# Architecture Aware Partitioning Algorithms*

Irene Moulitsas[1,2] and George Karypis[1]

[1] University of Minnesota, Department of Computer Science and Engineering
and Digital Technology Center and Army HPC Research Center
Minneapolis, MN 55455
[2] The Cyprus Institute, P.O. Box 27456, 1645 Nicosia, Cyprus
{moulitsa,karypis}@cs.umn.edu

**Abstract.** Existing partitioning algorithms provide limited support for load balancing simulations that are performed on heterogeneous parallel computing platforms. On such architectures, effective load balancing can only be achieved if the graph is distributed so that it properly takes into account the available resources (CPU speed, network bandwidth). With heterogeneous technologies becoming more popular, the need for suitable graph partitioning algorithms is critical. We developed such algorithms that can address the partitioning requirements of scientific computations, and can correctly model the architectural characteristics of emerging hardware platforms.

## 1 Introduction

Graph partitioning is a vital pre-processing step for many large-scale applications that are solved on parallel computing platforms. Over the years the graph partitioning problem has received a lot of attention [3, 12, 5, 1, 2, 7, 10, 11, 14, 15, 18, 19, 23, 20]. Despite the success of the existing algorithms, recent advances in science and technology demand that new issues be addressed in order for the partitioning algorithms to be effective.

The Grid infrastructure [9, 4] seems to be a promising viable solution for satisfying the ever increasing need for computational power at an affordable cost. Metacomputing environments combine hosts from multiple administrative domains via transnational and world-wide networks into a single computational resource. Even though message passing is supported, with some implementation of MPI [8], there is no support for computational data partitioning and load

* This work was supported in part by NSF EIA-9986042, ACI-0133464, ACI-0312828, and IIS-0431135; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

A. Bourgeois and S.Q. Zheng (Eds.): ICA3PP 2008, LNCS 5022, pp. 42–53, 2008.

balancing. Even on a smaller scale, clusters of PCs have become a popular alternative for running distributed applications. The cost effectiveness of adding new, and more powerful nodes to an existing cluster, and therefore increasing the cluster potential, is an appealing solution to a lot of institutions and researchers. We can clearly see that upcoming technologies have introduced a totally new class of architectural systems that are very heterogeneous in terms of computational power and network connectivity.

Most of the graph partitioning algorithms mentioned above compute a data partitioning that is suitable for homogeneous environments only. Recently there has been some work on partitioning for heterogeneous architectures, namely PaGrid [16, 24], JOSTLE [22], MiniMax [21], and DRUM [6].

In the context of the widely used METIS [17] library, we have developed graph partitioning algorithms for partitioning meshes/graphs onto heterogeneous architectures. Our algorithms allow full heterogeneity in both computational and communication resources. We use a more accurate model to describe the communication cost instead of the notion of edgecut used in the algorithms mentioned above. We also do not solve the expensive and unscalable quadratic assignment problem, and we do not enforce a dense processor-to-processor communication.

In the remainder, Section 2 discusses the modeling of the computational graph and the heterogeneous architecture system. In Section 3 we present the problem formulation. In Section 4 we describe our proposed algorithms. Section 5 presents a set of experimental results. Finally Section 6 provides some concluding remarks.

## 2 Problem Modeling

The graph partitioning problem can be defined as follows: Given a graph $G = (V, E)$, where $V$ is the set of vertices, $n = |V|$ is the number of vertices, and $E$ is the set of edges in the graph, partition the vertices to $p$ sets $V_1, ..., V_p$ such that $V_i \bigcap V_j = \emptyset$ for $i \neq j$ and $\bigcup V_i = V$, for $i, j = 1, ..., p$. This is called a $p-$way partitioning and is denoted by $P$. Every one of the subsets $V_i$ is called a partition or subdomain. $P$ is represented by a partition vector of length $n$, such that for every vertex $v \in V$, $P[v]$ is an integer between 1 and $p$, indicating the partition which $v$ is assigned to.

