Optimization and evaluation of parallel I/O in BIPS3D parallel irregular application

Rosa Filgueira, David E. Singh,
Florin Isaila and Jesús Carretero

Antonio García Loureiro

Departement of Computer Science
University Carlos III of Madrid - Spain
{rosaf,desingh,florin,jcarrete}@arcos.inf.uc3m.es

Department of Electronics and Computer Science
Universidad de Santiago de Compostela - Spain
antonio@dec.usc.es

Abstract

This paper presents the optimization and evaluation of parallel I/O for the BIPS3D parallel irregular application, a 3-dimensional simulation of BJT and HBT bipolar devices. The parallel version of BIPS3D employs Metis, a library for partitioning graphs, finite element meshes, or sparse matrices. First, we show how the partitioning information provided by Metis can be used in order to improve the performance of parallel I/O. Second, we propose a novel technique, called Interval Data Grouping *(IDG), which exploits the data replication of mesh nodes for optimizing the scheduling of the parallel file operations. Finally, we evaluate the parallel I/O version of BIPS3D for various existing parallel I/O techniques and present an in-depth analysis of the IDG performance.*

1 Introduction

The performance of applications accessing large data sets is often limited by the speed of I/O subsystems. Additionally, the gap between the processor performance on the one hand and the memory and I/O performance keeps increasing at a high rate. Consequently, accessing large amounts of data may cause bottlenecks or underutilization of the computing resources.

For this reason, a lot of research has targeted the improvement of the parallel I/O performance of the data-intensive applications. In this paper, we present the parallelization and optimization of the file system accesses of BIPS3D [7], an irregular parallel scientific application.

We introduce a novel parallel I/O optimization strategy, IDG, by means of which, we demonstrate that, for certain access patterns, the data access locality may play a more important role than load balance.

The paper is structured as follows. Section 2 briefly overviews the BIPS3D application. Section 3 presents parallel I/O systems, optimizations and interfaces. Section 4 contains implementation details of the parallel I/O version of IDG. The novel locality-based parallel I/O technique IDG is described in section 5. The experimental results are presented in section 6. Finally, we summarize in section 7.

2 BIPS3D

In this section we give an overview of the BIPS3D application. Throughout the paper we will give more details about the particular application phases.

BIPS3D is a 3-dimensional simulation of BJT and HBT bipolar devices. The goal of the 3D simulation is to relate electrical characteristics of the device with its physical and geometrical parameters [1],[2]. The basic equations to be solved are Poisson's equation and electron and hole continuity, in a stationary state.

Finite element methods are applied in order to discretize the Poisson equation, hole and electron continuity equations by using tetrahedral elements. The result is an unstructured mesh in which we place more nodes in the areas of union between different areas of the transistor.

Using the METIS library [4], this mesh is divided into sub-domains, in such a manner that one sub-domain corresponds to one processor. The next step is decoupling the Poisson equation, hole and electron continuity equations, and linearize them using the Newton method. Then we construct, for each sub-domain, in a parallel manner, the part corresponding to the associated linear system. Each of these systems is solved using domain decomposition methods. Finally, the results are written to a file.

1-4244-0910-1/07/$20.00 ©2007 IEEE

3 Parallel I/O background

The compute nodes of a parallel application may write the data to a file system in parallel. However, if the file system manages a single disk, the parallel accesses are serialized. This is the case with the NFS distributed file system [9] exporting a local file system. A true parallel disk access can be obtained from parallel file systems such as PVFS [6] or GPFS [10], which employ in parallel several I/O severs or mount parallel disks.

The processes of a parallel application frequently access a common data set by issuing a large number of small non-contiguous I/O requests. Collective I/O addresses this problem by merging small individual requests into larger global requests in order to optimize the network and disk performance. Depending on the place where the request merging occurs, one can identify two collective I/O methods. If the requests are merged at the I/O nodes the method is called *disk-directed I/O* [5, 11]. If the merging occurs at intermediary nodes or at compute nodes the method is called *two-phase I/O* [2, 1]. Two-phase I/O is in ROMIO [12], an implementation of MPI-IO interface.

