Experiments on Sparse Matrix Partitioning

S. Riyavong[†]

CERFACS Working Note WN/PA/03/32

Abstract

We have undertaken experiments to determine the comparative quality of sparse matrix partitioners. A large selection of test matrices are partitioned and then permuted so that the resulting form exhibits a block structure. This form is useful for implementing sparse matrix-vector multiplication in a parallel computing environment where each block-row strip will be assigned to a single computing node.

Key words. sparse matrix, matrix-vector multiplication, matrix partitioning.

1 Introduction

Most modern iterative methods for solving a sparse linear system have as their key computational step the computation

$$y = Ax \tag{1}$$

where A is a sparse matrix, and x and y are input and output vectors, respectively. For large-scale computing, the calculation of (1) is very time comsuming when computed without paying particular attention to data structure and computer architecture. However, the elapsed computational time can be greatly reduced on a parallel computer by partitioning the original problem into blocks and then distributing them to different processors and performing the computation in parallel. A good partitioner should partition the vectors x and y and the nonzero entries of A so that each block contains almost the same number of entries and is as independent as possible from other blocks so that, when the blocks are distributed, communication between them is low. Usually, matrix partitioning is formulated as graph partitioning [4]. In this note, we consider graph partitioners that implement multilevel recursive algorithms [5] and test them with some application-derived matrices from the Rutherford-Boeing Collection [1]. In the next section, we introduce graph partitioners and the algorithms they use for partitioning. In Section 3, we describe the procedures for manipulating matrices, which are partitioned by different partitioners, to form block structures. Section 4 gives experimental results and a model for the communication cost. Finally, we draw some conclusions in Section 5.

2 Matrix Partitioning Programs

For a survey on graph partitioning and software with application in scientific computing, the references [12, 13] should be consulted. In our experiments, we tested only four graph partitioners: PaToH [17, 18, 14, 16, 15], hMeTiS [9, 10, 8, 7], Mondriaan [19], and Monet [6] as detailed below.

2.1 PaToH and hMeTiS

These partitioners use the hypergraph model to partition a graph. We first introduce the basic idea of a hypergraph. A hypergraph $H = (V, N)$ consists of a set of vertices $V = \{v_1, v_2, \ldots, v_m\}$ and a set of hyperedges or nets $N = \{n_1, n_2, \ldots, n_m\}$ where $n_i \in P(V)$, the set of all subsets of V. The entries of V are partitioned into K sets so that the number of entries in each set obeys a load balance criterion and the number of cut nets, i.e. the nets that have at least one entry in two sets or more, is minimized. This is important because communication between processors is proportional to the size of the cut nets. Both PaToH and hMeTiS use a multilevel hypergraph partitioning algorithm which consists of three phases.

[†]CERFACS, 42 Avenue G. Coriolis, 31057 Toulouse Cedex, France

1. **Coarsening Phase**
   The original hypergraph is coarsened into a sequence of smaller hypergraphs by grouping together vertices. This phase terminates when the size of the coarsened hypergraph is below a predetermined number. Then it can be partitioned by a heuristic method based on the Kernighan-Lin (KL) [11] algorithm.

2. **Initial Partitioning Phase**
   In this phase, the coarsened hypergraph with a small number of vertices is bipartitioned by heuristic methods based on the Kernighan-Lin algorithm. The hMeTiS package uses multilevel random bisection using KL followed by the Fiduccia-Matheyses (FM) [3] algorithm while PaToH uses Greedy Hypergraph Growing (GHG) [18].

3. **Uncoarsening and Refinement Phase**
   The coarsest hypergraph that is partitioned will be successively projected to the next level finer hypergraph. hMeTiS uses algorithms based on FM while PaToH implements algorithms based on Kernighan-Lin and Fiduccia-Matheyses (KLFM).

The hypergraph partitioners mentioned above can be used to partition the nonzero entries of a sparse matrix, where the columns of the matrix are viewed as vertices of the hypergraph and the rows are nets or hyperedges. Both PaToH and hMeTiS can partition hypergraphs with weights at vertices and/or nets but in this work, vertices are given equal weight, i.e. weight one, and only the nets have weights that are equal to the number of nonzero entries in that row of the sparse matrix. As shown in Figure 1, the sparse matrix on the left will have the associated hypergraph on the right, in which we have, for example, hyperedge (or net) n_1, corresponding to row 1, with entries in columns 1, 3, and 4.

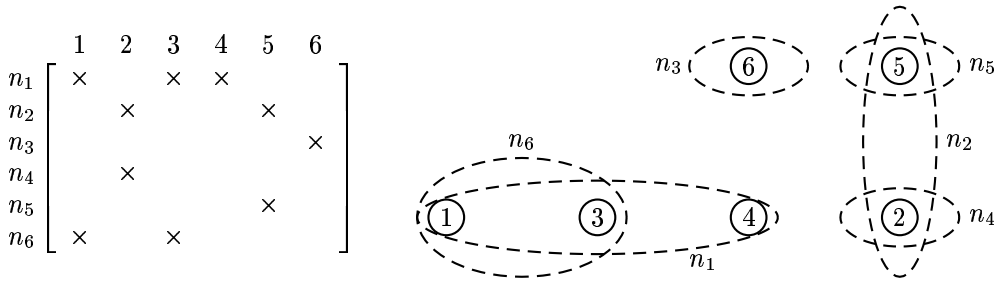


Figure 1: Sparse matrix and its hypergraph model.

By inspecting the hypergraph of this simple example, we can equally partition it into two parts and each has three vertices, as shown on the right of Figure 2. These two parts are independent from each other. Partitioning the hypergraph amounts to partitioning the columns of the corresponding matrix accordingly. So column 1, 3, and 4 are assigned the same partition number, for example 0, while columns 2, 5, and 6 are assigned number 1.

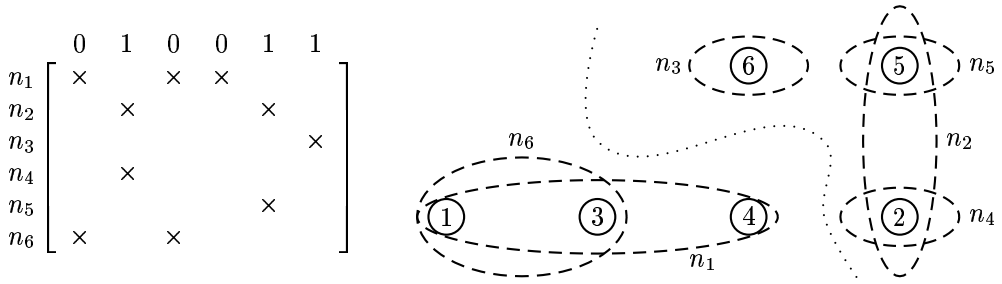


Figure 2: Partitioning of the hypergraph and its sparse matrix.

2.2 Mondriaan

Mondriaan is especially designed to partition sparse matrix-vector multiplication for parallel computing. It partitions both the matrix A and the vectors x and y. To minimize the communication volume

between processors and to distribute nonzero entries evenly over the processors it uses a two-dimensional partitioning, i.e. it partitions both the rows and the columns of the matrix. Mondriaan implements a recursive bipartitioning algorithm. From the user's point of view, it is easy to use to partition equation (1) because it only requires the entries of A to be stored in coordinate format and provides as output a partitioning of both the matrix and the vectors.

2.3 Monet

Monet is used to partition and reorder an unsymmetric matrix into singly bordered blocked diagonal (SBBD) form and is commonly used with direct methods [2]. It implements a multilevel recursive bisection algorithm and the Kernighan-Lin refinement method. The partitioned matrix can also be used in computing equation (1) for parallel matrix-vector multiplication.

After partitioning by PaToH, hMeTiS, Mondriaan, and Monet, we need to reorder rows and columns of the matrix to exhibit the block structures as described in the next section.

3 Permuting Partitioned Entries into Block Structures

Having partitioned a matrix we need to do row and column permutations in order to collect together the entries into blocks. This is convenient to facilitate counting data and message passing between processors and to perform (1) in parallel.

3.1 PaToH and hMeTiS

Both input and output files of PaToH and hMeTiS look almost the same and so can be described together. We use the matrix in Figure 1 as the illustrating example. The input is in the hypergraph format and the output file contains partition numbers that indicate to which processor the columns or vertices are assigned. These numbers will be used to compute the column permutation in the following way:

- read in partition numbers from output to $P(1)$, $P(2)$, ..., $P(n)$, where $P(i)$ is the processor number to which column i is assigned, as shown in Figure 3a.
- sort P in ascending order and permute the columns accordingly, see Figure 3b.

For row permutations, we have no explicit partition number, so we need to determine it at our convenience with the goal of moving most entries into blocks as described by the following steps:

- Let n_i be the number of columns assigned to processor i as shown in Figure 3b, we want to find n_i rows so that most nonzero entries of the matrix lie within the n_i columns in question. This makes the diagonal blocks square although the off-diagonal blocks maybe rectangular.
- We already know the n_i columns from Figure 2b and we now determine the n_i rows. We do this by sorting all the rows $j > n_1 + n_2 + \ldots + n_{i-1}$ so that the row with minimum number of entries lying within the n_i columns comes first and the row with maximum number of entries lying within the n_i columns comes last. The last n_i of the newly sorted rows are chosen to be the n_i rows in question. Finally, we obtain a block-structure matrix as shown in Figure 3c.

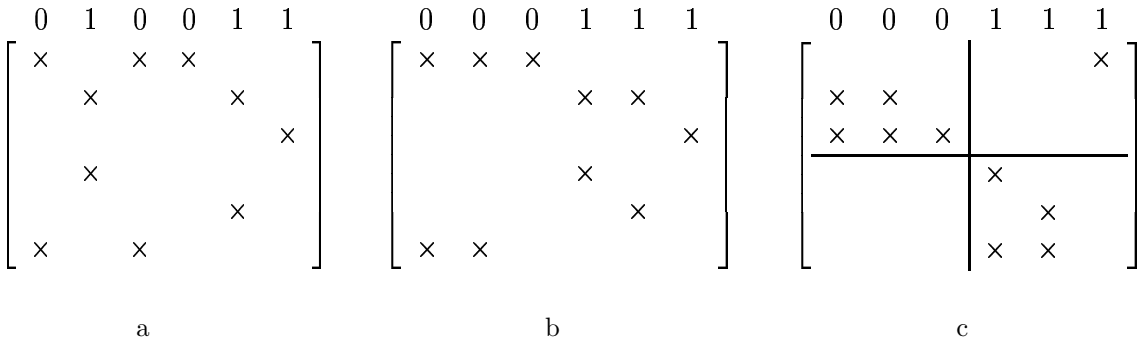


Figure 3. A block structure for the matrix of Figure 1 partitioned by PaToH and hMeTiS.

