An Evaluation of the Zoltan Parallel Graph and Hypergraph Partitioners

Sivasankaran Rajamanickam[1] and Erik G. Boman[1]

Sandia National Laboratories, NM, USA,
{srajama,egboman}@sandia.gov

Abstract. Graph partitioning is an important and well studied problem in combinatorial scientific computing, and is commonly used to reduce communication in parallel computing. Different models (graph, hypergraph) and objectives (edge cut, boundary vertices) have been proposed. For large problems, the partitioning itself must be done in parallel. Several software packages, such as ParMetis, PT-Scotch and Zoltan are widely available. In this paper we evaluate the performance of the parallel graph and hypergraph (PHG) partitioner in the Zoltan toolkit. For the first time, we compare the performance of PHG as a graph and hypergraph partitioner across a diverse set of graphs from the 10th DIMACS implementation challenge.

Keywords: graph partitioning, parallel computing

1 Introduction

Graph partitioning is a well studied problem in combinatorial scientific computing. An important application is the mapping of data and/or tasks on a parallel computer, where the goals are to balance the load and to minimize communication [7]. There are several variations of graph partitioning, but they are all NP-hard problems. Fortunately, good heuristic algorithms exist. Naturally, there is a trade-off between run-time and solution quality. In parallel computing, partitioning may be performed either once (static partitioning) or many times (dynamic load balancing). In the latter case, it is crucial that the partitioning itself is fast. Furthermore, the rapid growth of problem sizes in scientific computing dictates that partitioning algorithms must be scalable. The multilevel approach developed in the 1990s [1, 6, 12] provides a good compromise between run-time (complexity) and quality. Software packages based on this approach (Chaco [8], Metis [9], and Scotch [13]) have been extremely successful. Even today, all the major parallel software packages for partitioning in scientific computing (ParMetis [10], PT-Scotch [14], and Zoltan [5]) use variations of the multilevel graph partitioning algorithm.

The 10th DIMACS implementation challenge offers an opportunity to evaluate the state-of-the-art in partitioning software in 2011. This is a daunting task, as there are several variations of the partitioning problem (e.g., objectives), several software codes, and a large number of data sets. In this paper we limit the scope in the following ways: We only consider parallel software since our focus is high-performance computing. We focus on the Zoltan toolkit since its partitioner can be used to minimize either the edge cut (graph partitioning) or the communication volume (hypergraph partitioning). We include some baseline comparisons with ParMetis, since that is the most widely used parallel partitioning software. We limit the experiments to a subset of the DIMACS graphs. One may view this paper as an updated version of the IPDPS'06 paper that introduced the Zoltan PHG partitioner [5].

2 Models and Metrics

The term "graph partitioning" can refer to several different problems. Most often, it refers to the edge cut metric, though in practice the communication volume metric is often more important. For the latter objective, it is useful to extend graphs to hypergraphs. Here, we review the different models and metrics and explain how they relate.

2.1 Graph Models

Given an undirected graph $G = (V, E)$, the classic version of graph partitioning is to partition V into k disjoint subsets (parts) such that all the parts are approximately the same size and the total number of edges between parts are minimized. More formally, let $\Pi = \{\pi_0, \dots, \pi_{k-1}\}$ be a balanced partition such that

$$|V(\pi_i)| \leq (1+\epsilon)\,\frac{|V|}{k} \quad \forall i, \tag{1}$$

for a given $\epsilon > 0$. The edge cut problem (EC) is then to minimize the cut set

$$C(G, \Pi) = \{\{(u, v) \in E\} | \Pi(u) \neq \Pi(v)\}\,. \tag{2}$$

There are straight-forward generalizations for edge weights (minimize weighted cuts) and vertex weights (balance is weighted).

Most algorithms and software attempt to minimize the edge cut. However, several authors have shown that the edge cut does not represent communication in parallel computing [2, 7]. A key insight was that the communication is proportional to the *vertices* along the part boundaries, not the cut edges. A more relevant metric is therefore the communication volume, which roughly corresponds to the boundary vertices. Formally, let the communication volume for part p be

$$comm(\pi_p) = \sum_{v \in \pi(p)} (\lambda(v, \Pi) - 1)\,, \tag{3}$$

where $\lambda(v, \Pi)$ denotes the number of parts that v or any of its neighbors belong to, with respect to the partition Π.

We then obtain the following two metrics:

$$CV_{max}(G, \Pi) = \max_{p} comm(\pi_p) \tag{4}$$

$$CV_{sum}(G, \Pi) = \sum_{p} comm(\pi_p) \tag{5}$$

In parallel computing, this corresponds to the maximum communication volume for any process and the total sum of communication volumes, respectively.

2.2 Hypergraph Models

A *hypergraph* $H = (V, E)$ extends a graph since now E denotes a set of hyperedges. An hyperedge is any non-empty subset of the vertices V. A graph is just a special case of a hypergraph where each hyperedge has cardinality two (since a graph edge always connects two vertices). Hyperedges are sometimes called *nets*, a term commonly used in the (VLSI) circuit design community.

