A Scalability and Sensitivity Study of Parallel Geometric Algorithms for Graph Partitioning

Shad Kirmani*, Hongyang Sun†, Padma Raghavan†
*eBay Inc. Brisbane, CA, USA
†Vanderbilt University, Nashville, TN, USA

***Abstract*—Graph partitioning arises in many computational simulation workloads, including those that involve finite difference or finite element methods, where partitioning enables efficient parallel processing of the entire simulation. We focus on parallel geometric algorithms for partitioning large graphs whose vertices are associated with coordinates in two- or three-dimensional space on multi-core processors. Compared with other types of partitioning algorithms, geometric schemes generally show better scalability on a large number of processors or cores. This paper studies the scalability and sensitivity of two parallel algorithms, namely, recursive coordinate bisection (denoted by pRCB) and geometric mesh partitioning (denoted by pGMP), in terms of their robustness to several key factors that affect the partition quality, including coordinate perturbation, approximate embedding, mesh quality and graph planarity. Our results indicate that the quality of a partition as measured by the size of the edge separator (or cutsize) remains consistently better for pGMP compared to pRCB. On average for our test suite, relative to pRCB, pGMP yields 25% smaller cutsizes on the original embedding, and across all perturbations cutsizes that are smaller by at least 8% and by as much as 50%. Not surprisingly, higher quality cuts are obtained at the expense of longer execution times; on a single core, pGMP has an average execution time that is almost 10 times slower than that of pRCB, but it scales better and catches up at 32-cores to be slower by less than 20%. With the current trends in core counts that continue to increase per chip, these results suggest that pGMP presents an attractive solution if a modest number of cores can be deployed to reduce execution times while providing high quality partitions.**

***Index Terms*—Recursive coordinate bisection; geometric mesh partitioning; graph embedding**

I. Introduction

Long-running computational modeling and simulation workloads are typically executed on supercomputers using a large number of processors or cores to reduce the execution time that may otherwise be prohibitive. Many such workloads can be modeled as graphs, which are then partitioned recursively to provide a mapping of the application to the processors [26]. A key step concerns partitioning a graph into two subgraphs so that they ideally have equal size to balance the workloads and the fewest number of edges connecting them (small separator size or *cutsize*) to reduce data movement or communication.

Graph partitioning has been extensively studied by the literature. For general graphs, partitioning has been shown to be NP-complete [13], and hence many heuristic solutions have been developed (e.g., [1], [6], [18], [25]). When additional information is available, partitioning can become somewhat easier. For instance, in scientific computing, many problems have associated coordinate information, such as when numerically solving partial differential equations (PDEs) on a mesh, and this is when geometric partitioning comes into play.

Geometric partitioning algorithms work on graphs whose vertices have associated coordinates in two or three dimensions, such as those involving finite difference or finite element methods. Parallel algorithms using line or plane separators, such as recursive coordinate bisection (RCB) [3], cartesian nested dissection (CND) [15] or Multi-Jagged [8] have been shown to provide good partition quality and scale well; the Zoltan library [4] for parallel RCB is well-established and used widely. Another algorithm called geometric mesh partitioning (GMP) that uses sphere separators was shown to be provably good for certain classes of well-shaped finite-element meshes [14], [23].

One limitation of geometric algorithms, however, is that not all graphs have associated coordinate information. For such graphs, multi-level partitioning algorithms, such as ParMetis [19] and Pt-Scotch [25], have good performance on a modest number of cores but do not scale well on higher core counts. Recently, the ScalaPart algorithm [20] was developed to combine the parallel scalability of RCB or CND with the higher partition quality typical of GMP and a broader applicability to arbitrary graphs without coordinates. ScalaPart is a multi-level method with graph embedding to impart coordinates, followed by geometric mesh partitioning with sphere separators and a geometric variant of the Fiduccia-Mattheyses method [11] to refine the separators and reduce cutsizes.

In this paper, we focus on graphs that have coordinates or associated embeddings to study the scalability and sensitivity of two parallel geometric algorithms RCB and GMP (henceforth denoted by pRCB and pGMP, respectively). We consider several factors that could impact their performance. For instance, the coordinates of a graph's vertices may originate from a physical process or a scientific simulation, and these coordinates may be subject to perturbations due to errors in measurement or the need to refine the mesh during the simulation. We are interested in how well parallel geometric algorithms scale with the number of cores in a system, and how their partition qualities are affected by coordinate perturbation and approximate embedding, as well as by other factors such as mesh quality and graph planarity. Such a study is instrumental in characterizing the parallel performance and robustness of the algorithms as well as their quality-performance trade-offs. The results will help not only in better utilizing these existing algorithms but also in the design of new parallel geometric schemes.

We conduct extensive experiments on a 32-core cluster using a set of sparse graph benchmarks with force-based embedding

and finite-element meshes. We compare the performance of pRCB as implemented in Zoltan, and pGMP as used in Scala-Part. The following summarizes our main findings:

- ***Scalability***: pRCB is much faster than pGMP, especially on a single core (where pRCB is almost 10 times faster). This is due to its efficient implementation in Zoltan and RCB's simpler line/plane separators to partition the graphs. However, pGMP scales well with increasing number of cores and catches up at 32 cores to be slower by less than 20%.
- ***Sensitivity***: pGMP is generally more robust to perturbations of the graph geometry due to the use of more sophisticated sphere separators. Specifically, pGMP consistently outperforms pRCB in terms of the cutsize by at least 8% and by as much as 50% for different graph perturbations starting from an original embedding where pGMP is on average 25% better.

