Adaptive Runtime Support for Direct Simulation Monte Carlo Methods on Distributed Memory Architectures *

Bongki Moon Joel Saltz

Institute for Advanced Computer Studies and
Department of Computer Science
University of Maryland
College Park, MD 20742
{bkmoon, saltz}@cs.umd.edu

Abstract

In highly adaptive irregular problems such as many Particle-In-Cell (PIC) codes and Direct Simulation Monte Carlo (DSMC) codes, data access patterns may vary from time step to time step. This fluctuation may hinder efficient utilization of distributed memory parallel computers because of the resulting overhead for data redistribution and dynamic load balancing. To efficiently parallelize such adaptive irregular problems on distributed memory parallel computers, several issues such as effective methods for domain partitioning and fast data transportation must be addressed. This paper presents efficient runtime support methods for such problems. A simple one-dimensional domain partitioning method is implemented and compared with unstructured mesh partitioners such as recursive coordinate bisection and recursive inertial bisection. A remapping decision policy has been investigated for dynamic load balancing on 3-dimensional DSMC codes. Performance results are presented.

1 Introduction

In sparse and unstructured problems, patterns of data access cannot be predicted until runtime. Since this prevents compile time optimization, effective use of distributed memory parallel computers may be achieved by utilizing a variety of profitable pre-processing strategies at runtime. In this class of problems, once data access patterns are known, the pre-processing strategies can exploit the knowledge to partition work, to map global and local data structures and to schedule the movement of data between the processor memories. The goal of the partitioning is to balance the computational load and reduce the net communication volume. Once data and work have been partitioned among processors, the prior knowledge of data access patterns makes it possible to build up a communication schedule that depicts which data elements need to be exchanged between processors. This communication schedule remains unchanged as long as the data access patterns remain unchanged.

In past work a PARTI runtime support library has been developed for a class of irregular but relatively static problems in which data access patterns do not change during computation. [14, 4, 5] To parallelize such problems, the PARTI runtime primitives coordinate interprocessor data movement, manage the storage of, and access to, copies of off-processor data, and partition work and data structures.

*This work was supported by NASA under contract No. NAG-11560, by ONR under contract No. SC 292-1-22913 and by ARPA under contract No. NAG-11485. The authors assume all responsibility for the contents of the paper.

PARTI primitives have been successfully used to port a variety of unstructured mesh single- and multi-grid codes, molecular dynamics codes onto distributed memory architectures. For these problems it is enough to perform pre-processing only once or occasionally.

In many irregular problems the data access patterns and computational load change frequently during computation. While the PARTI runtime primitives are able to handle many highly adaptive problems in which data access patterns change frequently, efficiency may be hindered by relatively high pre-processing costs. Thus the runtime primitives for interprocessor data movement and storage management need to be optimized for problems with varying data access patterns. Since the computational load may change from time step to time step, data arrays may need to be redistributed frequently to achieve load balance. This requires efficient methods to compute partitions of problem domain and to carry out remapping to balance load.

We have recently developed a new runtime support library called CHAOS which aims at parallelizing highly adaptive problems on distributed memory parallel computers. The runtime support library has been implemented on the Intel iPSC/860, Touchstone Delta and Paragon, Thinking Machine CM-5, and IBM SP-1 platforms. It subsumes the previous PARTI library which was mainly targeted at static irregular problems. The CHAOS library has been used to parallelize several highly adaptive application codes including two and three dimensional *Direct Simulation Monte Carlo (DSMC)* methods. This paper describes approaches for parallelizing the direct particle simulations on distributed memory parallel machines, emphasizing efficient data migration strategies, domain partitioning and dynamic remapping methods for load balance.

The rest of the paper is organized as follows. Section 2 describes general characteristics of DSMC methods and motivations behind these parallelization approaches. Section 3 introduces a new domain partitioner and compares it with other unstructured mesh partitioners. Periodic and dynamic remapping methods for dynamic load balancing issues are presented in Section 4. Communication optimization for fast and frequent data migration is discussed in Section 5. Experimental results are given in Section 6.

2 Motivations for parallelization

2.1 Overview of the DSMC method

The DSMC method is a technique for computer modeling of a real gas by a large number of simulated particles. It includes movement and collision handling of simulated particles on a spatial flow field domain overlaid by a Cartesian mesh [12, 18]. The spatial location of each particle is associated with a Cartesian mesh cell. Each mesh cell typically contains multiple particles. Physical quantities such as velocity components, rotational energy and position coordinates are associated with each particle, and modified with time as the particles are concurrently followed through representative collisions and boundary interactions in simulated physical space.

