A Performance Study of Cosmological Simulations on Message-Passing and Shared-Memory Multiprocessors [*]

Marios D. Dikaiakos
Departments of Computer Science-Engineering and Astronomy
University of Washington, Seattle, WA 98195
marios@astro.washington.edu

Joachim Stadel
Department of Astronomy
University of Washington, Seattle, WA 98195
stadel@astro.washington.edu

March 15, 1995

Abstract

In this paper we describe PKDGRAV, a parallel hierarchical tree-structured code used to conduct cosmological simulations on shared-memory and message-passing multiprocessors. We explore performance traits of cosmological N-Body simulations on 32K to 1.3 million particles, running PKDGRAV on KSR-2 and Intel Paragon multiprocessors with up to 128 nodes. We quantify the computation and communication requirements of PKDGRAV and study its scalability. We show that the shared-memory implementation performs and scales better than the message-passing. We investigate the causes of poor scalability of the Paragon implementation and identify an implementation-specific performance bottleneck in the software cache mechanism pertinent to the Paragon implementation.

1 Introduction

The N-Body problem addresses the evolution of systems of particles (bodies) under the influence of Newtonian gravitational forces. N-Body simulations proceed for hundreds or thousands of time-steps, each time-step computing the force on every particle and updating its position in space. Efficient algorithms for the N-Body problem construct a hierarchical

[*]Support for this research comes from the NASA Earth and Space Sciences High Performance Computing and Communications Program. We thank the San Diego Supercomputing Center for providing us with Intel Paragon time for our runs.

description of the mass distribution (a tree) and traverse it performing exact or approximate calculation of interactions. The parallel implementation of these hierarchical methods entails expensive, unstructured computations on huge data-sets and non-uniform communications [1, 2, 24, 28]. Consequently, a lot of attention has been focused upon the development and performance evaluation of parallel N-Body algorithms [1, 4, 13, 32]

The purpose of this paper is to explore the performance behavior of PKDGRAV, a "production-quality," tree-structured, gravity code running on message-passing (INTEL Paragon) and shared-memory (KSR-2) multiprocessors. PKDGRAV is being developed by the High-Performance Computing and Communications group at the University of Washington, for NASA's Earth and Space Sciences Project. The goal of this effort is to conduct parallel cosmological N-Body simulations, to calculate the nonlinear final states of theories of structure formation, to design and analyze observational programs, and to assess the performance of state-of-the-art parallel machines [24]. In our work, we investigate basic traits of cosmological N-Body simulations of *practical interest*, quantify their computation and communication requirements, and study their scalability. Furthermore, we analyze causes of bad performance and identify significant bottlenecks. Finally, we seek to understand the practical limitations of parallel algorithms employed, and assess the overhead introduced by their parallel implementation. Such a study is critical for guiding program and algorithmic design towards the achievement of faster, more portable, and *higher quality* cosmological simulations [24]. Moreover, it is helpful in exploring software and hardware aspects of state-of-the-art parallel computers that affect application performance, and in comparing the relative merits of different architectural and programming paradigms.

So far, a number of research projects have focused on studying the performance of parallel, hierarchical N-Body algorithms [18, 29, 10, 11, 22]. Conducting performance analyses of such parallel applications, however, is a difficult task for a variety of reasons, including the high computational requirements of these algorithms, the great volume of instrumentation data, the intrusiveness of instrumentation, and the low availability of large multiprocessor systems. To cope with these problems, most studies focus on "kernels" of N-Body codes, examine problem-instances of small data-sets, produce a few performance metrics over a small range of problem and machine sizes, or rely on analysis, simulation and modeling. Our approach differs, as: 1) We examine parallel codes developed and used by Astronomers to conduct research in Astrophysics (see [24]). 2) We run and evaluate these codes on data-sets of astrophysical interest. 3) We use parallel machines with the same architecture and size with those employed for "production" runs.

We take this approach because we are interested primarily in the efficient parallel implementation of the tree-codes developed by our group, rather than in conducting a more theoretical study of tree-algorithms. Notably, experience has shown that, when it comes to practical parallel performance, physical details are important. For example, the parallel times of two N-Body simulations of similar size running on the same architecture, can be very different if different criteria of physical accuracy are used [1]. Moreover, our experiments show that interesting bottlenecks appear only on runs involving many processors and large

[1]For a discussion on accuracy criteria of N-Body simulations, see the *N-Body Constitution* in [24]

data-sets; that is, on runs of practical interest!

The rest of the paper is organized as follows. The next section gives a brief survey of cosmological N-Body simulation and the hierarchical tree-algorithm employed in PKDGRAV. In Section 3 we discuss our experimental methodology. In Section 4 we present our measurements, explore the performance traits of the codes, and discuss performance bottlenecks. Section 5 looks at scaling issues for the INTEL Paragon implementation. Finally, Section 6 presents our conclusions and summarizes our results.

2 The Parallel k-D Tree Gravity Code

The N-Body problem addresses the gravitational evolution of astrophysical systems. It models the dynamical changes of a continuous mass distribution by approximating it with a number of particles interacting solely under the influence of gravity. The most computationally expensive part of this calculation is comprised of $\frac{1}{2}N(N-1)$ gravitational interactions between the N particles that represent the mass distribution; these interactions must be performed for 500 to 10,000 time-steps for typical astrophysical simulations. The values of choice for N are critical for the quality of astrophysical simulation results (see Figure 1 [2]). It is important to use values of N that do not compromise the validity of models for the set of physical phenomena under investigation. To this end, N should be greater than 10^6 for computations aiming at determining cosmological parameters through simulation of clusters of galaxies (like the density of the Universe) [24]. Other useful N-Body simulations conducted by our group use N equal to 250,000 for Local Group runs, 1.3 million (10,000 time-steps) for a Virgo cluster of galaxies (Figure 1), and 3 million (700 time-steps) for a 100Mpc CDM Cosmological Volume.

