On-Line Processing Model for Data Mining in Large Scientific Simulations [*]

Ying Liu [†] Wei-keng Liao [†] Steve Chiu [†] Alok Choudhary [†]

Abstract

With an increasing number of scientific applications manipulating enormous amounts of data, effective high-level data management has become an important requirement. Currently, data mining, as one stage in the sequence of stages during a scientific simulation cycle, is performed off-line for most applications, which involves a huge amount of data movement, I/O cost and initialization overhead. Therefore, the techniques to embed data mining within an on-line framework that consumes the data while the data are being produced can reduce the computational complexity and costs. In this paper, we present an on-line model of embedding data mining algorithms within the scientific simulation framework, where such algorithms are executed within the simulation cycle until the simulation satisfies certain application-dependent stopping criterion. The potential benefits of our on-line model are evaluated by applying the model to a parallel cosmological simulation.

1 Introduction.

Many large-scale scientific applications are data-intensive, processing large data sets ranging in size from megabytes to terabytes, such as applications in cosmology, astrophysics and hydrodynamics. Scientific simulations involve lengthy computation and produce enormous amounts of data. Figure 1 shows a typical scientific simulation cycle [3]. The cycle starts with domain decomposition, followed by the simulation stage. Then, various tools and algorithms are used during the data analysis and visualization stages. Finally, based on the results from these two stages, parameter adjustment for the next simulation cycle is performed. Output from one stage is used as the input for the next stage. Since the simulation produces an enormous amount of data, data management has become so overwhelming that scientists have to spend much time managing the data by using inefficient methods or by developing special-purpose solutions that may not work or scale with slight changes in the application or system configuration. Therefore, the design and development of scalable techniques, software and tools that address these problems becomes very important.

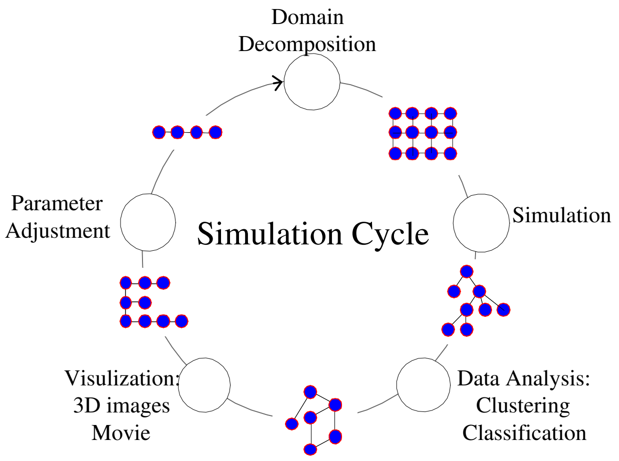

Domain Decomposition
Simulation
Data Analysis: Clustering Classification
Visulization: 3D images Movie
Parameter Adjustment
Simulation Cycle

Figure 1: A typical simulation cycle consists of several applications in which the data generated from one application is processed by another application. The white circles represent applications and the black circles connected in different structures represent the applications' output.

Data mining is becoming recognized as a promising data analysis method for scientific simulations. Data mining, also referred to as knowledge discovery, is a process of extracting implicit, previously unknown and potentially useful information from large data sets. It is commonly used in scientific computation, financial analysis, information retrieval, decision making, and World Wide Web, to name a few. Data mining techniques can be roughly categorized into classification, clustering, association rules, sequence mining, and similarity search.

However, existing data mining techniques have not considered the challenges posed by large-scale scientific simulations in terms of performance, scalability or ease of use. For example, classification, clustering or feature extraction is performed off-line and some parameters for subsequent computation are adjusted manually, which results in a major bottleneck. Storing the original simulation data is costly both in time and in media. Storing or retrieving the derived data for mining processes can easily overwhelm the mining process itself. Furthermore, the fact that data storage systems is often located remotely from the machines that run the mining applications makes the I/O cost even higher. Figure 2 illustrates an example of data flow for a cosmology simulation.

[*]This work was supported in part by DOE Lawrence Livermore National Laboratory under subcontract No.B524122/W-7405-ENG-48, the Scientific Data Management Center of the DOE SciDAC program under Contract No. W-31-109-ENG-38, NSF No. EIA-0103023, and a gift from Intel Corporation.

