# Domain Decomposition for Particle Methods on the Sphere

Ömer Eğecioğlu and Ashok Srinivasan

Department of Computer Science, University of California, Santa Barbara, CA 93106

**Abstract.** We present an algorithm for efficient parallelization of particle methods when the domain is the surface of a sphere. Such applications typically arise when dealing with directional data. We propose a domain decomposition scheme based on geometric partitioning that provides domains suitable for practical implementation. This algorithm has the advantage of being fast enough to be applied dynamically, and at the same time provides good partitions, comparable in quality to those produced by spectral graph partitioning schemes.

## 1 Introduction

Particle methods are widely used in several applications [1, 5, 4, 3]. These typically involve a set of particles represented as points in some space, and a function that describes the interaction between pairs of particles. For each particle, one independently sums the interaction between it and all other particles, and a new state for each particle is then computed. This new state typically is a new position and velocity, and the calculations are repeated several times to observe the evolution of the system. This leads to an irregular computational problem in which the set of particles which interacts with any given particle changes with time in an unpredictable manner. Furthermore, in applications involving directional data such as certain complex fluid flow problems, the natural domain of representation is the unit sphere [17, 14].

In a parallel implementation, particles are assigned to processors by first breaking the domain into subdomains, and then mapping these subdomains to different processors. Each processor proceeds to compute the interactions of all the particles in the system with the particles in its subdomain. The interactions with particles that lie outside its subdomain require a processor to obtain the states of those particles from other processors. We call the particles in the subdomain of a processor the points *owned* by the processor. The points owned by a given processor that are needed by other processors are *shared* points. In many applications such as molecular dynamics and Smoothed Particle Hydrodynamics based methods, the interacting forces between the particles are short range and the effect of particles that are farther away than a certain cut-off distance can be ignored [4, 3]. In order to further mitigate high communication costs, one usually tries to overlap computation with communication. Hence, processors first send their shared data to the processors that need them, and following this perform their local computations. Subsequently, they receive the shared data they themselves need and perform their remaining computations, and update the states of the points in their subdomain. Some updating of the domains can also be done at this stage. Usually a full domain decomposition is not performed at the end of each iteration since the cost of the decomposition can be prohibitive. The basic scheme of such calculations is outlined in Figure 1.

```
1. Domain Decomposition.
2. Map domains to processors.
3. Start loop
   (a) Determine shared points.
   (b) Send shared points to processors that need them.
   (c) Compute using interior data.
   (d) Receive shared data needed from other processors.
   (e) Compute using received data and update state.
   (f) Update domain data.
   End loop
```

**Fig. 1.** Outline of a general parallel particle method calculation.

Efficient parallelization requires the selection of subdomains for each processor in such a way that only few particles outside interact with particles within the subdomain. It is also necessary to efficiently determine the set of shared particles at each processor. An important aspect of the computations is to be able to perform effective dynamic range searching so that interactions with only those particles that are within the cut-off distance of the short range interactions are computed.

The outline of this paper is as follows. Sect. 2 describes graph-theoretical and geometric domain decomposition strategies as they apply to particle methods on the sphere. The algorithm we present in Sect. 3 is essentially a geometric partitioning based on Orthogonal Recursive Bisection [2]. However, we take advantage of the geometry of the sphere to produce partitions with quality comparable to sophisticated methods such as spectral partitioning. Experimental results and comparisons with other popular schemes available in Chaco, version 2.0 [9] and Metis, version 2.0.3 [11] are presented in Sect. 4. These experiments show that our algorithm is an order of magnitude faster than even the relatively fast inertial method for large problem sizes, and demonstrate the high quality of the partitions obtained. Conclusions are given in Sect. 5.

## 2 Domain Decomposition

Domain decomposition has been widely studied [9, 10, 11, 12, 13] and several types of methods for its solution have been proposed: graph-theoretical and geometric, for example. Graph-theoretical schemes ignore coordinate information and treat domain decomposition as a general graph partitioning problem. Geometric algorithms, in contrast, use coordinate information of the points to divide the domain into contiguous regions.

