Universiteit Utrecht

MSc thesis

Improving the Mondriaan vector distribution

Author:
Hasan Kurt

Supervisor:
Prof. Dr. Rob H. Bisseling

Abstract

Mondriaan is a hypergraph based matrix partitioner, used to distribute the matrix and vectors in parallel sparse matrix-vector multiplication (SpMV) when calculating the product $u = Av$. In this study, we investigate the problem of distributing the input vector v over our P processors, in order to reduce the number of messages, while keeping the communication volume more or less equal. A novel method assigning each vector element to the lowest numbered processor gives us significantly lower total message count, while keeping the communication volume constant. Another method, a novel hypergraph based heuristic, roughly halves the total amount of messages, while it increases the communication volume. Both newly developed methods provide large reductions in the total amount of messages and can hence be considered as alternative vector distribution methods.

February 21, 2016

Contents

1 Introduction

1.1 Sparse Matrix Vector Multiplication

Sparse matrix-vector multiplication (SpMV) is an operation which is encountered very frequently in applied mathematics and is concerned with calculating $u = Av$, where A is an $m \times n$ sparse matrix, and u and v are dense vectors of length m and n respectively. Generally, we have huge matrices and the number of non-zeros is generally much smaller than the number of matrix elements. To tackle such problems, this operation is carried out in parallel, so P processors carry out a part of this task simultaneously. Mondriaan [12] is a software package developed to find a partitioning of the matrix A and the input and output vectors v and u respectively.

There are several software packages on the market to find a suitable partitioning of the matrix and the vectors. Generally, those are based on hypergraph partitioning.

1.2 Partitioning a matrix

The first and also the most expensive part is finding a partitioning of the matrix. Upon finding this partitioning, a partitioning of the input and output vectors can also be found.

There are several considerations which we must bear in mind in finding an optimal partitioning: On the one hand we want to spread the non-zeros as equally as possible over our P processors, while on the other hand we want to minimize the amount of communication.

Generally, we allow some kind of imbalance in distributing the non-zeros, as otherwise finding a solution would be nearly impossible, which is expressed in the imbalance criterion:

$$w(A_i) \leq \frac{w(A)}{P}(1 + \epsilon) \tag{1}$$

where $w(A_i)$ is the weight (amount of non-zeros generally) assigned to processor i and ϵ is the load imbalance. Generally, the load imbalance is around a few percent.

There are several ways to partition said matrix; many of them make use of the **hypergraph model** where a sparse matrix is represented as a hypergraph. [4]. The resulting problem is now a hypergraph partitioning problem, which is known to be NP-hard [7].

Mondriaan[12], Zoltan[6], PaToH[1], hMetis [10] are some hypergraph partitioners which are commonly used.

Some matrix partitionings are for instance: [9]

- 1D partitioning: Each matrix row (or column) is assigned to a processor.
- 2D-Block: Cartesian partitioning where each processor owns the intersection of a subset of rows with a subset of columns, where the processors are arranged in a $\sqrt{p}$ by $\sqrt{p}$ grid.

- 2D-random partitioning: Cartesian with rows/columns randomly, uniformly distributed to processors [8]
- Mondriaan partitioning, see below

1.3 Mondriaan Package

Mondriaan is an opensource software package released in 2002. [12]. It uses a bipartitioning approach, - recursively halving the original matrix until it is distributed over all processors. Over the years several additions ("bells and whistles") have been made, making the software more practical and faster, for instance to allow for values of P which are not a power of two.

As we are not interested in the matrix partitioning, we do not further go into its details. Once a matrix partitioning is obtained, the next problem is the one of the vector distribution. In this project, we only consider this problem.

See chapter 4 of the book by Bisseling [2] for more details about sparse matrix-vector multiplication and Mondriaan.

1.4 Project objective

The goal of our investigation is to find a vector distribution method which gives us a vector distribution ϕ that minimizes the **total amount of messages** (totmsg). If processor i sends one or more pieces of information (words) to processor j, that counts as one message. The metric which was of interest until now was the **communication cost** (maxcom), which is the maximum amount of words sent or received per processor. The vector partitioning method used now in the Mondriaan software package optimizes for this metric, by using the local-lower bound method [3], which is a heuristic.

Until now, it was not very interesting to optimize for the total amount of messages as generally the number of processors was quite small. However, with problems becoming larger and larger, and with $P \approx 10^6$, it becomes more important to also optimize for this metric.

Another metric which is of interest is the **maximum amount of messages sent** (maxmsg), which would be a metric similar to the communication cost as we are again looking at the maximum over all processors. This is, together with totmsg, another metric which we would like to minimize. However, totmsg will be our primary objective.

This project will be a mainly empirical/experimental study, where we will analyse lots of raw data in order to empirically gain new insights.

2 Vector distribution methods

Once a matrix partitioning is obtained, we can look at the problem of finding a vector distribution, where each vector component v_i is assigned to one of the P processors. In fact, two vector distributions (for both vectors v and u) need to be obtained. However, in this project we only look at finding a vector distribution for the input vector v, because the problem of finding a good output vector u is similar.

2.1 Reduced communication matrix

In order to gain more insight in the whole vector distribution method, we visualize the so-called reduced communication matrix (illustrated by Table 1.) See [11] for more details. Let us explain this figure: The reduced communication matrix (RCM) is a P by n' matrix, where P is the number of processors and n' is the number of columns which have at least two non-zeros. Generally, the fraction n'/n is quite small, where values of around $0.05 - 0.2$ are common. The columns which have no non-zeros can be discarded, as well as the columns which have only one nonzero, as the corresponding vector element can automatically be assigned to that processor in that case. If processor i has at least one non-zero in column j, where $0 \leq j < n'$, we denote it with a non-zero at position (i, j).

Using the RCM, we can also set up the **message matrix**, which can be seen in Table 2. This square matrix of dimensions P by P has a non-zero at position (i, j) if processor i sends a message to processor j. Of course, this depends on how you chose the vector distribution. In this case, we assigned v_i to the first occurring processor in column i of the RCM. Let us look again at our RCM: For instance, we might choose for our second vector element any of the four processors which are in the second column of our RCM. If we choose for instance the first processor (p_0), that processor will need to send a message to the second, fourth and fifth. To avoid increasing the communication volume, a vector element needs to be assigned to a processor which is in that column. We will see that it might be a good idea to deviate from this constraint.

Using our message matrix, we now see an intuitive definition of the total number of messages: it is simply the sum of all the off-diagonal non-zeros in this matrix, as the diagonal elements do not give rise to a message.

1	0	1	0	1	1	3	0	1	0
	1		1				1		1
1	1	1		1	1			1	
		1			1		1		1
1	1			1		1	1		1
	1		1					1	
			1		1	1	1		
				1				1	
			1			1	1	1	

$\underbrace{\hspace{20em}}_{n'}$

Table 1: The vector v (on top) and the communication matrix C. Here, v is distributed according to the 1STPROC method, so v_i is assigned to the first occuring processor in column i.

1	1	1	1	1	1		1
	1	1	1	1	1	1	1
			1		1		1

Table 2: The message matrix of the communication matrix from Table 1

2.2 Novel vector distribution methods

We have investigated a few novel vector distributions in order to minimize the total amount of messages.

2.2.1 Assign to P_0 (PROC0)

We assign every vector element to the same processor, e.g. to processor 0. As not every matrix column contains this processor, it leads to an increase in the communication volume. However, this also means that only processor 0 needs to send to all other $P-1$ processors, which allows us to reach the minimal possible number of messages of $P-1$.

A straightforward improvement of this trivial assignment is to choose the single processor which is present in the largest number of columns. However, we have observed that due to the balanced nature of the matrix distribution, this idea does not seem to make a large difference.

2.2.2 Assign to lowest numbered processor (1STPROC)

This straightforward and logical method is based on our observations of Table 2. We see in the matrix that only the first, second and fourth rows have non-zeros in them. This is due to the fact that we assigned each vector element to the first processor in the associated column. Even though real matrices would not show such a dramatic effect, where only three (out of eight) processors send a message, we nevertheless expect it to give us some advantage. A straightforward advantage of this method is that the communication volume stays the same.

2.2.3 Randomized from column (RNDCOL)

This method keeps the communication volume constant as it assigns every vector element to a processor which is present in that column. So, for instance, if column 2 of the RCM contains processors numbered 0,4,6 and 7, this method randomly assigns vector element 2 to one of these 4 processors.

2.2.4 Totally randomized (RND)

A slight generalization of the previous method, this method randomly choses a processor which it assigns to a vector element. Again, as in the trivial distribution PROC0, this dramatically increases the communication volume as the processor chosen might not necessarily be present in that column.

2.2.5 Hypergraph based method (HG)

We could try to model this problem as a hypergraph partitioning problem (HGP) by looking closer at the problem at hand, see Table 1.

We could visualize the RCM as a hypergraph, consisting of n' vertices and P nets (hyperedges). Doing this, we get figure 1. This approach, where we consider the HGP of the communication matrix, was also earlier investigated [11], however there it was part of a larger (two-phase) algorithm. In fact, this method is more of a stepping stone to another method, HG1ST, which is discussed in the next section. That method does differ significantly from the method discussed in the literature [11]. The difference of our method compared to the method discussed in [11], is that we drop the consistency requirement: we do the partitioning during the fan-out and fan-it independently rather than doing it together. Let us give a few definitions in order to further explain this method:

A hypergraph $H = (V, N)$ is defined as a set of vertices V and nets N, where every net n_i is a subset of vertices. The size $|n_i|$ of a net is equal to the number of vertices in that net. $Nets(v_j)$ is the set of nets in which vertex v_j occurs and the number of nets in which this vertex occurs is thus $|Nets(v_j)|$. In our case, $|V| = n'$ and $|N| = P$. In our case, the problem is finding a hypergraph partitioning (assigning each vertex to a processor) which minimizes the number of cut nets. The connectivity λ_i of a net is the number of parts in which it is partitioned. So, for instance, if all the vertices in a net go to the same processor, λ_i equals 1, hence this net is not cut.

