Data Locality on the Alewife Machine in the Barnes-Hut N-body Application

by

Carolos Livadas

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements
for the degree of

Bachelor of Science in Computer Science and Engineering

at the

Massachusetts Institute of Technology

September 1993

Author_____

Department of Electrical Engineering and Computer Science
Aug. 6, 1993

Certified by_____

Professor Anant Agarwal
Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by_____

Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

Data Locality on the Alewife Machine in the Barnes-Hut N-body Application

by

Carolos Livadas

Submitted to the Department of Electrical Engineering and Computer Science
on Aug. 6, 1993, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Abstract

This thesis investigates the impact of data locality on performance in the Barnes-Hut application. A program is said to exhibit data locality if the data references made by each processor are satisfied in local memory. Two questions regarding data locality are addressed in the context of this application. The primary question is how much can performance be improved by exploiting data locality? This is quantified by comparing the running times of two schemes: one in which data placement is optimal and another in which data placement is random. The optimal scheme reduces communication by servicing most of the cache misses through local memory. Specifically, for a 16 processor, 256 body simulation, the optimal scheme outperforms it's random counterpart by 6.8 %. Next, this thesis addresses the question of how much overhead is accumulated by distributing the data. Distributed data structures are more complex than centralized data structures and thus introduce additional overhead. This thesis shows that the reduction in communication contention through data distribution more than compensates for the overhead that is introduced by the distributed data structures.

Thesis Supervisor: Anant Agarwal
Title: Associate Professor Of Computer Science

Acknowledgments

I would like to thank my thesis supervisor Anant Agarwal for the help and the advice he has given me. Also special thanks to Donald Yeung for all the effort and the time he spent with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increasing need for high-performance computers and the physical constraints of single-processor machines have led to the development of highly-parallel multiprocessors. These usually involve a collection of nodes, each containing a processor and some memory, that communicate over an interconnection network. On such distributed memory architectures, communication costs increase with node distance. For this reason, this class of machines has been referred to as NUMA (Non-Uniform Memory Access) architectures. The NUMA nature of these machines introduces the concept of *locality*. That is, a distribution of data in which each processor requires progressively less information progressively less frequently from nodes which are progressively further away is desirable.

Many applications which naturally lend themselves to the exploitation of locality have been used to study the performance of NUMA machines. One such class of problems is N-body problems. These problems compute the positions and the velocities of bodies under the influence of inter-body forces. Because of the spatial nature of the problem, locality is inherent in the algorithm itself.

Two types of locality can be exploited to manifest improved multiprocessor performance: *task locality* and *physical locality*. Task locality refers to the performance gain that is achieved by assigning processes with common data requirements to the same processor. This scheme reduces communication latencies by effective caching. Physical locality refers to the local storage of the data that is used most frequently by each processor. Frequent accesses of remote data locations are thus avoided.

It is important, though, to make sure that task and physical locality do not interfere with

load balancing. Tradeoffs between locality and load-balance are often critical in determining the optimal approach to effectively parallelizing an application. The study of such locality schemes together with their effects on load balancing is very important in the evaluation of distributed memory multiprocessors. This thesis investigates performance issues related to physical locality on the Alewife multiprocessor [2].

Chapter 2

The Barnes-Hut Hierarchical N-body Application

2.1 The Importance of Hierarchical N-body Systems

N-body simulations investigate the interaction of particles. A straight-forward approach to the problem would be to consider the interaction of every possible pair of particles within the system. This brute-force solution introduces an impractical $O(n^2)$ time complexity. The need to simulate large N-body systems has led to the development of hierarchical tree-based methods that reduce the complexity to $O(n \ln n)$ [3, 4] for general distributions, and $O(n)$ for uniform distributions [5]. The efficiency of such algorithms is based on approximating the effect of large groups of bodies far away from the evaluation point with collective bodies. This approximation reduces the number of body interactions dramatically and the performance is therefore optimized. Furthermore, the accuracy of such algorithms can be traded against simulation time and this is very important due to the large simulation times associated with the study of multiprocessor architecture performance.

2.2 Galactic Simulation using Barnes-Hut

The algorithm used to simulate the N-body gravitational system is the $O(n \ln n)$ Barnes-Hut algorithm. The simulation involves the evolution of N-bodies under gravitational forces. The bodies are simulated by collisionless point mass particles which exert gravitational forces on all other particles in the system. During each time step, the gravitational forces exerted

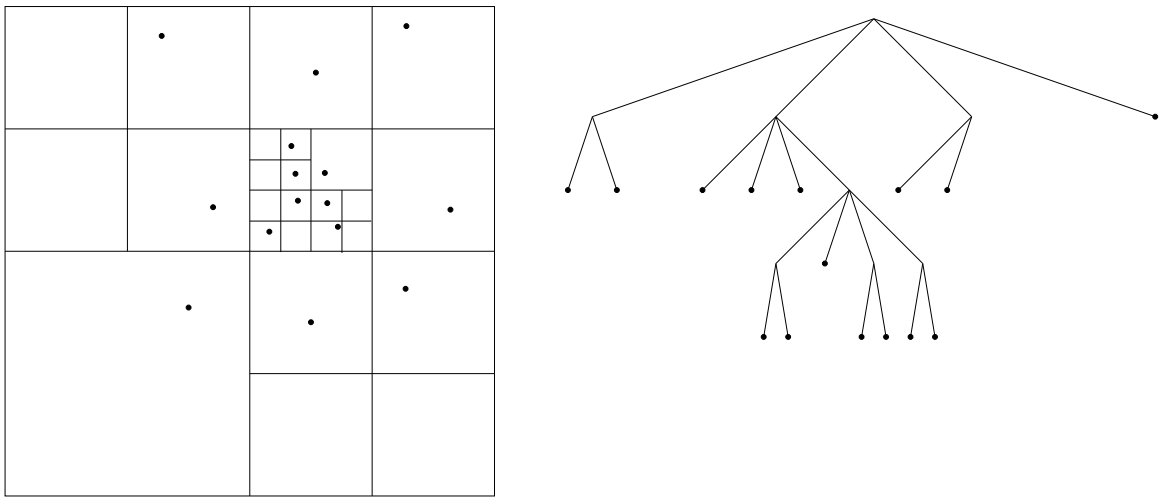

Figure 2-1: The tree of the Barnes-Hut algorithm and its spatial domain. The cell children are assigned quadrants of their parent-cell's space in a clockwise fashion. That is to say that the first child represents all bodies present in the top-left quadrant of its parent's region and the second, third, and forth children represent the top-right, bottom-right, and bottom-left quadrants, respectively.

on each of the particles is computed and the particle states are updated.

The Barnes-Hut algorithm is based, in the two dimensional case, on a hierarchical quadtree*. The inner nodes of the tree, usually referred to as *cells*, are collective bodies representing all bodies present in well-defined regions of space. The children of a cell are collective representatives of all particles present in the four quadrants of the region represented by their parent. The root of the tree is a cell containing all the particles in the system and the leaves of the tree comprise the individual particles. Furthermore, cells containing no particles are simply ignored. The tree is built by adding particles to the initially empty root cell and subdividing the cells into their four children as soon as they contain more than one particle. The tree thus extends to lower levels in regions of high particle concentration. Figure 2-1 provides an example of such a tree and its corresponding spatial domain.

Five phases comprise the Barnes-Hut algorithm:

(i) Computing the root cell dimensions of the tree and building the tree.

(ii) Computing the cell centers of mass by an upward pass through the tree.

*An octree is used in the three dimensional case.

(iii) Partitioning bodies among processors[†].

(iv) Computing the gravitational forces exerted on all particles by the whole ensemble.

(v) Updating the position, velocity, and acceleration properties of the particles.

The force calculation phase of the algorithm, which involves over 90% of the sequential execution time of the algorithm [6, 7], is carried out by traversing the tree once per body. Each force calculation is performed by recursively descending the tree and calculating the force exerted on the body by each of the cells. The force a cell exerts on the body is defined as being the force that the actual cell contributes, provided that the cell is far enough away from the body. If not, the vector sum of the forces exerted by the cell's children are applied to the body instead. The criterion as to what is considered "far enough" is the following condition:

$$\frac{l}{d} < \theta$$

where l is the length of the side of the cell region, d is the distance of the body from the center of mass of the cell, and θ is a user-defined parameter used to control the accuracy of the computed forces [4, 6, 7].

In a parallel implementation of the algorithm, each of the five algorithmic phases is computed in parallel. Even though parallelization across phases is not exploited explicitly, the removal of unnecessary synchronization barriers allows the overlapping of phases in a pipelined fashion. The parallelism exploited among all phases is across particles and cells. In the tree-building and the center-of-mass phases of the algorithm, inter-processor communication and synchronization is required since processors can modify the same parts of the tree. On the other hand, during the force-calculation phase, the synchronization of particles is not required since no information is modified. Nevertheless, communication is required in order to permit access to position and mass information of remote particles and cells throughout the domain of the system. Finally, the update-phase requires no synchronization but communication is unavoidable since all cached copies of the data being updated must be invalidated. Figure 2-2 presents the control flow of the Barnes-Hut algorithm.