3.2 Mondriaan

Because the output of Mondriaan contains partition numbers of nonzero entries and input/output vectors, it is easy to form diagonal blocks:

- read in partition numbers of input vector to $P(1)$, $P(2)$, ..., $P(n)$ and output vector to $Q(1)$, $Q(2)$, ..., $Q(n)$, where $P(i)$ and $Q(i)$ are the processor numbers to which column i and row i, respectively, are assigned, as shown in Figure 4a.
- sort P and Q in ascending order and permute columns and rows of the matrix accordingly, giving the block structure shown in Figure 4b.

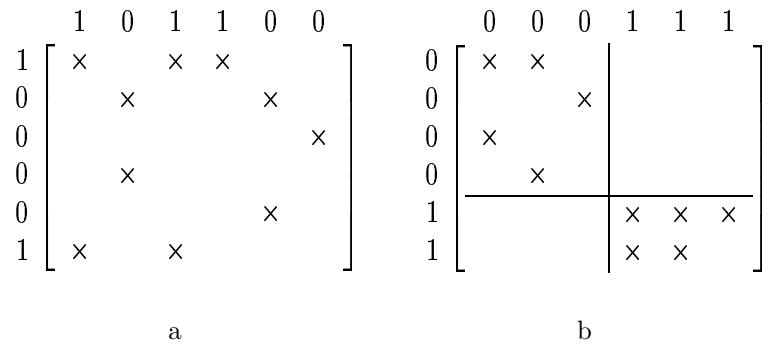


Figure 4. Block structure of the matrix partitioned by Mondriaan

3.3 Monet

The Monet output file contains ready-to-use column and row permutations, and the singly bordered blocked diagonal form can be directly obtained. In this example we have ordered-index set $\{1, 3, 4, 6, 2, 5\}$ for the column permutation and $\{1, 3, 6, 2, 4, 5\}$ for the row permutation; so after applying the permutations to the matrix in Figure 1, we obtain the final matrix as shown in Figure 5.

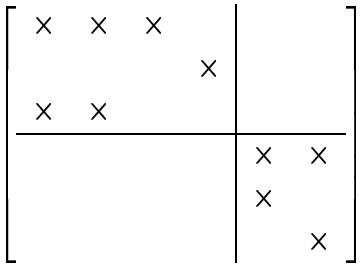


Figure 5. Block structure of the matrix partitioned by Monet

4 Experimental Results

The matrices used for testing are from the Rutherford-Boeing Collection [1]. These matrices are partitioned and then permuted into block structures. Then the message and data communications between processors are computed. Computing was done on a 168 MHz Sparc Ultra2 Sun machine with memory size 512 megabytes running SunOS.

In our experiments for hMeTiS and PaToH, we chose in the coarsening phase a matching scheme for grouping together graph vertices in such a way that hMeTiS uses the hybrid first-choice (HFC) scheme [9], where vertices are grouped together if they are present in multiple hyperedges and this grouping is biased towards a quick reduction in the number of hyperedges, and PaToH uses a heavy connectivity scheme [18], where a vertex, for example v, has maximum connectivity value equal to the number of hyperedges shared between it and another vertex. In the refinement phase, PaToH uses a tight balance boundary FM [18] while hMeTiS uses an early-exit FM scheme [9], i.e. the FM process is aborted if the quality of the

solution does not improve after moving a relatively small number of vertices. In addition, both PaToH and hMeTiS use V-cycle refinement in their partitionings. We use these parameters for partitioning all the matrices. In the case of Monet and Mondriaan, we use default parameters in all phases.

To analyse the experimental results, we model the communication cost as

$$t_c = \alpha + n\beta \tag{2}$$

where α and β are the latency and bandwidth, respectively, of the network and n is the length of messages exchanged. The number of messages and the amount of data exchanged in parallel computation must be small to reduce the latency cost and communication through bandwidth, respectively.

As shown in Figure 6, PaToH is the fastest partitioner for all the matrices and all the partition numbers although it uses the same hypergraph model for partitioning as hMeTiS which uses much more CPU time. If we look at matrices ex11.rb, olafu.rb, raefsky03.rb, and raefsky04.rb, we see that their CPU times differ by an order of magnitude. Mondriaan also uses as much CPU time as hMeTiS when partitioning while the CPU time of Monet is rather good.

The total number of received-sent messages by all processors is shown in Figure 7. For any processor, the number of received-sent messages amount to how many times it exchanges data with other processors. The maximum number of received-sent messages of one processor over the others is shown in Figure 8. The number of messages sent between processors directly influences the cost through the latency, i.e. the time needed to initialize the process of the communication through the network. The more messages that are sent or received, the higher the number of initializations, and consequently the communication cost is high. We find in the figures in the case of 2 partitions, matrices ex19.rb partitioned by PaToH, hMeTiS, and Monet and matrix ex35.rb partitioned by hMeTiS can exhibit completely block diagonal.

The total number of received-sent data by all processors is shown in Figure 9. This indicates the cost associated with the bandwidth of the network. For any bandwidth size, the more data is exchanged, the higher the communication cost. We notice in this figure that the total amount of received and sent data for matrices partitioned by Monet and hMeTiS is rather high while those partitioned by PaToH and Mondriaan are somewhat better.

These figures are complemented by tables in the appendix.

Figures 10 to 13 illustrate block structures of matrix ex35.rb when partitioned by PaToH, hMeTiS, Mondriaan, and Monet, respectively.

NOTE 1. From our experiments on running hMeTiS to partition sparse matrices we had different results when run more than once with the same parameters while PaToH with fixed random seed, Mondriaan, and Monet gave the same consistent results. For example, when used five times to partition matrix add20.rb into 8 partitions hMeTiS gave the results shown in Table 1.

hMeTiS		
total #msg	total #data	max #msg
80	2908	13
90	2470	13
78	2397	13
84	2540	13
78	2638	13

Table 1: Five runs of hMeTiS on matrix add20.rb

NOTE 2. From Figures 6 to 9 we find that matrix raefsky03.rb is problematic both in term of CPU time and message exchange when it was partitioned by partitioners with the parameters mentioned above. We thus partitioned it with hMeTiS and changed schemes for coarsening, refinement, and uncoarsening. We found that the refinement schemes have the greatest effect on the partitioning results. Anyway, this scheme dominates only for small numbers of partitions. When the number of partitioning increases, all schemes give very similar results.

5 Conclusion

From the experiments, we see that PaToH is the fastest partitioner. The quality of the partitioning by PaToH in terms of message and data exchange between processors is still good when compared to the other three partitioners. Monet is quite fast but its data exchange is high, which makes it inferior to PaToH. For the other two, they are rather slow partitioners and mostly give high message and data exchange.

Acknowledgement The author thanks Dr. Luc Giraud and Prof. Iain Duff for their helpful suggestions and comments.

References

[1] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report TR/PA/97/36, CERFACS, Toulouse, France, 1997. Also Technical Report RAL-TR-97-031 from Rutherford Appleton Laboratory and Technical Report ISSTECH-97-017 from Boeing Information & Support Services.

[2] I. S. Duff and J. A. Scott. A parallel direct solver for large sparse highly unsymmetric linear system. Technical Report RAL-TR-2002-033, 2003.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partition. *Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[4] Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. Preprint, 1999.

[5] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. will appear in proc. supercomputing '95, 1995.

[6] Y. F. Hu, K. C. F. Maguire, and R. J. Blake. A multilevel unsymmetric matrix ordering algorithm for parallel process simulation. *Computers and Chemical Engineering*, 23, 2000.

[7] George Karypis. Multilevel hypergraph partitioning. Technical Report, University of Minnesota.

[8] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. Technical Report, University of Minnesota, 1998.

[9] George Karypis and Vipin Kumar. hmetis: A hypergraph partitioning package version 1.5.3. Technical report, 1998.

[10] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, University of Minnesota, 1998.

[11] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, 1970.

[12] Alex Pothen. Graph partitioning algorithms with applications to scientific computing. In David E. Keyes, Ahmed Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*, pages 323–368. Kluwer Academic Publishers, the Netherlands, 1997.

[13] Kirk Schloegel, George Karypis, and Vipin Kumar. Graph partitioning for high performance scientific simulations. In J. Dongarra, I. Foster, G. Fox, K. Kennedy, and A. White, editors, *CRPC Parallel Computing Handbook*. USA, 2000.

[14] Ümit V. Çatalyürek and Cevdet Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. Technical report.

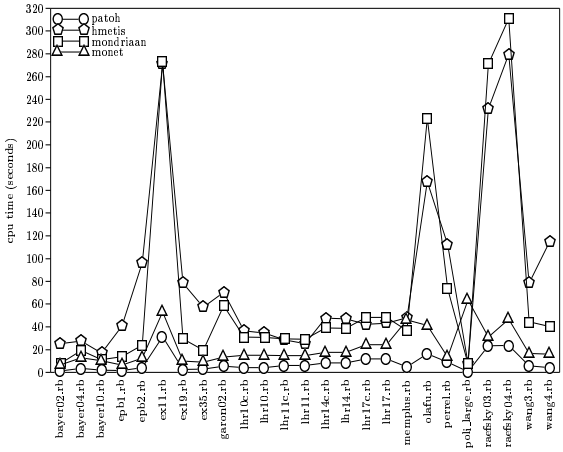
[15] Ümit V. Çatalyürek and Cevdet Aykanat. A fine-grain hypergraphs model for 2d decomposition of sparse matrices. Technical report.

[16] Ümit V. Çatalyürek and Cevdet Aykanat. Hypergraph model for mapping repeated sparse matrix-vector product computations onto multicomputers. Technical report.