Many existing file systems' interfaces are based on the Portable Operating System Interface(POSIX) [3]. The main limitation of POSIX is that it is not addressing the requirements of high-performance parallel applications. In 2005, a working group has started to work at a POSIX I/O API extension, with the goal to make the POSIX I/O API more friendly to HPC, clustering, parallelism, and high concurrency applications. However, the proposal is in work and it has not been widely adopted yet. List I/O [13] is an interface for describing non contiguous accesses both in file and in memory. Non-contiguous accesses are specified through a list of offsets of contiguous memory or file regions and a list of lengths. MPI-IO [8] is a standard interface for MPI-based parallel I/O. MPI data types are used by MPI-IO for declaring views and for performing non-contiguous accesses. A *view* is a contiguous window to potentially non-contiguous regions of a file. After declaring a view on a file, a process may see and access non-contiguous regions of the file in a contiguous manner.

4 Implementation of parallel I/O in BIPS3D

As many parallel scientific applications, BIPS3D consists of separate compute and I/O phases. In the first phase, based on the input distribution, the Metis library is used for providing a data distribution that would load balance the execution. Using this information, the data mesh is distributed over the processors. Subsequently, the computation is performed on the data. Finally, the data is written to the file system.

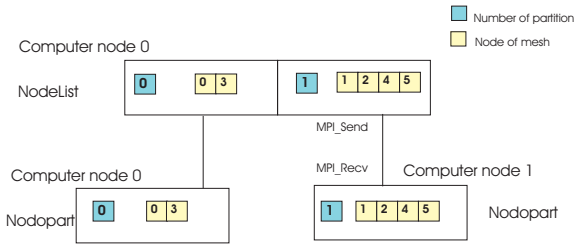


Figure 1. Data distribution example

An important observation is that the final data format is similar to the initial one. In other words, the partitioning provided by Metis can be used as well for the parallel I/O phase. In the initial BIPS3D version, in the final I/O phase, the results were gathered at a root node, which stored the data sequentially to the file system.

In the remainder of this section we describe the details of the parallel implementation.

The result of Metis mesh partitioning are the parameters of the data distribution over a given number of processors. The distribution is represented as an array of structures `distrib`.

Each element of the `distrib` array represents a compute node and contains, among other information, a vector of structures called `nodelist`, representing the mesh nodes assigned to the respective compute node by Metis.

The `node` structure contains the following information: an identifier of the mesh position (`id`), the number of assigned vertices `nr_of_vertices`, the vertex list `vertices`, the corresponding compute node `processor`, and a variable-sized vector `load` representing the physical parameters corresponding to the materials at each mesh node.

Initially, the program distributes to each compute node the corresponding `distrib` vector element, as shown in the example from Figure 1.

Subsequently, the computation is performed over the `load` vector. Finally, the data is stored to disk either sequentially or in parallel. The following I/O configurations are possible:

- sequential I/O over NFS
- sequential I/O over PVFS
- parallel I/O over PVFS
- parallel I/O over PVFS with two-phase I/O
- parallel I/O over PVFS with list I/O

The sequential I/O is the original one and was explained in the first paragraph of this section. In our configuration NFS or PVFS file systems can be used.

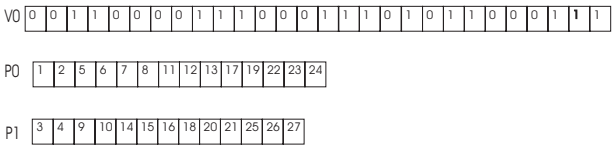


Figure 2. Mesh distribution example.

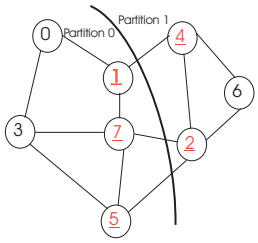


Figure 3. Example of a Mesh.

