# An Empirical Study of Static Load Balancing Algorithms*

Robert Leland

Department 1424
Sandia National Laboratories
Albuquerque, NM 87185

Bruce Hendrickson

Department 1422
Sandia National Laboratories
Albuquerque, NM 87185

**Abstract**

*We empirically compare a variety of current algorithms used to map scientific computations onto massively parallel computers. The comparison is performed using* **Chaco***, a publicly available graph partitioning code written by the authors. Algorithms are evaluated in terms of both computing cost and quality of partition as judged by execution time of of the parallel application.*

## 1 Introduction

Efficient use of a distributed memory parallel computer requires that a computation be balanced across processors in such a way that interprocessor communication is kept small. In most scientific applications this requirement can be phrased in terms of a graph with each vertex representing a computational task and each edge representing a data dependency. The problem is now to partition the vertices of the graph into sets of equal size in such a way that the number of edges (or more generally the cost of the edges) crossing between sets is minimized. Finding an optimal graph partition is known to be NP–complete [1], but the practical importance of this problem has motivated a variety of heuristic approaches. In this paper we compare empirically a number of the more recent and popular graph partitioning methods in the context of mapping parallel computations.

## 2 Background and methodology

The comparison was performed using **Chaco**, a publicly available graph partitioning software package developed by the authors [5]. **Chaco** allows for recursive application of any of several methods for finding small edge separators in edge and vertex weighted graphs. These methods include inertial [10], spectral [4, 6, 7, 12], Kernighan–Lin [8, 9], and multilevel [8] methods in addition to several simpler strategies. Each of these can be used to partition the graph into two, four or eight pieces at each level of recursion, but for simplicity we have limited this study to recursive bisection schemes. (The quadrisection and octasection shemes are discussed in [4, 5, 6, 7].) By recursive application of these methods, a weighted graph can be partitioned into any number of closely balanced sets. In addition, the Kernighan–Lin method can be applied to improve partitions generated by any of the other methods (which will be indicated by appending "KL" to the name of the initial partitioner), and can be used to minimize a wide variety of objective functions.

Tests were run on a variety of sample graphs generated from actual finite element and finite difference computations performed on parallel machines at Sandia and elsewhere. Partitions were evaluated in two ways. We tabulated four basic graph metrics characterizing the partition, and we recorded the execution time of some actual parallel codes using the various partitions.

The graph metrics used were the cut edge or *cuts* metric consisting of the total number of edges crossing between sets, the hypercube hop or *hops* metric in which each cross edge is weighted by the number of links it must traverse in a hypercube architecture, the *boundary vertex* metric which counts the number of vertices in each set which have a neighbor in another set, and the *adjacent set* metric which records the total number of sets connected to each set.

The relative importance of these graph metrics in predicting communication costs is a function of problem choice and a variety of machine properties. Cuts is the standard graph metric currently used to judge partition quality since it is proportional to total communication volume. Recent empirical work [2, 3] indicates that the hops metric better accounts for message congestion and hence is preferable, particularly for hypercube architectures. Boundary vertices are an important measure of partition quality in some applications like matrix–vector multiplication. The adjacent sets metric predicts the number of message startups required of each processor, which can be very important when an application generates a large number of relatively small messages.

The applications codes used were a stand–alone parallel conjugate gradient solver written for this purpose and a parallel finite element code developed at Sandia by Shadid *et al.* [11]. These are representative of the sort of unstructured communication intensive scientific codes where data partitioning matters.

Because machine efficiency can vary substantially with problem size, we ran sample problems from two regimes: large problems which nearly fill the available memory and are dominated by computation, and

*Appeared in Proc. Scalable High–Perf. Comput. Conf. IEEE, 1994, pp. 682–685.

medium-sized problems which do not fill the memory and are dominated by communication. These correspond respectively to the "scaled speed-up" and "fixed speed-up" paradigms commonly used in parallel computing.

# 3 Results

## 3.1 Local refinement

We first demonstrated the significant improvement in graph metrics obtained by appending a local refinement scheme to a good global partitioner. We have found a generalized version of the Kernighan-Lin algorithm to be the most effective of the several schemes we have investigated in this regard. One set of sample runs demonstrating this is summarized in Table 1, where we tested various methods on *Barth5*, a two-dimensional fluid dynamics mesh with $|V| = 15,606$ and $|E| = 45,878$ obtained from NASA Ames. The table shows the reduction in cut size, but the other graph metrics are each improved as well.

| | *Cut edges* | | | | |
|---|---|---|---|---|---|
| | *Inertial* | | *Spectral* | | *Multi-* |
| | Alone | + KL | Alone | + KL | level |
| 2 sets | 245 | 200 | 200 | 139 | 175 |
| 4 sets | 897 | 520 | 521 | 367 | 379 |
| 8 sets | 1441 | 917 | 888 | 693 | 662 |
| 16 sets | 2266 | 1383 | 1382 | 1148 | 1106 |
| 32 sets | 3141 | 2057 | 2075 | 1824 | 1824 |
| 64 sets | 4253 | 3128 | 3170 | 2927 | 2943 |
| Seconds | 2.0 | 13.8 | 136.7 | 146.0 | 28.4 |