### 2.1 Computational Graph Modeling

The graph $G$ is a weighted graph if every vertex $v$ is associated to either or both weights $w(v)$ and $c(v)$. If no specific weights are provided, we can assume that all vertices, have uniform weights. The first vertex weight $w$ is assigned depending on the amount of computations performed by a vertex. The second weight $c$ reflects the amount of data that needs to be sent between processors i.e., communication.

The majority of multilevel graph partitioning formulations have primarily focused on edgecut based models and have tried to optimize edgecut related objectives. In the edgecut model all the edges split between different partitions account as multiple communication messages. The edgecut metric is only an

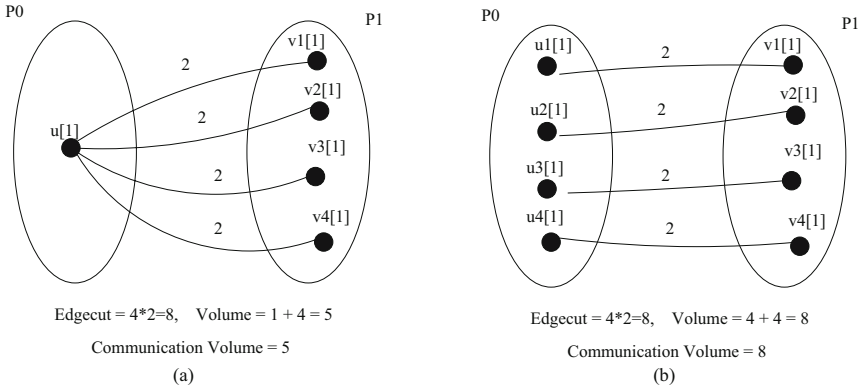


**Fig. 1.** Comparison between the edgecut and volume models

approximation of the total communication cost [13]. The actual communication may be lower and depends on the number of boundary vertices. For this reason we will focus on the **Volume** of a partitioning which we define as the total communication volume required by the partition. This measure is harder to optimize [13] than the edgecut.

Look at the two different scenarios presented in Figure 1(a) and (b). Let's assume vertex $u$ and vertices $u_1, u_2, u_3, u_4$ are assigned to partition $P_0$, while vertices $v_1, v_2, v_3, v_4$ are assigned to partition $P_1$. We have noted the communication weights of every vertex in square brackets in the figure (i.e., $c(u_i) = 1$ and $c(v_i) = 1$ for all $i$). If the edgecut model was used, each one of the cut edges would have an edge weight of 2, as each one of the incident vertices to the edge has communication size of 1. Both of the partitionings presented in Figure 1, would incur an edgecut of 8. However, in Figure 1(a) the actual communication volume is only 5, as processor $P_0$ will send a message of size 1 to $P_1$, and $P_1$ will send four messages of size 1 to $P_0$. Only if the volume model is used, will we have an accurate estimate of the actual communication for both cases.

### 2.2 Architecture Graph Modeling

Partitioning for a heterogeneous environment requires modeling the underlying architecture. For our model we use a weighted undirected graph $A = (P, L)$, that we call the *Architecture Graph*. $P$ is the set of graph vertices, and they correspond to the processors in the system, $P = \{p_1, \ldots, p_p\}$, $p = |P|$. The weights $w^*(\cdot)$ associated with the architecture graph vertices represent the processing cost per unit of computation. $L$ is the set of edges in the graph, and they represent communication links between processors. Each communication link $l(p_i, p_j)$ is associated with a graph edge weight $e^*(p_i, p_j)$ that represents the communication cost per unit of communication between processors $p_i$ and $p_j$.

If two processors are not "directly" connected, and the communication cost incurred between them is needed, we sum the squares of the weights of the

shortest path between them. This is called a *quadratic path length* (QPL). In [22] it is shown that a *linear path length* (LPL) does not perform as well as the QPL. The insight is that LPL does not sufficiently penalize for cut edges across links that suffer from slower communication capabilities.