In the parallel I/O, each compute node uses the distribution information initially obtained from Metis and constructs a view on the file. The view is based on an MPI data type. One example of a mesh distributed over two compute nodes is illustrated in Figure 2. The vector `v0` shows to which of the two processors the data is assigned. The first and second entries correspond to compute node 0, the third to compute node 1, and so on. The vectors `p0` and `p1` contain the file positions where each of the elements of compute node 0 and 1 are to be stored. In order to achieve the MPI data type `MPI_Type_Indexed` is used. This data type represents non-contiguous lying data of equal sizes and with different displacements between consecutive elements.

Once the view on the common file is declared, the compute nodes may write the data to its corresponding file part either independently or collectively, as chosen by the user.

5 IDG description

In this section we present a novel technique for improving the I/O performance. This technique, called *Interval Data Grouping* (IDG), exploits the data replication of mesh nodes for scheduling disk accesses in order to improve the performance of the parallel output operation. The goal of IDG is grouping data for I/O in order to increase the locality of data, as opposed to the Metis-based approach presented in the previous section, in which the goal was to improve the load balance.

As many other finite element applications, BIPS3D uses tetrahedral meshes for simulating the semiconductor devices. These meshes are distributed among the compute nodes using a pre-defined policy which tries to balance the computational load and decrease the communication. On one hand, the work load is associated to each mesh node [1] thus load balance can be expressed as distributing the mesh in close-equal portions. On the other hand, communication is performed between neighboring mesh nodes (connected through an edge) possibly assigned to different computing nodes. Minimizing the communications is equivalent to minimizing the number of edges cut by the partitions. BIPS3D uses Metis mesh partitioner for reaching both requisites. Figure 3 shows an example of a mesh with 8 nodes and 12 edges.

Once the mesh is partitioned and distributed its nodes are classified into two classes: local and shared. A local node is exclusively assigned to one specific partition and is not replicated. All its neighboring nodes are assigned to the same partition and its associated edges are not cut by the partitions. In contrast, a shared node has at least one edge cut, thus at least one neighboring node is assigned to other partition. Due to boundary conditions, shared nodes are replicated among the neighboring partitions. In the example of Figure 3, nodes $\{0, 3\}$ are local to partition 0; node $\{6\}$ is local to partition 1 and nodes $\{1, 2, 4, 5, 7\}$ are shared among these two partitions.

Note that the complete information of a shared node is replicated, and is computed with redundancy by the BIPS3D application. Note also that after the compute phase, both node replicas contain valid information, and, consequently, can be indistinctly used for performing the output disk operation. IDG algorithm exploits this property for choosing the most appropriate shared nodes for performing the disk access. The criterion used by IDG is increasing data locality for reducing the overall disk write time.

Figure 4 shows the IDG pseodocode. It consists of two stages: node classification and disk access scheduler. The node classification phase analyzes the mesh structure (using the mesh topology file) and the Metis distribution for classifying mesh nodes into local or shared. This stage works as follows: using the mesh topology, for each node, all its neighboring nodes are retrieved (`L1` label). This stage returns a data structure (called $partition\_list$) with as many rows as number of nodes. Each row contains the partition to which the specific node is assigned. In case of being a local node, the row will have one entry corresponding to the Metis partition (`L2`). In contrast, a shared node will have as many entries as partitions, in which is replicated. These partitions are obtained by analyzing all neighbors (`L3`). Then, their assigned partition is compared with the one associated to the considered node (`L4`). In case of all being the same, we conclude that the node is local. Otherwise, the node is shared and a new partition is added to the list (`L5`). For the example from Figure 3, the resulting data structure is shown