Improvements in the performance and quality of N-Body simulations has been sought in four general areas: 1) faster calculation of the gravitational accelerations; 2) multi-stepping, which is the reduction in number of time steps taken by particles in regions of the simulation where longer dynamical times are expected [25, 24]; 3) volume renormalization, where regions of interest are identified and populated with a greater density of particles than the surrounding volume [20, 21]; 4) the use of parallel and vector supercomputing techniques.

The need for rapid calculation of the gravitational accelerations has led to two basic approaches. The first uses grid techniques, relying mainly on the speed of FFT algorithms for the calculation of the gravitational field. This class includes the PM, P^3M [17] and AP^3M [9] algorithms. The other approach uses multipole expansions within a hierarchical description of the mass distribution (a tree). This second class includes well known algorithms such as the Barnes-Hut tree code [1, 2], the Fast Multipole Method [13, 14] and other variants including the *Parallel k-D Tree Gravity code* (PKDGRAV).

All of the approaches mentioned above set some level of approximation in their accuracy of the calculated gravitational accelerations. Therefore, the comparison of the relative

[2]This picture is in color; if the reviewer does not get a a hard-copy color version of it, he/she could find it online on our WWW site (URL: http://www-hpcc.astro.washington.edu/)

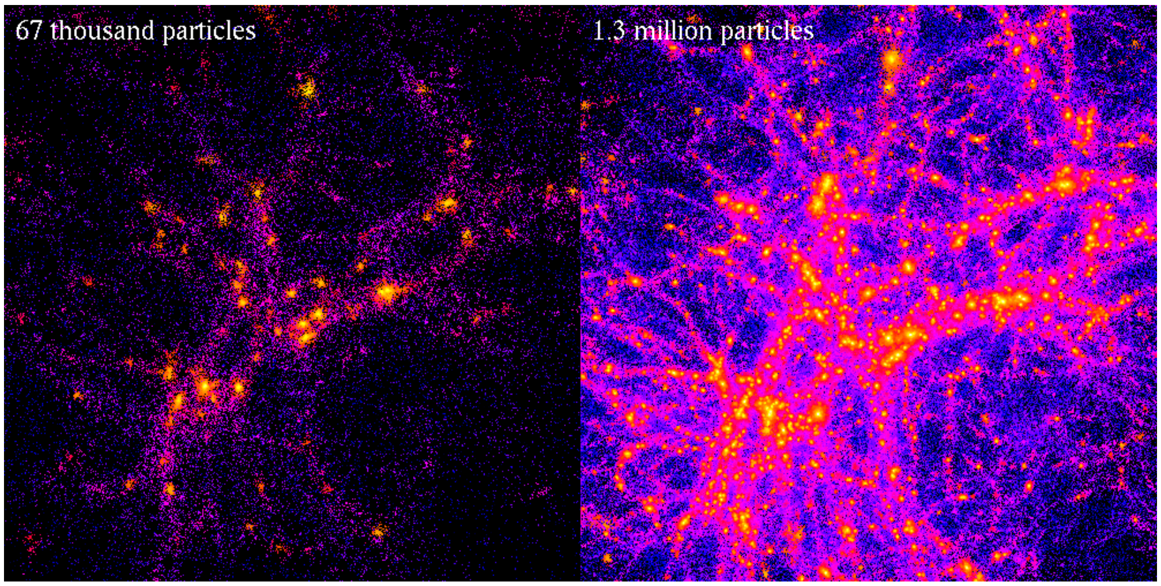


Figure 1: A comparison of a CRAY simulation with 67 thousand particles, to a KSR–1 simulation with 1.3 million particles, shown at a redshift of 2. The CRAY run was performed with a vectorized tree-code, whereas the KSR run used the PKDGRAV code.

efficiency of different approaches is hard, because one has to take into account factors like the type of the physical problem under study, and acceleration-error distributions and correlations. For this reason, such a comparison is not attempted in this article. Instead, we focus on performance results for PKDGRAV that correspond to error tolerances which we have found to be adequate for typical astrophysical problems.

2.1 The k-D Tree Structure

The central data structure in PKDGRAV is a tree structure which forms the hierarchical representation of the mass distribution. Unlike the more traditional *oct-tree* used in the Barnes-Hut algorithm, we use a *k-D tree* [3], which is a balanced binary tree. The *root-cell* of this tree represents the entire simulation volume. Other cells represent rectangular sub-volumes that contain the mass, center-of-mass, and quadrupole moment of their enclosed regions.

To build the k-D tree, we start from the root-cell and bisect recursively the cells through their longest axis, so that an equal number of particles lie in each sub-volume, and that quadrupole moments of cells are kept to a minimum (see Figure 2). This bisection is accomplished using Hoare's median finding algorithm, which is an $O(N)$ average time operation per level of the tree, making the tree building process an $O(N \log_2(N))$ operation. The depth of the tree is chosen so that we end up with at most 8 particles in the leaf cells (*buckets*). We have found this number to be near optimal for the parallel gravity calculation.

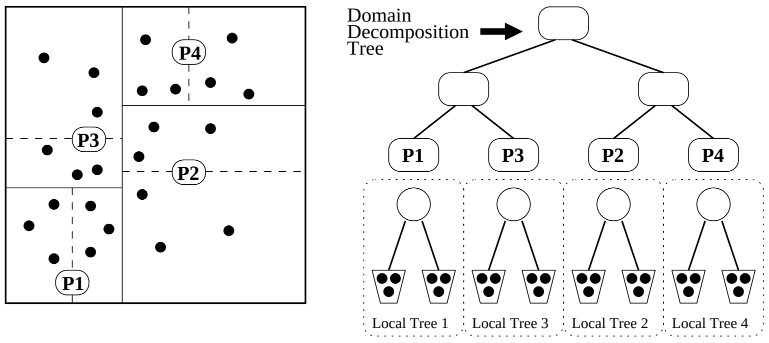


Figure 2: Two-dimensional k-D Tree distributed over four processors.

Several factors motivated the use of k-D tree structure over the classical oct-tree. The simplicity of the structure and the availability of fast median finding algorithms allow for a very efficient tree-construction. Pointers are unnecessary since each node in the tree can be indexed so that the finding of children, parent and sibling nodes are simple bit-shift operations. The use of buckets, by which only $2\lceil N/8 \rceil$ nodes are required, makes the tree structure memory-efficient. Most significantly, it can be extended to a parallel, distributed tree structure in a very natural way.

