A Load Balancing Package for Domain Decomposition on Distributed Memory Systems *

X. Yuan[1], B. He[1], D. Balsara[2] and R. Melhem[1]

[1] Department of Computer Science, The Univ. of Pittsburgh, Pittsburgh, PA 15260.
[2] The Nat. Center for Supercomputing Applications, The Univ. of Illinois at Urbana-Champain, Champaign, IL 61820.

Abstract. We present a tool for balanced decomposition of spatial domains. In addition to applying a nested bisection algorithm to determine the boundaries of each subdomain, the tool replicates a user specified zone along the boundaries of the subdomain in order to minimize future interactions between subdomains. Results of running the tool on the Cray T3D system, as well as using it in the implementation of a parallel Particle-particle/Particle-Mesh code are reported and discussed.

1 Introduction

A wide range of scientific applications suffer from load imbalances and excessive interprocessor communications when they are ported for execution on massively parallel processing systems. In this paper, we will introduce a dynamic load balancing package, Bisect, which we developed on the CRAY T3D system and which has been ported to the IBM SP-2 and to workstation clusters using MPI primitives. The package uses the *nested bisection* method to achieve dynamic load balancing for applications in which computations are non-uniformly distributed over some physical domain [3]. Specifically, given N objects, each being associated with a position in a physical domain, the domain is decomposed into disjoint subdomains, and each subdomain, along with the objects it contains, is mapped to a different processor. The goal is to choose a decomposition that will balance the computation among the processors in the system.

In addition to the load balancing feature, the Bisect package also adopts a technique called *domain extension* to optimize the communication in applications which exhibit local spatial dependencies. In such applications, the computation associated with an object, not only depends on the attributes of that object, but also on the attributes of the objects that are spatially close to it. Bisect has the capability of extending the subdomain assigned to each processor to contain those objects which are not in that subdomain, but affect the computation associated with it. These objects are called *ghost objects*.

* This work is in part supported by an NSF Grand Challenge grant ASC-9318185

The *nested bisection* technique and the *domain extension* technique are combined seamlessly in the Bisect package to achieve good performance. This package can be used for a large class of scientific applications which exhibit the *spatially dependent fine-grained parallelism*. Examples include Molecular Dynamics and N-body simulations [4, 5], Finite Elements and Finite Difference methods on irregular grids, Algebraic Multigrid computations, and particle-particle/particle-mesh (P^3M) methods [1, 4].

2 Nested Bisection with Domain Extension

The Bisect package assumes that, initially, each processor in a p processor distributed memory system has $n = \frac{N}{p}$ objects distributed over a physical domain. Each object has a number of data attributes and is associated with a weight which may be set to reflect the amount of computation needed to process that object.

The nested bisection algorithm [2] is incorporated in the Bisect package to redistribute the objects among the processors such that: 1) each processor is assigned objects that fall in a rectangular 3-dimensional subdomain, and 2) the sum of the weights of the objects assigned to each processor is almost uniform among the processors. In each iteration of the bisection algorithm, the domain is decomposed into two subdomains so that the difference between the sums of the weights on either subdomain is as small as possible. Then, the same process is applied on the two subdomains in parallel and this process is repeated $\log p$ times. The following lemmas establish some bounds on the performance of parallel bisection when all objects have equal weights.

Lemma 1. *If S_p is the execution time for a p processor nested bisection algorithm, then the speedup of the algorithm is bounded by $\frac{S_2}{S_p} = O(\frac{p}{\log p})$, here we assume p is a power of two.*

Lemma 2. *The maximum memory requirement for the nested bisection algorithm in each processor is $O(n\sqrt{p})$.*

To address the interprocessor communication problem, a *domain extension* technique is used in Bisect. In domain extension, each processor gets not only the "real objects" which are in the domain assigned to it, but also the "ghost objects" which are not in that domain but contribute to the computations performed on the "real objects". In other words, each processor gets objects in its own domain and copies of the objects that are surrounding the domain, within a user specified distance. This way, all the information needed to complete the computation in a processor is stored locally. Once the objects are redistributed by Bisect, no further communication is needed during the computation. The following lemma characterizes the memory requirement for the nested bisection with domain extension.

Lemma 3. *The maximum memory requirement for the nested bisection algorithm with domain extension in each processor is $O(np)$.*

By incorporating the domain extension into the nested bisection, the package is able to achieve both load balancing and optimization of the communication in the application. Our experimental results on both randomly generated objects and heavily clustered objects show that Bisect is a very efficient package both in terms of execution time and the effect on load balancing.

