Data Partitioning for Load-Balance and Communication Bandwidth Preservation

Data Partitioning for Load–Balance and Communication Bandwidth Preservation

S. Lennart Johnsson

TR-35-95

August 1995

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

To appear in *The Second International Conference on Massively Parallel Processing using Optical Interconnections, 1995*

Data Partitioning for Load–Balance and Communication Bandwidth Preservation

S. Lennart Johnsson
Department of Computer Science
University of Houston
Houston, Texas 77204-3475

Abstract

Preservation of locality of reference is the most critical issue for performance in high performance architectures, especially scalable architectures with electronic interconnection networks. We briefly discuss some of the issues in this paper and give results from the use of spectral bisection on irregular grid computations on the Connection Machine systems.

1 Introduction

Preserving locality of reference in memory hierarchies is the most critical issue with respect to performance in most computer systems and is becoming even more important for technological reasons. MOS memory chips, by way of the technology and standard designs, are considerably slower than processor chips. The improvement in memory speed is on the order of 5 – 10 % per year, while the improvement in processor speed is about one order of magnitude every 18 months. The architectural solutions to diminish the performance impact of this disparity in speed between memory chips and processors are banked and interleaved memories combined with one or more levels of cache. Whereas the difference in speed between typical processors and memory chips today may be a factor of two to five, this factor will increase rapidly over the next few years. For scalable architectures, the interconnection network represents a major expense and often a potential performance bottleneck. The network bandwidth per node in many current designs are one order of magnitude or more lower than the peak local memory bandwidth. Hence, the allocation of the address space to the memory, and the scheduling of operations are particularly critical for the achieved performance in scalable architectures, but significant for any high performance architecture.

For many architectures and applications, the optimum allocation and scheduling of operations may reduce the communications bandwidth requirement compared to a naive allocation by more than two orders of magnitude for common operations and granularities of systems. The average number of local references for each remote reference is shown in Table 1 for a few typical regular computations. For a random allocation of data array elements to the memory address space, there are at best only a few local data references for each remote reference. In addition to the difference in the nodal bandwidth requirement, the placement of data aggregates among the processing nodes may also affect the congestion for many network topologies. Various routing strategies, such as adaptive techniques [2, 3, 4, 7, 8] or ROMM routing [26, 27] can be used to reduce the impact of congestion, but even when such routing techniques are employed some degradation of performance may still prevail compared to placement that inherently offers a low congestion.

Operation	Local storage (words)					
	Processor		Chip		Board/Machine	
	Reg.	1k	32k	1M	32M	1G
Matrix Mult.	1	16	90	512	2896	16384
3D-Relaxation	2	19	32	77	307	1024
FFT	1	20	32	45	57	70

Table 1: Number of local operations per remote reference. 3D-Relaxation: 7-point stencil, vector-length 8 ($\alpha = 8$, $\beta = 96$).

For small to modest memory sizes per node, we have observed on the Connection Machine systems CM–2/200 and CM–5 communication performance improvements by over one order of magnitude for irregular grid computations by using techniques for maximizing the number of references local to a node, and local sharing of gathered remote data.

Though the discussion in this paper mainly focus on scalable architectures, the techniques can also be used to enhance the performance of uniprocessor systems with a memory hierarchy consisting of registers, one or more levels of cache, main memory and secondary storage, in particular if that has the form of a RAID disc system.

Applications requiring large amounts of computer time exist in, for instance, fluid dynamics (solving Navier–Stokes equations), in combustion, in electromagnetics, in structural mechanics (crash studies with airbags), crack propagation studies, and in computational chemistry and fundamental physics. Many of these problems require complex geometries to be modeled, resulting in so–called unstructured meshes for domain discretization, or in the case of hierarchical methods, unbalanced decomposition hierarchies

(trees). We will briefly describe some of the data allocation issues for both structured and unstructured space decompositions in this paper.

2 Partitioning of Data Structures

2.1 Regular arrays

For many applications that make use of regular arrays, the data reference pattern consists of a sequence of references local with respect to the index space. Moreover, if the data references are uniform with respect to the index space, i.e., the relative data references for each point of the index space are the same, then minimizing the surface area for the collection of indices mapped to a processing node is an effective mapping with respect to conserving the need for communication bandwidth. Many BLAS operations [19, 5, 6] such as matrix–vector multiplication and matrix–matrix multiplications fall in this category, and so does the Fast Fourier Transform. Explicit methods for the solution of partial differential equations on regular grids is another set of applications for which minimizing the surface area for a given volume often is very effective in optimizing the performance. It has also been shown that this allocation strategy is very effective for hierarchical methods [13, 35], such as the fast N–body algorithms by Anderson's [1], Rohklin and Greengard [12] and others.