2.2 Calculating Gravity

PKDGRAV calculates the gravitational accelerations using the well known *tree-walking* procedure of the Barnes-Hut algorithm [2], except that it collects interactions for entire buckets rather than single particles. Thus, it amortizes the cost of tree traversal for a bucket, over all its particles. In the tree building phase, PKDGRAV assigns to each cell of the tree an *opening radius* about its center-of-mass. This is defined as,

$$r_{\mathrm{open}} = \frac{2B_{\mathrm{max}}}{\sqrt{3}\,\theta} + B_{\mathrm{center}} \tag{1}$$

where B_{max} and B_{center} are the maximum distances from a particle in the cell to the center-of-mass and center-of-cell respectively. θ is a user specified accuracy parameter which is similar to the traditional θ parameter of the Barnes-Hut code; notice that decreasing θ in Equation 1, increases r_{open}.

The opening radii are used in the *Walk* phase of the algorithm as follows: for each bucket $\mathcal{B}_i$, PKDGRAV starts descending the k-D tree, "opening" those cells whose r_{open} intersect with $\mathcal{B}_i$ (see Figure 3). If a cell is "opened," then PKDGRAV repeats the intersection-test with $\mathcal{B}_i$ for the cell's children. Otherwise, the cell is added to the *particle-cell interaction list* of $\mathcal{B}_i$. When PKDGRAV reaches the leaves of the tree and a *bucket* $\mathcal{B}_j$ is opened, all of $\mathcal{B}_j$'s particles are added to the *particle-particle interaction* list of $\mathcal{B}_i$. Once the tree has been traversed in this manner we can calculate the gravitational acceleration for each particle of

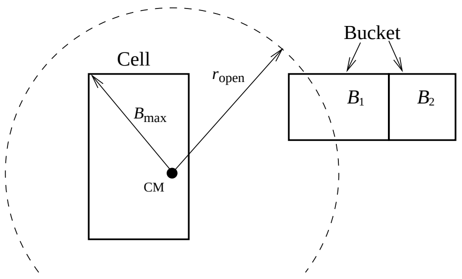


Figure 3: Opening radius for a cell in the k-D tree, intersecting bucket B_1 and not bucket B_2. This cell is "opened" when walking the tree for B_1. When walking the tree for B_2, the cell will be added to the particle-cell interaction list of B_2.

$\mathcal{B}_i$ by evaluating the interactions specified in the two lists. PKDGRAV uses a second-order multipole expansion to calculate particle-cell interactions.

Periodic Boundary Conditions

One disadvantage of tree codes is that they must deal with periodic boundary conditions explicitly, unlike grid codes where this aspect is taken care of implicitly. Although this adds complexity to any tree code, it is possible to incorporate periodic boundary conditions efficiently by using approximations to the *Ewald* summation technique [16, 12]. PKDGRAV differs significantly from the prescription given by [16], which is ill suited to a parallel code. Due to the mathematical technicality of the method we do not provide further description here except stating that it is ideally suited to parallel computation [30].

2.3 Parallel Aspects of the Code

2.3.1 Domain Decomposition

Achieving effective parallelism requires that work be divided equally amongst the processors in a way which minimizes interprocessor communication during the gravity calculation. Since we only need a crude representation for distant mass, the concept of data locality translates directly into spatial locality within the simulation. Each particle can be assigned a *work-factor*, proportional to the cost of calculating its gravitational acceleration in the prior time-step. Therefore, during domain decomposition, we divide the particles into spatially local regions of approximately equal work.

Experience has shown that using a data structure for the domain decomposition that does not coincide with the hierarchical tree for gravity calculation, leads to poor memory scaling with number of processors and/or tedious book-keeping. That is the case, for instance, when using an Orthogonal Recursive Bisection (ORB) tree for domain decomposition and an oct-tree for gravity [28]. Current domain decomposition techniques for the oct-tree case involve forming "costzones," that is, processor domains out of localized sets of oct-tree cells [29], or "hashed oct-trees" [31]. PKDGRAV uses the ORB tree structure to represent the domain decomposition of the simulation volume. The ORB structure is completely compatible with the k-D tree structure used for the gravity calculation (see Figure 2). Instead of a median finder, a root finder is used to recursively subdivide the simulation volume so that the sums of the work-factors in each processor domain are equal. Once this has been done, each processor builds a local tree from the particles within its domain. This entire domain decomposition and tree building process are fully parallelizable and incur negligible cost to the overall gravity calculation.

2.3.2 Parallel tree-walking phase

The *Walk* phase starts from the root-cell of the domain decomposition tree (ORB tree), each processor having a local copy of this tree, and descends from its leaf-cells into the local trees stored on each processor. PKDGRAV can index the parent, sibling and children of a cell. Therefore, it can traverse a k-D tree stored on another processor in an architecture independent way. Non-local cells are identified uniquely by their cell index and their domain number (or processor identifier). Hence, tree walking the distributed data structure is identical to tree walking on a single processor, except that PKDGRAV needs to keep track of the domain number of the local tree upon which the walk is performed. Interaction lists are evaluated as described earlier, making *Walk* the only phase where interprocessor communication takes place, after the domain decomposition.

2.3.3 The Machine Dependent Layer

A small library of high level functions called MDL (Machine Dependent Layer) handles all parallel aspects of the code. This keeps the main gravity code architecture-independent and simplifies porting. For example, MDL provides a memory swapping primitive to move particles between processors during domain decomposition. Furthermore, MDL provides memory sharing primitives allowing local arrays of data to be visible to all processors. These primitives support access to non-local cells and particles during the *Walk* phase. In particular, a procedure called `mdlAquire` can be used to request and receive non-local data by providing an index into a non-local array, and an identifier for the processor that owns that array. On the KSR, we rely on the shared address space to implement `mdlAquire`.