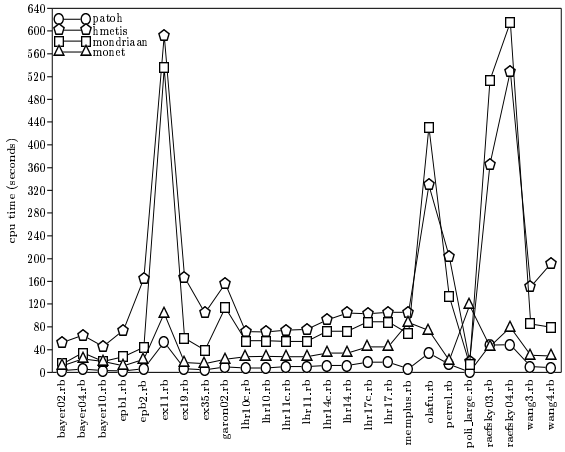
[17] Ümit V. Çatalyürek and Cevdet Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transaction on Parallel and Distributioned Systems*, 10:673–693, 1999.

[18] Ümit V. Çatalyürek and Cevdet Aykanat. Patoh: Patitioning tools for hypergraphs. Technical report, 2002.

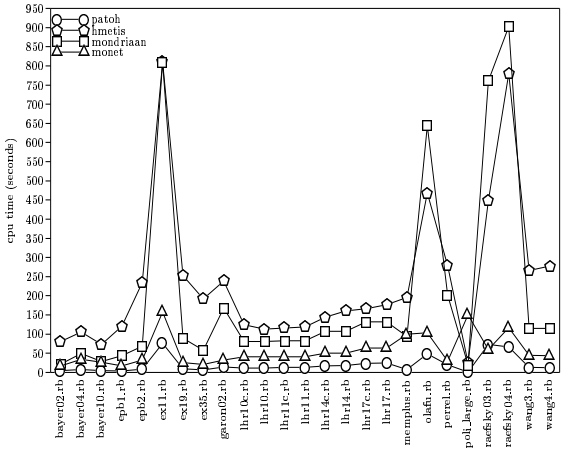
[19] Brendan Vastenhouw and Rob H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. preprint 1238, 2002.



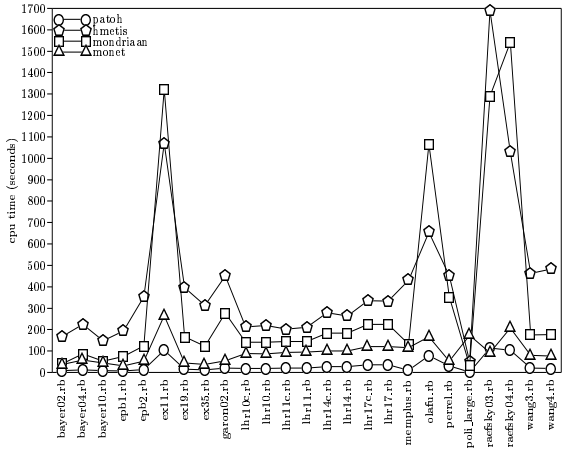
2 partitions



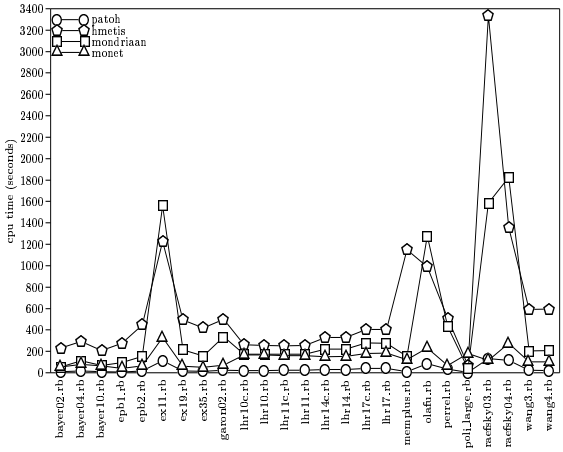
4 partitions



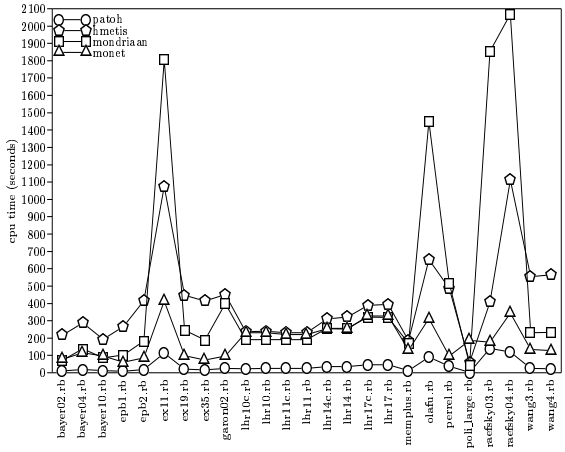
8 partitions



32 partitions

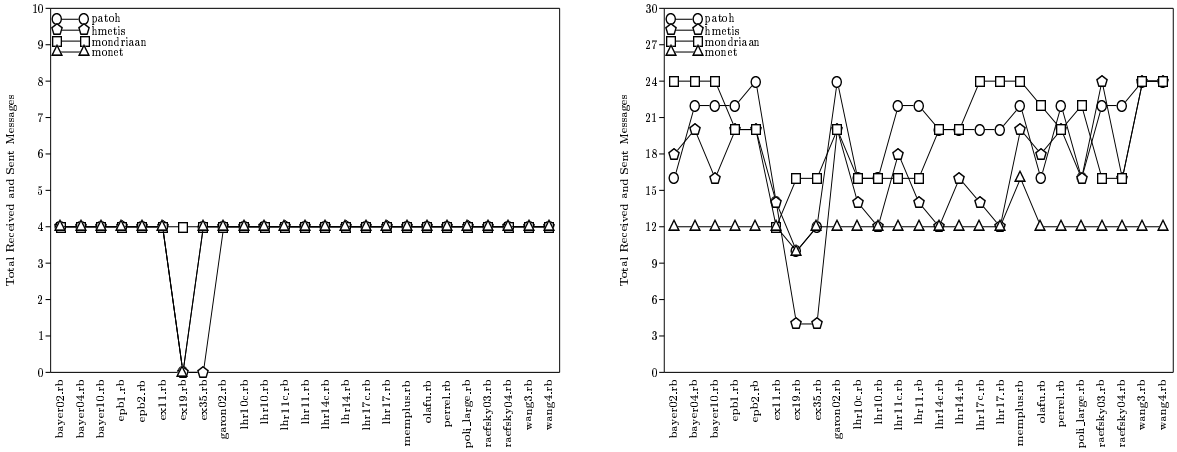


64 partitions



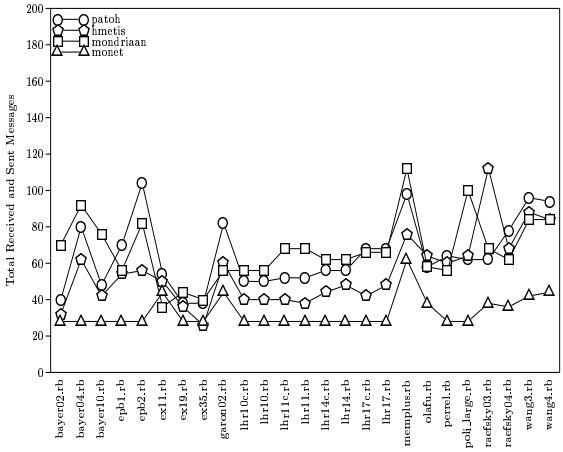
128 partitions

Figure 6: CPU times for different numbers of partitions.

