Performance of a Heterogeneous Grid Partitioner for N-body Applications*

Daniel J. Harvey
Dept. of Computer Science
Southern Oregon Univ.
Ashland, OR 97520
harveyd@sou.edu

Sajal K. Das
Dept. of Computer Science
Univ. of Texas at Arlington
Arlington, TX 76019
das@cse.uta.edu

Rupak Biswas
NAS Division
NASA Ames Research Ctr.
Moffett Field, CA 94035
rbiswas@nas.nasa.gov

Abstract

An important characteristic of distributed grids is that they allow geographically separated multicomputers to be tied together in a transparent virtual environment to solve large-scale computational problems. However, many of these applications require effective runtime load balancing for the resulting solutions to be viable. Recently, we developed a latency tolerant partitioner, called MinEX, specifically for use in distributed grid environments. This paper compares the performance of MinEX to that of METIS, a popular multilevel family of partitioners, using simulated heterogeneous grid configurations. A solver for the classical N-body problem is implemented to provide a framework for the comparisons. Experimental results show that MinEX provides superior quality partitions while being competitive to METIS in speed of execution.

Keywords: Grid computing, dynamic load balancing, graph partitioning, latency tolerance, N-body problem, heterogeneous distributed environments, performance analysis

1 Introduction

Computational grids hold great promise in utilizing geographically separated resources to solve large-scale complex scientific problems. The development of such grid systems has therefore been actively pursued in recent years [1, 4, 6, 9, 10, 14, 15]. The Globus project [9], in particular, has been remarkably successful in the development of grid middleware consisting of a general purpose, portable, and modular toolkit of utilities. A comprehensive survey of several grid systems is provided in [8].

*This work was partially supported by NASA Ames Research Center under Cooperative Agreement NCC 2-5395.

Examples of applications that could potentially benefit from computational grids are abundant in several fields including aeronautics, astrophysics, molecular dynamics, genetics, and information systems. It is anticipated that grid solutions for many of these applications will become viable with the advancement of interconnect technology in wide area networks. However, applications that require solutions to adaptive problems need dynamic load balancing during the course of their execution. Load balancing is typically accomplished through the use of a partitioning technique to which a graph is supplied as input. This graph models the processing and communication costs of the application. Many excellent partitioners have been developed over the years; refer to [2] for a survey. However, the most successful state-of-the-art partitioners are multilevel in nature [11, 12, 21] that contract the supplied graph by collapsing edges, partition the coarsened graph, and then refine the coarse graph back to its original size.

Although some research has been conducted to analyze the performance of irregular adaptive applications in distributed-memory, shared-memory, and cluster multiprocessor configurations [17, 18, 19], till date little attention has been focused on heterogeneous grid configurations. In [7], we proposed a multilevel partitioner, called MinEX, designed specifically for applications running in grid environments. MinEX operates by mapping a *partition graph* (that models the application) onto a *configuration graph* (that models the grid), while considering the anticipated level of latency tolerance that can be achieved by the application. Recently, this concept has been extended to heterogeneous grids [13]; however, latency tolerance was not considered in the implementation.

This paper provides several important extensions to the work presented in [7]. Our major contributions are to (i) demonstrate the practical use of MinEX with an actual application solver, (ii) present details of MinEX interaction with an application to achieve improved performance in a high-latency low-bandwidth grid environment, and (iii) directly compare MinEX performance to that of a state-of-the-art partitioner and establish the effectiveness of algorithms of this kind.

METIS [12] is perhaps the most popular of all multilevel partitioning schemes. However, when applied to a grid environment, METIS has some serious deficiencies. We enumerate below these METIS drawbacks and indicate how they are addressed by MinEX:

- METIS optimizes graph metrics like edge cut or volume of data movement and therefore operates in two distinct phases: partitioning and remapping. This approach is usually very inefficient in a distributed environment. MinEX, on the other hand, creates partitions that take data remapping into consideration and strives to overlap application processing and communication to minimize the total runtime of the application.

- For heterogeneous grids, the processing and communication costs are non-uniform. Assuming uniform weights for the underlying system (as METIS does) is therefore insufficient. Instead, MinEX utilizes a configuration graph to model grid parameters such as the number of processors, the number of distributed clusters, and the various processing and communication speeds. The partition graph is mapped onto this configuration graph to accommodate a heterogeneous environment.

- Traditional partitioners like METIS do not consider any latency tolerance techniques that could be employed by an application to hide the detrimental effects of low bandwidth in grid environments. However, MinEX has the proper interface to invoke a user-supplied problem-specific function that models the latency tolerance characteristics of the application.

To evaluate MinEX and compare its effectiveness to METIS for heterogeneous grids, we implemented a solver based on the Barnes & Hut algorithm [3] for the classical N-body problem. Test cases of 16K, 64K, and 256K bodies are solved. We simulate different grid environments that model 8 to 1024 processors configured in 4 or 8 clusters, having interconnect slowdown factors of 10 or 100 and possess various degrees of heterogeneity. Results show that MinEX reduces the runtime requirements to solve the N-body application by up to a factor of 6 compared to those obtained when using METIS in heterogeneous configurations. Results also show that MinEX is competitive to METIS in terms of partitioning speed.

The paper is organized as follows. Section 2 presents basic concepts of partitioners and defines the metrics that we use. An overview of our MinEX partitioner is provided in Section 3, while Section 4 outlines the N-body problem and our solution procedure. Section 5 describes the experimental methodology, analyzes the results, and draws comparisons with METIS. Finally, Section 6 concludes the paper.

2 Preliminaries

Here we present some basic partitioning concepts and define the metrics used for partitioner performance analysis.

2.1 Partition Graph

A graph representation of the application is supplied as input to partitioners so that the vertices can be assigned among the processors of a multicomputer (or a grid) in a load balanced fashion. Each vertex v of this *partition graph* has two weights, $\texttt{PWgt}_v$ and $\texttt{RWgt}_v$, while each defined edge (v, w) between vertices v and w has one weight, $\texttt{CWgt}_{(v,w)}$. These weights refer respectively to the processing, data remapping, and

communication costs associated with processing a graph vertex. Details of how these weights are computed for our N-body application are given in Section 4.3.

2.2 Configuration Graph

To predict performance on a variety of distributed architectures, a *configuration graph* is utilized by MinEX. This graph defines the heterogeneous characteristics of the grid environment and allows appropriate partitioning decisions to be made. It contains a vertex for each cluster c, where a cluster consists of one or more tightly-coupled processors. A graph edge (c, d) corresponds to the communication connections between the processors in clusters c and d. A self-loop (c, c) indicates communication among the processors of a single cluster. We assume that all processors within a cluster are fully connected and homogeneous, and that there is a constant bandwidth for intra-cluster communication.