[†]Department of Electrical and Computer Engineering, Northwestern University. {yingliu, wkliao, schiu, choudhar}@ece.northwestern.edu

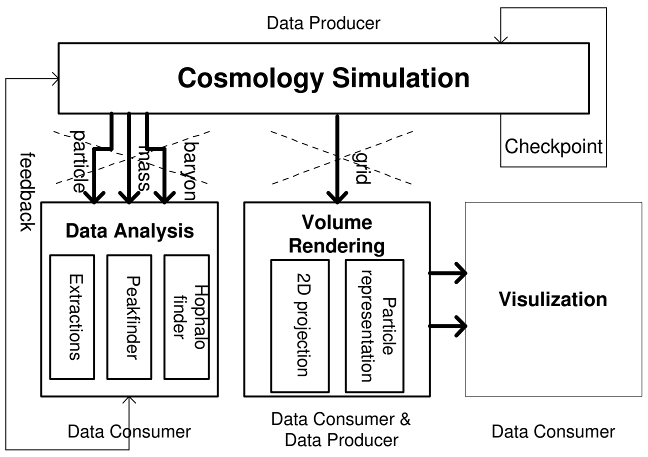


Figure 2: An example of data flow for cosmology simulation. Data analysis and visualization are performed off-line by domain scientists.

In this example, the cosmology simulation itself is the data producer, which dumps large amounts of data into disk files periodically at every checkpoint. Data analysis, as one of the data consumers, uses the data dumped from the simulation as input to perform certain data mining applications, then send the mining results back to the data producer, i.e. the simulation. However, such data analysis is performed off-line, and involves the complexity of data management, data storage and data movement.

Therefore, the concept of an on-line processing framework to automatically connect the data producer and data consumer becomes very desirable. An on-line model can reduce data movement, I/O access cost and computational complexity. Such a model can also automatically adjust the parameters for the subsequent simulation cycle. This way, the domain scientists would be relieved from the burden of manually executing the data analysis applications within the simulation cycle.

In this paper, we present an on-line processing model to perform data mining automatically within the simulation cycle until the simulation satisfies certain user-provided stopping criterion as defined by the domain scientists, e.g. total number of data dumps or converging number of clusters of stars or galaxies. Our model involves periodically checking specific parameters in the data evolution, collecting the data required by the ensuing data mining algorithm, invoking the data mining procedure and sending mining results to the subsequent processing stage. The potential benefits of this on-line model are evaluated by incorporating a clustering algorithm into a parallel cosmological application called ENZO. Our experiments demonstrate that the costs for I/O access and initialization overheads are significantly reduced by up to 57% on the IBM SP2 with the on-line model.

The remainder of this paper is organized as follows. Section 2 describes our on-line processing model. Section 3 discusses the cosmological simulation cycle and presents our experimental results. In Section 4, we discuss the difficulties in our study and future work items. We conclude the paper in Section 5.

2 On-line Data Mining.

Based on the large-scale simulation data flow in Figure 2, we propose an on-line data mining framework that embeds the data mining applications within the simulation cycle.

Since the characteristics of each simulation varies, there are two strategies to perform data mining applications on the output data from the simulation stage: one is to execute the data mining applications when the data is just beginning to be produced, that is, to perform incremental or evolving data mining techniques on the dynamic data; the other option is to wait until all the data is available in the memory. Each of these strategies has advantages and disadvantages. For the first strategy, the data processing is pipelined, thus saving memory and I/O cost and reducing overall execution time. However, the mining quality would be poor when the data evolves considerably over time. In addition, there are no well-established data mining techniques on incremental or evolving data at the present time. For the second strategy, data mining applications cannot start until all the data are available in the memory, which might result in out-of-memory problems when the data size is large. The choice between the strategies depends on the user or the particular application. In either case, the data mining application can be plugged into the simulation cycle with no extra I/O cost.

Figure 3 illustrates the flow of our on-line processing model, with the major steps itemized as below.

Step 1. Typically, a scientific simulation evolves through many cycles until the pre-defined stopping criterion is satisfied. During each cycle, data are periodically dumped from the simulation stage and fed into the data mining applications to discover the knowledge hidden in the data, and determine the parameters of the next step of the evolution. Whenever data are dumped, a runtime library is invoked to extract the data required by the specific data mining algorithm in use, such as 3-dimension coordinates, particle masses and other user input parameters, as well as filtering non-required data.

