LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-463495

A Novel Coarsening Method for Scalable and Efficient Mesh Generation

A. Yoo, D. Hysom, B. Gunney

December 2, 2010

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

A Novel Coarsening Method for Scalable and Efficient Mesh Generation*

Andy Yoo David Hysom Brian Gunney

Lawrence Livermore National Laboratory
Livermore, CA 94551

Abstract

In this paper, we propose a novel mesh coarsening method called *brick coarsening method*. The proposed method can be used in conjunction with any graph partitioners and scales to very large meshes. This method reduces problem space by decomposing the original mesh into fixed-size blocks of nodes called bricks, layered in a similar way to conventional brick laying, and then assigning each node of the original mesh to appropriate brick. Our experiments indicate that the proposed method scales to very large meshes while allowing simple RCB partitioner to produce higher-quality partitions with significantly less edge cuts. Our results further indicate that the proposed brick-coarsening method allows more complicated partitioners like PT-Scotch to scale to very large problem size while still maintaining good partitioning performance with relatively good edge-cut metric.

1 Introduction

Graph partitioning is an important problem that has many scientific and engineering applications in such areas as VLSI design, scientific computing, and resource management. Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, (k-way) graph partitioning problem is to partition the vertices of the graph (V) into k disjoint groups such that each group contains roughly equal number of vertices and the number of edges connecting vertices in different groups is minimized.

Graph partitioning plays a key role in large scientific computing, especially in *mesh-based* computations, as it is used as a tool to minimize the volume of communication and to ensure well-balanced load across computing nodes[1]. The impact of graph partitioning on the reduction of communication can be easily seen, for example, in different iterative methods to solve a sparse system of linear equation. Here, a graph partitioning technique is applied to the matrix, which is basically a graph in which each edge is a non-zero entry in the matrix, to allocate groups of vertices to processors in such a way that many of matrix-vector multiplication can be performed locally on each processor and hence to minimize communication. Furthermore, a good graph partitioning scheme ensures the equal amount of computation performed on each processor. Graph partitioning is a well known NP-complete problem [13], and thus the most commonly used graph partitioning algorithms employ some forms of heuristics [30, 15, 1, 29, 5, 7, 15, 12, 28, 24, 23]. These algorithms vary in terms of their complexity, partition generation time, and the quality of partitions, and they tend to trade off these factors.

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

[1]In this paper, graph and mesh are used interchangeably.

A significant challenge we are currently facing at the Lawrence Livermore National Laboratory is how to partition very large meshes on massive-size distributed memory machines like IBM BlueGene/P, where scalability becomes a big issue. For example, we have found that the ParMetis [20], a very popular graph partitioning tool, can only scale to 16K processors. An ideal graph partitioning method on such an environment should be fast and scale to very large meshes, while producing high quality partitions. This is an extremely challenging task, as to scale to that level, the partitioning algorithm should be simple and be able to produce partitions that minimize inter-processor communications and balance the load imposed on the processors.

Our goals in this work are two-fold:

- To develop a new *scalable* graph partitioning method with good load balancing and communication reduction capability.
- To study the performance of the proposed partitioning method on very large parallel machines using actual data sets and compare the performance to that of existing methods.

The proposed method achieves the desired scalability by reducing the mesh size. For this, it coarsens an input mesh into a smaller size mesh by coalescing the vertices and edges of the original mesh into a set of mega-vertices and mega-edges. A new coarsening method called *brick* algorithm is developed in this research. In the brick algorithm, the zones in a given mesh are first grouped into fixed size blocks called bricks. These brick are then laid in a way similar to conventional brick laying technique, which reduces the number of neighboring blocks each block needs to communicate. Contributions of this research are as follows.

- We have developed a novel method that scales to a really large problem size while producing high quality mesh partitions.
- We measured the performance and scalability of the proposed method on a machine of massive size using a set of actual large complex data sets, where we have scaled to a mesh with 110 million zones using our method. To the best of our knowledge, this is the largest complex mesh that a partitioning method is successfully applied to.
- We have shown that proposed method can reduce the number of edge cuts by as much as 65%.

