

SANDIA REPORT

SAND93-2339 • UC-405

Unlimited Release

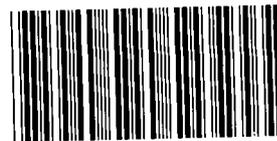
Printed November 1993

MICROFICHE

The Chaco User's Guide Version 1.0

Bruce Hendrickson, Robert Leland

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000



8605018

SANDIA NATIONAL
LABORATORIES
TECHNICAL LIBRARY

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

The Chaco User's Guide* Version 1.0

Bruce Hendrickson[†] and Robert Leland[‡]
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

Graph partitioning is a fundamental problem in many scientific settings. This document describes the capabilities and operation of **Chaco**, a software package designed to partition graphs. **Chaco** allows for recursive application of any of several different methods for finding small edge separators in weighted graphs. These methods include inertial, spectral, Kernighan-Lin and multilevel methods in addition to several simpler strategies. Each of these methods can be used to partition the graph into two, four or eight pieces at each level of recursion. In addition, the Kernighan-Lin method can be used to improve partitions generated by any of the other methods. Brief descriptions of these methods are provided, along with references to relevant literature. The user interface, input/output formats and appropriate settings for a variety of code parameters are discussed in detail, and some suggestions on algorithm selection are offered.

* This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract No. DE-AC04-76DP00789. The **Chaco** software package was developed by the authors at Sandia National Laboratories and is under copyright protection.

[†] Department 1422; electronic mail: bahendr@cs.sandia.gov; telephone: 505-845-7599.

[‡] Department 1424; electronic mail: rwlelan@cs.sandia.gov; telephone: 505-845-7387.

1. Introduction. Many problems which arise in the course of scientific computing have a combinatorial nature and can be conveniently described in terms of graphs. In particular, it is often useful to partition a graph into subgraphs that are in some measure as disjoint as possible. This is the case in divide-and-conquer algorithms for problems like devising efficient circuit layouts or constructing nested dissection orderings for sparse matrix factorizations. Another prominent instance is the problem of decomposing a large data structure to be mapped onto the processors of a parallel computer. This latter example motivated the development of **Chaco**¹, and we will assume that it is the main interest of the reader in the remainder of this user's guide. However we wish to emphasize that the code and its output can be used directly or with slight modification to address other important problems as well.

For concreteness, assume we want to solve a partial differential equation on a distributed memory parallel computer. If we use a finite difference discretization and an iterative solver, the graph to be partitioned will typically have the same topology as the computational grid: The iterate at each grid point must be updated using neighboring grid point values, so graph vertices correspond to the update computation, and graph edges indicate that information must be transferred from one grid point to another. On a serial computer this transfer is accomplished by writing to and reading from memory. However, when we map this computational grid to a parallel computer, two vertices joined by an edge and not owned by the same processor must communicate to exchange values. Since communication is expensive, a mapping that minimizes it is desirable. Of course, we could assign the entire grid to a single processor and have no communication at all, but that wouldn't be an effective use of the parallel machine since one processor would do all the work while the others remained idle. We must therefore also observe the important constraint that each processor should be assigned about the same amount of vertex work and therefore (in the simplest case) the same number of vertices. Hence we say informally that the objective of **Chaco** is to produce *balanced* sets with low communication overhead.

Not all problems have such a convenient correspondence between the computational grid and the mapping requirements of the application program. For instance in a finite element calculation, a more appropriate approach may be to consider each element as a vertex with some associated update work. We would then construct connecting edges corresponding to each face or corner in the discretization mesh since these edges correspond to the non-zero pattern in the global stiffness matrix. The most appropriate graph will depend upon the application and is left up to the user.

¹ **Chaco** is named in honor of Chaco Canyon, the site of spectacular Anasazi ruins in what is presently northwestern New Mexico. Between 1000 and 1100 AD a great society, considered the most complex and sophisticated on the continent north of Mexico, flourished there.

Furthermore, all vertices are not necessarily created equal. For example, a vertex encoding a computation on the boundary may have less work associated with it than a vertex in the interior of a domain. For this reason, **Chaco** allows weights to be associated with each vertex. The weight is supposed to correspond to the amount of work associated with the vertex. Similarly, edges may correspond to varying amounts of communication. For example, two finite elements touching at a corner may need to exchange less information than two sharing a face. **Chaco** also allows for a weight to be associated with each edge, corresponding to the amount of communication it represents.

The problem of interest can now be described more precisely. Given a graph G with n weighted vertices and m weighted edges, divide the vertices into p sets in such a way that the sum of the vertex weights in each set is as close as possible, and the sum of the weights of edges crossing between sets is minimized. Unfortunately, even in the simple case where $p = 2$ and the edge and vertex weights are uniform, this graph partitioning problem is NP-complete[4]. Hence there is no known efficient algorithm to solve the problem generally, and it seems unlikely that such an algorithm exists. We must therefore resort to heuristic solutions in which balance may be partially compromised or (more typically) the minimization is approximate.

A variety of such heuristic methods with different cost/quality tradeoffs have been published. **Chaco** includes methods based on several of these as well as several substantially new methods. The algorithms used are based on inertial, spectral, Kernighan–Lin (KL) and multilevel principles in addition to several simpler strategies. The methods are categorized as either local (currently just KL) or global (everything else). **Chaco** allows you to combine global and local methods, and we have found that this combination leads to significant improvements in both performance and robustness. Another advantage of **Chaco**'s design philosophy is that it offers flexibility. This is important because we believe that, given the complexity of the partitioning problem, *no single method will always work well*. **Chaco** gives you a fall-back option when your favorite method works poorly or has an inappropriate cost/quality ratio for a given problem. It also facilitates investigation into the relative strengths and weakness of a wide variety of methods.

Having set the basic context, we should raise some finer but nevertheless important issues. One such issue is the *dimensionality* of the partitioning scheme. Most graph partitioning codes rely on recursive bisection. That is, the graph is partitioned into two pieces, each of these pieces is partitioned into two more, etc. until a desired number of sets is reached. This strategy is simple and convenient, but may be somewhat limiting. Graphs can be constructed for which any bisection algorithm must necessarily perform poorly, and in practice we observe that bisection algorithms often choose separators

which look very good at one stage of recursion but not so good with the benefit of hindsight at a later stage. All of the partitioning algorithms implemented in **Chaco** are capable of partitioning graphs into two, four or eight sets at each stage of recursion. We have accumulated some empirical evidence that the quadrisection and octasection algorithms do perform better in some respects than their bisection counterparts. We have also found bisection algorithms preferable to their multi-dimensional versions in some situations. For readers interested in the interaction between communication metric and partitioning dimensionality we can recommend several previous reports [7, 8, 10].

The basic difficulty in choosing the appropriate partitioning dimensionality is that the correct representation of communication costs in the graph model is somewhat ambiguous. Most graph partitioning schemes work to suppress the total number of edges crossing between sets without regard to the identity of the sets ². We say these methods try to minimize the total number of *cuts*. In contrast, several of the multidimensional schemes we have developed can take into account the identity of the sets an edge crosses between and work to minimize the hypercube distance between these sets. We say they try to minimize the total number of *hops*. For hypercube architectures and for 2D and 3D mesh architectures in some situations, the hypercube metric is more appropriate because it better models message congestion. In other cases it may be preferable to focus simply on the total number of bytes communicated and hence rely on a bisection scheme. When the communicated messages are short enough, the total communication time will correlate best with message *startups*. In the graph metric this measure corresponds to the number of neighboring sets each set has. We have also included a method designed to deal with this contingency by suppressing the maximum number of neighbors any set has. In fact, with an isolated change to the source code, **Chaco** can implement some methods with an arbitrary cost function. However, all of the methods currently implemented in **Chaco** nevertheless share the limitation that we must have $p = 2^k$ for a whole number k . While this limitation applies to most graph partitioning algorithms and codes, it is not fundamental, and we intend that future releases of **Chaco** will allow non-power-of-two partitioning.

The methods currently implemented in **Chaco** are described in the next section. In §3 we describe the input format and the menu options used to invoke the different methods. In §4 we discuss several easily modified parameters which allow the user to fine tune the code for a particular application. This section can be skipped on a first reading. Finally, in §5 we give some practical advice on obtaining, installing and using **Chaco**.

² For simplicity let us consider the unweighted graph model in this discussion.

2. Partitioning algorithms in Chaco. The five classes of partitioning algorithms currently implemented in **Chaco** are simple, spectral, inertial, Kernighan–Lin and multilevel. These methods are briefly described below, and references to appropriate literature are provided.

2.1. Simple Partitioning. For completeness and in order to facilitate certain comparisons, **Chaco** includes three very simple partitioning schemes. In the *linear* scheme, vertices are assigned in order to processors in accord with their numbering in the original graph, *i.e.* the first n/p vertices are assigned to set 0, the next to set 1, etc. This often produces surprisingly good results because data locality is implicit in the numbering of the graph. In the *random* scheme, vertices are assigned randomly to sets in a way that preserves balance, and in the *scattered* scheme vertices are dealt out card fashion to the p sets in the order they are numbered. These methods are like all the others in that they operate recursively and produce two, four or eight partitioned sets at each stage of recursion. Usually the random ordering produces partitions with quality between that of the linear and scattered partitioning. The run time of the random scheme is very small and of the other schemes is negligible.