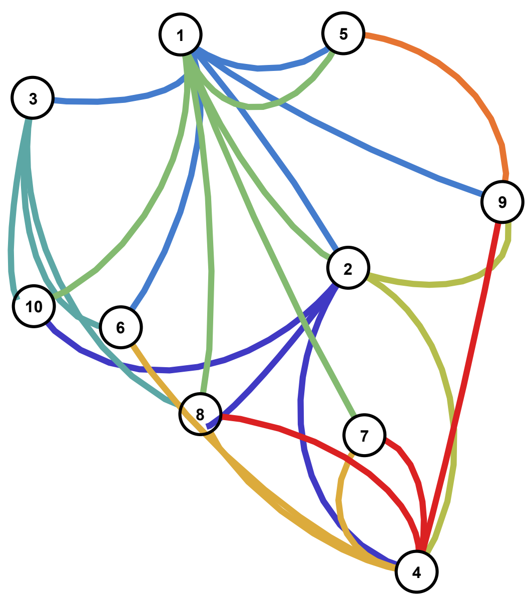

Figure 1: The hypergraph associated with the RCM in Table 1. Obtained by the row-net model, see [1] [11]. (Here, we begin counting the vertices from 1.). Each circle represents a vertex, and each color represents a net.

We can now find an expression for an upper bound of the total number of messages N_{msg} by looking at a partitioning, without making assumptions about whether a vector element v_i is assigned to a processor which is present in that column, and without taking into account the overlap between $Nets(v_i)$.

In order to explain this method, and why solving the HGP minimizes N_{msg}, we first define Δ_i, which gives us the i-th diagonal of the message matrix.

$$\Delta_i = \begin{cases} 1, & \text{if } \exists j : C_{i,j} = 1 \land \phi(j) = i \\ 0, & \text{otherwise} \end{cases}$$

Let us also give a formal definition of the connectivity of net i, λ_i:

$$\lambda_i = |\,\{s : 0 \le s < p \land \exists j : C_{ij} \ne 0 \land \phi(j) = s\}\,|$$

By noting that N_{msg} is the sum of all the off-diagonal elements of the message matrix, and by noting that the sum of λ_i gives us the total number of elements, we get the following formula:

$$N_{msg} = \sum_{i=0}^{P-1} \lambda_i - \sum_{i=0}^{P-1} \Delta_i = \sum_{i=0}^{P-1} (\lambda_i - 1) + \sum_{i=0}^{P-1} (1 - \Delta_i)$$

By noting that the last term lies between 0 and P, we can bound N_{msg} from below and from above as follows:

$$\sum_{i=0}^{P-1} (\lambda_i - 1) \le N_{msg} \le \sum_{i=0}^{P-1} (\lambda_i - 1) + P =: V_{\lambda-1} + P$$

where $V_{\lambda-1}$ is the total number of cut nets. The optimal hypergraph partitioning will minimize $V_{\lambda-1}$, and as N_{msg} is bound between those, it will thus also minimize N_{msg}, where we note that P is growing much slower than $V_{\lambda-1}$. Upon finishing the hypergraph partitioning, all our n' vertices will have been mapped to one of our P processors by the function ϕ.

2.2.6 Hybrid hypergraph/lowest numbered processor (HG1ST)

We could also take a mixture of the HG method with the 1STPROC method: Suppose for instance that vector elements 5, 9 and 12 are given to processor 6 during the hypergraph method. On the one hand, we want to assign all those three elements to the same processor, while on the other hand we do not know whether it is a good idea to give them all to processor 6, as we do not have a guarantee that that particular processor will be in one (or more) of the associated columns, thus increasing the communication volume.

By assigning all those three vector elements to the first occurring (i.e. lowest numbered) processor of the first vector element (element 5), we hope to improve HG. If, for instance, column 5 has processors 2, 7 and 9, we choose the lowest numbered processor (processor 2) and assign this processor to **all** vector elements 5, 9 and 12. As we will see in the results section, our intuition seems to be right and this modified HG method seems to show significant improvements in both communication volume as well as total number of messages.

2.3 Local search methods

As the problem of finding an optimal distribution of the vector elements is hard, one might also consider several tried and tested generic local search (LS) methods, including:

- Greedy improvement
- Simulated Annealing
- Monte Carlo (MC) methods

The advantage of these is their relative simplicity in designing and implementing the algorithm. However, upon further investigating a possible greedy improvement algorithm, we quickly found out one major drawback of these methods: as we are interested in minimizing the total amount of messages, we make no distinction of the actual amount of data sent from processor i to j. Hence, LS methods are at a disadvantage as it is very hard to assess whether a neighboring solution improves the current solution, as the number of messages being sent is unlikely to be reduced.

2.3.1 Greedy Improvement

Even though greedy improvement didn't prove to be an efficient method by the experiments we did, we nevertheless mention it here due to its simplicity. Also, in the original Mondriaan vector distribution, a greedy improvement is also applied at the end, aiming to minimize the communication cost instead of the messages. We have tried using a variation of this greedy improvement scheme in order to reduce our total messages count, however the results were very modest.

3 Results

We performed experiments on a large variety of matrices, all taken from the University of Florida Sparse Matrix Collection [5], a huge collection of sparse matrices of varying sparsities and structures. We grouped our matrices on which we performed our experiments into different groups, so one can better see how different methods perform on different matrices and whether the same results apply to different kinds of matrices.
In all our experiments, we took a maximum imbalance of $\epsilon = 0.1$ and took values of P varying from 128 to 8192. Due to time constraints, we could not take even larger matrices as the matrix distribution takes a very long time for large matrices.

3.1 Results symmetric matrices

We took several symmetric matrices and tested all our seven methods on them, see Table 3 for the list of the matrices. All our methods are compared to the original method which is currently in use, denoted by ORIG. For detailed results of some of these matrices, please look at Table 4. We see in Figures 2 and 3 that with increasing values of P, the maxcom and maxmsg metrics empirically seem to converge to each other, which suggests that minimizing maxcom could be a good idea to also minimize maxmsg.

Matrix	rows	cols	nnz
bcsstk15	3948	3948	60882
brainpc2	27607	27607	96601
delaunay_n18	262144	262144	786396
dixmaanl	60000	60000	179999
Dubcova2	65025	65025	547625
finance256	37376	37376	167936
k1_san	67759	67759	303364
mario001	38434	38434	114643
minsurfo	40806	40806	122214
nasa2910	2910	2910	88603
pkustk05	37164	37164	1121154
rajat09	24482	24482	64982
torsion1	40000	40000	118804

Table 3: Symmetric matrices

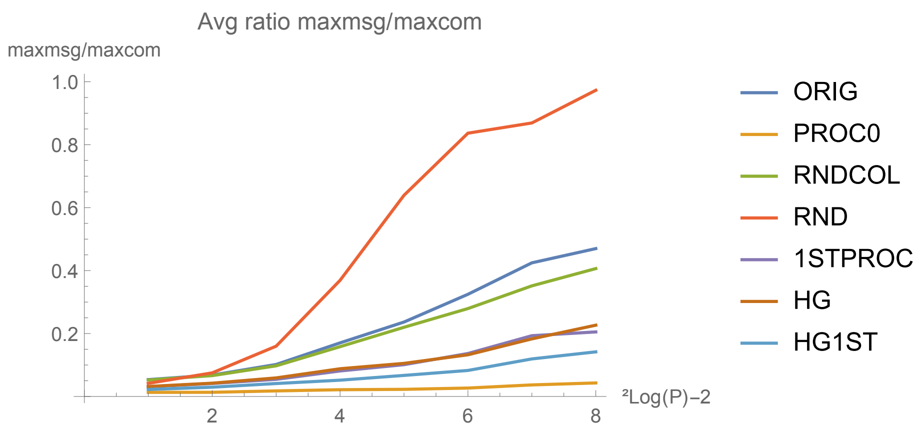

Figure 2: Geometric mean of ratios maxmsg/maxcom for several methods, mean over all symmetric matrices.

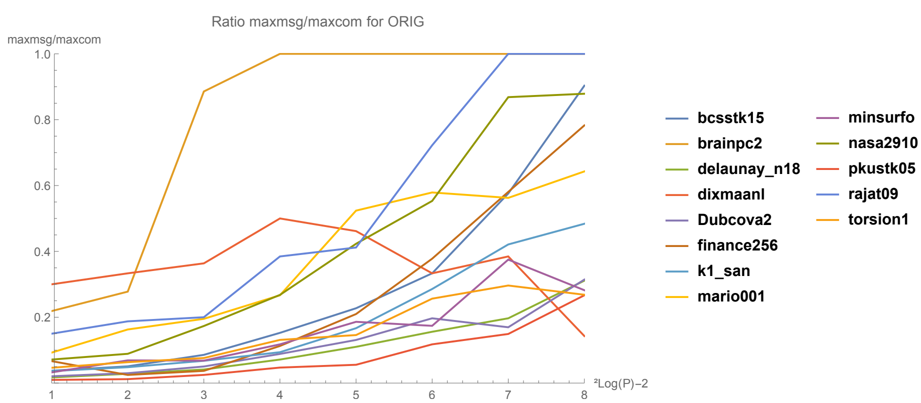

Figure 3: Ratios maxmsg/maxcom for ORIG, all symmetric matrices.