The paper is organized as follows. We first provide some preliminaries in Section 2. The proposed coarsening method is discussed in Section 3 in a greater detail, followed by the results from our experiments in Section 4. Related work are discussed in Section 5, and conclusions are drawn in Section 6.

2 Preliminaries

In a typical numerical simulation, a physical object being studied, referred to as *domain*, is first discretized to a model that computer can understand. The physical domain is often represented in a form of grid called *mesh*. Each cell in the mesh is referred to as *zone*. A zone can be of any shape, but it usually logically rectangular or hexahedral for two- and three-dimensional mesh, respectively.

Given a graph $G = (V, E)$, *k-way graph partitioning* is problem of finding k disjoint subsets of vertices in G that minimizes the *cut-size*, which is defined as the sum of the weights of edges between the subsets, while each subset has (approximately) equal vertex weight sum. Given a graph where each vertex represents a task to be performed associated with certain computational cost and inter-task communication cost, represented by the corresponding vertex and edge weights respectively, the k-way graph partitioning provides us with an optimal distribution of the tasks over k processors that ensures minimal communications and load balancing. Figure 1 shows 4-way graph partitioning on a small example graph.

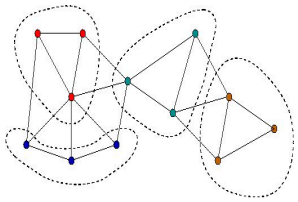


Figure 1: A 4-way partition of an example 12-vertex graph. With vertex and edge weights assumed to be 1, each subset of vertices has the cost of 3. The average cut-size of this partition is 2.5.

Two-way partitioning is often called as *bisection*. Many existing graph partitioning methods often partition graph by iterating a bisection algorithm. A class of graph partitioning algorithm we are mainly interested in this paper is called *multi-level* graph partitioning. A typical multi-level graph partitioner operates in three phases: *coarsening*, *partitioning*, and *uncoarsening* phases. In coarsening phase, the original graph is transformed into a coarser and smaller graph by (often repeatedly) clustering vertices and corresponding edges. Desired number of sets of coarsened vertices are obtained during the partitioning phase by running a partitioning algorithm. The coarsened vertices and edges are then transformed back to original vertices and edges in the uncoarsening phase.

3 Proposed Coarsening Method

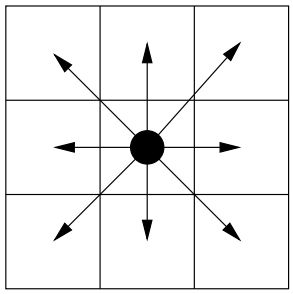
Most of existing graph partitioner aim at finding partitions that reduce edge cuts and ensure balanced computational load. As stated earlier, graph partitioning is an NP-Complete problem. Even many of the approximate graph partitioning algorithms are computationally expensive, and their complexity increases as the problem size increases. This becomes a serious bottleneck for scalability, especially when input meshes are comprised of hundreds of millions of zones or more.

A common approach used by multi-level graph partitioners [22, 20, 21] to solve this problem is to reduce the problem space by coarsening the original graph (or mesh). The coarsening is usually done by clustering the zones of the given mesh into blocks of zones, which are then passed to partitioning algorithms.

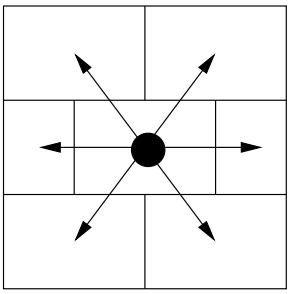
In this paper, we propose a new mesh coarsening method, called *brick coarsening* method, that can be used in conjunction with with any existing graph partitioner to improve their scalability, performance, and partitions quality. Given a mesh, the method conceptually divides the mesh into the blocks of zones of roughly equal size (in terms of the number of zones in them), which we refer to as *bricks*. When assigning zones to bricks, the method attempts to allocated about equal number of zones to each brick to ensure the load balancing.

The key to the reduction in communication lies in the way how the given mesh is divided into bricks, or conversely, how the bricks are combined into the original mesh. In the proposed method, the bricks are laid in a very similar way to the conventional brick laying in masonry. The basic idea of this method is as follows. Consider two brick layouts shown in Figure 2.

