Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 97-060

PARMETIS: Parallel Graph
Partitioning and Sparse Matrix
Ordering Library

by: George Karypis, Kirk Schloegel
and Vipin Kumar

PARMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library*

Version 1.0

George Karypis, Kirk Schloegel and Vipin Kumar

University of Minnesota, Department of Computer Science

Minneapolis, MN 55455

{karypis, kirk, kumar}@cs.umn.edu

Last updated on July 18, 1997 at 7:52am

1 Introduction

PARMETIS is an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs and for computing fill-reducing orderings of sparse matrices. PARMETIS is particularly suited for parallel numerical simulations involving large unstructured meshes. In this type of computation, PARMETIS dramatically reduces the time spent in communication by decomposing the mesh and distributing it among the processors in a way that minimizes the number of interface elements.

The algorithms in PARMETIS are based on the multilevel partitioning and fill-reducing ordering algorithms that are implemented in the widely used serial package METIS [1]. However, PARMETIS extends the functionality provided by METIS and includes routines that are especially suited for parallel computations and large scale numerical simulations. In particular, PARMETIS provides the following four major functions:

- Partition an unstructured graph.
- Improve the quality of an existing partition.
- Repartition a graph that corresponds to an adaptively refined mesh.
- Compute a fill-reducing ordering for sparse direct factorization.

 This work was supported by IST/BMDO through Army Research Office contract DA/DAAH04-93-G-0080, and by Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to computing facilities were provided by Minnesota Supercomputer Institute, Cray Research Inc, and by the Pittsburgh Supercomputing Center.

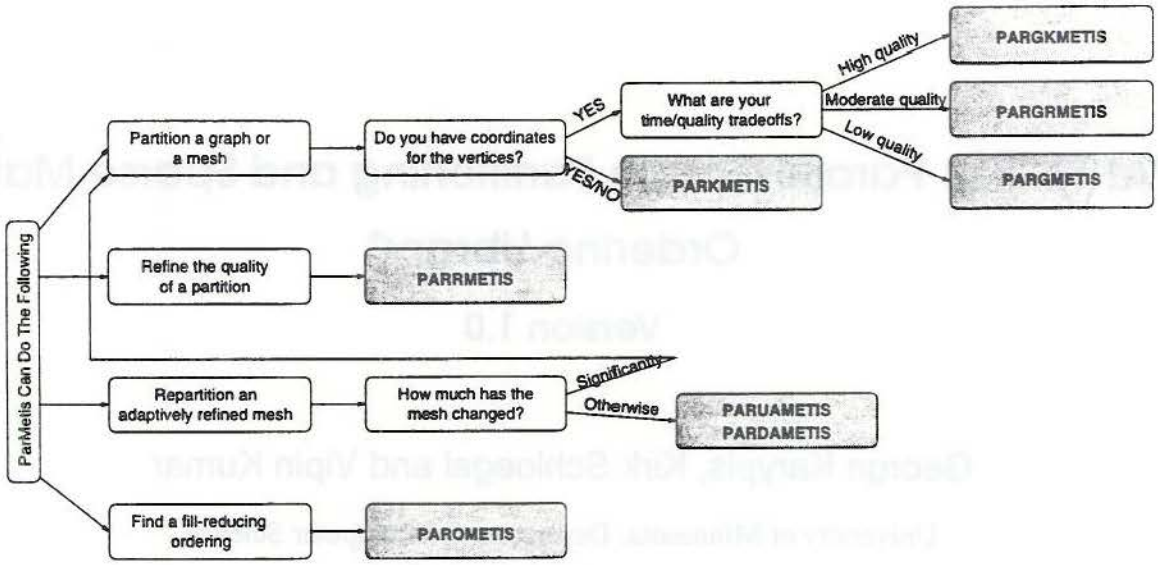


Figure 1: A brief overview of the functionality provided by ParMETIS. The shaded boxes correspond to the actual routines in ParMETIS that implement each particular operation.

This manual is organized as follows. In Section 2 we briefly describe the various algorithms that are used in ParMETIS. Section 3 describes the format of the basic parameters that are supplied to the various routines. Section 4 provides a detailed description of the calling sequence of ParMETIS's functions. Finally, Section 5 provides contact information.

2 Algorithms Used in ParMETIS

ParMETIS provides a variety of algorithms that can be used to compute a number of different partitionings as well as fill-reducing orderings. Figure 1 provides an overview of the functionality provided by ParMETIS and how it can be used to solve a particular problem.

