	0.56	0.73	0.84
<i>PORD</i>	0.44	0.58	0.57	0.76	0.74	0.84		
<i>METIS</i>	0.46	0.42	0.40	0.42	0.65	0.82	0.90	1.07
<i>AMF</i>	0.46	0.67	1.27	1.44	1.63	2.10	1.99	1.66
<i>AMD</i>	0.52	0.51	0.67	0.68	0.83	1.24	1.34	1.60
<i>ND</i>	0.45	0.41	0.36	0.44	0.46	0.63	0.73	0.81

6.2.3 Figures

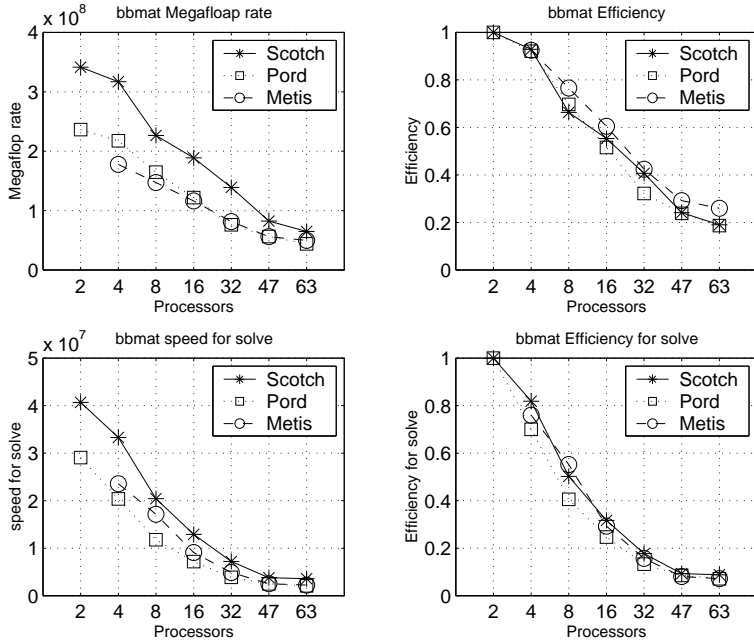


Figure 1: BBMAT

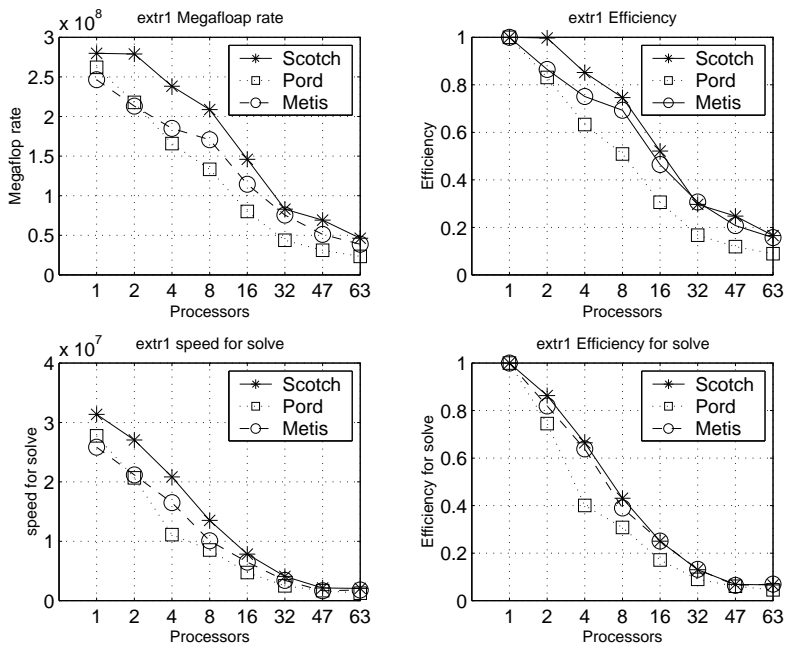


Figure 2: INVEXTR1

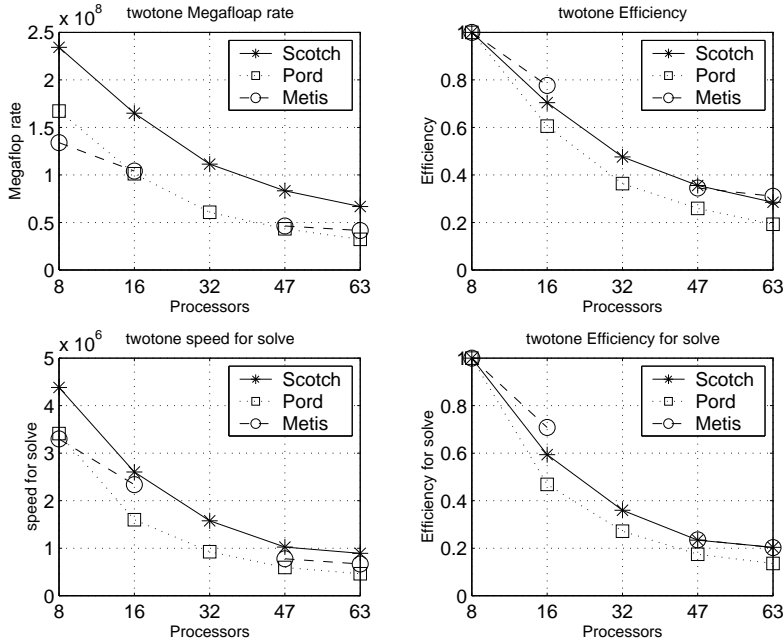


Figure 3: TWOTONE