Analogous to graph partitioning, one can define several hypergraph partitioning problems. As before, the balance constraint is on the vertices. Several different cut metrics have been proposed. The most straight-forward generalization of edge cut to hypergraphs is:

$$C(H, \Pi) = \{\{e \in E\} | \Pi(u) \neq \Pi(v) \texttt{where} u \in e, v \in e\} \, . \tag{6}$$

However, a more popular metric is the so-called $(\lambda - 1)$ metric:

$$CV(H, \Pi) = \sum_{e \in E} (\lambda(e, \Pi) - 1) \, , \tag{7}$$

where $\lambda(e, \Pi)$ is the number of distinct parts that contain any vertex in e.

While graphs are restricted to structurally symmetric problems (undirected graphs), hypergraphs make no such assumption. Furthermore, the number of vertices and hyperedges may differ, making the model suitable for rectangular matrices. The key advantage of the hypergraph model is that the hyperedge $(\lambda - 1)$ cut (CV) accurately models the total communication volume. This was first observed in [2] in the context of sparse matrix-vector multiplication. The limitations of the graph model were described in detail in [7]. This realization led to a shift from the graph model to the hypergraph model. Today, many partitioning packages use the hypergraph model: PaToH [2], hMetis [11], Mondriaan [15], and Zoltan-PHG [5].

Hypergraphs are often used to represent sparse matrices. For example, using row-based storage (CSR), each row becomes a vertex and each column becomes a hyperedge. Other hypergraph models exist: in the "fine-grain" model, each non-zero is a vertex [3]. For the DIMACS challenge, all input is symmetric and given as undirected graphs. Given a graph $G(V, E)$, we will use the following derived hypergraph $H(V, E')$: for each vertex $v \in V$, create an hyperedge $e \in E'$ that contains v and all its neighbors. In this case, it is easy to see that $CV(H, \Pi) = CV_{sum}(G, \Pi)$. Thus, we do not need to distinguish between communication volume in the graph and hypergraph models.

2.3 Which is More Relevant?

Most partitioners minimize either the total edge cut (EC) or the total communical volume (CV-sum). A main reason for this choice is that algorithms for these metrics are well developed. Less work has been done to minimize the maximum communication volume (CV-max), though in a parallel computing setting this may be more relevant as it corresponds to the max communication for any process.

In order to compare the three metrics and how they correspond to the actual performance in an application we use a GMRES iteration as the "application". We used the matrices from the UF sparse matrix collection group of the DIMACS challenge for these tests. We use no preconditioner as the performance characteristics will be different based on whether the preconditioner is expensive or not. Instead our goal is to compare just the matrix-vector multiply time in the GMRES iteration. As there is no preconditioner and some of these problems are ill-conditioned we might not converge at all, so we report the solve time for 1000 iterations for the three options: no partitioning, graph partitioning with ParMetis, hypergraph partitiong with Zoltan hypergraph partitioner. The results are shown in table 1. We repeated the experiments thrice and used the median for our results. We can see from the results that it is important to do partitioning to balance the load and minimize the communication (except in the af_shell problem). We also observe that hypergraph partitioning does slightly better than graph partitioning in terms of solve time. However, for these symmetric problems the difference between graph and hypergraph partitioning is small in terms of real performance gain in the application. We will show in section 4.2 that the partitioners actually differ in terms of the measured performance metrics for the problems shown in Table 1. However, we do not see that difference in the metrics translate to measurable real performance gain in the time for the matrix-vector multiply.

Table 1. Solve Time for 1000 iterations of GMRES for different partitioning options

Matrix Name	No Partitioning	ParMetis	Zoltan PHG
audikw1	8.81	3.22	3.01
nlpkkt120	8.45	6.18	6.00
G3_circuit	2.76	1.59	1.55
af_shell10	2.61	2.74	2.79

3 Zoltan PHG

Zoltan was originally designed as a toolkit for dynamic load-balancing [4]. It included several geometric partitioning algorithms, plus interfaces to external (third-party) graph partitioners, such as ParMetis. Later, a native parallel hypergraph partitioner (PHG) was developed [5] and added to Zoltan. While PHG was designed for hypergraph partitioning, it can also be used for graph partitioning but it is not optimized for this use case. (Note: “PHG” now stands for Parallel Hypergraph and Graph partitioner.)

Zoltan PHG is a parallel multilevel partitioner, consisting of *coarsening*, *initial partitioning*, and *refinement* phases. The coarsening scheme is a variation of *heavy connectivity* or *inner product* matching, where we match (merge) vertices that have many neighbors in common. If A is the edge-vertex incidence matrix of the hypergraph, then A^TA is the vertex similarity matrix used for the matching (aggregation). Note that we do not compute A^TA explicitly (which would be both expensive and require a lot of memory), but rather compute selected entries as needed. This is often the most expensive part of the entire partitioning code. We believe it is not necessary to compute the vertex similarities between all pairs of vertices to do the matching, and indeed PHG has options to reduce the work by giving preference to local data. However, we decided to use only the default option (full matching) in our experiments. One improvement added after [5] is that we allow more than two vertices to merge into a coarse vertex. We have observed that this improves the partition quality in many cases. The initial partition is a randomized greedy heuristic. The refinement is the Fidduccia-Matheyses (FM) method for the communication volume metric. In parallel, exact FM is too costly so instead we use a simplified version of FM. Although the parallel version may yield worse quality results than the serial (true) FM, we have observed that in practice the difference is fairly small.