Matrix	P	Communication Volume							Total Messages							Max messages						
		ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST
bcsstk15	8	829	1396	829	1421	829	1420	1343	36	7	38	56	18	29	17	6	7	6	7	6	7	7
	32	2031	3747	2031	3752	2031	3860	3241	165	31	181	967	88	185	107	9	31	10	31	7	11	16
	64	3052	5540	3052	5560	3052	5638	4792	353	63	381	3004	196	379	226	10	63	10	56	10	15	16
	256	6870	10597	6870	10607	6870	10525	8913	1909	255	2136	9787	1107	1357	921	14	255	17	65	14	14	17
	1024	19328	23254	19328	23249	19328	23233	20945	14141	1023	13692	22989	8838	9481	6659	27	1023	36	66	49	26	46
	avg	1.	1.61117	1.	1.6164	1.	1.62442	1.41488	1.	0.146446	1.05792	3.8399	0.55638	0.85224	0.548203	1.	7.74032	1.11301	3.2619	1.0228	1.14879	1.48876
brainpc2	8	154	236	154	250	154	240	239	40	7	41	54	15	31	21	7	7	7	7	7	7	7
	32	419	706	419	697	419	681	571	226	31	214	470	92	130	107	31	31	31	31	31	31	31
	64	724	1184	724	1182	724	1181	951	489	63	467	991	199	283	211	63	63	63	63	63	63	63
	256	2161	3278	2161	3292	2161	3280	2539	1732	255	1652	3176	672	918	694	255	255	255	255	255	255	255
	1024	7932	11748	7932	11743	7932	11747	9085	6763	1023	6583	11682	2700	3562	2682	1023	1023	1023	1023	1023	1023	1023
	avg	1.	1.58003	1.	1.5854	1.	1.57572	1.2837	1.	0.144562	0.956134	1.7909	0.393851	0.596799	0.444796	1.	1.	1.	1.	1.	1.	1.
delaunay_n18	8	1650	2794	1650	2880	1650	3059	2649	28	7	28	56	15	26	17	5	7	5	7	5	5	5
	32	4099	7857	4099	7917	4099	8135	6764	156	31	157	992	83	161	116	7	31	7	31	7	10	15
	64	5956	11711	5956	11666	5956	11649	9964	354	63	350	3776	181	377	260	9	63	9	63	7	12	14
	256	13146	25897	13146	25890	13146	25685	22023	1548	255	1541	21267	806	1530	1153	12	255	12	116	10	14	21
	1024	27163	53053	27163	53055	27163	53102	43972	6147	1023	6086	51750	3280	5619	4170	14	1023	15	87	13	16	24
	avg	1.	1.90846	1.	1.91917	1.	1.91391	1.64936	1.	0.184886	0.991808	7.30034	0.522698	0.972447	0.71673	1.	9.66162	1.0188	5.17304	0.862588	1.19765	1.57742
dixmaan1	8	28	28	28	28	28	28	28	11	11	11	11	11	11	11	3	3	3	3	3	3	3
	32	143	143	143	143	143	143	143	59	59	59	59	59	59	59	4	4	4	4	4	4	4
	64	199	199	199	199	199	199	199	88	88	88	88	88	88	88	4	4	4	4	4	4	4
	256	862	1723	862	1717	862	1700	912	397	236	374	1689	219	429	225	5	236	5	26	4	6	4
	1024	3828	7636	3828	7633	3828	7638	2180	1470	1003	1410	7590	790	1698	420	6	1003	6	79	6	17	5
	avg	1.	1.18873	1.	1.18816	1.	1.18678	0.938635	1.	0.893333	0.987411	1.47137	0.859004	1.0281	0.796463	1.	3.06999	1.	1.69604	0.972492	1.16529	0.95058
Dubcova2	8	1300	2380	1300	2275	1300	2333	2040	26	7	26	56	13	34	28	5	7	5	7	4	7	7
	32	3047	5902	3047	5850	3047	5670	4963	146	31	146	985	74	158	110	7	31	7	31	6	9	12
	64	4782	9304	4782	9279	4782	9355	7798	324	63	325	3574	164	366	252	10	63	10	62	10	11	17
	256	10321	19951	10321	19943	10321	19997	17117	1392	255	1395	17150	723	1592	1141	12	255	12	102	10	13	22
	1024	21987	41347	21987	41337	21987	41306	34315	5618	1023	5605	40539	3031	5382	3957	11	1023	12	73	9	12	21
	avg	1.	1.91205	1.	1.89184	1.	1.88408	1.62685	1.	0.199303	1.00149	7.01122	0.513951	1.10201	0.803261	1.	10.1635	1.02305	5.13817	0.875211	1.28525	1.73756
finance256	8	195	195	195	195	195	195	195	16	16	16	16	16	16	16	2	2	2	2	2	2	2
	32	1886	1886	1886	1886	1886	1886	1886	80	80	80	80	80	80	80	3	3	3	3	3	3	3
	128	6040	10626	6040	10611	6040	10573	9434	1012	127	1102	7771	591	903	539	13	127	14	84	11	16	17
	256	8566	14842	8566	14838	8566	14892	13110	2300	255	2478	13249	1271	1711	1148	16	255	18	85	18	16	24
	1024	16424	27016	16424	27037	16424	27046	23317	9946	1023	9974	26671	6438	6578	5250	18	1023	29	69	40	33	34
	avg	1.	1.41831	1.	1.41759	1.	1.41393	1.30219	1.	0.263592	1.02414	2.80498	0.704261	0.835895	0.635682	1.	6.08275	1.1073	2.80485	1.10611	1.15944	1.31373
mario001	8	325	325	325	325	325	325	325	26	26	26	26	26	26	26	5	5	5	5	5	5	5
	32	849	849	849	849	849	849	849	164	164	164	164	164	164	164	8	8	8	8	8	8	8
	64	1269	1269	1269	1269	1269	1269	1269	339	339	339	339	339	339	339	8	8	8	8	8	8	8
	256	2736	5449	2736	5447	2736	5462	4159	1284	255	1246	5228	742	995	725	10	255	10	40	9	9	13
	1024	6682	13295	6682	13294	6682	13289	9629	4336	1023	4146	13211	2992	3350	2336	9	1023	10	31	9	9	14
	avg	1.	1.41021	1.	1.4105	1.	1.41153	1.23219	1.	0.457047	0.983244	1.94717	0.780656	0.883219	0.752498	1.	6.08555	1.01326	1.948	0.986916	0.959769	1.17689
minsurfo	8	760	1276	760	1332	760	1434	1235	24	7	24	56	12	30	14	5	7	5	7	4	6	6
	32	1680	3256	1680	3244	1680	3167	2701	128	31	128	938	67	142	77	6	31	6	31	6	10	10
	64	2439	4801	2439	4782	2439	4862	3970	275	63	273	2721	141	331	185	8	63	8	57	6	11	20
	256	4946	9777	4946	9776	4946	9708	7766	1035	255	1016	9064	546	1050	719	8	255	9	78	7	12	11
	1024	11147	21937	11147	21932	11147	21918	16294	4158	1023	4011	21687	2380	3399	2241	10	1023	10	76	12	12	16
	avg	1.	1.91419	1.	1.92547	1.	1.94644	1.57557	1.	0.249215	0.984935	6.17837	0.52564	1.0754	0.624479	1.	11.7083	1.01483	4.98929	0.844377	1.27448	1.58134
All	8	1.	1.42245	1.	1.45106	1.	1.46296	1.36958	1.	0.361044	1.01705	1.51901	0.608418	0.991865	0.695872	1.	1.14924	1.01193	1.14924	0.961138	1.08074	1.09363
	16	1.	1.59164	1.	1.58909	1.	1.56337	1.39859	1.	0.313149	1.00428	2.49433	0.598503	1.0586	0.720224	1.	1.7896	1.0091	1.7896	0.909884	1.2039	1.34693
	32	1.	1.6165	1.	1.60823	1.	1.59472	1.38959	1.	0.303358	1.00463	3.67425	0.593209	0.996969	0.70158	1.	2.65867	1.00814	2.6098	0.9046	1.22346	1.36864
	64	1.	1.69559	1.	1.69329	1.	1.69891	1.42171	1.	0.253784	1.00701	5.51115	0.566817	1.01432	0.654121	1.	4.39188	1.01696	3.89829	0.880574	1.20577	1.41469
	128	1.	1.7553	1.	1.75589	1.	1.75915	1.45538	1.	0.206221	1.0191	6.52117	0.550894	0.919943	0.616822	1.	7.88443	1.02756	4.86987	0.881699	1.08906	1.34625
	256	1.	1.80907	1.	1.80956	1.	1.80579	1.42319	1.	0.183738	1.01262	6.17161	0.530042	0.849074	0.563554	1.	16.1622	1.0441	4.95814	0.910098	1.02252	1.32633
	512	1.	1.66965	1.	1.6696	1.	1.67005	1.37631	1.	0.182509	1.017	4.44303	0.574597	0.794131	0.559897	1.	20.5655	1.09349	3.86462	1.03434	1.04913	1.48069
	1024	1.	1.71199	1.	1.71184	1.	1.71203	1.28282	1.	0.166535	1.00025	3.81683	0.577434	0.784412	0.510503	1.	49.731	1.1934	4.23356	1.22028	1.27227	1.58599

Table 4: Results of all symmetric matrices. The last row for each matrix gives the geometric means of the normalized metrics (wrt original method). The "**All**" rows give the geometric means of normalized metrics for a given P value over all matrices.

3.2 Results (other) square matrices

Same observations regarding both the maxmsg/maxcom ratios as well as the generals trends concerning the different metrics also hold for other (not necessarily symmetric) matrices, see Table 5 for a list of those. Detailed data of some of these matrices are in Table 6.

Matrix	rows	cols	nnz
add32	4960	4960	23884
arc130	130	130	1282
bloweybq	10001	10001	39996
epb2	25228	25228	175027
g7jac010	2880	2880	19635
gyro_m	17361	17361	178896
jpwh_991	991	991	6027
mark3jac060	27449	27449	170695
rbsb480	480	480	17088
tuma1	22967	22967	50560
watt_2	1856	1856	11550