[†]This phase applies only to the parallel version of the algorithm.

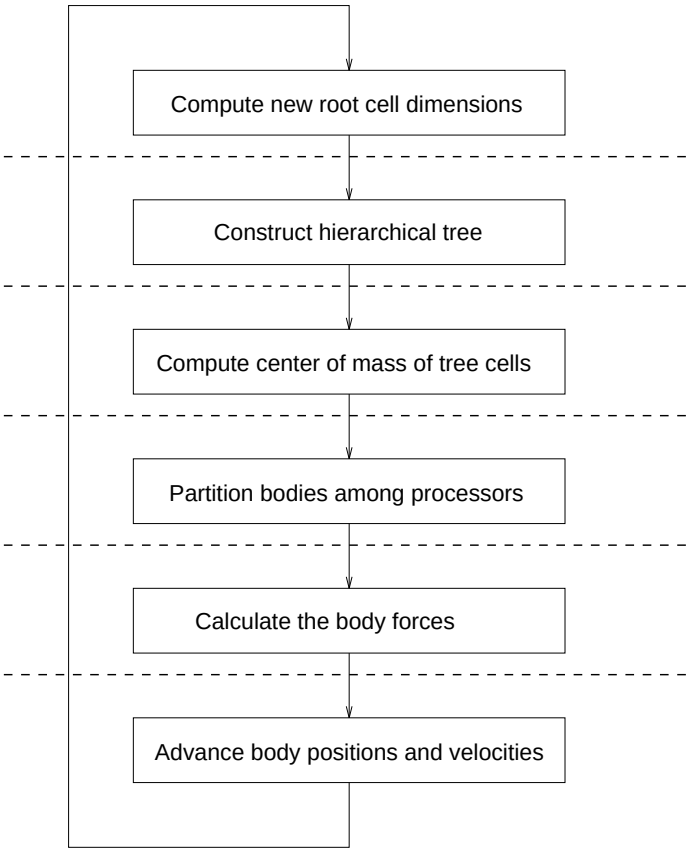

Figure 2-2: Control flow of the Barnes-Hut algorithm. The dashed lines signify barriers used for processor synchronization.

2.3 Previous Work

The force-calculation phase of the Barnes-Hut application involves the largest portion of computation [6, 7, 8]. Consequently, load balancing and communication issues specific to the force calculation phase are of primary interest.

2.3.1 Obtaining Load Balance and Data Locality

In [7], the following five different approaches to load balancing and data locality are compared:

Static Approach: Every processor is initially assigned an equal number of particles. These particles remain assigned to the same processor throughout all the simulation time steps.

Dynamic Approach (*loadbalance*): The processors are assigned different sets of particles during each time step in an effort to avoid load imbalances. This assignment is irrespective of the physical location of the particles.

Space-Locality: This approach involves an extension of the static scheme. Particles are evenly distributed among processors according to their physical locations. However, the initial particle-to-processor assignment is maintained throughout the simulation.

Costzones: This approach splits up the Barnes-Hut tree into zones of equal cost (loading) and assigns all particles of specific zones to the same processor. This scheme achieves partitions which are contiguous to the hierarchical tree and are of equal loading. Here, the partitioning is conducted during each iteration.

Orthogonal Recursive Bisection (ORB): This approach builds a hierarchical tree which ensures load balancing and optimal spatial locality. This is done by recursively dividing space into two subspaces of equal costs. The subdivisions are continued until there exists a one-to-one correspondence among the subspaces and the processors. Once more, the partitioning is conducted during each iteration.

The comparison of the above approaches is conducted using a distribution where particles are split in two Plummer model [1] clusters slightly offset from each other in each of the three dimensions. The simulations are performed assuming infinite per-processor caches. The use of infinite per-processor caches allows the investigation of task locality. Since data is not invalidated throughout the force calculation phase, data is fetched into the cache exactly once on a cold start miss and is then present in the cache for the entire time step. Therefore, the physical locality of the data itself, or rather the effect of physical locality is not an issue when assuming infinite per-processor caches. The comparison of the above algorithms is thus reduced to evaluating the tradeoffs between load balancing and task locality.

2.3.2 Costzones

Costzones is an extension of the Barnes-Hut algorithm that is geared towards effective load balancing and task locality during the force-calculation phase. Load balancing is obtained using a simple load profiling scheme. The work that is associated with each of the bodies

is profiled by the algorithm and is used as a measure for assigning bodies to processors; the ultimate goal being the even distribution of computational load. The work profiling is obtained by incrementing interaction counters during the force calculation phase of the simulation. The slow motion of the bodies ensures small variations in body loading between successive time steps and thus the loading of one time step is a realistic forecast of the body loading of the next time step. Task locality is attained by assigning bodies which are contiguous in the hierarchical tree to the same processor. The combination of these schemes for obtaining load balancing and task locality result in the partitioning of bodies into zones of equal loading and whose bodies are contiguous in the tree (refer to Figure 2-3). Subsequently, each of the processors gets assigned the bodies which are present in one of these zones.

In [7], it is argued that costzones does not provide optimal task locality. Bodies that are contiguous in the hierarchical tree are not guaranteed to be neighbors in real space. As a result, task locality is compromised. As true as this may be, it is shown that an algorithm that attains optimal task locality, namely orthogonal recursive bisection, proves to be too costly. The speedup obtained does not exceed the overhead involved in the tree-generation and the partitioning phases and thus less optimal algorithms, such as costzones, are preferred. Ultimately however, scaling effects may prove to reverse this phenomenon, but unfortunately current computational constraints limit the problem sizes that can be simulated.

2.4 Contributions of this Thesis

The contributions of this thesis lie in two domains. First, the thesis investigates the parallelization of one of the previously mentioned approaches to the Barnes-Hut N-body system, namely costzones, and secondly it studies the effects of data locality on performance when simulating an N-body application on the Alewife multiprocessor. A more detailed explanation of both follows in the next sections.

2.4.1 Costzones and Physical Locality

Because of the infinite per-processor cache assumption, the evaluation of costzones in [7] is only from the standpoint of task locality. However, physical locality is very important for

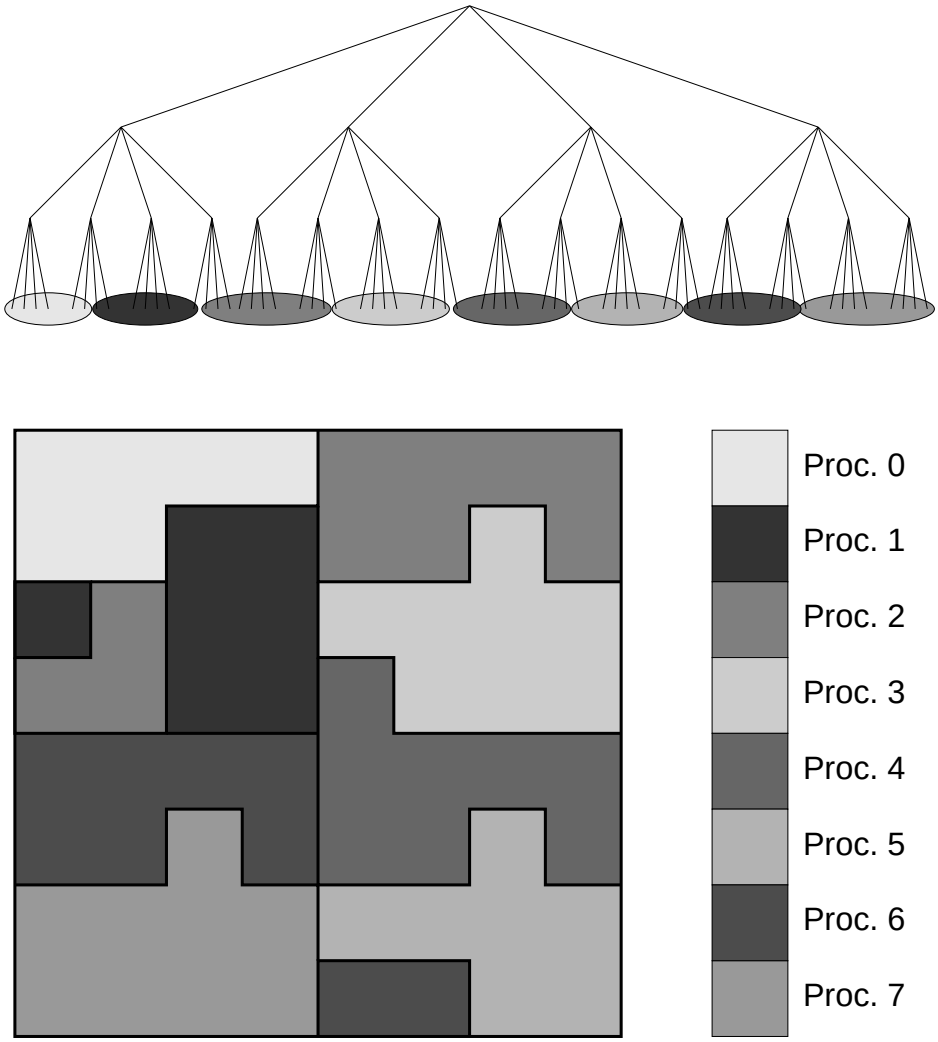

Figure 2-3: The spatial distribution of the costzones.

obtaining optimal performance. As the problem size increases, the amount of data required by each processor increases. Eventually, cache capacity is exceeded and thus data is multiply fetched from remote memory. These capacity cache misses increase in frequency with the system size and introduce significant communication overhead. This thesis investigates the speedup that is obtained from the exploitation of physical locality.