2.2. Spectral Partitioning. Most of the code in **Chaco** is devoted to spectral methods. These methods use eigenvectors of a matrix specially constructed from the graph to decide how to partition it. A full accounting of this surprising connection between eigenvectors and partitions is too involved to present here, but the articles mentioned below offer plenty of detail on the subject.

The simplest spectral method in the code is a weighted version of *spectral bisection*. A description of the unweighted algorithm is given in [16, 17], and the extension to use both edge and vertex weights is described in [7]. This method uses the second lowest eigenvector of the *Laplacian* matrix of the graph to divide the graph into two pieces. This eigenvector is known as the *Fiedler* vector.

The *spectral quadrisection* algorithm divides a graph into four pieces at once using the second and third lowest eigenvectors of the Laplacian matrix. Similarly, *spectral octasection* uses the second, third and fourth eigenvectors to divide into eight pieces. These *multidimensional* spectral methods were introduced in [7, 8] where they were shown to have some advantages over spectral bisection.

In particular, we note that spectral quadrisection and octasection try to minimize communication cost in a more complex metric. Suppose the partitioned sets are numbered from 0 to 3 for quadrisection or 0 to 7 for octasection. Spectral bisection would try to minimize the total weight of edges crossing between different sets, whereas the multidimensional methods would use a metric in which the cost of an edge crossing between two sets is the edge weight multiplied by the number of bits that are different

in a binary representation of the two sets.

Although this *hops* metric may seem odd at first glance, it has a nice interpretation in the context of parallel computing. In a parallel computer consisting of four processors connected in a mesh, and numbered in the natural way, a message traveling between processors 0 and 3 must travel over two wires, whereas one between 0 and 1 need only traverse a single wire. This number of wires is exactly the weighting implicit in spectral quadrisection. Similarly, spectral octasection counts wires used on a three dimensional mesh architectures, and both quadrisection and octasection applied recursively do so for higher dimensional hypercubes.

One might suppose that this correspondence between cost metric and wires used was irrelevant given the advent of cut-through routing in which the delay associated with a message is nearly independent of the number of links it traverses. In fact this independence only holds for isolated messages in which there is no competition for the links in the communication network. In a great many computations, and most scientific applications, communication occurs in the form of bursts of messages during which there is very significant competition for the network. Hence when network congestion is important, weighting messages by the number of wires they consume should lead to better problem mappings. Empirical evidence supporting this and further discussion of the issue can be found in [6].

The computational kernel of spectral methods is the calculation of a small number of eigenvectors. We have implemented a variety of eigensolvers with different speed/robustness tradeoffs. Roughly in order of increasing speed, these are Lanczos with full orthogonalization, Lanczos with full orthogonalization using the inverse operator, Lanczos with selective orthogonalization against both ends of the spectrum, Lanczos with selective orthogonalization against the left end only, and a multilevel method combining Rayleigh Quotient Iteration[5] and the linear solver Symmlq[14]. Several of the issues governing the choice between these methods are touched upon below. It should be noted that our conclusions are based on limited testing with the particular class of matrices arising in these applications, and may not be applicable to any wider domain. *These are all iterative methods!*

In our experience, full orthogonalization Lanczos is the most robust method for problems of order up to a few hundred. The requirement of saving all the Lanczos vectors for orthogonalization is not that burdensome since the problems are small and we use them anyway in assembling the eigenvectors. The weak point of this method is that for larger problems the orthogonalization work becomes prohibitively expensive.

The inverse operator full orthogonalization Lanczos method replaces the matrix vector multiply in the basic Lanczos iteration with a linear solve using Symmlq. It is generally less accurate and robust than direct Lanczos with full orthogonalization

and is often slower as well because the total number of matrix vector multiplies (which are hidden within `Symmlq`) may be significantly higher. In addition it introduces the tricky problem of how to tune the inner/outer loop combination. Thus the only reason to recommend this method is that it requires much less memory since it converges in many fewer Lanczos iterations.

Our implementation of selective orthogonalization is based on the original paper by Parlett and Scott [15], with the main difference being that the Ritz spectrum is monitored directly to assess the need for orthogonalization. A bisection algorithm on the Sturm sequence and various heuristics governing which Ritz pairs to monitor are used to keep this overhead small. Most of the orthogonalization work occurs at the right end of the spectrum, and is, it turns out, unnecessary. Orthogonalizing at the left end only generally produces more accurate eigenpairs in substantially less time. This latter algorithm seems, for our purposes, essentially as accurate as full orthogonalization and is our method of choice for small and medium sized systems (up to about 10,000 vertices), provided sufficient memory is available. Since all the Lanczos vectors must be saved for the contingency that the iterate must be orthogonalized against a convergent Ritz vector, this method can cause the program to run out of memory on very large systems. This difficulty can be avoided by employing a restarting scheme or by giving up on maintaining orthogonality in the Lanczos basis. These alternatives are, however, slower and, in the latter case, impose an added risk of numerical breakdown. We decided to optimize over the likely range of application and assumed that for problems in which memory would be a problem for Lanczos, a partitioning method designed for larger problems would be employed.

For partitioning larger graphs by the spectral method, we recommend the multi-level RQI/`Symmlq` eigensolver. This is based on the method developed by Barnard and Simon [1], with the main difference being that we have used an edge contraction coarsening scheme based on the physical analogy described in [10]. This contraction scheme preserves the low modes of the operator sufficiently well that we need only perform RQI refinement periodically as we work back through the grid hierarchy. We have also modified the `Symmlq` iteration to terminate when the norm of the iterate reaches a preset limit since RQI is essentially performing inverse iteration. The resulting method is several times faster than Lanczos with selective orthogonalization for solving large problems to the same accuracy, and also requires far less memory. A drawback is that the method seems more prone to misconvergence than Lanczos. Experience indicates, however, that for large graphs, eigenvectors other than the Fiedler vector usually give partitions of similar quality to those generated with the Fiedler vector (occasionally better!). So slight misconvergence is not that serious a problem, especially if you are applying a local cleanup scheme. Another drawback of the RQI/`Symmlq` algorithm is

that its run time is essentially proportional to the number of eigenvectors solved for. This erodes its speed advantage when used as the eigensolver for one of the multidimensional spectral partitioning schemes.

A critical issue in the proper use of iterative eigensolvers is the choice of the tolerance on the eigen residual. This is treated in some detail later during the discussion of the various code parameters in §4, but it is appropriate to mention here that all of the eigensolvers have direct residual checks to determine whether the requested eigen tolerance has been achieved. In addition, the selective orthogonalization schemes have safety checks to monitor the effectiveness of the orthogonalization, and the multilevel RQI/Symm1q code incorporates a heuristic to detect misconvergence. From time to time and depending upon how the error and warning condition flags are set, one or more of these conditions will be noted by **Chaco**. In most cases these are not show stoppers: the desired safety standards have not been met, but the computation will proceed and generate reasonable partitions. If certain error or warning conditions occur chronically, you may need to choose different tuning parameters. (Or, of course there may be a problem with the code.)

In general, spectral methods are quite good at finding promising regions of the graph in which to cut. However, they often do poorly in the fine details. Consequently, we have found that it is advantageous to apply a local cleanup procedure to the spectral output. The procedure we use is a generalized version of an algorithm due to Kernighan–Lin, and is described below in §2.4 and in more detail in [9]. The actual improvement due to this cleanup phase is problem dependent, but is typically 10-30%. The cost of this cleanup phase is generally a small fraction of the total partitioning cost, typically less than 10% on large graphs.

2.3. The inertial method. The *inertial bisection* method is a relatively simple and fast partitioning strategy that uses geometric information. In addition to a graph, the user supplies geometric coordinates for each vertex in one, two or three dimensions. The code considers the vertices as point masses with mass equal to the vertex weight. The principle axis of this collection of point masses, which is likely to be a direction in which the graph is elongated, is found. The vertices are then divided into sets of equal mass by plane(s) orthogonal to the principle axis. Descriptions of this method can be found in [13, 17].

Chaco allows inertial partitioning into two, four or eight sets at once. This is accomplished by using one, three or seven planes, each of which is orthogonal to the principle axis. Partitions generated by inertial quadrisection or octasection will appear to be banded, with parallel planes dividing the sets. This “striping” will typically lead to a fairly large surface-to-volume ratio, indicating a large volume of communication.

However, each set only has a small number of neighboring sets which helps reduce the number of messages startups each processor must make. If the cost of initiating messages is important, then partitions using inertial quadrisection or octasection may lead to faster application execution times than those generated with inertial bisection. Furthermore, the multidimensional inertial methods are somewhat faster than inertial bisection since fewer inertial axes must be computed, and the overhead of recursion is avoided. Currently, the four or eight sets are assigned to processors in such a way that neighboring sets go to adjacent hypercube processors. This *gray coding* may not be optimal for other architectures and can be switched off by modifying the routine “rec_median_1” in the file “/code/assign/rec_median.c”.