The quality of the partitions produced can be judged by the load imbalance introduced and the communication cost incurred. We try to keep the load balanced, i.e., ensure that the amount of computation performed by each processor is about equal. Subject to this restriction, we further wish to keep the communication cost low. There are several measures for estimating the communication cost. Before describing the criterion used in our algorithm, we shall briefly describe the communication pattern implied by Figure 1. In order to overlap computation and communication, each processor determines the shared data it needs to send to other processors. It is appropriate to send the required data to each processor as a single message. This reduces the startup time and is especially advantageous in systems that support long messages. If $n_i$ is the number of processors that need the data of point $i$, excluding the processor to which $i$ has been assigned, then we define the communication cost as $\sum n_i$. This measure ignores the startup time and can be justified if the messages are sufficiently long. It differs from the *hop* metric in that the number of links traversed are ignored, providing an architecture independent measure. With cut-through routing being widely prevalent, this criteria seems justified. However, it should be noted that too many messages in the system could cause network congestion and the number of links traversed could affect the true communication cost [9]. Our communication measure also differs from the commonly used edge-cut metric in graph partitioning which tries to minimize the number of edges cut. If more than one point in a particular processor needs data related to one point from another processor, the remote processor need send the data only once. The processor that receives the data can store this and reuse it when needed. In contrast to the edge-cut metric, our communication cost takes this factor into account as well.

Graph-theoretical algorithms such as spectral methods produce high quality partitions especially when combined with a local refinement strategy [13], but require too much time. When combined with multilevel methods, these give good partitions much faster [10], however, they are still not fast enough to be used frequently. Since the distribution of the points could change significantly in the types of applications we are considering, the quality of the partitions may degrade quickly.

Geometric algorithms make use of the coordinates of the points to find partitions fast. In this case the quality of the partitions obtained is usually not very good. Orthogonal Recursive Bisection (ORB) for example, bisects the domain along a coordinate. This is recursively applied using different coordinates. This scheme is fast, though the quality, as judged by the communication cost incurred, could be poor. Another method that uses coordinate information is the Inertial Method. This method produces partitions which are usually of a higher quality than those produced by ORB, at the expense of a slight increase in the computational effort required to produce the partitions. Alternate approaches to parallelization of particle methods can be found in [15]. Our scheme resembles ORB, but takes advantage of certain metric properties of the surface of the sphere to give good partitions.

Before describing our algorithm, we note that one can consider stereographically projecting the points on the sphere onto the plane and then using an existing partitioning algorithm for the plane. However, points close to the projecting pole are widely separated in the projection. This distortion on locality makes geometric algorithms unsuitable.

## 3 The Algorithm

We propose an essentially geometric scheme for decomposing the surface of the sphere into regions bounded by a pair of latitudes and and a pair of longitudes. The input to the domain decomposition algorithm consists of: (i) the number $P$ of processors available, (ii) the cut-off distance $h$ for the short-range interactions, and (iii) a set of $N$ points defined by their coordinates (latitude and longitude) on the unit sphere. We assume that each point is dynamically assigned a positive weight which is proportional to the computational effort required for computing the new state of the point. Each point represents a particle in the system.

The computational effort for a given point is roughly proportional to the number of points within a distance $h$ of the given point. There are different options available to obtain a reasonable estimate of what the weight should be. A large class of problems involves compressible fluid flow calculations in which the density of the fluid has to be determined [4]. For sufficiently small $h$, the number of points within a distance $h$ of any point is approximately proportional to the density at the point. Thus, one may take the weight to be proportional to the density. If such data is not available, one may use non-parametric density estimation techniques to estimate the density [6]. In our implementation of the domain decomposition algorithm on the sphere we have used positive integral weights, though using floating point weights does not present any additional difficulty.