[1] Each mesh node contains values of different physical measures of the semiconductor discrete element that represents. This information is read, analyzed and processed by BIPS3D, representing the application computational load. Consequently, we can say that the work load associated with a mesh (or portion) is linearly dependent on the number of nodes that contains.

```
 Begin algorithm IDG:

input: mesh  Mesh topology
       metis_distr  Metis node distribution

input: idg_distr  IDG node distribution

NODE CLASSIFICATION
  for each node ∈ Meshtopology
L1    neighbor_list = take_neighbors(mesh, node)
L2    add_partition(partition_list, node, metis_distr(node))
L3   for each neighbor ∈ neighbor_list
L4     if(metis_distr(node) ≠ metis_distr(neighbor))
L5       add_partition(partition_list, node, metis_distr(neighbor))
       end if
     end for
  end for

DISK ACCESS SCHEDULER
  for each node ∈ Meshtopology
L7   if  (number_partitions(partition_list, node) = 1)
L8   idg_distr(node) = metis_distr(node)
     else
L9     for each partition ∈ partition_list(node)
L10      if(partition = metis_distr(node − 1))
L10        idg_distr(node) = metis_distr(node − 1)
L11      else if(partition = metis_distr(node + 1))
L11        idg_distr(node) = metis_distr(node + 1)
         end if
       end for
  end for

 End algorithm
```

Figure 4. IDG algorithm pseudocode.

Node	Neighbors	Metis node distrib.	Partition list
0	0 1 3	0	0
1	0 4 7	0	0 1
2	4 5 6 7	1	1 0
3	0 5 7	0	0
4	1 2 6	1	1 0
5	2 3 7	0	0 1
6	2 4	1	1
7	1 2 3 5	0	0 1

Table 1. Data structure of example mesh.

in Table 1.

The second stage (Figure 4) schedules disk write operations. Each mesh partition is assigned to a different computation node. In the case of local nodes (`L7`) there is a fixed scheduling policy, i.e. they are written by the compute node, which they belong to (`L8`). In the case of shared nodes, this stage chooses (among all the compute nodes where they are replicated), the most appropriate one. The criterion used for assigning each shared node is to look to its previous and subsequent nodes[2]. This stage works as follows. For each shared node(`L9`), its previous node is considered (`L10`). If its assigned partition belongs to the shared node partition list, then it is associated to the same partition (increasing the locality). Otherwise, we apply the same procedure with the subsequent node (`L11`). If the considered node is still unassigned after both checks, we assign it to the first entry of its partition list (not shown in the figure). For the example from Figure 3, shared node 1 is the first one considered. Given that previous node (node 0) is local, it is assigned to partition 0. This procedure is applied to nodes $\{1, 2, 4, 5, 7\}$. The resulting node assignation can be seen in Table 2.

[2]Note that we are trying to improve the disk access locality, thus here we are referring *previous* and *subsequent* in terms of disk storage order. For a given node i, its previous and subsequent nodes are $i - 1$ and $i + 1$ respectively.

Node	Partition list	IDG
0	0	0
1	0 1	0
2	1 0	0
3	0	0
4	1 0	0
5	0 1	0
6	1	1
7	0 1	1

Table 2. IDG distribution of example mesh.

6 Experimental results

In this section we present an extensive evaluation of the parallel I/O version of the BIPS3D application. In the first subsection five existent I/O techniques are evaluated. The second subsection shows an in-depth analysis of the IDG based on the list I/O technique, which showed the best results in the experiments from the first section.

We performed our experiments on a cluster of 16 dual processors Pentium III 800MHz, having 256 KBytes L2 cache and 1024 MB RAM, interconnected by Fast Ethernet and Myrinet LANai 9 cards at 133 MHz, capable of sustaining a throughput of 2 GB/s in each direction. For Myrinet we have used MPICH 1.7.15 and for Fast Ethernet MPICH 1.2.6. The PVFS version 1.6.3 with a striping factor of 64 KBytes.

6.1 Evaluation of existing parallel I/O techniques

We have executed BIPS3D for four different meshes, with different number of nodes, having a load between 1 and 500 data entries per node, using 4, 7, 8, 10, 14 partitions and both Myrinet and Fast Ethernet networks. Our goal was to investigate the relationship among these parameters and write performance for each mesh in order to predict the most appropriate I/O configuration for each case.