The DSMC method is similar to Particle-in-cell (PIC) method in that it tries to simulate the physics of flows directly through Lagrangian movement and particle interactions. However, the DSMC method has a unique feature that distinguishes it from PIC method: the movement and collision processes are completely uncoupled over a time step [13]. McDonald and Dagum have compared implementations of direct particle simulation on SIMD and MIMD architectures. [7]

2.2 Computational characteristics

Changes in position coordinates may cause the particles to move from current cells to new cells according to their new position coordinates. This implies that the cost of transmitting particles among cells may be significant on distributed memory parallel computers since a substantial number of particles migrate in each time step, and each particle usually has several words of data associated with it. In the particular 3-dimensional DSMC program reported here, each particle consumes about 10 words of storage. On average, more than 30 percent of the particles change their cell locations every time step. However, particle movements are very local. In our simulations, we observed that particles only move between neighboring cells.

Applications which have characteristics like the DSMC computation require runtime support that provides efficient data transportation mechanisms for particle movement. Computational requirement tends to depend on the number of particles. Particle movement therefore may lead to variation in work load distribution among processors. The problem domain overlaid by a Cartesian mesh needs to be partitioned in such a way that work load is balanced and the number of particles that move across processors is minimized. It may also need to be repartitioned frequently in order to rebalance the work load during the computation. These characteristics raise an issue of *dynamic load balancing* and require

- effective domain partitioning methods, and
- adaptive policy for domain repartitioning decisions.

3 Domain partitioning methods

This section presents some techniques for domain partitioning. It is assumed that work units (e.g. cells) in the problem domain may require different amounts of computation. Partitioning of the problem domain is important for parallel computation because efficient utilization of distributed memory parallel computers may be affected by work load distribution and the amount of communication between processors.

3.1 Recursive bisection

There have been several theoretical and experimental discussions of partitioning strategies based on spatial information for many years. *Recursive coordinate bisection (RCB)* [1] is a well-known algorithm which bisections a problem domain into two pieces of equal work load recursively until the number of subdomains is equal to the number of processors. *Recursive inertial bisection (RIB)* [10] is similar to RCB in that it bisects a problem domain recursively based on spatial information, but RIB uses minimum moment of inertia when it selects bisectioning directions, whereas RCB selects bisectioning directions from x-, y-, or z-dimensions. In other words, RIB bisects a 3- dimensional domain with a plane at any angle with respect to the coordinate axes, whereas RCB does so only with a plane perpendicular to one of the x, y and z coordinate axes. Clearly the number of processors has to be a power of 2 to apply these algorithms on parallel computers.

Recursive bisection algorithms produce partitions of reasonable quality for static irregular problems, with relatively low overhead when compared with *Recursive spectral bisection* [11] and *Simulated Annealing* [17]. von Hanxleden [16] and Williams [17] discuss the qualities of partitions produced by the recursive bisection algorithms, and compare their performance with other partitioning methods in several aspects.

3.2 Chain partitioner

Minimization of partitioning overheads is particularly important in problems that require frequent repartitioning. We therefore considered low overhead partitioning methods, especially *chain partitioners*. Chain partitioners decompose a chain-structured problem domain whose task graph is a chain of work units into a set of pieces which satisfies *contiguity constraints* – each processor has a contiguous subdomain of work units assigned to it. That is, a problem domain has to be partitioned in such a way that work units i and $i+1$ are assigned to the same or to adjacent processors. Relatively simple algorithms for finding the optimal partition of a chain-structured problem have been suggested [2, 8, 3].

While these algorithms are developed to optimize computation and communication costs at the same time, we have developed and used a new chain partitioning algorithm which considers computation cost only. This algorithm requires only one step of global communication and a few steps of local computation. Supposing that P is the number of processors, each processor i $(0 \leq i \leq P-1)$ executes the same algorithm :

1. Compute $S = \sum_{k=0}^{i-1} L_k$ and $T = iB$ where L_k is the amount of work owned by processor k and $B = \frac{1}{P}\sum_{k=0}^{P-1} L_k$. Then S is a scan sum that indicates the relative position of the subchain owned by processor i within the current distribution of work. T indicates the relative position of the subchain that processor i has to own under the target distribution of work.

2. If S is less than or equal to T, then find a processor index j such that $j = \max\{j \mid j \leq i$ and $T-S \leq (i-j)B\}$. If S is greater than T, then find a processor index j such that $j = \min\{j \mid j \geq i$ and $S-T < (j-i+1)B\}$. Then j is the index of the processor that has the smallest index number among processors some of work units of processor i must be moved into. Note that if $S = T$ then $j = i$.

3. Compute processor j's amount of work δ that will be moved from processors with index numbers less than i. If S is less than or equal to T, $\delta = (i-j)B-(T-S)$, otherwise $\delta = (S-T)-(j-i)B$.

4. Compute m the number of processors to which work of processor i will be moved. m is an integer such that $\frac{L_i+\delta}{B} \leq m < \frac{L_i+\delta}{B} + 1$.

5. Compute λ_k $(0 \leq k \leq m-1)$ the amount of work that has to be transferred to processor $j+k$; $\lambda_0 = B-\delta, \lambda_1 = \ldots = \lambda_{m-2} = B, \lambda_{m-1} = L_i - (m-1)B + \delta$.