Step 2. Performing the data mining applications within the simulation cycle avoid any I/O cost, data movement or initialization overhead associated with off-line methods. In addition, while many data mining algorithms have been implemented in parallel, some scientific simulations are not designed for parallel processing. If the data mining application is implemented in parallel, in order to achieve high performance, a runtime library is invoked to distribute the data across multiple processors and start the parallel computation.

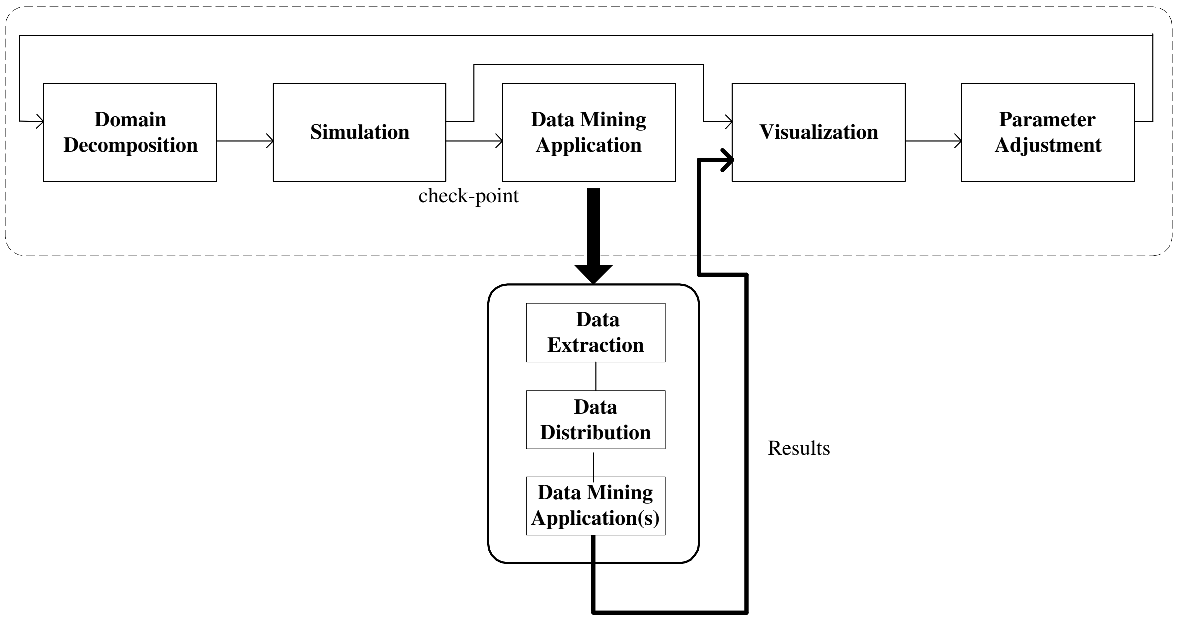


Figure 3: On-line processing model for data mining in scientific simulations. The data mining applications are plugged into the simulation cycle and will send mining results to the subsequent stage.

Step 3. The data mining algorithm is executed to discover important knowledge from the data sets. The mining results are important to scientists since these results present a global view of the current state of the simulation. It helps the scientists understand the underlying scientific phenomena. With our on-line processing model, scientists can obtain mining results automatically and immediately during the simulation cycle, thus avoiding the cumbersome work performed by a separate set of tools in the off-line model. In addition, the mining outcome (which is sent to the next stage) may contribute significantly to help steer subsequent simulation cycles.

Based on the afore-mentioned steps, our proposed model is designed to be portable and easy to use. It adds no complexity to the simulation cycle, and it hides all the detailed data access processes that are not of primary importance to the end user.

3 Evaluation of the On-Line Processing Model.

To evaluate the significance of our on-line processing model, we use ENZO, a real production cosmology simulation developed at NCSA [2], along with the HOP clustering algorithm proposed in [4]. Designed to show the evolution of the galaxy formation, periodic data dumps are performed during ENZO evolutions. The dumped data are input into subsequent data mining processes, where patterns such as clusters of stars could be discovered. In the current implementation of ENZO, such clustering application is performed off-line. We discuss these major components of our evaluation and present our experimental results as follows.