We have used the meshes 1, 2, 3 for both Myrinet and Fast Ethernet and 4 only for Myrinet. The four meshes consist of 47219, 32888, 73260 and 289650 nodes, respectively.

For each sequential configuration, we report the sum of gather and file write time. For parallel configurations, we

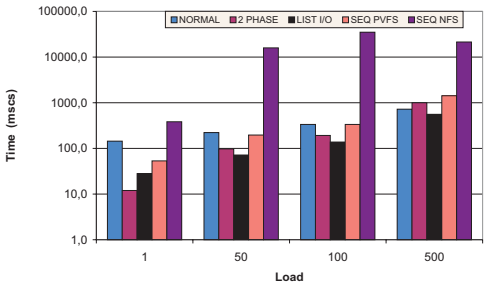


Figure 5. Examples of output techniques for $mesh3$ over Myrinet.

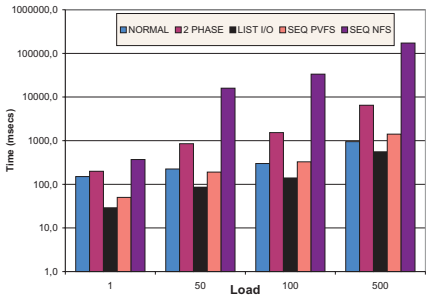


Figure 6. Examples of output techniques of Mesh 3/Fast Ethernet.

report the sum of the time to construct the data type and file write time.

All four meshes show simmilar results. Figures 5 and and 6 show the results of mesh 3 for 8 partitions. The scale is logarithmic.

It can be noticed that the worst performance is obtained for the sequential NFS configuration, which is the one used in the original BIPS3D version.

For Myrinet, when the data set is small, the most adequate configuration is the parallel two-phase I/O one. For larger sizes the best configuration turns to be list I/O. This is due to the fact that list I/O works with data intervals, which is more efficient for large data blocks. Additionally, for list I/O the data is sent only one time over the network, while for two-phase I/O twice. Therefore, the communicating data volume is much larger for two-phase I/O. On the other hand, for small intervals, the data block management is larger and the locality smaller for list I/O. In this case two-phase I/O is more adequate to utilize, because the network is not congested and the overhead of block management lower.

For Fast Ethernet, the most adequate I/O configuration is list I/O in all the cases. This is due to the fact that Fast Ethernet is considerably slower than Myrinet, which results in a penalty for all the configurations requiring a high number of communication operations, like two-phase I/O. Another good performing configuration is sequential PVFS, due to the small number of communication operations involved. However, this configuration will not scale for larger clusters, due to the central point of data gathering.

Based on the results from this subsection, we have constructed a decision tree shown in Figure 7. This tree helps choosing the adequate configuration based on the network type, mesh size and data set size.

As shown in Figure 7, when working with a fast network like Myrinet, for networks smaller than 70,000 nodes (Nx = 70,000) and loads smaller than 90 (Nld = 90), the best configuration is two-phase I/O. If the load is larger than Nld, the proper configuration is list I/O. If the number of nodes

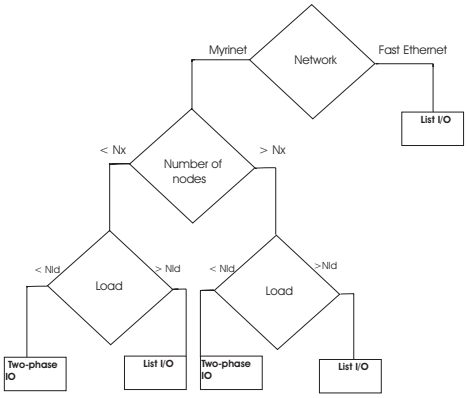


Figure 7. Decision tree for choosing the adequate configuration.

is larger than Nx, for loads smaller than 50 (Nld = 50) the adequate configuration is two phase I/O and for loads larger than Nld list I/O is recommended. Finally, for Fast Ethernet, the best configuration is list I/O.