2.4.2 Overhead of Achieving Data Locality on the Alewife Multiprocessor

Although data locality on distributed memory multiprocessors is advantageous, one must always ensure the minimization of potential overhead. Such overhead may involve commu-

nication delays due to contention, data structure complications due to data distribution, and data referencing overhead. This thesis will investigate whether the implementation of physical locality introduces overhead that outweighs the benefits it provides.

Chapter 3

Porting the Barnes-Hut Code from SPLASH

3.1 Conversion to Semi-C

A major part of the work comprising the thesis involved porting a sequential implementation of the Barnes-Hut algorithm from the SPLASH benchmark to semi-C. Semi-C is a language very similar to C that can handle parallel language constructs and can be used with the NWO Alewife simulator. Because semi-C is a subset of C, many changes were required in order to get the sequential version of the code correctly ported. The problems involved during this process are discussed in the following sections.

3.1.1 Removing Pointer Arithmetic

Semi-C does not support pointer arithmetic. In order to avoid the use of pointers in ways that are not supported, many of the data structures of the Barnes-Hut code were altered. In the SPLASH code, bodies and cells are represented with structures which are allocated consecutively in memory and are accessed using pointer arithmetic. Since this scheme is not permissible in semi-C, arrays containing the pointers of the body and cell structures had to be created. Thus the cells and the bodies had to be referenced using indices to these arrays. Additional fields were added to the body and cell structures such that their respective indices could be stored. This is especially useful when the body and cell structures are independently passed as arguments to procedures.

3.1.2 Local Processor Variables

Another limiting feature of semi-C is the fact that it does not support static variables. In the SPLASH code, static variables provided each processor with its own variable environment. In semi-C this could either be done by declaring the variables local to one procedure and passing them by reference wherever needed, or by declaring a structure that would contain all the processor variables and passing this structure (or more precisely, a pointer to the structure) to the appropriate procedures. The second solution was chosen because of simplicity.

3.1.3 Distribution of Global Storage

The distribution of the data among the processors is important for two reasons. First, the memory of each processor of the Alewife Machine, and for all shared memory multiprocessors, comprises only a small fraction of the total memory available. Using the local-global memory of a single processor unnecessarily limits the size of problems that can ultimately be simulated. In the case of a gravitational system of bodies, the larger the number of particles, the better the simulation and consequently this constraint would be inappropriate. Secondly, in terms of the communication network, storing all the data on one processor introduces communication contention and unnecessary communication delays. Especially as the number of processors increases, these delays increase since all the processors access the local-global memory of one specific processor. The distribution of data alleviates contention and reduces communication distance when machine size is scaled.

In order to avoid the above problems, an equal distribution of data among processors is implemented. In the case of the body array, it is subdivided into equally sized subarrays each of which are allocated on different processors. In addition, a dope vector array is created on each of the processors. These locally stored dope vector arrays are used in order to reference the body subarrays distributed among the processors. Figure 3-1 shows how the body data array is distributed.

Since cells are allocated dynamically by each of the processors during the tree building phase, a similar subdivision scheme was not implemented for the cell array. Alternatively, a dope vector array for all the cells is stored on Processor 0 and its elements are set during the tree generation phase of the algorithm according to where each of its respective cells is

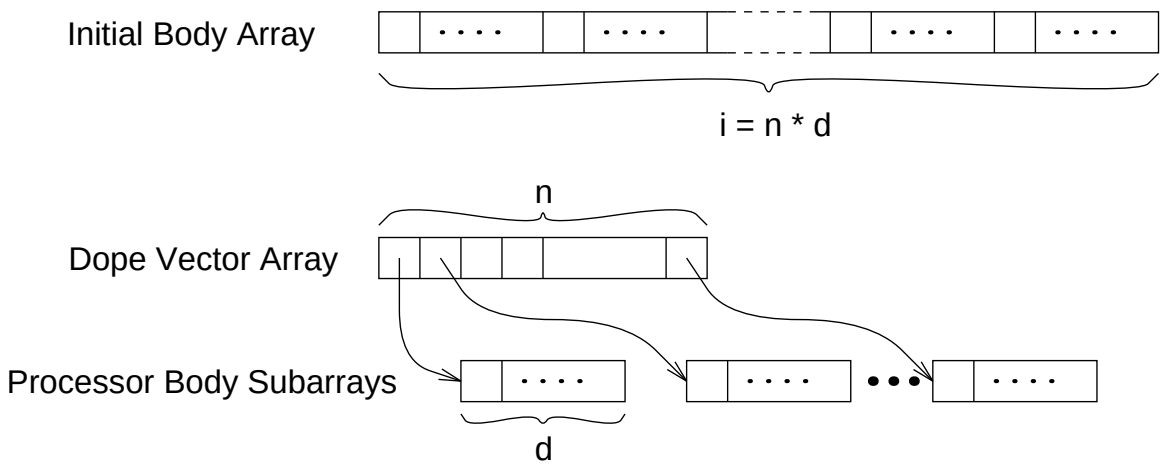

Figure 3-1: Distribution method of body arrays.

allocated.

3.1.4 Initial Placement of Data

As previously described, costzones is an algorithm which attempts to optimize data locality with small overhead in computation. However, it is not clear how the data is to be placed at the correct processor or even if it is worth migrating the data as the simulation progresses. An initial placement of data was implemented in order to determine the improvement in performance between an optimal and a random data distribution scheme. This is achieved by allocating backup body subarrays on each of the processors. Once the bodies are assigned to processors, the data of each of the bodies is simply copied into the backup subarrays and finally the dope vector arrays of each of the processors are reset to reference the backup instead of the initial subarrays. The tree is then rebuilt using the newly sorted body array. This is a valid placement only for the first iteration since all processors are guaranteed to be assigned an equal number of bodies. In later time steps, the number of bodies placed on each processor is not the same for all processors due to load balancing effects. A more complicated referencing scheme is required in this case. Currently, a body indexed by i is the $(i \bmod d)$th element of the $(i \operatorname{div} d)$th body subarray, where d is the number of bodies stored per processor. This condition is not necessarily true in an uneven distribution of the bodies.

3.1.5 Verification of Correct Execution

The correctness of the code was verified by comparing the obtained results with those attained by running the SPLASH benchmark code on a SPARC workstation. Intermediate values as well as the final body velocities and positions were printed and compared. Since semi-C does not support double precision floating numbers, precision was slightly relaxed. This resulted in small discrepancies between the values obtained by the NWO Alewife simulator and the SPARC workstation.

3.2 Guide to Present Barnes-Hut Code

The final version of the parallelized code follows the control flow as shown in Figure 2-2. Some configuration flags and initial parameters are stored in a file and are used during compilation to set the program specifications. These are described in the following two sections.

3.2.1 Configuration Flags

The code presently available is capable of providing a variety of simulation configurations. These can be set by the following compilation flags:

TWO-CLUSTER: indicates the simulation of a two cluster system. If this flag is not set, the simulation involves a single cluster system.

UNIFORM: indicates the simulation of a system comprised of uniformly distributed bodies within a sphere of fixed radius. If this flag is not set, the simulation involves a Plummer model distribution [1] *.

TWODIM: indicates the simulation of a two dimensional system.

THREEDIM: indicates the simulation of a three dimensional system.

QUADPOLE: indicates the use of quadrupole moments in the force calculation phase of the simulation.

REALLOCATE: indicates that the body data must be reallocated during the first step in order to attain an initially optimal data distribution. The reallocation assumes

*It is worth noting that among the TWO-CLUSTER and the UNIFORM configuration flags, four distinct configurations are possible.

that all processors are assigned an equal number of bodies which is true during the first iteration since an initially equal loading distribution is assumed.

OUTPUT: enables the output of potentially useful information such as a list of the cells comprising the tree during each time step and the processor-body assignments.

DEBUG: enables the output of information useful for debugging purposes.

WRITELN-ON: enables the tracing of the control flow by print statements informing the user when procedures are called and when these procedures return. This option is very useful for debugging purposes.