In our experience, inertial methods are quite fast but give partitions of fairly low quality in comparison with spectral methods. In particular, the local details of a partition are often quite poor. However, when coupled with the Kernighan–Lin local optimization method described below, the results significantly improve. Our experiments indicate that inertial plus KL usually produces better partitions than pure spectral partitioning, whereas spectral coupled with KL does better than inertial paired with KL. For very large problems in which coordinates are available and the emphasis is more on low partitioning time rather than high partitioning quality, we are inclined to recommend the inertial plus KL method.

2.4. Kernighan–Lin. One of the most popular methods for partitioning graphs dates back to work done in the early 70’s by Kernighan and Lin [12]. Various extensions and improvements of the original idea have been proposed through the years, including the important linear time implementation of Fiduccia and Mattheyses [3], but at its heart, Kernighan–Lin (KL) is a greedy, local optimization strategy. Vertices are moved between sets in an effort to reduce the cost of the partition. Although the original algorithm was for graph bisection, Suaris and Kedem [18] showed how it could be extended to quadrisection. We have generalized this idea so that our code works on an arbitrary number of sets at once. Unfortunately, the runtime of the algorithm and its memory requirements increase steeply with the number of sets, so in practice we use only bisection, quadrisection and octasection to match the other methods in **Chaco**.

A description of our generalization of KL is contained in [10]. In our experience, KL does not produce very good answers unless it is given a good starting guess. For this reason, we find its value to be greatest when used in conjunction with one of the global partitioners. To test KL essentially on its own, you can invoke the simple random method to provide a starting partition.

2.5. A multilevel method. Our method of choice for large problems in which high quality partitions are sought is the multilevel algorithm described in [9]. This

method is similar in approach to the method described in [2, 11]. It works by creating a sequence of increasingly smaller graphs approximating the original graph, partitioning the smallest graph, and projecting this partition back through the intermediate levels. Kernighan–Lin is invoked every few levels to refine the partition, and the current implementation of the code uses a spectral method to partition the smallest graph.

The algorithm for constructing smaller approximations to the graph relies upon finding a maximal matching in the graph, and then contracting edges in the matching. Edge contraction is intuitively attractive because it largely preserves the graph topology. When edges are contracted, a single vertex is created out of the two endpoints with weight given by the sum of the weights of the endpoints. In addition, any edges which become coincident have their weights summed and become a single edge. These operations have the effect of preserving some of the basic properties of a partition as it is moved between graphs in the hierarchy. The size of the smallest graph is an input option, and the frequency with which to invoke KL is a user modifiable parameter as described in §4.

In our experience, this method gives very high quality answers in moderate time. It is not as quick as the inertial plus KL method, but it generally produces better partitions. In most cases it produces partitions which are better than those generated by spectral plus KL, but runs significantly faster than any of the spectral methods. More on the workings and performance of this multilevel method can be found in [9].

3. Input and output formats. **Chaco** input consists of one or more files, and the response to several interactive queries. Files are used to describe the graph, and if necessary to give geometric coordinates. The interactive input specifies the partitioning method.

3.1. Format of graph input file. The essential **Chaco** input is a graph, which is read from a file. Any lines in this file that begin with the character “%” are considered comments and ignored. The file should contain $n + 1$ non-comment lines, where n is the number of vertices in the graph. At its simplest, the first non-comment line contains two integers. The first integer is the number of vertices in the graph, and the second is the number of edges. Note that the number of edges is half of the sum of the number of neighbors of each vertex. Vertices in the graph are assumed to be numbered from 1 to n . The remaining n non-comment lines contain neighbor lists for each vertex from 1 to n in order. These lists are just sets of integers separated by spaces that contain all the neighbors of the given vertex. The list of neighbors can be in any order. Examples of graph files can be found in subdirectory “executable”; they have names beginning with “graph”.

Chaco also allows for the input of graphs with weights on vertices and/or edges. This is indicated by including a third parameter on the first non-comment line of the input file. This number has three digits. If the 1's digit is nonzero, then edge weights will be read. If the 10's digit is nonzero then vertex weights will be read. And if the 100's digit is nonzero then vertex numbers will be read, as described below.

Edge and vertex weights should have small integer values (to be conservative, the sum of all edge or vertex weights should be representable in a standard integer). If any vertex has a weight, then weights must be given for all of them, and similarly for edge weights. If the edge weight option is selected, then edge weights are included in the graph file immediately after the corresponding entry in the neighbor list. That is, a neighbor list will look like

```
neighbor1 edge-weight1 neighbor2 edge-weight2 ...
```

If the vertex weight option is selected, then each neighbor list must begin with the weight of the vertex the list belongs to.

If for some reason you wish to list the graph vertices in other than the natural order from 1 to n , you can do so by including vertex numbers. The number of a vertex will be the first value on a line comprising a (weighted) neighbor list. The vertex numbers assigned this way must contain the values from 1 to n and only those values.

The most general form of the graph input file is illustrated below. The different optional parameters are indicated by the different styles of parenthesis.

```
% This is the format of the graph input file
Number-of-vertices  Number-of-edges  {1}[1](1)
{Vertex-number} [Vertex-weight]  neighbor1 (edge-weight1) ...
      :
```

There is one exception to this general graph format. If you are using the inertial or one of the simple methods without Kernighan–Lin, then it is not essential to include a graph. The partitioning is based entirely on geometric data. A graph file is still needed to read the number of vertices, but the remaining lines describing the edge lists can be skipped. Note that the code will be unable to evaluate the quality of a partition without the graph. Normally several measures of the partition quality are computed and printed out, but this is skipped if the graph is not present.

3.2. Format of coordinate information input file. If you are using the inertial method partitioning option, you will need to provide geometric coordinates for all vertices. These are placed in a different file, examples of which can be found in subdirectory “executable” with names beginning with “coords”. These geometry files have n lines, and line i contains the coordinates of vertex i . Each line must have 1, 2

or 3 real values, corresponding to a one-, two- or three-dimensional geometry. **Chaco** determines the dimensionality by looking at the number of values on the first line.

3.3. Operating the code. To operate the code you must answer a sequence of questions. With a basic understanding of the code structure and the methods described in §2, these questions should be mostly self-explanatory. A brief outline and a few notes are, however, in order.

First you will be asked to provide the names of the graph input file. If the `OUTPUT_ASSIGN` or `ECHO` parameters from §4.1 are set appropriately, you will also be asked for the names of output files. You will then select a partitioning method from those described in §2. Depending upon your selection, you may need to answer a few additional questions. You must then specify the dimension of the partition, which is simply the \log_2 of the number of partition sets you desire. Finally you will choose whether to apply the partitioning method in bisection, quadrissection or octasection form. Note that if you choose quadrissection or octasection and an integral number of steps will not produce the specified number of steps, **Chaco** will automatically change to either quadrissection or bisection on the last stage of recursion so as to generate the required number of steps.

Because some of the coarsening mechanisms are common to both methods, you are not allowed to invoke the RQI/Symmlq eigensolver and the multilevel partitioning technique at the same time. With either method you will be asked how many vertices you wish to coarsen down to. The coarsening technique removes about half the vertices at each level, and it will continue until the number of vertices is no larger than the limit you specify. We generally use values in the range 50 to 500 for this parameter. Note that because quadrissection and octasection make use of higher frequency information, they may need a slightly larger coarsest graph to resolve things as well as bisection does.

3.4. Output formats. **Chaco** has various output options which are controlled by parameters described later in §4. As these parameters are increased, more detailed information is printed. If they are all set to zero, no output is produced.

The parameter `OUTPUT_METRICS` controls the calculation and printing of several partition metrics. Cuts, hops, number of boundary vertices and number of set neighbors can all be displayed in detailed or summary form. Assorted timing information is displayed under the control of `OUTPUT_TIME`. This information, along with the input parameters and the settings for many of the user accessible internal parameters can be written to either the screen or both the screen and a designated file under the control of the `ECHO` parameter.

In addition, **Chaco** can write an output file containing the partition assignments.

If the graph has n vertices this file will have n lines. Line i contains a single number, indicating the set to which vertex i is assigned. (The set numbers begin at zero.) The generation of this file is controlled by the parameter `OUTPUT_ASSIGN`.

4. User-modifiable parameters. As a convenience, we have collected most of the internal parameters which control the operation of **Chaco** into the file “User_params.c” in the directory “/code”. These parameters can be modified to tune the code to your application. (It might be prudent to save a copy of the original file so that you can return to the “factory settings” easily.) There are two types of parameters, those that change the execution of the program, and those that merely generate additional output for debugging. The default values for the debugging parameters generate a modest amount of output, which can be increased or decreased as desired. The defaults for the execution parameters were selected to provide a reasonable balance between run time and quality of the solution, but we make no claim to having selected them optimally for your problem. The parameters and their functions are described below.

4.1. Input and output control parameters.

CHECK_INPUT If nonzero, the graph and input parameters are checked for errors. Although checking the graph can take a few seconds for large problems, this feature should probably be left active for robustness. (The time for this checking will be printed out if you set the parameter `OUTPUT_TIME` to be greater than zero.)