3.1 ENZO Cosmology Simulation. ENZO is a three-dimensional parallel application that simulates the formation of a cluster of galaxies consisting of gas and stars [9]. The simulation starts near the beginning of the universe, a few hundred million years after the big bang, when the galaxy is in a relative uniform radiation distribution, and continues till the present day, when it is in a highly irregular star particle distribution. ENZO is used to test theories of how galaxy and clusters of galaxies form by comparing the results with what is really observed in the sky today [5], where simulation results from ENZO compared favorably with high precision against established astrophysical models [2]. One core technique of ENZO is to use *Adaptive Mesh Refinement* (AMR) to partition recursively the problem domain into subdomains. That is, a single grid covers the entire computational volume; and in regions that require higher resolution, a finer subgrid is added. If a region requires yet a higher resolution, an even finer subgrid is added [8]. This process repeats recursively with each adaptation, resulting in a tree of grids as shown in Figure 4. Check-pointing is performed periodically to save current results into files, which enables the application to be resumed. In the process of galaxy evolution, the grids are refined dynamically and, therefore, may result in a hierarchy with a different number of levels and new structure. Each grid contains particle coordinates in three dimensions, particle masses, gradient fields, baryon fields, etc. A grid can only be owned by one processor,

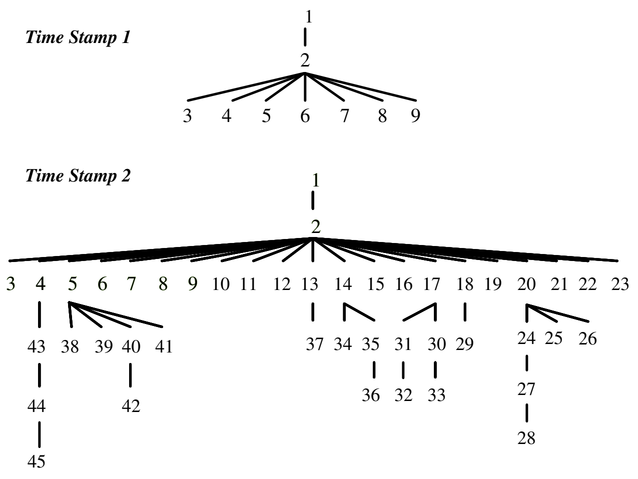


Figure 4: Spatial relationship of the grid data generated by a cosmological simulation using the AMR method. Two hierarchical trees depict the data set relationship at two time stamps, respectively.

but one processor can own multiple grids. During checkpointing, each grid in a processor is dumped to a separate file [6]. Subsequent data mining applications are performed on the simulation output. Both the galaxy simulation output and the data mining results are processed by a visualization application to render images of galaxies and clusters of galaxies. By visually examining the rendered images, astrophysicists can digest the discovered knowledge hidden in the data, adjust the simulation parameters, and then proceed to the next simulation cycle. As shown in Figure 1, the simulation cycle continues until its stopping criterion is satisfied.

3.2 HOP Clustering Algorithm. Clustering, an important data mining technique, groups a set of data based on the conceptual clustering principles: maximizing the intra-class similarity and minimizing the inter-class similarity. Clustering algorithms are widely used in many fields to group data with similar attributes or to describe various dense regions in the output of a simulation, such as cosmology, astrophysics and geology.

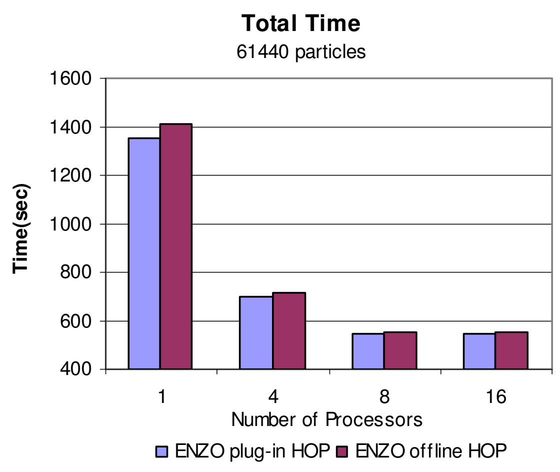
HOP [4], proposed for cosmological simulations, is a density-based clustering algorithm. The density of a particle is estimated by its N_{dens} nearest neighbors, where N_{dens} is a user-provided parameter. Having assigned to every particle an estimate of its local density, HOP associates each particle with the densest neighbor of its N_{hop} nearest particle neighbors, where N_{hop} is also a user-provided parameter. All particles that associate to the same densest particle constitute a single cluster. The input data for HOP are 3D coordinates, particle masses and several user specified parameters. A portable and scalable parallel HOP algorithm [7] has been implemented which distributes the data across processors evenly, and accesses remote data through communication. This algorithm is applicable to many fields, where large data sets are to be processed with similar clustering or neighbor-finding procedure. Therefore, we use this parallel HOP algorithm as the data mining application in our evaluation. HOP has been designed to perform clustering after all the data are dumped.