However, far from all computations making use of regular arrays as data structures make use of data references that are uniform relative to each index in the index space. A good example is direct solvers such as LU and QR factorization and the associated triangular system solvers. Whenever the indices in the index space are treated differently, load–balance becomes an issue. A *cyclic* data allocation [17] as opposed to the *consecutive (block)* mapping described above is often suggested as a means to achieve load–balance. Though the cyclic mapping does offer load–balance for the direct solvers, it may not be desirable for other computations on the same data structures. Fortunately, for the factorization and triangular solvers, it is possible to schedule the computations with respect to the indices of the address space in such a way (cyclic) for a block allocation, that the same load–balance is achieved as for a cyclic data allocation [20].

Hence, we notice that though block allocation often is very effective in conserving the demands on communications bandwidth, it may require a change of the order in which indices are processed compared to the conventional consecutive processing of indices. Which such a change in the scheduling of computations, data reallocations can be avoided without loss of efficiency.

2.2 Irregular arrays

As is the case for regular array computations with localized data references with respect to the index space, a partitioning of the data structure into subdomains preserve locality of reference when the computations use data with indices in some local neighborhood of the data being updated. This property holds for most iterative methods for the solution of sparse systems of equations, and for explicit methods for the solution of partial differential equations.

Two general partitioning techniques of significant recent interest are the recursive spectral bisection technique proposed by Pothen et. al. [28] and the geometric approach proposed by Miller et. al. [23, 22, 32]. The recursive spectral bisection technique has been used successfully by Simon [31] for partitioning of finite volume and finite element meshes. A parallel implementation of this technique has been made by Johan [14].

The spectral partitioning technique is based on the eigenvector corresponding to the smallest nonzero eigenvalue of the Laplacian matrix associated with the graph to be partitioned. The Laplacian matrix is constructed such that the smallest eigenvalue is zero and its corresponding eigenvector consists of all ones. The eigenvector associated with the smallest nonzero eigenvalue is called the *Fiedler vector* [9, 10, 11]. Grid partitioning for finite volume and finite element methods is often based on a dual mesh representing finite volumes or elements and their adjacencies (or some approximation thereof) rather than the graph of nodal points. The reason for using a volume or element based graph is that the computations are naturally organized as volume or elementwise computations. These computations exhibit locality of reference within the volumes or elements and can often be performed as a (large) collection of dense matrix operations. Communication is required when passing data between the global representation, and the representation of the collection of local elements [18, 21]. The purpose of the partitioning is to minimize this communication.

For finite element computations, the adjacency in applying the spectral bisection method has been approximated by elements that share faces. This adjacency accurately represents the communication requirements for finite volume methods. However, in finite element methods, communication is also required between elements sharing edges and corners. With nodes representing elements and edges connecting elements that share faces, the spectral partitioning yields a partitioning of the elements. Finite element nodal points internal to a partition are mapped to the processing node to which the partition is assigned. Boundary nodes must be assigned to one of the partitions among which they are shared, or replicated among these processing nodes. In our implementations, we have used a random assignment. Only boundary nodes require communication.

One advantage of the spectral bisection technique is that it is based on the topology of the graph underlying the sparse matrix. It requires no geometric information. However, it is computationally quite demanding. The geometric partitioning technique by Miller et. al. holds promise to be computationally less demanding than the spectral decomposition technique [24, 25]. Geometric information is typically available for meshes generated for the solution of partial differential equations, but may not be present in other applications.

The results of applying the spectral bisection technique to two model problems are reported in [14, 15] and shown in Tables 2 and 3. One of the model prob-

Number of partitions	Number of shared edges	%of total	Number of shared nodes	% of total
8	188	0.8	195	2.4
16	381	1.6	396	4.8
32	752	3.1	773	9.3
64	1483	6.0	1479	17.8
128	2154	8.8	2101	25.3

Table 2: Partitioning of a planar mesh with inner boundary in the form of a double ellipse.

Number of partitions	Number of shared edges	%of total	Number of shared nodes	% of total
8	5186	2.4	2735	13.4
16	8005	3.7	4095	20.1
32	11553	5.3	5747	28.2
64	16055	7.3	7721	37.9
128	21502	9.8	9827	48.2

Table 3: Partitioning of a tetrahedral mesh between concentric spheres.

lems consists of a planar triangular mesh between an outer ellipse and an inner double ellipse. The other problem is a grid of tetrahedra between concentric cylinders. The planar grid has 8,307 nodes, 16,231 triangles, and 24,537 edges. The numbers of shared nodes and edges as a function of the number of partitions are given in Table 2. The grid for the concentric spheres consists of 20,374 nodes, 107,416 tetrahedra, and 218,807 faces.