2.1 Unstructured Graph Partitioning

`PARKMETIS` is the function in ParMETIS that can be used to partition an unstructured graph. This routine takes a graph and computes a k-way partition (where k is equal to the number of processors). `PARKMETIS` makes no assumptions on how the graph is initially distributed among the processors. It can effectively partition a graph that is randomly distributed as well as a graph that is nicely distributed[1]. If the graph is already nicely distributed among the processors, `PARKMETIS` will take less time. However, the quality of the produced partitions does not depend on the initial distribution. We will get to this point later in this section.

The parallel graph partitioning algorithm used in `PARKMETIS` is based on the serial multilevel k-way partitioning

[1]The reader should note the difference between the terms *graph distribution* and *graph partition*. In order to partition a graph in parallel, we need to first distribute the nodes and edges of the graph among the processors. Of course you can think of this initial distribution as a partition; however, most likely this partition will be of extremely poor quality. For example, consider a graph that corresponds to the dual of a finite element mesh. An initial distribution of this mesh will assign n/p elements to each processor where n is the number of elements and p is the number of processors. If this distribution was done in a simple-minded fashion (*e.g.*, you read the mesh from a file and assign consecutive n/p elements to each processor), then most likely it will not be a very good partition. This is because you may end-up assigning to each processor elements that belong to many different parts of the mesh (*i.e.*, each processor gets a lot of small disjoint sub-domains as opposed to a single large contiguous sub-domain). The purpose of a parallel partitioner is to take such a simple-minded distribution and compute another distribution (*i.e.*, partition), such that each processor will end up getting just a single sub-domain, and the number of interface elements will be minimized. Of course, it may happen that the initial distribution is already a pretty good partition, as in the case of meshes that are adaptively refined. In this case we will refer to this situation as one in which the graph is *nicely* distributed.

algorithm described in [2] and parallelized in [3]. This k-way partitioning algorithm has been shown to quickly produce partitions that are of very high-quality. The multilevel k-way partitioning algorithms works as follows: The graph is first gradually coarsened down to a graph containing a few hundred vertices, a k-way partition of this much smaller graph is computed, and then this partitioning is projected back to the original graph (finer graph) by periodically refining the partition. Since the finer graph has more degrees of freedom, such refinements improve the quality of the partition. Comparisons performed in [2] have shown that the serial multilevel k-way partitioning algorithm is over 50 times faster than the popular multilevel spectral bisection [4] while producing partitions that cut 10% to 50% fewer edges. PARKMETIS further reduces the amount of time required for partitioning. Experiments on a 128-processor Cray T3D have shown that PARKMETIS can partition an eight million element 3D mesh in about 14 seconds!

Recall earlier we mentioned that if the graph is nicely distributed among the processors PARKMETIS runs faster. In fact, our experiments have shown that when we use PARKMETIS to partition a graph that is distributed according to the partition produced by an earlier call to PARKMETIS[2], the amount of time required for this second partitioning is often smaller by a factor of two to four. The reason for this has to do with the amount of communication required by PARKMETIS. If the graph is initially distributed randomly (*i.e.*, there are many interface vertices), PARKMETIS spends a lot of time communicating information about these interface vertices. On the other hand, if the graph is nicely distributed, the number of interface vertices is much smaller, reducing the overall runtime of the partitioner. Of course, this is the typical chicken and egg problem. How can we initially distribute the graph nicely without actually having to run the partitioner? Hopefully, when partitioning graphs that correspond to finite element meshes we can quickly compute a fairly nice initial distribution by using the coordinate values of the mesh. We can then move the graph according to this initial distribution (partition) and call PARKMETIS on this moved graph. ParMETIS provides the PARGKMETIS function for doing just that. Given a graph that is distributed among the processors and the coordinates of the vertices[3] PARGKMETIS quickly computes an initial partitioning using space-filling curves, redistributes the graph according to this partition and then calls PARKMETIS to compute the final partition. Our experiments have shown that PARGKMETIS is often two times faster than PARKMETIS, and achieves identical partitioning quality.

ParMETIS also provides two additional functions for partitioning unstructured graphs when coordinates for the vertices are available. These functions are PARGMETIS and PARGRMETIS. PARGMETIS computes a partition based only on the space-filling curves. It is extremely fast (often 5 to 10 times faster than PARGKMETIS), but it produces poor quality partitions (it may cut 2 to 3 times more edges than PARGKMETIS). However, it can be useful for a very large number of processors or for computations in which the use of space-filling curves is the appropriate partitioning technique (*e.g.*, n-body computations). The second function PARGRMETIS is in the spirit of PARGKMETIS. However, after redistributing the graph according to the initial geometric partition, it uses the multilevel refinement algorithm PARRMETIS described in Section 2.2 to improve the quality of the partition. Our experiments have shown that PARGRMETIS is usually faster than PARGKMETIS, especially for a large number of processors, but its quality is 30% to 40% worse than PARGKMETIS.

