Hierarchical Partitioning and Dynamic Load Balancing for Scientific Computation

James D. Teresco

[1] Department of Computer Science, Williams College
Williamstown, MA 01267 USA
[2] Computer Science Research Institute, Sandia National Laboratories
Albuquerque, NM 87185 USA

Abstract. Cluster and grid computing has made hierarchical and heterogeneous computing systems increasingly common as target environments for large-scale scientific computation. A cluster may consist of a network of multiprocessors. A grid computation may involve communication across slow interfaces. Modern supercomputers are often large clusters with hierarchical network structures. For maximum efficiency, software must adapt to the computing environment. We focus on partitioning and dynamic load balancing, in particular on hierarchical procedures implemented within the Zoltan Toolkit, guided by DRUM, the Dynamic Resource Utilization Model. Here, different balancing procedures are used in different parts of the domain. Preliminary results show benefits to using hierarchical partitionings on hierarchical systems.

Modern three-dimensional scientific computations must execute in parallel to achieve acceptable performance. Target parallel environments range from clusters of workstations to the largest tightly-coupled supercomputers. Hierarchical and heterogeneous systems are increasingly common. Grid technologies make Internet execution more likely. Modern supercomputers often include hierarchical interconnection networks. Software efficiency may be improved using optimizations based on system characteristics and domain knowledge. Our focus has been on resource-aware partitioning and dynamic load balancing, achieved by adjusting target partition sizes or the choice of a dynamic load-balancing procedure or its parameters, or by using a combination of load-balancing procedures. For hierarchical and heterogeneous systems, different choices may be appropriate in different parts of the parallel environment. There are tradeoffs in execution time and partition quality (*e.g.*, surface indices, interprocess connectivity, strictness of load balance) [31] and some may be more important than others in some circumstances. For example, consider a cluster of symmetric multiprocessor (SMP) nodes connected by Ethernet. A more costly graph partitioning can be done to partition among the nodes, to minimize communication across the slow network interface, possibly at the expense of some computational imbalance. Then, a fast geometric algorithm can be used to partition independently within each node.

An effective partitioning or dynamic load balancing procedure maximizes efficiency by minimizing processor idle time and interprocessor communication.

While some applications can use a static partitioning throughout a computation, others, such as adaptive finite element methods, have dynamic workloads that necessitate dynamic load balancing during the computation. Partitioning and dynamic load balancing can be performed using recursive bisection methods [1, 28, 30, 34], space-filling curve (SFC) partitioning [6, 23–25, 33] and graph partitioning (including spectral [26, 28], multilevel [5, 19, 21, 32], and diffusive methods [9, 20, 22]). Each algorithm has characteristics and requirements that make it appropriate for certain applications; see [3, 31] for examples.

The Zoltan Parallel Data Services Toolkit [10, 12] provides dynamic load balancing and related capabilities to a wide range of dynamic, unstructured and/or adaptive applications. Using Zoltan, application developers can switch partitioners simply by changing a run-time parameter, facilitating comparisons of the partitioners' effect on the applications. We focus here on the hierarchical balancing procedures we have implemented within Zoltan, where different procedures are used in different parts of the computing environment.

Hierarchical Balancing

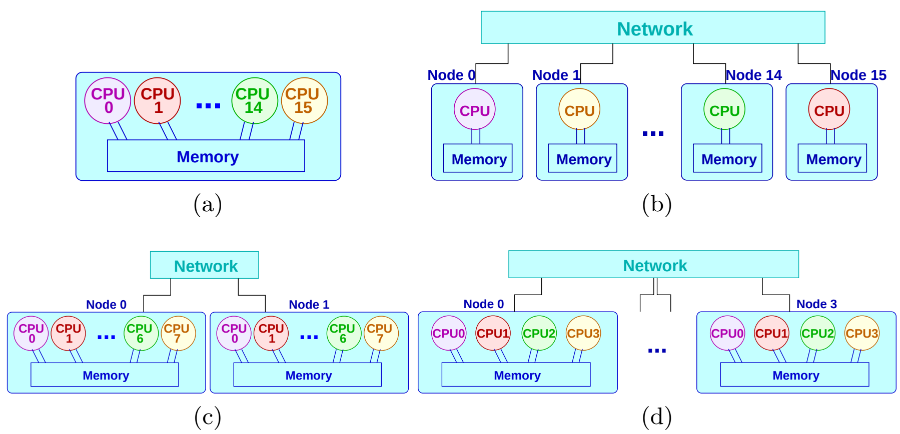

Fig. 1. Examples of parallel computing environments with 16 processors: (a) a 16-way SMP workstation; (b) a 16-node computer with all uniprocessor nodes, connected by a network; (c) two 8-way SMP workstations connected by a network; and (d) four 4-way SMP workstations connected by a network.