3.2.2 Specification Parameters

These are values which determine the N-body system size, and the accuracy, length and memory requirements of the simulation. They involve the following:

def_nbody: specifies the number of bodies to be simulated (default is 128).

def_seed: an integer seed for the random number generator used to determine initial conditions (default 123).

def_dtime: specifies the physical time between simulation steps (default is 0.025).

def_eps: a softening parameter used to avoid singularities when particles get too close to each other (default is 0.05).

def_tol: specifies the value of the accuracy parameter θ (default is 0.6).

def_fcells: specifies the amount of memory that should be allocated for cells (default is 0.8). The maximum number of cells is assumed to be $\mathbf{fcells} \cdot \mathbf{nbody}$, where nbody is the number of bodies involved in the simulation.

def_tstop: specifies the physical time at which to stop the simulation (default is 0.25). The number of time steps run is therefore $\frac{\text{def\_tstop}}{\text{def\_dtime}} + 1$, i.e. a default of 11 time steps.

def_NPROC: the number of processors to be used (default is 4).

Chapter 4

Experimental Framework

4.1 The Alewife Multiprocessor

The evaluation of data locality is performed using the NWO Alewife multiprocessor simulator. The Alewife multiprocessor is a shared-memory multiprocessor geared towards *latency minimization* and *latency tolerance*. The Alewife machine involves a set of processing nodes connected in a 2-D mesh topology. Each of the nodes consist of a 33 MHz RISC processor, a 64 k byte cache, a portion of globally-shared distributed memory, a Controller and Memory Management Unit (CMMU), a floating-point coprocessor, and the Caltech Mesh Routing Chip (MRC). Figure 4-1 shows the layout of the Alewife multiprocessor. Latency minimization is achieved by caching shared data. Caches reduce the effect of communication latencies and furthermore reduce the need for a careful initial placement of data. The caches are kept coherent by the *LimitLESS* cache-coherence protocol [2]. If the system cannot avoid a remote memory access, Alewife's processor can rapidly context switch to another task in place of the stalled process. Rapid context switching is used to hide the latency of both remote memory accesses and synchronization faults. Refer to [2] for the specifics of the architecture.

The goal of minimizing interprocessor communication emphasizes the importance of data locality for shared memory multiprocessor architectures, and especially for the Alewife machine. The study of data locality thus provides feedback as to the speedups attained by optimal placement of data and gives insight as to how to achieve optimal performance. In addition, the potential overhead due to data distribution and the cost suffered in order to ensure efficient scaling is determined.

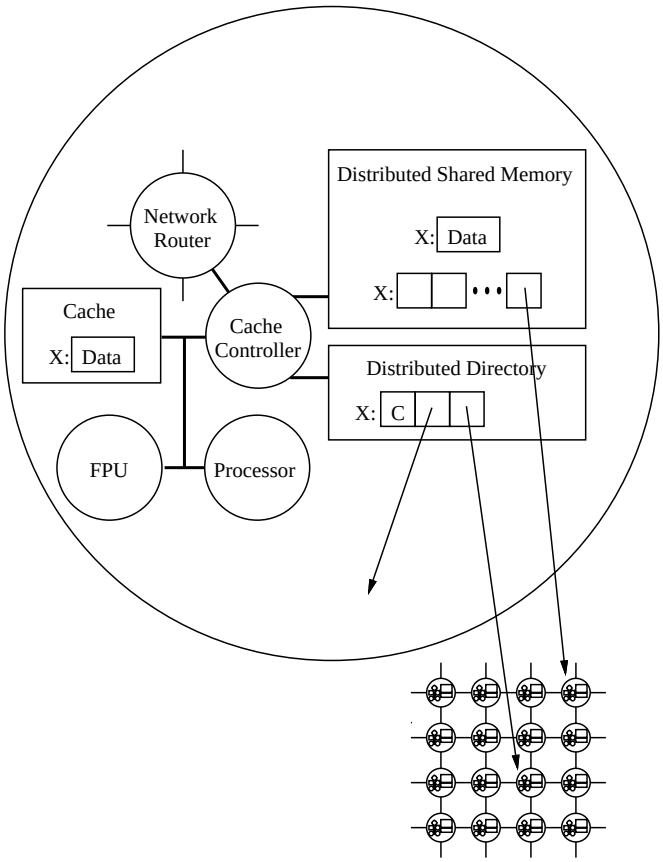

Figure 4-1: The structure of the Alewife multiprocessor, showing the LimitLESS directory extension.

4.2 Experimental Approach

The steps involved in porting the SPLASH code to a parallel semi-C version resulted in the generation of four different versions of the Barnes-Hut code:

- **Sequential version:** a one processor version of the code with all barriers and locks removed in order to avoid the synchronization overhead.
- **Parallel version:** a parallel version where all the top level variables are stored on one distinct processor, namely Processor 0.
- **Randomly distributed version:** a parallel version where the global array of bodies is evenly distributed among all processors. The distribution is done randomly and thus does not satisfy physical locality.

- **Optimally distributed version:** a distributed version where the body data is placed optimally during the first time step in order to attain physical locality. This scheme simulates an inherently optimal distribution of data among the processors and thus the computational overhead involved in placing the bodies at their optimal locations is not taken into account.

This thesis involves the comparison of the performance of these four versions of essentially the same code when run on the NWO Alewife simulator. The goal of the experiments is to determine how data locality affects the running time of an N-body application on a shared memory multiprocessor, specifically the Alewife machine. Due to time constraints, the test matrix consists of unrealistically small test cases. These involve two-dimensional, one-cluster, uniformly distributed systems comprising of 32, 64, 128, and 256 particles. The performance is obtained allowing machine size to vary between four, eight and sixteen processors. The simulations are run for two time steps. The first is an initialization step and the second is the data collection step. The first step is not used for data collection since the costzones are determined according to an initial assumption of even body loading. This is not necessarily true and thus load imbalance could lead to invalid data.

4.2.1 Distributed Memory and Data Locality

Distributed memory machines avoid the contention introduced by central memory schemes. However, the distribution of data among the processors involves protocol overhead which might actually affect performance. The experiments of this thesis will actually determine whether the distribution of data is associated with performance overhead. It is very important to keep in mind that this overhead might compromise performance but at the same time may be necessary due to central memory contention especially as the number of processors increases.

4.2.2 Sequential versus Parallel Approach

One of the most interesting questions when studying parallel architectures is the actual speedup that can be attained through parallelism. The speedup is defined as the ratio of the sequential to the parallel running times. Thus, for a four processor machine, the optimal speedup would be four. Unfortunately, the speedup of a multiprocessor is reduced due to

communication overhead between processors and synchronization effects. The investigation of the speedup and the causes of suboptimal performance is critical to understanding how to use the multiprocessor to achieve optimal performance. How data locality affects performance is thus of great importance. This question will be addressed by varying the number of processors as described earlier. Ideally, the larger the number of processors, the greater the speedup. Unfortunately though, communication overhead as well as synchronization effects may degrade performance.

Chapter 5

Experimental Results

The data that was gathered from the simulations involve the running times of individual phases of the algorithm and communication overhead information. A detailed description of the information gathered is presented in the next sections.

5.1 Running Times of the Different Algorithmic Phases

The running times of the different phases in a single time step were gathered. This was achieved by having each processor keep track of the time that it spent in the tree-building, partitioning, force-calculation and advancing phases of the algorithm. In order to get an overall performance criterion for each of the simulations, the running time required by each of the processors for the complete time step was also recorded. These running times in Alewife machine cycles are present in Table 5.1. Their respective speedups are present in Table 5.2.

Since the force calculation phase comprises the largest portion of the computation time, the speedups attained specifically to this phase of the simulation are critical. These speedups are present in Figure 5-1. It is important to notice that the optimal data distribution outperforms both the random distribution and the centralized scheme. It is expected that the performance gain due to spatial locality will actually increase considerably with system size. It is believed that the unrealistically small system sizes used in the context of this thesis retain effective caching. The larger the system sizes, the greater the expected cache misses and the larger the subsequent remote access delays. Thus, physical locality will be of more importance for larger system sizes.