Table 5: Other matrices

Matrix	P	Communication Volume							Total Messages							Max messages						
		ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST
add32	8	35	63	35	57	35	60	47	26	7	23	36	18	19	13	5	7	5	6	5	4	5
	32	134	227	134	231	134	230	170	110	31	108	206	86	100	72	8	31	8	14	10	8	12
	128	478	803	478	807	478	805	593	394	127	382	786	324	391	276	8	127	12	19	12	10	13
	512	4323	6660	4323	6657	4323	6667	5448	2819	511	2672	6558	2189	2415	1719	14	511	23	40	34	25	44
	avg	1.	1.67388	1.	1.65576	1.	1.66497	1.26194	1.	0.263692	0.948003	1.88693	0.768029	0.880561	0.624583	1.	7.72359	1.23251	1.99305	1.36504	1.18322	1.46273
arc130	8	59	67	59	67	59	65	60	47	7	35	43	15	35	14	7	7	7	7	7	7	7
	16	153	193	153	197	153	198	171	125	15	88	138	55	65	53	15	15	15	15	15	15	15
	32	279	360	279	360	279	362	313	250	31	220	316	124	163	114	27	31	28	28	30	28	30
	avg	1.	1.22724	1.	1.23566	1.	1.22757	1.08438	1.	0.130377	0.772696	1.08483	0.411443	0.632034	0.386177	1.	1.04713	1.0122	1.0122	1.03574	1.0122	1.03574
bloweybq	8	26	41	26	40	26	38	29	17	7	15	27	11	15	11	7	7	7	7	7	7	7
	32	95	159	95	156	95	150	101	67	31	64	143	50	71	53	31	31	31	31	31	31	31
	64	238	409	238	409	238	402	250	159	63	149	384	111	163	115	63	63	63	63	63	63	63
	256	877	1498	877	1495	877	1496	901	597	255	569	1470	435	653	439	255	255	255	255	255	255	255
	1024	8592	15246	8592	15243	8592	15239	11655	5208	1023	4988	15126	3712	4435	3267	1022	1023	1022	1022	1022	1022	1022
	avg	1.	1.68545	1.	1.68532	1.	1.63901	1.10471	1.	0.371395	0.94048	2.38497	0.699864	0.987802	0.69914	1.	1.00012	1.	1.	1.	1.	1.
epb2	8	682	1160	682	1193	682	1270	1136	26	7	26	56	13	25	15	5	7	5	7	5	6	6
	32	1871	3421	1871	3446	1871	3318	2983	184	31	198	945	99	165	114	11	31	11	31	10	11	15
	64	2562	4678	2562	4667	2562	4689	3956	385	63	433	2713	219	376	248	14	63	15	55	10	14	13
	256	5159	9481	5159	9485	5159	9481	7786	1372	255	1646	8806	860	1210	902	23	255	23	60	25	22	23
	1024	11257	19925	11257	19921	11257	19943	15712	5880	1023	5973	19742	3744	4161	3044	19	1023	29	50	56	54	59
	avg	1.	1.79128	1.	1.81192	1.	1.82366	1.53933	1.	0.179841	1.07156	4.39754	0.561399	0.872404	0.597355	1.	6.48162	1.11563	2.5664	1.16687	1.2715	1.35844
g7jac	8	399	649	399	682	399	667	654	45	7	44	56	24	36	21	7	7	7	7	7	7	7
	32	888	1534	888	1531	888	1556	1412	272	31	265	766	157	207	164	17	31	13	30	14	12	18
	64	1230	2058	1230	2057	1230	2082	1812	551	63	569	1591	355	382	280	16	63	17	39	16	14	25
	128	1877	2945	1877	2950	1877	2956	2578	1136	127	1180	2702	858	793	622	18	127	20	44	30	22	30
	256	2917	4278	2917	4281	2917	4267	3695	2280	255	2248	4143	1769	1717	1392	15	255	24	42	28	33	42
	avg	1.	1.6329	1.	1.64572	1.	1.64024	1.49062	1.	0.121359	1.00403	2.10691	0.634804	0.760799	0.57068	1.	3.20138	1.0632	1.8238	1.15292	1.10284	1.45913
gyro_m	8	228	228	228	228	228	228	228	34	34	34	34	34	34	34	6	6	6	6	6	6	6
	32	848	1609	848	1637	848	1633	1346	110	31	109	766	57	135	89	6	31	6	31	6	10	16
	64	1834	3566	1834	3540	1834	3541	2846	218	63	217	2281	116	292	171	9	63	8	53	6	12	16
	256	5895	10663	5895	10661	5895	10642	8762	1255	255	1294	9758	705	1209	725	13	255	12	68	10	13	16
	1024	16901	27408	16901	27413	16901	27438	22488	6656	1023	6896	27034	4292	4960	3396	16	1023	22	63	28	23	25
	avg	1.	1.68469	1.	1.68995	1.	1.68078	1.43223	1.	0.267031	1.00873	5.11259	0.597745	1.0556	0.716616	1.	9.15342	0.987817	3.80736	0.970234	1.21131	1.52927
jpwh_991	8	265	458	265	439	265	430	417	46	7	47	56	28	34	25	7	7	7	7	7	7	7
	16	348	615	348	616	348	609	561	121	15	123	225	74	94	61	10	15	12	15	11	12	14
	32	519	920	519	917	519	914	815	252	31	256	602	170	176	135	12	31	13	25	14	14	25
	64	898	1492	898	1492	898	1485	1284	512	63	533	1244	384	370	295	15	63	17	32	18	19	20
	128	1798	2622	1798	2621	1798	2630	2298	1266	127	1207	2418	991	944	750	16	127	20	42	30	27	34
	avg	1.	1.6733	1.	1.65851	1.	1.64638	1.48738	1.	0.123427	1.00927	1.90511	0.681898	0.736087	0.5495	1.	2.64388	1.12991	1.77259	1.23624	1.24511	1.52559
All	8	1.	1.4541	1.	1.45163	1.	1.4381	1.34398	1.	0.288844	0.959693	1.38957	0.601067	0.915316	0.561144	1.	1.1461	1.01411	1.13015	0.993748	1.0741	1.09611
	16	1.	1.57614	1.	1.59821	1.	1.58009	1.38897	1.	0.209669	0.967609	2.00316	0.589828	0.885172	0.627004	1.	1.47774	1.02287	1.42426	0.997959	1.14553	1.17498
	32	1.	1.60138	1.	1.6062	1.	1.57562	1.36352	1.	0.20298	0.991845	2.62853	0.617136	0.843009	0.612031	1.	2.16843	1.00257	1.91831	1.0215	1.04139	1.31865
	64	1.	1.7277	1.	1.72717	1.	1.71689	1.40889	1.	0.160825	1.01164	3.6159	0.623871	0.846972	0.59024	1.	3.99857	1.08237	2.64454	0.955898	1.06091	1.30752
	128	1.	1.54355	1.	1.54521	1.	1.54202	1.29236	1.	0.180557	0.997136	2.78671	0.696889	0.845666	0.634354	1.	5.45976	1.17851	2.58051	1.34589	1.2513	1.50596
	256	1.	1.61886	1.	1.61811	1.	1.61414	1.32425	1.	0.144264	1.00598	2.86597	0.709023	0.819941	0.6121	1.	10.0476	1.23012	2.49992	1.39603	1.24617	1.66397
	512	1.	1.74762	1.	1.74725	1.	1.7485	1.40794	1.	0.157702	0.990768	3.37438	0.687663	0.787691	0.573404	1.	16.8074	1.19982	2.49407	1.47051	1.22298	1.77208
	1024	1.	1.73306	1.	1.73307	1.	1.7338	1.4094	1.	0.152446	1.00454	2.95134	0.724309	0.766051	0.579471	1.	25.8571	1.25072	2.3539	1.65453	1.50262	2.01556

Table 6: Results of other matrices. The last row for each matrix gives the geometric means of the normalized metrics (wrt original method). The "**All**" rows give the geometric means of normalized metrics for a given P value over all matrices.

3.3 Results rectangular matrices

We also performed experiments on rectangular (non-square) matrices, see Table 7 for a list of those matrices. Again, it looks like our novel method HG1ST seems to provide the largest reduction in the total number of messages. Detailed data for some of these matrices can be found in Table 8.

Matrix	rows	cols	nnz
192bit	13691	13682	154303
cat_ears_3_4	5226	13271	39592
ch8-8-b2	18816	1568	56448
EternityII_Etilde	10054	204304	1170516
flower_8_4	55081	125361	375266
Franz10	19588	4164	97508
Franz8	16728	7176	100368
ge	10099	16369	44825
kneser_10_4_1	349651	330751	992252
NotreDame_actors	392400	127823	1470404
photogrammetry2	4472	936	37056
relat7	21924	1045	81355
TF17	38132	48630	586218
tomographic1	73159	59498	647495