The vertices of the configuration graph have a single weight, $\texttt{Proc}_c \geq 1$. The weight represents the processing slowdown for the processors of cluster c, relative to the fastest processor in the entire grid (normalized to unity). Likewise, edges have a weight $\texttt{Connect}_{(c,d)} \geq 1$ to model the interconnect slowdown when processors of cluster c communicate with processors of cluster d. If $\texttt{Connect}_{(c,d)}$ is unity, there is no slowdown (this represents the most efficient connection in the network). If $c = d$, $\texttt{Connect}_{(c,c)}$ is the intra-connect slowdown when processors of c communicate internally with one another. In addition to the configuration graph, a processor-to-cluster mapping $\texttt{CMap}_p$ determines which cluster is associated with each processor p in the grid.

2.3 Time Unit Metrics

The MinEX partitioner is unique in that it's partitioning objective is to minimize application runtime. To accomplish this goal, the partition graph that models the application is supplied by the application and is used to measure units of computation, communication, and data remapping. The partition graph is then mapped onto the grid configuration graph. The following three metrics are used for these purposes.

- **Processing Cost** is the computational cost to process vertex v assigned to processor p in cluster c and is given by $\texttt{Wgt}_p^v = \texttt{PWgt}_v \times \texttt{Proc}_c$.
- **Communication Cost** is the cost to interact with all vertices adjacent to v but whose data sets are not local to p (assuming that v is assigned to p). If vertex w is adjacent to v, while c and d are the clusters associated with the processors assigned to v and w, this metric is given by $\texttt{Comm}_p^v = \sum_{w \notin p} \texttt{CWgt}_{(v,w)} \times \texttt{Connect}_{(c,d)}$. If the data sets of all the vertices adjacent to v are also assigned to p, $\texttt{Comm}_p^v = 0$.

– **Redistribution Cost** is the transmission overhead associated with copying the data set of v from p to another processor q. It is 0 if $p = q$; otherwise it is given by $\texttt{Remap}_p^v = \texttt{RWgt}_v \times \texttt{Connect}_{(c,d)}$. Here we assume that p is in cluster c while q is in cluster d.

2.4 System Load Metrics

The following six metrics define values that determine whether the overall system load is balanced:

– **Processor Workload** ($\texttt{QWgt}_p$) is the total cost to process all the vertices assigned to processor p and is given by $\texttt{QWgt}_p = \sum_{v \in p} (\texttt{Wgt}_p^v + \texttt{Comm}_p^v + \texttt{Remap}_p^v)$.

– **Queue Length** ($\texttt{QLen}_p$) is the total number of vertices assigned to p.

– **Total System Load** (QWgtTOT) is the sum of $\texttt{QWgt}_p$, over all P processors.

– **Average Load** (WSysLL) is QWgtTOT / P.

– **Heaviest Processor Load** (MaxQWgt) is the maximum value of $\texttt{QWgt}_p$ over all processors, and indicates the total time required to process the application.

– **Load Imbalance Factor** (LoadImb) represents partitioning quality, and is given by the ratio MaxQWgt / WSysLL.

2.5 Partitioning Metrics

These metrics are used by MinEX to make partitioning decisions:

– Gain represents the change in QWgtTOT that would result from a proposed vertex reassignment. A negative value indicates that less processing is required after such a reassignment. The partitioning algorithm favors vertex migrations with negative or small Gain values that reduce or minimize the overall system load.

– Var is computed using the workload ($\texttt{QWgt}_p$) for each processor p and the average system load (WSysLL) in accordance with the formula $\texttt{Var} = \sum_p (\texttt{QWgt}_p - \texttt{WSysLL})^2$. Basically, it is the variance in processor workloads. The objective is to initiate vertex moves that lower this value. Since individual terms of this formula with large values correspond to processors that are most out of balance, minimizing Var will tend to bring the system into better load balance. ΔVar is the change in Var after moving a vertex from one processor to another; a negative value indicates a reduction in variance.

3 The MinEX Partitioner

The MinEX partitioner was originally introduced in [7]. In this paper, we present an overview of MinEX and introduce refinements that we have made since that previous publication. Specifically, a *filter function* described in Section 3.3 has been added to speed up MinEX execution. The strategy for choosing vertex reassignments has also been modified to accommodate this filter function and is described in Section 3.2. Finally, the interface between the user application and MinEX is described in Sections 3.4 and 3.5.

3.1 Overview

MinEX [7] can execute either in a diffusive manner [5] where an existing partition is used as a starting point or it can create partitions from scratch [17]. The entire partitioning process occurs in three steps: contraction, partitioning, and refinement, similar to other multilevel partitioners. However, MinEX is unique in that it redefines the partitioning goal to minimizing `MaxQWgt` rather than balancing partition workloads and reducing the total edge cut. In addition, MinEX allows applications to provide a function to achieve latency tolerance, if available. This user-defined function is described in Section 3.5.

The first step in MinEX is to sequentially contract the graph one vertex at a time instead of repeatedly contracting it in halves as is common with other multilevel partitioners. The advantage of this approach is that a decision can be made each time a vertex is later refined as to whether it should be assigned to another processor, making the algorithm more flexible. If $|V|$ is the number of vertices in the graph, contraction requires $O(|V|)$ steps which is asymptotically equal to the complexity of contracting it sequentially in halves. Once the graph is contracted, the remaining vertices (metavertices) are reassigned according to the criteria followed by the partitioning algorithm. Finally, the graph is expanded back to its original form through the refinement process. MinEX maintains pairs of merged vertices in a stack so refinement of vertices proceeds in reverse order from the contraction. During reassignment, as each metavertex is refined, a decision is made as to whether it should be reassigned. A metavertex reassignment essentially migrates all of the vertices that the metavertex represents.

3.2 Partitioning Criteria

Partitioning involves reassigning vertices from overloaded processors (where $\texttt{QWgt}_p > \texttt{WSysLL}$) to underloaded processors (where $\texttt{QWgt}_p < \texttt{WSysLL}$). To facilitate reassignment decisions, MinEX maintains a list of processors sorted by $\texttt{QWgt}_p$ values that is updated after each vertex reassignment. Because only a small subset of processors change positions in this list after a vertex reassignment (3 or 4 of 32 processors in our

experiments), the overhead associated with maintaining this list is acceptable. Any vertex reassignment that projects a negative ΔVar value is executed. We call this the *basic partitioning criteria.*

3.3 Reassignment Filter

The most computationally expensive part of MinEX is the requirement that each adjacent edge must be considered to determine the impact of potential reassignments. To minimize this overhead, we have added a *filter function* to heuristically estimate the effect of a vertex reassignment. Reassignments that pass through the filter are then further considered in accordance with the basic partitioning criteria described in Section 3.2. The filter utilizes edge outgoing and incoming communication totals that are maintained with each vertex to estimate QWgt values for the source and destination processors ($\texttt{newQWgt}_{from}$ and $\texttt{newQWgt}_{to}$, respectively). Using these values, the pseudo code shown in Figure 1 is executed to decide whether a potential vertex reassignment should be accepted.