Table 5.1: Running Times of the Different Algorithmic Phases

Scheme	# Proc	Tree Gen.	Part. Part.	Force Calc.	Advance	Total
32 Particle Simulation						
Sequential	1	65132	11535	617881	17303	793748
Centralized Data	4	47196	6159	153360	6616	258899
	8	97954	6004	130456	7263	291298
	16	379960	8886	150431	11600	685875
Random Data Distribution	4	48327	5770	171873	9713	287296
	8	95666	4858	98046	6574	263493
	16	276341	7002	64261	4900	493893
Optimal Data Distribution	4	50375	5468	167964	6052	272760
	8	85950	4042	89353	4991	245846
	16	275927	3947	58280	3766	454743
64 Particle Simulation						
Sequential	1	140797	23556	1620422	32732	1958553
Centralized Data	4	92867	11253	395028	12104	574481
	8	161811	9189	252135	7501	483427
	16	531892	12074	200919	17718	813766
Random Data Distribution	4	86836	10746	415105	15643	593142
	8	134725	8084	285682	10767	504427
	16	353548	9281	156270	7295	618158
Optimal Data Distribution	4	96474	10424	414575	12549	588924
	8	127893	7593	265710	8769	473576
	16	501131	6652	147005	6144	755922
128 Particle Simulation						
Sequential	1	286448	47485	4592104	62684	5242792
Centralized Data	4	170637	23502	1163667	22859	1477091
	8	262015	18385	678995	14462	1050900
	16	849506	18345	464374	28764	1466260
Random Data Distribution	4	169300	20566	1219625	32432	1558299
	8	259453	13247	653251	20784	1028268
	16	820463	10572	375578	17539	1305939
Optimal Data Distribution	4	179232	19161	1150897	24484	1466863
	8	278735	12026	635977	15067	1013764
	16	797354	8795	356517	10671	1252956
256 Particle Simulation						
Sequential	1	597313	96897	11777318	124072	13074019
Centralized Data	4	325417	43606	3007852	44363	3594940
	8	521385	30100	1548008	28676	2235415
	16	1853379	28557	1012834	33380	3019460
Random Data Distribution	4	316503	36145	2957108	54890	3552864
	8	478143	22562	1679049	35150	2339054
	16	1788052	14410	891980	28910	2835192
Optimal Data Distribution	4	328821	35903	2931048	44927	3494512
	8	488095	20599	1512709	27536	2155388
	16	1792128	12694	835523	15720	2762851

Table 5.2: Speedup Factors of the Different Algorithmic Phases

Scheme	# Proc	Tree Gen.	Part. Part.	Force Calc.	Advance	Total
32 Particle Simulation						
Sequential	1	1	1	1	1	1
Centralized Data	4	1.380	1.873	4.029	2.615	3.066
	8	0.665	1.921	4.736	2.382	2.725
	16	0.171	1.298	4.107	1.492	1.157
Random Data Distribution	4	0.832	1.999	3.595	1.781	2.763
	8	0.681	2.374	6.302	2.632	3.012
	16	0.236	1.647	9.615	3.531	1.607
Optimal Data Distribution	4	1.293	2.110	3.679	2.859	2.910
	8	0.758	2.854	6.915	3.467	3.229
	16	0.236	2.922	10.602	4.595	1.745
64 Particle Simulation						
Sequential	1	1	1	1	1	1
Centralized Data	4	1.516	2.093	4.102	2.704	3.409
	8	0.870	2.563	6.427	4.364	4.051
	16	0.265	1.951	8.065	1.847	2.407
Random Data Distribution	4	1.621	2.192	3.904	2.092	3.302
	8	1.045	2.914	5.672	3.040	3.883
	16	0.398	2.538	10.369	4.487	3.168
Optimal Data Distribution	4	1.459	2.260	3.909	2.608	3.326
	8	1.101	3.102	6.098	3.733	4.136
	16	0.281	3.541	11.023	5.327	2.591
128 Particle Simulation						
Sequential	1	1	1	1	1	1
Centralized Data	4	1.679	2.020	3.946	2.742	3.549
	8	1.093	2.583	6.763	4.334	4.989
	16	0.337	2.588	9.889	2.179	3.576
Random Data Distribution	4	1.692	2.309	3.765	1.933	3.364
	8	1.104	3.585	7.030	3.016	5.099
	16	0.349	4.492	12.227	3.574	4.015
Optimal Data Distribution	4	1.598	2.478	3.990	2.560	3.574
	8	1.028	3.949	7.221	4.160	5.172
	16	0.359	5.399	12.880	5.874	4.184
256 Particle Simulation						
Sequential	1	1	1	1	1	1
Centralized Data	4	1.836	3.222	3.916	2.797	3.637
	8	1.146	3.219	7.608	4.327	5.849
	16	0.322	3.393	11.628	3.717	4.330
Random Data Distribution	4	1.887	2.681	3.983	2.260	3.680
	8	1.249	4.295	7.014	3.530	5.589
	16	0.334	6.724	13.204	4.292	4.611
Optimal Data Distribution	4	1.817	2.699	4.018	2.762	3.741
	8	1.224	4.704	7.786	4.506	6.066
	16	0.333	7.663	14.096	7.893	4.732

It is also noteworthy that the centralized case performs relatively worse as the number of processors increases. This is due to the bottleneck that is generated as the increasing number of processors access data which are stored on a single processor. It is interesting to see that as the number of bodies increases, the severity of the bottleneck effect drops. This could be caused by the increase of the computation involved in the force computation phase and the drop in the ratio of communication and synchronization latencies to computation time.

The outperformance of the centralized scheme by the randomly distributed scheme indicates that the overhead of data distribution is not as significant as the communication overhead that it alleviates. Thus, the distribution of data is encouraged.

However, the promising speedups of the force calculation phase do not lead to equivalent speedups for the complete simulation time step. Figure 5-2 shows the speedups involving the complete time step. As the number of processors increases, the relative performance of the multiprocessor deteriorates. This was an unexpected result. It is interesting to see that the cause of this result is the overhead that is incurred during the tree generation phase of the algorithm. Figure 5-3 shows the overhead accumulated in the tree generation phase. This overhead constitutes the inverse of the speedup, i.e. the ratio of the multiprocessor running time to the sequential running time for the specific phase. The overhead increases dramatically as the number of processors increases. Since this part of the algorithm involves both processor synchronization and communication, it does not parallelize well. Specifically, the generation of the tree involves continuous invalidation of the cells of the tree. The tree is a data structure that is used by all the processors, so this phase of the computation is undergoing continuous communication between the processors. In addition, the cell array is not distributed across the processors and thus accessing cells is done through a single processor. This introduces communication contention. Moreover, as the processors increase the communication overhead increases. This is natural as more processors are accessing the centrally stored data. Nevertheless, the optimal data distribution again outperforms the other schemes. This is another indication of the importance of physical locality.

The results obtained introduce the question of whether the use of larger multiprocessors actually increases speedup. The data at hand actually shows that the 8 processor machine actually outperforms the 16 processor machine. However, one must keep in mind that the test cases considered involve unrealistically few particles. In an effort to spread some light

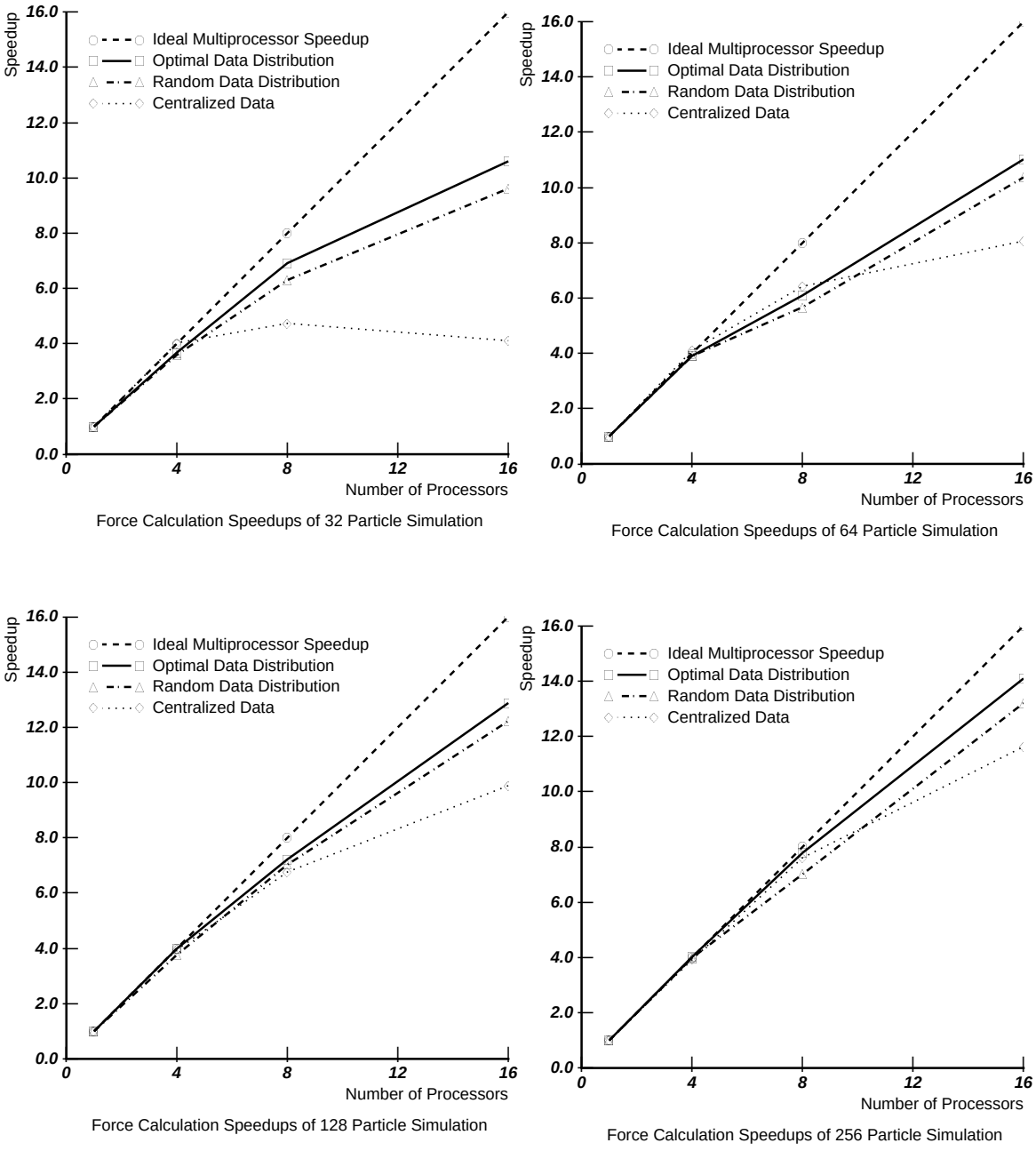

Figure 5-1: Speedups of force calculation phase.