In Figure 2.a, the bricks are layered as a grid. In this lay out, the number of neighboring processors that each processor needs to communicate, assuming that single brick is assigned to a processor, is 9. If the bricks are laid out as shown in Figure 2.b, the number of communicating neighboring processor is reduced to 6. Each arrow in Figure 2 represents a potential edge cut, when the coarsened mesh is partitioned by a graph partitioner. Since the number of edge cuts is reduced by the proposed coarsener initially, the final number of edges cuts obtained by the graph partitioner is also reduced. Furthermore, the number of edge cuts is reduced exponentially as the dimension of the mesh increases. For example, in a 3D mesh, the the number of edge cuts obtained by using two methods shown in Figures 2.a and 2.b are 26 and 14, respectively.

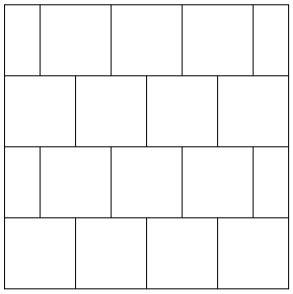


(a) Bricks laid in grid pattern

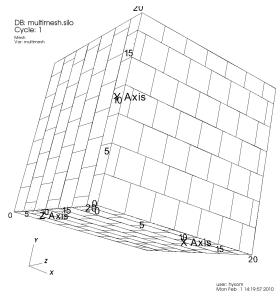


(b) Bricks laid in offset pattern

Figure 2: Comparison of two brick laying techniques in a 2D mesh.



(a) 2D mesh coarsening



(b) A screen shot of a 3D mesh coarsening

Figure 3: Brick coarsening methods for 2D and 3D meshes.

(a) Original mesh

(b) Decomposed mesh

Figure 4: An example of 2D mesh that is composed of four blocks. Original irregularly-structured mesh is divided into four blocks, each depicted in different colors.

Figure 3 presents two examples that illustrate show given meshes are coarsened using brick coarsening. Figure 3.a shows a layout of coarsened 2D mesh. In this layout, the maximum number of communicating neighbors per each processor is 6. Fewer communication may be required for the processors on corners and sides. However, a slightly different brick layout is needed to for 3D mesh, as shown in Figure 3.b that shows a screen capture of coarsened 3D mesh using hexahedral bricks. Readers should note that the brick laying method is also applied on the additional Z dimension to minimize the edge cuts per processor. Here, the brick coarsening method ensures the minimum number of communicating neighbors per processor (six) on X-Y, Y-Z, and Z-X planes.

Meshes typically used in scientific computing has irregular structure as shown in Figure 4.a, for example. Here, an irregular 2D mesh can be decomposed into three (semi-) structured meshes (Figure 4.b). In this case, the brick coarsening method can be applied to each submesh first, and then the coarsened submeshes are combined together before graph partitioning algorithm is applied. Therefore, at the boundary where two submeshes joined, the brick laying pattern may not hold and therefore, there may be more communications from those processors at the boundaries.

Algorithm 1 describes the proposed coarsening method in greater detail. For the sake of discussion, we assume that input mesh and bricks are cubes (that is a 3D mesh with square faces).

In this algorithm, we assume that the zones in the input mesh are numbered by increasing their Z, Y, and then X coordinates. The brick IDs are numbered similarly. The core of the brick coarsening algorithm is a mapping function, *find-brick*, that, given a logical coordinate of a zone, returns the logical coordinate of the brick into which the zone is agglomerated. This function computes the coordinate a brick corresponding to each zone, properly offsetting the bricks at each dimension. From the logical coordinate of a brick, we can easily calculate the corresponding brick ID. In addition, a function that does inverse mapping can also be easily constructed.