The results of applying the spectral bisection technique in a more realistic finite element application [16] are summarized in Table 4. The spectral bisection technique in this example offered a reduction in the number of remote references by a factor of 13.2. The speedup for the gather operation was a factor of 13 and of the scatter operation the speedup was a factor of 9.6 (the scatter operation includes the time required for addition which is unaffected by the partitioning).

Another important aspect of computations with irregular grids is that address computation may be very time consuming. On a distributed memory machine, the address computations require the computation of routing information as well as local addresses. In an iterative (explicit) method, the underlying grid may be fixed for several or all iterations, and it is important with respect to performance to cache the addressing information.

In an arbitrary sparse matrix, there is no simple way of encoding the global structure. Yet, arbitrary sparse matrices may still have some local structure resulting in a block sparse matrix. Taking advantage of such a block structure for both economy in data representation, data storage and efficiency of operations, is significantly simplified by explicitly representing the blocks.

Operation	Standard allocation	Spectral bisection
Partitioning	—	66
Gather	298	23
Scatter	445	46
Computation	180	181
Total time	923	316

Table 4: Gather and scatter times in seconds on a 32–node CM–5 for 50 time steps with a 1–point integration rule for finite element computations on 19,417 nodes and 109,914 elements.

3 Allocation of aggregates

For regular arrays with uniform references along one or more axis of the array, minimizing the bandwidth requirement is equivalent to embedding a mesh in the network interconnecting the processing nodes with minimum dilation. Such embeddings are known for some networks. However, contention may be a much more significant factor for performance in many networks, and minimum congestion emulations may be the desirable target. Much less is known about such embeddings. They are more complex to find since they require consideration of both placement and routing.

For irregular grids, mapping of data aggregates to processing nodes such that proximity is preserved or contention is minimized, is a much more difficult problem than the mapping of aggregates of arrays. Instead of attempting to find the best possible map, it may be more profitable to search for a map that is guaranteed to have an acceptable worst case behavior. A randomized data placement [30, 29] reduces the risk for bottlenecks in the routing system. The randomized placement of data achieves the same communication load characteristics in a single (deterministic) routing phase as randomized routing achieves in two phases [33, 34].

Figures 1 and 2 give examples of the performance improvements achieved on the CM–2 through the use of randomized data allocation in a finite element computation on an unstructured grid. The horizontal axis shows the number of degrees of freedom and elements, while the vertical axis denotes the execution time. Each element has 24 degrees of freedom. The performance improvement for the gather instruction due to randomization is in the range 2.1 - 2.4. The improvement is increasing with the problem size. Figure 2 shows the execution times for two methods of accumulating the product vector: using the combining features of the router, and accumulation after the routing operation. Randomization of the addresses improved the router combining time by about a factor of two, but performing the routing without combining is even more effective. Table 5 gives the gather scatter times with and without randomization for a solid mechanics application [21] on the CM–200. The performance enhancement is a factor of 1.5 – 2.2, which in our experience is typical. It is rarely the case that randomization has caused a performance degradation.

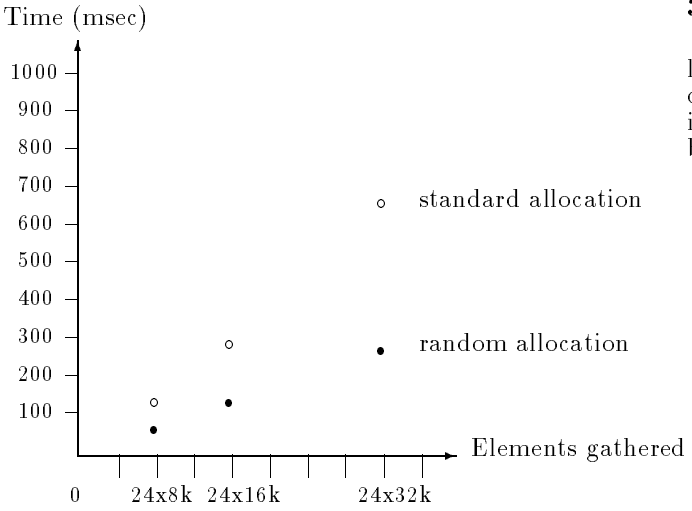

Figure 1: Gather with binary and randomized addresses. 8K CM-2.