For our model we assume that communication in either direction across a given link is the same, therefore $e^*(p_i, p_j) = e^*(p_j, p_i)$, for $i, j = 1, \ldots, p$. We also assume that $e^*(p_i, p_i) = 0$, as the cost for any given processor to retrieve information from itself is incorporated in its computational cost $w^*(p_i)$.

Although the existing heterogeneous partitioning algorithms assume a complete weighted architecture graph, we find that this approach is not scalable and therefore avoid it. We provide more details in Section 4.

## 3 Metrics Definition

Given the proposed models for the computational graph and the architecture graph, we now define several metrics that will be used in our partitioning algorithms.

**Computational Cost.** This first metric is the cost a processor $p_i$ will incur to perform computations, over all its assigned portion of vertices $V_i$:

$$CompCost_{p_i}^{V_i} = w^*(p_i) \times \sum_{v \in V_i} w(v)$$

The computational cost reflects the time needed by a certain processor to process the vertices assigned to it.

**Communication Cost.** This metric is the cost a processor $p_i$ will incur for communicating, sending and receiving, any necessary information.

Each partition can distinguish between three types of vertices: (i) interior (local) vertices, those being adjacent only with local vertices, (ii) local interface vertices, those being adjacent both with local and non–local vertices, and (iii) external interface nodes, those vertices that belong to other partitions but are coupled with vertices that are assigned to the local partition. In the context of a parallel application, communication is performed only due to the internal and external interface vertices. Specifically, vertices that belong to category 2 will need to be sent to the corresponding neighboring processors, and vertices belonging to category 3 will need to be received from their hosting processor/partition.

The cost that a processor $p_i$ will incur for communicating any information associated to its assigned portion of the vertices $V_i$ of the computational graph:

$$CommCost_{p_i}^{V_i} = \sum_{v \in V_i} \left( \sum_{P(w), w \in adj(v)} e^*(p_i, P(w)) \times c(v) \right) + \sum_{v \in V_i} \left( \sum_{w \in adj(v)} e^*(p_i, P(w)) \times c(w) \right)$$

where $adj(v)$ indicates the vertices adjacent to vertex $v$, and $P(w)$ is the processor/partition a vertex $w$ is assigned to. In the above equation, please note that no communication links are double counted.

**Processor Elapse Time.** For every processor $p_i$, its elapse time (**ElTime**) is the time it spends on computations plus the time it spends on communications. Therefore, using the above definitions, the elapse time of processor $p_i$ is:

$$ElapseTime_{p_i}^{V_i} = CompCost_{p_i}^{V_i} + CommCost_{p_i}^{V_i}$$

**Processor Overall Elapse Time.** By summing up the elapse times of all individual processors, we have an estimate of the overall time (**SumElTime**) that all processors will be occupied:

$$TotalElapseTime = \sum_{p_i \in P} ElapseTime_{p_i}^{V_i}$$

**Application Elapse Time.** The actual run time of the parallel application (**MaxElTime**) will be determined by that processor that needs the most time to complete. Therefore, no matter how good the quality of a partitioning is, its overall performance is driven by its "worst" partition:

$$ElapseTime = \max_{p_i \in P} \{ElapseTime_{p_i}^{V_i}\}$$

## 4 Framework for Architecture-Aware Partitioning

One of the key ideas of our architecture-aware partitioning algorithms is that they follow the two-phase approach. The purpose of the first phase is to focus entirely on the computational and memory resources of each processor and compute a problem decomposition that balances the demands on these resources across the different processors. The purpose of the second phase is to take into account the interconnection network characteristics (and its potential heterogeneity) and modify this partitioning accordingly so that it further optimizes the final problem decomposition. We will refer to this as the *predictor-corrector* approach, since the purpose of the second phase can be thought of as *correcting* the decomposition computed by the first phase. The motivation behind this approach is that it allows us to leverage existing high-quality partitioning algorithms for achieving the first phase, which even though in the context of network heterogeneity they tend to produce suboptimal partitionings, these partitionings are not arbitrarily poor. As a result, these partitionings can be used as building blocks for constructing good architecture-aware partitionings.