3 The Bisect Package

Although the main idea in Bisect is to redistribute the objects so that the computational load is balanced in each processor and to obtain the extended domain, Bisect provides many features that give the user maximum flexibility. Specifically, besides specifying the domain and the problem size in terms of the number of objects and the number of attributes for each object, the user may specify the following:

- A bin size to indicate the accuracy of the load balancing algorithm, as described later,
- A choice among the following load balancing criteria: 1) balance on the domain size. The resulting subdomains will have the same sizes. 2) balance on the number of objects. The resulting subdomains will have almost the same number of objects. 3) balance on the weights. The sum of weights in the resulting subdomains will be almost the same.
- The size by which each subdomain is to be extended.
- A choice of periodic and non-periodic boundary conditions for extending the exterior boundaries of the subdomains.
- The sequence of dimensions that are to be bisected, or a default automatic bisection. The automatic bisection results in the domain being bisected in each dimension in a round-robin manner. If the user has some specific knowledge of object distribution, (s)he can manage the bisected dimensions to achieve the best load balancing results.

Assume that, before the i^{th} iteration of Bisect, $1 \leq i \leq \log p$, the objects in a given subdomain are distributed among processors $P_1, \ldots, P_k$, where $k = \frac{p}{2^{i-1}}$. The process to carry out a bisection along a specific dimension with domain extension is divided into three parts:

1) Determining the bisection line using Bin Sorting: A parallel sorting algorithm is needed for this task. While an exact sorting along the dimension to be bisected can find the optimum bisection line, it introduces a high overhead of $(O(N \lg(N)))$ for sorting and counting. For efficiency considerations, we use a parallel bin sorting algorithm, where, the dimension to be bisected is divided into a certain number of bins (slices). Each processor sums the weights of its local objects that belong to each bin, and a global sum algorithm is used to obtain the total weight of the objects in each bin. This information is then used by each processor to determine the bisection line along a bin boundary. Note that the bin sorting algorithm is linear in the number of objects, and logarithmic in the number of processors (for the sum), which greatly reduces the sorting overhead. However, bin sorting restricts the cut line to be along bin boundaries and therefore, some load imbalance may be introduced. Thus, larger bin numbers result in larger runtime overhead and more balanced load.

In some scientific applications, the distribution of objects is extremely non-uniform. For these applications, the load imbalance incurred by the bin sort may cause some processors to get a null subdomain. This may cause runtime errors in the original application, since application programs generally assume non-null physical domains. Precaution is taken to prevent this situation; because the number of bisections along each dimension is known, Bisect can guarantee that each processor gets at least one bin size domain.

2) Particle handling: Once the bisection line is decided, each processor determines how to handle each of the objects it has, such that the objects in one of the two resulting subdomains are in processors $P_1, \ldots, P_{\frac{k}{2}}$ and the objects in the other subdomain are in processors $P_{\frac{k}{2}+1}, \ldots, P_k$. Specifically, a processor P_j, examines each of its "real objects", and determines one of the following: 1) to keep the object as a real object, 2) to send the object to processor $\bar{P}_j$ as a real object, where $\bar{P}_j = P_{j-\frac{k}{2}}$ or $P_{j+\frac{k}{2}}$ depending on whether $j \leq k/2$ or $j > k/2$, respectively. 3) to keep the object as a real object and send it to $\bar{P}_j$ as a ghost object, or 4) to keep the object as a ghost object and send it to $\bar{P}_j$ as a real object. In other words, when a cut is placed along a dimension, an internal boundary condition is created. All the objects residing within the user-specified distance of the boundary are duplicated as real objects in one processor and as ghost objects in another processor. Similarly, P_j examines each of its "ghost objects" and determines whether it should keep the object, send it to $\bar{P}_j$, or both. This way all the objects in the extended domain will be moved to the proper processors.

Besides the internal boundary conditions, some applications require periodic or wrap-around boundary condition for the entire problem domain. The objects in one side of the initial domain will affect the computation of the objects in the opposite side of the domain. As a result, the objects in the boundary bins of the initial problem domain must be duplicated.

3) Moving Particles: After the second phase, each processor, P_j, packs all the objects that it should send to $\bar{P}_j$ in a message and sends that message. From Lemmas 2 and 3, however, we can see that in the worst case, the memory requirement for Bisect may be quite large. Object rebalancing within each group before each bisection step will reduce the maximum memory requirements in each processor from the $O(n\sqrt{p})$ specified in Lemma 2 to $O(2n)$.