Zoltan PHG uses recursive bisection to partition into k parts. Note that k can be any integer greater than one, and does not need to be a power of two. Also, Zoltan can run on p processes, where $k \neq p$. However, the typical use case is $k = p$.

A novel feature of Zoltan PHG is that internally, the hypergraph is mapped to processes in a 2D block fashion. That is, the processes are logically mapped to a p_x by p_y grid, where $p = p_x p_y$. The advantage of this design is to reduce communication. Instead of expensive all-to-all or any-to-any communication, all communication is limited to process rows or columns. The drawback of this design is that there are more synchronization points than if an 1D distribution had been used. For further details on PHG, we refer to [5]. The basic algorithm remains the same, though several improvements have been made over the years.

When PHG is used as a graph partitioner, each hyperedge is of size two. When we coarsen the hypergraph, only vertices are coarsened, not hyperedges. This means that the symmetry of graphs is destroyed already after the first level of coarsening. We conjecture that PHG is not particularly efficient as a graph partitioner because it does not take advantage of the special structure of graphs (in particular, symmetry). Still, we believe it is fair to compare PHG as a graph partitioner because it uses exactly the same code as the hypergraph partitioner, so any performance difference is due to the model not the implementation.

4 Experiments

4.1 Software, Platform, and Data

Our primary goal is to study the behavior of Zoltan PHG as a graph and a hypergraph partitioner, using different objectives and a range of data sets. We use Zoltan 3.5 (Trilinos 10.8). Our test program was based on the test driver **zdrive**. We used ParMetis 4.0 as a reference and compare it with Zoltan PHG.

Our compute platform was Hopper, a Cray XE6 at NERSC. Hopper has 6,384 compute nodes, each with 24 cores (two 12-core AMD MagnyCours) and 32 GB of memory. The 10th DIMACS challenge has an abundance of test graphs, and it was impractical to run all the test data. Instead, we selected five test families that are relevant to the computational problems we have encountered at Sandia. Within each family, we selected (up to) five graphs. Usually, we picked the largest ones, though we tried to avoid graphs that were too similar. In addition we picked four graphs two each from the street networks and clustering instances to compile our 22 test problems. The test problems we used are listed in Table 2.

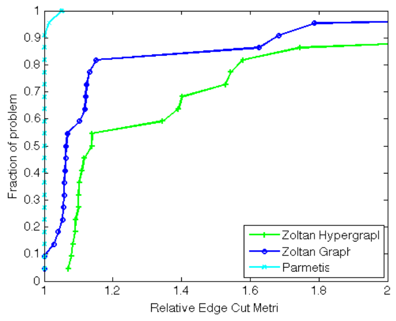
Table 2. Test Matrices from the Challenge problems used in our experiments

Matrix Name	DIMACS Group	Vertices	Edges
af_shell10	UF matrix	1508065	25582130
G3_circuit	UF matrix	1585478	3037674
cage15	UF matrix	5154859	94044692
audikw1	UF matrix	943695	38354076
nlpkkt120	UF matrix	3542400	46651696
auto	Walshaw	448695	3314611
matrix144	Walshaw	144649	1074393
wave	Walshaw	156317	1059331
fe_ocean	Walshaw	143437	409593
m14b	Walshaw	214765	1679018
venturiLevel3	Numerical	4026819	8054237
packing	Numerical	2145852	17488243
channel	Numerical	4802000	42681372
hugbubbles-0020	Dynamic Frames	21198119	31790179
hugetrace-0020	Dynamic Frames	16002413	23998813
hugetric-0020	Dynamic Frames	7122792	10680777
coPapersDBLP	Coauthor	540486	15245729
coPapersCiteseer	Coauthor	434102	16036720
asia.osm	Streets	11950757	12711603
europe.osm	Streets	50912018	54054660
road_central	Clustering	14081816	16933413
road_usa	Clustering	23947347	28854312

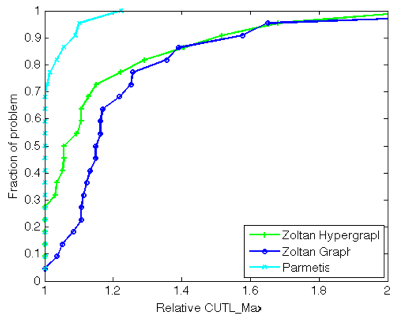
We partitioned the graphs into 16, 64, 256, and 1024 parts. In the parallel computing context, this covers everything from a multicore workstation to a small-sized parallel computer. Except where stated otherwise, we ran the partitioner on the same number of cores as the target number of parts.