OUTPUT_TIME This value determines how much information gets printed about the runtime of **Chaco**. A value of 0 means that nothing is printed, and values of 1 and 2 allow for increasingly detailed timing output.

OUTPUT_METRICS This parameter controls how much information about the quality of the computed partition will be computed and printed on the screen. A zero value means that no evaluation will be performed or printed. Values up to a maximum of 3 display increasing amounts of information. The meaning of the output metrics is described in §3.4.

OUTPUT_ASSIGN If this value is nonzero, then you will be prompted for the name of a file in which the vertex assignment will be printed. A description of the format of this output file can be found in §3.4.

ECHO This parameter controls the printing of the values of the input parameters, as well as whether to copy results of the run to a file. If this value is 1 or -1, the input selections will be echoed to the screen. If it is 2 or -2, then the relevant parameters from “User_params.c” will also be echoed. If the value is less than zero, then you will be asked for the name of a file in which to record the results of a run. This file will contain the same input selections and parameters that

are copied to the screen, along with partition metrics and run time breakdown controlled by `OUTPUT_METRICS` and `OUTPUT_TIME`. Saving these results in a file can be useful if you are doing a sequence of runs for later analysis.

4.2. Eigenvector calculation parameters.

EIGEN_TOLERANCE This one probably deserves its own short paper. All we can do here is make a few general remarks and urge caution. If you are using a pure spectral method or the multilevel partitioning method then you need to calculate eigenvectors. This parameter controls how accurately you compute them. It is a tolerance on the eigen residual $\|Au - \lambda u\|$ where (λ, u) is the eigenpair of A in question. An extremely accurate calculation is expensive, and probably unnecessary, particularly if you are using Kernighan–Lin to refine the spectral partition. However, in general the quality of the partition gradually degrades as the accuracy is reduced below some critical point. This can be a result of inaccuracy in the eigenvector, or it may be because the eigensolver has converged to an entirely wrong eigenpair. This latter phenomenon of misconvergence occurs quite frequently if you use too large an eigen tolerance because there are many eigenvalues in any interval of that width. So to be really correct one should probably relate the eigen tolerance to the expected gap between eigenvalues in the relevant portion of the spectrum using, for example, the graph size. But, as discussed earlier in §2, slight misconvergence is not a grave problem since misconverged eigenvectors often give good partitions. The multidimensional spectral methods do in general require somewhat higher accuracy than spectral bisection to perform at their best. Apart from this, however, the question of the appropriate eigen tolerance and risk of misconvergence is more a question of being able to reproduce partitions reliably and of having a fair basis on which to compare eigensolvers. **Chaco**'s design philosophy on this issue is that you should get the accuracy you request, and, failing that, you should be warned and told the accuracy you did get. We feel the largest value of **EIGEN_TOLERANCE** that is advisable for general use is about 10^{-3} , and that is what we ship the code with. If you are really pressing for speed and are using a local cleanup phase, a value of 10^{-2} might be reasonable. At the other extreme, a value of 10^{-6} should prove acceptably tight in most situations — if you're working on a graph large enough to require higher accuracy, you should probably switch to the multilevel partitioning method, which generally gives better answers in less time for large problems.

LANCZOS_SO_INTERVAL If you are using the selective orthogonalization variant of Lanczos, then the convergence of the process is checked indirectly through the Ritz

pairs every few steps. The number of Lanczos iterations between checks is set by the value of this parameter. Choosing a large value will generally make the computation run marginally faster, but increases the risk of degraded accuracy or misconvergence. A smaller value is more robust since numerical breakdown due to the convergence of Ritz pairs will be detected sooner. If you encounter convergence problems while using selective orthogonalization, try reducing this parameter.

BISECTION_SAFETY When using selective orthogonalization, some of the extremal eigenvalues of the tridiagonal matrix must be found periodically (see **LANCZOS_SO_INTERVAL**). If the number of eigenvalues to be found is small, a bisection algorithm is used to find roots of the Sturm sequence which correspond to the eigenvalues. This parameter amplifies or shrinks the convergence tolerance on the bisection algorithm. A higher value specifies a tighter (smaller) tolerance and results in more accurate computation of these eigenvalues, but a slightly longer run time.

LANCZOS_SO_TIME If you desire a detailed breakdown of the time spent in different stages of the Lanczos eigensolver, then this parameter should be set to 1. Lanczos will run very slightly faster if you leave this value at 0, since many fewer calls to the timing function will be made. This may be noticeable if many calls are made to Lanczos.

WARNING_EVECS If this parameter has a value greater than 0, the occurrence of several possible numerical problems in the eigensolvers is monitored. When using RQI/Symmlq, a value above 0 means you will be notified if the eigen residual is not converging monotonically, an indication of possible misconvergence. When using Lanczos, a value above 0 means you will be notified if the requested eigen tolerance was not achieved, if there has been a minor or severe loss of orthogonality in the computation, or if the maximum number of Lanczos iterations was reached. A value above 1 means that if any of the preceding warning conditions occur, you will be notified of the eigenvalues and predicted and actual eigen residual tolerances computed. A value above 2 means you will be notified when the computation of the eigenvector of the tridiagonal matrix is not very accurate.

WARNING_ORTHTOL This parameter determines the level of loss of orthogonality in Lanczos which is considered minor but worth reporting. If the ratio between the estimate of the eigen residual and the computed eigen residual is above this value, the minor loss of orthogonality condition is triggered. Refer to the discussion on **WARNING_EVECS**.

WARNING_MISTOL Same as **WARNING_ORTHTOL**, but this value indicates a more serious loss of orthogonality. In some cases this may indicate misconvergence, hence

the name.

WARNING_SRESTOL If the residual encountered at the end of the recurrence used to compute the eigenvector of the tridiagonal matrix in Lanczos is above this value, a corresponding warning condition is flagged. Refer to the discussion on **WARNING_EVECS**.

4.3. Other parameters for spectral methods.

MAKE_CONNECTED Spectral methods can break down if the graph is disconnected. Even if the original graph is connected, disconnected graphs can be generated in the recursion. To avoid any associated problems, we use a breadth-first-search algorithm to find connected components and add a minimal number of edges to make the graph connected. If **MAKE_CONNECTED** is nonzero, then this connectivity check will be invoked whenever a spectral option is selected. You should only change this parameter if you plan to use a spectral method and you are certain that you will only operate on connected graphs (*i.e.* if you aren't recursing).

PERTURB Spectral methods can encounter problems if the graph has symmetry since its eigenvalues can then have multiplicity greater than 1. For spectral bisection, all you can hope for is selecting some vector (which depends on the starting Lanczos vector) in the subspace of second lowest eigenvectors. However, since they work within a subspace of 2 and 3 vectors respectively, spectral quadrisection and octasection can handle two or three degrees of multiplicity respectively. Unfortunately, Lanczos can't easily identify this multiplicity. We can, however, avoid the issue by randomly perturbing the matrix. The parameter **PERTURB** controls whether or not this perturbation is invoked. Using this option helps avoid problems in some degenerate cases like the square grid graph, at the cost of a very slight increase run time. We recommend that you leave this feature activated unless you are sure you don't need it.

NPERTURB If the **PERTURB** option is being used, this parameter indicates how many random edges are added to the graph to break the symmetry.

PERTURB_MAX If the **PERTURB** option is being used, this parameter is the maximum value of an edge weight for one of the randomly added edges. A small value will perturb the eigenvectors a small amount, but if the perturbation is too small, then Lanczos might not be able to separate the eigenvectors. This value should probably be a small multiple of **EIGEN_TOLERANCE**.

COARSE_NLEVEL_RQI The parameter applies if you are using the spectral method with the **RQI/Symmlq** eigensolver option. As you work back through the intermediate graphs, the approximation to the eigenvector is refined with Rayleigh

Quotient Iteration every few levels. This parameter indicates how many levels occur between these refinements. A small value for this parameter is more robust, but a large value will reduce execution time.

MAPPING_TYPE We have implemented several methods for generating an assignment from two or three eigenvectors. This flag allows the user to switch between them. In our experience, the clear winner was a bipartite matching algorithm described in [7]. This option is invoked by a value of 3, and we encourage users to leave this parameter alone.

OPT3D_NTRIES If you are using spectral octasection, then when mapping back to a discrete solution you need to solve a constrained, global optimization problem as described in [7]. In our experience, this problem usually has a small number of local minimizers, so we solve it using local minimization techniques from random starting points. This parameter controls how many local minimizations get done, and should only be modified by sophisticated users.

4.4. Kernighan–Lin parameters.

KL_METRIC When dividing into more than 2 sets at once, our implementation of Kernighan–Lin can try to minimize any inter-set metric. Two are currently built into the code, and are controlled by this parameter. If the value of **KL_METRIC** is one, then all edges crossing between two sets are treated the same. If the value is two, then edges are weighted in a hypercube hop metric. That is, an edge between sets 0 and 1 costs one third of an edge between 0 and 7. Note that the spectral quadrisection and octasection algorithms automatically use a hypercube hop metric. If you wish to use a different metric, you can tinker with the appropriate code in “/code/main/balance.c”.