In all of our algorithms, the partitioning for the first phase is computed using the *kvmetis* algorithm from the MéTiS [17] library. This algorithm computes a $p$-way partitioning that takes into account the resource capabilities of each

processor and minimizes the total communication volume. The partitioning for the second phase is computed by utilizing a randomized greedy refinement algorithm (similar to those used in MℓIS's $p$-way partitioning algorithms) that moves vertices between partitions as long as such moves optimize the quality of the resulting decomposition.

We used two different approaches to assess the quality of the architecture-aware partitioning. The first is based on the maximum communication volume and the second is based on the application elapsed time. This leads to two different objectives functions that drive the refinement routines of the second phase. The first objective function tries to minimize the maximum communication volume while keeping the computational load proportional to the computational power of each processor. The second objective function couples the communication and computational requirements and tries to directly minimize the application elapsed time (i.e., the maximum elapsed time across the $p$ processors). Note that both of these formulations attempt to compute decompositions that will be balanced. However, they use a different notion of "balance". The first treats computation and communication as two different phases and attempts to balance them individually, whereas the second one treats them in a unified way and attempts to balance them in a coupled fashion.

Our discussion so far assumed that each processor has full information about the communication cost associated with sending data between each pair of processors (i.e., $e^*(p_i, p_j)$). This is required in order to properly compute either the maximum communication volume or the application elapsed time. If the number of processors is small, this is not a major drawback, as the cost associated with determining and storing these values is rather small. However, for large number of processors, such an approach creates a number of problems. First, if we need to have accurate estimates of these costs, these values need to be determined during the execution of the partitioning algorithm (e.g., by using a program to send messages between all pairs of processors to explicitly measure them). This will increase the time required by the partitioning algorithm and impose a quadratic memory complexity, which in some cases can be the determining factor of the scalability of these algorithms. Second, if we rely on a network topology model to infer some of these communication costs, then we introduce a level of approximation in our models, which their inherent errors may nullify any improvements that can potentially be achieved by architecture-aware partitionings.

To overcome this problem, we augmented the maximum volume- and application elapsed time-based formulations to operate on a *sparse* representation of the architecture graph. The idea behind these formulations is to constraint the refinement algorithms of the second phase so that not to create decompositions that require communication between any additional pairs of processors beyond those required by the first phase decomposition. By imposing this addition constraint, then our two-phase algorithm needs to only estimate the communication costs associated with the pairs of communicating processors of the first phase, and use those to accurate evaluate the maximum communication volume and application elapsed time objectives. Since the first-phase decomposition was obtained using

state-of-the-art graph partitioning algorithms, the pairs of processors that need to communicate is rather small and independent of the number of processors in the system. On the average, each subdomain will need to communication with a constant number of other subdomains. This greatly reduces the memory and time complexity associated with constructing the architectural graph, and leads to scalable architecture-aware partitioning algorithms.

In summary, using the above predictor-corrector framework, we developed four different architecture-aware partitioning algorithms that differ on the objective function that they use (maximum communication volume or application elapsed time) and whether or not they use a dense or a sparse architectural graph. We will refer to these algorithms using the names *VolNS* (maximum volume, non-sparse), *VolS* (maximum volume, sparse), *ElTNS* (elapsed time, non-sparse), and *ElTS* (elapsed time, sparse).

## 5 Experimental Results

We evaluated the performance of our algorithms using a wide variety of graphs and architecture topologies. The characteristics of the computation graphs are presented in Table 1. The size of these graphs ranged from 14K to 1.1M vertices.