Consider four different 16-way partitionings of a 1,103,018-element mesh used in a simulation of blood flow in a human aorta [29]. Here, the geometric method recursive inertial bisection (RIB) [27] and/or the multilevel graph partitioner in ParMetis [21] are used for partitioning. Only two partition quality factors are considered: computational balance, and two *surface index* measures, although other factors [4] should be considered. The *global surface index* (*GSI*) measures the overall percentage of mesh faces inter-partition boundaries, and the *maximum local surface index* ($MLSI$) measures the maximum percentage of any one partition's faces that are on a partition boundary. Partitioning using RIB achieves an excellent computational balance, with partitions differing by no more than one element, and medium-quality surface indices with $MLSI = 1.77$ and $GSI = 0.61$. This would be useful when the primary concern is a good computational balance, as in a shared-memory environment (Figure 1b). Using only ParMetis achieves excellent surface index values, $MLSI = 0.40$ and $GSI = 0.20$, but at the expense of a large computational imbalance, where partition sizes range from 49,389 to 89,302 regions. For a computation running on a network of workstations (NOW) (Figure 1b), it may be worth accepting the significant load imbalance to achieve the smaller communication volume.

For hierarchical systems, a hybrid partitioning may be desirable. Consider the two SMP configurations connected by a slow network as shown in Figure 1c,d. In the two 8-way node configuration (Figure 1c), ParMetis is used to divide the computation between the two SMP nodes, resulting in partitions of 532,063 and 570,955 regions, with $MLSI = 0.06$ and $GSI = 0.03$. Within each SMP, the mesh is partitioned eight ways using RIB, producing partitions within each SMP balanced to within one element, and with overall $MLSI = 1.88$ and $GSI = 0.56$. Since communication across the slow network is minimized, this is an appropriate partitioning for this environment. A similar strategy for the four 4-way SMP configuration (Figure 1d) results in a ParMetis partitioning across the four SMP nodes with 265,897, 272,976, 291,207 and 272,938 elements and $MLSI = 0.23$ and $GSI = 0.07$. Within each SMP, partitions are again balanced to within one element, with overall $MLSI = 1.32$ and $GSI = 0.32$.

Zoltan's hierarchical balancing automates the creation of such partitions. It can be used directly by an application or be guided by the tree representation of the computational environment created and maintained by the Dynamic Resource Utilization Model (DRUM) [11, 14, 15]. DRUM is a software system that supports automatic resource-aware partitioning and dynamic load balancing for heterogeneous, non-dedicated, and hierarchical computing environments. DRUM dynamically models the computing environment using a tree structure that encapsulates the capabilities and performance of communication and processing resources. The tree is populated with performance data obtained from *a priori* benchmarks and dynamic monitoring agents that run concurrently with the application. It is then used to guide partition-weighted and hierarchical partitioning and dynamic load balancing. Partition-weighted balancing is discussed further in [15].

The hierarchical balancing implementation utilizes a lightweight intermediate structure and a set of callback functions that permit an automated and efficient hierarchical balancing which can use any of the procedures available within Zoltan without modification and in any combination. Hierachical balancing is invoked by an application the same way as other Zoltan procedures. Since Zoltan is data-structure neutral, it operates on generic "objects" and interfaces with applications through callback functions. A hierarchical balancing step begins by building an intermediate structure using these callbacks. This structure is an augmented version of the graph structure that Zoltan builds to make use of the ParMetis and Jostle [32] libraries. The hierarchical balancing procedure then provides its own callback functions to allow existing Zoltan procedures to be used to query and update the intermediate structure at each level of a hierarchical balancing. After all levels of the hierarchical balancing have been completed, Zoltan's usual migration arrays are constructed and returned to the application. Thus, only lightweight objects are migrated internally between levels, not the (larger and more costly) application data.

We have tested our procedures using a software package called *LOCO* [18], which implements a parallel adaptive discontinuous Galerkin [2, 7, 8] solution of the compressible Euler equations. We consider the "perforated shock tube" problem, which models the three-dimensional unsteady compressible flow in a cylinder containing a cylindrical vent [16]. This problem was motivated by flow studies in perforated muzzle brakes for large calibre guns [13]. The initial mesh contains 69,572 tetrahedral elements. We stop the computation after 4 adaptive steps, when the mesh contains 254,510 elements.