These results suggest that pGMP presents an attractive alternative to pRCB and a robust solution for partitioning large graphs from a modest number of cores with competitive parallel performance and high partition quality.

The rest of this paper is organized as follows. Section II provides the relevant background and a brief overview of the related work. Section III develops our evaluation methodology by discussing the factors that could affect the partitioning quality. The performance evaluation of pRCB and pGMP with respect to these factors are presented in Section IV. We summarize our findings and remark on future directions in Section V.

II. Background and Related Work

Many graph partitioning algorithms have been developed, including multi-level methods, spectral methods and geometric methods, as well as their combinations [9]. This section gives a background of these methods and surveys the related work with an emphasis on the geometric algorithms.

A. Graph Partitioning Problem

First, we briefly describe the graph partitioning problem to be considered: Given an unweighted and undirected graph $G = (V, E)$, where V is the set of vertices, E is the set of edges. For geometric algorithms, a set C of two- or three-dimensional coordinates corresponding to the set of vertices is also provided. We focus on sparse graphs with $|V| = N$ and $|E| = M$ such that M is a small constant times N. Using a graph partitioning algorithm, we would like to partition the set of vertices V into two disjoint subsets V_1 and V_2 of nearly equal size such that the total number of edges connecting the vertices in V_1 and the vertices in V_2 is minimized. In other words, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \phi$, and $|V_1| \approx |V_2| \approx \frac{1}{2}|V|$, with an edge separator S whose cutsize $|S|$ is as small as possible, where $S = \{(i,j)|(i,j) \in E$, and $i \in V_1, j \in V_2\}$.

B. Brief Overview of Different Partitioning Algorithms

Graph partitioning using *multi-level methods* has been proven to be very successful. These methods coarsen a graph in multiple steps, each of which preserves the graph's global structure, and the coarsest graph is partitioned. The graph is then uncoarsened, and after each uncoarsening step, the partition is refined till a partition is obtained for the original graph [16]. ParMetis [19] and Pt-Scotch [25] are two popular implementations of parallel multi-level partitioning algorithms. *Spectral methods* provide another popular approach to partition graphs with many practical applications, such as computer vision and VLSI circuit design. These methods rely on computing the eigenvectors of the Laplacian matrix that correspond to the graph for defining a partition [10], [12], and this involves solving a relaxed constraint optimization problem. Spectral methods can produce high-quality partitions but are computationally expensive for computing the eigenvectors for large graphs [28]. This paper focuses on *geometric methods*, which work for graphs that have coordinates associated with the vertices in 2D/3D spaces [3], [4], [15], such as scientific computing problems that involve finite difference or finite element methods. If a graph does not already have associated coordinates, graph embedding can be applied to assign coordinates to them [17], [20]. The assigned coordinates can then be used to partition the graph. The rest of this section gives more details on embedding and geometric methods.

C. Force-Based Graph Embedding

For graphs that do not have associated coordinates, *force-based embedding* has been proven useful in providing coordinates to these graphs, such as the one proposed by Hu [17], which calculates attractive and repulsive forces for all the vertices of a graph and moves them iteratively in the direction of the net force. The attractive forces are applied only between the neighbors while the repulsive force of a vertex is exerted by all the other vertices in the graph. Specifically, for a graph $G = (V, E)$, the attractive force on a vertex $i \in V$ is $F_a(i) = \sum_{(i,j)\in E} \frac{||c_i - c_j||^2}{K}$ while the repulsive force is $F_r(i) = -\sum_{j\in V, j\neq i} \frac{CK^2}{||c_i - c_j||}$, where c_i and c_j represent the coordinates of vertices i and j, respectively, $||c_i - c_j||$ represents the geometric distance between the two vertices, and C and K are empirical constants [17]. Note that computing the repulsive forces is in fact N-body type of calculations and can therefore be approximated by using the Barnes-Hut approach [2].

D. Recursive Coordinate Bisection

One geometric algorithm to partition a graph with associated or embedded coordinate information is to find the median of the coordinates in one of the dimensions. The vertices on one side of the median are assigned to one partition and the vertices on the other side are assigned to the other partition. If more partitions are required, this step is repeated recursively on each of the partitions. This algorithm, despite its simplicity, provides good results and is called recursive coordinate bisection (RCB) [3]. The Zoltan library [4] has implemented a parallel version of RCB to partition graphs with coordinate information, and the execution times have been shown to be less than those required by multi-level algorithms, such as ParMetis and Pt-Scotch. The partition quality produced by the multi-level methods, however, is generally better than the one produced by RCB.

E. Geometric Mesh Partitioning

Another geometric algorithm called geometric mesh partitioning (GMP) has been shown to compute provably good partitions under certain assumptions regarding the mesh [14], [23]. GMP constructs sphere separators (as opposed to line or plane separators used in RCB) by projecting the vertices of a graph onto the surface of a sphere in a higher dimension, computing a center point, selecting a random great circle through the center point and then projecting this great circle back to the original coordinate space to define a partition. A serial Matlab implementation of GMP is provided by [14], [23]. Recently, the ScalaPart algorithm [20] implemented a parallel multi-level version of GMP by combining it with Fiduccia-Matheyeses refinement [11] on the geometric band around the partition. It provides very high quality partitions in parallel over a large number of cores. Performing refinement in the band around the partition was first proposed in a multi-level algorithm by Pellegrini and Roman as part of the Scotch package [25], but the band they calculated was based on graph distance whereas ScalaPart computes the band using geometric distance.