On distributed memory machines, such as the INTEL Paragon and IBM's SP-2, we maintain a local software cache on each processor. When a processor requests a cell or particle that is not in its cache, the request is sent to the appropriate processor that owns the data. While waiting for the data to be sent back, the requesting processor handles cache-requests sent from other processors. The owner processor sends back a cache line comprised of more than a single cell or particle, in an attempt to amortize the effects of latency and message passing overhead. This cache line is inserted into the local cache, and a pointer to the requested element is returned. To improve responsiveness of the software cache, after every tenth access to the cache MDL checks whether any requests for data have arrived; if so, it services these first.

3 Experimental Methodology

In this section we present the framework that we used to study the performance characteristics of the N-Body codes under investigation.

3.1 Parameters employed

A number of parameters determine the accuracy and validity of the physical conclusions drawn from cosmological simulation results; the same parameters affect parallel simulation performance. The most important are: the number of particles N; the accuracy parameter of the "opening criterion" θ; *Redshift*, which determines the spatial configuration of particles at the beginning of the simulation; and *nSteps*, the total number of time-steps for which the N-Body algorithm is executed. These parameters define a wide range of possible runs. For the simulations presented in this paper we used the values of Table 1.

N	θ	*Redshift*	*nSteps*
$32,768$ to $1,300,000$	0.55	49	2

Table 1: Physical parameters used in our simulations.

The maximum number of particles of a large simulation is constrained by the processor-cycles available for the computation of gravitational interactions and by the main memory required to store the particles and the tree structure. These constraints become more stringent when conducting performance experiments: in general, less computing-time is dedicated to performance monitoring; instrumented codes are slower; instrumentation increases memory consumption and traces take large amounts of disk space. Nevertheless, we can run a larger number of simulations on smaller data-sets and collect enough evidence to extrapolate the behavior of the "production-runs." In this paper we present results for N ranging from $32,768$ to $1,300,000$.

We use a Redshift value of 49, which corresponds to the initial configuration of particles used in the cosmological simulations of our group. This configuration is spatially uniform; therefore, we are conducting further experiments for a Redshift of 2, which corresponds to highly clustered particle-configurations. [3]

We run the tree-codes only for two time-steps due to limitations on available multiprocessor resources. This is not a problem from the performance analysis standpoint, however, because the parallel execution of the tree-algorithm has very similar performance characteristics across successive time-steps: particles move slowly from time-step to time-step and, thus, the overall particle-configuration changes very slowly. We use 2 time-steps instead of 1, because the domain decomposition in the first time-step is conducted after assigning all particles with an identical work-factor. From the second time-step and on, PKDGRAV assigns each particle with a work-factor proportional to the time spent computing its acceleration in the previous time-step. Running simulations for two time-steps allows us to examine any possible effects that this may have.

[3] The results will be included in the final draft.

Processor	INTEL i860XP	Custom VLSI
Clock cycle	50MHz	40MHz
Data Cache	16KB	256KB
Instruction Cache	16KB	256KB
Memory per Node	32MB	32MB
Peak Mflops per sec per node	75	80
Topology	Mesh	Ring
Operating System	OSF/1 Release 1.0.4	KSR OS

Table 2: System characteristics.

θ is set to 0.55 because, for the simulation experiments of interest, extensive tests showed that this value guarantees satisfactory error-bounds for physical criteria defined by Astronomers (e.g. relative and absolute acceleration error).

3.2 Parallel Platforms employed

The tree-codes were initially developed in C, for the KSR's shared-memory paradigm (*pthread* libraries) and later ported to the PVM message-passing library [23]. Due to the large overhead of PVM runs on networks of workstations, we ported the programs onto INTEL's *NX* message-passing library, in order to assess their performance on the Paragon scalable message-passing environment. We conducted our experiments on two different platforms: a 64-processor Kendall Square Research KSR-2 multiprocessor [5] and a 16-processor INTEL Paragon system [7]. Both machines are installed in the Department of Computer Science-Engineering at the University of Washington. We also used the INTEL Paragon of the San Diego Supercomputing Center (with a maximum queue size of 256 nodes).

The KSR-2 is a "Cache Only Memory Architecture" multiprocessor with 40MHz custom-built processors, configured as a hierarchy of slotted, packetized rings with 32 processors on each leaf-level ring. The KSR processor can execute two instructions per cycle; one instruction can be an address calculation, load/store or branch and the other can be either an integer or floating-point instruction. KSR-2 provides a shared address space with physically distributed memory. Memory modules of each node are playing the role of a very large, hardware-managed cache. Cache coherence is provided through a hierarchical directory scheme which enforces sequential consistency.

The INTEL Paragon is a scalable message-passing system, based on the INTEL i860XP, 50MHz microprocessor. The i860 has independent integer and floating point units, and the floating point unit has an independent pipeline adder and multiplier. Communication between the processors is carried through a mesh interconnection network with 200 MByte-per-second links. Table 2 summarizes the basic system characteristics of the KSR-2 and the INTEL Paragon multiprocessors.

Machine Size (P)														
	1		2		4		8		16		32		64	60
N	Intel	ksr	Intel	ksr	Intel	ksr	Intel	ksr	Intel	ksr	Intel	ksr	Intel	ksr
32k	69	102	111	181	201	348	313	668	458	1040	na	1724	na	2340
67k	63	99	112	187	199	361	361	675	551	1182	486	1854	na	3267
125k	84	95	143	178	244	323	450	628	715	1101	964	1865	442	3378
250k	56	82	101	156	184	292	333	528	537	1004	461	1694	409	2232
500k	47	72	92	132	169	243	301	502	486	921	462	1557	448	na
1.3M	na	na	na	na	na	na	na	na	154	315	333	na	242	na

Table 3: Science Rate (in *particles per seconds*).

3.3 Performance Characteristics sought

A first goal of our study is to identify *system requirements* of the tree-codes in terms of running time, memory allocation, and sustained communication rates. We are interested in examining the scaling of these requirements with increasing problem and machine sizes and in comparing the relative performance of available parallel architectures. A second goal is to discover and identify bottlenecks of the cosmological simulation that hinder its performance. A third goal is to attribute performance bottlenecks either to the algorithms employed, their parallel implementation, or the inherent limitations of the multiprocessors used. Also, we are interested in studying various aspects of the scalability of PKDGRAV. Finally, we need to describe PKDGRAV's performance in terms of metrics of "physical significance." For instance, it is more relevant to use the number of gravitational interactions performed per second as an estimate for the aggregate performance of a certain parallel machine, than the count of bare Mflops per second. Notably, in the case of tree-structured gravitational algorithms, the Mflop rate does not characterize accurately the performance of the parallel codes, because these involve also a large number of integer and/or pointer operations during tree-walking.