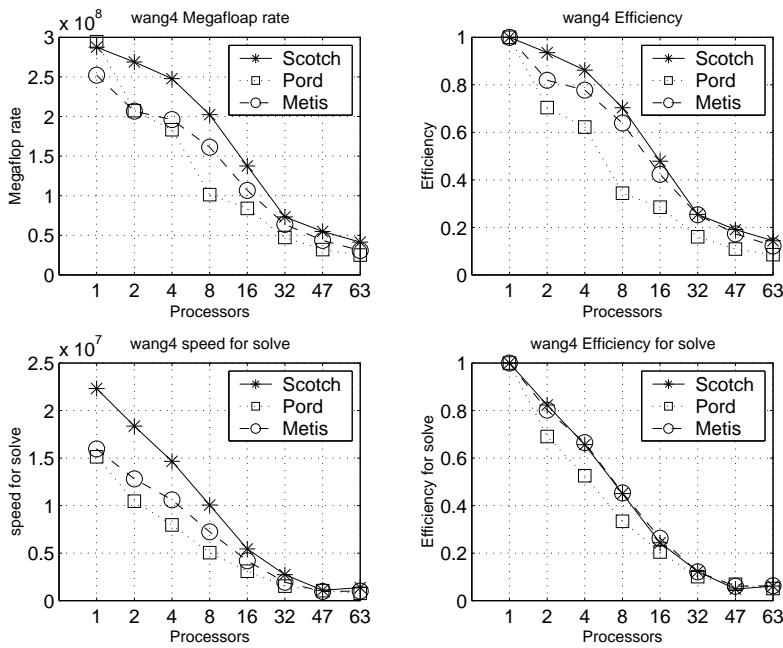


Figure 4: WANG4

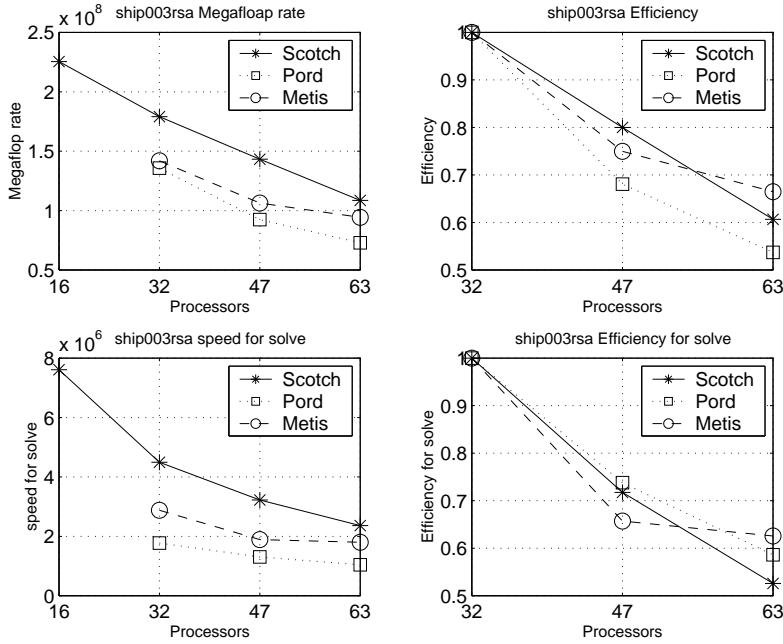


Figure 5: SHIP_003.RSA

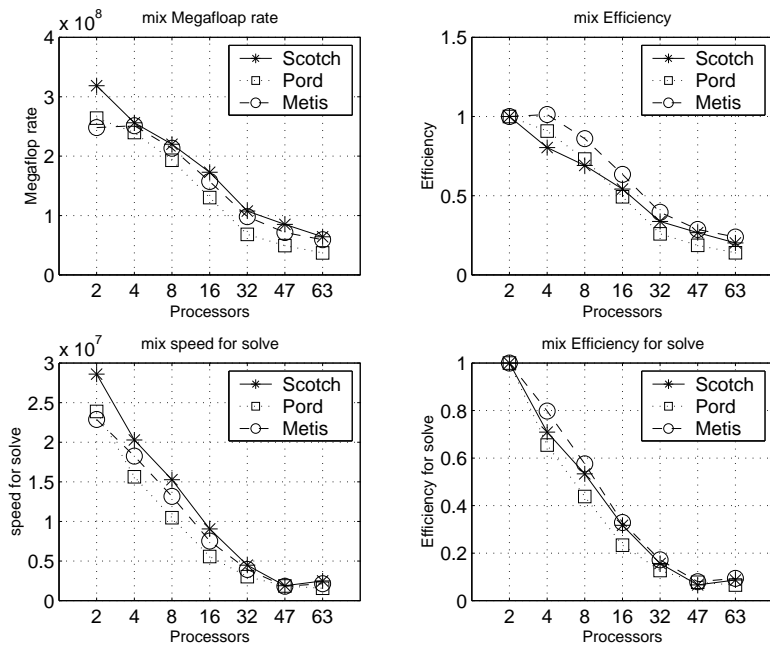


Figure 6: MIXTANK

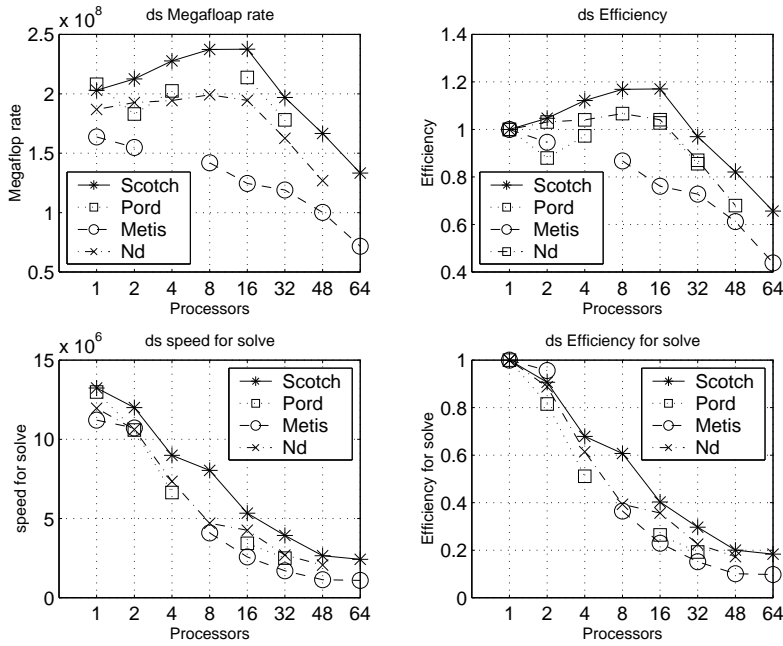


Figure 7: square symmetric grid