Due to limited compute time on Hopper, we ran each partitioning test only once. Zoltan uses randomization, so results may vary from run to run. However, for large graphs, we found that the random variation is relatively small. We believe it is fair to draw conlusions based on several data sets, though one should be cautious about overinterpreting any single data point.

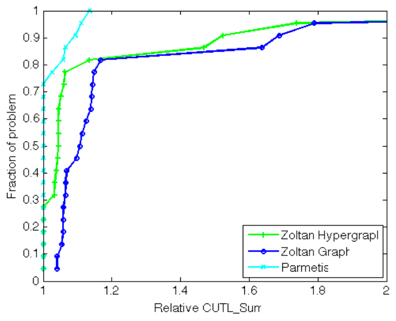
4.2 Zoltan vs. ParMetis



(a) Edge Cut



(b) Communication Volume (Max)



(c) Communication Volume (Sum)

Fig. 1. Zoltan Vs Parmetis: Comparing Zoltan's partitioning with graph and hypergraph model with Parmetis for symmetric problems for 256 parts and 256 MPI processes.

In this section, we compare Zoltan's graph and hypergraph partitioning with ParMetis's graph partitioning. We partition the graphs into 256 parts with 256 MPI processes. The results for the three metrics total edge cut (EC), the maximum communication volume (CV-max) and the total communication volume

(CV-sum) are shown in Figure 1. We present the performance profile for the 22 matrices for the three metrics.

Note that the main advantage of the hypergraph partitioners is the ability to handle unsymmetric problems and to reduce the communication volume for such problems directly (without symmetrizing the problems). However, the DIMACS challenge problems are all symmetric problems. We take this opportunity to compare against the graph partitioners even for symmetric problems.

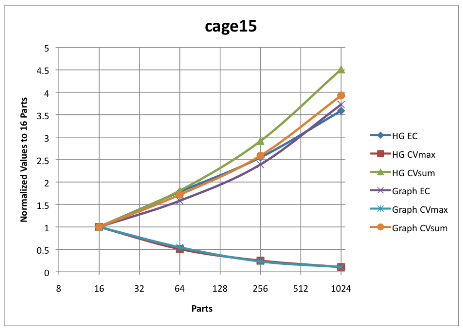
In terms of the edge cut metric ParMetis does better than Zoltan for 20 of the matrices and Zoltan's graph model does better for just two matrices. However, Zoltan's graph model is within 15% of ParMetis's edge cuts for 82% of the problems (see Figure 1(a)). The four problems that cause trouble to Zoltan's graph model are the problems from the street networks and clustering instances.

In terms of the CV-sum metric Zoltan's partitioning with the hypergraph model, is able to do better than Zoltan's graph model in all the instances, and is better than ParMetis for 33% of the problems, and is within 6% or better of CV-sum of the ParMetis for another 44% of the problems (see Figure 1(c)). Again the street networks and the clustering instances are the ones that cause problems for the hypergraph partitioning. In terms of the CV-max metric Zoltan's hypergraph partitioning is better than the other two methods for 27% of the problems, and within 15% of the CV-max for another 42% of the problems (see Figure 1(b)).

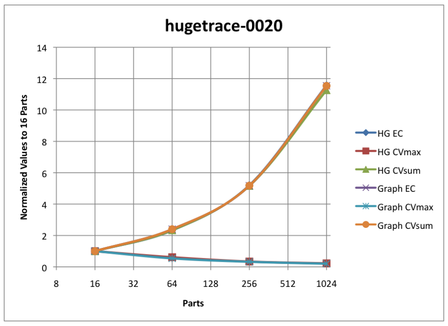
From our results, we observe that even for symmetric problems hypergraph partitioners can perform nearly as well as (or even better than) the graph partitioners depending on the problems and the metrics one cares about. We also note that four of these 22 instances come from the same problems we used in Section 2.3 and Zoltan does better in two problems and ParMetis does better on other two problems in terms of the CV-max metric. In terms of EC metric ParMetis does better for all these four problems. However, as we can see from Table 1 the actual solution time is slightly better when we use the hypergraph partitioning (for the three problems with performance improvements) irrespective of which method is better in terms of the metrics we compute. To be precise, we should again note that the differences in actual solve time between graph and hypergraph partitioning are minor for those four problems. We would like to emphasize that we are not able to observe any difference in the performance of the actual application when the difference in the metrics is a small percentage. We study the characteristics of Zoltan's graph and hypergraph partitioning in the rest of this paper.