The output data of ENZO is grid, while the data structure in the clustering algorithm, i.e. HOP, is KD tree [1]. KD tree is a balanced tree that partitions the spatial domain recursively along the longest axis into sub-domains. Each sub-domain contains approximately the same number of particles. The root node represents the entire simulation domain that covers all the particles, and each tree node represents a sub-domain of its parent node. Only the leaf nodes contain the particle data. One key property of a KD tree is that particles spatially closed are located in the same bucket or sibling buckets of the same tree branch. In order to embed HOP clustering within the simulation cycle, we have two options: one is to convert the data structure from grid to KD tree before HOP clustering begins; the other is to perform HOP clustering algorithm using grid structure.

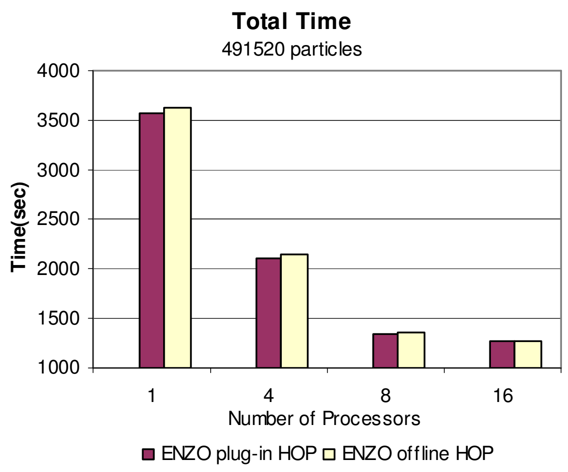
3.3 Experiments and Results. The potential benefits of our on-line processing model are evaluated by running ENZO on the 375MHz Power3 IBM SP2 at the San Diego Supercomputing Center. In our experiments, we used two sets of simulation data from ENZO: data set 1 contains 61,440 particles the attributes of which consist of three-dimensional Euclid coordinates and particle mass, and data set 2 contains 491,520 particles specified by the same attributes. We varied the number of processors from 1 to 16 to study the scalability of our on-line model. We experimented with both options as discussed in Section 3.2, and observed that the overall performance of the simulation with embedded HOP using grids is worse than that with embedded HOP using KD trees, when the number of grids increases dramatically (as the universe evolves). The grids' unbalanced nature resulted in a huge amount of unnecessary neighborhood searching. Therefore, we decided to proceed with our performance evaluation by running HOP using KD tree. The measurement metrics we used are as follows:

Total time: For ENZO with plug-in HOP, it is the running time from the beginning of the simulation to the end, including computation time, I/O time for 8 data dumps, and the time for 8 plug-in HOP clustering runs. For ENZO with off-line HOP, it is the same as the above except to replace the 8 plug-in HOP runs with 8 off-line HOP runs for each data dump.

Total I/O time: For ENZO with plug-in HOP, it is the I/O cost for 8 data dumps. For ENZO with off-line HOP, it is the sum of the I/O cost for 8 data dumps plus the I/O cost