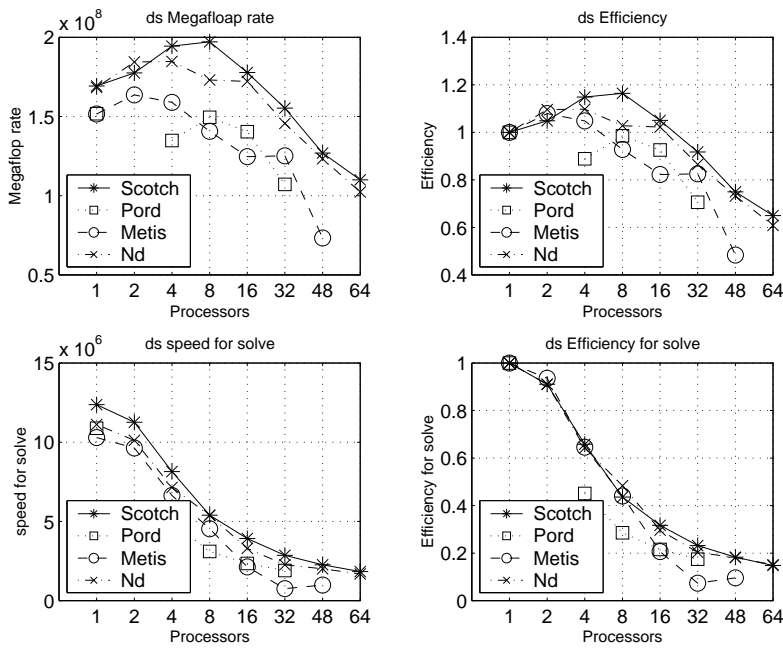


Figure 8: rectangular symmetric grid

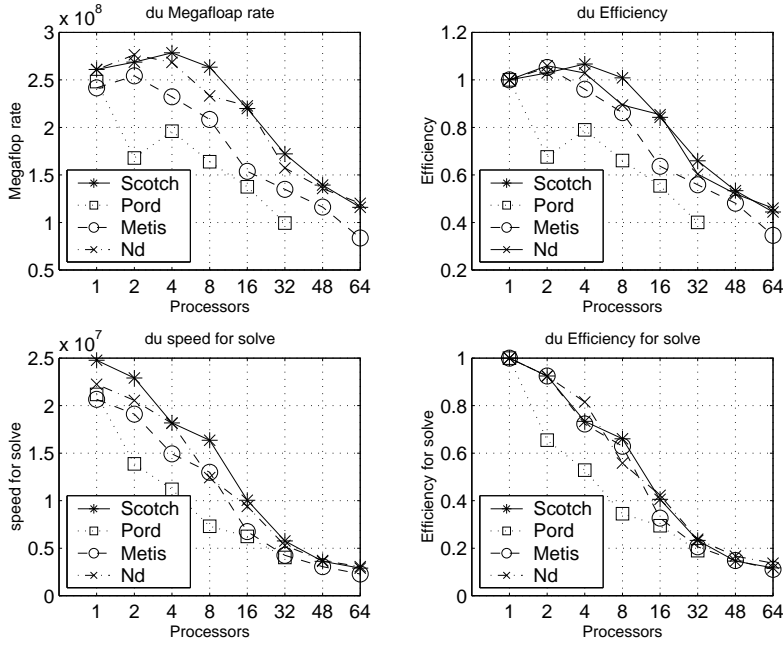


Figure 9: rectangular unsymmetric grid

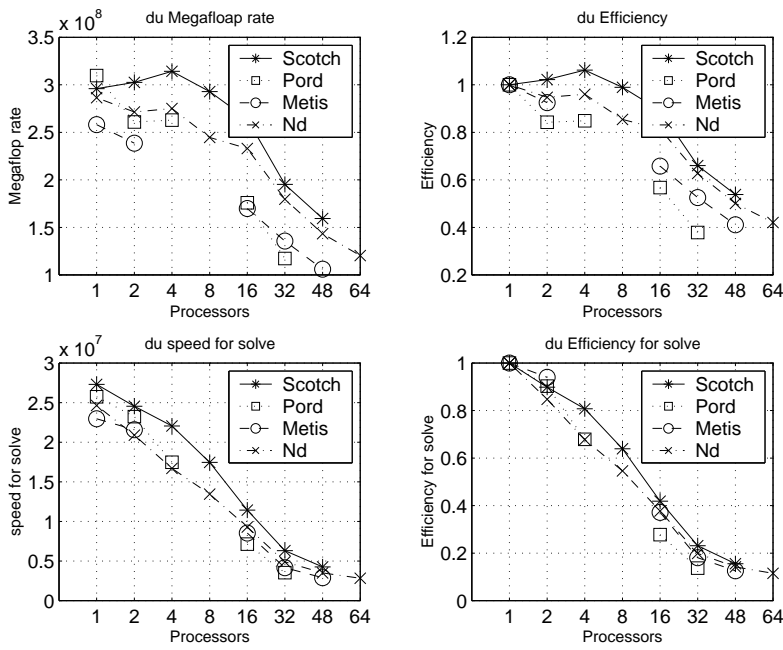


Figure 10: square unsymmetric grid