6.2 IDG performance evaluation

We have divided this subsection into three parts: a comparison of list I/O based IDG technique with other parallel I/O methods, an in-depth analysis of the IDG performance and the evaluation of the IDG overhead.

6.2.1 Improvement of I/O performance

We have used IDG technique together list I/O technique. This technique was selected because it showed on average the best performance in the evaluation from the previous subsection. We have compared the performance of IDG with other three strategies: Metis, Random and First Position.

Metis uses the original node distribution performed by Metis for parallel I/O.

Random approach consists of the first stage of IDG technique (node classification) and a variant of the second stage (disk access scheduler). In this second stage variant, each shared node is assigned to a potential partition randomly.

Finally, First Position uses the same first stage of IDG, but, unlike the Random strategy, instead of choosing a random value, it chooses the smallest partition (among all possible). The reason for developing these two variants is to evaluate the load balance effect on our proposal. Random strategy will produce more balanced disk schedules than IDG technique, and, in contrast, First Position will produce the worse schedules.

Table 3 shows the characteristics of the considered input meshes. All of them correspond to semiconductor devices used by BIPS3D simulator.

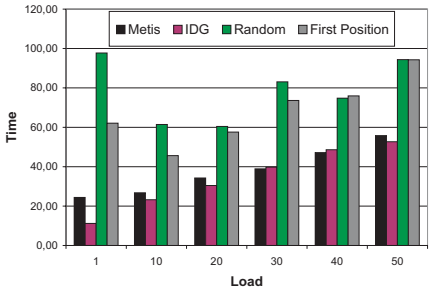
Figure 8 shows the performance of the I/O stage using these proposals together with List I/O technique. Each figure corresponds to an input mesh where different load parameters are evaluated. We can see that IDG is the most competitive approach in almost all the considered scenarios. We can also observe than the larger number of nodes is, the more the performance increases for IDG. We can also notice that both Random and First Position show poor results. Next section analyses the reasons of these performance results.

6.2.2 Performance evaluation

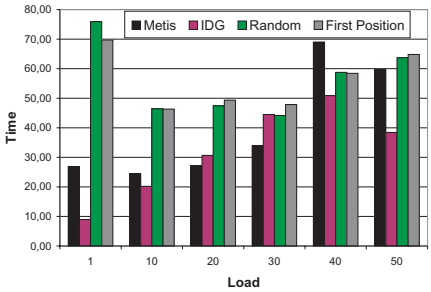
We have developed different tests for evaluating the scheduler performance. First, we evaluated the effect of data aggregation. Table 4 shows the number of disk intervals and average length for each mesh using Metis and IDG distribution for 8, 16 32, and 64 partitions. One interval is defined as a set of consecutive nodes that are assigned to the same partition. That is, a set of consecutive entries (as many entries as the number of nodes of the interval multiplied by the load factor) that are written to disk by the same compute node. We can see that IDG approach drastically increases the average length and reduces the number of intervals. We can also observe that this improvement increases for large numbers of partitions. The reason of this effect is shown in Figure 9.This figure shows, for each input mesh, the percentage of shared nodes from the overall number of nodes for 8, 16, 32 and 64 partitions. As expected, the replication percentage drastically increases with the number of processors. As we can see, for large numbers of partitions, IDG

Mesh	1	2	3	4
Nodes	47219	32888	73260	289650
Vertices	305120	210437	416950	2027885

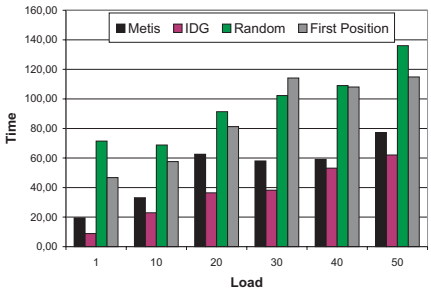
Table 3. Main characteristics of the input meshes.