III. Evaluation Methodology

This section presents the methodology used to evaluate the performance of the following two parallel geometric partitioning algorithms.

- *pRCB*: a parallel version of the recursive coordinate bisection algorithm [3] implemented in the Zoltan library [4]. It partitions a graph using line or plane separators.
- *pGMP*: a parallel version of the geometric mesh partitioning algorithm [14], [23] developed as the ScalaPart algorithm [20]. It partitions a graph using multiple sphere separators and refines the partitions based on a geometric variant of the Fiduccia-Mattheyses method [11].

We motivate the study by discussing several factors in the following that could affect their performance.

A. Number of Tries in pGMP and Performance Scalability

We first consider a parameter specific to pGMP, which uses a number of tries and on each try produces a different sphere separator. The ScalaPart algorithm [20] uses 7 tries by default and out of them it selects the top 3 in terms of cutsize and performs Fiduccia-Matheyeses refinement [11] in the band around the partitions. The best of the 3 partitions in terms of cutsize after the refinement is selected as the final partition.

In general, the quality of a partition produced by geometric mesh partitioning algorithms can be affected by the number of tries [14], [23]. Since pGMP is built upon these earlier work, it should also be sensitive to the number of tries. On the one hand, a larger number of tries produces more separators, increasing the search space and hence the probability of finding a better quality partition. On the other hand, using a larger number of tries increases the amount of computation and communication, and hence the execution time. In particular, for every try, pGMP first computes a partition in parallel and then evaluates the quality of that partition (also in parallel) by calculating the cutsize. The latter requires all the vertices to know on which side of the partition their neighbors lie. Since a vertex and its neighbors can be distributed across different processors, calculating the cutsize involves communication among the processors.

Given the trade-off between the partition quality and execution time, we would like to evaluate the impact of the number of tries on pGMP. Moreover, the execution times and hence speedups of both pGMP and pRCB should be affected by the number of cores. Hence, we are interested in answering the following questions: *What is the impact of the number of tries on the performance of pGMP? How does the parallel performance of both pGMP and pRCB scale with the number of cores?*

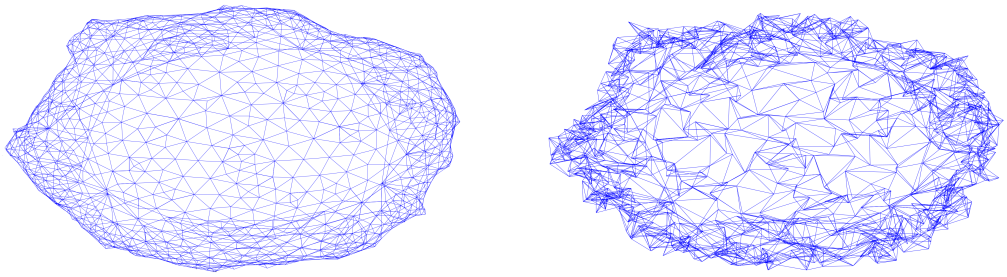
B. Geometry of Graphs

We now discuss several factors concerning the geometry of the graphs that could affect the partitioning algorithms.

1) Coordinate perturbation: Recall that geometric partitioning is only feasible for graphs that have coordinate information. In many scientific computing problems related to physical processes or scientific simulations, the graphs already have coordinates associated with them, such as when numerically solving partial differential equations using finite difference or finite element methods. For these problems, the coordinates associated with the vertices of a graph are sometimes subject to perturbations due to errors in measurement or the need to refine the mesh during the simulation. An illustration of a graph with associated coordinates and its perturbed coordinates are shown in Figure 1. The original coordinates are obtained using force-based embedding [17] and the perturbed coordinates are created by moving each vertex by a small constant displacement in a random direction. Ideally, we would like a partitioning algorithm to preserve the partition quality even for graphs with largely perturbed coordinates. Hence, we are interested in the answer to the following question: *How sensitive are pRCB and pGMP to perturbations of the graphs' coordinates?*

2) Approximate embedding: Although some graphs do not have coordinate information available, we can still use geometric partitioning algorithms by assigning coordinates to their vertices with an embedding algorithm, as demonstrated in ScalaPart [20]. However, graph embedding can be an expensive pre-processing step, even if done in parallel [20]. The most popular method to embed a graph in 2D/3D spaces is via force-based embedding, which requires a number of iterations to balance the repulsive and attractive forces on each vertex of the graph, as explained in Section II-C. To reduce the computation cost, approximations are often used for calculating the forces, such as applying Barnes-Hut type of space decomposition, coarsening the graph and stopping at a certain number of iterations. In such cases, we would have to deal with graphs whose coordinates are not optimally embedded. Similarly to the coordinates perturbation scenario stated previously, we are interested in the answer to the following question: *How robust are the parallel geometric partitioning algorithms to the non-optimal coordinates obtained from an approximate embedding algorithm?*

3) Mesh quality: For many problems in scientific computing, such as the ones using finite element or finite difference methods, the graph is often represented as a 2D mesh. Sometimes, the mesh generated for solving these problems may not be of very high quality [21], [22], thus we have to deal with