If $\texttt{newQWgt}_{from} > \texttt{QWgt}_{from}$ **Reject Assignment**

If $\texttt{newQWgt}_{to} < \texttt{QWgt}_{to}$ **Reject Assignment**

$$\Delta\texttt{Var} = (\texttt{newQWgt}_{from} - \texttt{WSysLL})^2 + (\texttt{newQWgt}_{to} - \texttt{WSysLL})^2$$
$$-(\texttt{QWgt}_{from} - \texttt{WSysLL})^2 - (\texttt{QWgt}_{to} - \texttt{WSysLL})^2$$

If $\Delta\texttt{Var} \geq 0$ **Reject Assignment**

$$\texttt{newGain} = \texttt{newQWgt}_{from} + \texttt{newQWgt}_{to} - \texttt{QWgt}_{from} - \texttt{QWgt}_{to}$$

If $\texttt{newGain} > 0$ **And** $\texttt{newGain}^2/ - \Delta\texttt{Var} > \texttt{ThroTTle}$ **Reject Assignemt**

If $fabs(\texttt{newQWgt}_{from} - \texttt{newQWgt}_{to}) > fabs(\texttt{QWgt}_{from} - \texttt{QWgt}_{to})$

  If $\texttt{newQWgt}_{from} < \texttt{QWgt}_{to}$ **Reject Assignment**

  If $\texttt{newQWgt}_{to} > \texttt{QWgt}_{from}$ **Reject Assignment**

Assignment Passes Filter

Figure 1: Pseudo code to determine promising vertex reassignments.

The reassignment filter is designed to minimize increases in projected runtime (MaxQWgt). It also rejects reassignments that project a positive ΔVar value. Increases in projected Gain is controlled by the ThroTTle parameter. Essentially, ThroTTle acts as a gate to prevent reassignments that cause excessive increases in Gain. The Gain metric is squared because Var is also of second order. A low ThroTTle could prevent MinEX from finding a balanced partitioning allocation, while a high value could converge to a point where

runtime is unacceptable.

Table 1: Runtime comparisons for 64K bodies and different grid configurations using various `ThroTTle` values

	$P = 32$		$P = 64$		$P = 128$	
ThroTTle	$I = 10$	$I = 100$	$I = 10$	$I = 100$	$I = 10$	$I = 100$
0	3345	5449	1708	3100	843	1619
1	3341	5134	1715	3100	845	1619
2	3301	4906	1639	3004	845	1619
4	3180	4780	1671	2968	848	1614
8	3183	4937	1688	2759	839	1554
16	3182	4908	1642	2688	847	1484
32	3184	4882	1625	2705	822	1438
64	3183	4918	1627	2693	821	1411
128	3183	4922	1627	2707	817	1426
256	3183	4836	1626	2714	820	1433
512	3183	4820	1628	2711	818	1430

Table 1 demonstrates the effect on runtimes (shown in thousands in units) using different `ThroTTle` values in our experiments. The columns labeled $I = 10$ and $I = 100$ refer to grid configurations with interconnect slowdowns of 10 and 100, respectively. The table shows results for grids with 32, 64, and 128 processors. Based on these tests, the experiments that we present in Section 5 use a `ThroTTle` value of 32.

Table 2 demonstrates the effectiveness of the reassignment filter for 8, 128, and 1024 processors. We show the total number of vertex assignments considered (`Total`), the number of assignments that passed through the filter (`Accepted`), and the number of potential reassignments that subsequently failed the basic partitioning criteria described in Section 3.2 (`Failed`). The partition graph represents N-body problems consisting of 16K, 64K, and 256K bodies. The results clearly demonstrate that the reassignment filter eliminates almost all of the edge processing overhead associated with reassignments that are rejected. For example, for 128 processors and 256K bodies, a total of 4608 of 51876 potential vertex reassignments passed through the filter. Only one of these potential reassignments were subsequently rejected.

3.4 Application Interface

Application programs supply the partition graph and the configuration graph to MinEX. The partition graph is represented by arrays that represent the processing weights, redistribution weights, and edge weights. The configuration graph is created by a stand-alone utility program. The MinEX function signature is similar to

Table 2: Filter effectiveness for 16K, 64K, and 256K bodies processed by 8, 128, and 1024 processors

	16K n-bodies			64K n-bodies			256K n-bodies		
P	Total	Accepted	Failed	Total	Accepted	Failed	Total	Accepted	Failed
8	6011	110	0	14991	212	0	25183	222	0
128	19192	2562	0	49082	5240	4	51876	4608	1
1024	18555	2790	7	23986	6569	4	35605	12639	2

that used by METIS and is shown below:

```
void MinEX_PartGraph (int vertices, int *adjcncy, int *cwgt, int *ewgt,
                      int *vadj, int *vwgt, int *rwgt, int *vown, int *part,
                      Ipg *ipg, User *user)
```

where

`vertices`= number of nodes in the partitioning graph,

`adjcncy` = adjacency list of vertices,

`cwgt` = outgoing edge weights ($\texttt{CWgt}_{(v,w)}$),

`ewgt` = incoming edge weights ($\texttt{CWgt}_{(w,v)}$),

`vadj` = initial integer offsets into adjcncy, cwgt, and ewgt for each vertex,

`vwgt` = processing weights ($\texttt{Wgt}_v$) of each vertex,

`rwgt` = redistribution weights ($\texttt{RWgt}_v$) of each vertex,

`vown` = original processor assignment for each vertex,

`part` = partition computed by MinEX and returned to the user,

`ipg` = grid configuration graph, and

`user` = user-supplied options containing:

(a) `ThroTTle` value,

(b) whether application latency tolerance function is provided,

(c) number of vertices that the contracted graph should contain,

(d) whether the partitioning is to be diffusive or be from scratch, and

(e) whether duplicate transmissions of edges are avoided by the application.

3.5 Latency Tolerance

MinEX interacts with a user-defined function (called MinEX_LatTol), if one is supplied, to account for possible latency tolerance that can be achieved by the application. This is a novel approach to partitioning

that is not employed by existing partitioners, including METIS. The calling signature of this function is as follows:

```
double MinEX_LatTol (User *user, Ipg *ipg, QTot *tot)
```

where

`user` = user-supplied options originally passed to `MinEX_PartGraph`,

`ipg` = grid configuration originally passed to `MinEX_PartGraph`, and

`tot` = projected totals computed by MinEX for a particular processor and contains:

(a) p, the processor to which this call applies,

(b) $\texttt{Pproc}_p = \sum_{v \in p} \texttt{Wgt}_p^v$, the total processing weight,

(c) $\texttt{Crcv}_p = \sum_{v \in p} \texttt{Comm}_p^v$, the cost associated with data communication,

(d) $\texttt{Rrcv}_p = \sum_{v \in p} \texttt{Remap}_p^v$, the transmission cost associated with data relocation, and

(e) $\texttt{QLen}_p$, the total number of vertices assigned to p.