As its output, the domain decomposition algorithm associates an integer in the interval $[0, P-1]$ with each point, which indicates which partition (subdomain) a point belongs to. In addition, the algorithm can also produce a mapping of partitions to processors, thus determining the processor to which each point is assigned. Our algorithm produces a mapping for a tree topology, with the processors located at the leaves of the tree. This is not unduly restrictive since efficient schemes for embedding trees into other topologies exist [8].

We use a recursive bisection procedure. At each stage,

1. we first consider a cut on the subregion along the latitude that gives a balanced load, and also a longitudinal cut that gives a balanced load;
2. we then calculate and compare the communication cost of each cut, and choose the one that with the lower cost.

Thus initially, one of the first two types of cuts shown in Figure 2 is made. Being

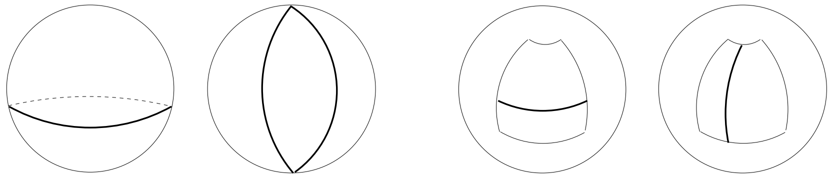

**Fig. 2.** Initial latitudinal and longitudinal cuts; generic latitudinal and longitudinal cuts.

on the surface of the sphere adds an additional complication if the region we are currently considering has not been cut by a longitude before. Such regions look either like rings or polar caps, as shown in Figure 3 (a). In this case there is no unique pair of longitudes that balances the load. In fact if there are $n$ points in such a region, then there are $O(n^2)$ possible pairs of longitudes, and $O(n)$ possible pairs among these along which we can perform a cut while keeping the load balanced. A naive strategy would determine for each fixed choice of the starting longitude of the region, the corresponding longitude to end the region which gives a balanced load. This can be done by using a sorted array of longitudes in $O(\log n)$ time per initial choice. Since there are $O(n)$ possible initial choices, the overall time appears to be $O(n \log n)$ for the determination of the cut. However, we show that this type of a longitudinal cut can actually be determined using only $O(n)$ operations. Our algorithm requires a preprocessing overhead of $O(N \log N)$. However $O(N \log N)$ preprocessing is required for sorting the $N$ points in the system according to a number of parameters anyway, so this does not add to the overall complexity. In this way, the rest of the computation takes only linear time at each level of the recursion.

The algorithm is outlined in greater detail below. Being on the surface of the sphere leads to some complicated boundary conditions to maintain. In order to ensure the clarity of the presentation, these are not elaborated on here.

### 3.1 Latitudinal cut

We first describe latitudinal cuts since they are the simpler of the two. For a latitudinal cut, we wish to divide the sphere into two parts along a certain latitude. If we sweep a latitude from the South to the North Pole, then the load balance of the two parts it divides changes discontinuously, and does so exactly at those latitudes at which the data points are located. Hence, we need to consider only these latitudes. In the preprocessing step, we keep a list of the points sorted in ascending order of latitude. We traverse this array, and perform a prefix computation to calculate the cumulative weights of all those points that have been encountered so far, including the current location. In order to find the latitude at which to cut, we perform a binary search on the array of cumulative weights to find the location that gives the best load balance, i.e., has cumulative weight closest to half the total weight of the partition. It is possible for two adjacent locations to have equally good load balance. In this case we choose one of the cuts arbitrarily. We observe that if several points have the same latitude, they may fall in different partitions. In order to compute the cost of the chosen partition, we need to know the number of points that lie within a distance $h$ of the closest latitudes of either region. We use the property that the shortest distance between a point and a latitude is the absolute value of their differences in latitude. We locate the farthest points within the cut-off distance of the latitudes defining the boundaries of the two regions. This is performed by binary search on the sorted array of latitudes. The number of points that will be communicated is one more than the difference in the positions of these points in the sorted list.