With the above goals in mind, we instrumented the tree-codes to generate and collect application traits characterizing their computation, communication, and memory requirements, and their scalability. To gather "raw" performance data, we measured running times for the execution of the N-Body algorithm, as well as the execution of the most important of its phases. We instrumented *malloc* to keep track of dynamically allocated memory. In the case of INTEL's Paragon message-passing multiprocessor, we instrumented communication routines manually to collect communication traces. Furthermore, we used the performance monitoring environment on the INTEL Paragon [26]: *Xipd* [8] for automatic instrumentation and trace collection and *Paragraph* [15] for performance visualization. On the KSR-2, we used manual instrumentation only, because the *pmon* monitoring libraries failed to produce reliable data.

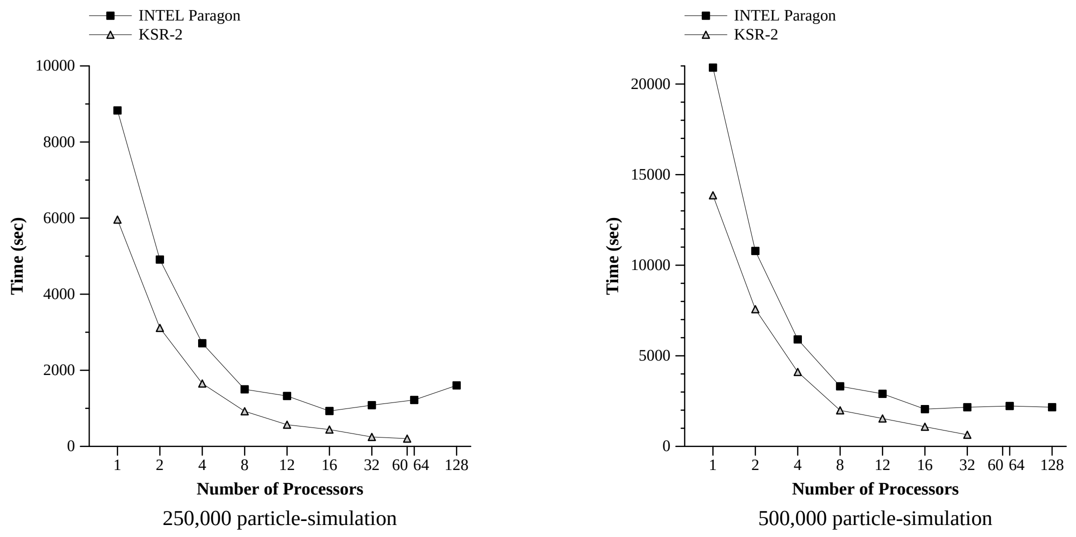


Figure 4: Running Times of two time-step simulations.

4 Performance Measurements

In this section, we summarize results from our performance experiments and assess the system requirements of the tree-codes examined. In early runs, it turned out that the INTEL Paragon sequential performance was twice as slow as that of the KSR, regardless of the value of N. This is partly due to the use of IEEE floating-point arithmetic on the Paragon. To improve performance, we compiled the tree-codes on the Paragon with the *-knoieee* option. We expect that this does not affect severely the accuracy of Paragon simulations.

4.1 Processing

Figure 4 presents measurements of the running times of two simulations with 250,000 and 500,000 particles respectively, proceeding for two time steps. Table 3 gives the *Science Rate*, an application-oriented performance metric defined as the number of particles N over the time it takes to complete *one time-step* of a cosmological simulation [6]. From Figure 4 and Table 3 we can easily see that KSR-2 outperforms the INTEL Paragon, both in terms of absolute Science Rate figures as well as in terms of the scaling of the Science Rate with the number of available processors. In particular, the Science Rate on the KSR-2 increases as we scale the machine size to 60 processors, whereas on the Paragon it starts decreasing over 16 processors. This can be seen also in Figure 4, where the running time on the Paragon levels-off as we scale the number of available processors to values larger than 16. We elaborate on this in the next section.

Another interesting remark is that, for our benchmarks, the single-processor performance of the KSR-2 is approximately 1.5 times better than that of the Paragon. This is

Machine Size (P)	*Flop Count in MFlops*	*Flop Rate (MFlops/sec)*	
		KSR-2	Paragon
1	22803.4	12.07	7.87
4	23232.41	47.55	31.79
16	23661.73	190.95	127.69
32	24189.15	373.66	253.95
60/64	24247.75	579.34	512.64
128	24465.65	n/a	1029.27

Table 4: Mflop counts and rates: two time-steps, 250,000 particles.

	Machine Size (P)									
	1		2		4		8		16	
N	Intel	ksr	Intel	ksr	Intel	ksr	Intel	ksr	Intel	ksr
32k	17.6	na	30.1	na	55.08	na	105.14	na	205.2	na
67k	22.9	27.3	35.4	26	60.4	29.6	110.38	27	210.35	29
125k	32.34	41.3	44.83	39.4	69.8	39.6	119.77	40	219.7	40
250k	52.19	78.5	64.68	78.6	89.6	78.8	139.6	80.2	239.58	82.2
500k	91.9	na	104.38	na	129.36	na	179.32	na	279.29	na

Table 5: Aggregate Memory Allocation (in MBytes)

attributed mainly to the substantially larger cache of the KSR-2 and the shared-memory paradigm which allows a single-processor run to use memory pages residing on other processors and, thus, to reduce paging. Both parallel codes, achieve a high Mflop/sec rate in the floating-point intensive *Interact* phase of the algorithm. Table 4 presents the aggregate number of flops and the flop rate for the *Interact* phase of a two time-step, 250,000-particle simulation. To derive the flop count we measured the floating-point operations in the codes examined. This gave a 34 flop count for a particle-particle interaction and a 71 flop count for a particle-cell interaction.

