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 using simulated heterogeneous grid configurations. A solver for the classical N-body problem is implemented to provide a benchmark for the comparisons. Simulation results show that MinEX provides superior quality partitions while being competitive to METIS in speed of execution.

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, 3, 5, 8, 12, 13]. The Globus project [1], 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 [7].

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; however, the most successful ones are multilevel in nature [9, 10, 18] that contract the input 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 [14, 15, 16], little attention has been focused on heterogeneous grids till date. In [6], 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 [11]; however, latency tolerance was not considered.

This paper provides several important extensions to the work presented in [6]. Our major contributions are to (i) demonstrate the practical use of MinEX with an actual application solver, (ii) present details of MinEX interaction with the application to improve performance in high-latency low-bandwidth grid environments, and (iii) compare MinEX performance to that of a state-of-the-art partitioner and establish the effectiveness of algorithms of this kind.

METIS [10] is the most popular multilevel partitioning scheme; however, it has some serious deficiencies when applied to grid environments. We enumerate 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 mapping. This approach is usually inefficient in a distributed environment. Instead, MinEX creates partitions that consider data remapping and strives to overlap application processing and communication to minimize the total application runtime.
- For heterogeneous grids, the processing and communication costs are non-uniform. Instead of assuming uniform weights (as METIS does), 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 application latency tolerance capability 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 [2] for the classical N-body problem. Test cases of 16K 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. Results show that MinEX reduces the runtime requirements to solve the N-body application by up to a factor of 8 compared to those obtained when using METIS in heterogeneous configurations. Results also show that MinEX is competitive to METIS in terms of partitioning speed.

2 Preliminaries

2.1 Partition Graph

A graph representation of the application is supplied as input to partitioners so that the vertices can be assigned among processors in a load balanced fashion. Each vertex v of this *partition graph* has weights $\mathtt{PWgt}_v$ and $\mathtt{RWgt}_v$, while each defined edge (v, w) between vertices v and w has weight $\mathtt{CWgt}_{(v,w)}$. These weights refer respectively to the computation, data remapping, and communication costs associated with processing a graph vertex. Section 4.3 describes how they are computed for our N-body application.

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 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. Edge (c, d) corresponds to the communication links between processors in clusters c and d. A self-loop (c, c) indicates communication among the processors of a single cluster. All processors within a cluster are fully connected and homogeneous, with constant intra-cluster communication bandwidth.

The vertices of the configuration graph have a single weight, $\mathtt{Proc}_c \geq 1$, that represents the processing slowdown for the processors of cluster c, relative to the fastest processor in the entire grid. Likewise, edges have a weight $\mathtt{Conn}_{(c,d)} \geq 1$ to model the interconnect slowdown when processors of cluster c communicate with processors of cluster d. If $\mathtt{Conn}_{(c,d)} = 1$, it represents the fastest connection in the network. If $c = d$, $\mathtt{Conn}_{(c,c)}$ 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 $\mathtt{CMap}_p$ determines the cluster associated with processor p in the grid.

2.3 Time Unit Metrics

MinEX is unique in that its objective is to minimize application runtime. To accomplish this goal, the application partition graph is first mapped onto the grid configuration graph. The following three metrics are then used to measure computation, communication, and data remapping costs:

- **Processing Cost** is the computational cost to process vertex v assigned to processor p in cluster c, and expressed as $\mathtt{Wgt}_p^v = \mathtt{PWgt}_v \times \mathtt{Proc}_c$.
- **Communication Cost** is the cost to interact with all vertices adjacent to v but whose data sets are not local to p (assuming 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 $\mathtt{Comm}_p^v = \sum_{w \notin p} \mathtt{CWgt}_{(v,w)} \times \mathtt{Conn}_{(c,d)}$. If the data of all vertices adjacent to v are also assigned to p, $\mathtt{Comm}_p^v = 0$.
- **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 $\mathtt{Remap}_p^v = \mathtt{RWgt}_v \times \mathtt{Conn}_{(c,d)}$. Here we assume that p is in cluster c while q is in cluster d.