(a) Original coordinates (b) Perturbed coordinates

Fig. 1: Original coordinates for a graph obtained using force-based embedding [17] and its perturbed coordinates.

bad quality meshes and partition them for parallel processing. Triangle meshes are the most popular kind of meshes used. In an ideal triangle mesh, all three sides of each triangle should be of the same length, i.e., the mesh is made of equilateral triangles. In reality, getting such an ideal mesh is difficult if not impossible. We use a simple measure for the quality of a mesh triangle known as the *edge ratio* [24], defined as the ratio of the shortest edge and the longest edge in the triangle. The overall mesh quality is then defined as the average edge ratio of all triangles in the mesh. Hence, for an ideal triangle mesh, this average ratio should be one. A smaller number for the average edge ratio implies a mesh of poorer quality. We are interested in answering the following question: *How does the mesh quality affect the partition qualities of pRCB and pGMP?*

4) Graph planarity: Finally, the planarity of a graph plays an important role in graph embedding and partitioning. It has been found that partitioning graphs that are non-planar is substantially more difficult than partitioning planar graphs [5]. For some graphs in particular, such as social network graphs, extra edges may be dynamically added to the existing set of edges, which may require the graph to be re-embedded or re-partitioned. These extra edges could make a planar graph non-planar or increase its non-planarity. We are interested in the performance of the parallel geometric partitioning algorithms on these types of graphs. In other words, *what is the impact of additional edges (that turn a planar graph into a non-planar one or increase its non-planarity) on the performance of pRCB and pGMP?*

IV. Experimental Results

In this section, we evaluate the performance of the pRCB and pGMP algorithms and study the effects of different factors described in the previous section. Experiments are conducted using graphs from the following two test suites:

- *UFL sparse matrix collection* [7]: The graphs in this test suite are listed in Table I. The number of vertices ranges from 1 to 21 millions and the number of edges ranges from 5 to 100 millions. This test suite contains structural problems from both 2D and 3D spaces, and is used in all but the mesh quality and graph planarity experiments.
- *Triangle mesh graphs*: Three mesh graphs are created by the Triangle mesh generator [27], and they are called twoholes, mechanic, square_with_hole from [24] as listed in Table II. The number of elements in this test suite is less than the UFL test suite, and it is used in the mesh quality and graph planarity experiments.

TABLE I: Graphs from the UFL Sparse matrix collection [7].

Graph	$N(10^6)$	$M(10^6)$
ecology1	1	4.99
ecology2	0.99	4.99
delaunay_n20	1.05	6.29
G3_circuit	1.58	7.66
kkt_power	2.06	12.77
hugetrace-00000	4.59	13.76
delaunay_n23	8.39	50.33
delaunay_n24	16.77	100.66
hugebubbles-00020	21.20	63.58

TABLE II: Three mesh graphs from [24].

Mesh	N	M
mechanic	12618	49959
twoholes	11337	45071
square_with_hole	116559	461076

Since both partitioning algorithms are geometric, they require coordinates for the vertices of the graphs. All graphs in the UFL test suite are provided with coordinates using force-based multi-level graph embedding [17]. The experiments are conducted on a cluster of servers with two quad-core Intel Nehalem processors (Intel Xeon X5550 processor, 2.66GHz, 32GB RAM, 32KB L1 cache, 256KB L2 cache per core, 8MB shared L3 cache). Four servers are used interconnected by QDR infiniband providing up to 32 cores.

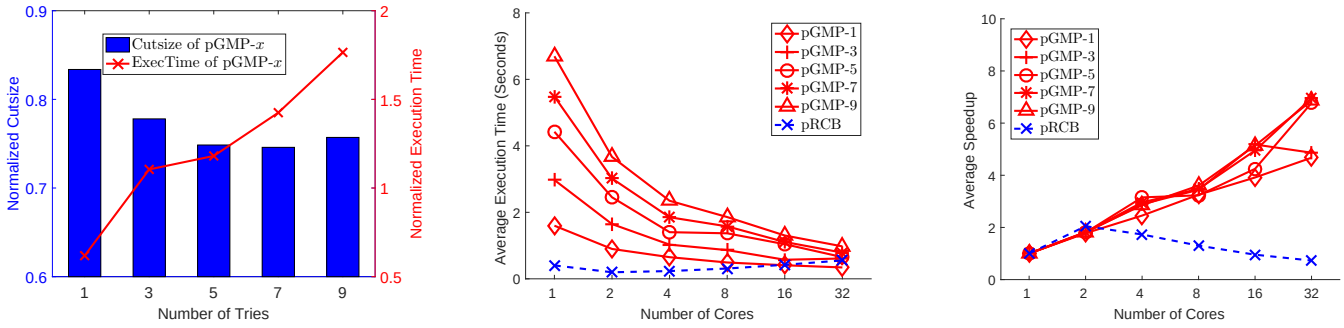
While the quality of the partitions should be measured by both the cutsize and the balance of the partitions, most applications can tolerate a slight partition imbalance from the ideal if it leads to a smaller cutsize. Consequently, we report only the cutsizes as the quality measure, and allow a small imbalance of at most 1% in the partition size.

A. Impact of Number of Tries on pGMP

To study the impact of number of tries on the performance of pGMP, we experiment with 1, 3, 5, 7 and 9 tries, which have the corresponding numbers of sphere separators. We denote the resulting algorithm by pGMP-x, where x denotes the number of tries.