4.3 Zoltan graph vs. hypergraph model

We did more extensive experiments on the DIMACS problems with the graph and hypergraph partitioning of Zoltan. For each problem from Table 2 we compute the three metrics (EC, CV-max, CV-sum) for part sizes 16, 64, 256, 1024. All the experiments use the same number of MPI processes as the part sizes. All the data from these test runs are in Tables 4 and 5. We can observe from these results that based on the EC metric, Zoltan's graph partitioning is the best for most problems. In terms of the CV-sum metric the hypergraph partitioning fares

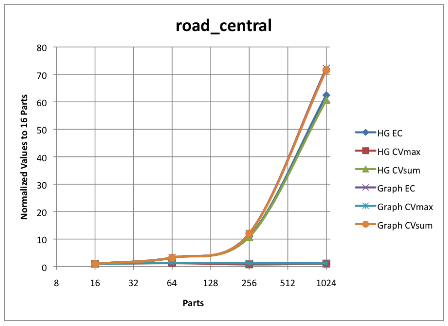


(a)

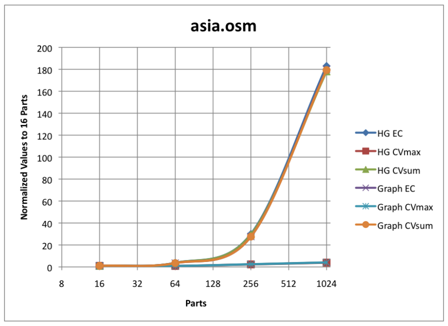


(b)

Fig. 2. Comparing Zoltan's partitioning with graph and hypergraph model Quality metrics for different part sizes.



(a)



(b)

Fig. 3. Comparing Zoltan's partitioning with graph and hypergraph model Quality metrics for different part sizes.

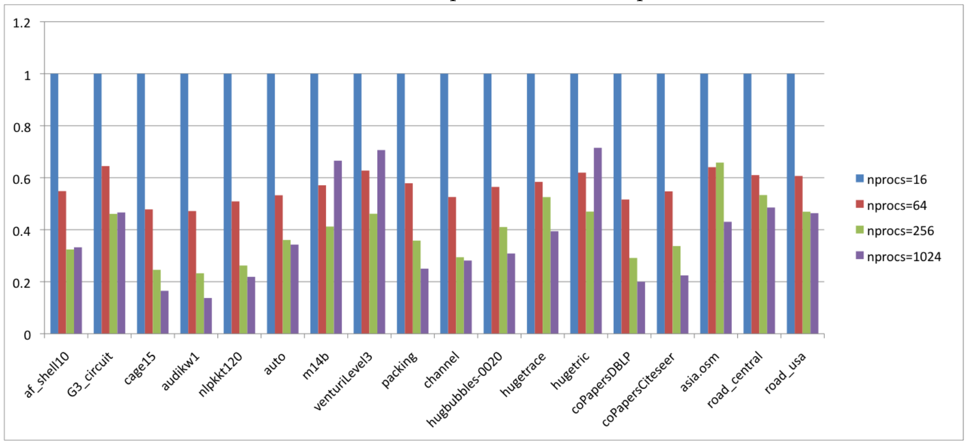
better. Neither of the algorithms optimize, CV-max metric and as expected the results are mixed for this metric.

We show the scalability of the three metrics for graph and hypergraph partitionings for two problems – cage15 and hugetrace-0020 – in Figure 2. We normalize the metrics with respect to the values for the 16 parts case in these figures. These scalability results are for the "good" problems and from the results we can see why we call these problems the "good" problems – EC and CV-sum go up by a factor of 3.5 to 4.5 when going from 16 parts to 1024 parts. In contrast, we also show the scalability of the metrics from one problem from the street networks and clustering set each (road_central and asia.osm) in Figure 3. Note that for the some of these problems the metrics scale with similar values that the lines overlap in the graph. These second set of problems are challenging for both our graph and hypergraph partitioners as EC and CV-max go up by a factor 60-70 going from 16-1024 (for road_central).

4.4 Zoltan scalability

A lot of Zoltan's users use Zoltan within their parallel applications dynamically. As a result it is important for Zoltan to be a scalable parallel hypergraph partitioner. We have made several improvements within Zoltan over the past few years and we evaluate our parallel scalabilty for the DIMACS problems instances in this section. Note that being a parallel hypegraph partitioner also enables us to solve large problems than does not fit into the memory of a compute node in a scalable way. However, we were able to partition all the DIMACS instances except the matrix europe.osm with 16 cores. We omit, the europe.osm matrix and three small matrices from the walshaw group that get partitioned within two seconds even with 16 cores, from these tests. The scalability results for the rest of the 18 matrices are shown in Figure 4. We normalize the time for all the runs with time to compute 16 parts. Note that even though the matrix size remains the same, this is not a strong scaling test as we compute 1024 parts in 1024 MPI processes case.

Fig. 4. Scalability of Zoltan Hypergraph Partitioning time for DIMACS challenge matrices normalized to the time for 16 MPI processes and 16 parts.



Even with the increase in the amount of work for large matrices like cage15 and hugebubbles-0020 we see performance improvements as we go to 1024 MPI processes. However, for smaller problems like the auto or m14b the performance remains flat (or degrades) as we go from 256 MPI processes to 1024 MPI processes.