### 3.2 Longitudinal cut

There are two cases to consider for a longitudinal cut. If any ancestor (in the recursion tree) of the region being cut has been subjected to a longitudinal cut before, then this case leads to a simpler algorithm since cutting along a single longitude will necessarily divide the region into two parts. We choose this longitude in a manner analogous to that of the latitude, using a pre-sorted array of longitudes. Computing the cost requires a little more attention since the shortest distance between a point and a longitude is in general not the difference of their longitudes. If we sweep a longitude around the sphere and consider the neighborhood within distance $h$ of this longitude, then points may enter and leave this neighborhood at different values of the longitude. We can calculate the longitudes at which any point enters and leaves this neighborhood. If a point is within distance $h$ of all longitudes (near the poles), then we consider it as entering at $-\infty$ and leaving at $\infty$. We have a preprocessing step in which we create two sorted arrays based on the longitudes at which a point enters and leaves the neighborhood respectively. In order to compute the number of points within distance $h$ of a given longitude, we find the last point that has just joined the neighborhood and the next point that leaves the neighborhood. These points are found by binary search on the two sorted arrays constructed above. The difference between the number of points that have joined the neighborhood and the number that have already left gives the number of points still in the neighborhood.

In the second type of longitudinal cut, no ancestor of the region has been subjected to a longitudinal cut before. Now we need to decide on two longitudes at which to perform the cut, since one longitude alone will not divide the region into two pieces (see Figure 3 (a)). We start with the first longitude, say $A_1$, in the sorted list of longitudes, and find the corresponding longitude $A_2$ that balances the load if we start the region defined by the cut at $A_1$. We compute the cost as in the case of a cut which requires a single longitude; however, we now need to add the number of points on both boundaries. We next look at the longitude $B_1$ that comes second in the sorted list of longitudes. We find the other end $B_2$ of the possible cut not by performing a binary search on the list, but by incrementing the location of $A_2$ until we achieve load balance. This construction is as shown in Figure 3 (b). A similar procedure is carried out in computing the

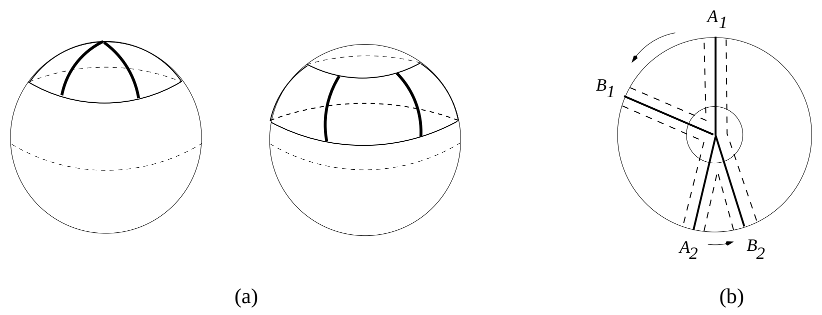

**Fig. 3.** (a) Longitudinal cuts that require a pair of longitudes; (b) Pairs of longitudes that balance the load. The center is the North Pole.

communication cost as well. We proceed in this manner around the sphere and choose the cut that has the lowest communication cost.

An example of a sequence of various latitudinal and longitudinal cuts made using this decomposition scheme is given in Figure 4.