TABLE III: Cutsizes and execution times (in seconds) of pRCB and pGMP-x on 32 cores for the UFL test suite.

	pRCB (cutsize, exectime)	pGMP-1 (cutsize, exectime)	pGMP-3 (cutsize, exectime)	pGMP-5 (cutsize, exectime)	pGMP-7 (cutsize, exectime)	pGMP-9 (cutsize, exectime)
ecology1	(1091, 0.324)	(1260, 0.14)	(1065, 0.278)	(1076, 0.35)	(1073, 0.308)	(1100, 0.392)
ecology2	(1060, 0.284)	(1106, 0.128)	(1254, 0.249)	(1063, 0.191)	(1051, 0.213)	(1038, 0.34)
delaunay_n20	(2922, 0.208)	(2813, 0.241)	(2456, 0.297)	(2040, 0.218)	(1970, 0.328)	(2285, 0.298)
G3_circuit	(1372, 0.646)	(1147, 0.232)	(1149, 0.384)	(1257, 0.253)	(1142, 0.339)	(1162, 0.427)
kkt_power	(48301, 0.386)	(28945, 0.254)	(22586, 0.585)	(24376, 0.489)	(22451, 0.469)	(22806, 0.598)
hugetrace-00000	(1013, 0.506)	(730, 0.411)	(737, 0.522)	(696, 0.842)	(747, 0.896)	(814, 1.045)
delaunay_n23	(8275, 0.733)	(5923, 0.565)	(5889, 0.909)	(6374, 1.305)	(6897, 1.731)	(6292, 2.068)
delaunay_24	(13328, 1.253)	(11388, 0.954)	(10343, 2.386)	(8004, 2.871)	(8382, 3.858)	(8148, 4.667)
hugebubbles-00020	(2373, 2.188)	(1803, 1.136)	(1622, 2.275)	(1729, 3.782)	(1729, 5.753)	(1686, 7.883)
Geometric mean	(3392, 0.553)	(2828, 0.342)	(2639, 0.611)	(2539, 0.653)	(2530, 0.788)	(2568, 0.976)
Normalized	(1.00, 1.00)	(0.83, 0.62)	(0.78, 1.1)	(0.75, 1.18)	(0.75, 1.42)	(0.76, 1.76)



(a) (b) (c)

Fig. 2: (a) Normalized cutsizes and execution times of pGMP-x with different numbers of tries on 32 cores. (b)-(c) Performance scalability of pRCB and pGMP-x with different numbers of tries when executed on up to 32 cores.

Table III presents the cutsizes and execution times of pRCB and pGMP-x (with different numbers of tries) on 32 cores for the UFL test suite. The last two rows show the geometric means of the cutsizes and execution times for all algorithms across all graphs, as well as the normalized geometric means with respect to those of pRCB. Here, geometric mean is used because it provides an unbias average over all test graphs with large numerical ranges. Figure 2(a) further plots the normalized cutsizes and execution times of pGMP-x as a function of the number x of tries on 32 cores. All reported execution times refer to the time to partition the graphs only without counting the embedding time (which is common to both algorithms).

For the partition quality, we can see that pGMP-1, which has just one try and thus one sphere separator, already improves the cutsize by 17% on average compared to pRCB. Moreover, the cutsize of pGMP-x improves further when the number of tries increases, although the marginal gain becomes smaller. Indeed, the cutsize of pGMP-7 is only slightly smaller than that of pGMP-5. The cutsize of pGMP-9 worsens because of the randomness in the execution and selection of sphere separators. Overall, pGMP-5 gives 25% improvement over pRCB, and improves over pGMP-1 by 9.6%.

For the execution time, pGMP-1 with only one try runs faster than pRCB on 32 cores. Since it also produces a smaller cutsize, this demonstrates that pGMP-1 is more effective than pRCB. As the number of tries increases, the execution time of pGMP-x monotonically increases and becomes more than that of pRCB. In particular, pGMP-x takes on average 18% more time to partition the graphs with 5 tries, and significantly more time with larger number of tries.

B. Performance Scalability

Figure 2(b) shows the scalability of pRCB and pGMP-x (in terms of average execution time) on up to 32 cores. We can see that pRCB runs much faster on a single core, but the gap between the two algorithms reduces with increasing number of cores. On a sequential execution, pGMP-x is 3x-15.5x slower than pRCB depending on the number of tries. This is because pRCB uses line/plane separators to compute the partitions, which runs more efficiently than the more complex sphere separators used by pGMP-x. As the number of cores increases, the execution time of pGMP-x reduces dramatically while pRCB is not significantly affected. On 32 cores, pGMP-5 is slower than pRCB by only 18%, while pGMP-1 and pGMP-3 become even faster than pRCB. Figure 2(c) further plots the average speedups of both algorithms with respect to their sequential executions. pGMP-x scales reasonably well with increasing number of cores and eventually achieves 5x-7x speedups on 32 cores depending on the number of tries. On the other hand, pRCB does not scale well on this test suite, which is largely due to its already very fast sequential implementation as well as the higher parallel overhead incurred by data movement and communication (during all-reduce steps). The result is not surprising given that fixed problem speedups on large core counts is often adversely impacted by data movement required to support parallel execution and NUMA effect.

TABLE IV: Cutsizes of pRCB and pGMP under different coordinate perturbations for the UFL test suite.