The scalability of Zoltan's graph partitioners is shown in Figure 5. We see that the graph partitioner tends to scale well for most problems. However, the hypergraph partitioner is faster than our graph partitioner in terms of actual execution time for lot of problems. The actual timings for the 18 problems are presented in Table 6 and Table 7 as Appendix A.

Fig. 5. Scalability of Zoltan Graph Partitioning time for DIMACS challenge matrices normalized to the time for 16 MPI processes and 16 parts.

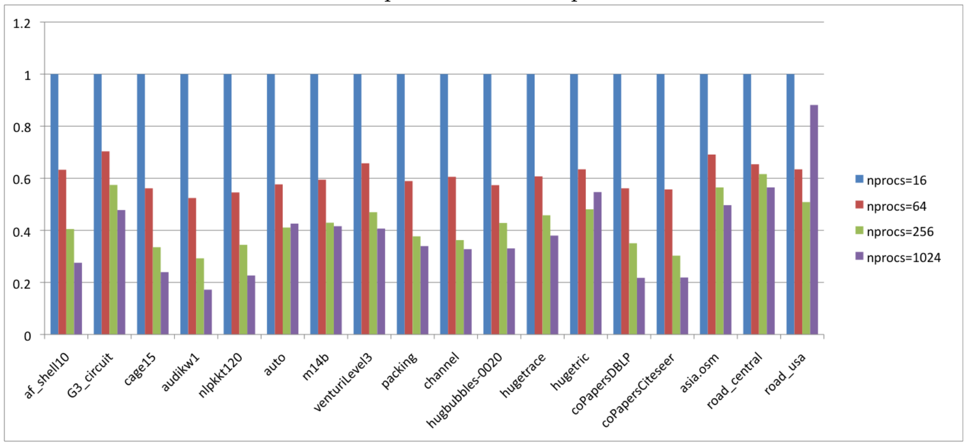


Table 3. Scalability of Metrics: Computing 1024 parts using 24 and 1024 MPI processes for Zoltan's hypergraph partitioning

	MPI procs =1024		MPI Procs = 24	
	nparts = 1024		nparts = 1024	
Matrix Name	Number of Cut Edges	CV-Sum	Number of Cut Edges	CV-Sum
af_shell10	2305000	484380	1885000	393175
G3_circuit	168000	244327	167000	230438
cage15	15000000	11040631	9350000	12245702
audikw1	16250000	2007867	10150000	1402269
nlpkkt120	9600000	2490467	7300000	2805320
auto	705000	407832	640000	392719
matrix144	312500	191915	284500	186071
wave	304500	204647	302500	203246
fe_ocean	119000	119483	81000	145516
m14b	440500	246973	386500	230171
venturiLevel3	224500	274551	226500	253868
packing	1775000	889457	1685000	841273
channel	4595000	2185454	4450000	2093403
hugbubbles-0020	235000	437720	220500	412565
hugetrace-0020	199000	372846	188000	354133
hugetric-0020	133500	247709	123500	231937
coPapersDBLP	8800000	2180871	9200000	2211715
coPapersCiteseer	7400000	1137207	7700000	1151086
asia.osm	63500	123418	60000	116363
europe.osm	119500	236043	116500	230746
road_central	137000	257598	127000	239519
road_usa	148500	280049	145000	272844

4.5 Partitioning on a single node to improve quality

As we discussed before Zoltan can compute a partitioning statically with fewer MPI processes for arbitrary number of parts. This usually results in better quality than the dynamic approach. However, the users have to retain the partition in this case for future use. We evaluate this case for the symmetric matrices from the DIMACS collection for just the hypergraph partitioning. We compute 1024 parts with 24 MPI processes. Our assumption is that the user will be willing to devote one compute node to compute the partition he needs. We present the results of these experiments for our 22 matrices in Table 3. On an average the edge cuts gets reduced by 10% and the CV-sum gets reduced by 4% when we compute 1024 parts with just 24 MPI processes instead of using the 1024 MPI processes. This confirms our conjecture that using fewer cores (MPI processes) gives higher quality results, and raises the question: Could we improve on-node partitioning further by developing a shared-memory partitioner?

5 Conclusions

We have evaluated the parallel performance of Zoltan PHG, both as a graph and hypergraph partitioner on test graphs from the DIMACS challenge data set. We also made comparisons to ParMetis, a popular graph partitioner. We observed that ParMetis consistently obtained best edge cut (EC), while the results for communication volume (CV) was mixed. The difference between the partitioners is small for most of the test problems. As expected, PHG as a graph partitioner was better at minimizing the edge cut (EC) while PHG as a hypergraph partitioner was better at minimizing the communication volume (CV). We used the example problems to show that small difference in the quality metrics usually do not translate into application performance. We also showed that Zoltan is scalable on large number of processors (up to at least 1024).