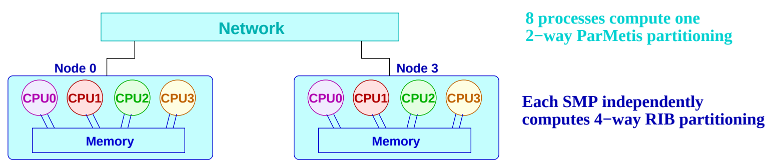

Fig. 2. Hierarchical balancing algorithm selection for two 4-way SMP nodes connected by a network.

Preliminary results are promising. We use two eight-processor computing environments: one with four Sun Enterprise 220R servers, each with two 450MHz Sparc UltraII processors, the other with two Sun Enterprise 420R servers, each with four 450MHz Sparc UltraII processors. In both cases, inter-node communication is across fast (100 Mbit) Ethernet. A comparison of running times for the perforated shock tube in these computing environments for all combinations of traditional and hierarchical procedures shows that while ParMetis multilevel graph partitioning alone often achieves the fastest computation times, there is some benefit to using hierarchical load balancing where ParMetis is used for inter-node partitioning and inertial recursive bisection is used within each node. For example, in the four-node environment (Figure 2), the computation time following the fourth adaptive step is 571.7 seconds for the hierarchical procedure with ParMetis and RIB, compared with 574.9 seconds for ParMetis alone, 702.7 seconds for Hilbert SFC partitioning alone, 1508.2 seconds for recursive coordinate bisection alone, and 822.9 seconds for RIB alone. It is higher for other hierarchical combinations of methods.

Discussion

While a traditional multilevel graph partitioning is often the most effective for this application and this computing environment, the results to date are promising and demonstrate the potential benefits of hierarchical procedures. We have demonstrated the ability to use the hierarchical balancing implemented within Zoltan as both an initial partitioning and a dynamic load balancing procedure for a realistic adaptive computation. Studies are underway that utilize hierarchical balancing on larger clusters, on other architectures, and with a wider variety of applications. We expect that hierarchical balancing will be most beneficial when the extreme hierarchies found in grid environments are considered.

Enhancements to the hierarchical balancing procedures will focus on usability and efficiency. A better integration with DRUM's machine model and graphical configuration tool will facilitate and automate the selection of hierarchical procedures. Efficiency may be improved by avoiding unnecessary updates to the intermediate structure, particularly at the lowest level partitioning step. Maintaining the intermediate structure across subsequent rebalancing steps would reduce startup costs, but is complex for adaptive problems. It would also be beneficial to avoid building redundant structures, such as when ParMetis is used at the highest level of a hierarchical balancing, however this would require some modification of individual Zoltan methods, which we have avoided thus far.

The intermediate structure itself has potential benefits outside of hierarchical balancing. It could be used to allow incremental enhancements and post-processing "smoothing" [17] on a decomposition before Zoltan returns its migration arrays to the application. The intermediate structure could also be used to compute multiple "candidate" decompositions with various algorithms and parameters, allowing Zoltan or the application to compute statistics about each and only accept and use the one deemed best.

Partitioning is only one factor that may be considered for an effective resource-aware computation. Ordering of computation and of communication, data replication to avoid communication across slow interfaces, and use of multithreading are other resource-aware enhancements that may be used. DRUM's machine model currently includes some information that may be useful for these other types of optimizations, and it will be augmented to include information to support others.

Acknowledgements

This work is in collaboration with Joseph Flaherty and Jamal Faik at Rensselaer Polytechnic Institute (Troy, New York, USA), and with Karen Devine at Sandia National Laboratories (Albuquerque, New Mexico, USA). Support provided by Sandia contract PO15162 and the Computer Science Research Institute at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