2.4 System Load Metrics

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

- **Processor Workload** ($\mathtt{QWgt}_p$) is the total cost to process all the vertices assigned to processor p and is given by $\mathtt{QWgt}_p = \sum_{v \in p} (\mathtt{Wgt}_p^v + \mathtt{Comm}_p^v + \mathtt{Remap}_p^v)$.
- **Total System Load** ($\mathtt{QWgtTOT}$) is the sum of $\mathtt{QWgt}_p$, over all P processors.
- **Average Load** ($\mathtt{WSysLL}$) is $\mathtt{QWgtTOT}$ / P.
- **Heaviest Processor Load** ($\mathtt{RT}$) is the maximum value of $\mathtt{QWgt}_p$ over all processors, and indicates the total time required to process the application.
- **Load Imbalance Factor** ($\mathtt{LI}$) represents partitioning quality, and is given by the ratio $\mathtt{RT}$ / $\mathtt{WSysLL}$.

2.5 Partitioning Metrics

These metrics are used by MinEX to make decisions:

- $\mathtt{Gain}$ represents the change in $\mathtt{QWgtTOT}$ that would result from a proposed vertex reassignment. A negative value indicates reduced processing after such a reassignment.

MinEX favors vertex migrations with negative or small Gain that reduce or minimize the overall system load.

– Var is the variance in processor workloads and is computed as $\texttt{Var} = \sum_p (\texttt{QWgt}_p - \texttt{WSysLL})^2$. The objective is to initiate vertex moves that lower this value. Since individual terms of this formula with large values correspond to severely unbalanced processors, minimizing Var tends to improve system load balance.

3 MinEX Partitioner

The MinEX partitioner was originally introduced in [6]. In this paper, we present an overview of MinEX and introduce refinements that we have made since that earlier report.

MinEX can execute either in a diffusive manner [4] or it can create partitions from scratch [14]. The entire process occurs in three steps: contraction, partitioning, and refinement, similar to other multilevel partitioners. However, MinEX redefines the partitioning goal to minimizing RT rather than balancing partitions and reducing the total edge cut. In addition, MinEX allows applications to provide a function to achieve latency tolerance, if available.

3.1 Partitioning Criteria

Partitioning involves reassigning vertices from overloaded processors ($\texttt{QWgt}_p > \texttt{WSysLL}$) to underloaded processors ($\texttt{QWgt}_p < \texttt{WSysLL}$). To facilitate this process, MinEX maintains a list of processors sorted by $\texttt{QWgt}_p$ that is updated after each vertex reassignment. Since a very small subset of processors change positions in this list after a reassignment, the overhead for maintaining it is acceptable. Any reassignment that projects a negative $\Delta\texttt{Var}$ value is executed. This is the *basic partitioning criteria*.

3.2 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 estimate the effect of a vertex reassignment. Only those reassignments that pass through the filter are considered in accordance with the basic partitioning criteria (see Sec. 3.1). The filter utilizes edge outgoing and incoming communication totals of each vertex to estimate QWgt values for the source and destination processors ($\texttt{newQWgt}_{from}$ and $\texttt{newQWgt}_{to}$). The pseudo code shown in Fig. 1 then decides whether a potential vertex reassignment should be accepted.

The reassignment filter is designed to minimize increases in RT. It also rejects vertex reassignments that project a positive $\Delta\texttt{Var}$ value. The ThroT parameter acts as a gate to prevent reassignments that cause excessive increases in Gain. A low ThroT could prevent MinEX from finding a balanced partitioning allocation, while a high value could converge to a point where runtime is unacceptable.

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** $\texttt{newGain}^2 / -\Delta\texttt{Var} > \texttt{ThroT}$) **Reject Assignment**
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**
Accept Assignment

Figure 1. Pseudo code to determine promising vertex reassignments.