4 Experimental Results

In this section, we present experimental results for Bisect. We study the performance of Bisect on both uniformly distributed objects and non-uniformly distributed objects. We also study the effect of the number of bins and the number of processors on the performance. Here, the performance means both the runtime of Bisect and its effectiveness to achieve load balancing.

Table 1 shows the performance of Bisect on uniformly-distributed objects. We use 1000 bins along each dimension with each subdomain extended by one bin. The program is run on a 32 PE system. The results show that, in all cases, the load imbalance is less than one percent with regard to the average number of objects. It also shows that Bisect runs very fast. With 32 Cray T3D processors, it only takes a fraction of a second to decompose a domain with 1 million objects.

Table 1. Performance of Bisect on uniformly distributed objects

Total no. of objects	no. of objects per PE without ghosts				no. of objects per PE with ghosts				Time (sec)
	min	max	ave	imb. ratio	min	max	ave	imb ratio	
131072	4065	4126	4096	0.7%	4145	4213	4176	0.9%	0.10
262144	8135	8246	8192	0.7%	8312	8420	8356	0.8%	0.19
524288	16238	16514	16384	0.8%	16575	16855	16712	0.8%	0.38
1048576	32558	32982	32768	0.7%	33206	33674	33427	0.7%	0.74

Table 2. Performance for different number of bins on 524288 uniformly distributed objects

Bin #	no. of objects per PE without ghosts				no. of objects per PE with ghosts				Time (sec)
	min	max	ave	imb. ratio	min	max	ave	imb ratio	
500	16273	16514	16384	0.8%	16925	17164	17044	0.7%	0.39
1000	16306	16462	16384	0.5%	16642	16772	16713	0.4%	0.39
5000	16371	16403	16384	0.1%	16434	16472	16452	0.1%	0.43
10000	16378	16392	16384	0.0%	16404	16435	16420	0.1%	0.47

Tables 2 and 3 show the effect of the number of bins on the performance of Bisect for uniformly and non-uniformly distributed objects, respectively. Here we can see the general pattern that larger number of bins leads to better load balancing. For the uniformly distributed objects, 500 bins in each dimension achieves good load balancing. The number of bins in each dimension especially affects the performance of Bisect for the nonuniformly distributed objects. With 500 bins in each dimension, load balancing after Bisect is poor. However, when the number of bins is larger than 1000, the load balancing is good. It is in-

Table 3. Performance for different number of bins on 512000 non-uniformly distributed objects

Bin #	no. of objects per PE without ghosts				no. of objects per PE with ghosts				Time (sec)
	min	max	ave	imb. ratio	min	max	ave	imb ratio	
500	11263	21926	16000	37.0%	17483	53098	33352	59.2%	0.73
1000	12904	18213	16000	13.8%	17084	32639	23913	36.5%	0.49
5000	15585	16425	16000	2.7%	16177	18678	17442	7.1%	0.59
10000	15686	16196	16000	1.2%	16147	17353	16717	3.8%	0.89

teresting to see that the number of bins has a minor effect on the runtime of Bisect. Therefore, using large numbers of bins to achieve better load balancing is recommended for application programs that invoke Bisect.

Table 4 shows the performance of Bisect on different number of processors assuming 512K objects, 1000 bins along each dimension and a 1 bin extension domain. The second column and the third column show that the imbalance ratio increases with the increase in the number of PEs. The fifth column shows the speedup over two-processor execution time. The sixth column lists the theoretical upper bound of the algorithm which results from Lemma 1. The last column shows the percentage of actual speedup to the upper bound. The results show that, up to 128 processors, Bisect achieves 87.4 percent of the achievable speedup.

Table 4. Performance for different number of processors on 524288 uniformly distributed objects

PE #	imb. ratio without ghosts	imb. ratio with ghosts	time (sec)	actual speedup	speedup upper bound($\frac{p}{2 \times lg(p)}$)	speedup percentage
2	0.0%	0.0%	1.158	1	1	100.0%
4	0.1%	0.1%	1.163	0.996	1	99.6%
8	0.2%	0.2%	0.880	1.316	1.333	98.7%
16	0.6%	0.6%	0.593	1.953	2	97.6%
32	0.8%	0.8%	0.377	3.072	3.2	96.0%
64	1.0%	1.2%	0.234	4.949	5.333	92.8%
128	1.5%	1.6%	0.145	7.986	9.142	87.4%

5 Application to a Particle-Particle/Particle-Mesh Simulation

Bisect was used in a Particle-Particle/Particle-Mesh (P^3M) program that was written to study galaxy formation. The fundamental computational problem in particle based study of galaxy formation is to evaluate the forces at each timestep and advance the system in time. There is a close similarity between this problem and problems in plasma physics. The program is based on the algorithm given in Hockney and Eastwood [4] though other algorithms like the one in Balsara and Brandt [1] would also benefit from the strategies developed here. Thus the utility of Bisect extends to almost all forms of plasma physics simulations.