References

1. M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36:570–580, 1987.
2. R. Biswas, K. D. Devine, and J. E. Flaherty. Parallel, adaptive finite element methods for conservation laws. *Appl. Numer. Math.*, 14:255–283, 1994.
3. E. Boman, K. Devine, R. Heaphy, B. Hendrickson, M. Heroux, and R. Preis. LDRD report: Parallel repartitioning for optimal solver performance. Technical Report SAND2004–0365, Sandia National Laboratories, Albuquerque, NM, February 2004.
4. C. L. Bottasso, J. E. Flaherty, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. The quality of partitions produced by an iterative load balancer. In B. K. Szymanski and B. Sinharoy, editors, *Proc. Third Workshop on Languages, Compilers, and Runtime Systems*, pages 265–277, Troy, 1996.
5. T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization". In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452. SIAM, 1993.
6. P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.
7. B. Cockburn, S.-Y. Lin, and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-Dimensional systems. *J. Comput. Phys.*, 84:90–113, 1989.
8. B. Cockburn and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: General framework. *Math. Comp.*, 52:411–435, 1989.
9. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel and Dist. Comput.*, 7:279–301, 1989.
10. K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.
11. K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. Technical Report Technical Report CS-04-02, Williams College Department of Computer Science, 2004. Submitted to *Appl. Numer. Math.*
12. K. D. Devine, B. A. Hendrickson, E. Boman, M. St. John, and C. Vaughan. *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377.
13. R. E. Dillon Jr. A parametric study of perforated muzzle brakes. ARDC Tech. Report ARLCB-TR-84015, Benét Weapons Laboratory, Watervliet, 1984.
14. J. Faik, L. G. Gervasio, J. E. Flaherty, J. Chang, J. D. Teresco, E. G. Boman, and K. D. Devine. 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 '04.
15. J. Faik, L. G. Gervasio, J. E. Flaherty, J. Chang, J. D. Teresco, E. G. Boman, and K. D. Devine. A model for resource-aware load balancing on heterogeneous clusters. Technical Report CS-04-03, Williams College Department of Computer Science, 2004. Submitted to Cluster '04.
16. J. E. Flaherty, R. M. Loy, M. S. Shephard, M. L. Simone, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Distributed octree data structures and local refinement method for the parallel solution of three-dimensional conservation laws. In M. Bern,

J. Flaherty, and M. Luskin, editors, *Grid Generation and Adaptive Algorithms*, volume 113 of *The IMA Volumes in Mathematics and its Applications*, pages 113–134, Minneapolis, 1999. Institute for Mathematics and its Applications, Springer.

17. J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws. *J. Parallel Distrib. Comput.*, 47:139–152, 1997.
18. J. E. Flaherty, R. M. Loy, M. S. Shephard, and J. D. Teresco. Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In B. Cockburn, G. Karniadakis, and S.-W. Shu, editors, *Discontinous Galerkin Methods Theory, Computation and Applications*, volume 11 of *Lecture Notes in Compuational Science and Engineering*, pages 113–124, Berlin, 2000. Springer.
19. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*, 1995.
20. Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. Preprint DL-P-95-011, Daresbury Laboratory, Warrington, WA4 4AD, UK, 1995.
21. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scien. Comput.*, 20(1), 1999.
22. E. Leiss and H. Reddy. Distributed load balancing: design and performance analysis. *W. M. Kuck Research Computation Laboratory*, 5:205–270, 1989.
23. W. F. Mitchell. Refinement tree based partitioning for adaptive grids. In *Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computing*, pages 587–592. SIAM, 1995.
24. A. Patra and J. T. Oden. Problem decomposition for adaptive *hp* finite element methods. *Comp. Sys. Engng.*, 6(2):97–109, 1995.
25. J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with space filling curves. *IEEE Trans. on Parallel and Distributed Systems*, 7(3):288–300, 1996.
26. A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430–452, 1990.
27. M. S. Shephard, J. E. Flaherty, H. L. de Cougny, C. Özturan, C. L. Bottasso, and M. W. Beall. Parallel automated adaptive procedures for unstructured meshes. In *Parallel Comput. in CFD*, number R-807, pages 6.1–6.49. Agard, Neuilly-Sur-Seine, 1995.
28. H. D. Simon. Partitioning of unstructured problems for parallel processing. *Comp. Sys. Engng.*, 2:135–148, 1991.
29. C. A. Taylor, T. J. R. Hugues, and C. K. Zarins. Finite element modeling of blood flow in arteries. *Comput. Methods Appl. Mech. Engrg.*, 158(1–2):155–196, 1998.
30. V. E. Taylor and B. Nour-Omid. A study of the factorization fill-in for a parallel implementation of the finite element method. *Int. J. Numer. Meth. Engng.*, 37:3809–3823, 1994.
31. J. D. Teresco and L. P. Ungar. A comparison of Zoltan dynamic load balancers for adaptive computation. Technical Report CS-03-02, Williams College Department of Computer Science, 2003.
32. C. Walshaw and M. Cross. Parallel Optimisation Algorithms for Multilevel Mesh Partitioning. *Parallel Comput.*, 26(12):1635–1660, 2000.
33. M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proc. Supercomputing '93*, pages 12–21. IEEE Computer Society, 1993.
34. R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency*, 3:457–481, October 1991.