The function utilizes these quantities to compute the projected value of $\texttt{QWgt}_p$, that is returned to the partitioner. The projected value differs from the $\texttt{QWgt}_p$ definition given in Section 2.4 because some of the processing is overlapped with communication.

4 N-body Application

The N-body application is the classical problem of simulating the movement of a set of bodies based upon gravitational or electrostatic forces. Many applications in the fields of astrophysics, molecular dynamics, computer graphics, and fluid dynamics can utilize N-body solvers. The basic solution involves calculating the velocity and position of N bodies at discrete time steps, given their initial positions and velocities. At each step, there are N^2 pairwise interactions of forces between bodies.

Of the many N-body solution techniques that have been proposed, the Barnes & Hut algorithm [3] is perhaps the most popular. The approach is to approximate the force exerted on a body by a cell of bodies that is sufficiently distant using the center of mass and the total mass in the remote cell. In this way, the number of force calculations can be significantly reduced. The first step is to recursively build a tree of cells in which the bodies are grouped by their physical positions. A cell v is considered close to another cell w if the ratio of the distance between the two furthest bodies in v to the distance between the centers of mass of v and w is less than a specified parameter δ. In this case, all the bodies in v must perform pairwise force calculations with each body in w. However, if w is far from v, cell w is treated as a single body using its total mass and center of mass for force interaction calculations with the bodies of v. An example of a

parallel Barnes & Hut implementation using message passing is described in [22]; this was later refined in [16]. In this paper, we modify the basic Barnes & Hut approach to construct a novel graph-based model of the N-body problem to integrate the application with MinEX and METIS. We then run the N-body solver to directly compare the runtime effects of both partitioning schemes in a distributed grid environment.

4.1 Overall Framework

The pseudo code in Figure 2 gives an overview of the framework for implementing the N-body application. At each time step, a new or modified tree of cells is recursively constructed to allocate the bodies to cells. Either MinEX or METIS is then invoked to balance the load among the available processors of the grid. The solver then computes the forces, and updates the position and velocity of each of the bodies. Relevant statistical and visualization information are provided at the end of each time step. The entire cycle is repeated for the desired number of time steps.

```
Output Initial N-body Visualization Data
Do
    Create or Modify the Barnes & Hut N-body Tree
    Call Partitioner
    Call N-body Solver
    Output N-body runtime Statistics
    Output N-body Visualization Data
Until all time steps are processed
```

Figure 2: Framework for implementing the N-body application.

4.2 Tree Creation

The first step in solving the N-body problem is to recursively build an octree of cells. The process begins by inserting bodies into an initial cell until it contains `CellMax` number of bodies. This parameter is set to minimize the number of calculations required by the solver to compute the body forces. Before the next body can be inserted, this cell is split into eight octants. Each of these eight smaller cells contain the previously-inserted bodies based on their centers of mass. Insertion of bodies into this tree continues until one of the cells have more than `CellMax` bodies. This cell is then further subdivided into eight octants, and the process continues. Naturally, all the bodies reside in the leaves of the octree. Figure 3 illustrates this concept: both

the spatial and tree representations are shown. The cell's center of mass is used for subsequent searches of the tree. Traversal direction is determined by the octant where a body resides relative to this center of mass.

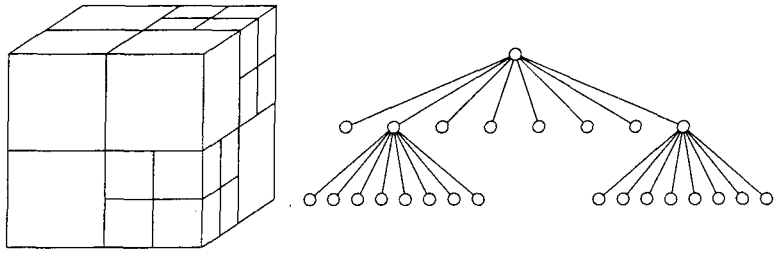

Figure 3: A three-level octree and the corresponding spatial representation.

4.3 Partition Graph Construction

When the tree creation phase of the Barnes & Hut algorithm is finished, a graph G is constructed. This graph is presented to the MinEX and METIS partitioners to balance the load among the available processors. However, for METIS to execute successfully, G must be somewhat modified to another graph G_M (described later in Section 4.4). For direct comparisons between the two partitioners, experiments are conducted with the modified graph G_M.

Each vertex v of G corresponds to a leaf cell C_v (of $|C_v|$ bodies) in the N-body octree and has two weights, $\texttt{PWgt}_v$ and $\texttt{RWgt}_v$. Each defined edge (v, w) has one weight, $\texttt{CWgt}_{(v,w)}$. These weights (described in Section 2.1) model the processing, data remapping, and communication costs incurred when the solver processes C_v. The total time required to process the vertices assigned to a processor p must take into account all three metrics. Their values are set in accordance with the formulae below:

- $\texttt{PWgt}_v = |C_v| \times (|C_v| - 1 + \texttt{Close}_v + \texttt{Far}_v + 2)$ is the number of computations that are executed by the solver to calculate new positions of the bodies residing in C_v. Here $\texttt{Close}_v$ is the number of bodies in cells close to C_v and $\texttt{Far}_v$ is the number of cells that are far from C_v. The 2 in the equation represents the double integration of acceleration that is performed to arrive at body postions at the next time step once the affect of gravitational forces are determined.

- $\texttt{RWgt}_v$ defines the cost of relocating cell C_v from one processor to another. Thus, $\texttt{RWgt}_v = |C_v|$, since each of the bodies in C_v must be migrated.

- $\texttt{CWgt}_{(v,w)}$ represents the communication cost when cell C_v is close to another cell C_w. In this case, the mass and position of each body in C_w must be transmitted to the processor to which C_v is assigned. Thus, $\texttt{CWgt}_{(v,w)} = |C_w|$ if C_w is close to C_v; otherwise, it is 0.

Note that the edge $(v, w) \in G$ only if either C_v is close to C_w or vice-versa. Also, G is a directed graph because $\texttt{CWgt}_{(v,w)} \neq \texttt{CWgt}_{(w,v)}$ if $|C_v| \neq |C_w|$, or whenever C_w is close to C_v but C_v is far from C_w. We do not model the cost to communicate the C_w center of mass when C_w is far from C_v because each processor contains the tree of internal nodes making these communications unnecessary.

4.4 Graph Modifications for METIS

The METIS partitioner has two limitations that must be addressed before its performance can be directly compared to that of MinEX. First, METIS does not allow zero edge weights; second, it is unable to process directed graphs. Zero edge weights occur in N-body partition graphs because cell C_v being close to cell C_w does not necessarily imply that C_w is close to C_v. N-body graphs are directed because edge (v, w) has a weight equal to the number of bodies in cell C_v whereas edge (w, v) has a weight equal to the number of bodies in cell C_w. These quantities are not always equal.