4.2 Memory

To get an estimate of the memory requirements of cosmological simulations, we instrumented *malloc*. The collected data are presented in Table 5. It is noted that most of the arrays statically allocated on the KSR-2, are allocated dynamically on the Paragon for performance reasons. On the other hand, *floats* are 8-bytes long on the KSR-2, and 4-bytes long on the Paragon. This explains the higher memory consumption on single-processor KSR-2 runs. Things change drastically for larger pools of processors, because of the extra memory allocated on every Paragon node for the implementation of "particle" and "cell" caches. Each cache is 2 million bytes large. The object code is 1.34MB on the KSR-2 and 0.35MB on the Paragon (both codes were compiled with the *-O2* option).

N	2	4	8	16	32
32k	1.98	4.5	8.57	14.04	15.63
67k	3.75	8.87	15.67		25.75
125k	5.19	11.25	18.62	30.86	30.57
250k	9.1 n/a	19.52	31.69	n/a	51.18
500k	15.62	35	55.85	87.1	n/a

Table 6: Aggregate Communication Volume per time-step (in MBytes)

4.3 Communication

Table 6 gives information about the total volume of messages sent between processors during our Paragon simulations. The left diagram of Figure 5 gives a pictorial representation of data from this Table. As expected, the total communication traffic increases with problem size and the number of processors. To estimate the average communication bandwidth that each Paragon processor utilizes during cosmological simulation, we divide the average volume of communication per processor and time-step, over the average duration of a time-step. We call this, *per-processor communication rate*; its values are presented in the right diagram of Figure 5. On the Paragon, given its 200MBytes/sec node-to-router sustained bandwidth,

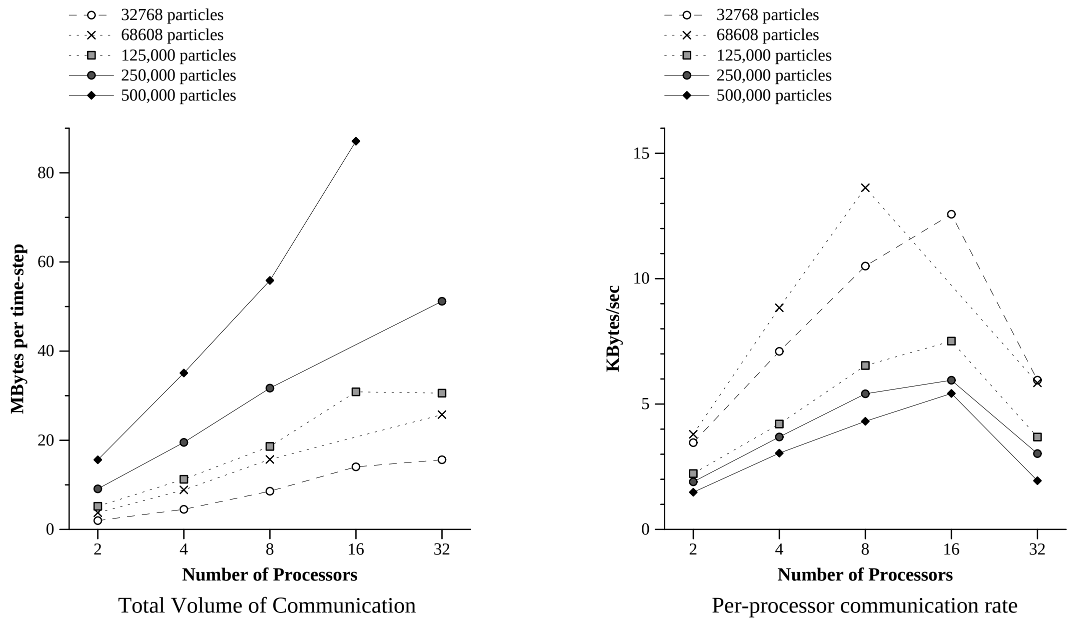


Figure 5: Volume of Communication

node-to-node communication does not become a bottleneck. Furthermore, even under the more restrictive assumption of a random message traffic, the per-processor communication rate does not exceed the sustainable per-processor bandwidth, which is determined by the bisection width of the Paragon mesh [19, 27]. Another interesting remark from Figure 5

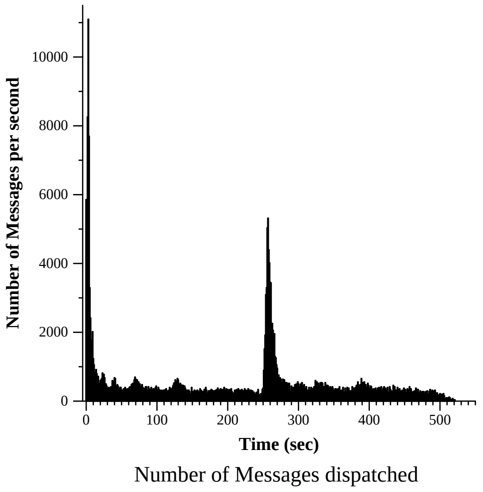


Figure 6: Communication profile: 32 processors, 2 time steps, 125,000-particle simulation

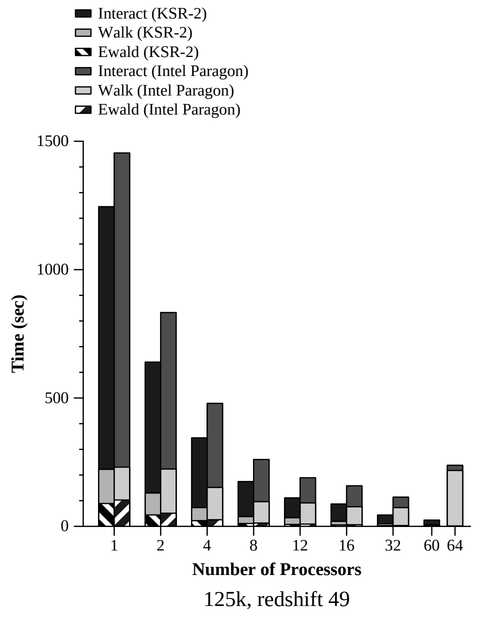
(right) is that, in most cases, the per-processor communication rate decreases when increasing the problem-size (N) and keeping the number of processors fixed. This means that the communication-to-computation ratio decreases as the problem size gets bigger.