**Table 1.** Performance of partitioning algorithms on *Barth5*, a 2D CFD mesh.

The success of the Kernighan-Lin algorithm at reducing the cut size of partitions led to the development of a multilevel algorithm[8]. In this method, a sequence of successively smaller approximations to a graph are constructed, the smallest graph in the sequence is partitioned, and the partition is projected back through the sequence with the periodic application of Kernighan-Lin. The data in Table 1 confirm the effectiveness of this approach. On this problem the multilevel method found partitions with essentially as low cut sizes as the other best method (spectral with KL) in much less time.

## 3.2 Large problems

Next, we verified the validity of the hops metric as a measure of the communication requirements of a large, memory filling application. To do this, we ran several mapping strategies [1] on an unstructured finite element problem and compared the metric against the application's actual communication time on 64 processors of an nCUBE 2 multiprocessor. The sample grid for the data presented here was of a 3D chemical vapor deposition reactor and had 186K finite elements. The sample calculation consisted of a single time step in the solution of a complex set of partial differential equations solved in a chemically reacting flow code developed by Shadid *et al.* [11]. (**Chaco** has been embedded in this code to allow a flexible choice of partitioning strategies: when a small computation is to be run just once, a simple and cheap partitioning strategy like the inertial method may be employed; when a large, performance critical computation is to be run, a more expensive partitioner like the multilevel method may be preferred.)

The results for this problem, presented in Table 2, demonstrate a close correlation between the hops metric and the application communication time and solver efficiency. We have included as a reference point results for the *linear* partitioning method in which the the $n$ graph vertices are mapped to the $p$ processors according to the numbering of the graph, *i.e.* the first $n/p$ vertices are assigned to processor 0, the next $n/p$ to processor 1, etc. This simple method, which mimics what might typically happen if the partitioning problem were ignored, actually tends to produce quite good mappings in comparison with a random algorithm because there is often considerable locality implied by the mesh numbering scheme.

| *Method* | *Hops* ($\times 10^6$) | *Part. time (s)* | *Comm. time (s)* | *Solve effic.* |
|---|---|---|---|---|
| Linear | 3.2 | 0.0 | 24 | .69 |
| Inertial | .62 | 7.0 | 7.7 | .87 |
| InertialKL | .47 | 85 | 4.7 | .90 |
| Multilevel | .27 | 370 | 2.2 | .96 |

**Table 2.** Scaled parallel performance for 3D horizontal chemical vapor deposition reactor mesh (186k elements).

We conclude based on this and other similar test cases that the multilevel method produces the best partitions. It is also the most flexible in that it can be readily adapted to implement new partitioning objectives and tuned to produce higher or lower quality partitions. While the inertial and inertialKL methods generate mappings which induce several times as much communication, large, memory filling problems tend to be dominated by computation, so the discrepancy in over-all efficiency is not great. Given that and the very competitive run times of inertial and inertialKL, these methods may be preferred if the resulting partition will not be sufficiently used to amortize the higher cost of the multilevel method.

## 3.3 Medium problems

Finally we examined some medium-sized problems which use only a fraction of the memory on each node. *Barth5* when mapped to hundreds of proces-

[1] A spectral decomposition of this large graph proved prohibitively expensive and so was omitted.

sors is one such problem; a second is *Mpart*, a three-dimensional finite element mesh with $|V| = 6,673$ and $|E| = 55,664$ generated at Sandia. In Figures 1 and 2 we show the performance in Megaflops per second that different partitionings obtained in a conjugate gradient linear solver applied to these meshes. We used a range of hypercube sizes on the nCUBE 2. Note that in this regime none of the more sophisticated partitioning schemes produces clearly superior mappings on these sample problems, leading us to consider ease of use and cost of partitioning in choosing a scheme. The spectral and multilevel algorithms have the advantage that they require only connectivity information (encoded in the graph) and not spatial coordinate information as the inertialKL scheme does. However, the inertialKL scheme produces partitions of nearly the same quality and is much less expensive.