Table 7: Other matrices

Matrix	P	Communication Volume							Total Messages							Max messages						
		ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST
192bit	8	11438	14535	11438	14801	11438	15078	14535	56	7	56	56	28	41	7	7	7	7	7	7	7	7
	32	20213	25774	20213	25804	20213	25796	25771	992	31	992	992	496	697	93	31	31	31	31	31	31	31
	64	24842	31101	24842	31092	24842	31042	30975	4013	63	4004	4029	1966	2295	542	63	63	63	63	63	63	63
	256	34077	41101	34077	41085	34077	41062	40665	27590	255	26057	30219	13678	13000	8103	255	255	255	255	255	255	255
	1024	43180	50682	43180	50679	43180	50674	49401	42590	1023	42222	49484	32225	30850	25731	934	1023	937	939	1013	1005	1020
	avg	1.	1.23824	1.	1.2385	1.	1.24143	1.22656	1.	0.023974	0.982751	1.05374	0.532004	0.601202	0.212848	1.	1.01294	1.0004	1.00092	1.01169	1.01069	1.01256
cat_ears_3_4	8	439	781	439	756	439	759	716	51	7	52	56	27	30	24	7	7	7	7	7	7	7
	32	858	1677	858	1650	858	1641	1478	300	31	289	774	195	217	172	18	31	17	31	14	15	23
	64	1091	2116	1091	2116	1091	2148	1825	526	63	508	1629	349	379	300	17	63	18	42	16	16	26
	256	1524	2980	1524	2989	1524	2990	2351	1290	254	1235	2916	1060	1117	875	14	254	15	27	17	11	26
	1024	3268	6415	3268	6415	3268	6417	4544	2840	1006	2720	6394	2370	2975	2116	8	1006	11	21	11	9	15
	avg	1.	1.90493	1.	1.91233	1.	1.90378	1.58267	1.	0.167721	0.975405	2.18993	0.702899	0.787846	0.595157	1.	7.13307	1.06495	1.8633	1.10243	0.934657	1.47887
ch8-8-b2	8	4968	5744	4968	5730	4968	5871	5727	56	7	56	56	28	42	14	7	7	7	7	7	7	7
	32	10041	11195	10041	11230	10041	11239	11216	971	31	991	992	367	666	199	31	31	31	31	31	31	31
	64	12606	13937	12606	13954	12606	13962	13872	3461	63	3661	3906	1424	2319	709	62	63	63	63	63	63	63
	256	17791	19296	17791	19281	17791	19294	18957	14698	255	14697	16654	9467	10804	7014	72	255	109	135	201	101	253
	1024	22539	24082	22539	24090	22539	24082	23189	22291	1023	21912	23839	19772	21181	18661	32	1023	88	92	260	48	286
	avg	1.	1.10331	1.	1.10445	1.	1.10829	1.0898	1.	0.0330796	1.01064	1.07675	0.536719	0.749707	0.35106	1.	2.57182	1.35377	1.43281	1.92281	1.17577	2.08819
EternityII_Etilde	8	9	9	9	9	9	9	9	4	4	4	4	4	4	4	2	2	2	2	2	2	2
	32	24	24	24	24	24	24	24	19	19	19	19	19	19	19	3	3	3	3	3	3	3
	64	111	111	111	111	111	111	111	50	50	50	50	50	50	50	4	4	4	4	4	4	4
	256	1760	3502	1760	3496	1760	3475	2299	521	235	494	3364	315	648	336	5	235	6	50	6	10	8
	1024	9384	18566	9384	18571	9384	18561	13603	3383	1015	3265	18366	2156	3469	2063	11	1015	11	66	12	15	17
	avg	1.	1.40914	1.	1.40895	1.	1.406	1.16316	1.	0.633732	0.978901	2.43463	0.781992	1.08831	0.803468	1.	6.83287	1.00967	2.6673	1.00579	1.21662	1.25944
flower_8_4	8	2159	3771	2159	3744	2159	3901	3550	56	7	56	56	28	38	19	7	7	7	7	7	7	7
	32	2934	5660	2934	5620	2934	5513	5232	435	31	435	984	250	290	211	23	31	22	31	21	21	21
	64	4237	8273	4237	8240	4237	8191	7630	1059	63	1026	3481	634	688	537	27	63	26	63	27	36	49
	256	7925	15634	7925	15643	7925	15661	13907	2967	255	2881	13743	1921	2115	1688	24	255	24	96	24	28	43
	1024	14507	28570	14507	28572	14507	28604	23930	8384	1023	8136	28170	6046	6040	4853	25	1023	24	66	22	17	30
	avg	1.	1.92145	1.	1.91652	1.	1.92066	1.73142	1.	0.0878037	0.979281	2.72031	0.607997	0.703296	0.512037	1.	4.47679	0.98958	2.17058	0.97097	1.01861	1.30417
Franz10	8	6458	8579	6458	8793	6458	8850	8916	56	7	56	56	28	39	25	7	7	7	7	7	7	7
	32	11738	15436	11738	15400	11738	15342	15346	902	31	970	992	434	629	297	31	31	31	31	31	31	31
	64	14687	18632	14687	18538	14687	18548	18206	2978	63	3275	3945	1356	1747	917	61	63	62	63	63	63	63
	256	20054	24116	20054	24112	20054	24132	23328	12076	255	13202	19950	7237	6912	5827	82	255	98	144	209	184	219
	1024	24765	28899	24765	28903	24765	28874	27145	21580	1023	22505	28500	17301	16210	14487	35	1023	77	102	182	150	266
	avg	1.	1.25696	1.	1.25765	1.	1.25966	1.22886	1.	0.0366272	1.06157	1.29783	0.560193	0.642074	0.414908	1.	2.455	1.21132	1.41054	1.69273	1.60273	1.95631
ge	8	177	304	177	297	177	303	291	39	7	35	52	22	34	13	7	7	7	7	6	7	7
	32	450	865	450	857	450	878	724	184	31	176	527	123	151	96	14	31	14	29	13	11	15
	64	600	1156	600	1157	600	1177	906	312	63	299	990	216	251	174	12	63	12	32	13	11	19
	256	1332	2462	1332	2465	1332	2444	1770	932	254	915	2416	710	833	582	10	254	11	27	11	9	17
	1024	2786	4932	2786	4930	2786	4929	3385	2411	989	2269	4918	1889	2590	1703	8	989	10	20	10	9	10
	avg	1.	1.83204	1.	1.83564	1.	1.8382	1.43696	1.	0.232254	0.950765	2.35043	0.68648	0.886065	0.551555	1.	8.81394	1.08836	2.17223	1.07257	0.976784	1.26203
kneser_10_4_1	8	2045	3725	2045	3610	2045	3672	3336	49	7	47	56	25	38	21	7	7	7	7	7	7	7
	32	5020	9612	5020	9701	5020	9801	9199	802	31	797	991	429	357	253	31	31	31	31	29	26	30
	64	5378	10538	5378	10518	5378	10472	9961	1834	63	1808	3660	1125	882	650	51	63	53	63	46	39	50
	256	5788	11423	5788	11421	5788	11436	10509	3972	255	3891	10145	3235	2544	2064	39	255	43	139	50	31	68
	1024	7802	15350	7802	15343	7802	15334	12984	6825	999	6629	15120	6011	6131	5109	32	999	37	81	44	34	59
	avg	1.	1.92874	1.	1.92383	1.	1.93264	1.7663	1.	0.0692042	0.977351	1.79219	0.664652	0.619358	0.452831	1.	3.1197	1.03267	1.70549	1.09557	0.866133	1.29763
TF17	8	16500	27851	16500	27830	16500	29011	24509	46	7	49	56	23	33	17	7	7	7	7	6	6	6
	32	40709	70496	40709	70733	40709	70937	67638	671	31	692	992	366	425	229	30	31	30	31	26	30	30
	64	55540	92158	55540	92335	55540	92192	89149	2362	63	2405	4032	1352	1158	764	53	63	54	63	52	60	60
	256	86127	130461	86127	130534	86127	130053	126283	16254	255	16390	55936	10325	7878	5875	130	255	116	243	120	145	158
	1024	118409	165676	118409	165666	118409	165684	161085	59607	1023	62838	153170	46887	38275	31355	105	1023	122	211	157	179	276
	avg	1.	1.59207	1.	1.59265	1.	1.60465	1.52327	1.	0.0326051	1.04834	2.04892	0.61318	0.592323	0.371138	1.	1.85842	1.0157	1.4152	0.987403	1.15661	1.27445
All	8	1.	1.47286	1.	1.46268	1.	1.47986	1.41409	1.	0.162727	0.997193	1.07035	0.540335	0.712682	0.37327	1.	1.	1.	1.	0.972362	0.986084	0.986084
	16	1.	1.48668	1.	1.50328	1.	1.49992	1.44067	1.	0.0983828	1.00328	1.20216	0.544636	0.710069	0.384112	1.	1.05801	1.	1.05801	0.988693	0.999246	1.03028
	32	1.	1.51028	1.	1.50664	1.	1.51015	1.44175	1.	0.0644716	1.00612	1.38854	0.555233	0.667218	0.352389	1.	1.16393	0.990805	1.1569	0.944664	0.93917	1.01746
	64	1.	1.49191	1.	1.49125	1.	1.49329	1.40791	1.	0.0482799	1.01317	1.66935	0.578403	0.61768	0.368542	1.	1.48003	1.01141	1.34127	0.999476	1.00995	1.16688
	128	1.	1.56968	1.	1.5705	1.	1.5671	1.40058	1.	0.0431341	1.02716	2.39775	0.615624	0.638688	0.418342	1.	3.04155	1.05443	2.04928	1.11802	1.13355	1.36308
	256	1.	1.54598	1.	1.54595	1.	1.54395	1.36808	1.	0.0458864	1.00238	2.42884	0.679583	0.67802	0.494593	1.	5.55543	1.09721	2.35484	1.35936	1.27098	1.77932
	512	1.	1.52748	1.	1.52771	1.	1.52659	1.33397	1.	0.0570227	0.999707	2.28592	0.740425	0.758529	0.58029	1.	11.1983	1.1839	2.47569	1.60609	1.39492	2.35623
	1024	1.	1.5072	1.	1.50722	1.	1.50701	1.30144	1.	0.073799	0.993169	2.07573	0.792311	0.821209	0.651379	1.	25.1356	1.36603	2.5034	1.84588	1.53558	2.48026

Table 8: Results of rectangular matrices. The last row for each matrix gives the geometric means of the normalized metrics (wrt original method). The "**All**" rows give the geometric means of normalized metrics for a given P value over all matrices.

3.4 Some other square matrices

Even though rectangular matrices still show the same patterns concerning both the ratios between the metrics as well as how the different methods behave with increasing P, we still include a selection of (randomly chosen) square matrices with number of nonzeros around 400 thousand. In Table 10 one can find more detailed information, again they are very similar to our previous results, further confirming our observations. We can see in Figures 4 and 5 the same behaviour of different metrics when P becomes larger and larger: namely that for all methods (and in particular ORIG) maxmsg goes to maxcom as P becomes larger.

Matrix	rows	cols	nnz
bips07_1998	15066	15066	62198
dawson5	51537	51537	531157
ex37	3565	3565	67591
hcircuit	105676	105676	513072
ibm_matrix_2	51448	51448	1056610
lung2	109460	109460	492564
onetone1	36057	36057	341088
poisson3Da	13514	13514	352762
RFdevice	74104	74104	365580
shallow_water1	81920	81920	204800
soc-Epinions1	75888	75888	508837
std1_Jac2_db	21982	21982	498771
usroads-48	126146	126146	161950
viscoplastic2	32769	32769	381326