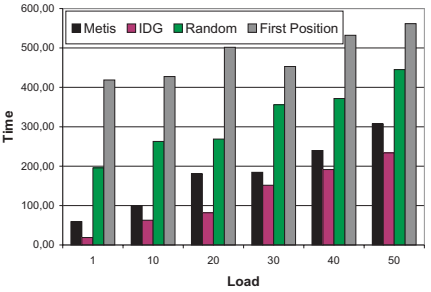
(a)



(b)



(c)



(d)

Figure 8. Performance of the List I/O for 8 processor execution: (a) $mesh1$, (b) $mesh2$, (c) $mesh3$ and (d) $mesh4$.

Nr. of partitions	Mesh	Metis 1	Metis 2	Metis 3	Metis 4	IDG 1	IDG 2	IDG 3	IDG 4
8	Nr. of intervals	258.0	203.8	244.0	965.5	88.3	67.1	57.0	126.0
	Average length	22.5	19.8	96.6	38.7	70.6	63.1	251.8	306.5
16	Nr. of intervals	217.4	171.3	189.8	642.4	71.6	58.3	49.5	99.8
	Average length	13.4	11.5	27.9	29.7	43.5	37.9	151.2	195.4
32	Nr. of intervals	169.6	136.7	149.4	511.7	54.5	44.9	38.7	75.2
	Average length	8.0	6.9	15.8	19.3	28.1	24.6	73.6	141.5
64	Nr. of intervals	137.0	99.1	112.9	344.4	41.2	31.7	25.4	51.7
	Average length	4.5	4.3	10.3	13.7	19.1	17.5	55.1	99.2

Table 4. Comparison of number of intervals and average length for Metis and IDG.

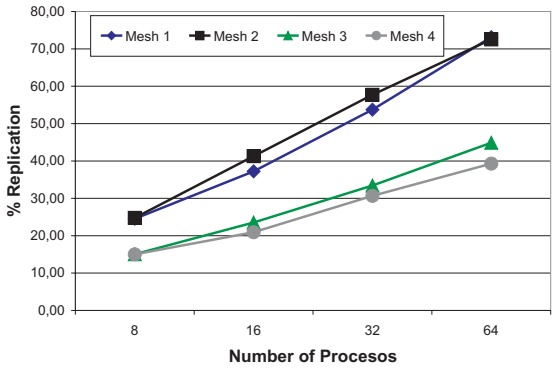


Figure 9. Percentage of data replication for different meshes and partitions.

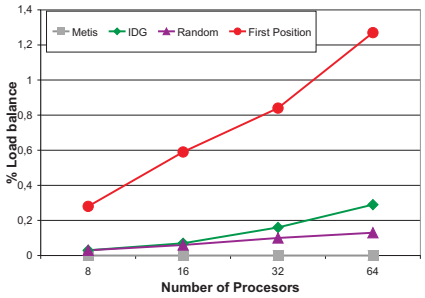


Figure 10. Percentage of load balance for different partitions of $mesh3$.

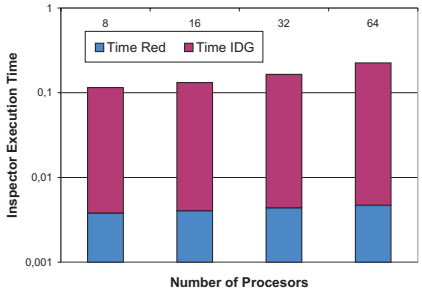


Figure 11. Computation time (in secs.) of IDG algorithm of $mesh3$.

technique has more shared nodes to group, therefore, obtaining larger data intervals.