2.2 Partitioning Refinement

PARRMETIS is the routine provided by ParMETIS to improve the quality of an existing partition. Once a graph is partitioned and it has been redistributed accordingly, PARRMETIS can be called to compute a new partition that further

[2] That is, we first called PARKMETIS to find a good partition of a graph. Next we moved the vertices of the graph according to the partition found by PARKMETIS, and then we called PARKMETIS to partition this newly distributed graph.

[3] If PARGKMETIS is used to partition the dual of a mesh, the coordinates of the centers of each element must be provided.

improves the quality. Thus, as opposed to PARKMETIS this routine assumes that the graph is already nicely distributed among the processors. As discussed in Section 2.1 when we talked about PARGRMETIS, PARRMETIS can be used to improve the quality of partitionings that are produced by other partitioning algorithms. PARRMETIS can also be used repeatedly to further improve the quality. That is, you can call PARRMETIS once, move the graph according to the new partition, and then call PARRMETIS again. However, each successive call to PARRMETIS will tend to produce smaller improvements in the quality.

The refinement algorithm implement in PARRMETIS is based on the multilevel k-way refinement algorithm, also used by PARKMETIS. It is in nature similar to the the multilevel k-way partitioning algorithm; however, only vertices belonging to the same partition are coarsened together. Thus, the original distribution of the vertices determines the initial k-way partitioning of the coarse graph. Since PARRMETIS starts from a graph that is already nicely distributed, it is very fast.

2.3 Partitioning Adaptively Refined Meshes

For a large class of irregular mesh applications, the structure of the mesh changes from one phase of the computation to the next, due to mesh refinement or derefinement. Eventually, as the mesh evolves, the adapted mesh has to be repartitioned to ensure good load balance. Even though this repartitioning can be done by computing a new partition from scratch using either PARKMETIS or PARGKMETIS, this approach will in general lead to significant vertex/data movement in order to redistribute the mesh according to the new partitioning. For this reason ParMETIS provides the routines PARUAMETIS and PARDAMETIS to repartition these adaptively refined meshes. These routines assume that the mesh is initially nicely distributed among the processors, but the distribution is not load balanced.

Both of these routines can be used to load balance the mesh either after or before the adaptation. In the former case, each processor first locally adapts its mesh, leading to different processors having different number of elements. Then calling PARUAMETIS or PARDAMETIS on the graph corresponding to either the mesh or the dual of the mesh will result in a partition that after redistribution of the mesh will be load balanced. However, the load balancing can also be done before adaptation. For example, if we can estimate for each element the degree of refinement (*i.e.*, into how many elements we will subdivide each element), we can then use this as a weight on the vertex that corresponds to the dual of the mesh. PARUAMETIS or PARDAMETIS can then be called to load balance this weighted graph. Now we can redistribute the mesh according to this new partition and perform the refinement. After the refinement, each domain will have an equal number of elements.

These adaptive repartitioning algorithms are based on the multilevel diffusion algorithms described in [5]. They perform local multilevel coarsening followed by multilevel diffusion and refinement to balance the graphs while maintaining the edge-cut. These algorithms produce partitions of quality comparable to that of PARKMETIS, but tend to dramatically reduce the amount of data that needs to be moved due to the repartitioning. The difference between these two functions is that PARUAMETIS performs undirected diffusion based on local balancing criteria, whereas PARDAMETIS uses a 2-norm minimization algorithm at the coarsest graph to guide the diffusion (for this reason we refer to it as directed diffusion). Both of these routines should be capable of load balancing graphs, and PARUAMETIS is in general faster. Furthermore, since these routines start from a graph that is already nicely distributed, they are very fast. Experiments on a 256-processor Cray T3D show that PARUAMETIS is able to compute a partition for an 8-million element mesh in under 3 seconds.

2.4 Computing Fill-Reducing Orderings

PAROMETIS is the routine provided by PARMETIS for computing a fill-reducing ordering, suited for Cholesky-based direct factorization algorithms. PAROMETIS makes no assumptions on how the graph is initially distributed among the processors. It can effectively compute a fill-reducing ordering for a graph that is randomly distributed as well as a graph that is nicely distributed.

The algorithm implemented by PAROMETIS is based on a multilevel nested dissection algorithm. This algorithm has been shown to produce low fill orderings for a wide variety of matrices. Furthermore, it leads to balanced elimination trees which is essential for parallel direct factorization. PAROMETIS uses a multilevel node-based refinement algorithm that is particularly suited for directly refining the size of the separators. To achieve high performance, PAROMETIS first uses PARKMETIS to compute a high quality partition and accordingly redistributes the graph. Next it proceeds to compute the $\log p$ levels of the elimination tree concurrently. When the graph has been separated in p parts (where p is the number of processors), the graph is redistributed among the processor so that each processor receives a single subgraph, and a multiple minimum degree algorithm is used to order these smaller subgraphs.