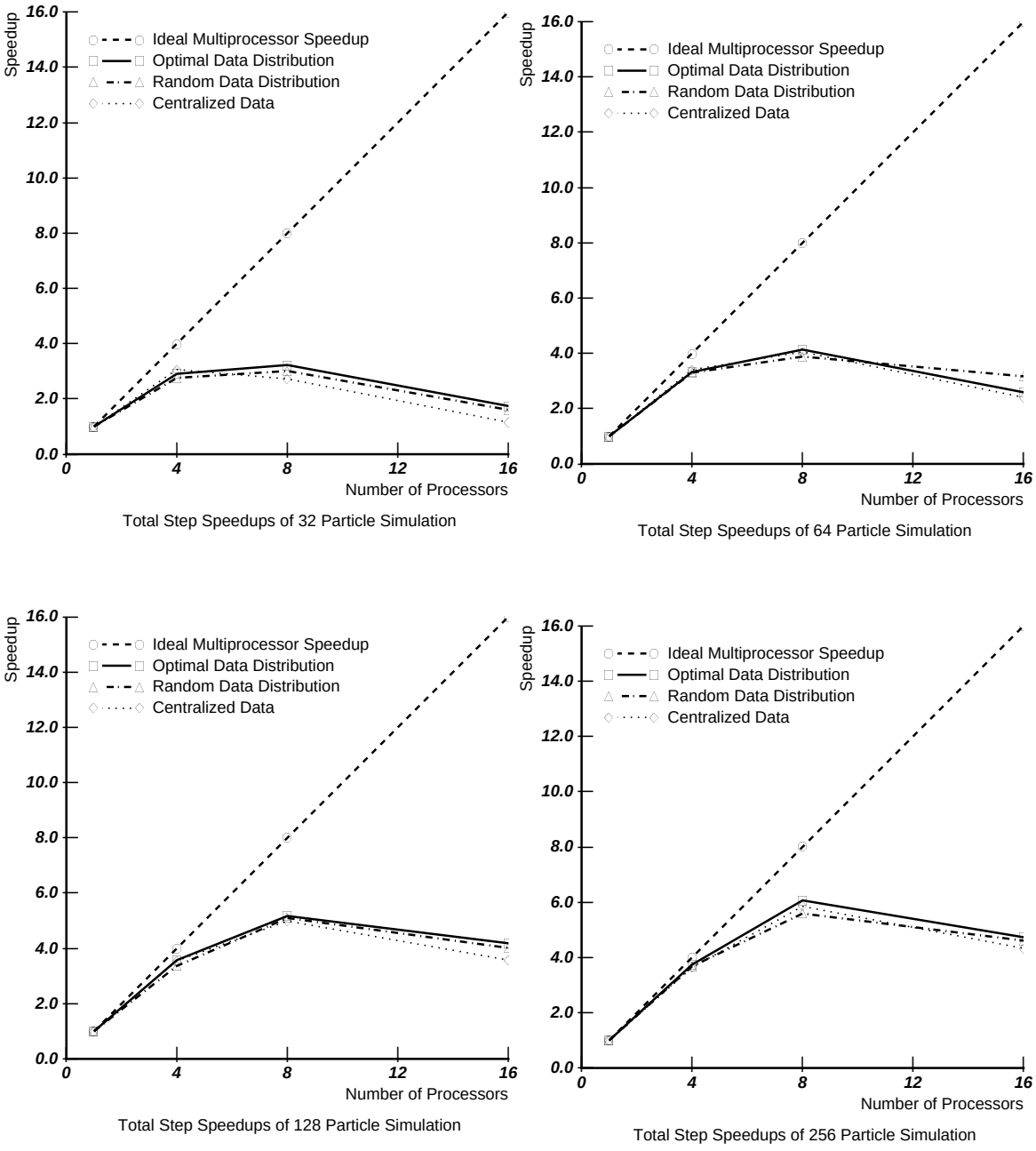

Figure 5-2: Speedups of complete simulation step.

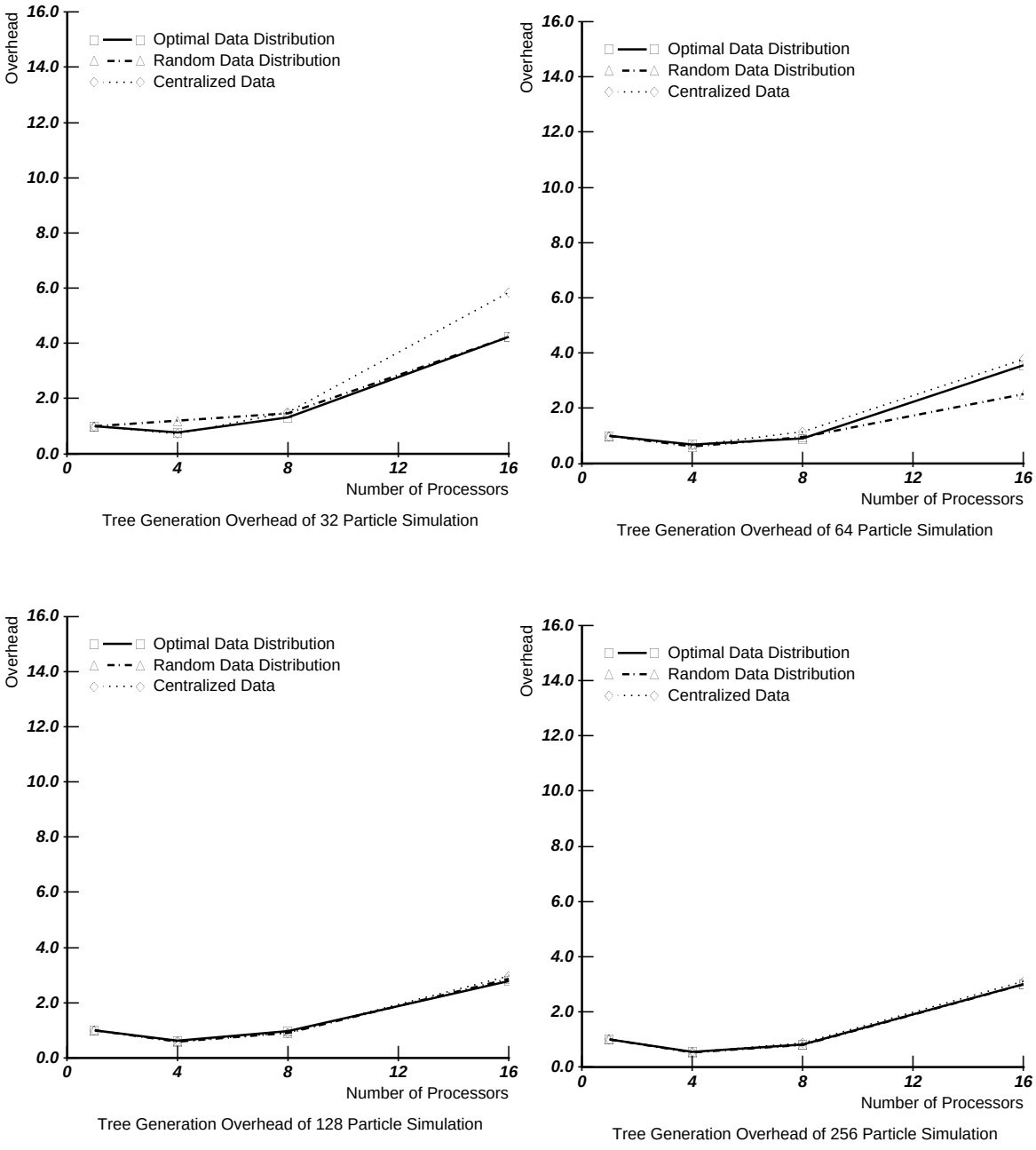

Figure 5-3: Overhead of tree building phase.