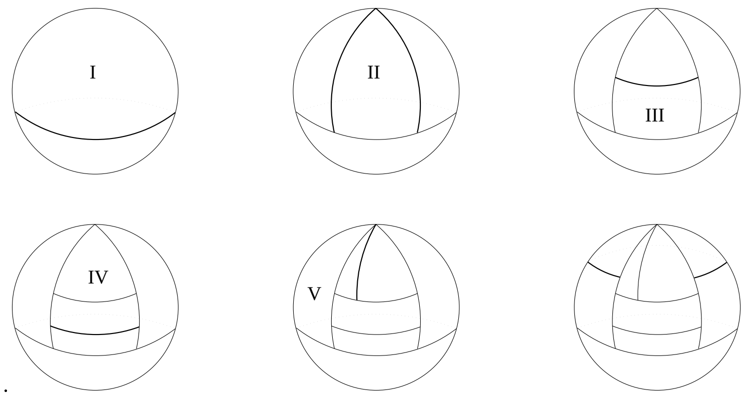

**Fig. 4.** An example: The first cut is latitudinal, producing domain I and its complement. Sample subsequent cuts on some of the subdomains are: longitudinal cut on I, latitudinal cuts on II and III, longitudinal cut on IV, and latitudinal cut on V.

### 3.3 Partitioning

After we decide on a particular cut, we need to partition the set of points. An important step here is to obtain the sorted arrays for each partition without needing to perform an extra $O(n \log n)$ sorting step. We can get the sorted arrays in linear time by scanning each already sorted array in order and placing a point at the end of the array of whichever partition it belongs too. In addition, if we performed the longitudinal cut involving two longitudes, we need to change the values of the longitudes of certain points in one of the partitions by subtracting $2\pi$ from their coordinates, so that if we sweep a longitude from one end of the partition to the other, we will remain within the partition. Getting the sorted arrays in this case also requires two passes over the sorted arrays, rather than one as in the other cases.

### 3.4 Complexity

The complexity of the initial preprocessing step is $O(N \log N)$ required for sorting the $N$ points. If there are $n$ points in a partition that is to be further partitioned, then locating the latitude, or the longitude in a longitudinal cut involving only one longitude, can be done in $O(\log n)$ time. This takes $O(n)$ time for the longitudinal cut that requires two longitudes, since the "arms" $A_1, A_2$ and $B_1, B_2$ of Figure 3 (b) sweep through the points once, without any backtracking. Generating the new partition requires $O(n)$ time for any of these types of cuts, since we need to make one or two passes to extract the sorted arrays for the subdomains themselves. Thus each level of the recursion requires $O(N)$ time, leading to $N \log P$ complexity for the recursion if $P$ parts are needed (assuming $P$ is a power of 2). In addition, we have $O(N \log N)$ preprocessing time, which gives a total complexity of $O(N \log N + N \log P) = O(N \log N)$.

## 4 Experimental Results

We performed experiments to test the performance of our algorithm, and also to compare it with existing algorithms. As a measure of the load imbalance, we considered the quantity: $tP/T$ where $P$ is the number of processors desired, $T$ is the sum of the weights of all the points, and $t$ is the sum of the weights of all the points in the processor with the largest such sum. If the load were perfectly balanced, then this quantity would be 1. In all of the experiments, we found that all the methods tested gave well balanced partitions, especially with a large number of particles. Thus we do not report further on this aspect of the experiments, and judge the quality of the partitioning based only on the communication cost incurred. In the rest of this section, the term *quality* refers to the communication cost incurred. The ratio of the communication costs that appear in our plots to $P \cdot N$ is the average communication cost incurred between pairs of processors, as a fraction of the total number of points. We obtained our data points on the sphere by generating samples from two probability density distributions. We used the rejection-acceptance technique to generate the points. We assigned weights to each point proportional to the density of the distribution at the point. The weights were rounded to integers with a minimum value of 1. We chose a value of $h$ such that it gave a reasonably good estimate when using kernels for non-parametric estimation of the probability density. This was done on the basis of practical applications in which our scheme is particularly useful [14].