Perturbation	0%		10%		20%		50%		100%		200%	
	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP
ecology1	1091	1076	1087	1113	1097	1146	1151	1140	1241	1106	1574	1068
ecology2	1060	1063	1064	1199	1076	1141	1136	1130	1250	1135	1510	1066
delaunay_20	2922	2040	2920	1949	2938	2295	2996	2288	3277	2227	4257	2735
G3_citcuit	1372	1257	1393	1340	1389	1273	1529	1132	1856	1357	2680	1448
kkt_power	48301	24376	48725	16328	49767	18655	55065	18361	67921	18087	96248	17994
hugetrace-00000	1013	696	1014	753	1013	742	1027	854	1071	749	1354	704
delaunay_23	8275	6374	8289	7262	8299	5468	8493	6385	9295	6277	12651	6720
delaunay_24	13328	8004	13492	9422	13632	8375	14363	8475	15647	8192	18983	10999
hugebubbles-00020	2373	1729	2415	1688	2445	1745	2567	1711	2796	1751	3298	1883
Geometric mean	3392	2539	3413	2573	3438	2529	3617	2567	4028	2554	5204	2712
Normalized	1.00	0.75	1.01	0.76	1.01	0.75	1.07	0.76	1.19	0.75	1.53	0.80

TABLE V: Cutsizes of pRCB and pGMP under different approximate embeddings for the UFL test suite.

Approx. embedding	100%		70%		55%		40%	
	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP
ecology1	1091	1076	1519	1655	1603	1625	1656	1596
ecology2	1060	1063	1483	1591	1601	1602	1821	1754
delaunay_20	2922	2040	2992	2596	3092	2725	3019	2805
G3_citcuit	1372	1257	2244	2108	2350	2038	2497	2090
kkt_power	48301	24376	45584	35895	46797	33170	45606	28546
hugetrace-00000	1013	696	1138	1076	1102	1064	1140	1159
delaunay_23	8275	6374	9748	7298	9305	8250	9145	8382
delaunay_24	13328	8004	13161	10644	13091	11231	13646	12101
hugebubbles-00020	2373	1729	2314	2350	2410	2319	2545	2286
Geometric mean	3392	2539	3948	3600	4033	3631	4163	3687
Normalized	1.00	0.75	1.16	1.06	1.19	1.07	1.23	1.09

Scalability Result*: Despite the larger execution time on one core, pGMP-x (with an appropriate number of tries) can be competitive with pRCB on as few as 32 cores, thus making it an attractive solution when high quality partitions are desired.*

Given the above result as well as the result from the last section, we will use pGMP-5 and 32-core executions in all subsequent experiments for the sensitivity study.

C. Performance Sensitivity to Geometry of Graphs

1) Impact of perturbation of coordinates: We first study the impact of coordinates perturbation on the quality of the partitions. In this experiment, we start with the graphs in the UFL test suite with original coordinates imparted to them by force-based embedding [17], which is considered to have 0% perturbation. We then perturb the coordinates by moving each vertex in a random direction by 10%, 20%, 50%, 100% and 200% of the average edge length from the original embedding.

Table IV shows the cutsizes of pRCB and pGMP under different coordinates perturbations. The last two rows show the geometric mean for all graphs, and the normalized value with respect to that of pRCB under 0% perturbation. We observe that the average cutsizes computed by both algorithms are not much affected by perturbations less than 50%. However, the partition quality of pRCB starts to degrade quickly after the perturbation reaches 50%, whereas the average cutsize of pGMP does not increase much even when the perturbation increases to 200%. Specifically, pGMP has a performance degradation of merely 6.7% at 200% perturbation, whereas pRCB suffers from 53% degradation. Moreover, at 200% perturbation, pGMP improves upon pRCB by 48% on average, and it is still 20% better than pRCB on unperturbed graphs.

Sensitivity Result 1*: The partition qualities of both pGMP and pRCB are insensitive to small coordinates perturbations, but pGMP is more robust to large perturbations than pRCB.*

2) Impact of approximate embedding: We now study the robustness of the geometric partitioning algorithms to approximate graph embeddings. In this experiment, we again use the UFL test suite. To obtain approximate embeddings, we stop the force-based embedding prematurely at a certain number of iterations before convergence. Specifically, we obtain 3 sets of approximate embeddings for the graphs by stopping the embedding algorithm at 70%, 55% and 40% of the number of iterations required to get the original embedding.

Table V shows the impact of approximate embeddings on the quality of the partitions. The last two rows report the geometric mean for all graphs, and the normalized value with respect to that of pRCB under the original embedding. As expected, the partition qualities of both algorithms become worse when the embedding algorithm is stopped prematurely. We observe that the quality of pRCB keeps dropping with fewer embedding iterations. In particular, the cutsize increases by 16%, 19% and 23% at 70%, 55% and 40% of the original number of iterations, respectively. The quality of pGMP, on the other hand, experiences a substantial degradation at 70% of the original number of iterations, but it remains relatively stable after that. Specifically, the cutsize increases by 41.3%, 42.7% and 45.3% at 70%, 55% and 40% of the original number of

TABLE VI: Cutsizes of pRCB and pGMP for the original mesh test suite ($q = 0$) and their shifted versions ($q = 1, 2, 3$).