The architecture graphs we used are presented in Figure 2. Figure 2(a) presents a one dimensional array. Figure 2(b) is a two dimensional array. Figure 2(c) presents an 8–node, 32–processor cluster. Each node has four tightly connected processors, and a fast interconnection network among its 4 processors. Communication between different nodes is slower Finally, Figure 2(d) shows a typical grid architecture. The top and bottom part may each be physically located in the same geographical location and each is a metacomputer. The intra-communication across the two parts is slower than the inter-communication locally for each one.

### 5.1 Quality of the Results

We compared the characteristics of the partitionings produced by the four algorithms described in Section 4 by looking at four performance metrics: maximum elapsed time (i.e., application elapsed time), sum of elapsed time over all processors, (total) edgecut, and total communication volume. Due to space constraints,

**Table 1.** Characteristics of the test data sets

| | Name | # Vertices | # Edges | Description |
|---|---|---|---|---|
| 1 | 144 | 144,649 | 1,074,393 | Graph corresponding to a 3D FEM mesh of a parafoil |
| 2 | auto | 448,695 | 3,314,611 | Graph corresponding to a 3D FEM mesh of GM's Saturn |
| 3 | brack2 | 62,631 | 366,559 | Graph corresponding to a 3D FEM mesh of a bracket |
| 4 | cylinder93 | 45,594 | 1,786,725 | Graph of a 3D stiffness matrix |
| 5 | f16 | 1,124,648 | 7,625,318 | Graph corresponding to a 3D FEM mesh of an F16 wing |
| 6 | f22 | 428,748 | 3,055,361 | Graph corresponding to a 3D FEM mesh of an F22 wing |
| 7 | finan512 | 74,752 | 261,120 | Graph of a stochastic programming matrix for financial portofolio optimization |
| 8 | inpro1 | 46,949 | 1,117,809 | Graph corresponding to a 3D stiffness matrix |
| 9 | m6n | 94,493 | 666,569 | Graph corresponding to a 3D FEM mesh of an M6 wing |

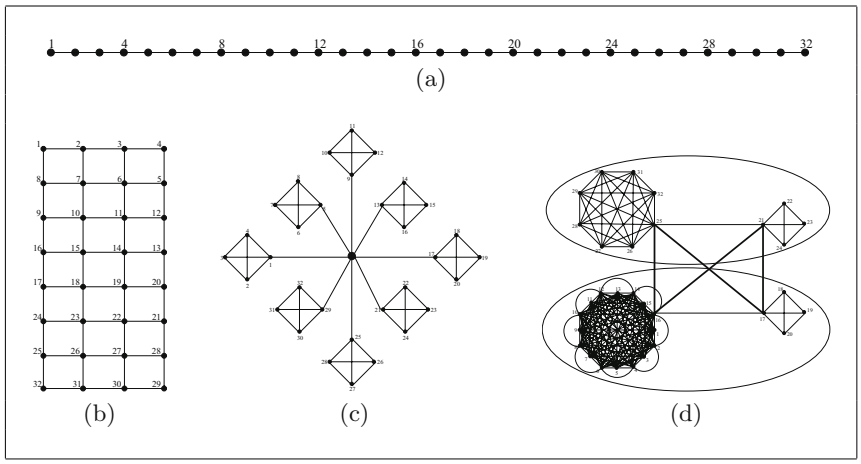


**Fig. 2.** (a) Arch32_1: 1D Array Processor Graph, (b) Arch32_2: 2D Array Processor Graph, (c) Arch32_3: Cluster of 8 compute nodes, (d) Arch32_4: Metacomputer.

for each one of the architectures and algorithms, we report the average of these metrics over the nine graphs. These results are summarized in Figure 3.