KL_RANDOM This flag turns on and off the randomness in the Kernighan–Lin routines. We recommend that you leave this parameter alone since it increases the quality and robustness of Kernighan–Lin for a tiny increase in run time.

KL_BAD_MOVES Our version of Kernighan–Lin can exit a pass early if it doesn’t seem to be making any progress. This parameter controls how quickly KL will hit this cutoff. A large value will make KL more effective, but will also increase the run time.

KL_NTRIES_BAD This parameter controls the speed at which the Kernighan–Lin code is exited. The KL routine will exit after **KL_NTRIES_BAD** passes in which no improvement is detected. Because of randomness, a pass with no improvement can be followed by one that finds a better partitioning. However, if you set **KL_RANDOM** to zero, then you should set **KL_NTRIES_BAD** to 1. A large value for this parameter will produce better results, but will cause the code to run

longer.

KL_ONLY_BNDY At one point it seemed like a good idea to only require Kernighan–Lin to consider moving vertices that were on the boundary between sets. Our implementation of this idea actually runs slower and gives worse answers than true KL, so we emphatically discourage changing this parameter.

KL_UNDO_LIST This parameter turns on an optimization that dramatically reduces the run time of Kernighan–Lin for large graphs. Instead of bucket sorting the entire set of possible vertex moves before each pass, this option preserves the moves that haven’t been changed; typically the vast majority. This leads to a dramatic increase in speed, with no perceptible change in quality. We strongly encourage you to leave this parameter alone.

4.5. Parameters for multilevel methods.

COARSEN_RATIO_MIN This value is employed if you are using either the RQI/Symmlq eigensolver, or the multilevel partitioning algorithm. It should have a value between .5 and 1.0, representing the minimal acceptable reduction in number of vertices associated with a coarsening step. If a step fails to achieve this reduction, the coarsening algorithm exits prematurely, and the resulting calculations will be performed on a larger graph than anticipated. The coarsening algorithm cannot reduce the number of vertices by more than half, so this value should always be greater than .5.

COARSE_NLEVEL_KL If you are using the multilevel partitioning algorithm, then Kernighan–Lin gets invoked periodically on successively finer graphs. This parameter indicates how many levels occur between these invocations. A small value for **COARSE_NLEVEL_KL** will lead to better partitions, while a large value will reduce execution time.

4.6. Parameters that control debugging output.

DEBUG_EVECS This parameter controls the quantity of debug output concerning calculation of eigenvectors. When set to zero, no output is generated except when an unrecoverable error condition is encountered, in which case a short message is printed before the program aborts. A value of 1 will produce a moderate amount amount of information, 2 a bit more, and so on up to a maximum value of 5.

DEBUG_KL This flag controls the output in the Kernighan–Lin routines. No debugging output is generated if the value is 0, while the improvement due to KL at each step is shown if the value is 1. Values of 2 and 3 generate mass quantities of output, and should only be invoked by an expert.

DEBUG_INERTIAL If you are using the inertial method, this flag will turn on output concerning the computation of the principle axis of the mesh.

DEBUG_CONNECTED If you are enforcing connectivity and using a spectral method, a value of 1 for this flag turns on a small amount of output in the routines that identify connected components. This will tell you if subgraphs have become disconnected in the course of a decomposition.

DEBUG_PERTURB A value of 1 for this flag turns on a small amount of output in the routines for randomly perturbing the matrix.

DEBUG_ASSIGN When using a spectral method, the mapping from the eigenvectors to an assignment can be complicated, particularly for spectral quadrisection and octasection. This parameter turns on output in the routines that compute this mapping.

DEBUG_OPTIMIZE With spectral quadrisection or spectral octasection, part of the mapping to an assignment involves a nonlinear optimization. This flag controls debugging output in the optimization subroutines.

DEBUG_BPMATCH When using spectral quadrisection or octasection, the trickiest part of the mapping from eigenvectors to a partition involves solving a maximal cost assignment problem in a bipartite graph. This flag turns on the output in the corresponding sections of the code. A value of 1 gives a moderate amount of cryptic output, while a value of 2 does more error checking and can generate a lot of output.

DEBUG_COARSEN If you invoke a multilevel method, the code will construct a sequence of increasingly coarse approximations to the original graph. This parameter controls the output for the routines performing this process.

DEBUG_MEMORY This variable turns on some consistency checks in the allocation and freeing of memory. Unless you encounter problems you think might be memory related, this value should be left at 0.

DEBUG_INPUT If this is set to 1, a message is printed confirming that the input files have been read.

4.7. Miscellaneous parameters.

RANDOM_SEED This is the seed for the random number generators “rand()” and “rand48()”.

NSQRTS If you are using either of the multilevel options, then coarse versions of the graph get created with vertex weights. We also need the square roots of these vertex weights. Since these are typically integers, instead of repeatedly calculating square roots of integers, **Chaco** computes them once and stores them in the array **SQRTS**. The value of **NSQRTS** is the length of this array, and for best performance should be somewhat larger than the number of vertices in the

original graph, divided by the number of vertices in the coarsest graph. A large value may use a small amount of unnecessary space, while a small value may lead to an unnecessary excess of computation.

5. Helpful hints.

5.1. Implementation details. **Chaco** is written entirely in ANSI standard C and is about 15,000 lines long. C performs floating point computations in double precision (8 byte) format, and **Chaco** stores the results in double precision format (except in a few cases where precision is clearly not an issue). In order to maximize the size of graphs which can be partitioned, memory is allocated dynamically when needed and released as soon as possible without seriously degrading efficiency. **Chaco** can be run in a stand-alone mode or called as a subroutine from either C programs or (with the addition of a simple wrapper) Fortran programs. The interface routine used to invoke **Chaco** as a subroutine is called “interface.c” and resides in the subdirectory “/code/main”. However, invocation as a subroutine requires a detailed understanding of some data structures and parameters, and should not be attempted without first gaining familiarity with the code.

5.2. Installation instructions. If you are using an ANSI standard compiler, then **Chaco** should compile correctly, and it should do fine on many non-standard compilers as well. **Chaco** uses several machine and compiler dependent parameters that are defined within the ANSI standard. If these values aren’t defined, then **Chaco** tries to compute them, but this is difficult to do in a machine independent way. One thing the user can do to improve robustness with a non-standard compiler is to define appropriate values for three parameters in the file “machine_params.c” in “/code/util”. These parameters are `DBL_EPSILON`, the machine precision, `DBL_MAX` the largest double precision value, and `RAND_MAX`, the largest value returned by the system random number generator “rand()”.

5.3. Some things to watch out for. Most of these points have been made earlier, but they bear repeating, if only for the sake of those readers who would rather not read the whole guide.

- Use of the Lanczos-based eigensolvers on large problems may cause the program to run out of memory on your system. This is a result of a design decision to favor speed and robustness over memory conservation in this situation. The assumption is that for very large graphs you will want to use either the RQI/Symmlq eigensolver, or the inertial or multilevel partitioning methods. See §2.2, §2.3 and §2.5.

- It is your responsibility to either choose an appropriate eigen tolerance or to live quietly with our choice on your behalf. **Chaco** tries hard to deliver the accuracy you request, but can't help much if your request is unwise. If you choose a very tight (small) tolerance, things will slow down considerably and you may run into memory trouble. If you choose a very loose (big) tolerance, your results will generally degrade and become erratic due to poor accuracy or misconvergence. See §2.2 and §4.2.
- The eigensolvers and the Kernighan-Lin heuristic make use of randomization techniques, so results generated using these methods are only strictly reproducible if the program is used in a way that generates the same sequence of random numbers. This is sometimes a very noticeable effect in the RQI/SymmIq solver, where a different random seed can result in large swings in execution time.
- The multidimensional inertial methods return sets with a gray coded mapping. This is appropriate for hypercubes, but probably not for other architectures. Although they are reasonably effective at reducing the volume of messages given their short run time, the multidimensional inertial techniques are designed to compromise on the goal of low message volume in order to produce partitions with fewer message start-ups than the other methods. See §2.3.
- The routine "func3d.c" takes a long time to compile with optimization. It's not a significant part of the execution time, so if, for some reason, you are recompiling the code often, you may wish to compile this routine without optimization.
- If you are using a compiler that is not ANSI standard, **Chaco** is probably computing a few numerical constants for you. Although we don't expect any problems to arise, *this computation is not exact*. If you are using a non-ANSI standard compiler, it may be prudent to define these constants. See §5.2 for further details.

5.4. Obtaining the code. **Chaco** is publicly available for research purposes and may be licensed for commercial application. The code is distributed along with technical documentation and sample input files via the internet. If you are interested in obtaining a copy, you should contact us at the addresses given on the cover page of this report.

Upon receipt, the "Chaco" directory will have three subdirectories, "code", "executable" and "documentation". The "documentation" directory contains postscript files of this user's guide, and three of our technical reports which are referenced in this guide. The "code" subdirectory contains all of the source code and the makefile. The makefile is set up to place the executable version of **Chaco** in the "executable" subdi-

rectory. The “executable” directory also contains several sample graph and coordinate input files.