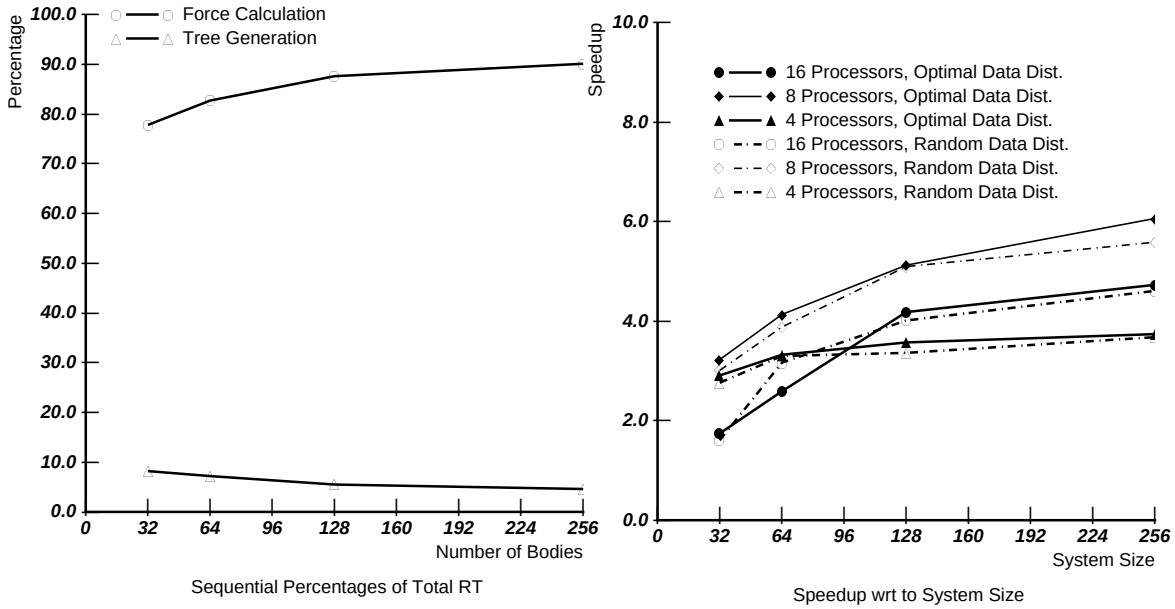

Figure 5-4: Effects of machine and problem scaling on speedup.

on this question, the percentage of sequential computation time for the force calculation and the tree generation time was plotted (refer to Figure 5-4). As the problem size increases, the force calculation time comprises an increasing portion of the total computation. In addition, with increasing problem size the force calculation speedup seems to increase whereas the overhead obtained by the tree generation time remains more or less the same. These facts indicate the tendency that as the number of particles increases, the speedup of the force calculation will increasingly outweigh the overhead of the tree generation phase. This tendency can be seen in the plot of speedup with respect to system size of Figure 5-4. The performance improves as the ratio of system size to number of processors increases. This suggests that with realistic problem sizes, the speedup of the entire time step will resemble the curves in Figure 5-1. However, test cases involving larger problem sets must be conducted in order to back up this conjecture.

5.2 Communication Overhead

The communication profiles pertaining to the complete time step were attained by the simulator. The complication in attaining the statistics involved the method by which to turn on and off the statistic gathering tools of the simulator from within the actual code.

This was achieved by incorporating some simhooks within the program that enabled the isolation of the statistics involving the second time step. The data gathered are present in Appendix A. Unfortunately, due to time constraints, this data was not analyzed.

Chapter 6

Conclusion

6.1 Summary

The results of the simulations indicate that physical locality improves the performance of the multiprocessor. Especially as the problem size increases the amount of speedup that is gained by optimally distributing the data is significant. This is especially promising since the system sizes that are to be ultimately simulated are very large and thus the speedup would amount to significant running time. Moreover, the superior performance of the random data distribution scheme over the centralized data storage scheme indicates the positive effect of data distribution. In addition, it is interesting to see that the speedup of the force calculation phase is accompanied by a large computational overhead due to the tree generation phase. However, this overhead is common across all schemes. That is to say that it is actually due to the inherent problem at hand. The tree generation phase is one which involves the synchronization of the processors and the communication due to updating the tree cells which are all accessed through one single processor.

The conclusion of this thesis is that physical locality introduces significant speedup. Specifically, the performance of a 16 processor machine simulating a 256 body system improves by 6.8 %. The exploitation of data locality reduces remote communication, takes advantage of local storage and thus should be attained wherever possible. It is also conjectured that the speedup due to physical locality will actually increase with problem size. Increasing data sets will result in increasing capacity cache misses and in additional overhead that can be minimized through effective data placement.

6.2 Future Work

Apart from the questions concerning the speedup that is attained due to data locality, many other interesting questions remain unanswered. Most of these questions involve the algorithm that is used to attain physical locality. Even though the speedup is evident, how expensive is it to attain and maintain physical locality throughout a complete simulation? Some of the questions that still need to be tackled are presented in the following sections.

6.2.1 Optimal Data Locality and Data Reallocation

The data attained in this thesis illustrate the performance of a system that inherently appeared to satisfy physical locality. However, in order to attain this optimal distribution of data, an inefficient sorting algorithm was used. Had the running time overhead of this algorithm been taken into account, the effects of physical locality may well have been reversed. The question that arises is how expensive is it to reallocate data such that it satisfies the spatial locality dictated by the application? In addition, the discovery of an efficient reallocation strategy may prove worthwhile. Such an algorithm may actually perform incremental reallocations during each time step and thus minimize overhead and retain optimal physical locality without effectively sorting the entire data set.

6.2.2 Degradation of Data Locality

In conjunction with the previous question, it would be very interesting to determine the rate at which data locality deteriorates. Due to the motion of the bodies, the physical locality is slowly distorted. The rate of such a distortion may introduce a time frame as to how often the data has to be reallocated such that the speedup attained by physical locality outweighs the overhead of the reallocation scheme. Unfortunately, this rate will depend on the actual dynamics of the system at hand and is thus potentially difficult to determine.

6.2.3 Scalability

Due to the unrealistic data sets that were used in this thesis, it may very well be the case that ultimate speedups due to physical locality were not accurately determined. Larger data sets could exceed the capacity of the cache sizes to a much greater extent. The subsequent frequent communication requirements may introduce greater speedup and thus physical

locality may prove to be critical for efficient execution. This could actually prove to be of greatest importance when dealing with larger multiprocessors where remote communication introduces greater overhead. Once again, one must keep in mind that the advantages of data locality must be traded against the cost of attaining and retaining it.

Appendix A

Communication Data

Each of the simulations produced specific communication statistics for the complete time step. These statistics are summarized in the following counters:

Index	Counter Name	# of
1	**STATS-N-LOCAL-LO**	accesses to system local memory
2	**STATS-N-LOCAL-HI**	accesses to user local memory
3	**STATS-N-LOCAL-GLOBAL**	accesses to local global memory
4	**STATS-N-REMOTE-GLOBAL**	accesses to remote global memory
5	**STATS-N-IFETCH-LOCAL-LO**	ifetches to system local memory
6	**STATS-N-IFETCH-LOCAL-HI**	ifetches to user local memory
7	**STATS-N-IFETCH-LOCAL-GLOBAL**	ifetches to local global memory
8	**STATS-N-IFETCH-REMOTE-GLOBAL**	ifetches to remote global memory
9	**STATS-N-BUSIES**	total busies
10	**STATS-N-CACHE-HIT**	cache hits
11	**STATS-N-TXNBUF-HIT**	transaction buffer hits
12	**STATS-N-LOCAL-NO-NET**	all local transactions, no network
13	**STATS-N-LOCAL-LO-NO-NET**	just lower local accesses to memory
14	**STATS-N-LOCAL-HI-NO-NET**	just upper local accesses to memory
15	**STATS-N-LOCAL-WITH-NET**	local transactions, with network
16	**STATS-N-REMOTE-MISS**	remote transactions
17	**STATS-N-CANT-FIND**	lost transactions (simulation artifact)
18	**STATS-N-LOCAL-FE-OPT**	full/empty optimizations for local
19	**STATS-N-LOCDLY-FE-OPT**	full/empty optimizations for local
20	**STATS-N-REMOTE-FE-OPT**	full/empty optimizations for remote

The values of these counters for each of the simulations are present in the tables that follow.