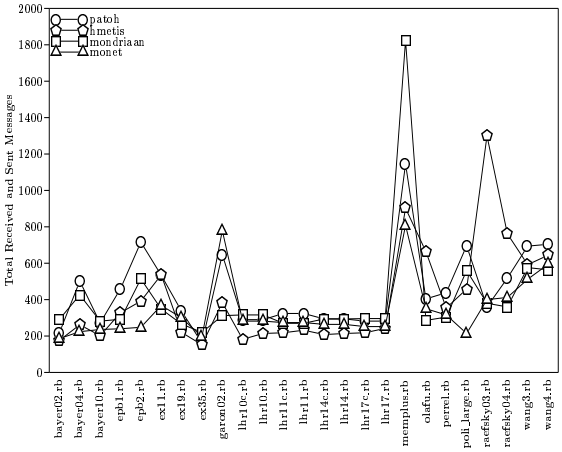


2 partitions

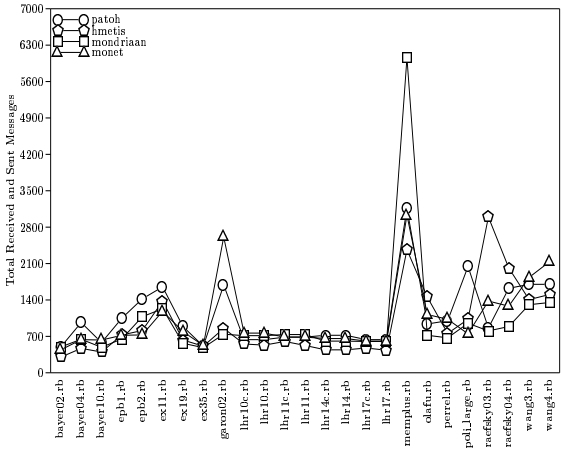
4 partitions



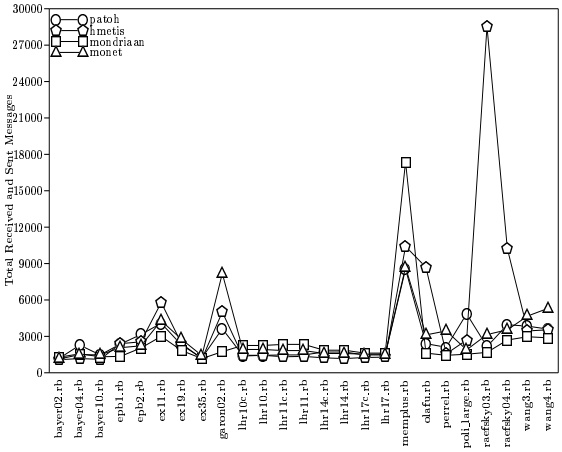
8 partitions



32 partitions

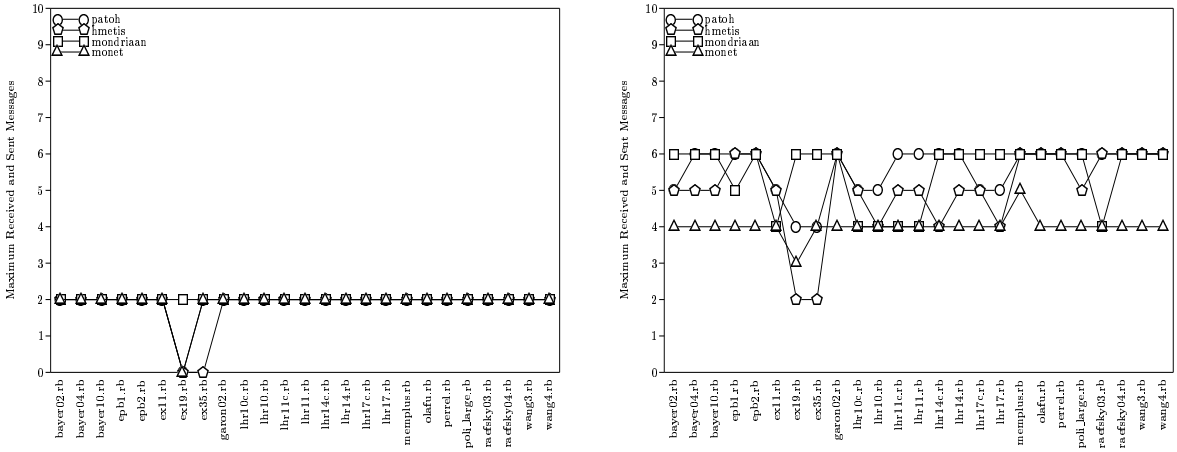


64 partitions



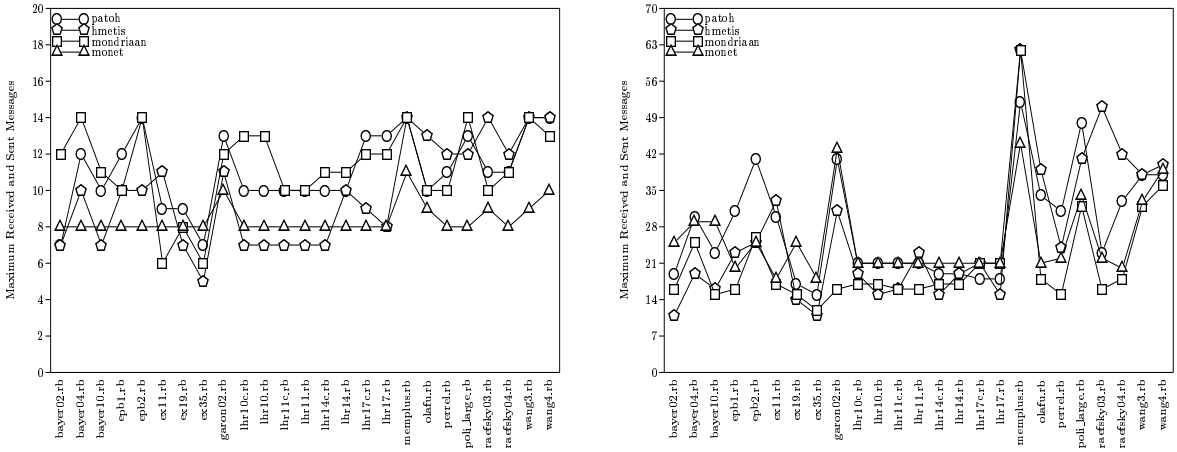
128 partitions

Figure 7: Total number of received and sent messages for different numbers of partitions.



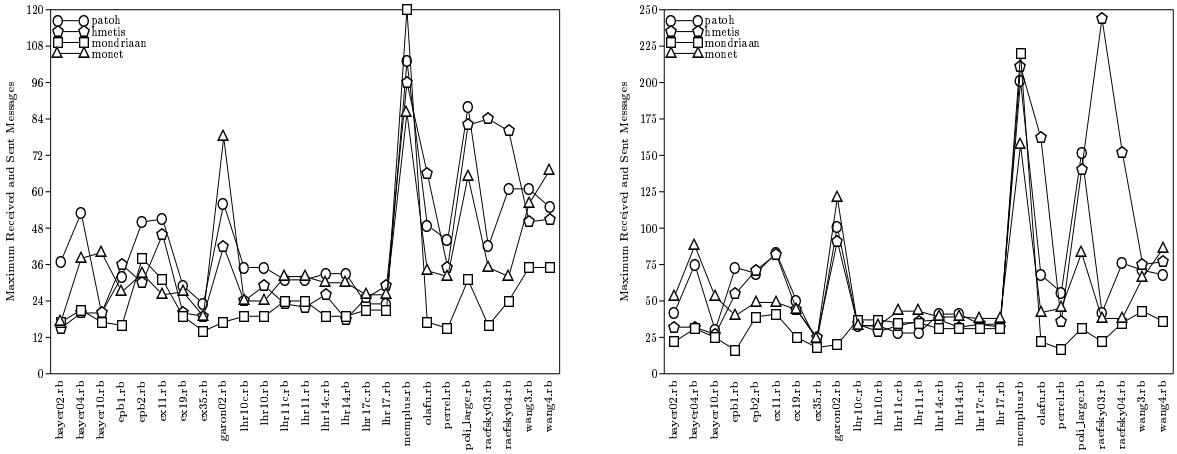
2 partitions

4 partitions



8 partitions

32 partitions



64 partitions

128 partitions

Figure 8: Maximum number of received and sent messages for different numbers of partitions.

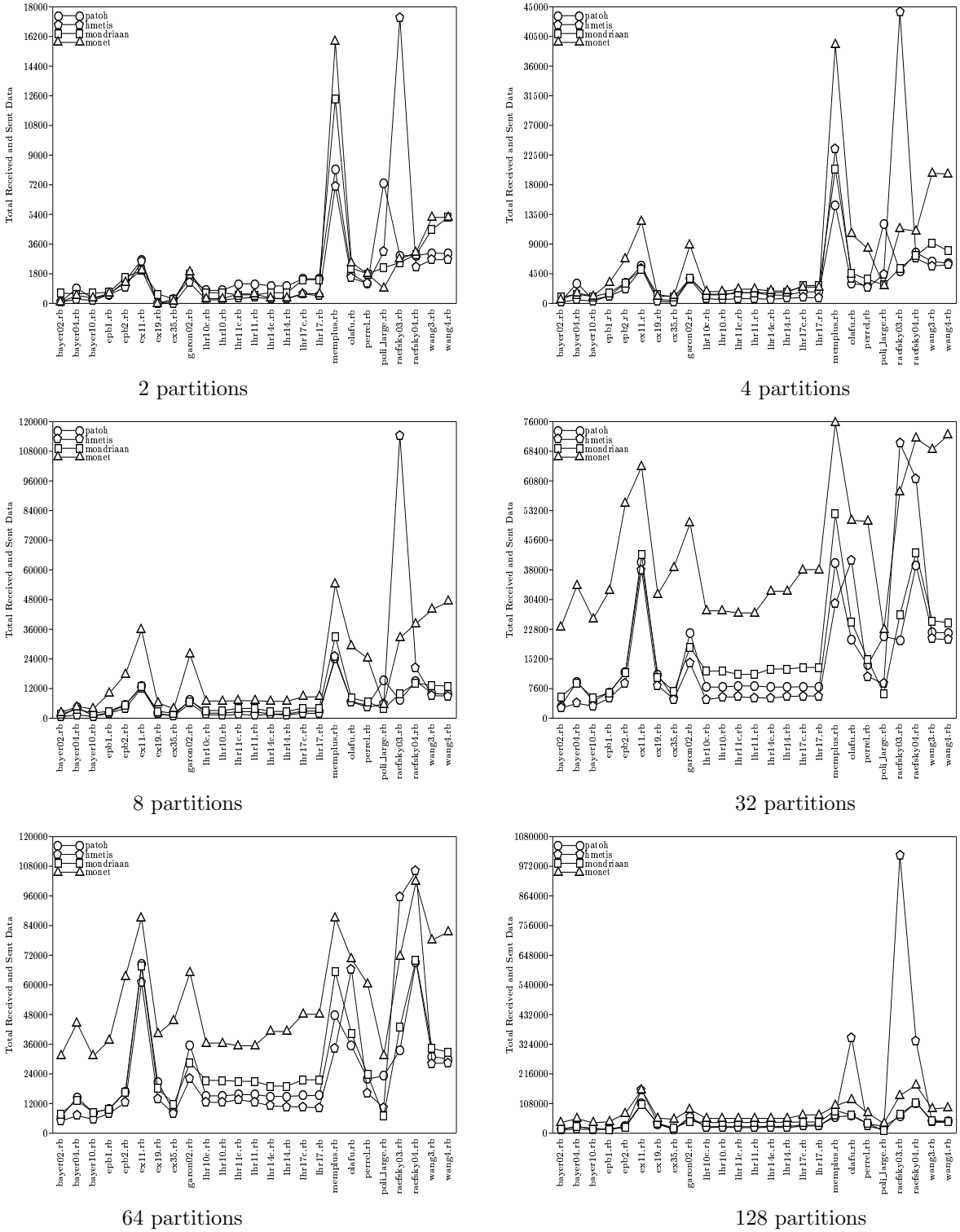
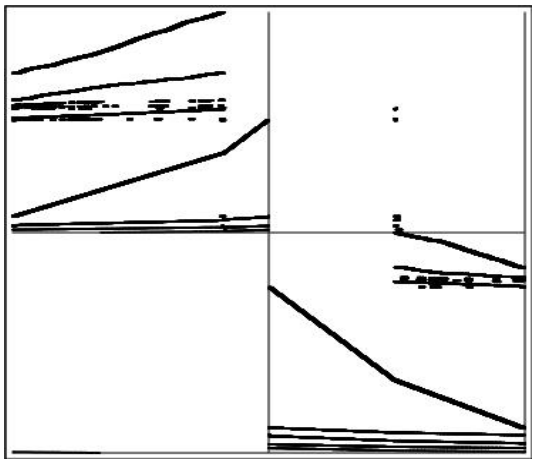
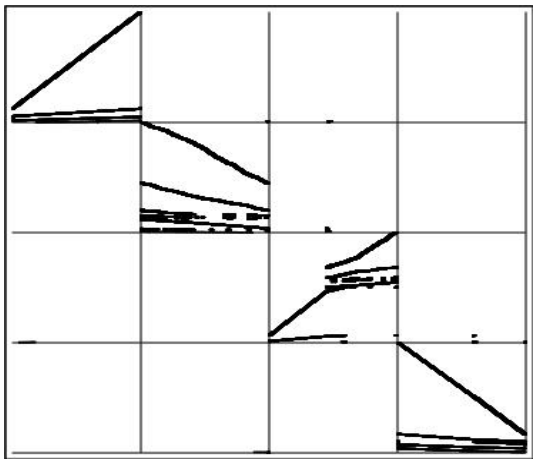


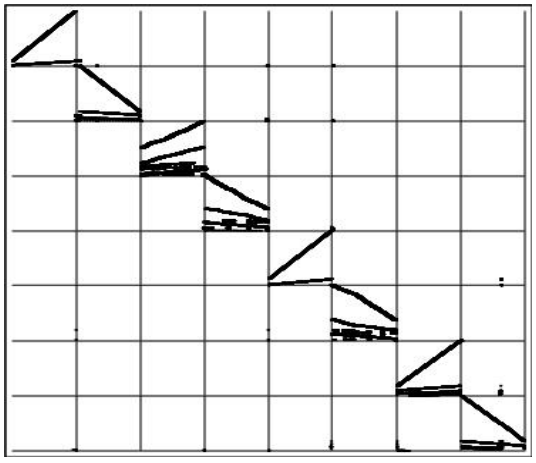
Figure 9: Total amount of received and sent data for different numbers of partitions.