6. Using the $\lambda_0, \lambda_1, \ldots, \lambda_{m-1}$, compute a list of processor indices to which each work unit of processor i has to be moved.

In applying our chain partitioner to 3- dimensional problems that have directional biases in communication requirements, we make two assumptions. First, communication costs may be ignored except along one direction with the most communication, so that 3- dimensional Cartesian mesh cells can be viewed as a chain of cells. Second, it is also assumed that communication costs between any pair of cells are all the same. Then it is not necessary to take the communication costs into account. These assumptions can be supported by the highly directional nature of particle flow that characterizes some DSMC communication patterns. In some of our tests, more than 70 percent of particles moved along the positive x-axis, and the standard deviation of the numbers of particles which moved across cell boundaries along the positive x-axis per each cell was less than 2.

3.3 Performance of domain partitioners

We have experimented with the above partitioning methods for a NASA Langley 3-dimensional DSMC code which simulates a corner flow on Intel iPSC/860 parallel computer with 128 processing nodes. When we carried out 1000 time steps of DSMC computation using the chain partitioner, the average number of messages sent by each processor was reduced by about 20 percent, while average volume of communication was increased by about 32 percent, compared with the use of recursive coordinate bisection. We measured message traffic required only by DSMC computation and communication incurred by partitioners themselves were not considered. Since a few long messages are more desirable than

(Time in secs)	Partitioning algorithms		
N_{procs}	RCB	RIB	Chain
8	0.4804	0.5662	0.0058
16	0.6531	0.6527	0.0035
32	0.9993	0.9652	0.0029
64	2.3708	2.2050	0.0030
128	7.1666	6.5897	0.0036

Table 1: Costs of partitioners on iPSC/860

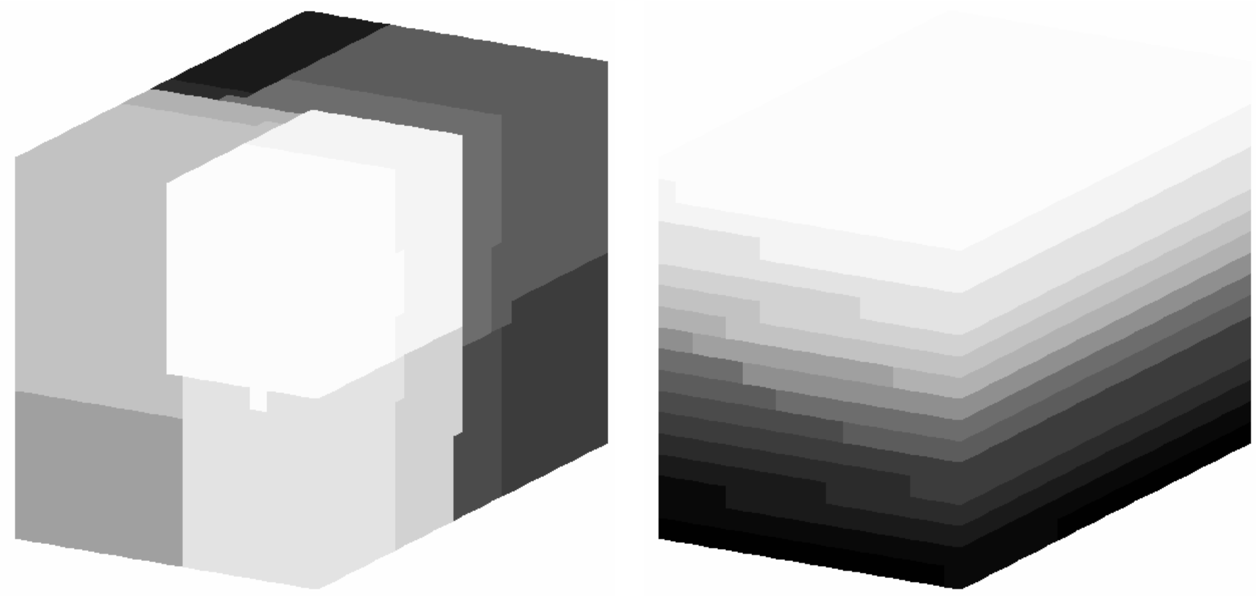


Figure 1: Domain partitions produced by RCB (left) and Chain partitioner (right)

many short messages on most distributed memory architectures, the chain partitioner, in some cases, may actually produce partitions with less communication overhead. Figure 1 illustrates the domain partitions of the 3- dimensional DSMC code. The first partition was produced by recursive coordinate bisection (RCB) and the second one was done by the chain partitioner.

Table 1 compares the execution time of recursive bisections and chain partitioner which have been implemented and benchmarked with the NASA Langley 3- dimensional $30 \times 18 \times 18$ Cartesian mesh cells on the Intel iPSC/860. It can be observed that the overheads associated with recursive bisections increase with the number of processors in a non-linear manner so that partitioning overheads that are acceptable on a relatively small number of processors may no longer be acceptable on a very large number of processors. This can be explained by the fact that our parallel implementations of recursive bisection algorithms require multiple phases of communication to exchange work load information among processors whereas the chain partitioning algorithm needs only one step of global communication that can be efficiently implemented by most message passing parallel computers.