Mesh quality	$q = 0$		$q = 1$		$q = 2$		$q = 3$	
	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP
mechanic-q	196	86	195	140	213	152	243	252
twoholes-q	208	202	215	201	237	248	263	236
square_with_hole-q	407	421	411	384	453	449	504	423
Geometric mean	255	194	258	221	284	257	318	293
Normalized	1.00	0.76	1.01	0.87	1.11	1.01	1.25	1.15

TABLE VII: Three original meshes ($q = 0$) and their shifted versions ($q = 1, 2, 3$) with respective average edge ratios.

	Average edge ratio			
Mesh quality	$q = 0$	$q = 1$	$q = 2$	$q = 3$
mechanic-q	0.74	0.63	0.48	0.41
twoholes-q	0.74	0.63	0.47	0.40
square_with_hole-q	0.73	0.62	0.47	0.40

iterations, respectively. However, the quality of pGMP is still better under all approximate embeddings, with improvements of 8.6%, 10.1%, 11.4%, respectively, over pRCB.

Sensitivity Result 2*: The partition quality of pGMP degrades more than that of pRCB with fewer embedding iterations, but pGMP remains better than pRCB with the same approximate embedding.*

3) Impact of mesh quality: In this experiment, we use the mesh test suite shown in Table II to evaluate the impact of mesh quality on the performance of the partitioning algorithms. Recall that the quality metric we use is the *average edge ratio* of all triangles in the mesh. Thus, meshes with higher average edge ratios are of better quality than meshes with lower average edge ratios. To generate meshes of different qualities, we randomly shift the non-boundary points of the mesh in a way similar to the coordinates perturbation experiment. However, the graphs considered in that experiment are not mesh graphs and all the vertices (including the boundary ones) are subject to perturbation. In this experiment, the (non-boundary) corners of the triangles are moved while the validity of the mesh is maintained by checking the following property [24]: A mesh is *valid* if all the triangles in the mesh are valid, and a triangle is *valid* if the determinant of its Jacobian is greater than zero. For instance, if the three corners of a triangle have coordinates (x_0, y_0), (x_1, y_1) and (x_2, y_2), then the triangle is valid if

$$J = \begin{vmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{vmatrix} > 0. \quad (1)$$

As we increase the random step size to shift the mesh points, the quality of the mesh progressively decreases. In Table VII, we list the original three meshes in the test suite ($q = 0$) and their shifted versions in increasing order of step size ($q = 1, 2, 3$), together with their respective average edge ratios. We point out that the average edge ratio of all three meshes are approximately the same in each of the three versions.

Table VI presents the cutsizes of pRCB and pGMP for the three meshes and their shifted versions. The last two rows report the geometric mean for all three meshes, and the normalized value with respect to that of pRCB for the original meshes. We observe that the mesh quality does have a significant impact on the partition quality of the algorithms. For the original meshes (with average edge ratio around 0.74), pGMP is 24% better than pRCB on average. As the average edge ratio reduces down to around 0.63, 0.47 and 0.40, the quality of pRCB degrades by 1%, 11% and 25%, respectively, while pGMP degrades more by 14.5%, 32.9% and 51.3%, respectively. However, pGMP are still better than pRCB by 13.9%, 9% and 8% in terms of the average cutsize under the three shifted meshes.

Sensitivity Result 3*: The partition quality degrades more for pGMP than for pRCB with decreasing quality of the mesh, but pGMP remains better than pRCB regardless of the mesh quality.*

4) Impact of graph planarity: Finally, we evaluate the impact of planarity of a graph by comparing the qualities of pRCB and pGMP for a set of planar and non-planar graphs. As in the previous experiment, we use the three mesh graphs shown in Table II, which are all planar graphs. To obtain non-planar graphs, we randomly add 0.1%, 0.3%, 0.5% and 0.7% of extra edges to a set of randomly selected vertex-pairs in the original meshes, which turn them into non-planar graphs.

Table VIII presents the cutsizes of pRCB and pGMP for the three planar graphs and their non-planar versions. The last two rows report the geometric mean for all three graphs, and the normalized value with respect to that of pRCB for the planar graphs. Again, pGMP is 24% better than pRCB on average for the planar graphs. We can see that when only 0.1% of extra edges (corresponding to 45-461 extra edges) are added, the average cutsizes of pRCB and pGMP more than tripled compared to their original cutsizes. When 0.7% of extra edges are added, the partition quality degrades by almost 9 times for pRCB and more than 6 times for pGMP compared to their respective qualities for the planar graphs. With 0.7% extra edges, pGMP becomes almost twice as good as pRCB.

Sensitivity Result 4*: The qualities of both algorithms are significantly affected by adding extra edges that turn a graph from planar to non-planar. Furthermore, the quality gap between pGMP and pRCB widens with increased non-planarity of the graphs.*

V. Conclusion and Future Work

We have conducted a scalability and sensitivity study for the performance of two parallel geometric graph partitioning algorithms (pRCB and pGMP). The goal was to evaluate the robustness of the two algorithms by examining a number of factors that could impact their performance. Extensive experiments

TABLE VIII: Cutsizes of pRCB and pGMP for the three planar graphs (with 0% extra edge) in the mesh test suite and their non-planar versions (with $0.1\%, 0.3\%, 0.5\%, 0.7\%$ extra edges).