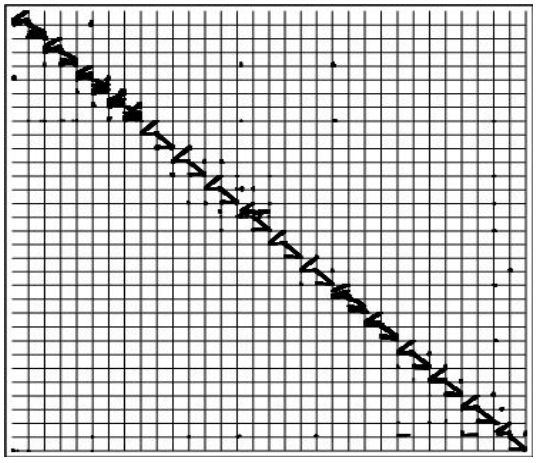
2 partitions



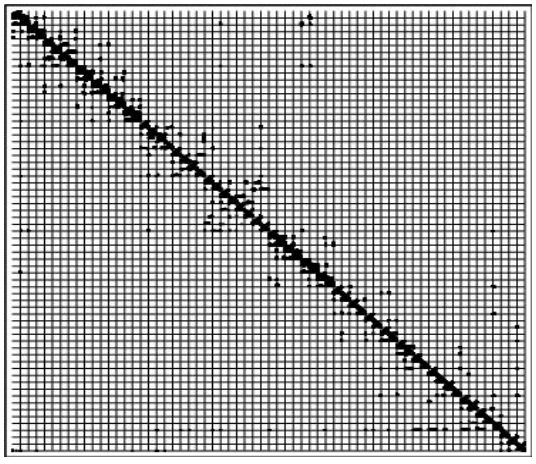
4 partitions



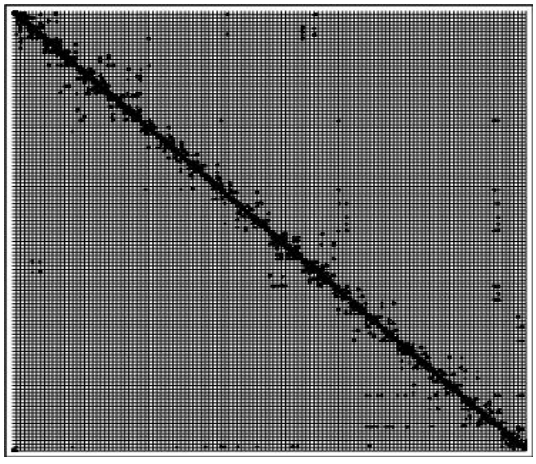
8 partitions



32 partitions

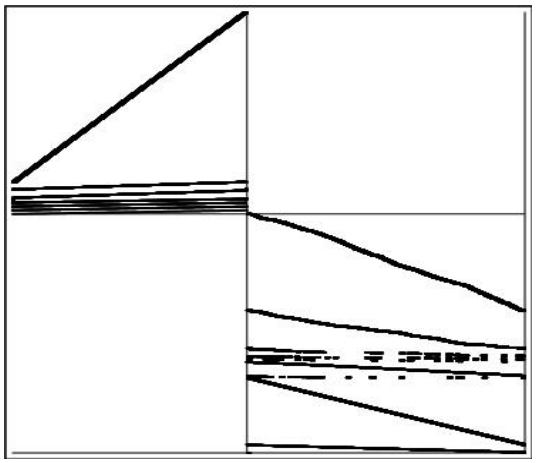


64 partitions

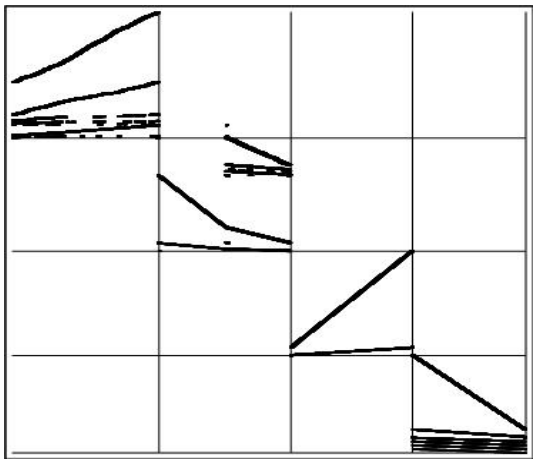


128 partitions

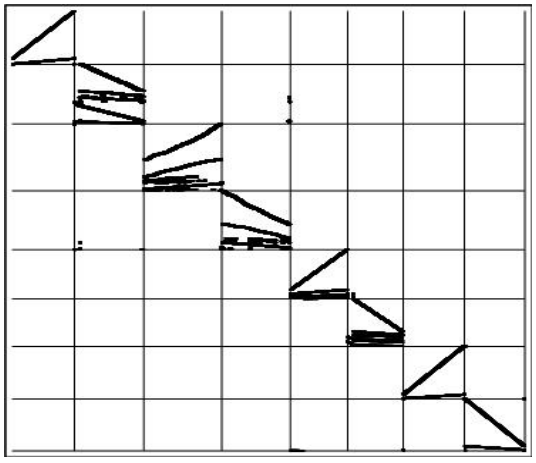
Figure 10: matrix ex35.rb partitioned by PaToH for different numbers of partitions.