REFERENCES

- [1] S. T. BARNARD AND H. D. SIMON, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 711–718.
- [2] T. BUI, C. HEIGHAM, C. JONES, AND T. LEIGHTON, *Improving the performance of the Kernighan–Lin and simulated annealing graph bisection algorithms*, in Proc. 26th IEEE Design Automation Conference, IEEE, 1989, pp. 775–778.
- [3] C. M. FIDUCCIA AND R. M. MATTHEYSES, *A linear time heuristic for improving network partitions*, in Proc. 19th IEEE Design Automation Conference, IEEE, 1982, pp. 175–181.
- [4] M. GAREY, D. JOHNSON, AND L. STOCKMEYER, *Some simplified NP-complete graph problems*, Theoretical Computer Science, 1 (1976), pp. 237–267.
- [5] G. GOLUB AND C. VAN LOAN, *Matrix Computations, Second Edition*, Johns Hopkins University Press, Baltimore, MD, 1989.
- [6] S. HAMMOND, *Mapping unstructured grid computations to massively parallel computers*, PhD thesis, Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, NY, 1992.
- [7] B. HENDRICKSON AND R. LELAND, *An improved spectral graph partitioning algorithm for mapping parallel computations*, Tech. Rep. SAND 92-1460, Sandia National Laboratories, Albuquerque, NM, September 1992.
- [8] ———, *An improved spectral load balancing method*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 953–961.
- [9] ———, *A multi-level algorithm for partitioning graphs*, Tech. Rep. SAND 93-1301, Sandia National Laboratories, Albuquerque, NM, June 1993.
- [10] ———, *Multidimensional spectral load balancing*, Tech. Rep. SAND 93-0074, Sandia National Laboratories, Albuquerque, NM, January 1993.
- [11] C. A. JONES, *Vertex and Edge Partitions of Graphs*, PhD thesis, Penn State, Dept. Computer Science, State College, PA, 1992.
- [12] B. KERNIGHAN AND S. LIN, *An efficient heuristic procedure for partitioning graphs*, Bell System Technical Journal, 29 (1970), pp. 291–307.
- [13] B. NOUR-OMID, A. RAEFSKY, AND G. LYZENGA, *Solving finite element equations on concurrent computers*, in Parallel computations and their impact on mechanics, A. K. Noor, ed., American Soc. Mech. Eng., New York, 1986, pp. 209–227.
- [14] C. C. PAIGE AND M. A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.
- [15] B. PARLETT AND D. SCOTT, *The Lanczos algorithm with selective orthogonalization*, Math. Comp., 33 (1979), pp. 217–238.
- [16] A. POTHEN, H. SIMON, AND K. LIOU, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal., 11 (1990), pp. 430–452.
- [17] H. D. SIMON, *Partitioning of unstructured problems for parallel processing*, in Proc. Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications, Pergamon Press, 1991.
- [18] P. SUARIS AND G. KEDEM, *An algorithm for quadrisection and its application to standard cell placement*, IEEE Trans. Circuits and Systems, 35 (1988), pp. 294–303.

EXTERNAL DISTRIBUTION:

Alpesh Amin
4401 Dayton-Xenia Rd.
Dayton, OH 45432

Steve Ashby
Lawrence Livermore Nat. Lab.
M/S L-316
PO Box 808
Livermore, CA 94551-0808

Brian Aubert
Los Alamos National Lab
PO Box 1666, MS C931
Los Alamos, NM 87545

D. M. Austin
Army High Per. Comp. Res. Cntr.
University of Minnesota
1100 S. Second St.
Minneapolis, MN 55415

Scott Baden
University of California, San Diego
Dept. of Computer Science
9500 Gilman Drive
Engineering 0114
La Jolla, CA 92091-0014

Steve Barnard
NAS Systems Division
Applied Research Branch
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035

Edward Barragy
Dept. ASE/EM
University of Texas
Austin, TX 78712

Roman J. Bednarz
Cray Research Park
655F Lone Oak Drive
Eagan, MN 55121

M. Berzins
School of Computer Studies
The University of Leeds
Leeds, LS29OT
United Kingdom

Professor N. L. Biggs
Mathematics Dept.
LSE
Houghton Street
London WC2A 2AE
United Kingdom

Rob Bisseling
Shell Research B.V.
Postbus 3003
1003 AA Amsterdam
The Netherlands

Michael Bockelie
Mail Stop 125
NASA Langley Res. Center
Hampton, VA 23665

Kenneth J. Bongort
Mail Station D12-025
Grumman Data Systems
Bethpage, NY 11714-3584

Ravi Boppana
Department of Computer Science

NYU
251 Mercer Street
New York, NY 10012

J. Browne
University of Texas
Dept. of Computer Science
Taylor Hall 5.126
Austin, TX 78712

John Brunet
Thinking Machines Corporation
245 First St.
Cambridge, MA 02142

Thang Bui
Computer Science Department
Penn State Harrisburg
Middletown, PA 17057

G. F. Carey
ASE/EM Dept., WRW 305
University of Texas
Austin, TX 78712

J. M. Cavallini
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

T. Chan
UCLA
405 Hilgard Ave.
Los Angeles, CA 90024-7009

Pak K. Chan
225 Applied Sciences
Computer Engineering
University of California
Santa Cruz, CA 95064

Ted Charrette
MIT Bldg. E3 554
42 Carleton St.
Cambridge, MA 02142

Siddhartha Chatterjee
RIACS
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035-1000

Warren Chernock
Scientific Advisor DP-1
US Department of Energy
Forestal Bldg. 4A-045
Washington, DC 20585

Tzi-cker Chiueh
Computer Science Division
U.C. Berkeley
571 Evans Hall
Berkeley, CA 94720

Doug Cline
The University of Texas System
Center for High Performance Computing
Balcones Research Center
10100 Burnett Road, CMS 1.154
Austin, Texas 78758

Tom Coleman
Dept. of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853

Neil Coleth

Supercomputing Research Center
17100 Science Dr.
Bowie, MD 20715-4300

Anil Deane
MC 934
NASA Goddard
Greenbelt, MD 20771

Professor P. Diaconis
Department of Mathematics
Harvard University
Cambridge, MA 02138

Ralf Diekmann
University of Paderborn
Dept. of Comp. Science
33095 Paderborn, Germany

Peter Dobreff
GE/KAPL
1 River Rd.
Schenectady, NY 12301

Sean Dolan
nCUBE
919 E. Hillsdale Blvd.
Foster City, CA 94404

J. J. Dongarra
Computer Science Dept.
104 Ayres Hall
University of Tennessee
Knoxville, TN 37996-1301

I. S. Duff
CSS Division
Harwell Laboratory
Oxfordshire, OX11 0RA
United Kingdom

Alan Edelman
University of California, Berkeley
Dept. of Mathematics
Berkeley, CA 94720

Steve Elbert
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

H. Elman
Computer Science Dept.
University of Maryland
College Park, MD 20842

Dr. D.R. Emerson
Theory and Computational Science
Daresbury Laboratory
Daresbury, Warrington
Cheshire, WA4 4AD
United Kingdom

Dr. Giovanni Erbacher
Dipartimento di Tecnologia dei
Centro Di Calcolo Interuniversita
dell'Italia Nord-Orientale
6/3 Via Magnanelli 40033
Casalecchio di Reno
Bologna
Italy

R. E. Ewing
Mathematics Dept.
University of Wyoming
PO Box 3036 University Station
Laramie, WY 82071

Charbel Farhat
Dept. Aerospace Engineering
UC Boulder
Boulder, CO 80309-0429

Prof. Miroslav Fiedler
Institute of Mathematics
Czech Academy of Sciences
Prague
Czech Republic

Salvatore Filippone
IBM ECSEC
Viale Oceano Pacifico 171/173
00144 Roma, Italy

J. E. Flaherty
Computer Science Dept.
Rensselaer Polytech Inst.
Troy, NY 12181

Rupert Ford
CNC Rm. 2122
Dept. Computer Science
University of Manchester
Manchester M139PL
United Kingdom

Ron Fowler
Building R1
Rutherford Appleton Laboratory
Chilton, Didcot, OX11 0QX,
United Kingdom.

G. C. Fox
Northeast Parallel Archit. Cntr.
111 College Place
Syracuse, NY 13244

Jon Frankle
Xilinx Corporation
2100 Logic Drive
San Jose, CA 95124

R. F. Freund
NRaD- Code 423
San Diego, CA 99152-5000

D. B. Gannon
Computer Science Dept.
Indiana University
Bloomington, IN 47401

Apostolos Gerasoulis
Dept. Computer Science
Rutgers University
New Brunswick, NJ 08903

Bashkar Ghosh
Department of Computer Science
Yale University
POB 2158, Yale Station
New Haven, CT 06520

Dr. Horst Gietl
nCUBE Deutschland
Hanauer Str. 85
8000 Munchen 50
Germany

John Gilbert
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Dr. M. Giles
Oxford University Computing Laboratory
8-11 Keble Rd.