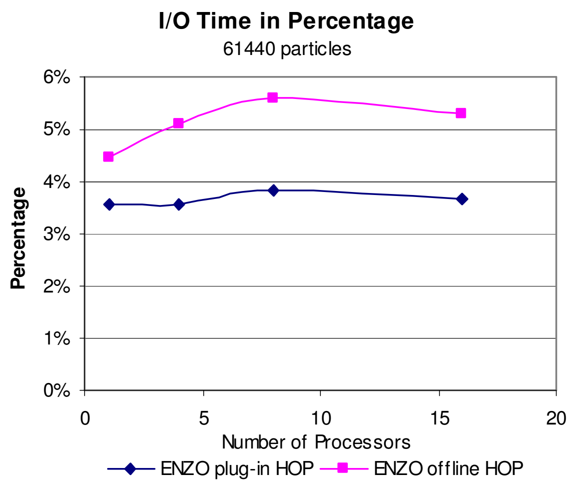


(a) Data set 1

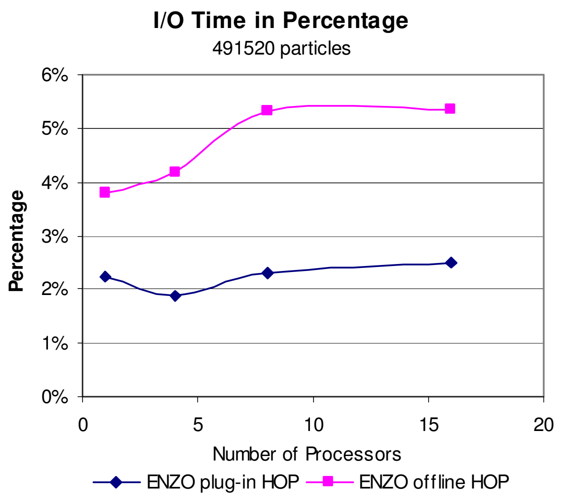


(b) Data set 2

Figure 5: Total execution time on IBM SP2. Running ENZO with plug-in HOP is faster than running ENZO with the off-line HOP.

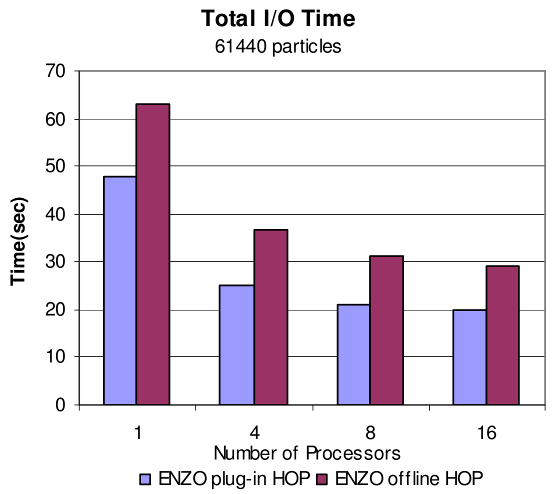


(a) Data set 1

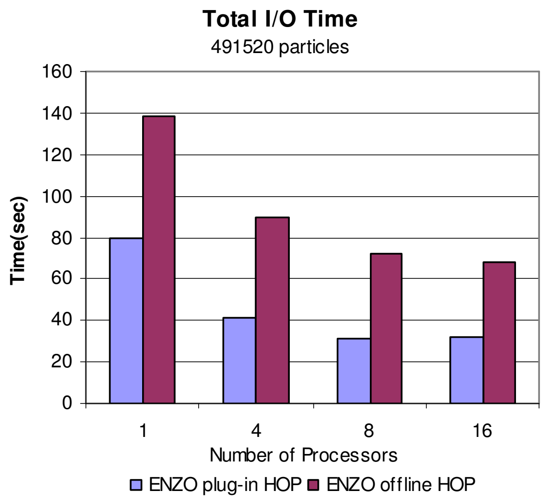


(b) Data set 2

Figure 6: I/O time as a fraction of the total execution time on IBM SP2.



(a) Data set 1



(b) Data set 2

Figure 7: Total I/O time on IBM SP2. Our on-line processing model significantly reduced the I/O cost and initialization overhead.

and initialization overhead for 8 off-line HOP runs.

Figure 5 presents the measured total execution times of ENZO with plug-in HOP versus ENZO with off-line HOP for the two data sets on multiple processors. When the number of processors is larger than 1, the I/O operation is performed in parallel. For both data set 1 and data set 2, we observed that the total execution time with plug-in HOP is smaller. Since the I/O cost in ENZO simulation represents less than 5% of the total execution time, as shown in Figure 6, the benefit on total time presented in Figure 5 is not very evident. However, for applications that are more data-intensive, the impact of such I/O cost reduction is expected to be much more significant. The benefit on data set 1 is more significant than data set 2 as we observed. That is, in the case of a smaller data set, the observed I/O time takes a larger fraction in the total execution time than the case of a larger data set, due to a higher fraction of processing time attributed to computation in the larger data set.