Table 1 demonstrates the effect on runtimes (shown in thousands in units) using different ThroT values in our simulations with 16K bodies. The columns labeled $I = 10$ and $I = 100$ refer to grid configurations with interconnect slowdowns of 10 and 100, respectively. The configuration type is UP (defined later in Sec. 5). The processors are always grouped into 8 clusters. The table shows results for grids with 32, 64, and 128 processors. Based on these tests, our simulations in Sec. 5 use a ThroT value of 32.

Table 1. Runtimes (RT) for 16K bodies using various ThroT values

	$P=32$		$P=64$		$P=128$	
ThroT	I=10	I=100	I=10	I=100	I=10	I=100
0	1157	1353	907	830	606	583
2	777	1345	457	829	221	465
8	758	1347	426	813	221	465
32	760	1347	398	794	213	465
128	755	1347	402	789	204	465
512	755	1347	402	790	206	465

Table 2 demonstrates the effectiveness of the reassignment filter for 8, 128, and 1024 processors, grouped into 8 clusters. The interconnect slowdown is set to 10. The partition graph represents N-body problems consisting of 16K and 256K bodies. We show the total number of vertex assignments considered (Total), the number of assignments that passed through the filter (Pass), and the number of potential reassignments that subsequently failed the basic partitioning criteria described in Sec. 3.1 (Fail). Results

Table 2. Filter effectiveness for 16K and 256K bodies

	16K bodies			256K bodies		
P	Total	Pass	Fail	Total	Pass	Fail
8	6011	110	0	25183	222	0
128	19192	2562	0	51876	4608	1
1024	18555	2790	7	35605	12639	2

clearly demonstrate that the reassignment filter eliminates almost all of the edge processing overhead associated with reassignments that are rejected.

3.3 Application Interface

The partition graph is supplied by the application program while the configuration graph is created using Grid Information Services (GIS). The MinEX function signature is similar to that of METIS and is shown below:

void MinEX_PartGraph (nvert, *adjcy, *cwgt, *ewgt, *vadj, *vwgt, *rwgt, *vown, *part, *ipg, *user)

- nvert — number of nodes in the partition graph,
- adjcy — adjacency list of vertices,
- cwgt — outgoing edge weights ($\texttt{CWgt}_{(v,w)}$),
- ewgt — incoming edge weights ($\texttt{CWgt}_{(w,v)}$),
- vadj — offsets into adjcy, cwgt, ewgt for each vertex,
- vwgt — processing weights ($\texttt{PWgt}_v$) of each vertex,
- rwgt — redistribution weights ($\texttt{RWgt}_v$) of each vertex,
- vown — original processor assignment for each vertex,
- part — partitioning generated by MinEX,
- ipg — grid configuration graph, and
- user — user-supplied options containing:
  (a) ThroT value,
  (b) number of vertices in the contracted graph,
  (c) application latency tolerance function, and
  (d) diffusive or from-scratch partitioning.

3.4 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, *ipg, *tot)

- user — user-supplied options to MinEX_PartGraph,
- ipg — grid configuration, and
- tot — projected totals computed by MinEX containing:
  (a) p, the processor to which this call applies,
  (b) $\texttt{QLen}_p$, the number of vertices assigned to p,
  (c) $\texttt{Pproc}_p = \sum_{v \in p} \texttt{Wgt}_p^v$,
  (d) $\texttt{Crcv}_p = \sum_{v \in p} \texttt{Comm}_p^v$, and
  (e) $\texttt{Rrcv}_p = \sum_{v \in p} \texttt{Remap}_p^v$.

The MinEX_LatTol 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 Sec. 2.4 because some of the processing is overlapped with communication.

4 N-body Application

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

Of the many N-body solution techniques that have been proposed, the Barnes & Hut algorithm [2] 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.