4 Dynamic load balancing

Dynamic load balancing is essential for applications such as DSMC codes and PIC codes where work load distribution fluctuates as particles move around. This section outlines two remapping decision policies which determine when a problem domain needs to be repartitioned : a periodic remapping policy and a dynamic remapping policy. The periodic remapping repartitions the problem domain in a fixed interval preset by programmers and the dynamic remapping determines whether repartitioning is required at every time step. A heuristic has been studied and manipulated to help make remapping decisions dynamically at runtime. A related work that achieves dynamic load balance by monitoring load imbalance at user specified interval has been reported [6].

4.1 Periodic remapping

DSMC codes can be characterized by statistical calculations involving particles associated with each cell, particles moved to new cells as a result of calculations, and cells partitioned over the processing nodes. Each cell may require a different amount of computation and the work load distribution over the entire Cartesian mesh may change from time step to time step as particles move across cell boundaries. The combination of these characteristics may lead to system performance which deteriorates over time if the initial partition of problem domain remains unchanged statically.

We have optimized performance of 3- dimensional DSMC codes with periodic redistribution of computational load by using recursive bisections (RCB and RIB) and the chain partitioner described in Section 3. Periodic remapping, with a properly chosen inter-remapping frequency, provides significantly better performance compared with static partitioning. Experiments with the NASA Langley 3-dimensional DSMC code on Intel iPSC/860 with 128 processors show that the degree of load imbalance [1] does not exceed 30 percent of perfect load balance, whereas static partitioning exceeds 400 percent.

4.2 Dynamic remapping

Since the periodic remapping method repartitions Cartesian mesh cells at intervals of a fixed number of steps, it is insensitive to variations in computational requirements. In addition, the periodic remapping method requires potentially impractical pre-runtime analysis to determine an optimal periodicity.

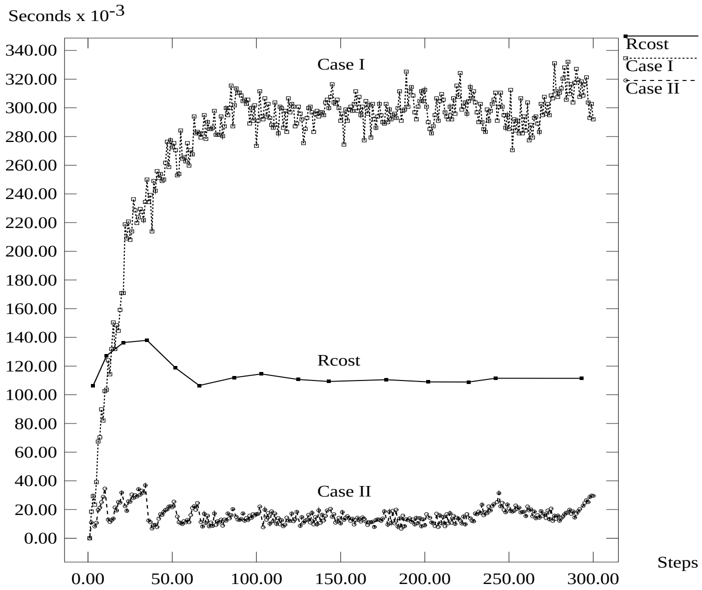
Stop-At-Rise (SAR) remapping decision policy has been implemented and experimented, which was introduced previously [9]. The SAR heuristic trades the cost of problem remapping against time wasted due to load imbalance. It assumes that processors synchronize globally at every time step, that the cost of remapping and idle time for each processor since the last remapping are known at runtime. Supposing that the problem domain was last remapped n steps ago, SAR defines a heuristic function

$$W(n) = \frac{\sum_{j=1}^{n}(T_{max}(j) - T_{avg}(j)) + C}{n}$$

to formulate the average processor idle time per step that could be achieved by remapping immediately at the current step. In the above heuristic function, $T_{max}(j)$ is the computation time required by a bottleneck processor to complete the jth step, $T_{avg}(j)$ is the average computation time required to complete the jth step, and C is the time spent in one remapping operation.

The heuristic function $W(n)$ incorporates two cost terms to determine whether remapping the problem domain is needed; C is the cost of a single remapping operation and $\sum_{j=1}^{n}(T_{max}(j) - T_{avg}(j))$

[1] The degree of load imbalance is defined as the ratio of the computation time of the bottleneck processor to the average computation time. That is, if t_i is the computation time of processor i, $t_{max} = max(t_i)$, and $t_{avg} = \sum_{i=1}^{P} t_i/P$ where P is the number of processors, then the degree of load imbalance is $L = t_{max}/t_{avg}$.