To accommodate these two limitations in METIS, a modified graph G_M is generated that is usable by both partitioning schemes. G_M differs from G in its edge weights: $\texttt{CWgt}_{(v,w)} = \max(|C_v|, |C_w|)$ for all edges (v, w). This guarantees that the edges in G_M have positive weights, and that $\texttt{CWgt}_{(v,w)} = \texttt{CWgt}_{(w,v)}$.

4.5 Solution Algorithm

The force between two bodies (and/or cells if they are far enough) are calculated using Newtonian gravitational formulae. For the sake of completeness, these formulae are enumerated below:

- The position vector $p^b = \langle x^b, y^b, z^b \rangle$ represents the location of body b.
- The distance scalar $r(b, c) = \sqrt{(x^b - x^c)^2 + (y^b - y^c)^2 + (z^b - z^c)^2}$ is the Euclidean distance between bodies b and c.
- The gravitational force between bodies in the x direction is given by $F_x(b, c) = Gm^b m^c(x^b - x^c) \ / \ (r(b, c))^3$. Here, G is the gravitational constant, while m^b and m^c indicate the body masses of b and c, respectively. Note that a small smoothing constant is added to $r(b, c)$ to prevent division by zero. The forces between bodies in the y and z directions are similarly defined.
- The acceleration vector $a^b = \langle a_x^b, a_y^b, a_z^b \rangle$ for b is computed using the formula $F = ma$. Acceleration is then integrated to compute the velocity vector $v^b = \langle v_x^b, v_y^b, v_z^b \rangle$. A second integration is performed on v^b to compute the position of b at the next time step. All integrations use a leap-frog method. For example, if p_n^b, v_n^b, and a_n^b respectively indicate the position, velocity, and acceleration of b at time step n, and Δt is the size of the time step, the position and velocity of b at time step $(n + 1)$ are:

  $$v_{n+1/2}^b = v_n^b + a_n^b \times \Delta t/2,$$

$$p^b_{n+1} = p^b_n + v^b_{n+1/2} \times \Delta t,$$
$$v^b_{n+1} = v^b_{n+1/2} + a^b_{n+1} \times \Delta t/2.$$

4.6 Parallel Implementation

We have implemented the N-body solver using a message passing model. Each processor contains the internal nodes of the Barnes & Hut tree so that excessive communication between processors is avoided. The pseudo code in Figure 4 indicates solver execution by each processor. The processing steps are designed so that the application can minimize the deleterious effects of low bandwidth. Basically, processors distribute data sets and communication information as early as possible so that computation can be overlapped with communication.

All to All Broadcast of changes to the internal nodes of the Barnes & Hut tree
Relocate all n-bodies that are to be reassigned based on the computed partition
For each data set that is relocated to this processor
    Unpack and store the data
    Calculate force interactions between all local close cells
For each body assigned to this processor
    Transmit body and cell position and mass data of remote close bodies
For each body assigned to this processor
    Calculate force interactions with local far cells
While more position and mass data remain to be received
    Receive position and mass communication data
    Calculate force interactions using data received
For each body assigned to this processor
    Integrate to determine new body position

Figure 4: Pseudo code for the N-body solver on each processor.

A reduction in communication is achieved by recognizing that position data need not be obtained for the bodies that have been relocated away from a processor during the time step. This is because the solver maintains position information for cells that were relocated in the current time step. MinEX automatically accommodates this optimization without special user interface logic.

5 Experimental Study

In our experiments, we simulate a grid environment in which the N-body solver is executed. Our simulator models communication using message passing primitives similar to those implemented in mpi (http://www-unix.mcs.anl.gov/mpi). The grid environment is modeled using discrete time simulation and uses the grid configuration graph (defined in Section 3.4) to account for latency and bandwidth. Experimental test cases with 16K, 64K, and 256K bodies are considered that model two neighboring Plummer galaxies that are about to merge [20]. The partition graphs for these test cases respectively contain 4563, 8091, and 14148 vertices, and 99802, 159496, and 236338 edges.

Graphs labeled G in the following tables refer to the directed graph described in Section 4.3 and are used only by MinEX. Graphs labeled G_M are undirected as described in Section 4.4 to accommodate the requirements of METIS. Both MinEX and METIS are run on G_M to obtain direct comparisons between the two partitioning schemes. The METIS k-way partitioner is used selecting its option to minimize edge cuts.

The grid configuration graph is varied to evaluate performance over a wide spectrum of heterogeneous grid environments. The total number of processors (P) varies between 8 to 1024 depending on the experiment. The number of clusters (C) is either 4 or 8, while interconnect slowdowns (I) are 10 or 100. We always assume a constant value of I for communication within clusters as it is typically true for real geographically distributed grids.

Three configuration types (**HO**, **UP**, and **DN**) are used in our experiments. The **HO** configurations assume that all processors are homogeneous and grouped evenly among the clusters with intra-connect and processing slowdown factors of unity. The ones labeled **UP** assume that processors in cluster i have intra-connect and processing slowdown factors of $2i - 1$. Therefore, processors in clusters with lower ids have greater communication and processing capability than those with higher ids. Finally, the configurations labeled **DN** assume that the processors in cluster i have intra-connect slowdown factors of $2C - 1 - 2i$ and processing slowdown factors of $2i - 1$. These configurations assume that clusters with lower ids have greater communication capability but lesser processing power than clusters with higher ids. This spectrum of configurations allows us to consider a variety of heterogeneous grid characteristics. The results of our experiments are presented in the following subsections.

5.1 Multiple Time Step Test

This set of experiments determines whether running multiple time steps are likely to significantly impact the overall performance. Table 3 presents runtimes (in thousands of units) and `LoadImb` when executing 1 and

50 time steps. MinEX is run with both the G and G_M graphs, METIS is run only with the G_M graph, and both partitioners are invoked before the solver executes each time step. The partition graph represents 16K bodies; the configuration type is **UP**; while interconnect slowdowns, number of processors, and number of clusters are respectively set as $I = 10$, $P = 64$, and $C = 8$. Results show that running multiple time steps have little impact. Our subsequent experiments therefore execute only a single time step.

Table 3: Performance for 1 and 50 time steps

	1 time step		50 time steps	
Type	Runtime	LoadImb	Runtime	LoadImb
MinEX$-G$	401	1.03	388	1.01
MinEX$-G_M$	413	1.05	398	1.02
METIS$-G_M$	1630	2.16	1534	2.03

5.2 Scalability Test

The purpose of the next test is to determine how the application scales with the number of processors. We process graphs representing 16K, 64K, and 256K bodies using the **UP** configuration containing between 8 and 1024 processors that are distributed among 8 clusters. Table 4 reports runtimes (in thousands of units) and shows that the application scales well to 128 processors. As a result, our subsequent load balance comparison tests are conducted only for $P = 32$, $P = 64$, or $P = 128$.