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

At each iteration in the N-body application, a new or modified tree of cells is recursively constructed to allocate the bodies to cells. 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. The entire cycle is repeated for the desired number of time steps.

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 in-

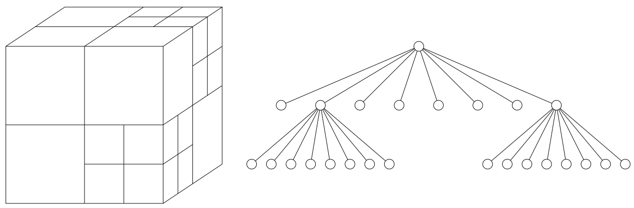

Figure 2. A three-level octree and the corresponding spatial representation.

serting bodies into an initial cell until it contains `CellMax` bodies. The parameter `CellMax` is chosen to minimize the number of force calculations. Before the next body can be inserted, this cell is split into eight octants and the previously-inserted bodies are distributed based on their centers of mass. Insertion of bodies continues until one of the cells again have `CellMax` bodies, and the process is repeated. Naturally, all the bodies reside in the leaves of the octree. Figure 2 illustrates this concept: both the spatial and tree representations are shown. The cell's center of mass is used for subsequent searches of the octree. Traversal direction is determined by the octant where a body resides relative to this center of mass.

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 Sec. 4.4). For direct comparisons between the two partitioners, simulations are conducted with the modified graph G_M.

Each vertex v of G corresponds to a leaf cell C_v (containing $|C_v|$ 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 Sec. 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 expression represents the double integration of acceleration to obtain body positions at the next time step once the effects 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 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 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 also directed because edge (v, w) has weight equal to the number of bodies in C_v whereas edge (w, v) has weight equal to the number of bodies in C_w. These quantities are not necessarily equal.

To accommodate these two limitations of 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 ensures that the edges in G_M have non-zero weights, and that $\texttt{CWgt}_{(v,w)} = \texttt{CWgt}_{(w,v)}$.

4.5 Solution Algorithm

The force between two bodies (cells if they are far apart) are calculated using Newtonian gravitational formulae:

- The position vector $p^b = \langle x^b, y^b, z^b \rangle$ represents the location of body b.
- The Euclidean distance between bodies b and c is given by $r(b, c) = \sqrt{(x^b - x^c)^2 + (y^b - y^c)^2 + (z^b - z^c)^2}$.
- The gravitational force between b and c 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. A small constant is added to $r(b, c)$ to prevent division by zero. Body forces in the y and z directions are similarly defined.
- The acceleration vector $a^b = \langle a^b_x, a^b_y, a^b_z \rangle$ for b is computed by dividing its total force by m^b. It is then integrated to obtain the velocity vector $v^b = \langle v^b_x, v^b_y, v^b_z \rangle$. A second integration on v^b provides the position of b at the next time step. All integrations use the leap-frog method.

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 is avoided. The pseudo code in Fig. 3 indicates solver execution by each processor. The 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.

```
Broadcast changes to the Barnes & Hut tree
Relocate bodies based on the computed partition
For each data set that is relocated to this processor
        Unpack and store the data
        Calculate forces between local close cells
For each body assigned to this processor
        Transmit body and cell data of remote close bodies
For each body assigned to this processor
        Calculate forces with local far cells
While more position and mass data remain to be received
        Receive position and mass data
        Calculate forces using data received
For each body assigned to this processor
        Integrate to determine new body position
```

Figure 3. 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 Simulation Study

We use discrete time simulation to mimic a grid environment in which the N-body solver is executed. Communication is via message passing primitives similar to those in MPI. A configuration graph (defined in Sec. 2.2) is used to model the grid. Test cases containing 16K and 256K bodies that represent two neighboring Plummer galaxies about to merge [17] are considered. The partition graphs (defined in Sec. 2.1) for these test cases respectively contain 4563 and 14148 vertices, and 99802 and 236338 edges.