From these results we can see that all four proposed algorithms lead to decompositions that have a lower application elapsed time than those computed by `kvmetis` (Figure 3(a)). This is true for both the maximum volume- and the application elapsed time-based formulations. These results show that non trivial reductions (10%–25%) in the applications elapsed time can be obtained by explicitly modeling and optimizing the communication characteristics of the architecture and problem. Comparing the two different objective functions, we see that the one that explicitly minimizes the application elapsed time leads to consistently better results than the one that just tries to minimize the maximum volume. This is not surprising, as the former is capable of better trading communication and computational costs towards the goal of reducing the maximum elapsed time. The results comparing the sum of the elapsed times over all processors (Figure 3(b)) provide some additional insights on the type of solutions produced by the two objective functions. In general, the volume-based objective function achieves lower values than those achieved by its elapsed time-based counterpart. This suggests that the dramatic improvements at the application elapsed time (i.e., maximum elapsed time) come at the expense of uniformly increasing the amount of time spent by all the processors.

Comparing the edgecut and volume of the resulting partitions (Figures 3(c) and(d)), we see that in general, the architecture-aware algorithms produced decompositions that have higher edgecuts than those produced by `kvmetis`, but lower communication volumes. This is not surprising, as the refinement algorithms used in the corrector phase, entirely ignore the edgecut and focus either on the maximum volume or the application elapsed time. These two objective functions better correlate with the total volume and as the results suggest, in

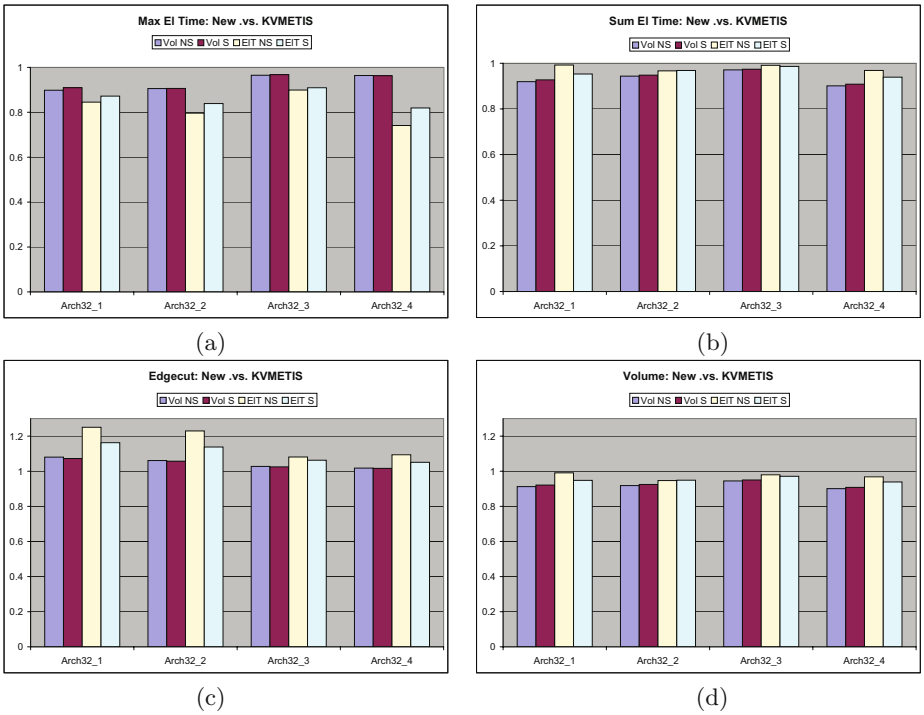


**Fig. 3.** Characteristics of the induced 32-way partitioning for Arch32_1, Arch32_2, Arch32_3, and Arch32_4

some cases it is at odds with minimizing the edgecut. This is also an indirect verification of the argument that eventhough the edgecut gives an indication of the communication volume, it is by no means an accurate measure of it. Indeed, by looking at Figure 3(c) we would have been misled as to say that our algorithms would have higher communication needs, which is not true as shown in Figure 3(d).