We compared our algorithm with general graph partitioning algorithms, since these have been found to give good quality partitions [10]. We also compared our scheme with the inertial method, since this is a geometric method which is much faster than the general graph partitioning methods. For problems of large size, even the multilevel graph partitioning algorithms were at least two orders of magnitude slower than our algorithm. Therefore we have presented timing results comparing our algorithm only with the inertial method. The inertial method used was that implemented by Chaco, version 2.0. The spectral bisection method used the multilevel spectral eigensolver implemented in Chaco, version 2.0. The multilevel spectral bisection scheme implemented in Metis, version 2.0.3 gave partitions of similar quality, and requiring a similar run time, as that of Chaco. However, the latter was marginally faster and gave partitions of slightly better quality in a few tests, and so was used in our experiments. It should be noted that though the number of vertices in our graphs is not very high compared with many of the graphs used for tests in current literature, the relative denseness of our graphs results in a large number of edges. Many of our larger graphs have millions of edges.

**Experiment 1**: We first performed experiments to observe how the speed of our method scales with the number of points. The experiments were performed on Sun SPARCstation 5, and the timing results are reported in seconds. We present results showing the total time required, and also the time required by just the partitioning phase, ignoring the initial preprocessing time. We compare it in Figure 5 with the inertial method implemented in Chaco, version 2.0, without the Kernighan-Lin refinement. The time reported is that taken just for the partitioning. We did not use the Kernighan-Lin refinement for the inertial method because this increased the time taken significantly. Since speed is the major advantage of a geometric algorithm, we decided not to use this refinement. It can be seen that as the number of points increases, the inertial method is about an order of magnitude slower than our scheme. We also note that our scheme scales well as the number of points increases. It should also be noted that with a large number of points, the majority of the time is spent in the initial sorting, which can be easily parallelized.

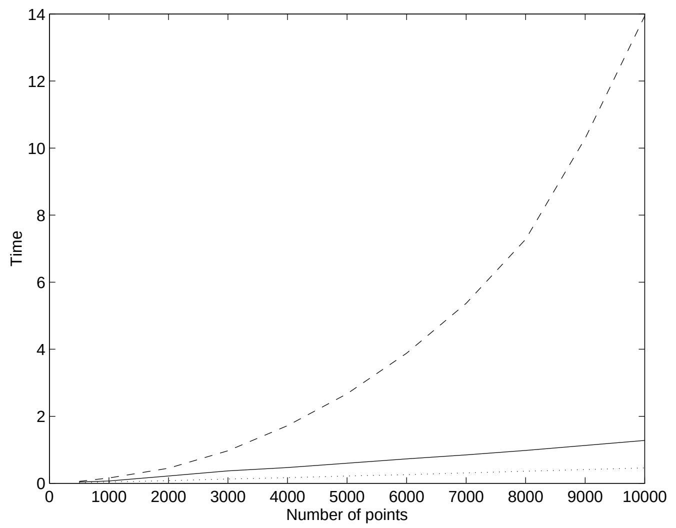

**Fig. 5.** Comparison of the speed of our algorithm and inertial partitioning into $P = 32$ parts, without Kernighan-Lin refinement. The solid line shows the time taken by our algorithm. The dotted line shows the time taken in just the partitioning phase. The dashed line shows the results for the inertial method. Time is in seconds.

**Experiment 2**: We next investigated the quality of the partitions obtained. In all the experiments reported below, the data was partitioned into $P = 32$ parts. We consider points generated on the unit sphere with the distribution: $\psi(\phi, \theta) = \exp(U \sin^2 \phi)/A$, where $U$ is a parameter, $A$ normalizes $\psi$ to a probability density, and $\phi$ and $\theta$ are the latitude and the longitude respectively. This particular function arises in the solution of a certain problem in complex fluids [14], and the distribution depends only on the latitude. Here, $-\pi/2 \leq \phi \leq \pi/2$, and $0 \leq \theta < 2\pi$. In our experiments, we have used the special case of $U = 4.6$, $A = 25.6$. The value of $h$ was taken to be 0.2.

We can see from Figure 6 that our method, inertial partitioning, and the multilevel spectral method give partitions of comparable quality for this distribution, though the spectral method gives slightly better partitions.