Finally, we note that cosmological simulations produce bursty message traffic. This becomes clear from Figure 6, which displays the number of messages sent between processors during the first two time-steps of a 500,000-particle simulation, running on a 32-processor Paragon. Since the data presented here correspond to a Redshift of 49, the two communication bursts of Figure 6 are expected to be due to the filling of the MDL caches rather than to domain decomposition. Further experiments, however, showed that the total parallel time spent in tree-building is negligible and, thus, the effects of the observed message bursts to parallel performance are not significant.

5 Scaling Issues

To investigate the scaling properties of the tree-codes, we plot the partition of the running time to the three most time-consuming phases of the N-Body algorithm: *Interact*, *Walk*, and *Ewald*. The diagrams in Figure 7 present the breakups for two time-steps of 125,000- and 250,000-particle simulations. From the plots in Figure 7 we can see that the scaling of the *Interact* and the *Ewald* phases is satisfactory on both platforms. The *Walk* phase, which involves communication between processors, however, becomes the bottleneck to Paragon performance, as we increase the number of processors over 16.

Figure 8, presents the *relative speedups* of the three phases of the algorithm, that is the parallel execution time of a given phase over its sequential time on a single processor of the same architecture, versus the number of available processors. Notably, the execution time of the *Walk* phase on the Paragon is practically constant. Consequently, given the linear



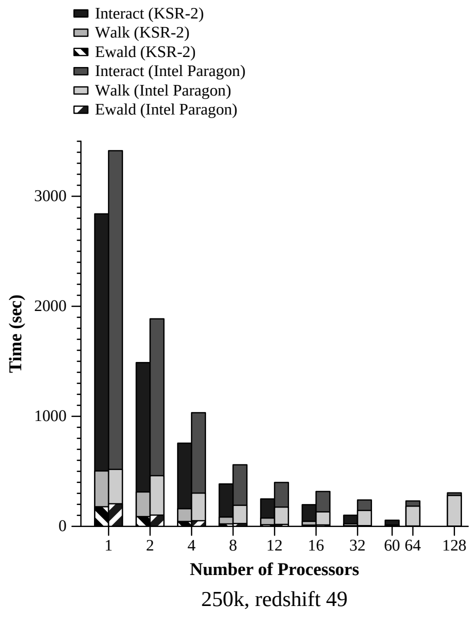


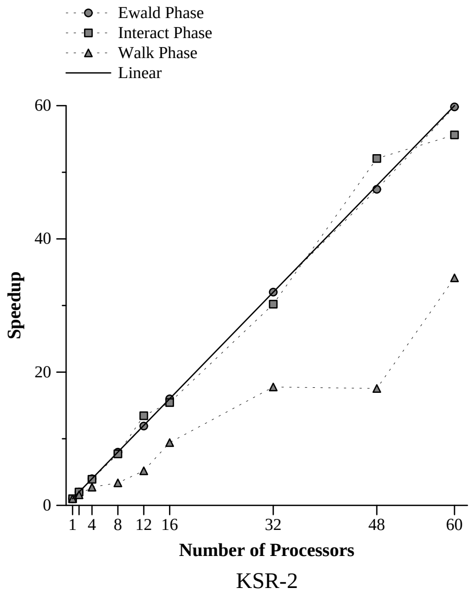
Figure 7: Fixed problem-size scaling of *Interact*, *Walk*, and *Ewald* on the KSR-2 and the Paragon

scaling of the other two phases, the overall parallel time on the Paragon becomes eventually bound by the time it takes to "walk" the k-D tree.

The poor scaling of the *Walk* phase on the Paragon could be attributed to communication overhead. Indeed, a close look at the k-D tree algorithm shows that when we increase the number of processors, there is an increase in the amount of non-local monopole and quadropole interactions. This effect is reflected at the sublinear scaling of the *Walk* phase on the KSR-2 (see Figure 8 - left) . The total lack of scaling on the Paragon, however, cannot be a result either of the increase of non-local interactions, or of communication congestion. Data concerning the communication rate (diagram on the right of Figure 5) make the latter assumption unlikely. Further experimentation showed that this phenomenon is a result of high-overhead in the MDL software cache that is implemented on the Paragon. Ways of improving the MDL cache are under investigation. In particular, either a background process running at real time priority or an interrupt handler may be required to implement more efficiently the MDL cache-request handler.

6 Conclusions

In this paper we described PKDGRAV, a parallel hierarchical tree-structured code used to conduct cosmological simulations on shared-memory and message-passing multiprocessors. To investigate the performance characteristics of PKDGRAV on the KSR-2 and the INTEL Paragon multiprocessors, we performed a series of simulations on data-sets and machine-



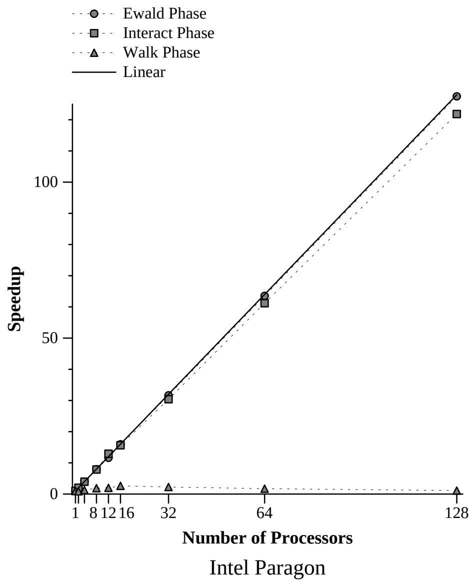


Figure 8: Speedup scaling of *Interact*, *Walk*, and *Ewald* on the KSR-2 and the Paragon (250,000-particle simulation)

sizes of practical interest. Our experiments showed that the floating-point intensive part of PKDGRAV reaches a performance of 1 GFlop/sec on a 128-processor Paragon and 0.5 GFlop/sec on a 60-node KSR-2, for a medium-sized problem. On the Paragon, communication traffic is bursty and the communication-to-computation ratio is very low and decreases with problem size. Overall, the KSR-2 outperforms the Intel Paragon, both in terms of absolute performance numbers as well as in terms of its scaling properties.