The P^3M algorithm evaluates the gravitational force on a system of self-gravitating particles. It does this by splitting the force into two parts, a long range force and a short range force. The long range force evaluation is an accurate representation of the force contribution to a given particle from all particles that are farther than a certain distance (known as the cut off distance) from it. It is obtained by making a triangular shaped charge (TSC) interpolation of the particle's mass on to a grid. A fast Fourier transform step yields the potential on the grid. The forces evaluated from this potential are then interpolated back to the particles. Because of the fact that a mesh was used by the particles this step of evaluating the long range forces is also known as a particle mesh (PM) step. Because of the interpolations involved in the PM step the force exerted on a particle from other particles that are within a cut off distance from it is not fully represented. Thus an extra step has to be put in to supply the remaining force contribution. This is the short range force and because it operates between two particles this step is known as the particle-particle (PP) step.

It is important to realize that in order to balance the load, the PM step needs a mapping of particles onto processors that is reasonably consistent with the underlying distribution of the mesh. Likewise, in order to balance the load, the PP step needs a mapping of particles onto processors that balances the local work done by the particles. Thus in each case we need a spatially dependent load balancing algorithm which is exactly what Bisect is. Moreover, the capability of Bisect to provide ghost particles to each processor minimizes the communication that is needed in the PM step and eliminates communication totally in PP steps.

In Figure 1, we present the results of executing the P^3M program on a cubic domain containing 262144 particles on the T3D using 16, 32, 64 and 128 processors. The figure shows the speedup obtained when Bisect is used to balance the number of particles in each subdomain, as opposed to producing equal size domains (spatially uniform partitions). The speedup for the PP and the PM parts are shown separately. In general, however, the execution time of the PP part is much larger than the PM part. For example, with 32 processors, the PP part in each iteration took about 27 seconds to complete, while the PM part in each iteration took about 3 seconds to complete. The time to execute Bisect was about 0.7 seconds. These execution times as well as the curves in Figure 1, highly depend on the number of particles and their initial distribution in the problem domain. A complete analysis of the effect of Bisect on balancing the

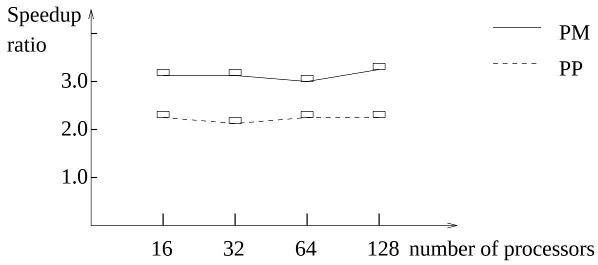

Fig. 1. The effect of load balancing on the execution of P^3M

computation load and reducing the communication load of P^3M is outside the scope of this paper.

6 Conclusion

Bisect is a powerful tool for domain decomposition on parallel systems. In addition to balancing the load in each subdomain, Bisect may augment each subdomain with copies of the objects it will need from other subdomains, thus reducing the communication overhead that may be needed for interaction on subdomain boundaries. From experimental data, we can see that the effectiveness of Bisect for balancing the load increases with the number of objects in the system. That is, Bisect is most effective when it is most needed. Moreover, by changing the number of bins in each direction, one can reach a compromise between the load balance achieved and the cost of achieving the load balance (the time to execute Bisect). In general, however, the cost of running Bisect is small relative to the gain obtained from balancing the load.

References

1. Balsara, D.S. and Brandt, A., "Multilevel Methods for Fast Solution of N-Body and Hybrid Systems", Int. Ser. Num. Math, 98, 131. 1991.
2. Bokhari, S.H., "Assignment problems in Parallel and Distributed Computing", Kluwer Academic Publishers, 1981.
3. Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. "Solving Problems on Concurrent Processors". Prentice-Hall, 1988.
4. R.W.Hockney and J.W. Eastwood, "Computer Simulation Using Particles." New York: Mcgraw Hill International, 1981.
5. Warren, M.S. and Salmon, J.K "A Parallel Hashed Oct-tree N-body Algorithm". In Supercomputing '93, PP. 12-21, 1993.

This article was processed using the LaTeX macro package with LLNCS style