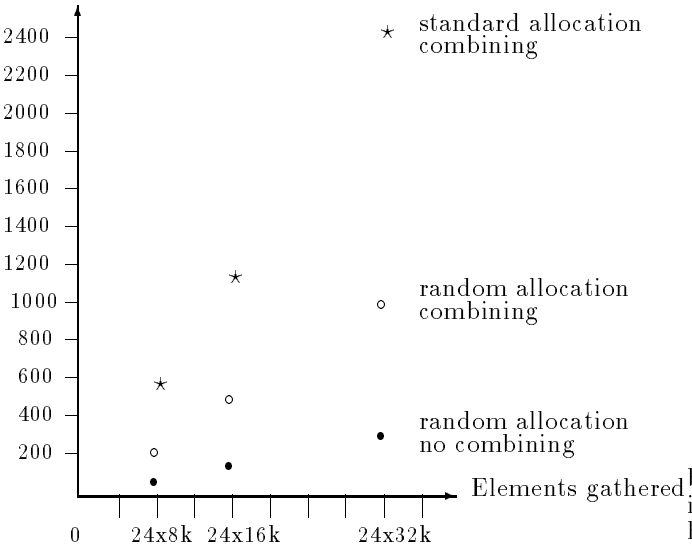

Figure 2: Accumulation of vector elements. Binary and randomized addresses. 8K CM-2.

Problem	Gather		Scatter	
	standard alloc.	random alloc	standard alloc.	random alloc
3200 20–node brick elements	75	50	124	55
864 8–node brick elements	5.6	3.7	7.2	3.4

Table 5: The effect of randomization on gather and scatter performance. Times in msec on an 8K CM–200.

3.1 A summary of data allocation issues

The assignment of data to memory units affects load–balance, communication requirements, network contention, and the performance of the computations in each node (by affecting the ability to carry out local blocking and pipelining of operations).

- Consecutive distribution preserves locality of reference along data array axes, and is suitable for stencil–like reference patterns. It also offers the possibility to improve the efficiency of the operations in each node by increasing the chance for good cache behavior through optimal blocking, and through long vectors for pipelined processors.

- Cyclic distribution significantly increases the communication requirements for relaxation methods and explicit methods for the solution of partial differential equations, and shall be avoided. Cyclic distribution may offer a reduction in the communication requirements for the FFT by a factor of two. Cyclic distribution is *not* required for load–balance in LU and QR factorization, or for the solution of triangular systems of equations.

- Irregular grids can be successfully partitioned into subdomains using recursive spectral bisection and geometric partitioning.

- An optimum distribution of partitions requires a data dependence analysis and an understanding of optimum embeddings and routing algorithms for the network at hand. For irregular computations, and when proximity preserving embeddings may not be possible, minimizing the contention through randomized distribution has shown to be effective.

Finally, we remark that data distributions cannot be determined until run–time, not only because the index sets may not be known until run–time, but it is highly desirable not to constrain the number of processors used for execution at compile time. Moreover, the decisions of what algorithm to choose for a given function, and whether or not a redistribution shall be performed before and after executing the function, must be made at run–time.

Acknowledgments

Much of the work on which this paper is based was carried out in collaboration with Zdenek Johan and Kapil Mathur while the author and the collaborators were with Thinking Machines Corp. This paper would not have been possible without the very significant contributions of Johan and Mathur.

Supporting and subsequent work has been carried out with partial support of the U.S. Air Force under grant AFOSR F49620-93-1-0480 and the U.S. Navy under ONR Grant N00014-93-1-0192 both with Harvard University. This support is gratefully acknowledged.

References

[1] Christopher R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM J. Sci. Stat. Comp.*, 13(4):923–947, July 1992.

[2] P. E. Berman, L. Gravano, G. D. Pifarré, and J. L. C. Sanz. Adaptive Deadlock– and Livelock–Free Routing With All Minimal Paths in Torus Networks. In *Proc. of the 4th Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 3–12. ACM Press, July 1992.

[3] K. Bolding. *Chaotic Routing – Design and Implementation of an Adaptive Multicomputer Network Router*. PhD thesis, Univ. of Washington, 1993.

[4] W. J. Dally and H. Aoki. Deadlock–Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Trans. on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[5] Jack J. Dongarra, J. Bunch, Clieve Moler, and G. Stewart. An extended set of basic linear algebra subprograms. *ACM TOMS*, 14(1):1 – 17, March 1988.

[6] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM TOMS*, 16(1):1 – 17, March 1990.