Planarity	0% extra edge		0.1% extra edges		0.3% extra edges		0.5% extra edges		0.7% extra edges	
	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP	pRCB	pGMP
mechanic	196	86	434	329	615	489	936	518	1069	616
twoholes	208	202	390	291	698	606	861	607	1022	480
square_with_hole	407	421	4062	4102	6395	4263	9447	5649	10905	6283
Geometric mean	255	194	882	732	1400	1081	1967	1211	2284	1229
Normalized	1.00	0.76	3.46	2.87	5.49	4.24	7.71	4.75	8.95	4.82

were conducted on a 32-core cluster using two sets of graph benchmarks. The following summarizes our main findings:

- pGMP with 5 tries (or pGMP-5) appears to be the most appropriate choice for running the algorithm. It provides on average 9.6% improvement over only a single try (or pGMP-1) and 25% improvement over pRCB.
- pGMP has slower execution time but scales well when executed on multiple cores, while pRCB has better overall execution time due to its efficient implementation.
- Small coordinate perturbations barely impact the performance of both algorithms. Under large perturbations, pGMP consistently outperforms pRCB by as much as 48%.
- Approximate embeddings of the graphs degrade the partition qualities of both algorithms, and pGMP maintains around 10% quality advantage over pRCB.
- Both algorithms experience increased cutsizes for meshes with poorer quality, while pGMP again performs 8-14% better than pRCB.
- Non-planar graphs are significantly harder to partition than planar graphs, and the performance gap between pGMP and pRCB increases by up to 50% with less than 1% of extra edges that turn a planar graph into a non-planar one.

A future direction is to optimize the geometric partitioning algorithms where multi-level graph embedding and partition refinement are performed as a pre-processing step and a post-processing step, respectively [20]. It will be particularly useful to seek optimal combinations of approximate embedding, partitioning and refinement, as well as a performance profiler to determine the optimal amount of resources for partitioning a graph. Together, these may provide a robust partitioning framework that could offer scalable parallel performance while producing high quality solutions.

Acknowledgment: This research was supported in part by the National Science Foundation under the award CCF1719674.

REFERENCES

[1] N. Alon, P. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *STOC'90*, pages 293–299, 1990.

[2] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, page 324, 1986.

[3] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36(5):570–580, 1987.

[4] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, V. Leung, L. A. Riesen, C. Vaughan, U. Catalyurek, D. Bozdag, W. Mitchell, and J. Teresco. Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User's Guide. Technical Report SAND2007-4748W, Sandia National Laboratories, Albuquerque, N.M., 2007.

[5] T. N. Bui and A. Peck. Partitioning planar graphs. *SIAM J. Comput.*, 21(2):203–215, Apr. 1992.

[6] F. R. K. Chung and S.-T. Yau. A near optimal algorithm for edge separators (preliminary version). In *STOC'94*, pages 1–8, 1994.

[7] T. A. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 92, 1994.

[8] M. Deveci, S. Rajamanickam, K. D. Devine, and U. V. Çatalyürek. Multi-jagged: A scalable parallel spatial partitioning algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):803–817, 2016.

[9] K. D. Devine, E. G. Boman, and G. Karypis. Partitioning and load balancing for emerging parallel applications and architectures. In *Parallel Processing for Scientific Computing (Software, Environments and Tools)*, chapter 6, pages 99–126. SIAM, 2006.

[10] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Dev.*, 17(5):420–425, Sept. 1973.

[11] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC'82*, pages 175–181, 1982.

[12] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

[13] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *STOC'74*, pages 47–63, 1974.

[14] J. R. Gilbert, G. L. Miller, and S.-H. Teng. Geometric mesh partitioning: Implementation and experiments. In *IPPS'95*, pages 418–427, 1995.

[15] M. T. Heath and P. Raghavan. A Cartesian Parallel Nested Dissection Algorithm. *SIAM J. Matrix Anal. Appl.*, 16(1):235–253, Jan. 1995.

[16] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing'95*, page 28, 1995.

[17] Y. F. Hu. Efficient, high-quality force-directed graph drawing. *The Mathematica Journal*, 10:37–71, 2006.

[18] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing'96*, page 35, 1996.

[19] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[20] S. Kirmani and P. Raghavan. Scalable parallel graph partitioning. In *Supercomputing'13*, pages 51:1–51:10, 2013.

[21] P. M. Knupp. Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. part I–A framework for surface mesh optimization. *International Journal for Numerical Methods in Engineering*, 48(3):401–420, 2000.

[22] P. Lamata, I. Roy, B. Blazevic, A. Crozier, S. Land, S. Niederer, D. Hose, and N. Smith. Quality metrics for high order meshes: Analysis of the mechanical simulation of the heart beat. *IEEE Transactions on Medical Imaging*, 32(1):130–138, 2013.

[23] G. L. Miller, S.-H. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *FOCS'91*, pages 538–547, 1991.

[24] J. Park and S. M. Shontz. An alternating mesh quality metric scheme for efficient mesh quality improvement. In *ICCS'11*, pages 292–301, 2011.

[25] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, pages 493–498, 1996.

[26] E. Saule, E. O. Baş, and U. V. Çatalyürek. Load-balancing spatially located computations using rectangular partitions. *J. Parallel Distrib. Comput.*, 72(10):1201–1214, 2012.

[27] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In *Proceedings of Workshop on Applied Computational Geormetry Towards Geometric Engineering (FCRC'96/WACG'96)*, pages 203–222, 1996.

[28] D. A. Spielmat. Spectral partitioning works: Planar graphs and finite element meshes. In *FOCS'96*, page 96, 1996.