### 5.2 Comparison between Sparse and Non-sparse Algorithms

As discussed in Section 4, one of the main contributions of this work is that it also proposes sparse algorithms that are more scalable compared to the non-sparse refinement ones. Of course there lies a question regarding how much we have sacrificed in quality in order to achieve this scalability.

In Figure 4 we compare the sparse volume refinement algorithm with its non-sparse counterpart, and the sparse elapse time refinement algorithm with the non-sparse one. We have taken the ratio of the application elapse times of the sparse algorithms, over the application elapse times of the non-sparse ones. We have a total of 72 comparisons. We can see that in only 5 of the cases, did the

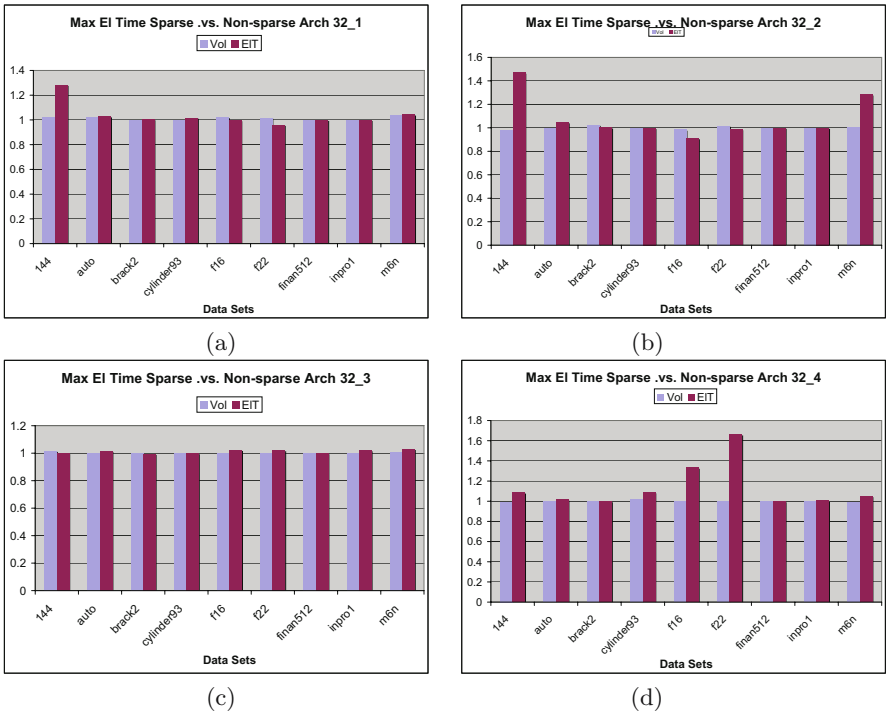


**Fig. 4.** Comparison of the sparse and non sparse approaches

sparse scalable algorithms produce worse results. In the remaining 67 cases, the qualities were comparable, and therefore we did not see any degradation.

## 6 Conclusions

The field of heterogeneous graph partitioning is a very new field and there is a lot of room for improvement. However the approaches described above represent a scalable solution that merits further investigation and development. We were able to produce partitions of high quality that can correctly model architecture characteristics and address the requirements of upcoming technologies.

## References

1. Barnard, S.T.: Pmrsb: Parallel multilevel recursive spectral bisection. In: Supercomputing 1995 (1995)
2. Barnard, S.T., Simon, H.: A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes. In: Proceedings of the seventh SIAM conference on Parallel Processing for Scientific Computing, pp. 627–632 (1995)