Oxford, OX1-3QD
United Kingdom

G. H. Golub
Computer Science Dept.
Stanford University
Stanford, CA 94305

Anne Greenbaum
New York University
Courant Institute
251 Mercer Street
New York, NY 10012-1185

John Greenfield
EECE Department
University of New Mexico

Satya Gupta
Intel SSD
Bldg. CO6-09, Zone 8
14924 NW Greenbrier Parkway
Beaverton, OR 97006

J. Gustafson
Computer Science Dept.
236 Wilhelm Hall
Iowa State University
Ames, IA 50011

Ray Hagstrom
Hagforce HQ
823 S. Racine #D
Chicago, IL 60607-4123

Lama Hamandi
The Ohio State University
Electrical Engineering Department
205 Dreese Lab
2015 Neil Avenue
Columbus, Ohio 43210

Steve Hammond
NCAR
PO Box 3000
Boulder, CO 80307

Doug Harless
NCUBE
2221 East Lamar Blvd., Suite 360
Arlington, TX 76006

Michael Heath
Univ. of Ill., Nat. CSA
4157 Bechman Institute
405 North Matthews Ave.
Urbana, IL 61801-2300

Greg Heileman
EECE Department
University of New Mexico
Albuquerque, NM 87131

Mike Heroux
Cray Research Park
655F Lone Oak Drive
Eagan, MN 55121

A.J. Hey
University of Southampton
Dept. of Electronics and Computer Science
Mountbatten Bldg., Highfield
Southampton, SO9 5NH
United Kingdom

W. D. Hillis
Thinking Machines, Inc.
245 First St.

Cambridge, MA 02139

Prof. E. Hinton
Civil Engineering
University of Swansea
Wales
United Kingdom

Dan Hitchcock
US Department of Energy
SCS, ER-30 GTN
Washington, DC 20585

David Hodgson
School of Computer Studies
University of Leeds,
Leeds LS2 9JT
United Kingdom

Adolfy Hoisie
Cornell University
Cornell Theory Center
631 E&TC Bldg
Ithaca, NY 14853

Graham Horton
Universitat Erlangen-Numberg
IMMD III
Martensstrase 3
8520 Erlangen
Germany

Fred Howes
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Ronald R. Hoyt
Computing Devices Internatio
8800 Queen Avenue South
Minneapolis, MN 55431-1996

Yi-Fan Hu
Daresbury Laboratory
Science & Eng. Research Cou
Daresbury, Warrington, WA4
United Kingdom

T. C. Hu
Professor of Computer Science
University of California
La Jolla, CA 92093

Dr. Chua-Huang Huang
Ohio State University
Computer & Information Scie
228 Boltz Hall
2036 Neil Avenue
Columbus, OH 43210-1277

Yuan-Shin Hwang
Computer Science Department
A.V. Williams Building
University of Maryland
College Park MD, 20742

Arthur Jaffe
Dept. Mathematics
Harvard University
1 Oxford St.
Cambridge, MA 02138-2901

Peter Jimack
School of Computer Studies
University of Leeds,
Leeds LS2 9JT
United Kingdom

Zdenek Johan
Thinking Machines Corp.
245 First Street
Cambridge, MA 02142-1264

Professor C. R. Johnson
Mathematics Dept.
College of William and Mary
PO Box 8795
Williamsburg, VA 23187-8795

Gary Johnson
US Department of Energy
SCS, ER-30 GTN
Washington, DC 20585

Lennart Johnsson
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

C. Jones
Math Department
Bloomsburg University
Bloomsburg, PA 17815

Andrew Kahng
UCLA Computer Science Dept.
6291 Boelter Hall
Los Angeles, CA 90024-1596

Joe Kaitschuck
EDS
800 Tower Drive
Troy, MI 48098

Herb Keller
Applied Math 217-50
California Institute of Technology
Pasadena, CA 91125

Brian Kernighan
Bell Labs
600 Mountain Ave.
New Providence, NJ 07974

R. Keunings
Unite de Mecanique Appliquee
Universite Catholique de Louvain
B-1348 Louvain-la-Neuve
Belgium

David Keyes
Dept. of Mechanical Engineering
Yale University
PO Box 2159, Yale Station
New Haven, CT 06520-2159

David Kincaid
Center for Numerical Analysis
RLM 13.150
University of Texas
Austin, TX 78713-8510

T. A. Kitchens
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Scott Kohn
CSE 0114
UC San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114

V. Kumar
Computer Science Department
University of Minnesota

Minneapolis, MN 55455

Arne Laukholm
Senter for Informasjonsteknologi
Universitetet I Oslo
Gaustadalleen 23
Postboks 1059 Blindern, N-0316 Oslo
Norway

Julian Lebensold
CRIM
3744 Jean-Brillant Street
Suite 500
Montreal, Quebec
Canada

Charles Leete
Mathematical Sciences Section
Oak Ridge Nat. Lab.
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Dr. H.R. Leland
Calspan Corporation
PO Box 400
Buffalo, NY 14225

M. Lesoinne
Dept. Aerospace Engineering
UC Boulder
Boulder, CO 80309-0429

John Lewis
Boeing Corp.
M/S 7L-21
P.O. box 24346
Seattle, WA 98124-0346

Ray Loy
Computer Science Dept.
Amos Eaton Bldg.
RPI
Troy, NY 12180

J. G. Malone
Dept. of Mechanical Eng.
General Motors Res. Lab.
30500 Hound Rd.
Warren, MI 48090

T. A. Manteuffel
Department of Mathematics
University of Co. at Denver
Denver, CO 80202

Kapil Mathur
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

Tim Mattson
Intel Corporation
Supercomputer Systems Division
CO6-09 Bldg., Zone 8
14924 N.W. Greenbrier Parkway
Beaverton, OR 97006

Richard J. Matus
Fluent Inc.
Centerra Resource Park
10 Cavendish Court
Lebanon, NH 03766-1442

William McColl
Oxford Univ. Computing Lab
8-11 Keble Road
Oxford, OX1 3QD
United Kingdom

S. F. McCormick
Computer Mathematics Group
University of CO at Denver
1200 Larimer St.
Denver, CO 80204

Robert McLay
University of Texas at Austin
Dept. ASE-EM
Austin, TX 78712

Dr. Russell Merris
Dep. of Mathematics and Comp.
California State University
Hayward, CA 94542

Jill Mesirov
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

George Messina
Computational Fluid Dynamics
Nuclear Reactor Research Dept.
EG&G Idaho, Inc.
PO Box 1625
Idaho Falls, ID 83415-2403

P. C. Messina
158-79
Mathematics & Comp Sci. Dept.
Caltech
Pasadena, CA 91125

Gary Miller
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Bojan Mohar
Department of Mathematics
University of Ljubljana
Jadranska 19, 61111 Ljubljana
Slovenia

C. Moler
The Mathworks
24 Prime Park Way
Natick, MA 01760

Gary Montry
Southwest Software
11812 Persimmon, NE
Albuquerque, NM 87111

D. B. Nelson
US Department of Energy
OSC, ER-30, GTN
Washington, DC 20585

Kwong T. Ng
Electrical & Computer Engineerin
New Mexico State University
Box 30001
Las Cruces, NM 88003-0001

D. M. Nosenchuck
Mech. and Aero. Engr. Dept.
D302 E Quad
Princeton University
Princeton, NJ 08544

J. M. Ortega
Applied Mathematics Dept.
University of Virginia
Charlottesville, VA 22903

Can Ozturan
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

John Palmer
Thinking Machines Corp.
245 First St.
Cambridge, MA 02142

Kevin Parrot
Oxford University Computing Laboratory
8-11 Keble Road
Oxford, OX1 3QD
United Kingdom

Glauscio Paulino
Civil Engineering
Cornell University
Hollister Hall 413
Ithaca, NY 14853

Diniz, Pedro
Computer Science Department
Engineering I Bldg, Room 2106
University of California at Santa Barbara
Santa Barbara, CA 93106

Linda Petzold
L-316
Lawrence Livermore Natl. Lab.
Livermore, CA 94550

Barry Peyton
Mathematical Sciences Section
Oak Ridge National Laboratory
PO. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Paul Plassman
Math and Computer Science Division
Argonne National Lab
Argonne, IL 60439

Claude Pommerell
AT&T Bell Labs
600 Mountain Ave, Room 2C-548A
Murray Hill, NJ 07974-0636

Ravi Ponnusamy
Computer Sc. Dept
AV Williams Bldg
Univ of Maryland,
College Park, MD 20742

Alex Pothen
Computer Science Department
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

D. Powers
Dept. of Math. and Comp. Sci.
Clarkson Univ.
Potsdam, NY 13699-5815

Robert Preis
University of Paderborn
Dept. of Comp. Science
33095 Paderborn, Germany

Mike Quayle
Cadence Design Systems
2 Lowell Research Center Drive
Lowell, MA 01857