Another important aspect is load balance. Figure 10 shows the percentage of load balance for mesh 3 and all partitions. This percentage was computed with the following formula:

$$LB = \frac{max(assigned\_nodes_i) - min(assigned\_nodes_i)}{avg(assigned\_nodes_i)} \quad (1)$$

where $assigned\_nodes_i$ represents the number of mesh nodes assigned to each computational node (i) for a disk write operation. Max, min and avg compute, respectively, the maximum, minimum and average value for all the existing compute nodes. We can note that Metis obtains the best results in all the considered scenarios, followed by the Random and IDG approaches. On the other hand, First Position run results in a poor load balance. IDG produces good balanced distributions when the number of partitions is not very large. For 8 partitions, IDG generates more unbalanced accesses than Metis. Note that load balance has not an important impact on the disk access performance, for instance when comparing the performance of Random and First Position in Figure 8. The latter one is considerably more unbalanced, but obtains better performance results than Random approach. Similar results were obtained when comparing IDG and Metis.

6.2.3 IDG overhead

We have evaluated the IDG overhead by using two metrics: CPU time and the memory consumption. Figure 11 shows the inspector execution time (in secs.) for mesh 3 and all partitions using logarithmic scale. For each measurement we divided the time of each stage into a node classification

component (labeled *Time red* in the figure) and a disk access scheduler component (labeled time IDG). We can see that the former one in negligible if compared with the latter. The overall execution time of IDG algorithm is very small. Also, it is important to remark, that the IDG inspector is applied once (for a given mesh partition) and its information can be reused during different disk write operations[3]. In this figure we also notice that the inspector overhead linearly increases with the number of partitions, which demonstrates the scalability of the solution.

7 Conclusions

In this paper we presented the optimization and evaluation of parallel I/O operations for BIPS3D parallel irregular application. First of all we showed how we parallelized the file access operation by using the partitioning information provided by Metis library. Then, we introduced a novel technique, called *Interval Data Grouping* (IDG), which exploits the data replication of mesh nodes for scheduling disk accesses in order to improve the performance of the parallel output operation.

In the evaluation section we have evaluated several existing I/O techniques and found out that list I/O performed the best for BIPS3D application. Using list I/O, we have compared IDG with three different I/O strategies, including the initial Metis based strategies. IDG performed better in most of the cases, even though it did not achieve the best load balance. This is due to the high locality of write operations, which, for this application, is shown to play a more important role than the load balance.

References

[1] R. Bordawekar. Implementation of Collective I/O in the Intel Paragon Parallel File System: Initial Experiences. In *Proc. 11th International Conference on Supercomputing*, July 1997. To appear.

[2] J. del Rosario, R. Bordawekar, and A. Choudhary. Improved parallel I/O via a two-phase run-time access strategy. In *Proc. of IPPS Workshop on Input/Output in Parallel Computer Systems*, 1993.

[3] http://www.unix systems.org/. *The Portable Operating System Interface*, 1995.

[4] G. Karypis and V. Kumar. METIS — A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis, 1998.

[3]This is very useful for checkpointing, where data is repeatedly stored to disk using the same pattern.

[5] D. Kotz. Disk-directed I/O for MIMD Multiprocessors. In *Proc. of the First USENIX Symp. on Operating Systems Design and Implementation*, 1994.

[6] W. Ligon and R. Ross. An Overview of the Parallel Virtual File System. In *Proceedings of the Extreme Linux Workshop*, June 1999.

[7] A. Loureiro, J. Gonzlez, and T.F.Pena. A parallel 3d semiconductor device simulator for gradual heterojunction bipolar transistors. *Journal of Numerical Modelling: electronic networks, devices and fields*, 16:53–66, 2003.

[8] Message Passing Interface Forum. *MPI2: Extensions to the Message Passing Interface*, 1997.

[9] R. Sandberg, D. Goldberg, S. Kleinman, D. Walsh, and B. Lyon. Design and implementation of the sun network filesystem. In *Proc. of the Summer USENIX Conference*, 1985.

[10] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of FAST*, 2002.

[11] K. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective I/O in Panda. In *Proceedings of Supercomputing '95*.

[12] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. In *Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189, February 1999.

[13] R. Thakur, W. Gropp, and E. Lusk. On Implementing MPI-IO Portably and with High Performance. In *Proc. of the Sixth Workshop on I/O in Parallel and Distributed Systems*, pages 23–32, May 1999.