4 Experiment Results

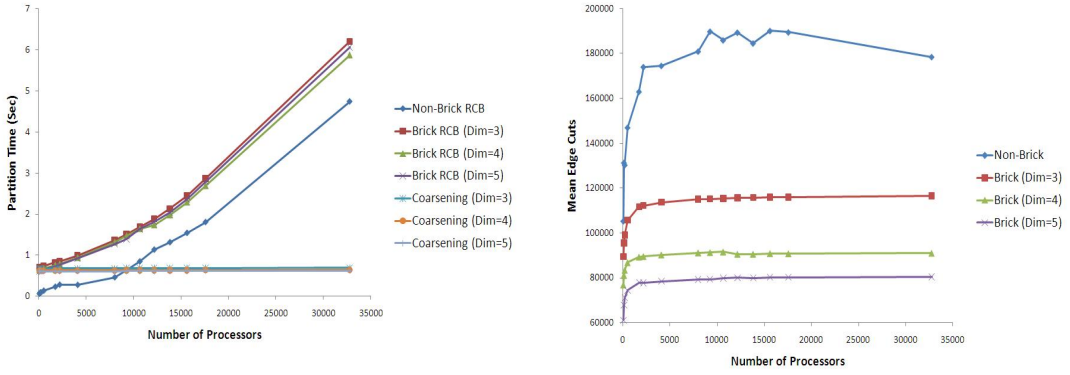
This section presents experimental results for the proposed brick coarsening method. We have conducted our experiments on uBGL, a massively parallel machine developed by IBM [33]. We are mainly interested in the effect of the brick method on the performance of existing graph partitioners. Two state-of-art graph

Algorithm 1 Brock Coarsening Method

Input: A 3D mesh M with $X \times Y \times Z$ zones and m denoting the number of zones in each dimension
Output: A coarsened 3D mesh and a map Γ that maps a zone to brick coordinate

Divide M into a group of bricks that are laid out in an offset manner at each dimension
for all $\forall$ zones z in M with coordinate (x, y, z) **do**
  (x', y', z') = `find-brick`(x, y, z)
  Add mapping $(x, y, z) \Rightarrow (x', y', z')$ to Γ
end for

function `find-brick`(x, y, z)
$z' = \frac{z}{m}$
if z' **mod** $2 == 0$ **then**
  $y' = \frac{y}{m}$
else
  if $y == 0$ **then**
    $y' = 0$
  else
    $y' = \frac{y-1}{m}+1$
  end if
end if
if z' **mod** $2 == 0$ **then**
  if y' **mod** $2 == 0$ **then**
    $x' = \frac{x}{m}$
  else
    if $x == 0$ **then**
      $x' = 0$
    else
      $x' = \frac{x-1}{m}+1$
    end if
  else
    if y' **mod** $2 == 0$ **then**
      if $x == 0$ **then**
        $x' = 0$
      else
        $x' = \frac{x-1}{m}+1$
      end if
    else
      if x < 2 **then**
        $x' = 0$
      else
        $x' = \frac{x-2}{m}+2$
      end if
    end if
  end if
end if
end function



(a) Partition time

(b) Mean edge cuts

Figure 5: Comparison of the performance of the brick-coarsened and naive RCB methods. 3375 zones per processor are used and cube bricks with side length of 3, 4, and 5 are used in these experiments.

partitioners, RCB [3] and PT-Scotch [8], are evaluated in this study. The quality of partitions generated by these partitioners and their execution time were the metrics of interest.

4.1 Description of Experimental Environment

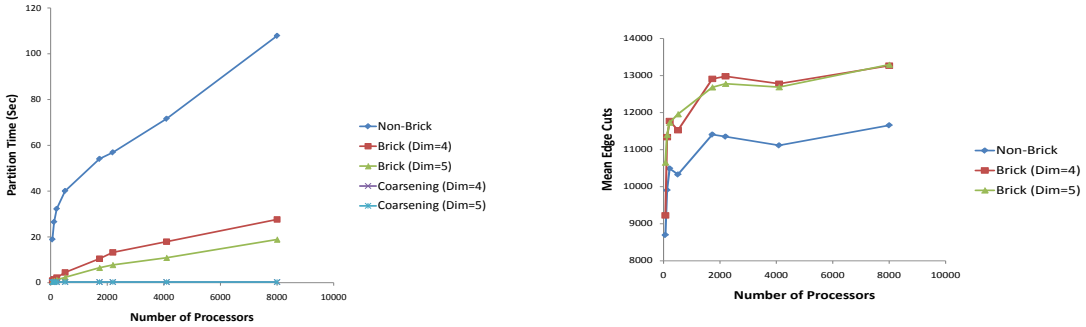
Our experiments were performed on IBM uBGL and the Lawrence Livermore National Laboratory. The uBGL is based on the original IBM BlueGene/L architecture [4]. The machine consists of 40960 compute nodes (CNs), each of which where each CN contains two IBM PowerPC processors, and hence the system has 81920 processors in total. The total peak performance of the system is 229.4 TFLOP/s running at 700 MHz. The uBGL is equipped with 512 MB of main memory per CN (and 22 TB of total memory).