[7] J. Duato. A New Theory of Deadlock–Free Adaptive Routing in Wormhole Networks. *IEEE Trans. on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[8] J. Duato and P. López. Performance Evaluation of Adaptive Routing Algorithms for k–ary n–cubes. In *Proc. of the 1994 Parallel Computer Routing and Communication Workshop*. Springer-Verlag, May 1994. To appear.

[9] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305, 1973.

[10] M. Fiedler. Eigenvectors of acyclic matrices. *Czechoslovak Mathematical Journal*, 25:607–618, 1975.

[11] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25:619–633, 1975.

[12] Leslie Greengard. *The Rapid Evaluation of Potential Fileds in Particle Systems*. MIT Press, 1988.

[13] Yu Hu and S. Lennart Johnsson. Implementing $O(N)$ N–body algorithms efficiently in data parallel languages. *Journal of Scientific Programming*, 1996. Also available as TR–24–94, Harvard University, Division of Applied Sciences, September, 1994.

[14] Zdenek Johan. *Data Parallel Finite Element Techniques for Large–Scale Computational Fluid Dynamics*. PhD thesis, Department of Mechanical Engineering, Stanford University, 1992.

[15] Zdenek Johan and Thomas J. R. Hughes. An efficient implementation of the spectral partitioning algorithm on the connection machine systems. In *International Conference on Computer Science and Control*. INRIA, 1992.

[16] Zdenek Johan, Kapil K Mathur, S. Lennart Johnsson, and Thomas J.R. Hughes. An efficient communication strategy for Finite Element Methods on the Connection Machine CM–5 system. *Computer Methods in Applied Mechanics and Engineering*, 113:363–387, 1994.

[17] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Computing*, 4(2):133–172, April 1987.

[18] S. Lennart Johnsson and Kapil K. Mathur. Data structures and algorithms for the finite element method on a data parallel supercomputer. *International Journal of Numerical Methods in Engineering*, 29(4):881–908, 1990.

[19] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM TOMS*, 5(3):308–323, September 1979.

[20] Woody Lichtenstein and S. Lennart Johnsson. Block cyclic dense linear algebra. *SIAM Journal of Scientific Computing*, 14(6):1257–1286, 1993.

[21] Kapil K. Mathur, Alan Needleman, and V. Tvergaard. Dynamic 3-d analysis of the charpy v–notch test. *Modelling and Simulation Materials Science and Engineering*, 1(4):467–484, July 1993.

[22] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Automatic mesh partitioning. In *Sparse Matrix Computations: Graph Theory Issues and Algorithms*. The Institute of Mathematics and its Applications, 1992.

[23] Gary L. Miller, Shang-Hua Teng, and Stephen A. Vavasis. A unified geometric approach to graph separators. In *32nd Annual Symposium on Foundations of Computer Science*, pages 538–547. IEEE Computer Society, 1991.

[24] Gary L. Miller and William Thurston. Separators in two and three dimensions. In *Proceedings of the 22th Annual ACM Symposium on the Theory of Computing*, pages 300–309. ACM Press, 1990.

[25] Gary L. Miller and Stephen A. Vavasis. Density graphs and separators. In *Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 331–336. ACM Press, 1991.

[26] Ted Nesson and S. Lennart Johnsson. ROMM routing: A class of efficient minimal routing algorithms. In Kevin Bolding and Lawrence Snyder, editors, *Proceedings of the Parallel Computer Routing and Communication Workshop*, pages 185–199. Springer–Verlag, 1994. Lecture Notes in Computer Science, 853.

[27] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 275–287. ACM Press, 1995.

[28] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[29] Abhiram Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, pages 185–194. IEEE Computer Society, October 1987.

[30] Abhiram G. Ranade, Sandeep N. Bhatt, and S. Lennart Johnsson. The Fluent abstract machine. In *Advanced Research in VLSI, Proceedings of the fifth MIT VLSI Conference*, pages 71–93. MIT Press, 1988.

[31] Horst D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.

[32] Shang-Hua Teng. *Points, Spheres, and Separators: a unified geometric approach to graph partitioning*. PhD thesis, Carnegie-Mellon University, School of Computer Science, 1991.

[33] Leslie Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11:350–361, 1982.

[34] Leslie Valiant and G.J. Brebner. Universal schemes for parallel communication. In *Proc. of the 13th ACM Symposium on the Theory of Computation*, pages 263–277. ACM, 1981.

[35] Feng Zhao and S. Lennart Johnsson. The parallel multipole method on the Connection Machine. *SIAM J. Stat. Sci. Comp.*, 12(6):1420–1437, 1991.