Table 9: Other matrices

Matrix	P	Communication Volume							Total Messages							Max messages						
		ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST
bips07_1998	8	101	143	101	167	101	172	152	29	7	27	51	15	28	17	7	7	7	7	7	7	7
	16	170	300	170	299	170	304	258	66	15	64	163	40	63	38	10	15	10	15	11	10	12
	32	304	557	304	552	304	549	418	137	31	140	406	86	117	80	14	31	14	24	17	15	17
	64	482	847	482	871	482	875	642	255	63	276	789	169	216	151	18	63	21	26	19	19	19
	128	743	1343	743	1333	743	1325	953	431	127	447	1263	287	390	265	21	127	24	34	21	23	23
	256	1127	2007	1127	2012	1127	1998	1350	727	253	762	1984	499	688	459	24	253	25	26	24	27	26
	512	1822	3172	1822	3167	1822	3175	2093	1222	504	1273	3139	856	1264	781	24	504	24	28	26	27	28
	1024	3268	5623	3268	5623	3268	5598	3618	2302	1010	2354	5606	1630	2410	1496	32	1010	32	41	35	36	38
	avg	1.	1.7227	1.	1.75842	1.	1.76509	1.30032	1.	0.294712	1.01832	2.58476	0.643945	0.941663	0.608581	1.	5.14529	1.04192	1.32911	1.06625	1.0735	1.12293
dawson5	8	998	1882	998	1745	998	1972	1353	13	5	14	42	7	24	7	3	5	3	7	3	6	4
	16	1582	2861	1582	2965	1582	2825	2404	30	12	30	195	15	57	23	5	12	5	15	4	7	11
	32	2833	5521	2833	5452	2833	5496	4505	88	29	90	876	45	138	68	6	29	6	31	6	10	11
	64	4707	9121	4707	9152	4707	9147	7577	184	54	189	2991	93	288	140	7	54	7	63	5	12	15
	128	7433	14387	7433	14494	7433	14448	11912	427	122	449	9016	228	563	287	7	122	8	106	6	10	15
	256	11404	22328	11404	22299	11404	22378	18184	828	251	861	18440	434	1171	633	8	251	8	130	6	10	17
	512	17871	34290	17871	34252	17871	34294	27798	2019	508	2121	31936	1077	2453	1414	10	508	10	124	8	12	20
	1024	26984	51091	26984	51075	26984	51087	41104	4197	1020	4458	49814	2310	4757	2794	11	1020	12	98	11	12	17
	avg	1.	1.91025	1.	1.89969	1.	1.91955	1.54179	1.	0.30695	1.04111	11.472	0.52439	1.47255	0.700754	1.	11.8322	1.02795	7.38513	0.858066	1.44206	1.8891
ex37	8	329	578	329	537	329	499	491	26	7	26	56	13	28	24	5	7	5	7	4	6	6
	16	598	1057	598	1051	598	1079	934	66	15	71	236	35	84	55	8	15	8	15	8	10	9
	32	1068	1850	1068	1851	1068	1823	1594	146	31	170	809	79	162	109	9	31	9	30	8	11	11
	64	1686	2798	1686	2806	1686	2847	2337	364	62	387	1989	189	326	203	13	62	12	48	11	11	15
	128	2618	4016	2618	4011	2618	3993	3329	869	127	940	3555	475	600	422	12	127	17	47	12	11	15
	256	4167	5804	4167	5792	4167	5803	4858	2207	250	2202	5545	1286	1380	952	17	250	19	53	21	16	17
	512	6343	8186	6343	8189	6343	8197	7034	4638	502	4372	8066	2871	3155	2240	17	502	26	49	30	22	37
	1024	9384	11277	9384	11278	9384	11282	9802	8463	1004	7851	11230	5928	7424	5196	17	1004	33	46	46	30	55
	avg	1.	1.52713	1.	1.51217	1.	1.50292	1.30244	1.	0.16183	1.02899	2.93665	0.56434	0.877301	0.612829	1.	7.64793	1.20136	2.72906	1.17157	1.14988	1.42241
hcircuit	8	262	411	262	449	262	440	398	39	7	38	53	23	30	19	7	7	7	7	7	6	7
	16	498	818	498	910	498	888	863	86	14	88	211	53	78	36	9	14	11	15	9	11	14
	32	687	1329	687	1301	687	1302	1120	199	30	187	626	129	163	115	12	30	14	30	16	13	14
	64	990	1869	990	1849	990	1851	1608	428	63	411	1389	281	325	229	18	63	19	46	19	18	26
	128	1418	2570	1418	2575	1418	2584	2118	809	125	778	2309	551	612	450	23	125	27	42	31	28	29
	256	1975	3456	1975	3453	1975	3461	2724	1348	251	1278	3358	988	1053	788	42	251	46	53	44	48	51
	512	2997	4975	2997	4973	2997	4943	3833	2359	498	2231	4914	1740	1988	1475	58	498	62	67	62	82	89
	1024	4622	7598	4622	7591	4622	7600	5549	3902	986	3735	7566	3119	3679	2629	100	986	102	126	128	155	156
	avg	1.	1.73323	1.	1.76884	1.	1.75959	1.47206	1.	0.177719	0.963479	2.36464	0.679645	0.819667	0.552141	1.	3.72947	1.09783	1.56353	1.13319	1.16785	1.32686
ibm_matrix_2	8	5746	9962	5746	9971	5746	8590	8818	31	7	31	56	16	27	10	6	7	6	7	6	7	7
	16	8536	15912	8536	15770	8536	15811	14984	89	15	91	240	48	102	48	9	15	8	15	8	13	15
	32	11321	21440	11321	21489	11321	22110	18496	242	31	242	992	128	204	118	13	31	17	31	15	14	21
	64	14515	27611	14515	27791	14515	26961	24094	518	63	541	4021	283	433	264	21	63	26	63	18	20	24
	128	20383	38786	20383	38753	20383	38164	33862	1124	127	1190	14482	615	954	597	38	127	38	126	41	37	36
	256	26472	49737	26472	49664	26472	49695	43855	2397	255	2590	34527	1349	1976	1330	52	255	53	179	51	49	50
	512	34294	63446	34294	63467	34294	63564	55880	4673	511	5176	56238	2692	4114	2762	81	511	86	166	85	82	84
	1024	45199	82055	45199	82063	45199	82047	72483	8846	1023	10201	78875	5303	7644	5346	119	1023	126	177	122	121	121
	avg	1.	1.85434	1.	1.85407	1.	1.81698	1.64082	1.	0.131631	1.05719	6.47427	0.551469	0.884435	0.509643	1.	3.24728	1.06452	2.16572	1.0001	1.06317	1.16734
lung2	8	65	65	65	65	65	65	65	30	30	30	30	30	30	30	7	7	7	7	7	7	7
	16	134	242	134	254	134	250	195	54	15	52	151	29	45	26	8	15	7	13	6	6	11
	32	218	218	218	218	218	218	218	107	107	107	107	107	107	107	5	5	5	5	5	5	5
	64	417	810	417	814	417	822	577	215	63	194	723	114	178	106	9	63	10	26	9	9	12
	128	667	1311	667	1309	667	1321	867	381	126	341	1251	201	316	197	14	126	13	36	11	9	9
	256	1034	2019	1034	2016	1034	2025	1273	658	248	595	1971	358	601	351	8	248	9	26	8	7	11
	512	1674	3278	1674	3283	1674	3274	1983	1140	494	1013	3258	630	1019	610	9	494	10	24	8	8	14
	1024	3083	6061	3083	6056	3083	6061	3421	2139	990	1880	6036	1181	2099	1139	11	990	12	30	10	12	13
	avg	1.	1.63803	1.	1.64851	1.	1.64966	1.19775	1.	0.460855	0.927829	2.28603	0.63039	0.907111	0.607926	1.	8.07567	1.02633	2.0273	0.911428	0.894288	1.1462
All	8	1.	1.42549	1.	1.42426	1.	1.39836	1.32278	1.	0.324745	1.00478	1.424	0.62412	0.946961	0.565112	1.	1.09807	1.	1.12478	0.97341	1.07632	1.05717
	16	1.	1.6733	1.	1.69474	1.	1.68949	1.53194	1.	0.206323	1.01699	2.54021	0.563282	1.03818	0.581115	1.	1.58254	0.990216	1.62518	0.935509	1.13597	1.35578
	32	1.	1.6313	1.	1.62329	1.	1.62726	1.4397	1.	0.195053	1.0417	3.38171	0.602369	0.964527	0.619224	1.	2.29977	1.05034	2.26358	1.01191	1.1719	1.34213
	64	1.	1.7588	1.	1.76246	1.	1.75919	1.49855	1.	0.142648	1.04133	5.54558	0.547999	0.913812	0.55764	1.	4.12067	1.05409	3.40041	0.902876	1.07248	1.36287
	128	1.	1.72517	1.	1.72429	1.	1.72502	1.44552	1.	0.129617	1.03773	5.81955	0.561691	0.828934	0.539395	1.	6.9109	1.07947	4.04549	0.927639	1.0642	1.33827
	256	1.	1.67743	1.	1.67677	1.	1.67773	1.38735	1.	0.125592	1.02328	5.05893	0.569338	0.787611	0.526288	1.	12.0545	1.05157	3.87525	1.0423	1.01943	1.44205
	512	1.	1.6217	1.	1.6213	1.	1.62123	1.32472	1.	0.126212	1.01604	3.95313	0.590894	0.768399	0.528448	1.	20.7535	1.1211	3.22364	1.10903	1.12702	1.57475
	1024	1.	1.58242	1.	1.5821	1.	1.58202	1.2701	1.	0.13562	1.00085	3.11426	0.619949	0.786144	0.545709	1.	36.0378	1.1765	2.77863	1.30197	1.20101	1.63425

Table 10: Results of other square matrices. The last row for each matrix gives the geometric means of the normalized metrics (wrt original method). The "**All**" rows give the geometric means of normalized metrics for a given P value over all matrices.

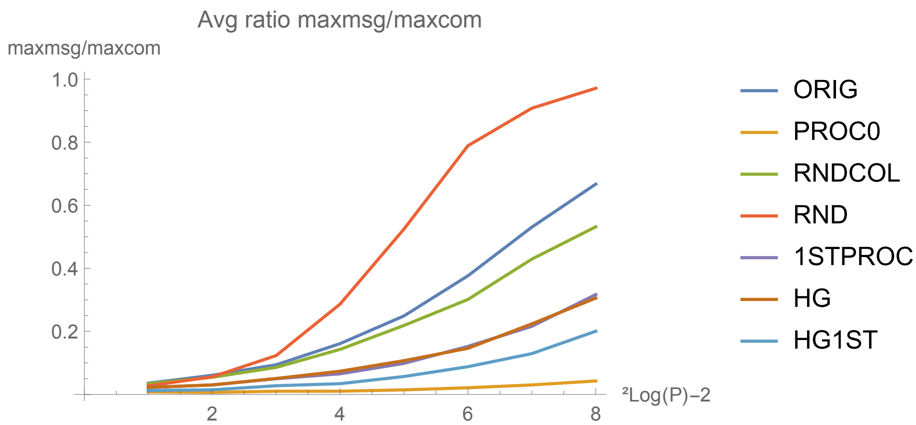

Figure 4: Geometric mean of ratios maxmsg/maxcom for several methods, mean over selected matrices.