Rcost : Time step and cost of remapping
Case I : Idle time without remapping
Case II : Idle time with dynamic remapping

Figure 2: Remapping cost and Idle time

is the idle time cost of not remapping. There is an initial tendency for $W(n)$ to decrease as n increases because the remapping cost C is amortized over an increasing number of time steps. Increase in the cost term $\sum_{j=1}^{n}(T_{max}(j) - T_{avg}(j))/n$ indicates that work load balance is deteriorating. The SAR policy is to remap when $W(n)$ first begins to rise, i.e. the first time that $W(n) > W(n-1)$.

Figure 2 presents idle time incurred during each time step of DSMC simulation. `Case I` and `Case II` curves illustrate the idle time of a 3-dimensional DSMC code without and with remapping, respectively. That is, results of the `Case II` were obtained by repartitioning the problem domain dynamically with chain partitioner, whereas those of the `Case I` were generated by keeping the partition static during the entire computation process. Both experiments were carried out on a 16-node Intel iPSC/860 parallel computer. Each data point on these curves represents idle time wasted during each time step. The `Rcost` curve presents time steps when the problem domain is repartitioned and depicts the amount of time spent in repartitioning. For instance, the 8th data point in the `Rcost` curve shows that the Cartesian mesh cells of the DSMC code was repartitioned at the completion of the 103rd time step and about 114 milliseconds was spent for repartitioning. The idle time wasted in the 104th time step was reduced from about 292 milliseconds in the `Case I` to about 8 milliseconds in the `Case II`. This demonstrates that a substantial amount of idle time can be reduced by dynamic remapping with the SAR decision policy.

5 Communication optimization

The cost of transmitting particles between processors tends to be significant in the DSMC codes because a substantial number of particles change their cell locations in each time step, and each particle usually has several words of data associated with it. However, a particle's identity is completely determined by its state information (spatial position, momentum, etc), and computation depends only on a particle's

Table 2: Inspector/executor vs. Light-weight schedule (iPSC/860)

(Time in secs)	48x48 Cells				96x96 Cells			
	Processors				Processors			
	16	32	64	128	16	32	64	128
Inspector/executor	63.74	50.50	79.58	95.50	226.89	131.99	125.64	118.89
Light-weight schedule	20.14	11.54	7.60	6.77	79.89	40.46	21.77	14.23

state information and on the cell to which the particle belongs, and not on the particular numbering scheme for the particles. This implies that communication only need append particle state information to lists associated with each cell. This avoids the overhead involved in specifying locations within destination processors to which particle state information must be moved.

This property of the DSMC computation helps build *light-weight schedules* which are cheaper to compute than regular communication schedules built by PARTI primitives. A light-weight schedule for processor p stores the numbers and sizes of inbound and outbound messages, and a data structure that specifies to where the local particles of p must be moved. A set of data migration primitives has also been developed which can perform irregular communications efficiently using the light-weight communication schedules. While the cost of building light-weight schedules is much less than that of regular schedules, light-weight schedules and data migration primitives still provide communication optimizations by aggregation and vectorization of messages.

6 Experimental results

This section presents the performance results for the 2-dimensional and the 3-dimensional DSMC codes implemented on various distributed memory architectures including Intel iPSC/860, Intel Paragon and IBM SP-1.

6.1 Light-weight schedules

Table 2 presents the elapsed times of 2-dimensional DSMC codes parallelized using light-weight communication schedules and data migration primitives compared with the results from the same code parallelized by using regular schedules of PARTI runtime support. Since computational requirements are uniformly distributed over the whole domain of the 2- dimensional DSMC problem space, we have used a regular block partitioning method for the 2- dimensional DSMC code reported here. Thus load balance is not an issue for this problem and we can study the effectiveness of the light-weight schedule and data migration primitives without interference from other aspects such as partitioning methods and remapping frequencies. The *inspector/executor* method of PARTI runtime library [15, 4] which is applied to the 2-dimensional problem carries out pre-processing of communication patterns every time step because the reference patterns to off-processor data change from time step to time step. Consequently preprocessing cost is greater for the inspector/executor method than using light-weight schedules. Moreover, since the fraction of local particles to the initial distribution becomes smaller as the computation proceeds, the communication volume tends to grow. The performance of the inspector/executor method degenerates on a large number of processing nodes, and it actually leads to a decrease in performance when a large number of processors are used on a small problem.