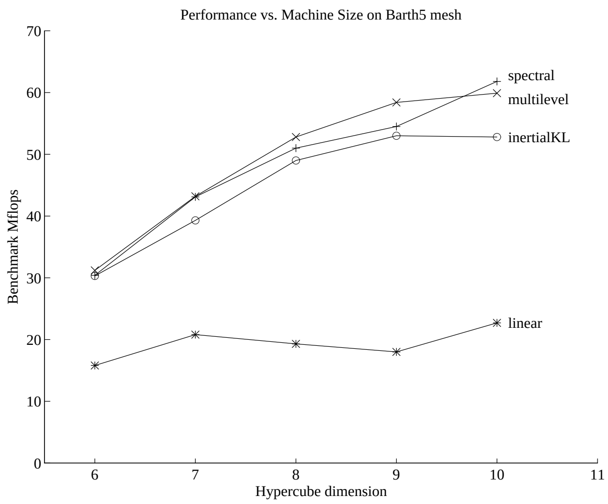


**Figure 1.** Performance of various partitionings on CG benchmark using *Barth5* mesh.

In an effort to quantify this trade-off, we combined the data in Figures 1 and 2 with the execution times of the partitioning methods. For each graph we considered the various methods and normalized both the time to partition to a particular number of sets and the observed execution speed on that number of processors. We present this data as a cost/quality chart in fig 3. The multilevel and inertialKL methods emerge as high quality and low cost partitioning strategies and hence are to be preferred over the other methods tested. We note that the multilevel method can be conveniently tuned to produce even higher quality partitions at moderately greater cost [8, 5], which lends further weight to the conclusion that if we we desire very high quality partitions at moderate cost, we should use the multilevel method. If we desire very fast partitions of high quality, we should employ the

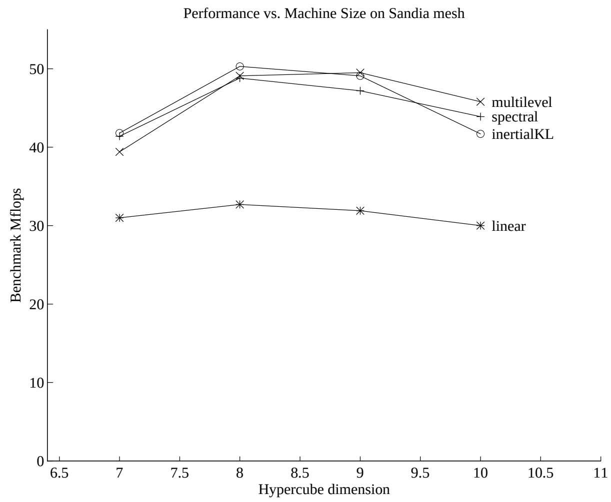


**Figure 2.** Performance of various partitionings on CG benchmark using *Mpart*, a 3D mesh finite element mesh.

inertialKL method. Finally if we do not have coordinate information, the multilevel method should be the first choice.

## 4 Conclusions

Combining the previous observations, we make the following recommendations. If coordinate information is available and the application run time is not critical, then the inertialKL method (or if partitioning time is very critical, the inertial method) should be used. If coordinate information is not available, or if the quality of the partition is critical, then the multilevel method is preferable.

## References

[1] M. Garey, D. Johnson, and L. Stockmeyer, *Some simplified NP-complete graph problems*, Theoretical Computer Science, 1 (1976), pp. 237–267.

[2] S. Hammond. Personal Communication, November 1992.

[3] ———, *Mapping unstructured grid computations to massively parallel computers*, PhD thesis, Rensselaer Polytechnic Institute, Dept. of Computer Science, Troy, NY, 1992.

[4] 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.

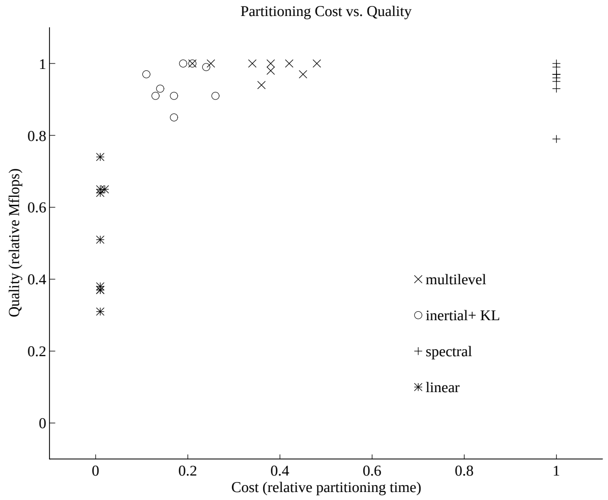


**Figure 3.** Quantitative ranking of partitioning methods on *Barth5* and *Mpart* meshes.


[5] ——, *The Chaco user's guide, version 1.0*, Tech. Rep. SAND 93-2339, Sandia National Laboratories, Albuquerque, NM, October 1993.

[6] ——, *An improved spectral load balancing method*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 953–961.

[7] ——, *Multidimensional spectral load balancing*, Tech. Rep. SAND 93-0074, Sandia National Laboratories, Albuquerque, NM, January 1993.

[8] ——, *A multilevel algorithm for partitioning graphs*, Tech. Rep. SAND 93-1301, Sandia National Laboratories, Albuquerque, NM, October 1993.

[9] B. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Technical Journal, 29 (1970), pp. 291–307.

[10] 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.

[11] J. Shadid, S. Hutchinson, and H. Moffat, *A parallel unstructured finite element implementation for MIMD machines*, tech. rep., Sandia National Laboratories, Albuquerque, NM, In preparation 1993.

[12] H. D. Simon, *Partitioning of unstructured problems for parallel processing*, in Proc. Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications, Pergammon Press, 1991.