Graphs labeled G in the following tables refer to the directed graph (see Sec. 4.3) and are used only by MinEX. Graphs labeled G_M are undirected (see Sec. 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, with its option to minimize edge cuts, is used.

The configuration graph is varied to evaluate performance over a spectrum of heterogeneous grid environments. The total number of processors (P) varies between 8 and 1024, the number of clusters (C) is either 4 or 8, while interconnect slowdowns (I) are 10 or 100. Communication within clusters is assumed to be constant. Three configuration types (HO, UP, and DN) are used in our simulations. HO assumes that all processors are homogeneous and grouped evenly among the clusters with intra-connect and processing slowdown factors of unity. UP assumes processors in cluster i have intra-connect and processing slowdown factors of $2i - 1$, while DN assumes processors in cluster i have intra-connect slowdown factors of $2C-1-2i$ and processing slowdown factors of $2i - 1$.

5.1 Multiple Time Step Test

This set of tests determines whether running multiple time steps are likely to significantly impact the overall performance. Table 3 presents `RT` (in thousands of units) and `LI` when executing 1 and 50 time steps. The partition graph represents 16K bodies and the configuration type is UP, with $P = 64$, $C = 8$, and $I = 10$. Results show that running multiple time steps have little impact. Our subsequent simulations therefore execute only a single time step.

Table 3. Performance for 1 and 50 time steps

	1 time step		50 time steps	
Type	RT	LI	RT	LI
MinEX$-G$	398	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

To determine partitioner scalability, we process graphs of 16K and 256K bodies using the UP configuration containing between 16 and 1024 processors with $C = 8$ and $I = 10$. Table 4 reports `RT` (in thousands of units) and shows that scalability is good to 256 processors.

Table 4. Application runtimes (`RT`)

		Number of processors P			
Bodies	Graph Type	16	64	256	1024
16K	MinEX$-G$	1445	398	124	268
	MinEX$-G_M$	1466	413	124	268
	METIS$-G_M$	5330	1630	1395	1209
256K	MinEX$-G$	39335	10187	2917	1310
	MinEX$-G_M$	39448	10183	2927	1310
	METIS$-G_M$	151983	38379	9901	5220

5.3 Partitioner Speed Comparisons

We then compare MinEX partitioning speed to that of METIS for P between 16 and 1024, with partition graphs representing 16K and 256K bodies. The UP configuration is used with $C = 8$ and $I = 10$. Results in Table 5 show that MinEX executes faster in the majority of cases; however, METIS has a clear advantage when processing the 256K case with $P = 1024$. In general though, we can conclude that MinEX is competitive with METIS in execution speed.

Table 5. Partitioner runtimes (in secs)

		Number of processors P			
Bodies	Graph Type	16	64	256	1024
16K	MinEX$-G$	0.20	0.33	1.09	2.36
	MinEX$-G_M$	0.20	0.32	1.13	2.39
	METIS$-G_M$	0.23	1.02	1.46	2.88
256K	MinEX$-G$	0.53	0.71	2.27	9.08
	MinEX$-G_M$	0.55	0.69	2.30	9.17
	METIS$-G_M$	0.49	0.76	2.57	4.18

5.4 Partitioner Quality Comparisons

Finally, we present results that extensively compare the quality of partitions generated by MinEX and METIS in terms of N-body application runtimes (RT) and load imbalance factors (LI). Table 6 presents simulation results using partition graphs representing 16K and 256K bodies. Notice that MinEX has a significant advantage over METIS for the heterogeneous UP and DN configurations. In fact, if $P = 128$, $C = 8$, and $I = 10$, MinEX reduces RT by a factor of almost 8 for 16K bodies. The improvement in LI is also dramatic. For 256K bodies, some of the MinEX results with the DN configuration are worse than the corresponding results with UP (see the cases when $P = 128$ and $C = 4$). The discrepancy is because processors incur excessive idle time when processing the application with the DN configuration so that the MinEX estimates are not realized.