Even though data migration primitives that use light-weight schedules are invoked in every time step for the purpose of particle relocation, it still outperforms the inspector/executor method significantly. This performance benefit is achieved by the aggregated communication which is automatically carried

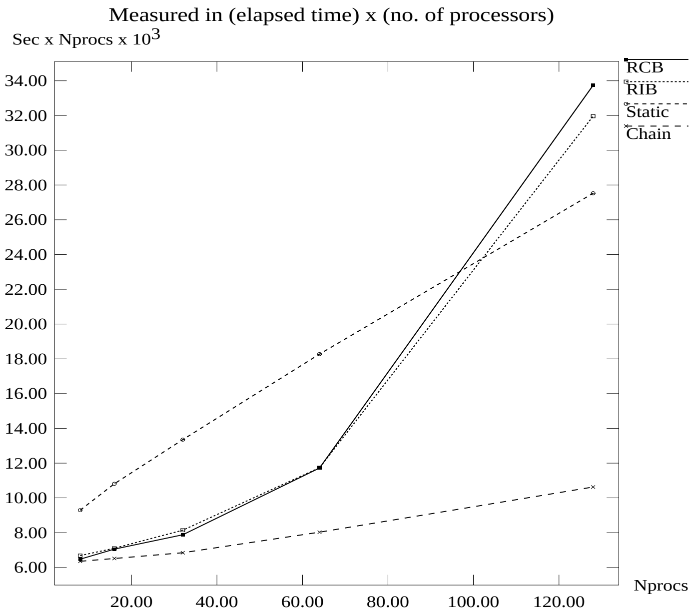


Figure 3: Performance of periodic remapping

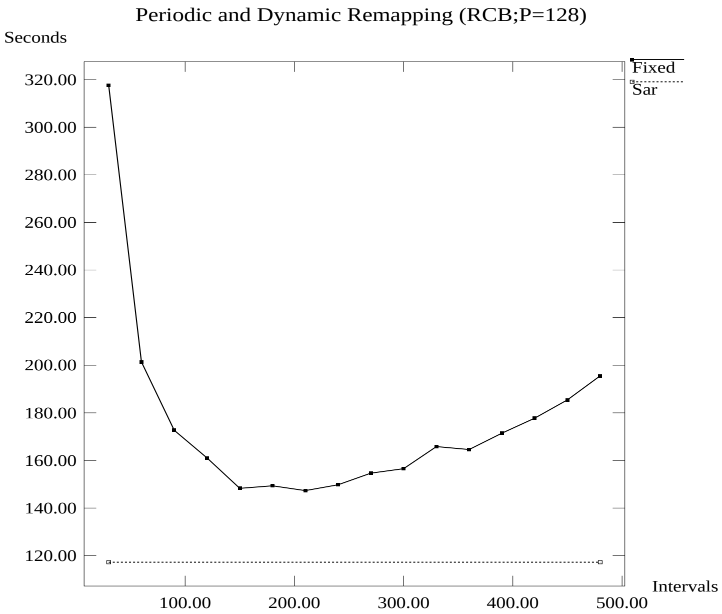
out by the migration primitives, which incurs minimum communication costs. Each processor aggregates information of all the moved particles into a single message, and sends and receives at most one message to and from its neighboring processors. More importantly, the migration primitives allow each processor to locally own all the particles required for its computation of each time step. Provided workload is optimally distributed, this feature guarantees the efficient utilization of processors as no further communication is required to execute remaining computation of the time step.

6.2 Periodic remapping

Figure 3 compares periodic domain remapping methods with static partitioning (i.e. no remapping), which are applied to 3-dimensional DSMC codes on Intel iPSC/860 with varying numbers of processors. We have measured performance numbers in a normalized manner by multiplying the elapsed time and the number of processors used. The `Static` curve represents the performance numbers produced by the static partitioning method. Curves `RCB`, `RIB` and `Chain` are produced by repartitioning the problem domain with recursive coordinate bisection, recursive inertial bisection and chain partitioner respectively. The problem domain is repartitioned every 40 time steps based on work load information collected for each Cartesian mesh cell. For an accurate workload estimate, actual computation time is measured in microseconds for each cell in every time step. Results indicate that all of the periodic repartitioning methods generate almost the same quality of partitions and significantly outperform static partitioning on a small number of processors.

Recursive bisection, however, leads to performance degradation when a large number of processors are used. For instance, the performance of these methods was poorer than that of static partitioning on iPSC/860 with 128 processors. This performance degradation is a result of large communication overhead which increases as the number of processors increases. The chain partitioner on the other hand appears to be very efficient in partitioning the problem domain with a minimum of additional overhead for this set of problems.

6.3 Dynamic remapping



Fixed : remapped periodically in varying intervals
Sar : remapped dynamically with SAR policy