3 Input/Output Formats used by PARMETIS

3.1 Format of the Input Graph

All of the routines in PARMETIS take as input the adjacency structure of the graph, the weights of the vertices and edges (if any), and an array describing how the graph is distributed among the processors. Note that depending on the application this graph can represent different things. For example, in the case of finite element computations, the vertices of the graph can correspond to nodes (points) in the mesh while edges represent the connections between these nodes (*e.g.*, triangular edges in the case of triangle- or tetrahedron-based meshes). Alternatively, the graph can correspond to the dual of the finite element mesh. In this case each vertex corresponds to an element and two vertices are connected via an edge if the corresponding elements share a face. In another case, when PARMETIS is used to compute a fill-reducing ordering, the graph corresponds to the non-zero structure of the matrix (excluding the diagonal entries). However, regardless of what the graph actually represents, it must be undirected. That is, for every pair of connected vertices v and u, it must contain both edges (v, u) and (u, v).

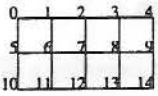
The adjacency structure of the graph is stored using the compressed storage format (CSR), extended for parallel storage. We will first describe how the CSR format is used to store a graph serially and then describe how it is extended for storing a graph that is distributed among processors.

Serial CSR Format The CSR format is a widely used scheme for storing sparse graphs. In this format the adjacency structure of a graph is represented using two arrays `xadj` and `adjncy`, and the weights on the vertices and edges (if any) are represented by using two additional arrays called `vwgt` and `adjwgt`. For example, consider a graph with n vertices and m edges. In the CSR format, this graph is stored serially using the arrays of the following sizes:

$$\texttt{xadj}[n+1], \texttt{vwgt}[n], \texttt{adjncy}[2m], \text{ and } \texttt{adjwgt}[2m]$$

Note that the reason both `adjncy` and `adjwgt` are of size $2m$ is because for each edge between vertices v and u we actually store (v, u) as well as (u, v). Also note that in the case in which the graph is unweighted (*i.e.*, all vertices and edges have the same weight), then the arrays `vwgt` and `adjwgt` can be set to NULL.

The adjacency structure of the graph is stored as follows. Assuming that vertex numbering starts from 0 (C style), then the adjacency list of vertex i is stored in array `adjncy` starting at index `xadj[i]` and ending at (but not including)



(a) A sample graph

Description of the graph on a serial computer (serial MeTiS)

xadj	0 2 5 8 11 13 16 20 24 28 31 33 36 39 42 44
adjncy	1 5 0 2 6 1 3 7 2 4 8 3 9 0 6 10 1 5 7 11 2 6 8 12 3 7 9 13 4 8 14 5 11 6 10 12 7 11 13 8 12 14 9 13

(b) Serial CSR format

Description of the graph on a parallel computer with 3 processors (ParMeTiS)

Processor 0:	xadj	0 2 5 8 11 13
	adjncy	1 5 0 2 6 1 3 7 2 4 8 3 9
	vtxdist	0 5 10 15
Processor 1:	xadj	0 3 7 11 15 18
	adjncy	0 6 10 1 5 7 11 2 6 8 12 3 7 9 13 4 8 14
	vtxdist	0 5 10 15
Processor 2:	xadj	0 2 5 8 11 13
	adjncy	5 11 6 10 12 7 11 13 8 12 14 9 13
	vtxdist	0 5 10 15

(c) Distributed CSR format

Figure 2: An example of the parameters passed to PARMETIS in a three processor case. The arrays `vwgt` and `adjwgt` are assumed to be NULL.

index `xadj[`$i+1$`]` (*i.e.*, `adjncy[xadj[`i`]]` through and including `adjncy[xadj[`$i+1$`]-1]`). That is, for each vertex i, its adjacency lists is stored in consecutive locations in the array `adjncy`, and the array `xadj` is used to point to where it begins and where it ends. Figure 2(b) illustrates the CSR format for the 15-vertex graph shown in Figure 2(a). If the graph was weights on the vertices, then `vwgt[`i`]` is used to store the weight of vertex i. Similarly, if the graph has weights on the edges, then the weight of edge `adjncy[`j`]` is stored in `adjwgt[`j`]`. For those familiar with the METIS package, this is exactly the format that is used by the user callable library of METIS.