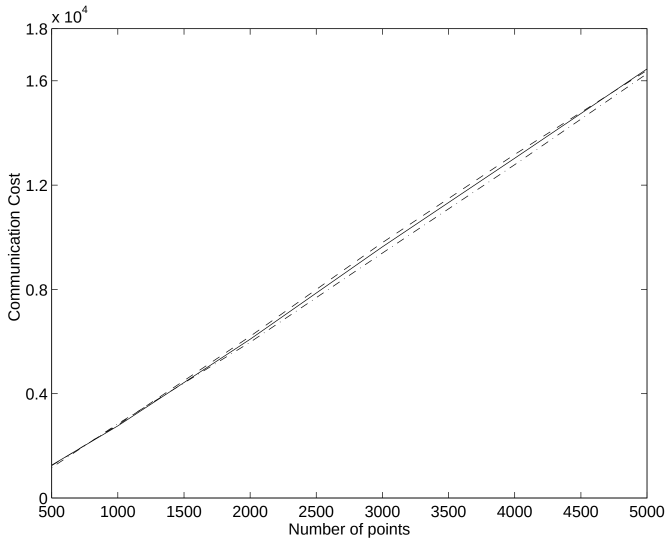

**Fig. 6.** Comparison of the quality of the partitions, as judged by the communication cost incurred for the $\psi$ distribution. The solid line presents the results for our algorithm. The dashed line shows the same for the inertial method. The dashed-dotted line shows the results for the multilevel spectral method with Kernighan-Lin refinement.

**Experiment 3**: Our final comparison is with the distribution given by: $\cos(\phi) \cdot \beta(\theta, 6, 2)$ where the $\beta$ distribution is similar to the *beta* distribution, with the range scaled to $[0, 2\pi]$. The value of $h$ was taken to be 0.3. This is non-uniform in both the latitude and in the longitude. It should be noted that most of the points are concentrated near the equator of the sphere due to the $\cos(\phi)$ term. Hence, we can expect that the latitude cuts in our algorithm will not be particularly effective and most of the cuts will need to be longitudinal. Thus, this tests our algorithm under situations in which it is likely to be less effective. However, the results presented in Figure 7 show that our algorithm performs almost as well as the other algorithms in this case, though it is slightly worse than the multilevel spectral method.

The experiments conducted show that all the methods considered here give partitions with good load balance. It also appears that the communication cost incurred are comparable. However, the scheme we have presented is much faster. This is probably due to the fact that our method has been specifically designed for points on the surface of the sphere, whereas the other methods are much more general.

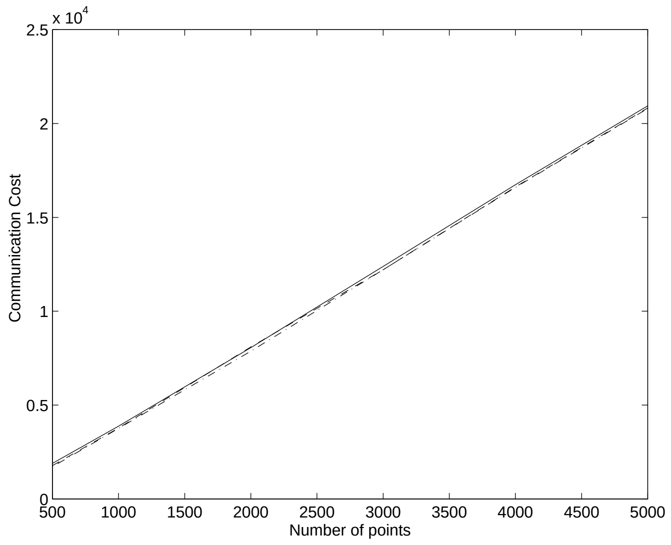

**Fig. 7.** Comparison of the quality of the partitions, as judged by the communication cost incurred, for the distribution proportional to $\cos(\phi) \cdot \beta(\theta, 6, 2)$. The solid line presents the results for our algorithm. The dashed line shows the same for the inertial method. The dashed-dotted line shows the results for the multilevel spectral method with Kernighan-Lin refinement.