3. Bui, T., Jones, C.: A heuristic for reducing fill in sparse matrix factorization. In: 6th SIAM Conf. Parallel Processing for Scientific Computing, pp. 445–452 (1993)
4. Chapin, S.J., Katramatos, D., Karpovich, J., Grimshaw, A.S.: The Legion resource management system. In: Feitelson, D.G., Rudolph, L. (eds.) Job Scheduling Strategies for Parallel Processing, pp. 162–178. Springer, Heidelberg (1999)
5. Diniz, P., Plimpton, S., Hendrickson, B., Leland, R.: Parallel algorithms for dynamically partitioning unstructured grids. In: Proceedings of the seventh SIAM conference on Parallel Processing for Scientific Computing, pp. 615–620 (1995)
6. Faik, J., Gervasio, L.G., Flaherty, J.E., Chang, J., Teresco, J.D., Boman, E.G., Devine, K.D.: A model for resource-aware load balancing on heterogeneous clusters. Technical Report CS-03-03, Williams College Department of Computer Science (2003), Submitted to HCW, IPDPS 2004
7. Fiduccia, C.M., Mattheyses, R.M.: A linear time heuristic for improving network partitions. In: Proc. 19th IEEE Design Automation Conference, pp. 175–181 (1982)
8. Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230 (1994)
9. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. The International Journal of Supercomputer Applications and High Performance Computing 11(2), 115–128 (1997)
10. Gilbert, J.R., Miller, G.L., Teng, S.-H.: Geometric mesh partitioning: Implementation and experiments. In: Proceedings of International Parallel Processing Symposium (1995)
11. Goehring, T., Saad, Y.: Heuristic algorithms for automatic graph partitioning. Technical report, Department of Computer Science, University of Minnesota, Minneapolis (1994)
12. Heath, M.T., Raghavan, P.: A Cartesian parallel nested dissection algorithm. SIAM Journal of Matrix Analysis and Applications 16(1), 235–253 (1995)
13. Hendrickson, B.: Graph partitioning and parallel solvers: Has the emperor no clothes (extended abstract). In: Workshop on Parallel Algorithms for Irregularly Structured Problems, pp. 218–225 (1998)
14. Hendrickson, B., Leland, R.: An improved spectral graph partitioning algorithm for mapping parallel computations. Technical Report SAND92-1460, Sandia National Laboratories (1992)
15. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories (1993)
16. Huang, S., Aubanel, E.E., Bhavsar, V.C.: Mesh partitioners for computational grids: A comparison. In: Kumar, V., Gavrilova, M.L., Tan, C.J.K., L'Ecuyer, P. (eds.) ICCSA 2003. LNCS, vol. 2669, pp. 60–68. Springer, Heidelberg (2003)
17. Karypis, G., Kumar, V.: METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota (1998), `http://www.cs.umn.edu/~metis`
18. Karypis, G., Kumar, V.: Multilevel $k$-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing 48(1), 96–129 (1998), `http://www.cs.umn.edu/~karypis`
19. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1) (1999); A short version appears In: Intl. Conf. on Parallel Processing 1995, `http://www.cs.umn.edu/~karypis`
20. Schloegel, K., Karypis, G., Kumar, V.: Graph partitioning for high performance scientific simulations. In: Dongarra, J., et al. (eds.) CRPC Parallel Computing Handbook, Morgan Kaufmann, San Francisco (2000)

21. Kumar, R.B.S., Das, S.K.: Graph partitioning for parallel applications in heterogeneous grid environments. In: Proceedings of the 2002 International Parallel and Distributed Processing Symposium (2002)
22. Walshaw, C., Cross, M.: Multilevel Mesh Partitioning for Heterogeneous Communication Networks. Future Generation Comput. Syst. 17(5), 601–623 (2001) (originally published as Univ. Greenwich Tech. Rep. 00/IM/57)
23. Walshaw, C., Cross, M.: Parallel optimisation algorithms for multilevel mesh partitioning. Parallel Computing 26(12), 1635–1660 (2000)
24. Wanschoor, R., Aubanel, E.: Partitioning and mapping of mesh-based applications onto computational grids. In: GRID 2004: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID 2004), Washington, DC, USA, pp. 156–162. IEEE Computer Society, Los Alamitos (2004)