Sanjay Ranka
School of Computer and Information Science

Suite 4-116
Center for Science and Technology
Syracuse, NY 13244-4100

Satish Rao
NEC Research Institute,
4 Independence Way,
Princeton, NJ, 08540

J. Rattner
Intel Scientific Computers
15201 NW Greenbriar Pkwy.
Beaverton, OR 97006

Franz Rendl
Technische Universitat Graz
Institute fur Mathematik
Kopernikusgasse 24, A-9010 Graz
Austria

John Richardson
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

John Rollett
Oxford University Computing Laboratory
8-11 Keble Road
Oxford, OX1 3QD
United Kingdom

Diane Rover
Michigan State University
Dept. of Electrical Engineering
260 Engineering Bldg.
East Lansing, MI 48824

Jim Ruppert
NASA Ames Research Center M/S T045-1
Moffett Field, CA 94035-1000

Ralph Rye
Northern States Power Company
414 Nicollet Mall, TN4
Minneapolis, MN 55401

Y. Saad
University of Minnesota
4-192 EE/CSci Bldg.
200 Union St.
Minneapolis, MN 55455-0159

P. Sadayappan
Ohio State University
Computer & Information Science
228 Boltz Hall
2036 Neil Avenue
Columbus, OH 43210-1277

Joel Saltz
Computer Science Department
A.V. Williams Building
University of Maryland
College Park, MD 20742

A. H. Sameh
CSR
305 Talbot Laboratory
University of Illinois
104 S. Wright St.
Urbana, IL 61801

P. E. Saylor
Dept. of Comp. Science
222 Digital Computation Lab
University of Illinois
Urbana, IL 61801

Dr. H. Schellwag
Department of Technology
Box 923
S-70130 Orebro, Sweden

Martine Schlag
225 Applied Sciences
Computer Engineering
University of California
Santa Cruz, CA 95064

Rob Schreiber
RIACS
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035-1000

Elliot Schulman
nCUBE Corp.
3575 9th St.
Boulder, Co. 80304

M. H. Schultz
Department of Computer Scienc
Yale University
PO Box 2158
New Haven, CT 06520

Eric Schwabe
Department of EECS
Northwestern University
2145 Sheridan Road
Evanston, IL 60208

Mark Seager
LLNL, L-80
PO box 803
Livermore, CA 94550

Dr. J. J. Seidel
Vesaliuslaan 26
5644 HK Eindhoven
Netherlands

Horst Simon
NAS Systems Division
Applied Research Branch
NASA Ames Research Center
Mail Stop T045-1
Moffett Field, CA 94035

Richard Sincovec
Mathematical Sciences Section
Oak Ridge Nat. Lab.
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

Vineet Singh
HP Labs, Bldg. 1U, MS 14
1501 Page Mill Road
Palo Alto, CA 94304

Anthony Skjellum
Lawrence Livermore National
7000 East Ave., Mail Code L-:
Livermore, CA 94550

L. Smarr
Director, Supercomputer App
152 Supercomputer Applicatio
Bldg. 605 E. Springfield
Champaign, IL 61801

Burton Smith
Tera Computer Co
400 N. 34th St., Suite 300
Seattle, WA 98103

Barry Smith
Department of Mathematics
UCLA
Los Angeles, CA 90024-1555

Barry Smith
Department of Mathematics
UCLA
Los Angeles, CA 90024-1555

Sharon Smith
CERFACS
42 Ave. Gustave Coriolis
31057 Toulouse
France

Prof. L. Snyder
Dept. of Computer Sci. and Eng.
Mail Stop FR35
University of Washington
Seattle, WA 98195

D. C. Sorenson
Department of Math Sciences
Rice University
PO Box 1892
Houston, TX 77251

Dr. F. Stanisiewski
Institut für Mathematische Maschinen
und Datenverarbeitung
der Universität Erlangen-Nürnberg
Martensstrasse 3
91058 Erlangen
Germany

Mike Stevens
nCUBE
919 E. Hillsdale Blvd.
Foster City, CA 94404

Margaret StPierre
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142-1214

Judy Sturtevant
Mission Research Corporation
1720 Randolph Rd. SE
Albuquerque, NM 87106-4245

Wing-Kei Szeto
Department of Computer Science
The Chinese University of Hong Kong
Shatin, NT
Hong Kong

Shanghai Teng
Department of Mathematics
MIT
Cambridge, MA 02139

Kenneth Traub
Exa Corporation
One Kendall Square, Bldg. 300
Cambridge, MA 02139

Harold Trease
Los Alamos National Lab
PO Box 1666, MS F663
Los Alamos, NM 87545

Shari Trewin
Edinburgh Parallel Computing Centre
The King's Buildings
Mayfield Road
Edinburgh, EH9 3JZ
United Kingdom

Ray Tuminaro
CERFACS
42 Ave. Gustave Coriolis
31057 Toulouse Cedex
France

John A. Turner
Los Alamos National Lab
PO Box 1666, MS B226
Los Alamos, NM 87545

D. Vanderstraeten
Unite de Mecanique Appliquee
Universite Catholique de Louvain
B-1348 Louvain-la-Neuve
Belgium

Stefan VanDeWalle
Katholieke Universiteit Leuven
Dept. of Computer Science
Celestijnenlaan 200A
B-3001 Leuven, Belgium

C. VanLoan
Department of Computer Science
Cornell University, Rm. 5146
Ithaca, NY 14853

John VanRosendale
ICASE, NASA Langley Research Center
MS 132C
Hampton, VA 23665

Steve Vavasis
Dept. of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853

Sesh Venngopal
Dept. Computer Science
Rutgers University
New Brunswick, NJ 08903

Reiner Vogelsang
KFA, Zulich, ZAM
Postfach 1913
5170 Zulich
Switzerland

R. G. Voigt
MS 132-C
NASA Langley Resch Cntr, ICASE
Hampton, VA 36665

Phuong Vu
Cray Research, Inc.
19607 Franz Road
Houston, TX 77084

Steven J. Wallach
Convex Computer Corp.
3000 Waterview Parkway
PO Box 833851
Richardson, TX 75083-3851

C. Walshaw
School of Computer Studies
University of Leeds
Leeds LS2 9JT
United Kingdom

Robert Weaver
University of Colorado at Boulder
Dept. of Computer Science
Campus Box 430
Boulder, CO 80309

G. W. Weigand
DARPA/CSTO
3701 N. Fairfax Ave.
Arlington, VA 22203-1714

Paul Wesson
Oxford University Computing Laboratory
8-11 Keble Rd.
Oxford, OX1-3QD
United Kingdom

A. B. White
MS-265
Los Alamos National Lab
PO Box 1663
Los Alamos, NM 87544

Olof B. Widlund
Dept. Computer Science
Courant Institute of Math Science
New York University
251 Mercer St.
New York, NY 10012

Roy Williams
California Institute of Technology
206-49
Pasadena, CA 91104

K. G. Wilson
Physics Dept.
Ohio State University
Columbus, OH 43210

Henry Wolkowicz
Dept. Combinatorics & Optimization
University of Waterloo
Waterloo, Ontario, N2L 3G1
Canada

L. A. Wolsey
Center for Operations Res. and Economet
Universite Catholique de Louvain
B-1348 Louvain-la-Neuve
Belgium

Minoru Yao
Recruit Company Scientific Systems
1-13-1, Kachidoki, Chuo-ku, Tokyo 104
Japan

Shing-Tung Yau
Dept. Mathematics
Harvard University
1 Oxford St.
Carbridge, MA 02138-2901

David Young
Center for Numerical Analysis
RLM 13.150
The University of Texas
Austin, TX 78713-8510

Yin Zhang
Dept. Mathematics & Statistics
University of Maryland
Baltimore County Campus
Baltimore, MD 21228-5329

Jason Y. Zien
225 Applied Sciences
Computer Engineering
University of California
Santa Cruz, CA 95064

O. Zone
Unite de Mecanique Appliquee

Universite Catholique de Louvain
B-1348 Louvain-la-Neuve
Belgium

INTERNAL DISTRIBUTION:

A.R.C. Westwood	1000
Ed Barsis	1400
Sudip Dosanjh	1402
George Davidson	1403
Jim Ang	1404
Bill Camp	1421
David Gardner	1421
Grant Heffelfinger	1421
Scott Hutchinson	1421
Martin Lewitt	1421
Steve Plimpton	1421
Mark Sears	1421
John Shadid	1421
Julie Swisshelm	1421
Dick Allen	1422
Bruce Hendrickson (35)	1422
David Womble	1422
Ernie Brickell	1423
Kevin McCurley	1423
Robert Benner	1424
Carl Diegert	1424
Art Hale	1424
Rob Leland (35)	1424
Courtenay Vaughan	1424
Steve Attaway	1425
Johnny Biffle	1425
Mark Blanford	1425
Jim Schutt	1425
Mike McGlaun	1431
Allen Robinson	1431
Kent Budge	1431
John Greenfield	1431
Paul Yarrington	1432
Phil Stanton	1433
David Martinez	1434
Dona Crawford	1900
William Mason	1952
Technical Library (5)	7141
Technical Publications	7151
Document Processing for DOE/OSTI (10)	7613-2
Central Technical File	8523-2
Charles Tong	8117