Distributed CSR Format PARMETIS extends the CSR format for the case in which the vertices of the graph and their adjacency lists are distributed among the various processors. In particular, PARMETIS assumes that each processor P_i stores n_i consecutive vertices of the graph and the corresponding m_i edges, so that $n = \sum_i n_i$, and $m = \sum_i m_i$. Each processor now stores its local part of the graph in the four arrays `xadj[`n_i+1`]`, `vwgt[`n_i`]`, `adjncy[2*`m_i`]`, and `adjwgt[2*`m_i`]`, using the CSR storage scheme. Again if the graph is unweighted, the arrays `vwgt` and `adjwgt` can be set to NULL. One way of thinking about this distributed CSR format is as follows. Serially you have a single `xadj` and `adjncy` arrays. When you go parallel, the vertices and their adjacency lists are equally distributed among the processors. That is, you can take n/p consecutive adjacency lists from `adjncy` and store them on consecutive processors (p is the number of processors). These portions of the whole `adjncy` array are then the `adjncy` arrays supplied by each of the processors in the distributed CSR format. In addition, each processor supplies its local `xadj` array to point to where each adjacency list begins and ends. Thus, if we take all the local `adjncy` arrays and concate-

nate them will get exactly the `adjncy` array that is used in the serial CSR. However, concatenating the local `xadj` arrays will not give us the serial `xadj` array, since the entries in each local `xadj` point to their local `adjncy` array.

In addition to these four arrays, each processor also supplies an additional array `vtxdist[p+1]`, that indicates the range of vertices stored at each processor. In particular, processor P_i stores the vertices starting from `vtxdist[i]`, up to (but not including) vertex `vtxdist[i+1]`. Figure 2(c) illustrates the distributed CSR format by an example on a three processor system. The 15-vertex graph in Figure 2(a) is distributed among the processors so that each processor gets 5 vertices and their corresponding adjacency lists. That is, processor 0 gets vertices 0 through 4, processor 1 gets vertices 5 through 9, and processor 2 gets vertices 10 through 14. This figure shows the elements that the arrays `xadj`, `adjncy`, and `vtxdist` store at each processor. Note that the array `vtxdist` will always be the same at each processor.

All five arrays that describe the distributed CSR format are defined in PARMETIS to be of type `idxtype`. By default `idxtype` is set to be equivalent to type `int` (*i.e.*, integers). However, `idxtype` can be made to be equivalent to a `short int` for certain architectures that use 64-bit integers by default. The conversion of `idxtype` from `int` to `short` can be done by modifying the file `struct.h` (instructions are included there). The same `idxtype` is used for the arrays that are used to store the computed partition and permutation vector.

3.2 Format of Vertex Coordinates

As discussed in Section 2.1 PARMETIS provides routines that can use the coordinate information of the vertices (when available) to quickly redistribute the graph in a fashion that will speedup the execution of the parallel k-way partitioning algorithms.

These coordinates are specified in an array called `xyz` of single precision floating point numbers (*i.e.*, `float`). If d is the number of dimensions (which is also supplied as a parameter) of the mesh (*i.e.*, $d = 2$ for 2D meshes or $d = 3$ for 3D meshes), then each processor supplies an array of size $d * n_i$ where n_i is the number of vertices that it stores. In this array, the coordinates of vertex i are stored starting at location `xyz[i*d]` up to (but not including) location `xyz[i*d+d]`. For example, if $d = 3$, then the x, y, and z coordinates of vertex i are stored in locations `xyz[3*i]`, `xyz[3*i+1]`, and `xyz[3*i+2]`, respectively.

3.3 Format of the Computed Partitions and Orderings

Format of the Partition Array In all the partitioning routines of PARMETIS each processor also supplies an array called `part` which stores the new partition found by PARMETIS. Each processor P_i supplies an array of size n_i (*i.e.*, the number of vertices stored at this processor), and upon return, for each vertex j, `part[j]` stores the partition number (*i.e.*, the processor label) that this vertex belongs to in the new partition.

Note that PARMETIS does not redistribute the graph according to the new partition, but it simply computes it and returns it to the calling program.

Format of the Permutation Array When PARMETIS is used to compute a fill-reducing ordering by calling `PAROMETIS`, the result of the ordering is returned in an array called `order`. Similar to the `part` array, the size of `order` is equal to the number of vertices stored at each processor. Upon return, for each vertex j, `order[j]` stores the new global number of this vertex in the fill-reducing permutation.