Table 4: Runtime (in thousands of units) comparisons for varying numbers of processors

		Number of processors P							
Bodies	Graph Type	8	16	32	64	128	256	512	1024
16K	MinEX$-G$	2792	1445	760	401	204	124	94	268
	MinEX$-G_M$	2867	1466	780	413	206	124	94	268
	METIS$-G_M$	10384	5330	2919	1630	1619	1395	847	1209
64K	MinEX$-G$	12035	6172	3184	1625	822	716	355	326
	MinEX$-G_M$	12085	6233	3217	1656	837	716	355	268
	METIS$-G_M$	43785	23235	11738	6150	3110	3113	3735	3664
256K	MinEX$-G$	78297	39335	20038	10187	5113	2917	1756	1310
	MinEX$-G_M$	78396	39448	20193	10183	5174	2927	1767	1310
	METIS$-G_M$	301573	151983	76017	38790	19734	9901	5071	5220

Table 5: MinEX and METIS partitioning speed (in seconds)

		Number of processors P							
Bodies	Graph Type	8	16	32	64	128	256	512	1024
16K	MinEX$-G$	0.17	0.20	0.23	0.33	0.53	1.09	1.58	2.36
	MinEX$-G_M$	0.18	0.20	0.23	0.32	0.53	1.13	1.51	2.39
	METIS$-G_M$	0.16	0.23	0.35	1.02	1.05	1.46	1.81	2.88
64K	MinEX$-G$	0.31	0.33	0.40	0.59	1.00	1.93	3.09	4.93
	MinEX$-G_M$	0.35	0.37	0.39	0.58	1.05	1.99	3.09	4.73
	METIS$-G_M$	0.21	0.22	0.45	0.60	1.55	1.82	2.32	3.42
256K	MinEX$-G$	0.48	0.53	0.57	0.71	1.08	2.27	5.37	9.08
	MinEX$-G_M$	0.50	0.55	0.55	0.69	1.08	2.30	5.88	9.17
	METIS$-G_M$	0.43	0.49	0.59	0.76	1.20	2.57	3.18	4.18

5.3 Partitioner Speed Comparisons

In this section, we compare the MinEX partitioning speed to that of METIS. For these experiments, P is varied between 8 and 1024 with partition graphs representing 16K, 64K, and 256K bodies. The **UP** configuration is used with $C = 8$ and $I = 10$. Results in Table 5 show that MinEX executes faster than METIS in the majority of cases. For example, if $P = 128$, MinEX outperforms METIS on all graph sizes. However, METIS has a clear advantage when processing the 256K case with $P = 512$ or $P = 1024$. In general though, we can conclude that MinEX is at least competitive with METIS in execution speed.

5.4 Partitioner Quality Comparisons

In this section, we present three tables to extensively compare the quality of partitions generated by MinEX and METIS. Tables 6, 7, and 8 show N-body application runtimes and `LoadImb` results for graphs representing 16K, 64K, and 256K bodies. Each table contains results of runs using the **UP**, **HO**, and **DN** grid configuration types. The number of processors is varied between 32 and 128, the number of clusters is 4 or 8, and the interconnect slowdowns are set to 10 or 100.

5.4.1 16K Bodies

Table 6 presents results of runs using partition graphs representing 16K bodies. These results show that MinEX has a significant advantage over METIS in the heterogeneous **UP** and **DN** configuration types. For example, if $P = 64$ and $C = 8$, MinEX shows an improvement in runtime by a factor of 4. If $P = 128$, the advantage increases to a factor of 6. In both cases, the improvement in load balance is also very significant.

Results for the homogeneous graph **HO** are less conclusive. Here, METIS is competitive with MinEX; however, in several cases, MinEX still has a significant advantage (e.g., $P = 64$, $C = 8$). These results are not surprising given that MinEX's strategy to minimize the edge cut should be effective in homogeneous configurations. Note that MinEX running with the graph G is in general superior to MinEX running with the graph G_M. This result is somewhat expected because G models the actual solver more closely than G_M does. However, MinEX using G_M still has similar advantages over METIS as we have just discussed. One final observation is that MinEX has a significant but smaller advantage over METIS for configurations where the interconnect slowdown is greater ($I = 100$). This is because as interconnect slowdowns increase, the communication overhead begins to dominate the application. The differences in intra-connect communication and processing speeds therefore become less significant and in effect the network becomes more homogeneous. Perhaps if MinEX is refined to put a greater focus on achieving a minimum communication cut, it could retain more of its advantage over METIS in these cases.

5.4.2 64K Bodies

The experiments shown in Table 7 are the same as those presented in Section 5.4.1 but with the larger partition graph representing 64K bodies. The results are very similar to those shown in Table 6 but with a few surprising differences. For example, some of the MinEX results with the **DN** configuration are worse than the corresponding results with the **UP** type. For example, when $P = 128$ and $C = 4$, MinEX$-G$ shows a runtime of 998 with **UP** but 1489 with **DN**. It is interesting to note that the MinEX partitioner estimated a runtime of 975 and a load balance of 1.0001 in the **DN** case. The discrepancy is explained in that processors incur excessive idle time when processing the application with the **DN** configuration so that the partitioner estimates are not realized. This illustrates a potential problem area in the use of partitioners for solving grid-based applications. Even if communication costs can be exactly predicted, the dynamics of the application can still result in unexpected idle time. To further investigate this problem, we modified our simulator to accommodate multiple i/o channels at each processor (the original version assumed that each process has only one i/o channel). With two channels per processor, the solver executed the application with a runtime of 975; exactly as MinEX estimated. With four input channels per processor, the runtime was 973; a minimal additional improvement. However, no such improvements were obtained when using METIS.

5.4.3 256K Bodies

The results shown in Table 8 are from comparison experiments on partition graphs representing 256K bodies. Performance with the **UP** and **DN** configuration graphs are consistent with those presented in Sections 5.4.1

and 5.4.2; however, the **HO** experiments produced additional surprises. Here, METIS has a clear advantage when $P = 32$ or $P = 64$, and $I = 100$. When investigating these cases, we discovered that MinEX is converging very tightly to a estimated partition (`LoadImb` of 1.0001) but is converging at too high a value. Evidently, the partitioning criteria for vertex reassignments needs to be refined to prevent this situation. This is an open research area that needs to be addressed if MinEX (or any other grid-based partitioner) is to be successfully utilized as a general purpose tool.

6 Conclusions

In this paper, we have used the classical N-body application to evaluate our latency-tolerant partitioner, called MinEX, designed specifically for heterogeneous distributed computing environments such as the NASA Information Power Grid (IPG). The MinEX design has significant advantages over those of traditional partitioners. For example, its partitioning goal to minimize application runtimes, its ability to map applications onto heterogeneous grid configurations, and its interface to application latency tolerance information make it well suited for grid environments. In addition, MinEX is also able to partition directed graphs with zero edge weights (which occur in graphs modeling N-body problems); a distinct advantage over popular state-of-the-art partitioners such as METIS.