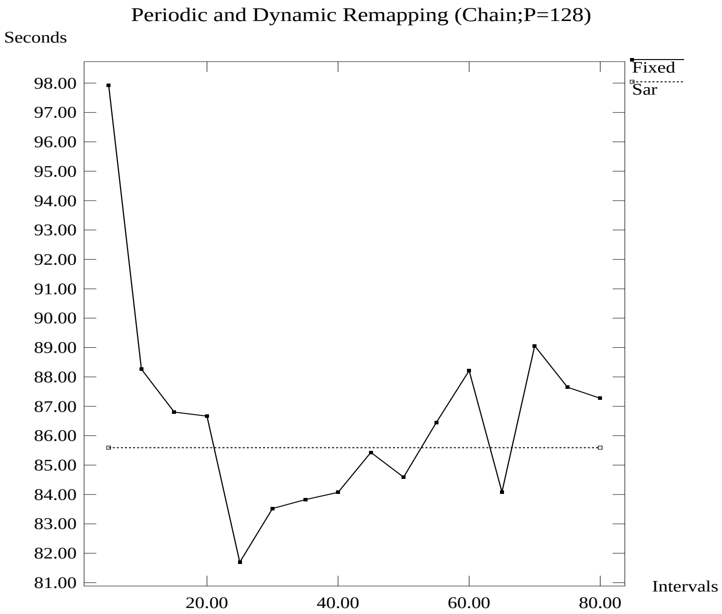
Figure 4: Dynamic remapping with RCB

The dynamic remapping method attempts to minimize the average processor idle time since the last remapping by trading remapping cost and idle time. Since the SAR remapping decision heuristic uses runtime information such as remapping cost and processing time of bottleneck processor, it adapts to the variations in system behavior without any a priori information about computational requirements and workload distribution.

This series of experiments with varying intervals for periodic remapping compared performance of optimal periodic remapping with that of dynamic remapping. Figure 4 and Figure 5 represent performance results measured in elapsed times when the problem domain is remapped by RCB and the chain partitioner respectively on Intel iPSC/860 with 128 processors. Note that the performance of dynamic remapping is better than that of periodic remapping with an optimal remapping interval when RCB is used to repartition the problem domain. When the chain partitioner is applied, dynamic remapping does not perform better than optimal periodic remapping; however, performance remains comparable to that of optimal periodic remapping.

In Figures 6 and 7, the performance of the dynamic remapping method is compared when carried out with each of three domain partitioning algorithms on various distributed memory parallel computers such as Intel iPSC/860, Intel Paragon and IBM SP-1. [2] Performance numbers are measured in a normalized manner by multiplying elapsed time by the number of processors used. In the Figure 7 the curves `RCB-P`, `RIB-P` and `Chain-P` represent performance of Intel Paragon, and `RCB-S`, `RIB-S` and `Chain-S` represent that of IBM SP-1. The chain partitioner outperformed other domain partitioning algorithms in all cases. Note that the chain partitioner can be applied to any number of processors, whereas RCB and RIB can be used only when the number of processors is a power of 2.

[2] Interrupt-driven EUI-H was selected for incoming messages on IBM SP-1.



Fixed : remapped periodically in varying intervals
Sar : remapped dynamically with SAR policy

Figure 5: Dynamic remapping with Chain partitioner

7 Concluding remarks

The results presented here indicate that data transportation procedures optimized for communication are crucial for irregular problems with fluctuating data access patterns, and effective domain partitioning methods and remapping decision policies can achieve dynamic load balancing.

Light-weight communication schedules reduced the costs of data transportation dramatically for applications where the numbering of data elements does not matter. A chain partitioner which partitions multidimensional problem domains in one-dimensional processor space was easy to implement. The chain partitioner also produced partitions of almost the same quality at very low cost for applications with directional biases in communicational requirements when compared with unstructured mesh partitioners such as RCB and RIB. A dynamic remapping method with the SAR policy carried out dynamic load balancing of DSMC codes without any a priori information about computational requirements on various distributed memory architectures.

DSMC codes described here employ a uniform mesh structure; however, for many applications it is necessary to use non-uniform meshes which, in some cases, adapt as computation progresses. Collaboration will continue with NASA Langley to address some of the many computational challenges posed by such non-uniform meshes.

Acknowledgements

The authors would like to thank Richard Wilmoth at NASA Langley for his constructive advice and the use of DSMC production codes. The authors thank Robert Martino at the National Institute of Health for support and use of NIH iPSC/860. The authors also gratefully acknowledge use of the Argonne High-Performance Computing Research Facility. The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

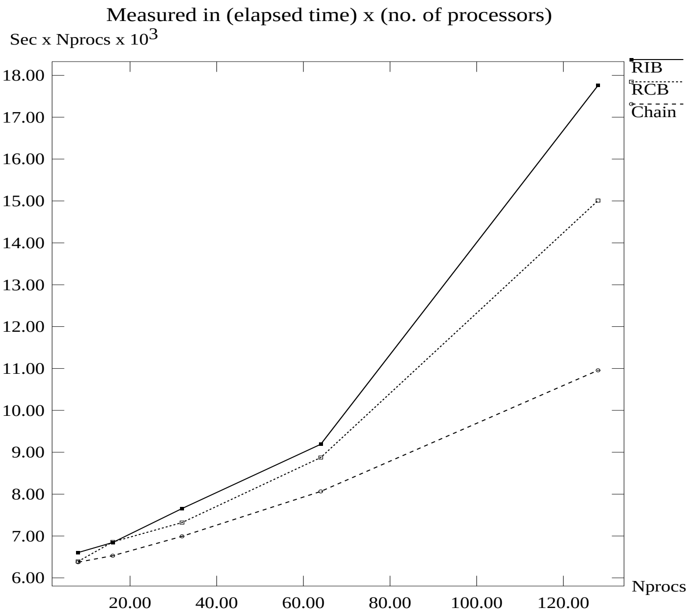


Figure 6: Dynamic remapping on Intel iPSC/860

References

[1] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. on Computers*, C-36(5):570–580, May 1987.