Besides the ordering vector, `PAROMETIS` also returns an additional array that stores information about the sizes of the different subdomains as well as the separators at different levels. This array is called `sizes` and is of size $2p$

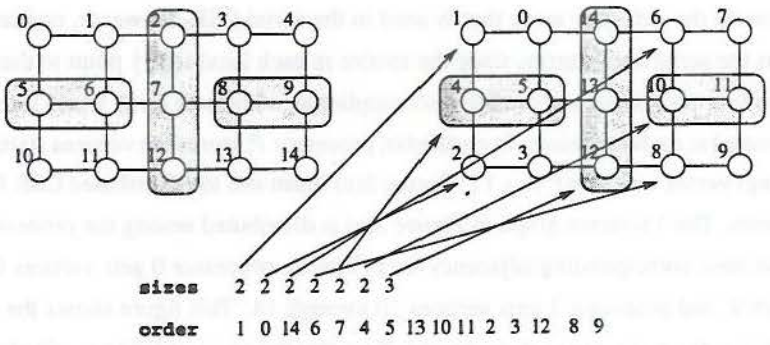


Figure 3: An example of the ordering produced by PAROMETIS. Consider the simple 3×5 grid and assume that we have four processors. PAROMETIS finds the three separators that are shaded. It first find the big separator and then for each of the two subdomains it finds the two smaller. At the end of the ordering, the `order` vector concatenated over all the processors will be the one shown. Similarly, the `sizes` array will be the one shown, corresponding to the regions pointed by the arrows.

(where p is the number of processors). Every processor supplies this array and upon return, this array is filled with the size information. Every processor receives exactly the same information.

The format of this array is as follows. The first p entries of `sizes` starting from 0 to $p-1$ store the number of nodes in each one of the p subdomains. The remaining $p-1$ entries of this array starting from `sizes[p]` up to `sizes[2p-2]` store the sizes of the separators at the log p levels of nested dissection. In particular, `sizes[2p-2]` stores the size of the top level separator, `sizes[2p-4]` and `sizes[2p-3]` store the sizes of the two separators at the second level (from left to right). Similarly, `sizes[2p-8]` through `sizes[2p-5]` store the sizes of the four separators of the third level (from left to right), and so on. This array can be used to quickly construct the separator tree (a form of an elimination tree) for direct factorization. Given this separator tree and the sizes of the subdomains, the nodes in the ordering produced by PAROMETIS are numbered in a postorder fashion. The use of `sizes` and the postorder ordering produced by PAROMETIS based on this tree is shown in Figure 3.

3.4 Numbering and Memory Allocation

PARMETIS allows you to specify a graph that the numbering starts either from 0 (C style) or from 1 (Fortran style). PARMETIS requires that same numbering scheme to be consistently used for all the arrays passed to and also numbers the returned arrays (`part` and `order`) in a similar fashion.

PARMETIS allocates all the memory it requires dynamically. This has the advantage that the user does not have to provide workspace, but if there is not enough memory, the routines in PARMETIS abort. Note that the routines in PARMETIS do not modify the arrays that store the graph.

4 Calling Sequence of the Routines in PARMETIS

The calling sequences of the routines provided by PARMETIS are described in the rest of this section.

PARKMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, idxtype *part, int *options, MPI_Comm comm)

Description

It is used to compute a p-way partition of a graph on p processors using the multilevel p-way partitioning algorithm.

Parameters

vtxdist, xadj, vwgt, adjncy, adjwgt

The various arrays describing the graph stored at each processor. Both `vwgt` and `adjwgt` can be set to `NULL`.

part

The array that returns the result of the partition.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is as follows:

options[0]	Returns the edge-cut of the computed partition (OUT).
options[1]	A parameter that controls successive folding ***folding factor*** (IN).
options[2]	Selects the type of initial partitioning algorithm at the coarsest graph (IN). If it is equal to 1, then a serial partitioning algorithm is used if it is equal to 2, then a parallel algorithm is used. Only for power of 2 partitions.
options[3]	It is used to specify whether or not the numbering starts from 0 or 1 (IN). If it is equal to 0, then the numbering starts from 0 (C style), and if it is equal to 1, then the numbering starts from 1 (Fortran style).
options[4]	Specifies the level of information to be returned during the execution of the algorithm (IN).

The folding factor is used to optimize the run time of the partitioning algorithm. If it is set to 0, no folding is performed and the partitioning algorithm may run somewhat slowly. The optimal value for this parameter depends on the underlying parallel computer as well as the graph being partitioned. Extensive tests on T3D show that a value of 150 produces acceptable results. In general, as the latency of the interconnection network increases, this parameter should also increase.