Each CN contains six bi-directional torus links directly connected to nearest neighbors in each of three dimensions. The CNs are also connected by a separate tree network in which any CN can be a root. The torus network is used mainly for communications in user applications and supports point-to-point as well as collective communications. The tree network is also used for CNs to communicate with I/O nodes. It can be also used for some collectives such as broadcast and reduce.

A CN runs on a simple run-time system called compute node kernel (CNK) that has a very small memory footprint. The main task of the CNK is to load and execute user applications. The CNK does not provide virtual memory and multi-threading support and provides a fixed-size address space for a single user process. Many conventional system calls including I/O requests are function-shipped to a separate I/O node which runs on a conventional Linux operating system.

4.2 Performance Study Results

First, we have measured the effect of the brick coarsening on the performance of the two graph partitioners considered in this study, RCB and PT-Scotch. Three dimensional synthetic meshes with 3375 local zones per processor are used in all of the weak-scaling experiments. Figure 5 presents the results for the RCB method for different brick sizes, where the performance of the naive and brick-coarsened RCB methods is



(a) Partition time

(b) Mean edge cuts

Figure 6: Comparison of the performance of the brick-coarsened and naive PT-Scotch methods. 3375 zones per processor are used and cube bricks with side length of 4 and 5 are used in these experiments.

compared. Figures 5.a and 5.b show the time to partition given mesh and the mean edge cuts obtained by the partitions, respectively.

The total partitioning time of the brick-coarsened RCB includes the time to coarsen given input meshes to smaller meshes. As shown in the Figure 5.a, the brick coarsening overhead is very small compared to partitioning time, especially for large number of processors. The coarsening time constitutes only 10% of total execution time when 32768 processors are used and we expect this number should grow smaller as we increase the number of processors used. It is also interesting to note that the proposed brick coarsening method scales very well as shown in the figure, where the brick coarsening time remains constant independent of the brick size and the number of processors used. The brick method achieves such high scalability, mainly because it clusters input meshes following a simple, predetermined order (similar to brick laying technique), and hence there is no need to optimize any global objective functions.

The run-time of the brick-coarsened RCB method, however, is consistently higher than that of the naive RCB method. In worst case, the brick-coarsened RCB method runs as much as 1.5 times slower than the naive RCB as shown in Figure 5.a. On the other hand, the brick-coarsened RCB generates partitions whose edge cuts are considerably smaller than those of the naive RCB method. The Figure 5.b shows that the brick-coarsened RCB can reduce the number of edge cuts by as much as 65%. This indicates that the brick-coarsened RCB actually performs more iteration and produces better partitions, enabled by the reduced problem size with the brick coarsening. Furthermore, the mean edge cuts decreases as we increase the brick size. We believe that it is mainly due to the synthetic meshes used in the experiments as input, as using larger brick sizes reduces the overall problem size while maintaining the overall structure of the input meshes, allowing the RCB method to obtain finer partitions.

Figure 6 compares the performance metrics for naive and brick-coarsened PT-Scotch methods, where Figures 6.a and 6.b show the partition time and the mean cut edges obtained by the PT-Scotch for different brick sizes, respectively. As in the RCB method, the coarsening time remains constant independent of the

brick sizes and the number of processors used, again proving high scalability of the brick coarsening method. Unlike the RCB method, however, the naive PT-Scotch method exhibits higher partition time than its coarsened counterparts as shown in the Figure 6.a. This is due to the higher complexity of the partitioning algorithm employed by the PT-Scotch, compared to that of the RCB method. Here, the brick coarsening simply reduces the problem space with which the PT-Scotch method runs faster. This is clearly indicated by the fact that the brick-coarsened PT-Scotch with brick size 5 outperforms the one with the brick size of 4.