Further investigation showed that the cause of PKDGRAV's poor scaling on the Paragon comes from the *Walk* phase of the algorithm, where the distributed tree structure is traversed. The poor scaling, however, is not inherent to the algorithm but lies on the software caching mechanism used to implement sharing of data during cosmological simulation. A redesign of the software cache is under way. Notably, both the *Interact* phase, which performs the gravitational calculations, and *Ewald*, which deals with boundary conditions, scale linearly with the number of processors on both platforms and for a large range of problem sizes. Finally, the parallel time spent in domain decomposition and tree-construction is negligible compared to the other phases.

References

[1] J. Barnes. An efficient N-body algorithm for a fine-grain parallel computer. In *The Use of Supercomputers in Stellar Dynamics*, pages 175–180. P. Hut and S. McMillan, eds, Springer Verlag, 1986.

[2] Josh Barnes and Piet Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, December 1986.

[3] J.L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communication of the ACM*, 18(9), September 1975.

[4] J.A. Board, Z.S. Hakura, W.D. Elliot, D.C. Gray, W.J. Blanke, and Jr J.F. Leathrum. Scalable implementations of multipole-accelarated Algorithms for Molecular Dynamics. In *Proceedings of the 1994 Scalable High-Performance Computing Conference (SHPCC94)*, pages 87–94. IEEE-Computer Society Press, 1994.

[5] Henry Burkhardt. Overview of the KSR1 computer system. Technical Report KSR-TR-9202001, Kendall Square Research, February 1992.

[6] Ray Carlberg. Personal communication, 1994.

[7] Intel Corporation. Paragon XP/S product overview. Technical report, 1992.

[8] Intel Corporation. *Paragon's Application Tools User's Guide*, June 1994.

[9] H.M.P. Couchman. Mesh-refined P^3M: a fast adaptive N-body algorithm. *Astrophys. J.*, (389):453–463, 1992.

[10] M. Dikaiakos, A. Rogers, and K. Steiglitz. Functional Algorithm Simulation of the Fast Multipole Method: Architectural Implications. *Parallel Processing Letters (accepted for publication)*, 1994.

[11] Marios Dikaiakos. *FAST: A Functional Algorithm Simulation Testbed*. PhD thesis, Dept. of Computer Science, Princeton University, January 1994.

[12] H-Q Ding, N. Karasawa, and W.A. Goddard III. The reduced cell multipole method for coulomb interactions in periodic systems with million-atom unit cells. *Chemical Physics Letters*, 196(1,2):6–10, August 1992.

[13] L. Greengard and W. Gropp. A Parallel Version of the Fast Multipole Method. In Garry Rodrigue, editor, *Parallel Processing for Scientific Computing*, pages 213–222. SIAM, 1987.

[14] Leslie Greengard. *The rapid evaluation of potential fields in particle systems*. PhD thesis, Yale University, Cambridge, Mass., 1988.

[15] M.T. Heath and J.A. Etheridge. Visualizing the Performance of Parallel Programs. *IEEE Software*, 8(5):29–39, September 1991.

[16] L. Hernquist, F. Bouchet, and Y. Suto. Application of the Ewald Method to Cosmological N-Body Simulations. *The Astrophysical Journal Supplement Series*, 75:231–240, February 1991.

[17] R.W. Hockney and J.W. Eastwood. *Computer Simulation Using Particles*. McGraw Hill, 1981.

[18] Chris Holt and Jaswinder Pal Singh. Hierarchical N-body Methods on Shared Address Space Multiprocessors. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 313–318. SIAM, February 1995.

[19] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw Hill, 1993.

[20] Neal Katz, Thomas Quinn, Edmund Bertschinger, and James M. Gelb. Formation of quasars at high redshift. *Mon. Not. R. Astron. Soc.*, (270):L71–L74, 1994.

[21] Neal Katz and Simon White. Hierarchical galaxy formation: overmerging and the formation of an x-ray cluster. *Astrophysical Journal*, 412(2):412–455, 1993.

[22] J. Katzenelson. Computational Structure of the N-Body Problem. *SIAM Journal of Scientific and Statistical Computing*, 10(4):787–815, July 1989.

[23] Oak Ridge National Laboratory. *PVM 3 User's Guide and Reference Manual*, May 1993.

[24] George Lake, Neal Katz, Thomas Quinn, and Joachim Stadel. Cosmological N-body Simulation. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 307–312. SIAM, February 1995.

[25] Thomas Quinn, Neal Katz, Joachim Stadel, and George Lake. Time stepping n-body simulations. In preparation.

[26] B. Ries, R. Anderson, W. Auld, K. Callaghan, E. Richards, and W. Smith. The Paragon Performance Monitoring Environment. In *Supercomputing '94*, pages 850–859, 1994.

[27] E. Rothberg, J.P. Singh, and A. Gupta. Working Sets, Cache Sizes and Node Granularity Issues for Large-Scale Multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 14–25, May 1993.

[28] J. K. Salmon. *Parallel Hierarchical N-body Methods.* PhD thesis, California Institute of Technology, 1990.

[29] Jaswinder Pal Singh. *Parallel Hierarchical N-Body Methods and their Implications for Multiprocessors.* PhD thesis, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, March 1993.

[30] Joachim Stadel and Tom Quinn. Pkdgrav: Using parallel computers for cosmological simulations. In preparation.

[31] Michael S. Warren and John K. Salmon. A Parallel, Portable and Versatile Treecode. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 319–324. SIAM, February 1995.

[32] F. Zhao and S.L. Johnsson. The Parallel Multipole Method on the Connection Machine. *SIAM Journal of Sci. and Stat. Comput.*, 12:1420–1437, 1991.