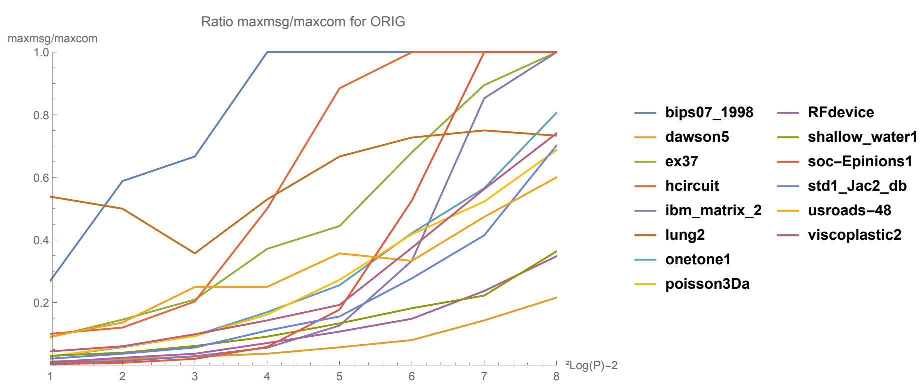

Figure 5: Ratios maxmsg/maxcom for ORIG, selected matrices.

3.5 Large square matrices, $P \leq 8192$

In order to make sure our results also hold for even larger matrices and even larger numbers of processors, we also performed experiments taking values of P till 8192. As the time required to find a matrix partitioning becomes very large in these cases, we could not take lots of different matrices. See Table 11 for a list of these matrices.

We see in Figures 6 and 7 very clearly that the ratio between maxcom and maxmsg goes to unity as P becomes larger and larger. This is the same observation we saw earlier, but this time it is even more pronounced as the number of matrices is larger, we go till higher values of P and the sizes of the matrices are larger.

As maxmsg goes to maxcom as P becomes larger, it means that ORIG will hence also always be the method that will optimize maxmsg. We can hence conclude that in order to really decrease maxmsg, you need to take this into account earlier, during the matrix partitioning.

Matrix	rows	cols	nnz
144	144649	144649	1074393
3dtube	45330	45330	1629474
bcsstk30	28924	28924	1036208
ct20stif	52329	52329	1375396
GaAsH6	61349	61349	1721579
gas_sensor	66917	66917	885141
gupta1	31802	31802	1098006
H2O	67024	67024	1141880
nasasrb	54870	54870	1366097
net100	29920	29920	1031560
pdb1HYS	36417	36417	2190591
pkustk06	43164	43164	1307466
ramage02	16830	16830	1441591
rma10	46835	46835	2374001
ship_001	34920	34920	2339575
smt	25710	25710	1889447
srb1	54924	54924	1508538
TSC_OPF_1047	8140	8140	1012521
tsyl201	20685	20685	1237821
vanbody	47072	47072	1191985
wave	156317	156317	1059331

Table 11: Large square matrices

Matrix	P	Communication Volume							Total Messages							Max messages						
		ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST	ORIG	PROC0	RNDCOL	RND	1STPROC	HG	HG1ST
144	8	6566	11198	6566	11296	6566	11655	9666	39	7	39	56	19	31	26	7	7	7	7	6	7	7
	32	15097	28409	15097	28611	15097	27589	25380	266	31	267	992	141	215	139	21	31	20	31	20	20	23
	128	29418	55997	29418	56200	29418	56580	49837	1207	127	1242	15604	653	988	680	25	127	28	127	26	26	38
	512	54472	101945	54472	101974	54472	101976	89656	5006	511	5212	83884	2782	4171	2843	35	511	34	244	24	23	55
	2048	99467	178420	99467	178403	99467	178401	160183	20130	2047	20762	174469	11792	17365	11758	28	2047	28	135	26	25	43
	8192	182529	300975	182529	300967	182529	301003	256826	78409	8191	78320	300240	50184	52283	39339	21	8191	23	75	24	20	33
	avg	1.	1.81726	1.	1.82212	1.	1.81395	1.61803	1.	0.112533	1.02458	6.39968	0.556342	0.814496	0.574644	1.	11.4297	1.02336	3.30256	0.901955	0.884766	1.40125
ct20stif	8	3650	6220	3650	6241	3650	6415	5549	34	7	34	56	17	27	18	6	7	6	7	4	6	7
	32	7450	13851	7450	13984	7450	13623	12336	216	31	230	991	115	213	123	12	31	12	31	9	15	23
	128	15133	28068	15133	28190	15133	27973	24663	868	127	982	12805	461	784	509	15	127	17	123	14	15	39
	512	31728	56720	31728	56680	31728	56636	49850	3544	511	3958	50583	1898	3380	2014	18	511	19	187	12	17	29
	2048	72630	116304	72630	116275	72630	116339	99418	13862	2047	17776	114638	8277	11713	7764	21	2047	26	109	20	18	30
	8192	157044	207937	157044	207953	157044	207893	173702	66767	8190	80546	207633	36532	42363	29511	25	8190	37	71	34	26	41
	avg	1.	1.70783	1.	1.70487	1.	1.70475	1.48125	1.	0.151713	1.15044	6.67651	0.550109	0.877607	0.546601	1.	16.1934	1.12541	4.20126	0.866333	0.97965	1.71397
gupta1	8	3292	3782	3292	3648	3292	3745	3776	56	7	56	56	25	43	13	7	7	7	7	7	7	7
	32	11057	11847	11057	11680	11057	11693	11760	952	31	983	992	289	604	207	31	31	31	31	31	31	31
	128	27969	34788	27969	34723	27969	34598	32830	10169	127	10179	13805	2005	1824	1368	125	127	127	127	127	127	127
	512	54572	72049	54572	72034	54572	72069	66884	31992	511	30045	62223	7290	6787	5187	182	511	302	329	397	319	428
	2048	99511	126063	99511	126059	99511	126016	114208	65085	2047	65565	123969	20930	22335	16174	347	2047	563	627	951	635	774
	8192	179741	209737	179741	209726	179741	209734	189026	138622	8191	142442	209411	71446	80637	57070	639	8191	667	852	1568	1260	1586
	avg	1.	1.2052	1.	1.19884	1.	1.20064	1.13625	1.	0.0305089	1.01019	1.43903	0.281986	0.36226	0.202132	1.	2.30039	1.19531	1.25559	1.52712	1.33171	1.47564
H2O	8	25402	44748	25402	43060	25402	40778	41306	43	7	44	56	22	32	18	7	7	7	7	6	7	7
	32	50077	87896	50077	88225	50077	86405	80956	339	31	381	992	191	280	194	21	31	20	31	16	21	24
	128	97371	155860	97371	155584	97371	154894	144994	2669	127	2999	16252	1555	1638	1006	45	127	44	127	40	57	71
	512	178275	244047	178275	244081	178275	244065	233906	19507	511	21790	157316	12116	10556	7510	79	511	79	366	83	83	104
	2048	275859	342685	275859	342644	275859	342716	330579	103326	2047	113974	328812	70453	54892	42646	93	2047	116	267	136	117	183
	8192	393649	460614	393649	460621	393649	460539	443093	297624	8190	308859	459013	230164	197041	168126	65	8190	90	149	172	89	228
	avg	1.	1.46157	1.	1.45818	1.	1.44619	1.38001	1.	0.0435886	1.09526	3.34422	0.613016	0.634937	0.432245	1.	5.8103	1.06478	2.22126	1.0867	1.07796	1.60169
gas_sensor	8	4906	9014	4906	8523	4906	8669	8553	21	7	22	56	11	30	11	4	7	4	7	3	7	7
	32	12498	24033	12498	23629	12498	23391	20445	138	31	143	992	69	154	81	9	31	9	31	8	10	11
	128	19956	37905	19956	37879	19956	37864	31442	646	127	731	14429	342	659	399	12	127	14	126	11	12	18
	512	40268	72784	40268	72786	40268	72787	63312	3605	511	4148	63269	1998	3487	2263	17	511	20	182	18	19	28
	2048	76089	130063	76089	130064	76089	130039	111596	15816	2047	17162	128014	8913	13074	9061	19	2047	20	107	23	17	28
	8192	147362	213306	147362	213304	147362	213296	180075	70042	8191	75016	212958	45094	45201	33305	19	8191	23	71	28	22	34
	avg	1.	1.78036	1.	1.76623	1.	1.75493	1.52449	1.	0.174733	1.08803	8.3873	0.549932	0.93344	0.579487	1.	20.5846	1.07811	5.37373	1.03873	1.11651	1.60118
nasasrb	8	1829	1829	1829	1829	1829	1829	1829	12	12	12	12	12	12	12	2	2	2	2	2	2	2
	32	6638	12754	6638	12652	6638	13099	10808	132	31	138	992	69	150	78	8	31	8	31	6	10	11
	128	15147	29363	15147	29350	15147	29467	24642	603	127	654	13420	316	670	414	9	127	9	123	7	13	14
	512	30514	58037	30514	58084	30514	58181	48194	2364	511	2618	51925	1249	2563	1490	10	511	12	151	9	12	19
	2048	69233	120739	69233	120737	69233	120796	99637	10944	2047	12632	118992	6024	10704	6698	16	2047	17	103	15	20	29
	8192	155052	209791	155052	209794	155052	209812	172336	60886	8191	71053	209443	32657	39085	26357	23	8191	31	70	30	33	32
	avg	1.	1.6982	1.	1.69091	1.	1.70592	1.43804	1.	0.231213	1.0949	8.58713	0.562276	1.01058	0.618394	1.	23.833	1.07957	5.86699	0.907092	1.26243	1.50137
pdb1HYS	8	3537	5916	3537	6115	3537	5571	5851	24	7	24	56	12	25	15	5	7	5	7	3	6	7
	32	11692	21731	11692	21748	11692	22017	18879	173	31	191	992	93	174	110	10	31	12	31	9	10	14
	128	28557	50074	28557	49991	28557	50431	44351	1140	127	1228	15268	596	943	511	19	127	19	127	14	17	29
	512	58946	90321	58946	90259	58946	90360	81509	5600	511	7230	75987	3372	4927	2982	28	511	30	230	27	29	41
	2048	129697	165275	129697	165286	129697	165115	150263	29829	2047	40861	161872	17096	19311	13301	44	2047	48	154	43	34	60
	8192	261094	297436	261094	297422	261094	297464	271097	137649	8191	170847	296759	67658	83997	56985	43	8191	68	119	85	36	76
	avg	1.	1.52886	1.	1.53779	1.	1.5319	1.38472	1.	0.111926	1.19014	6.12856	0.549057	0.833717	0.513699	1.	12.7705	1.1534	3.95978	0.968461	0.965849	1.44873
rma10	8	1061	1061	1061	1061	1061	1061	1061	24	24	24	24	24	24	24	4	4	4	4	4	4	4
	32	3600	6937	3600	6915	3600	7118	5744	99	30	102	941	51	138	73	5	30	5	31	5	8	10
	128	9645	18623	9645	18617	9645	18235	15796	514	127	541	10642	267	617	319	8	127	9	114	7	12	13
	512	24801	45197	24801	45206	24801	45217	39017	2530	511	2733	41402	1302	2939	1642	10	511	11	131	8	13	19
	2048	61923	96430	61923	96423	61923	96467	81145	13167	2047	16089	95262	7166	11167	7263	17	2047	20	89	22	17	31
	8192	140681	184929	140681	184931	140681	184887	153993	70352	8190	78290	184705	34195	42619	28810	24	8190	32	74	38	23	36
	avg	1.	1.63546	1.	1.64101	1.	1.63838	1.40487	1.	0.238722	1.08039	7.69912	0.545378	1.05123	0.624859	1.	25.3578	1.09628	5.92693	1.01666	1.2651	1.68037
All	64	1.	1.748	1.	1.75002	1.	1.74622	1.53841	1.	0.118288	1.14075	7.43291	0.494107	0.845082	0.500748	1.	3.6611	1.05652	3.6611	0.844827	1.0817	1.38231
	128	1.	1.72554	1.	1.72464	1.	1.7183	1.51814	1.	0.105191	1.1428	11.4475	0.496036	0.809217	0.492131	1.	6.18817	1.09691	6.03438	0.902995	1.13596	1.54156
	256	1.	1.66457	1.	1.66394	1.	1.66247	1.47207	1.	0.0939776	1.16314	12.861	0.506432	0.791242	0.496303	1.	9.35834	1.07537	6.99712	0.943211	1.05187	1.5471
	512	1.	1.58863	1.	1.58873	1.	1.58883	1.42028	1.	0.0845493	1.15474	10.4561	0.50311	0.77289	0.476471	1.	16.3115	1.1468	6.44748	0.989211	1.05957	1.60354
	1024	1.	1.50491	1.	1.50499	1.	1.50493	1.3455	1.	0.081039	1.18064	7.52709	0.522224	0.761677	0.485929	1.	29.6902	1.20486	5.06803	1.20134	1.12325	1.85458
	2048	1.	1.41471	1.	1.41466	1.	1.41456	1.25844	1.	0.0762988	1.17234	4.93753	0.528035	0.689832	0.461461	1.	51.9481	1.26981	3.73128	1.39014	1.08234	1.82959
	4096	1.	1.32389	1.	1.32389	1.	1.32391	1.16909	1.	0.0752986	1.15831	3.24171	0.537186	0.64447	0.445613	1.	96.9961	1.31589	3.03747	1.71468	1.12021	1.90581
	8192	1.	1.24729	1.	1.24728	1.	1.24728	1.09842	1.	0.0768809	1.08259	2.16709	0.526066	0.630929	0.440848	1.	188.808	1.42084	2.74442	2.02605	1.2529	2.2126