Results for the homogeneous HO configurations are less conclusive. For 16K bodies, METIS is competitive with MinEX; however, in some cases, MinEX is slightly better (e.g., $P = 128$, $C = 4$, $I = 10$). With 256K bodies, METIS is superior when $P = 32$ or 64, and $I = 100$. This is not surprising given that METIS's strategy to minimize the edge cut is more effective in homogeneous grids. To improve MinEX performance in these situations, the partitioning criteria for vertex reassignments needs to be refined. This is an open research issue that needs to be addressed if one is to build successful general-purpose grid partitioners.

Note that MinEX running with graph G is generally superior to running with graph G_M. This is because G models the solver more closely than G_M does. For slow interconnects ($I = 100$), the advantage of MinEX over METIS is reduced because the communication overhead begins to dominate. If MinEX were refined to put a greater emphasis on minimizing the communication cut, it could probably retain more of its advantage over METIS in these cases.

6 Conclusions

In this paper, we have used the classical N-body application to evaluate our MinEX latency-tolerant partitioner designed specifically for heterogeneous distributed computing environments. MinEX has significant advantages over traditional graph partitioners. For example, its 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 can 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.

Extensive simulations using an N-body solver showed that while MinEX produces partitions of comparable quality to those by METIS on homogeneous grids, it improves application runtimes by a factor of 8 on some heterogeneous configurations. The experiments demonstrated the feasibility and benefits of our approach to map applications onto grid environments, and to incorporate latency tolerance directly into the partitioning process. The simulations also revealed 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 of the application. Additional schemes for reassigning vertices in a grid environment must be explored to achieve consistent results in all configurations. We are actively investigating these enhancements.

References

[1] Globus Project. http://www.globus.org.

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

[3] H. Casanova and J. Dongarra. Netsolve: A network-enabled server for solving computational science problems. *Intl. Journal of Supercomputer Applications*, 11:212–223, 1997.

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

[5] J. Czyzyk, M. Mesnier, and J. Moré. The network-enabled optimization system (NEOS) server. Technical Report MCS-P615-1096, Argonne National Laboratory, 1997.

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

[7] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999.