Table A.1: Communication Statistics I

Scheme	# Proc	1	2	3	4	5
32 Particle Simulation						
Sequential	1	33617	10	257356	0	0
Centralized Data	4	70039	1848	284305	72031	0
	8	114832	21226	328341	127294	0
	16	242539	61526	462196	295048	0
Random Data Distribution	4	74055	1984	290253	75563	0
	8	135490	8225	341844	134887	0
	16	286605	21743	477395	294573	0
Optimal Data Distribution	4	75454	2332	316285	71773	0
	8	128247	21899	373922	135420	0
	16	310022	28689	523164	319391	0
64 Particle Simulation						
Sequential	1	80347	10	634278	0	0
Centralized Data	4	126992	2412	623201	149417	0
	8	201661	10706	670035	230671	0
	16	415797	29452	852571	469801	0
Random Data Distribution	4	136034	2616	640255	157301	0
	8	227936	11307	699543	248678	0
	16	428413	25585	868391	450327	0
Optimal Data Distribution	4	138649	3162	698498	144282	0
	8	212509	34019	763890	242037	0
	16	474515	37825	960974	513514	0
128 Particle Simulation						
Sequential	1	204203	10	1617201	0	0
Centralized Data	4	277265	3902	1519485	350946	0
	8	400616	16734	1577018	495919	0
	16	743645	45212	1842873	861430	0
Random Data Distribution	4	294358	4248	1703983	350098	0
	8	448468	17805	1785784	524433	0
	16	796165	44021	2085572	882848	0
Optimal Data Distribution	4	299623	4706	1566200	306805	0
	8	427319	55638	1649349	500043	0
	16	869295	61283	1901535	951135	0
256 Particle Simulation						
Sequential	1	494410	10	3935266	0	0
Centralized Data	4	623321	6684	3639260	790679	0
	8	823338	29998	3687313	1072518	0
	16	1257014	223580	4104934	1702085	0
Random Data Distribution	4	653348	6452	3748988	765115	0
	8	942542	29125	3850898	1113865	0
	16	1605524	81637	4325512	1839230	0
Optimal Data Distribution	4	664402	7684	4055757	644448	0
	8	892239	95693	4157762	1021581	0
	16	1691090	113811	4676340	1878655	0

Table A.2: Communication Statistics II

Scheme	# Proc	6	7	8	9	10
32 Particle Simulation						
Sequential	1	0	0	0	0	278541
Centralized Data	4	0	0	0	161	385757
	8	0	0	0	8676	500264
	16	0	0	0	85317	838763
Random Data Distribution	4	0	0	0	186	390671
	8	0	0	0	9392	514822
	16	0	0	0	72634	841791
Optimal Data Distribution	4	0	0	0	225	416442
	8	0	0	0	10277	553529
	16	0	0	0	101918	925011
64 Particle Simulation						
Sequential	1	0	0	0	0	693552
Centralized Data	4	0	0	0	356	835352
	8	0	0	0	12861	983721
	16	0	0	0	118982	1454386
Random Data Distribution	4	0	0	0	282	851358
	8	0	0	0	12684	1015373
	16	0	0	0	90886	1433881
Optimal Data Distribution	4	0	0	0	365	902127
	8	0	0	0	16485	1082007
	16	0	0	0	155652	1608833
128 Particle Simulation						
Sequential	1	0	0	0	0	1762167
Centralized Data	4	0	0	0	397	2018358
	8	0	0	0	21348	2243598
	16	0	0	0	189741	2963098
Random Data Distribution	4	0	0	0	473	2030622
	8	0	0	0	23753	2322939
	16	0	0	0	183109	3013044
Optimal Data Distribution	4	0	0	0	593	2147818
	8	0	0	0	34281	2454278
	16	0	0	0	262573	3303735
256 Particle Simulation						
Sequential	1	0	0	0	0	14280943
Centralized Data	4	0	0	0	679	4766268
	8	0	0	0	40679	5154262
	16	0	0	0	354923	6341428
Random Data Distribution	4	0	0	0	706	4782971
	8	0	0	0	41810	5299216
	16	0	0	0	356171	6680740
Optimal Data Distribution	4	0	0	0	794	5016415
	8	0	0	0	62058	5551059
	16	0	0	0	527760	7106570

Table A.3: Communication Statistics III

Scheme	# Proc	11	12	13	14	15
32 Particle Simulation						
Sequential	1	546	11896	259	7	0
Centralized Data	4	723	16576	376	80	542
	8	693	20595	656	1273	1017
	16	864	25806	942	1670	1171
Random Data Distribution	4	1345	18917	445	63	509
	8	3093	24994	3944	169	1030
	16	2451	29238	3597	205	1351
Optimal Data Distribution	4	643	19022	396	63	693
	8	1807	24376	703	1247	1490
	16	1369	30138	2569	184	1987
64 Particle Simulation						
Sequential	1	1369	19714	288	7	0
Centralized Data	4	1330	27972	551	103	881
	8	1627	31901	2023	137	1517
	16	2411	39173	3042	204	1863
Random Data Distribution	4	2639	32822	506	59	761
	8	5664	42732	6370	212	1530
	16	4035	45105	4749	210	1504
Optimal Data Distribution	4	953	33834	620	83	1150
	8	3235	41637	893	1418	2226
	16	2241	47777	3794	208	2820
128 Particle Simulation						
Sequential	1	2889	56358	444	7	0
Centralized Data	4	2684	59786	714	75	1570
	8	5036	71439	3852	130	2610
	16	4223	75430	5982	205	3088
Random Data Distribution	4	10943	80837	9403	142	1492
	8	12037	84836	12313	218	2530
	16	13701	94183	15365	208	2750
Optimal Data Distribution	4	5154	75149	641	134	2130
	8	7883	84009	1678	2545	4114
	16	7647	93223	6107	211	4654
256 Particle Simulation						
Sequential	1	8440	140303	736	7	0
Centralized Data	4	7985	139083	927	61	2881
	8	6571	139255	4981	228	4892
	16	8252	153560	2283	8434	5802
Random Data Distribution	4	18421	182411	19074	182	2574
	8	20855	185112	21959	309	4751
	16	11905	176402	11502	289	5022
Optimal Data Distribution	4	13494	174533	932	75	3679
	8	13250	182505	2308	3497	7608
	16	12677	192897	9243	302	8126

Table A.4: Communication Statistics IV

Scheme	# Proc	16	17	18	19	20
32 Particle Simulation						
Sequential	1	0	0	0	0	0
Centralized	4	24615	0	0	0	0
Data	8	69124	0	0	0	0
	16	194705	0	0	0	0
Random	4	30413	0	0	0	0
Data	8	76507	0	0	0	0
Distribution	16	205485	0	0	0	0
Optimal	4	29044	0	0	0	0
Data	8	78286	0	0	0	0
Distribution	16	222761	0	0	0	0
64 Particle Simulation						
Sequential	1	0	0	0	0	0
Centralized	4	36487	0	0	0	0
Data	8	94307	0	0	0	0
	16	269788	0	0	0	0
Random	4	48626	0	0	0	0
Data	8	122165	0	0	0	0
Distribution	16	288191	0	0	0	0
Optimal	4	46527	0	0	0	0
Data	8	123350	0	0	0	0
Distribution	16	325157	0	0	0	0
128 Particle Simulation						
Sequential	1	0	0	0	0	0
Centralized	4	69200	0	0	0	0
Data	8	167604	0	0	0	0
	16	447321	0	0	0	2
Random	4	91010	0	0	0	0
Data	8	217713	0	0	0	0
Distribution	16	500891	0	0	0	0
Optimal	4	84866	0	0	0	0
Data	8	218500	0	0	0	0
Distribution	16	558026	0	0	0	0
256 Particle Simulation						
Sequential	1	0	0	0	0	0
Centralized	4	143727	0	0	0	0
Data	8	308187	0	0	0	0
	16	778571	0	0	0	0
Random	4	187526	0	0	0	0
Data	8	426496	0	0	0	2
Distribution	16	977834	0	0	0	1
Optimal	4	164170	0	0	0	0
Data	8	413005	0	0	0	0
Distribution	16	1039626	0	0	0	1

References

[1] S. J. Aarseth, M. Henon, and R.Wielen. *J. Astrophys. and Appl.*, 37:183 ff., 1974

[2] Anant Agarwal, David Chaiken, Kirk Johnson, David Kranz, John Kubiatowicz, Kiyoshi Kurihara, Beng-Hong Lim, Gino Maa, and Dan Nussbaum. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. Laboratory for Computer Science, MIT.

[3] Andrew A. Appel. An efficient program for many body simulation. In *AFIPS Conference Proceedings*, pages 483-485, April 1967.

[4] Josh Barnes and Piet Hut. A Hierarchical $O(n \ln n)$ Force Calculation Algorithm. The Institute for Advanced Study, Princeton, NJ.

[5] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulation. *Journal of Computationl Physics*, 73(325), 1987.

[6] Jaswinder Pal Singh, John L. Hennessy, and Anoop Gupta. Implications of Hierarchical N-body Methods for Multiprocessor Architecture. Computer Systems Laboratory, Stanford University.

[7] Jaswinder Pal Singh, Chris Holt, Takashi Totsuka, Anoop Gupta, and John L. Hennessy. Load Balancing and Data Locality in Hierarchical N-body Methods. Computer Systems Laboratory, Stanford University.

[8] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. Computer Systems Laboratory, Stanford University.