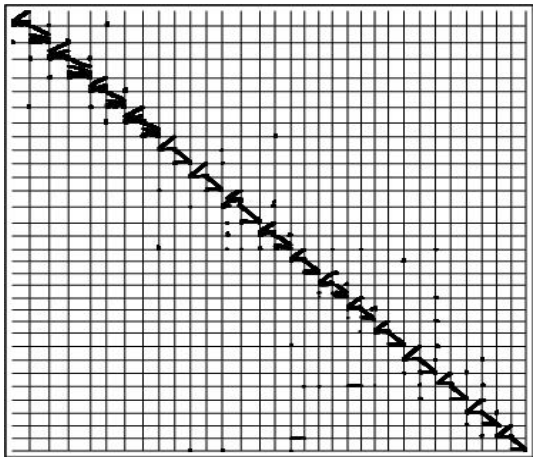
2 partitions



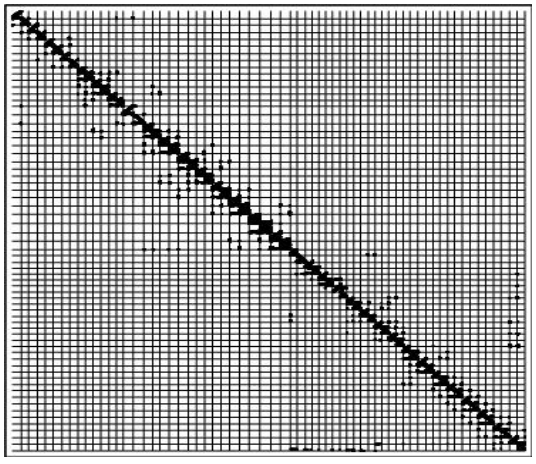
4 partitions



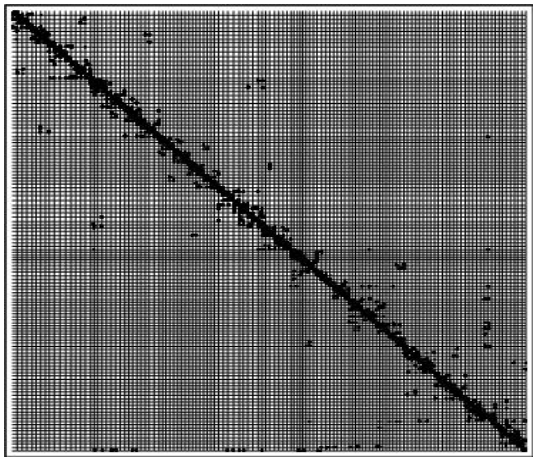
8 partitions



32 partitions

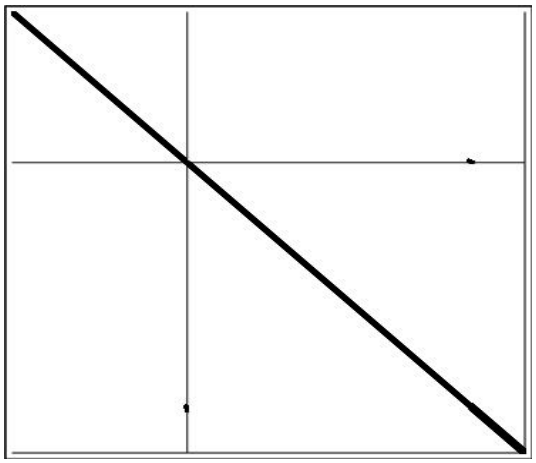


64 partitions

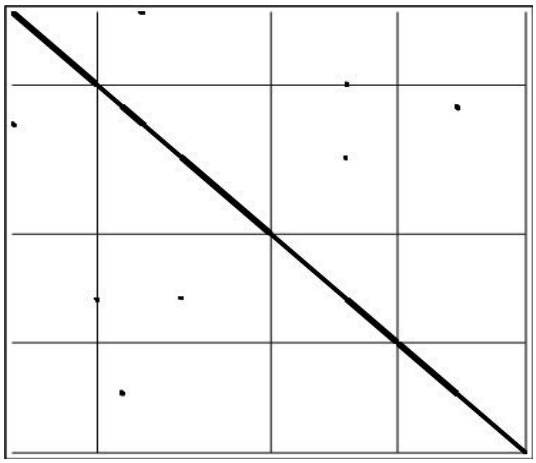


128 partitions

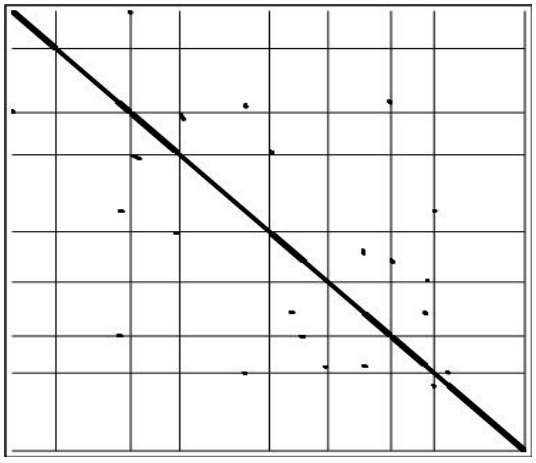
Figure 11: matrix ex35.rb partitioned by hMeTiS for different numbers of partitions.



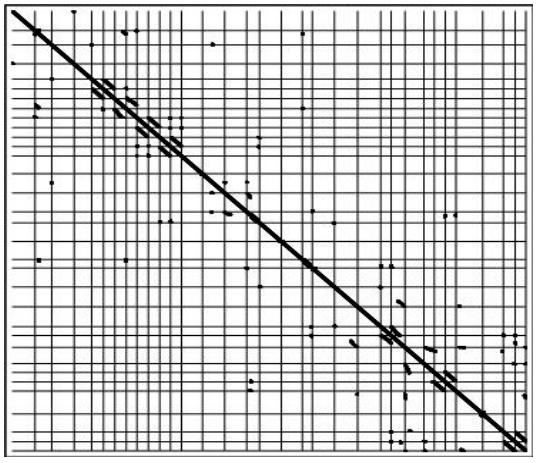
2 partitions



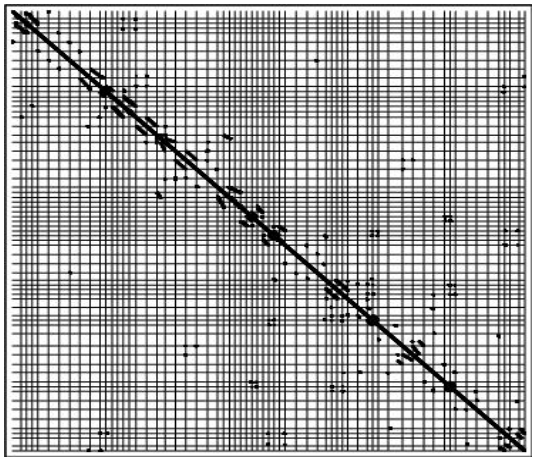
4 partitions



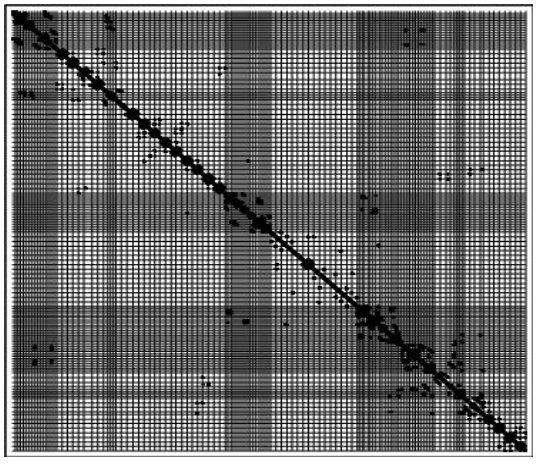
8 partitions



32 partitions

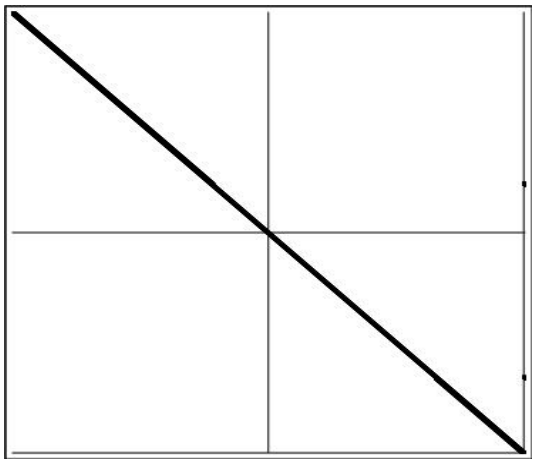


64 partitions

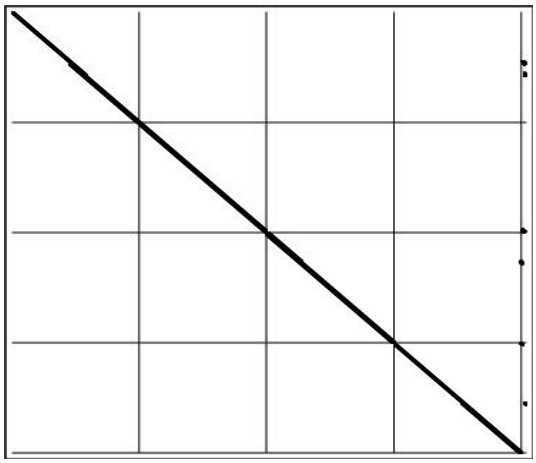


128 partitions

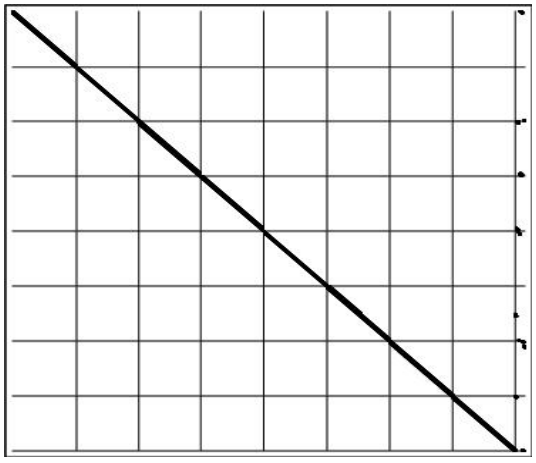
Figure 12: matrix ex35.rb partitioned by Mondriaan for different numbers of partitions.



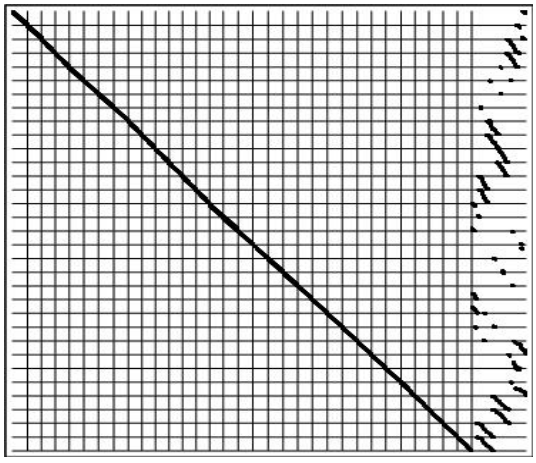
2 partitions



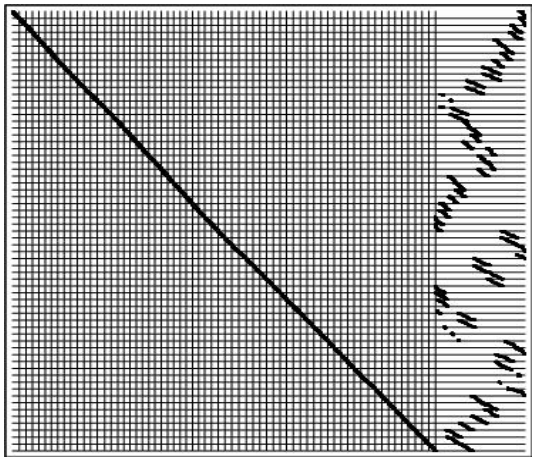
4 partitions



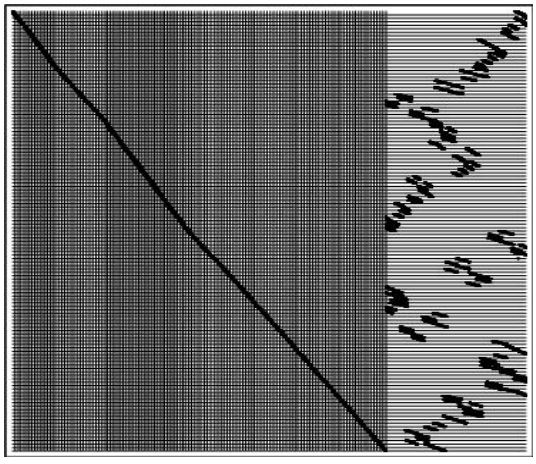
8 partitions



32 partitions



64 partitions



128 partitions

Figure 13: matrix ex35.rb partitioned by Monet for different numbers of partitions.