Table 12: Results of large, square matrices, with P going till 8192.

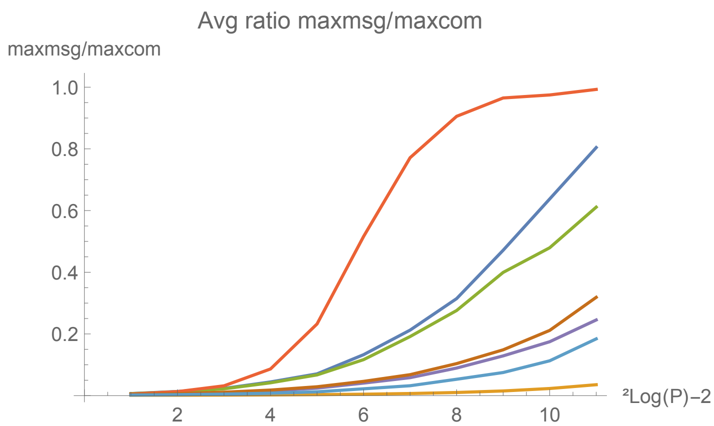

Figure 6: Geometric mean of ratios maxmsg/maxcom for several methods, mean over all large matrices.

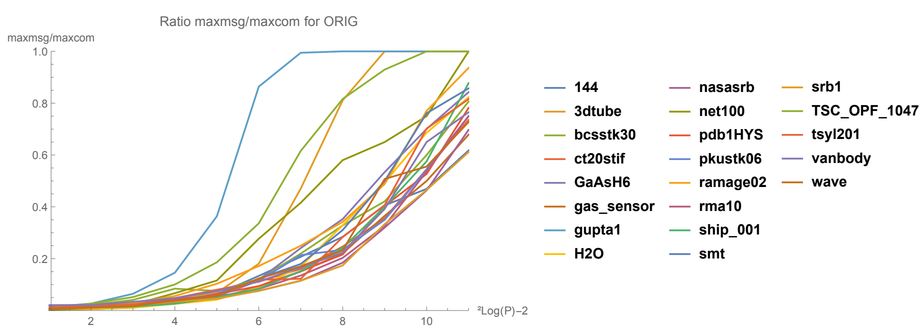

Figure 7: Ratios maxmsg/maxcom for ORIG, all large matrices.

3.6 Remarks concerning results

Symmetric, unsymmetric and also rectangular matrices seem to show the same general trends. Increasing the sizes of matrices or the sparsities do not seem to have a large influence on how the different metrics behave, which we discuss below.

maxmsg

The maxmsg metric doesn't show any significant improvement with the other methods: this because optimizing maxmsg empirically seems equivalent with optimizing maxcom, which is exactly what the current method (ORIG) is doing. The last entries in our tables show that RNDCOL and 1STPROC are the methods which show the least worsening of the maxmsg. If the main objective is minimizing maxmsg, we would recommend using **ORIG**, the method which is currently being used to minimize maxcom.

totmsg

Our 1STPROC method gives quite promising results, showing improvements of around 30 to 40%. Even better results are also seen in the column for HG1ST, though the percentage gain in totmsg is shadowed by a loss in comvol and even larger worsening in maxmsg. Our PROC0 method gives us very high gains, and the gains seem to become larger with increasing values for P. However, the very dramatic increase in maxmsg (and to a lesser extent in comvol) would probably mean that PROC0 is not a viable option. If minimizing totmsg is the main objective, we would recommend using **HG1ST** or **PROC0**.

comvol

As discussed earlier, ORIG, RNDCOL and 1STPROC show the lowest possible comvol. PROC0, RND and HG show surprisingly similar worsenings for the comvol(an increase of around 60 %) , while HG1ST roughly halves this increase. HG1ST was specifically developed as an improvement from HG to lower the comvol and totmsg, and the results seem to validate this. As comvol does not show any variation between **ORIG**, **RNDCOL** and **1STPROC**, any of them could be used if a low comvol is of largest importance.

4 Conclusion

We have seen that by employing our novel method **1STPROC**, we managed to improve our vector distribution by reducing totmsg by around **25%-40%** without compromising on the communication volume. If one is willing to compromise on the communication volume, however, the method **HG1ST** will yield the largest gain in minimizing the total number of messages (reductions of around **50%-55%**). However, as long as we do not have a clear idea of the relative importance of all the different metrics, it is not possible to give good recommendations. It might be possible, for instance, that any worsening of the maxmsg could be considered completely unacceptable, in which case the only option would be to use ORIG. Or a weighted mean between all the different metrics could be considered as an objective function which one would try to minimize. One would need to decide on an individual basis the relative importance of each metric, and use a vector distribution method accordingly.

4.1 Recommendations for future work

In this project, the main goal was to investigate the vector distribution problem of the Mondriaan matrix partitioner, in order to find a new method that would reduce the number of messages. Upon testing our novel methods HG1ST and 1STPROC, we have seen that significant improvements can be booked. However, there are still a few issues which could be investigated in future projects: We still do not know the importance of each metric (comvol, totmsg or maxmsg), and which needs to be optimized depending on the problem at hand. Optimizing one metric does not mean that another will also be optimized, though maxmsg and maxcom seem to be related to each other. We also have not looked what happens when the sizes of the matrices become even larger, like a billion nonzeros, and what happens to matrices with specific structures, like Hermitian or n-diagonal matrices. These could also be researched in future projects.

5 Acknowledgements

I would like to thank my supervisor Prof. Dr. Rob H. Bisseling for his valuable insights and comments. I would also like to thank the second reader, Dr. Paul A. Zegeling, for his remarks and suggestions.

References

[1] U. Catalyurek, C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Dist. Systems, 10(7)*, pages 673–693, 1999.

[2] Rob H. Bisseling. *Parallel Scientific Computation: A Structured Approach using BSP and MPI.* Oxford University Press, 2004.

[3] Rob H. Bisseling and Wouter Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*, pages 47–65, 2005.

[4] Umit V. Catalyurek. Hypergraph models for sparse matrix partitioning and reordering, 1999. PhD Thesis Bilkent University.

[5] Timothy A. Davis. The university of Florida sparse matrix collection. `http://www.cise.ufl.edu/research/sparse/matrices/`. (Date last accessed 1-10-2015).

[6] E. Boman, U. Catalyurek, C. Chevalier, K. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring. *Scientific Programming 20(2)*, pages 129–150, 2012.

[7] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Chichester, U.K.: Wiley, 1990.

[8] Andrew T. Ogielski and William Aiello. Sparse matrix computations on parallel processor arrays. *SIAM Journal on Scientific Computing*, pages 519–530, 1993.

[9] Erik G. Boman, Karen D. Devine, S. Rajamanickam. Scalable matrix computations on large scale-free graphs using 2D graph partitioning. *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013.

[10] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar. Multilevel hypergraph partitioning: application in VLSI domain. *Proc 34th Design Automation Conference*, pages 526–529, 1997.

[11] Bora Ucar and Cevdet Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM J. Sci. Comput. 25(6)*, pages 1837–1859, 2004.

[12] B. Vastenhouw and R.H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review 47(1)*, pages 67–95, 2005.