[2] S. H. Bokhari. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Trans. on Computers*, 37(1):48–57, January 1988.

[3] Hyeong-Ah Choi and B. Narahari. Efficient algorithms for mapping and partitioning a class of parallel computations. *Journal of Parallel and Distributed Computing*, 19:349–363, 1993.

[4] R. Das, D. J. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy. The design and implementation of a parallel unstructured Euler solver using software primitives, AIAA-92-0562. In *Proceedings of the 30th Aerospace Sciences Meeting*, January 1992.

[5] R. Das and J. Saltz. Parallelizing molecular dynamics codes using the Parti software primitives. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 187–192. SIAM, March 1993.

[6] P. C. Liewer, E. W. Leaver, V. K. Decyk, and J. M. Dawson. Dynamic load balancing in a concurrent plasma PIC code on the JPL/Caltech Mark III hypercube. In *Proceedings of the Fifth Distributed Memory Computing Conference, Vol. II*, pages 939–942. IEEE Computer Society Press, April 1990.

[7] J. McDonald and L. Dagum. A comparison of particle simulation implementations on two different parallel architectures. In *Proceedings of the Sixth Distributed Memory Computing Conference*, pages 413–419. IEEE Computer Society Press, April 1991.

[8] David M. Nicol and David R. O'Hallaron. Improved algorithms for mapping pipelined and parallel computations. *IEEE Trans. on Computers*, 40(3):295–306, March 1991.

[9] David M. Nicol and Joel H. Saltz. Dynamic remapping of parallel computations with varying resource demands. *IEEE Trans. on Computers*, 37(9):1073–1087, September 1988.

[10] B. Nour-Omid, A. Raefsky, and G. Lyzenga. Solving finite element equations on concurrent computers. In *Proc. of Symposium on Parallel Computations and their Impact on Mechanics*, Boston, December 1987.

[11] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430–452, June 1990.

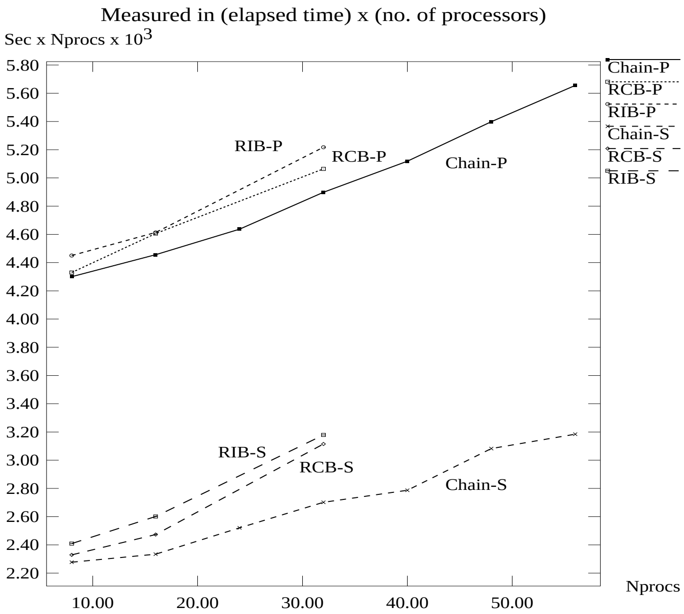


Figure 7: Dynamic remapping on SP-1 and Paragon

[12] D. F. G. Rault and M. S. Woronowicz. Spacecraft contamination investigation by direct simulation Monte Carlo - contamination on UARS/HALOE. In *Proceedings AIAA 31th Aerospace Sciences Meeting and Exhibit, Reno, Nevada*, January 1993.

[13] Patrick J. Roache. *Computational Fluid Dynamics.* Hermosa Publishers, Albuquerque, N.M., 1972.

[14] Joel Saltz, Harry Berryman, and Janet Wu. Multiprocessors and run-time compilation. *Concurrency: Practice and Experience*, 3(6):573–592, December 1991.

[15] Joel Saltz, Kathleen Crowley, Ravi Mirchandaney, and Harry Berryman. Run-time scheduling and execution of loops on message passing machines. *Journal of Parallel and Distributed Computing*, 8(4):303–312, April 1990.

[16] Reinhard v. Hanxleden and L. Riggway Scott. Load balancing on message passing architectures. *Journal of Parallel and Distributed Computing*, 13(3):312–324, November 1991.

[17] R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency, Practice and Experience*, 3(5):457–481, October 1991.

[18] M. S. Woronowicz and D. F. G. Rault. On predicting contamination levels of HALOE optics aboard UARS using direct simulation Monte Carlo. In *Proceedings AIAA 28th Thermophysics Conference, Orlando, Florida*, June 1993.