Appendix

		PaToH				hMeTiS				Mondriaan				Monet			
matrix	K	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)
bayer02.rb	2	4	106	2	1.41	4	96	2	25.14	4	632	2	8.08	4	88	2	6.85
	4	16	293	5	3.05	18	214	5	52.57	24	957	6	14.92	12	526	4	11.76
	8	40	721	7	4.53	32	552	7	80.96	70	1738	12	22.23	28	2230	8	16.76
	16	106	1480	14	5.72	80	1197	9	117.19	140	3235	13	32.17	60	8736	16	23.62
	32	218	3082	19	7.67	176	2554	11	168.25	292	5502	16	41.44	186	23346	25	36.63
	64	498	6337	37	9.36	300	4944	15	232.77	498	7835	17	55.94	444	31346	17	55.19
	128	1190	12830	42	10.58	1074	11854	32	216.48	1288	15483	22	70.93	1138	39080	53	77.71
bayer04.rb	2	4	911	2	3.63	4	245	2	27.60	4	569	2	19.12	4	490	2	12.79
	4	22	3049	6	5.83	20	553	5	65.26	24	1376	6	34.83	12	1578	4	23.94
	8	80	4483	12	7.97	62	1338	10	105.83	92	3628	14	49.62	28	4694	8	34.43
	16	200	7431	20	9.82	128	2122	13	176.78	246	5115	21	67.04	60	14082	16	44.25
	32	502	9382	30	12.05	262	3792	19	224.18	426	9057	25	86.01	224	33990	29	57.42
	64	968	14535	53	15.35	468	7126	20	291.10	642	13350	21	112.52	634	44554	38	79.51
	128	2272	25266	75	18.23	1164	14758	32	287.91	1568	23364	31	138.79	1506	52928	88	114.91
bayer10.rb	2	4	287	2	1.97	4	184	2	17.04	4	608	2	11.06	4	324	2	10.31
	4	22	603	6	3.26	16	354	5	45.64	24	1052	6	19.49	12	1138	4	19.11
	8	48	901	10	5.16	42	700	7	72.91	76	2020	11	28.37	28	3762	8	26.51
	16	106	2038	15	6.43	86	1571	9	109.26	152	3817	15	39.69	60	11344	16	33.41
	32	272	4413	23	8.79	202	3119	16	148.52	282	5155	15	52.25	236	25492	29	44.73
	64	578	8359	20	10.66	396	5589	20	208.42	490	8354	17	68.09	630	31214	40	65.44
	128	1480	15400	30	12.63	1110	12799	27	194.23	1306	17550	25	85.67	1484	37426	53	97.19
epb1.rb	2	4	545	2	1.57	4	542	2	40.60	4	677	2	13.91	4	620	2	6.32
	4	22	1145	6	2.69	20	1119	6	73.92	20	1607	5	27.55	12	3138	4	11.48
	8	70	2265	12	3.95	54	1996	10	120.49	56	2826	10	43.93	28	10128	8	16.73
	16	182	3934	18	5.22	162	3324	14	160.54	132	4483	12	60.16	80	24710	13	22.91
	32	460	6613	31	6.59	328	5137	23	194.36	292	6552	16	74.50	240	32574	20	30.77
	64	1052	9812	32	7.91	726	7983	36	274.44	648	9686	16	94.15	720	37544	27	42.44
	128	2374	15336	73	9.06	2404	14687	55	267.57	1346	13802	16	102.64	2060	42432	40	60.45
epb2.rb	2	4	1394	2	4.03	4	979	2	96.29	4	1587	2	23.88	4	1266	2	12.36
	4	24	3124	6	6.26	20	2158	6	164.94	20	3099	6	43.89	12	6752	4	23.16
	8	104	5419	14	8.52	56	3958	10	234.77	82	5541	14	68.17	28	17710	8	32.12
	16	248	8171	25	10.74	182	5618	16	278.44	202	7974	20	95.24	82	41204	17	41.63
	32	716	11784	41	13.18	392	8949	25	352.86	514	11718	26	124.89	248	55140	25	52.25
	64	1420	16963	50	15.54	814	12490	30	444.03	1076	16673	38	153.66	726	63346	33	66.07
	128	3160	24580	69	18.00	2560	23126	71	416.20	2040	22072	39	183.34	2226	72326	49	85.77
ex11.rb	2	4	2641	2	31.23	4	2542	2	271.36	4	2022	2	273.09	4	2022	2	53.24
	4	14	5771	5	53.06	14	5254	5	592.39	12	5234	4	535.30	12	12414	4	103.91
	8	54	12966	9	75.59	50	12228	11	810.16	36	12847	6	809.14	44	35704	8	157.96
	16	150	22731	15	96.93	154	20230	17	976.20	102	22928	10	1074.99	126	47224	11	209.78
	32	534	39950	30	104.53	536	38018	33	1069.97	348	41883	17	1322.40	368	64466	18	265.18
	64	1654	68463	51	113.71	1382	61093	46	1226.24	1230	67600	31	1561.89	1182	87144	26	329.65
	128	4080	109485	83	113.89	5780	157229	82	1075.60	3014	105767	41	1805.73	4304	157438	49	414.55
ex19.rb	2	0	0	0	2.46	0	0	0	78.53	4	570	2	29.40	0	0	0	9.95
	4	10	601	4	6.55	4	327	2	166.84	16	1362	6	60.08	10	1050	3	17.06
	8	38	1824	9	9.40	36	1844	7	252.87	44	2808	8	89.74	28	6212	8	25.36
	16	110	5204	14	12.55	66	3672	8	329.27	84	6176	8	125.73	102	23756	17	34.44
	32	336	11223	17	15.64	220	8331	14	396.53	266	10496	15	167.01	300	31800	25	45.05
	64	890	20729	29	18.51	630	13937	20	492.97	568	18279	19	215.54	788	40158	27	61.97
	128	2266	33325	50	20.48	2404	36725	44	447.33	1816	35730	25	246.66	2748	55160	44	99.50

		PaToH				hMeTiS				Mondriaan				Monet			
matrix	K	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)
ex35.rb	2	4	276	2	2.75	0	0	0	57.25	4	264	2	19.51	4	120	2	8.89
	4	12	438	4	4.60	4	176	2	104.17	16	960	6	38.88	12	1160	4	15.13
	8	38	1046	7	6.89	26	849	5	191.76	40	2075	6	57.69	28	3798	8	21.30
	16	80	2666	10	8.89	54	1971	7	251.23	60	3349	8	86.82	60	17340	16	28.58
	32	162	5099	15	11.74	152	4828	11	312.83	220	6952	12	121.72	196	38678	18	37.49
	64	514	8666	23	13.74	512	7829	19	418.06	486	11754	14	155.94	522	45468	19	50.54
	128	1218	14557	22	16.27	1152	13306	25	413.06	1174	18934	18	189.04	1404	51236	24	73.47
garon02.rb	2	4	1528	2	5.32	4	1243	2	70.35	4	1774	2	58.50	4	1924	2	13.52
	4	24	3828	6	9.32	20	3642	6	156.04	20	3838	6	113.63	12	8806	4	22.90
	8	82	7445	13	13.80	60	5955	11	238.80	56	6796	12	168.33	44	25854	10	32.76
	16	304	14630	27	17.06	158	8823	19	328.40	126	11400	16	222.95	192	38144	21	41.86
	32	648	21873	41	19.87	384	14050	31	450.68	312	18139	16	276.58	780	49898	43	54.43
	64	1688	35496	56	22.70	856	21936	42	499.88	742	28584	17	335.93	2612	65096	78	71.54
	128	3622	54415	101	25.38	5002	60285	91	452.20	1746	43504	20	399.12	8192	86912	121	96.84
lhr10c.rb	2	4	793	2	4.11	4	222	2	36.41	4	642	2	31.02	4	286	2	15.17
	4	16	1382	5	7.79	14	732	5	71.87	16	1307	4	55.94	12	1806	4	27.83
	8	50	2003	10	11.11	40	1456	7	124.90	56	3159	13	80.67	28	6976	8	40.95
	16	126	3676	15	14.38	100	2608	12	174.21	126	6544	15	107.33	96	20082	15	60.52
	32	290	7991	21	18.60	182	4740	19	213.26	316	12101	17	141.95	282	27532	21	87.55
	64	638	15120	35	21.86	552	12524	24	260.45	716	21343	19	171.79	764	36502	24	165.89
	128	1444	24043	33	24.00	1374	21123	34	237.32	2256	38835	37	191.24	1928	53498	33	229.98
lhr10.rb	2	4	793	2	4.16	4	222	2	34.48	4	642	2	30.83	4	286	2	15.05
	4	16	1382	5	7.75	12	594	4	70.75	16	1307	4	55.83	12	1806	4	27.94
	8	50	2003	10	11.13	40	1246	7	113.46	56	3159	13	80.79	28	6976	8	41.15
	16	126	3676	15	14.39	86	2712	11	175.91	126	6544	15	107.52	96	20082	15	60.28
	32	290	7991	21	18.64	214	5388	15	218.64	316	12101	17	141.60	282	27532	21	87.20
	64	638	15120	35	22.02	534	12507	29	253.98	716	21343	19	172.10	764	36502	24	165.53
	128	1444	24043	33	24.33	1432	21857	29	238.93	2256	38835	37	191.05	1928	53498	33	229.98
lhr11c.rb	2	4	1186	2	6.20	4	297	2	28.43	4	518	2	29.35	4	574	2	14.76
	4	22	1621	6	10.20	18	735	5	74.05	16	1613	4	54.32	12	2174	4	27.69
	8	52	2689	10	13.03	40	1508	7	116.02	68	3981	10	82.42	28	7050	8	40.54
	16	134	3779	15	16.79	92	2471	12	169.44	184	8450	18	109.36	88	18602	14	56.09
	32	324	8258	21	20.19	218	5468	16	202.56	270	11249	16	143.81	274	26954	21	94.22
	64	684	15589	31	24.04	600	13541	23	250.03	736	21024	24	173.23	686	35448	32	158.46
	128	1462	24018	28	25.91	1332	20386	33	231.20	2338	39038	35	191.42	1828	53854	43	219.51
lhr11.rb	2	4	1186	2	6.16	4	365	2	25.39	4	518	2	29.19	4	574	2	14.61
	4	22	1621	6	10.08	14	727	5	75.20	16	1613	4	54.30	12	2174	4	27.71
	8	52	2689	10	13.07	38	1242	7	118.87	68	3981	10	82.53	28	7050	8	40.65
	16	134	3779	15	16.66	90	2674	9	164.45	184	8450	18	109.81	88	18602	14	55.90
	32	324	8258	21	20.13	232	5215	23	211.50	270	11249	16	143.86	274	26954	21	94.40
	64	684	15589	31	23.75	528	12563	22	253.43	736	21024	24	173.00	686	35448	32	158.86
	128	1462	24018	28	25.72	1342	20706	36	231.36	2338	39038	35	191.35	1828	53854	43	219.64
lhr14c.rb	2	4	1062	2	8.16	4	288	2	47.75	4	648	2	38.93	4	298	2	17.61
	4	20	1252	6	12.33	12	629	4	92.36	20	1644	6	72.16	12	1870	4	34.06
	8	56	1686	10	17.53	44	1580	7	144.59	62	2813	11	107.55	28	6898	8	50.74
	16	124	4387	13	21.69	94	2787	11	220.29	148	7396	14	146.17	80	21434	16	70.51
	32	294	7965	19	27.22	208	5150	15	278.92	294	12588	17	183.18	264	32532	21	100.18
	64	724	14916	33	30.77	444	10998	26	329.83	606	19100	19	222.16	660	41176	30	151.25
	128	1704	24418	41	35.29	1236	21108	37	311.65	1870	39924	31	256.74	1594	53818	39	253.11