The initial partitioning option (*i.e.*, options[2]) is used to select how the partitioning is done at the coarsest graph. If the serial partitioning algorithm is selected, then each processor partitions the coarsest (*i.e.*, smallest) graph serially. If the parallel partitioning is selected, the coarsest graph is partitioned using a parallel formulation of multilevel recursive bisection. This parallel formulation is particularly useful for very large number of processors (over 128). Note that parallel initial partitioning currently works only with partitions that are power of two. If the number of partitions (*i.e.*, processor) is not a power of two and parallel partitioning is selected, `PARKMETIS` automatically switches to the serial partitioner.

The value for options[4] should be set to 0. Timing information can be obtained by setting it to 1. Additional options for this parameter can be obtained by looking at the end of the file `defs.h`. The numerical values there need to be added to obtain the results.

comm

This is the MPI communicator of the processes that call PARMETIS. For most programs this will be `MPI_COMM_WORLD`.

PARGKMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, int ndims, float *xyz, idxtype *part, int *options, MPI_Comm comm)

Description

It is used to compute a p-way partition of a graph on p processors by combining coordinate-based partitioning and the multilevel p-way partitioning algorithm.

Parameters

vtxdist, xadj, vwgt, adjncy, adjwgt

The various arrays describing the graph stored at each processor. Both `vwgt` and `adjwgt` can be set to `NULL`.

ndims, xyz

Used to specify the number of dimensions and the coordinates of the vertices, respectively.

part

The array that returns the result of the partition.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is identical to that of `PARKMETIS`.

comm

This is the MPI communicator of the processes that call ParMETIS. For most programs this will be `MPI_COMM_WORLD`.

PARGRMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, int ndims, float *xyz, idxtype *part, int *options, MPI_Comm comm)

Description

It is used to compute a p-way partition of a graph on p processors by combining coordinate-based partitioning and the multilevel p-way refinement algorithm.

Parameters

vtxdist, xadj, vwgt, adjncy, adjwgt

The various arrays describing the graph stored at each processor. Both `vwgt` and `adjwgt` can be set to `NULL`.

ndims, xyz

Used to specify the coordinates of the vertices. The format is identical to the one used by `PARGKMETIS`.

part

The array that returns the result of the partition.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is as follows:

options[0]	Returns the edge-cut of the computed partition (OUT).
options[1]	It is not used.
options[2]	It is not used.
options[3]	It is used to specify whether or not the numbering starts from 0 or 1 (IN). Similar to options[3] of `PARKMETIS`.
options[4]	Specifies the level of information to be returned during the execution of the algorithm (IN). Similar to options[4] of `PARKMETIS`.

comm

This is the MPI communicator of the processes that call ParMETIS. For most programs this will be `MPI_COMM_WORLD`.

Notes

The quality of the produced partitions are moderately worse than those produced by `PARGKMETIS` or `PARKMETIS`.

PARGMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *adjncy, int ndims, float *xyz, idxtype *part, int *options, MPI_Comm comm)

Description

It is used to compute a p-way partition of a graph on p processors by using coordinate-based space-filling curves.

Parameters

vtxdist, xadj, adjncy

The various arrays describing the graph stored at each processor. Note that this partitioning scheme operates on unweighted graphs.

ndims, xyz

Used to specify the coordinates of the vertices. The format is identical to the one used by `PARGKMETIS`.

part

The array that returns the result of the partition.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is as follows:

options[0]	It is not used.
options[1]	It is not used.
options[2]	It is not used.
options[3]	It is used to specify whether or not the numbering starts from 0 or 1 (IN). Similar to options[3] of `PARKMETIS`.
options[4]	Specifies the level of information to be returned during the execution of the algorithm (IN). Similar to options[4] of `PARKMETIS`.

comm

This is the MPI communicator of the processes that call ParMETIS. For most programs this will be `MPI_COMM_WORLD`.

Notes

The quality of the produced partitions are significantly worse than those produced by `PARGKMETIS` or `PARKMETIS`, but `PARGMETIS` is very fast.

`PARGMETIS` cannot handle graphs with weights on vertices or the edges.

PARRMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, idxtype *part, int *options, MPI_Comm comm)

Description

It is used to improve the quality of an existing a p-way partition p processors by using the multilevel p-way refinement algorithm.

Parameters

vtxdist, xadj, vwgt, adjncy, adjwgt

The various arrays describing the graph stored at each processor. Both `vwgt` and `adjwgt` can be set to `NULL`.

part

The array that returns the result of the partition.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is as follows:

options[0]	Returns the edge-cut of the computed partition (OUT).
options[1]	It is not used.
options[2]	It is not used.
options[3]	It is used to specify whether or not the numbering starts from 0 or 1 (IN). Similar to options[3] of `PARKMETIS`.
options[4]	Specifies the level of information to be returned during the execution of the algorithm (IN). Similar to options[4] of `PARKMETIS`.

comm

This is the MPI communicator of the processes that call PARMETIS. For most programs this will be `MPI_COMM_WORLD`.

PARUAMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, idxtype *part, int *options, MPI_Comm comm)

PARDAMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, idxtype *part, int *options, MPI_Comm comm)

Description

They are used to load balance the work load of a graph that corresponds to a mesh that has been adaptively refined. They utilize multilevel diffusion algorithms.

Parameters

vtxdist, xadj, vwgt, adjncy, adjwgt

The various arrays describing the graph stored at each processor. Both `vwgt` and `adjwgt` can be set to `NULL`.

part

The array that returns the result of the partition.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is as follows:

options[0]	Returns the edge-cut of the computed partition (OUT).
options[1]	It is not used.
options[2]	It is not used.
options[3]	It is used to specify whether or not the numbering starts from 0 or 1 (IN). Similar to options[3] of `PARKMETIS`.
options[4]	Specifies the level of information to be returned during the execution of the algorithm (IN). Similar to options[4] of `PARKMETIS`.

comm

This is the MPI communicator of the processes that call ParMETIS. For most programs this will be `MPI_COMM_WORLD`.

Notes

If the adapted graph is significantly different from the original graph, repartitioning the graph from scratch using either `PARKMETIS` or `PARGKMETIS` may be a more appropriate solution in order to minimize the edge-cut.

PAROMETIS (idxtype *vtxdist, idxtype *xadj, idxtype *vwgt, idxtype *adjncy, idxtype *adjwgt, idxtype *order, idxtype *sizes, int *options, MPI_Comm comm)

Description

It is used to compute a fill-reducing ordering of a sparse matrix by using multilevel nested dissection.

Parameters

vtxdist, xadj, vwgt, adjncy, adjwgt

The various arrays describing the graph stored at each processor. Both `vwgt` and `adjwgt` can be set to `NULL`.

order

The array that returns the result of the ordering.

sizes

The array that returns the number of nodes of each subdomain and each separator.

options

This is an array of 5 integers that is used to pass parameters in and out of the program. Their meaning is as follows:

options[0]	It is not used.
options[1]	It is not used.
options[2]	Selects the type of initial partitioning (IN). If it is equal to 1, then an edge-based bisection is computed at the coarser graph, if it is equal to 2, then a node-based bisection is computed.
options[3]	It is used to specify whether or not the numbering starts from 0 or 1 (IN). Similar to options[3] of `PARKMETIS`.
options[4]	Specifies the level of information to be returned during the execution of the algorithm (IN). Similar to options[4] of `PARKMETIS`.

The initial partitioning option (*i.e.*, options[2]) is used to select how the partitioning is done at the coarsest graph. If the edge-based bisection is selected, then the algorithm first computes a bisection and then uses a min-cover algorithm to compute the vertex separator. If the node-based bisection is selected, then the algorithm directly computes a vertex separator. In either case, the refinement is done using vertex-based refinement schemes.

comm

This is the MPI communicator of the processes that call PARMETIS. For most programs this will be `MPI_COMM_WORLD`.

Notes

`PAROMETIS` requires that the number of processors is a power of 2.

The current version of `PAROMETIS` switches to multiple minimum degree after log p levels of nested dissection. In many cases, the orderings produced on larger number of processors can lead to smaller fill, as more nested dissection levels are performed.

5 Contact Information

Instructions on how to build PARMETIS are available in the file called `INSTALL` in PARMETIS's distribution. Also, in the directory called `TEST` you will find a program that tests if PARMETIS was built correctly. You can use the file `main.c` in the `TEST` directory as an example of how to invoke the routines in PARMETIS.

Also, a header file called `parmetis.h` is provided that contains prototypes for the functions in PARMETIS.

PARMETIS have been extensively tested on many different parallel computers. However, even though PARMETIS contains no known bugs, it does not mean that all of its bugs have been found and fixed. If you find any problems, please send email to *karypis@cs.umn.edu*, with a brief description of the problem you have found.

References

[1] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1995. Available on the WWW at URL *http://www.cs.umn.edu/~karypis/metis/metis.html*.

[2] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 95-064, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[3] George Karypis and Vipin Kumar. A coarse-grain parallel multilevel k-way partitioning algorithm. In *Proceedings of the eighth SIAM conference on Parallel Processing for Scientific Computing*, 1997.

[4] Alex Pothen, H. D. Simon, Lie Wang, and Stephen T. Bernard. Towards a fast implementation of spectral nested dissection. In *Supercomputing '92 Proceedings*, pages 42–51, 1992.

[5] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel multilevel diffusion algorithms for repartitioning of adaptive meshes. In *Submitted to Journal of Parallel Computing*, 1997.