We have found, however, that the naive PT-Scotch method outperforms the brick-coarsened method in terms of the mean edge cuts as indicated in Figure 6.b. The partitions obtained by the brick-coarsened PT-Scotch method have 5 to 15% more edge cuts on average than the naive PT-Scotch method. We believe that this is closely related to the complex partition algorithm of the PT-Scotch method itself. With uncoarsened mesh, the naive PT-Scotch can perform partitioning with finer granularity (at the expense of high computing time), whereas coarsening increases the granularity of the partitions. This is an important trade-off that needs careful examination, because PT-Scotch, like other graph partitioners, is known to fail to scale to very large problems. For those problems, the brick coarsening is needed to reduce the problem size and hence critical to achieving high scalability. This needs further investigation.

5 Related Work

Graph partitioning is an extensively-studied problem, and many algorithms have been reported in the literature. This section briefly surveys some of the well-known graph partitioning algorithms. As mentioned earlier, graph partitioning is NP-hard problem, and therefore most of the existing algorithms are heuristics-based.

One of the earliest and widely-used partitioning methods developed is Kernighan-Lin (KL) algorithm [23]. The idea of the KL method is simple. Given a bisection, the KL method iteratively exchanges pairs of vertices from each partition if the exchange reduces the cut-size. For this, each vertex is associated with a *gain* value, which is the number of cuts reduced by moving the vertex from current group to another. The KL algorithm selects a pair of vertices with the largest gain value for exchange. The KL algorithm is simple but works relatively well. A drawback of this algorithm is its high computational complexity of $O(|V|^3)$. Another drawback, which is common to many local improvement methods, is that the final partition quality depends heavily on the initial partition. Dutt [9] has shown that the KL algorithm can be improved to have $O(|E|max(log|V|, d_{max}))$, where d_{max} denotes the maximum node degree. The KL method was further improved by Fiduccia and Matteyses (FM) [11], where an iteration can be done in $O(|E|)$ time.

It is relatively easy to parallelize the KL algorithms, as finding an optimal bisection can be performed by different processors independently. On the other hand, the SA and GA methods are inherently sequential. Gilbert and Zmijevski [14] proposed a parallel graph bisection algorithm based on the KL method. In this parallel implementation of the KL method, the adjacency list of vertices is assigned to different processors. At each iteration, a pair of vertices with the largest gain values are selected and the update of the gain values are made to the vertices adjacent to selected pair in parallel. The selected pairs are exchanged, then, if it improves the partition quality.

Simulated annealing (SA) [25] is a general local search technique based on statistics mechanics. An advantage of the SA method is that it does not get trapped in some local optima, in contrast to the greedy methods like KL. Starting with an initial random *solution* S and *temperature* T, the SA method performs following steps L times. A random solution S' of the current solution S is obtained. If S' improves the quality of solution, S' replaces S as current solution. However, even if the quality does not improve, S' replaces S with a certain probability, which is a step to avoid being trapped in a local optimum. After L iterations, SA method terminates if certain stopping criterion is met. Otherwise, another round of L

iteration is performed. The termination of SA method is controlled by manipulating the probability for a new solution S' to replace S.

Genetic algorithm (GA) is another method that tries to find optimal solution through searching random solutions. The GA method starts with a set of random solutions, each of which is often encoded as a sequence of bits, and it iterates a sequence of three operations, *crossover*, *mutation*, and *replacement*, until the solution at hand does not improve. In crossover, two offsprings are formed by cutting the parent chromosomes at a random position and concatenating the portions from the mother and father chromosomes. The offsprings are then mutated by flipping bits at random locations in the mutation phase. Finally, the offsprings replace existing chromosome in the replacement phase, if the replacement improves the quality of overall solutions. Several GA-based graph partitioning methods have been reported in the literature [31, 26, 6].