Using a solver that we developed for the N-body problem, we compared the performance of MinEX to METIS to determine whether actual results match theoretical expectations. Extensive experimental results showed that while MinEX produces partitions of comparable quality to those by METIS on homogeneous grids, it improves application runtimes by a factor of six on some heterogeneous configurations. The experiments demonstrate the feasibility and benefits of our approach to map application partition graphs onto multiprocessor grid environments, and to incorporate latency tolerance techniques directly into the partitioning process. The experiments also reveal issues that need to be addressed if a general grid-based partitioning tool is to be realized. For example, the number of i/o channels per processor affects the actual runtime and load balance that is achieved by the application because the resulting idle time is directly affected. Furthermore, additional schemes for reassigning vertices in a grid-based environment need to be explored so that consistent results can be achieved in all grid-based configurations. We are actively investigating these refinements. Additional performance studies are also being considered.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: A tool for performing parametised simulations using distributed workstations," *4th IEEE Symposium on High Performance Distributed Computing*, (1995) 112–121.

[2] C.J. Alpert and A.B. Kahng, "Recent directions in netlist partitioning: A survey" *Integration, the VLSI Journal*, 19 (1995) 1–81.

[3] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ force calculation algorithm," *Nature*, 324 (1986) 446–449.

[4] H. Casanova and J. Dongarra, "NetSolve: A network-enabled server for solving computational science problems," *International Journal of Supercomputer Applications*, 11 (1997) 212–223.

[5] G. Cybenko, "Dynamic load balancing for distributed-memory multiprocessors," *Journal of Parallel and Distributed Computing*, 7 (1989) 279–301.

[6] J. Czyzyk, M.P. Mesnier, and J.J. Moré, "The network-enabled optimization system (NEOS) server," Preprint MCS-P615-1096, Argonne National Laboratory, 1997.

[7] S.K. Das, D.J. Harvey, and R. Biswas, "MinEX: A latency-tolerant dynamic partitioner for grid computing applications," *Future Generation Computer Systems*, 18 (2002) 477–489.

[8] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.

[9] Globus Project, See URL *http://www.globus.org*.

[10] A.S. Grimshaw and W.A. Wulf, "The Legion vision of a worldwide computer," *Communications of the ACM*, 40 (1997) 39–45.

[11] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[12] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," Technical Report 96-036, University of Minnesota, 1996.

[13] S. Kumar, S.K. Das and R. Biswas, "Graph partitioning for parallel applications in heterogeneous grid environments," *16th International Parallel and Distributed Processing Symposium*, 2002.

[14] J. Leigh, A.E. Johnson, and T.A. DeFanti, "CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments," *Virtual Reality Research, Development and Applications*, 2 (1997) 217–237.

[15] M.J. Litzdow, M. Livny, and M.W. Mutka, "Condor — a hunter of idle workstations," *8th International Conference on Distributed Computing Systems*, (1988) 104–111.

[16] P. Liu and S. Bhatt, "Experiences with parallel N-body simulations," *6th ACM Symposium on Parallel Algorithms and Architectures*, (1988) 122–131.

[17] L. Oliker and R. Biswas, "Parallelization of a dynamic unstructured algorithm using three leading programming paradigms," *IEEE Transactions on Parallel and Distributed Systems*, 11 (2000) 931–940.

[18] H. Shan, J.P. Singh, L. Oliker, and R. Biswas, "A comparison of three programming models for adaptive applications on the Origin2000," *Journal of Parallel and Distributed Computing*, 62 (2002) 241–266.

[19] H. Shan, J.P. Singh, L. Oliker, and R. Biswas, "Message passing and shared address space parallelism on an SMP cluster," *Parallel Computing*, 29 (2003) 167–186.

[20] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy, "Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity," *Journal of Parallel and Distributed Computing*, 27 (1995) 118–141.

[21] C. Walshaw, M. Cross, and M. Everett, "Parallel dynamic graph partitioning for adaptive unstructured meshes," *Journal of Parallel and Distributed Computing*, 47 (1997) 102–108.

[22] M.S. Warren and J.K. Salmon, "A parallel hashed oct-tree N-body algorithm," *Supercomputing '93*, (1993) 12–21.

Table 6: Runtime (in thousands of units) comparisons for 16K bodies

				UP		HO		DN	
P	C	I	Type	Runtime	LoadImb	Runtime	LoadImb	Runtime	LoadImb
32	4	10	MinEX$-G$	459	1.00	197	1.02	461	1.01
			MinEX$-G_M$	479	1.04	206	1.06	480	1.05
			METIS$-G_M$	1362	1.88	196	1.07	1362	1.88
32	4	100	MinEX$-G$	1023	1.07	921	1.02	1046	1.07
			MinEX$-G_M$	1157	1.22	1167	1.25	1196	1.21
			METIS$-G_M$	1363	1.74	1167	2.49	1363	1.73
32	8	10	MinEX$-G$	760	1.01	197	1.02	763	1.01
			MinEX$-G_M$	780	1.04	211	1.09	792	1.04
			METIS$-G_M$	2919	2.01	196	1.07	2919	2.01
32	8	100	MinEX$-G$	1347	1.03	1198	1.02	1371	1.03
			MinEX$-G_M$	1562	1.17	1417	1.24	1574	1.17
			METIS$-G_M$	2920	1.91	1182	1.67	2920	1.90
64	4	10	MinEX$-G$	234	1.01	106	1.08	235	1.04
			MinEX$-G_M$	245	1.05	109	1.11	245	1.04
			METIS$-G_M$	761	2.01	108	1.04	761	1.04
64	4	100	MinEX$-G$	620	1.10	450	1.20	634	1.11
			MinEX$-G_M$	737	1.28	612	1.45	746	1.27
			METIS$-G_M$	763	1.72	621	2.12	763	1.73
64	8	10	MinEX$-G$	401	1.03	109	1.11	372	1.01
			MinEX$-G_M$	413	1.05	115	1.15	421	1.06
			METIS$-G_M$	1630	2.16	108	1.04	1630	1.04
64	8	100	MinEX$-G$	794	1.05	549	1.03	798	1.04
			MinEX$-G_M$	936	1.22	685	1.39	946	1.20
			METIS$-G_M$	1632	2.01	841	2.01	1632	2.00
128	4	10	MinEX$-G$	121	1.03	94	1.80	194	1.11
			MinEX$-G_M$	122	1.03	94	1.44	194	1.15
			METIS$-G_M$	755	3.98	131	1.05	917	4.60
128	4	100	MinEX$-G$	353	1.32	426	1.29	493	1.22
			MinEX$-G_M$	425	2.24	408	1.40	482	1.20
			METIS$-G_M$	1632	2.01	841	2.01	1632	2.00
128	8	10	MinEX$-G$	204	1.04	94	1.80	196	1.20
			MinEX$-G_M$	206	1.05	94	1.55	217	1.05
			METIS$-G_M$	1619	4.27	131	1.05	1619	1.05
128	8	100	MinEX$-G$	465	1.17	428	1.23	476	1.18
			MinEX$-G_M$	519	1.23	633	1.82	678	1.65
			METIS$-G_M$	1619	1.05	604	3.19	1619	3.97