Table 6. Performance for 16K and 256K bodies

				16K bodies						256K bodies					
				UP		HO		DN		UP		HO		DN	
P	C	I	Type	RT	LI	RT	LI	RT	LI	RT	LI	RT	LI	RT	LI
32	4	10	MinEX$-G$	459	1.00	197	1.02	461	1.01	12114	1.00	5137	1.01	12122	1.00
			MinEX$-G_M$	479	1.04	206	1.06	480	1.05	12186	1.01	5178	1.02	12223	1.01
			METIS$-G_M$	1362	1.88	196	1.07	1362	1.88	35474	1.79	5122	1.04	35474	1.79
32	4	100	MinEX$-G$	1023	1.07	921	1.02	1046	1.07	12344	1.04	7239	1.19	12355	1.04
			MinEX$-G_M$	1157	1.22	1167	1.25	1196	1.21	14089	1.09	6929	1.17	14628	1.10
			METIS$-G_M$	1363	1.74	1167	2.49	1363	1.73	35475	1.79	5132	1.04	37475	1.79
32	8	10	MinEX$-G$	760	1.01	197	1.02	763	1.01	20038	1.00	5177	1.02	20095	1.00
			MinEX$-G_M$	780	1.04	211	1.09	792	1.04	20193	1.01	5188	1.02	20220	1.01
			METIS$-G_M$	2919	2.01	196	1.07	2919	2.01	76017	1.92	5122	1.04	76017	1.92
32	8	100	MinEX$-G$	1347	1.03	1198	1.02	1371	1.03	21173	1.06	7399	1.19	21634	1.05
			MinEX$-G_M$	1562	1.17	1417	1.24	1574	1.17	23471	1.06	7541	1.23	24136	1.06
			METIS$-G_M$	2920	1.91	1182	1.67	2920	1.90	76018	1.93	5199	1.05	76018	1.92
64	4	10	MinEX$-G$	234	1.01	106	1.08	235	1.04	6109	1.00	2590	1.01	6124	1.00
			MinEX$-G_M$	245	1.05	109	1.11	245	1.04	6158	1.01	2625	1.03	6172	1.01
			METIS$-G_M$	761	2.01	108	1.16	761	2.01	18102	1.82	2650	1.07	18102	1.82
64	4	100	MinEX$-G$	620	1.10	450	1.20	634	1.11	6627	1.09	4231	1.22	6616	1.07
			MinEX$-G_M$	737	1.28	612	1.45	746	1.27	8237	1.11	4131	1.28	8276	1.11
			METIS$-G_M$	763	1.72	621	2.12	763	1.73	18102	1.82	3334	1.33	18102	1.82
64	8	10	MinEX$-G$	398	1.03	109	1.11	372	1.01	10187	1.01	2590	1.02	10160	1.00
			MinEX$-G_M$	413	1.05	115	1.15	421	1.06	10183	1.01	2625	1.03	10222	1.01
			METIS$-G_M$	1630	2.16	108	1.16	1630	2.16	38379	1.95	2656	1.07	38379	1.95
64	8	100	MinEX$-G$	794	1.05	549	1.03	798	1.04	11459	1.08	4241	1.27	11982	1.06
			MinEX$-G_M$	936	1.22	685	1.39	946	1.20	13400	1.08	4894	1.20	13797	1.08
			METIS$-G_M$	1632	2.01	841	2.01	1632	2.00	38791	1.95	3842	1.51	38791	1.95
128	4	10	MinEX$-G$	121	1.03	94	1.80	194	1.16	3094	1.01	1384	1.07	4362	1.07
			MinEX$-G_M$	122	1.03	94	1.44	194	1.15	3119	1.02	1384	1.07	4357	1.08
			METIS$-G_M$	755	3.98	131	2.73	917	4.60	9209	1.83	1443	1.15	10105	1.94
128	4	100	MinEX$-G$	353	1.32	426	1.29	493	1.22	3654	1.15	2324	1.24	6545	1.33
			MinEX$-G_M$	425	1.40	408	1.40	482	1.20	4284	1.09	2464	1.42	5896	1.12
			METIS$-G_M$	756	3.42	425	3.19	917	3.76	9210	1.83	2337	1.79	10185	1.91
128	8	10	MinEX$-G$	213	1.11	94	1.80	196	1.08	5113	1.01	1353	1.07	5174	1.02
			MinEX$-G_M$	206	1.05	94	1.55	217	1.05	5174	1.02	1372	1.07	5174	1.02
			METIS$-G_M$	1619	4.27	131	2.73	1619	4.27	19734	1.96	1443	1.15	19374	1.96
128	8	100	MinEX$-G$	465	1.17	428	1.23	476	1.18	6160	1.07	2111	1.11	6620	1.09
			MinEX$-G_M$	519	1.23	633	1.82	678	1.65	7109	1.07	2326	1.22	7421	1.12
			METIS$-G_M$	1619	3.99	604	3.19	1619	3.97	19735	1.96	2337	1.74	19735	1.96

[8] A. Grimshaw and W. Wulf. The Legion vision of a worldwide computer. *Comm. of the ACM*, 40:39–45, 1997.

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

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

[11] S. Kumar, S. Das, and R. Biswas. Graph partitioning for parallel applications in heterogeneous grid environments. In *Proc. 16th Intl. Parallel and Distributed Processing Symp.*, 2002.

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

[13] M. Litzdow, M. Livny, and M. Mutka. Condor — a hunter of idle workstations. In *Proc. 8th Intl. Conf. on Distributed Computing Systems*, pages 104–111, 1988.

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

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

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

[17] J. 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*, 27:118–141, 1995.

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