Many k-way partition algorithms are reported in the literature. Most of the methods are bisection-based in that they recursively apply a bisection step until k subsets of vertices are obtained. The best-known k-way partition method is *recursive coordinate bisection* (RCB) [3]. The RCB method obtains a bisection by first selecting a coordinate axis. Then, it finds a plane, orthogonal to the selected axis, that bisects the vertices of graphs into two subsets of roughly equal size. This process is repeated until k subsets are obtained. A more elaborate RCB-based method called *inertial* method has been proposed [10]. This method differs from RCB in that it selects the axis of minimum angular momentum of the set of vertices, instead of a coordinate axis. The inertial method is combined with the KL method, where bisection computed by inertial method is improved by the KL method [27],

A parallel RCB-like method called *unbalanced recursive bisection* (UCB), was proposed [17]. As in RCB, the cut planes are selected perpendicular to a selected axis. However, the bisection does not necessarily subdivide the vertices into equal-sized subsets. Rather, it is sufficient to have subsets whose sizes are multiples of $|V|/k$. Nakhimovski proposed that is similar to URB scheme, except that it tries to find a plane that cuts minimum edges.

In many cases, the graphs are not embedded in space, and hence only the combinatorial structure of the graphs, rather than geometric information (i.e., coordinate), can be used in partitioning. The recursive graph bisection (RGB) method [32] first finds a *pseudo peripheral vertex* in the given graph, which is defined as a pair of vertices that are approximately at the greatest distance from each other. Then graph distance from the selected vertex is calculated for every vertex in the graph by simple breadth-first search (BFS) and then the vertex list sorted in the order of graph distance is divided into two equal-sized sets.

The *recursive spectral bisection* (RSB) method [30, 32] uses the eigenvector that corresponds to the second smallest eigenvalue of *Laplacian matrix* of the graph, called *Fiedler vector*. The difference between coordinates of the Fiedler vector provides the information about the distance between corresponding vertices. The RSB method obtains a bisection by sorting the vertices with respect to their Fiedler vector coordinates and dividing the sorted list into two halves. A parallel implementation of the RSB method was made on Connection Machine CM-5 [34, 35] and on Cray T3D [2].

A partitioning method that employ multilevel approach was proposed [1]. A multilevel method consists of three phases: *coarsening*, *partitioning*, and *uncoarsening*. During the coarsening phase, the vertices and edges of given graph is coalesced into mega-vertices and mega-edges. When a sufficiently coarse graph is obtained, partitioning is performed on the coarse graph. During the uncoarsening phase, these partition is propagated back. Coarsening a graph into a smaller approximation of the original graph reduces the search space drastically and hence improves the partitioning performance. In the partitioning phase, a bisection partitioning [18] or k-way partitioning [16, 19] can be used. A KL-type of algorithms can be invoked to improve the partition quality. Karypis and Kumar presented the parallel implementation of their multilevel bisection method [22] and multilevel k-way partitioning algorithm [20, 21].

6 Conclusions

Graph partitioning plays an important role in large scientific parallel computing, as it boosts performance by enabling load balancing and reducing communication time. Graph partitioning is a very difficult problem that often requires computationally expensive algorithms to obtain good partitions. The high computational complexity of graph partitioning algorithms makes it non-trivial to scale them to very large meshes with hundreds of millions of zones, while generating high quality meshes.

We propose a novel mesh coarsening method called *brick coarsening* method in this paper to address this issue. The proposed coarsening algorithm improves the scalability and achieves good load balancing by clustering neighboring zones into equal-sized blocks of zones called *bricks*. More importantly, the proposed algorithm reduces the inter-processor communications by laying out the bricks in an offset manner, similar to conventional brick laying in masonry.

The proposed method, when used in conjunction with any existing graph partitioners, can improve their performance in terms of the scalability and the quality of resulting partitions. Our experiments conducted on a state-of-art massive parallel computer show that the proposed method scales to very large meshes while allowing simple RCB partitioner to produce higher-quality partitions with significantly less edge cuts. The results also indicate that the proposed brick-coarsening method allows more complicated partitioners like PT-Scotch to scale to very large problem size while still maintaining good partitioning performance with relatively good edge-cut metric.