In our opinion, the DIMACS graph partitioning challenge was biased towards graph partitioners (expected – given the name graph partitioning challenge) because only undirected (symmetric) graphs were included. Many real-world problems are unsymmetric (e.g. web link graphs, circuit simulation problems). Even when some such data was included, it had been symmetrized in some way. Usually, it is better to work with the unsymmetric data directly. Hypergraph partitioners are better at handling unsymmetric or rectangular data. Thus it is remarkable that a code like PHG that was designed for such cases, also did comparable to (or slightly worse than) graph partitioners on symmetric data. Thus, we believe Zoltan PHG is a good universal partitioner for all types of input data.

Acknowledgements

We thank Karen Devine for helpful discussions. We thank the Department of Energy's Office of Science, the Advanced Scientific Computing Research (ASCR)

Table 4. Comparison of the three metrics for Zoltan graph and hypergraph partitioning

		Hypergraph			Graph		
Matrix Name	nprocs	EC	CV_{Max}	CV_{Sum}	EC	CV_{Max}	CV_{Sum}
af_shell10	16	218500	3535	43840	195000	3340	39185
	64	456000	2175	92065	453000	2575	331290
	256	1025000	1365	209125	950000	1220	194180
	1024	2305000	945	484380	1935000	605	403085
G3_circuit	16	14300	1942	21538	11100	2030	21638
	64	31100	1179	47165	27050	1261	51124
	256	77500	769	113315	68000	894	124381
	1024	168000	449	244327	151000	496	278351
cage15	16	4180000	248414	2448287	2885000	312195	3166527
	64	7450000	126508	4427386	4570000	170636	5444892
	256	10650000	61678	7134164	6900000	72176	8193941
	1024	15000000	26551	11040631	10750000	33550	12438234
audikw1	16	1710000	17103	155421	1460000	16629	156375
	64	3795000	8707	344496	3085000	8430	345462
	256	7600000	4659	734685	5950000	4282	719280
	1024	16250000	3192	2007867	10300000	2230	1434570
nlpkkt120	16	1705000	33587	380658	1520000	44413	507311
	64	3325000	17405	739689	2685000	20098	890990
	256	5800000	8667	1400878	4645000	8718	1599229
	1024	9600000	4149	2490467	7500000	3944	2735928
auto	16	96500	4401	48443	97000	5097	51200
	64	207000	2530	106127	205000	3050	111579
	256	388500	1230	209196	376500	1292	214646
	1024	705000	681	407832	645000	585	398028
matrix144	16	46900	2041	23854	46750	2297	25079
	64	96000	1254	50687	91500	1696	51033
	256	176000	679	98349	167500	724	98699
	1024	312500	338	191915	286500	345	186927
wave	16	53000	2334	28957	52500	2555	30167
	64	100500	1256	57271	98500	1375	58962
	256	180500	641	109870	177000	619	112661
	1024	304500	320	204647	289000	336	204295
fe_ocean	16	15800	1329	14634	10450	1618	15925
	64	35950	764	33351	27100	1040	41725
	256	69000	404	68627	51000	470	80115
	1024	119000	182	119483	86000	233	148394
m14b	16	54500	2560	25878	54500	2434	26941
	64	117500	1526	57882	113500	1325	59052
	256	229000	870	118509	220000	912	120075
	1024	440500	466	246973	390000	399	233121
venturiLevel3	16	17250	1853	21272	14650	2181	25249
	64	44050	1291	53783	36800	1423	62250
	256	229000	870	120128	220000	912	143500
	1024	224500	539	274551	188000	568	313911
packing	16	242500	10442	114760	232000	11098	114920
	64	525000	7329	252639	510000	8472	254186
	256	960000	3880	467618	945000	3819	478043
	1024	1775000	2164	889457	1695000	1893	882719
channel	16	695000	22441	299236	680000	23767	299903
	64	1435000	14416	631783	1390000	12535	622254
	256	2610000	6722	1185657	2540000	6464	1174761
	1024	4595000	3458	2185454	4485000	3060	2184724

Table 5. Comparison of the three metrics for Zoltan graph and hypergraph partitioning