		PaToH				hMeTiS				Mondriaan				Monet			
matrix	K	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)
lhr14.rb	2	4	1062	2	8.24	4	288	2	47.05	4	648	2	38.73	4	298	2	17.60
	4	20	1252	6	12.12	16	758	5	104.46	20	1644	6	72.01	12	1870	4	33.87
	8	56	1686	10	17.66	48	1337	10	160.42	62	2813	11	107.76	28	6898	8	50.38
	16	124	4387	13	21.62	106	2975	12	230.42	148	7396	14	145.75	80	21434	16	70.69
	32	294	7965	19	27.42	214	5580	19	263.74	294	12588	17	183.62	264	32532	21	100.07
	64	724	14916	33	31.19	444	10819	18	329.56	606	19100	19	222.38	660	41176	30	151.38
	128	1704	24418	41	35.42	1178	20621	32	321.84	1870	39924	31	256.43	1594	53818	39	252.79
lhr17c.rb	2	4	1486	2	11.73	4	568	2	42.31	4	1448	2	48.29	4	594	2	24.25
	4	20	1750	5	17.86	14	920	5	102.90	24	2711	6	88.91	12	2454	4	45.02
	8	68	2878	13	24.17	42	2047	9	166.63	66	4042	12	131.73	28	8630	8	64.11
	16	150	4885	15	29.51	106	3400	12	250.03	172	8193	15	174.00	84	26108	17	87.32
	32	282	8062	18	35.07	218	5695	21	334.88	296	13024	21	224.29	252	38072	21	120.64
	64	634	15276	23	40.10	478	10525	25	404.69	600	21615	21	278.35	608	48102	26	181.74
	128	1456	29433	34	45.47	1268	26041	34	388.37	1622	40955	31	320.64	1538	66640	38	328.52
lhr17.rb	2	4	1486	2	11.79	4	475	2	43.30	4	1448	2	48.23	4	594	2	24.22
	4	20	1750	5	17.86	12	858	4	105.33	24	2711	6	88.73	12	2454	4	44.89
	8	68	2878	13	24.26	48	1970	8	176.60	66	4042	12	131.72	28	8630	8	64.07
	16	150	4885	15	29.58	110	2932	13	261.35	172	8193	15	173.59	84	26108	17	86.91
	32	282	8062	18	35.22	242	5667	15	333.73	296	13024	21	224.43	252	38072	21	120.58
	64	634	15276	23	39.67	432	10279	29	405.48	600	21615	21	276.85	608	48102	26	182.83
	128	1456	29433	34	45.55	1290	26551	32	395.30	1622	40955	31	319.83	1538	66640	38	328.56
memplus.rb	2	4	8112	2	4.92	4	7130	2	47.89	4	12376	2	36.88	4	15904	2	46.07
	4	22	14904	6	6.16	20	23555	6	105.28	24	20440	6	68.08	16	39300	5	87.58
	8	98	24094	14	7.38	76	25072	14	195.88	112	33151	14	94.09	62	54084	11	99.87
	16	360	30787	27	8.66	352	29296	30	241.91	470	46241	30	116.58	226	65186	22	108.99
	32	1146	39775	52	9.64	902	29309	62	430.10	1824	52397	62	132.63	808	75728	44	115.98
	64	3164	47585	103	10.83	2368	34450	96	1150.45	6054	65278	120	152.88	3034	86942	86	122.79
	128	8536	59691	201	11.92	10412	65625	211	184.68	17382	83567	220	171.13	8702	99004	157	129.58
olafu.rb	2	4	1788	2	16.41	4	1554	2	167.78	4	2136	2	223.39	4	2460	2	41.26
	4	16	3041	6	34.16	18	3720	6	328.65	22	4580	6	431.38	12	10548	4	73.32
	8	58	6742	10	48.73	64	6542	13	468.04	58	8113	10	644.42	38	29106	9	103.51
	16	154	12035	20	64.03	196	16140	21	537.75	132	15094	15	874.78	124	40206	13	133.41
	32	404	20153	34	78.18	664	40349	39	654.60	286	24711	18	1065.62	350	50712	21	165.85
	64	940	35452	49	83.77	1454	66363	66	990.18	720	40400	17	1278.87	1122	70598	34	232.35
	128	2394	63984	68	91.23	8694	348779	162	652.78	1630	66254	22	1449.94	3122	120604	42	313.58
perrel.rb	2	4	1200	2	9.04	4	1200	2	112.39	4	1800	2	73.71	4	1800	2	13.34
	4	22	2646	6	13.85	20	2365	6	203.63	20	3625	6	133.74	12	8346	4	21.07
	8	64	5185	11	20.18	60	4415	12	277.98	56	6645	10	201.45	28	24218	8	29.44
	16	172	8704	19	24.41	158	7055	16	371.75	138	10753	14	277.48	104	41994	15	39.56
	32	438	13614	31	30.72	354	10720	24	451.94	304	15103	15	353.09	316	50484	22	52.06
	64	1000	21952	44	34.17	754	16230	35	510.56	672	23792	15	430.68	1030	60446	32	69.64
	128	2104	34336	56	37.86	1574	25695	36	487.64	1454	36122	17	516.97	3464	72952	45	97.60
poli_large.rb	2	4	7310	2	0.50	4	3136	2	8.09	4	2158	2	7.52	4	924	2	63.75
	4	16	12038	6	0.72	16	4403	5	19.28	22	2837	6	13.81	12	2718	4	117.65
	8	62	15465	13	1.03	64	5762	12	25.66	100	3890	14	20.13	28	5570	8	149.80
	16	218	18565	25	1.29	160	7331	20	35.43	284	4768	25	24.79	60	11770	16	160.83
	32	696	20988	48	1.60	458	8834	41	51.38	560	6333	32	32.07	214	22680	34	172.60
	64	2044	23168	88	2.06	1030	10355	82	69.41	952	7020	31	38.63	748	31436	65	180.56
	128	4860	24829	152	2.58	2694	12655	140	61.35	1532	8381	31	46.03	1890	35944	83	186.52

		PaToH				hMeTiS				Mondriaan				Monet			
matrix	K	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)	total #msg	total #data	max #msg	CPU time(sec)
raefsky03.rb	2	4	2896	2	23.44	4	17339	2	231.63	4	2480	2	270.94	4	2704	2	30.95
	4	22	4860	6	47.92	24	44196	6	366.18	16	5360	4	513.83	12	11296	4	45.34
	8	62	7300	11	72.78	112	114303	14	448.66	68	9992	10	760.99	38	32480	9	59.85
	16	154	11752	16	102.24	480	210107	30	508.65	152	16392	14	1028.72	128	47616	16	74.73
	32	360	20020	23	113.07	1300	70444	51	1689.69	380	26456	16	1290.18	400	58016	22	92.37
	64	858	33600	42	132.64	2994	95454	84	3334.64	804	42784	16	1585.13	1380	71424	35	115.77
	128	2170	61024	42	141.68	28532	1011961	244	406.90	1690	69601	22	1853.86	3140	137344	38	177.02
raefsky04.rb	2	4	2895	2	23.73	4	2208	2	279.73	4	2964	2	310.89	4	3096	2	46.63
	4	22	7686	6	48.34	16	6844	6	529.09	16	7224	6	615.70	12	10994	4	79.67
	8	78	15021	11	66.26	68	20294	12	780.40	62	14184	11	902.41	36	38226	8	117.10
	16	208	24694	18	88.46	186	27159	22	1131.28	144	24382	14	1225.32	136	52476	14	158.44
	32	518	39247	33	103.91	762	61309	42	1029.13	360	42515	18	1541.71	412	71694	20	206.61
	64	1626	69227	61	113.95	2016	105949	80	1356.53	890	70037	24	1825.73	1280	101586	32	275.99
	128	3952	111318	76	118.90	10226	335212	152	1112.38	2702	110729	35	2068.88	3540	175396	38	345.25
wang3.rb	2	4	3062	2	5.88	4	2642	2	78.35	4	4492	2	44.42	4	5220	2	16.60
	4	24	6365	6	10.41	24	5690	6	150.12	24	9171	6	85.07	12	19734	4	29.78
	8	96	10020	14	13.74	88	9102	14	265.92	84	13324	14	114.94	42	43902	9	44.64
	16	256	14965	21	17.49	232	13717	26	366.03	240	17447	28	150.74	148	60030	18	60.38
	32	694	21964	38	20.18	592	20350	38	462.74	576	24976	32	175.08	512	68882	33	79.47
	64	1704	30962	61	22.71	1408	28226	50	589.98	1306	34309	35	205.46	1832	78114	56	103.54
	128	3838	42494	71	25.78	3456	40136	75	554.45	2974	45594	43	231.85	4686	88530	66	133.38
wang4.rb	2	4	3031	2	4.09	4	2669	2	114.40	4	5220	2	40.31	4	5220	2	15.88
	4	24	6118	6	8.51	24	5865	6	191.45	24	7972	6	79.65	12	19630	4	28.56
	8	94	9720	14	12.53	84	9048	14	277.66	84	12962	13	114.67	44	47202	10	43.42
	16	264	14672	25	15.35	228	13659	21	355.09	242	17734	26	144.83	170	62824	20	59.38
	32	704	21854	38	19.28	644	20132	40	482.51	564	24488	36	176.66	594	72602	39	77.29
	64	1704	30103	55	21.47	1506	28483	51	593.85	1352	32741	35	205.99	2132	81448	67	101.02
	128	3608	41675	68	23.71	3542	40146	77	564.78	2876	43652	36	232.85	5298	92574	86	128.19