References

[1] S. Barnard and H. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. In *6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 711–718, 1993.

[2] S. T. Barnard. Pmrsb: parallel multilevel recursive spectral bisection. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 27, New York, NY, USA, 1995. ACM.

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

[4] Blue Gene/L. http://cmg-rr.llnl.gov/asci/platforms/bluegenel.

[5] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *6th SIAM Conf. Parallel Processing for Scientific Computers*, pages 445–452, 1993.

[6] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. Comput.*, 45(7):841–855, 1996.

[7] C.-K. Cheng and Y.-C. A. Wei. An improved two-way partitioning algorithm with stable performance [vlsi]. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 10(12):1502–1511, 1991.

[8] C. Chevalier and F. Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel Comput.*, 34(6-8):318–331, 2008.

[9] S. Dutt. New faster kernighan-lin-type graph-partitioning algorithms. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 370–377, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[10] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Internat. J. Numer. Meth. Engrg*, 36(5):745–764, 1993.

[11] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *25 years of DAC: Papers on Twenty-five years of electronic design automation*, pages 241–247, New York, NY, USA, 1988. ACM.

[12] J. Garbers, H. J. Prömel, and A. Steger. Finding clusters in vlsi circuits. In *ICCAD*, pages 520–523, 1990.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[14] J. R. Gilbert and E. Zmijewski. A parallel graph partitioning algorithm for a message-passing multiprocessor. *Int. J. Parallel Program.*, 16(6):427–449, 1987.

[15] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical report, Sandia National Laboratories, 1993.

[16] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 28, New York, NY, USA, 1995. ACM.

[17] M. Jones and P. Plassman. Computational results for parallel unstructured mesh computations. Technical Report UT-CS-94-248, Computer Science Department, University of Tennesse, 1994.

[18] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, University of Minnesota, Dept. of Computer Science, 1995.

[19] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, University of Minnesota, Dept. of Computer Science, 1995.

[20] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, page 35, Washington, DC, USA, 1996. IEEE Computer Society.

[21] G. Karypis and V. Kumar. A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm. In *PPSC*, 1997.

[22] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *J. Parallel Distrib. Comput.*, 48(1):71–95, 1998.

[23] B. Kernighan and S. Lin. An efficient heuristics for partitioning graphs. Technical report, The Bell System Technical Journal, 1970.

[24] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[25] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. *Optimization by simulated annealing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[26] G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Fourth International Conference on Genetic Algorithms*, pages 45–52, 1991.

[27] R. Leland and B. Hendrickson. An emperical study of static load balancing algorithms. In *Scalable High-Performance Comput. Conf.*, pages 682–685, 1994.

[28] N. Mansour, R. Ponnusamy, A. Choudhary, and G. C. Fox. Graph contraction for physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers. In *ICS '93: Proceedings of the 7th international conference on Supercomputing*, pages 1–10, New York, NY, USA, 1993. ACM.

[29] G. L. Miller, S.-H. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *SFCS '91: Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 538–547, Washington, DC, USA, 1991. IEEE Computer Society.

[30] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[31] Y. G. Saab and V. B. Rao. Stochastic evolution: a fast effective heuristic for some generic layout problems. In *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 26–31, New York, NY, USA, 1990. ACM.

[32] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[33] uBGL at LLNL. https://computing.llnl.gov/?set=resources&page=ocf_resources#ubgl.

[34] Z. Zohan, K. Mathur, S. Johnson, and T. Hughes. An efficient communication strategy for finite element methods on the connection machine cm-5 system. Technical Report TR-11-93, Parallel Computing Research Group, Center for Research in Computing Technology, Harvard University, 1993.

[35] Z. Zohan, K. Mathur, S. Johnson, and T. Hughes. Parallel implementation of recursive spectral bisection on the connection machine cm-5 system. Technical Report TR-07-94, Parallel Computing Research Group, Center for Research in Computing Technology, Harvard University, 1994.