## 5 Conclusions

We have presented a geometric domain decomposition algorithm that partitions data on the surface of the sphere and gives partitions suitable for particle method applications. Experiments have shown that the quality of the partitions obtained are comparable to more sophisticated schemes. The method has the advantage of being extremely fast, even compared with the inertial method. This good performance is to be expected since the graph edges in our problem are a function of geometric locality, and therefore suitable for geometric algorithms. Furthermore, high storage costs associated with graph algorithms for dense graphs (which is typical for our applications) are avoided.

The method produces partitions such that other operations on the data can be implemented by fast algorithms [7]. The domains produced by the algorithm can be parameterized well using four coordinates. This property is useful for other operations, which are described in [7]. The source code of the implementation can be obtained from the authors.

## References

1. Barnes, J., Hut, P.: A hierarchical $O(N \log N)$ force-calculation algorithm. Nature **3** (1986) 446-449
2. Berger, M.J., Bokhari, S.H.: A partitioning strategy for nonuniform problems on multiprocessors. IEEE Transactions on Computers **C-36** (1987) 570–80
3. Young, W.S., Brooks III, C.L.: Implementation of a Data Parallel, Logical Domain Decomposition Method for Interparticle Interaction in Molecular Dynamics of Structured Molecular Fluids. Journal of Computational Chemistry **15** (1994) 44-53
4. Monaghan, J.J.: Particle Methods for Hydrodynamics. Computer Physics Reports **3** (1985) 71-124
5. Harlow, F.H.: The Particle-in-Cell Computing Method for Fluid Dynamics. Meth. Comput. Phys. **3** (1964) 319-343
6. Eğecioğlu Ö., Srinivasan, A.: Efficient Nonparametric Estimation of Probability Density Functions. Technical Report TRCS95-21, University of California at Santa Barbara, 1995
7. Eğecioğlu Ö., Srinivasan, A.: Parallelization of Particle Methods on the Sphere, Technical Report TRCS96-10, University of California at Santa Barbara, 1996
8. Leighton, F.T.: Introduction to parallel algorithms and architectures : arrays, trees, hypercubes. M. Kaufmann Publishers, San Mateo, California, 1992
9. Hendrickson, B., Leland, R.: The Chaco User's Guide, Version 2.0. SAND95-2344, Sandia National Laboratories
10. Hendrickson, B., Leland, R.: A Multilevel Algorithm for Partitioning Graphs. SAND93-1301, Sandia National Laboratories
11. Karypis, G., Kumar, V.: METIS, Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0. Dept. of Computer Science, University of Minnesota, 1995
12. Williams, R.D.: Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. Concurrency **3** (1991) 457-481
13. Kernighan, B.W., Lin S.: An Efficient Heuristic Procedure for Partitioning Graphs. Bell System Technical Journal, 1970
14. Chaubal, C., Leal, L.G.: The Effect of Flow Type on the Rheology of Liquid Crystalline Polymers. Society of Rheology, 67th Annual Meeting, Sacramento, CA Oct 8-12, 1995
15. Plimpton, S.: Fast Parallel Algorithms for Short-Range Molecular Dynamics. J. Comp. Phys. **117** (1995) 1–19
16. 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, San Francisco, CA, USA, 15-17 Feb. 1995. Edited by: Bailey, D.H.; Bjorstad, P.E.; Gilbert, J.R.; Mascagni, M.V.; and others. Philadelphia, PA: SIAM, 1995, 627–32
17. Szeri, A., Leal, L.G.: A new computational method for the solution of flow problems of microstructured fluids. Part 2. Inhomogeneous shear flow of a suspension. Journal of Fluid Mechanics **262** (1994) 171–204

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