		Hypergraph			Graph		
Matrix Name	nprocs	EC	CV_{Max}	CV_{Sum}	EC	CV_{Max}	CV_{Sum}
hugebubbles-0020	16	19750	3071	38474	21250	3475	42482
	64	45000	1790	87341	46800	1957	93598
	256	103000	1347	195311	106000	1333	212458
	1024	235000	794	437720	239500	849	479382
hugetrace-0020	16	17200	3191	33101	17500	3237	35032
	64	40000	1951	77276	42000	1753	84014
	256	89500	1074	170884	90500	1042	181392
	1024	199000	704	372846	202000	647	404158
hugetric-0020	16	11300	1955	22039	13050	2563	26098
	64	27250	1171	52309	28250	1362	56494
	256	60500	682	113981	61000	774	121534
	1024	133500	432	247709	136500	510	273330
coPapersDBLP	16	4490000	64230	660961	1630000	67653	812638
	64	6050000	30089	1020146	2160000	35879	1180556
	256	7350000	14752	1416774	2955000	16958	1613693
	1024	8800000	7105	2180871	3740000	7548	2099688
coPapersCiteseer	16	2555000	30840	310742	925000	32074	365210
	64	3285000	13935	439622	1290000	16428	504211
	256	4550000	8380	635241	1935000	9629	726315
	1024	7400000	4901	1137207	3185000	4495	959723
asia.osm	16	347	84	693	372	101	744
	64	1280	78	2526	1290	92	2583
	256	10400	196	20408	10300	247	20552
	1024	63500	329	123418	67000	414	133366
europe.osm	16	-	-	-	-	-	-
	64	5000	313	9968	4865	299	9724
	256	20850	385	41456	21350	361	42653
	1024	119500	517	236043	121500	630	242535
road_central	16	2195	414	4245	2195	466	4383
	64	7100	520	13562	7100	612	14176
	256	24150	337	45905	26600	556	52741
	1024	137000	447	257598	158500	554	313740
road_usa	16	2545	597	4971	2250	414	4487
	64	7200	545	13906	7350	467	14606
	256	28100	480	53570	28950	479	57603
	1024	148500	608	280049	160000	685	316701

office, and the National Nuclear Security Administration's ASC program for financial support. Sandia is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the DOE under Contract No. DE-AC02-05CH11231.

References

1. T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452. SIAM, 1993.
2. Ü. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Dist. Systems*, 10(7):673–693, 1999.
3. Ü. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *Proc. IPDPS 8th Int'l Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001)*, April 2001.
4. Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.
5. K.D. Devine, E.G. Boman, R.T. Heaphy, R.H. Bisseling, and U.V. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE, 2006.
6. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*. ACM, December 1995.
7. Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519 – 1534, 2000.
8. Bruce Hendrickson and Robert Leland. The chaco user's guide, version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, 1993.
9. G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Dept. Computer Science, University of Minnesota, 1995. *http://www.cs.umn.edu/˜karypis/metis*.
10. G. Karypis and V. Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Dept. Computer Science, University of Minnesota, 1997. `http://www.cs.umn.edu/~metis`.
11. George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain. In *Proc. 34th Design Automation Conf.*, pages 526 – 529. ACM, 1997.
12. George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.
13. F. Pelligrini. SCOTCH 3.4 user's guide. Research Rep. RR-1264-01, LaBRI, Nov. 2001.
14. F. Pelligrini. PT-SCOTCH 5.1 user's guide. Research rep., LaBRI, 2008.
15. Brendan Vastenhouw and Rob H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, 2005.

A Timings of Zoltan Graph and Hypergraph partitioner

Table 6. Time in Seconds to compute the 16, 64, 256 and 1024 parts using Zoltan's Hypergraph partitioner for DIMACS problems

Matrix Name	nprocs=16	nprocs=64	nprocs=256	nprocs=1024
af_shell10	25.4	13.93	8.23	8.44
G3_circuit	12.39	7.99	5.71	5.78
cage15	263.55	126.04	64.76	43.5
audikw1	55.63	26.25	12.93	7.64
nlpkkt120	103.71	52.79	27.2	22.68
auto	7.38	3.93	2.66	2.53
m14b	2.96	1.69	1.22	1.97
venturiLevel3	32.45	20.36	14.97	22.92
packing	40.31	23.33	14.44	10.09
channel	108.5	57.04	31.92	30.53
hugebubbles-0020	238.9	134.9	98	73.74
hugetrace	155.5	90.8	81.7	61.3
hugetric	63.34	39.24	29.75	45.29
coPapersDBLP	36.1	18.64	10.52	7.22
coPapersCiteseer	33.8	18.5	11.4	7.58
asia.osm	67.6	43.3	44.5	29.1
road_central	117.2	71.5	62.5	56.9
road_usa	180.3	109.4	84.6	83.6

Table 7. Time in Seconds to compute the 16, 64, 256 and 1024 parts using Zoltan's Graph partitioner for DIMACS problems

Matrix Name	nprocs=16	nprocs=64	nprocs=256	nprocs=1024
af_shell10	46.16	29.2	18.69	12.71
G3_circuit	11.09	7.8	6.37	5.3
cage15	234.94	131.89	78.76	56.26
audikw1	97.1	50.9	28.41	16.72
nlpkkt120	155.62	84.9	53.61	35.26
auto	8.69	5.01	3.57	3.7
m14b	3.75	2.23	1.61	1.56
venturiLevel3	29.9	19.66	14.04	12.16
packing	45.95	27.08	17.32	15.6
channel	113.76	68.88	41.23	37.28
hugebubbles-0020	200.3	114.9	85.8	66.22
hugetrace	131.9	80.1	60.4	50.1
hugetric	53.44	33.9	25.7	29.23
coPapersDBLP	34.72	19.49	12.17	7.55
coPapersCiteseer	35	19.5	10.6	7.66
asia.osm	60.2	41.6	34	29.9
road_central	95.6	62.5	58.9	54
road_usa	156.1	99	79.4	137.6