Table 7: Runtime (in thousands of units) comparisons for 64K bodies

				UP		HO		DN	
P	C	I	Type	Runtime	LoadImb	Runtime	LoadImb	Runtime	LoadImb
32	4	10	MinEX$-G$	1930	1.01	831	1.03	1933	1.01
			MinEX$-G_M$	1965	1.02	848	1.05	1974	1.03
			METIS$-G_M$	5574	1.82	802	1.05	5574	1.82
32	4	100	MinEX$-G$	3377	1.05	1529	1.06	3436	1.09
			MinEX$-G_M$	3812	1.18	1484	1.32	3843	1.18
			METIS$-G_M$	5775	1.80	1640	2.04	5775	1.80
32	8	10	MinEX$-G$	3184	1.00	828	1.02	3189	1.01
			MinEX$-G_M$	3217	1.02	851	1.05	3234	1.02
			METIS$-G_M$	11718	1.91	802	1.05	11738	1.71
32	8	100	MinEX$-G$	4822	1.08	1655	1.05	4916	1.05
			MinEX$-G_M$	5312	1.12	1634	1.03	5333	1.12
			METIS$-G_M$	11718	1.91	2018	1.72	11738	1.90
64	4	10	MinEX$-G$	979	1.00	417	1.02	981	1.01
			MinEX$-G_M$	1005	1.03	432	1.06	1004	1.03
			METIS$-G_M$	2878	1.25	425	1.09	2878	1.85
64	4	100	MinEX$-G$	1869	1.10	1042	1.03	1919	1.10
			MinEX$-G_M$	2297	1.28	1140	1.35	2299	1.27
			METIS$-G_M$	2879	1.81	1245	2.18	2879	1.81
64	8	10	MinEX$-G$	1625	1.01	419	1.02	1634	1.01
			MinEX$-G_M$	1656	1.03	430	1.06	1658	1.03
			METIS$-G_M$	6150	1.98	425	1.09	6150	1.98
64	8	100	MinEX$-G$	2705	1.07	1194	1.03	2716	1.06
			MinEX$-G_M$	3102	1.17	1308	1.29	3140	1.17
			METIS$-G_M$	6150	1.96	1434	1.47	6150	1.96
128	4	10	MinEX$-G$	498	1.19	265	1.24	854	1.10
			MinEX$-G_M$	520	1.17	265	1.24	848	1.09
			METIS$-G_M$	1478	1.55	276	1.54	1784	1.55
128	4	100	MinEX$-G$	998	1.14	854	1.38	1489	1.16
			MinEX$-G_M$	1177	1.31	983	1.20	1554	1.16
			METIS$-G_M$	1479	1.80	993	1.55	1784	1.55
128	8	10	MinEX$-G$	822	1.37	265	1.24	838	1.33
			MinEX$-G_M$	837	1.34	265	1.24	842	1.43
			METIS$-G_M$	3110	1.58	276	1.55	3110	1.56
128	8	100	MinEX$-G$	1438	1.11	1080	1.05	1436	1.10
			MinEX$-G_M$	1761	1.26	1330	1.34	1764	1.25
			METIS$-G_M$	3110	1.93	1068	2.54	3110	1.92

Table 8: Runtime (in thousands of units) comparisons for 256K bodies

				UP		HO		DN	
P	C	I	Type	Runtime	LoadImb	Runtime	LoadImb	Runtime	LoadImb
32	4	10	MinEX$-G$	12114	1.00	5137	1.01	12122	1.00
			MinEX$-G_M$	12186	1.01	5178	1.02	12223	1.01
			METIS$-G_M$	35474	1.79	5122	1.04	35474	1.79
32	4	100	MinEX$-G$	12344	1.04	7239	1.19	12355	1.04
			MinEX$-G_M$	14089	1.09	6929	1.17	14628	1.10
			METIS$-G_M$	35475	1.79	5132	1.04	37475	1.79
32	8	10	MinEX$-G$	20038	1.00	5177	1.02	20095	1.00
			MinEX$-G_M$	20193	1.01	5188	1.02	20220	1.01
			METIS$-G_M$	76017	1.92	5122	1.04	76017	1.92
32	8	100	MinEX$-G$	21173	1.06	7399	1.19	21634	1.05
			MinEX$-G_M$	23471	1.06	7541	1.23	24136	1.06
			METIS$-G_M$	76018	1.93	5199	1.05	76018	1.92
64	4	10	MinEX$-G$	6109	1.00	2590	1.01	6124	1.00
			MinEX$-G_M$	6158	1.01	2625	1.03	6172	1.01
			METIS$-G_M$	18102	1.82	2650	1.07	18102	1.82
64	4	100	MinEX$-G$	6627	1.09	4231	1.22	6616	1.07
			MinEX$-G_M$	8237	1.11	4131	1.28	8276	1.11
			METIS$-G_M$	18102	1.82	3334	1.33	18102	1.82
64	8	10	MinEX$-G$	10187	1.01	2590	1.02	10160	1.00
			MinEX$-G_M$	10183	1.01	2625	1.03	10222	1.01
			METIS$-G_M$	38379	1.95	2656	1.07	38379	1.95
64	8	100	MinEX$-G$	11459	1.08	4241	1.27	11982	1.06
			MinEX$-G_M$	13400	1.08	4894	1.20	13797	1.08
			METIS$-G_M$	38791	1.95	3842	1.51	38791	1.95
128	4	10	MinEX$-G$	3094	1.21	1384	1.03	4362	1.19
			MinEX$-G_M$	3119	1.21	1384	1.07	4357	1.21
			METIS$-G_M$	9209	1.23	1443	1.15	10105	1.94
128	4	10	MinEX$-G$	3654	1.27	2324	1.24	6545	1.20
			MinEX$-G_M$	4284	1.29	2464	1.42	5896	1.33
			METIS$-G_M$	9210	1.20	2337	1.79	10185	1.27
128	4	100	MinEX$-G$	5113	1.26	1353	1.07	5174	1.30
			MinEX$-G_M$	5174	1.29	1372	1.07	5174	1.27
			METIS$-G_M$	19734	1.20	1443	1.15	19374	1.96
128	8	100	MinEX$-G$	6160	1.07	2111	1.11	6620	1.09
			MinEX$-G_M$	7109	1.07	2326	1.22	7421	1.12
			METIS$-G_M$	19735	1.96	2337	1.74	19735	1.96