Since one of the main purposes of our on-line model is to reduce I/O cost and initialization overhead associated with the off-line approach, we measured the I/O time of both approaches. Figure 7 shows that the I/O cost is reduced by the on-line model for both data sets on 1 to 16 processors. Figure 8 presents the percentage reduction in I/O time provided by our on-line model on multiple processors, where we also observed that the performance is scalable. Figure 8 shows that in the case of the larger data set, the I/O time reduction is up to 57% of the total I/O cost incurred by ENZO with off-line HOP when running on 8 processors. For the smaller data set, the file open/close overheads happens more frequently when compared to the larger data set, which results in a lower percentage of reduction than that of the large data set. From

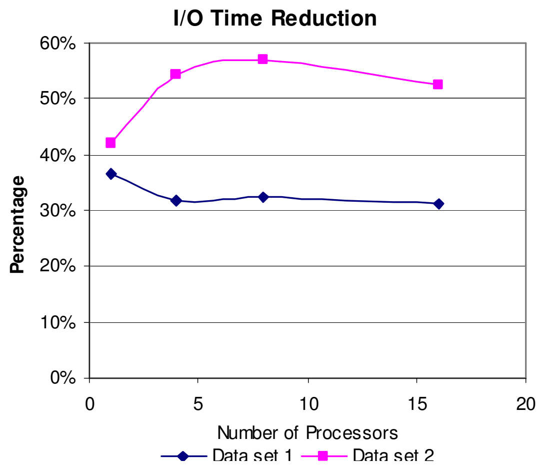


Figure 8: Percentage reduction in I/O time by the on-line processing model.

Figure 6, we see that the total I/O cost with respect to the total execution time is reduced by 1% to 2% in data set 1 and 1.5% to 3% in data set 2 when using our on-line (plug-in) HOP design.

Figure 9 illustrates the number of clusters identified by HOP during ENZO evolution. As the simulation time elapses, the number of clusters converges to a fixed number. That is, at the beginning of the universe, the entire galaxy is very uniformly distributed and sparse. The galaxy then continues further to become more and more irregular, generating more and more clusters/dense regions, until the end when it reaches a relatively steady state. If we consider converging to a fixed number of clusters to be the stopping criterion for the

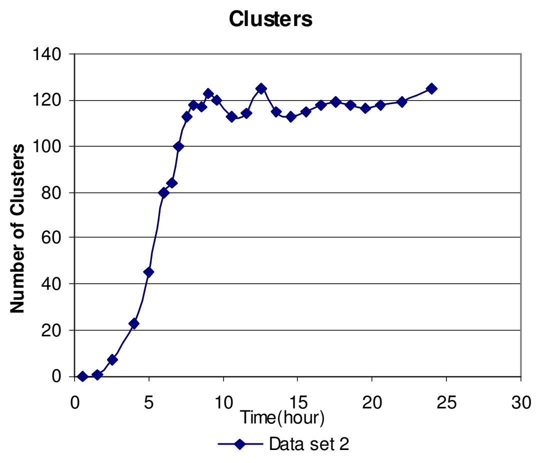


Figure 9: Number of clusters during evolution. It converges at the end of the simulation.

simulation cycles, then the simulation can be automatically stopped at certain point in time. In this way, scientists would be relieved from the burden of manually executing the HOP process to determine when the simulation will be finished.

4 Prospect and Future Work.

Since each scientific simulation is designed specifically to solve a particular problem, there are no universal data structures for all the simulations. For example, cosmological simulation uses dynamic grid, gene expression data analysis uses matrix, etc. On the other hand, the data structures in the existing data mining algorithms may not be compatible with the data structures used in the simulation. To solve this problem, we proposed two strategies: one is to convert the output of the simulation to the data structure that conforms to the input of the data mining application; the other is to implement the data mining algorithms using the data structure of the output of the simulation. Each strategy has advantages and disadvantages. For the first strategy, while it is relatively easy to convert the data structure, it would take so much memory to keep a duplicate copy of data at runtime that it may run out of memory or seriously affect the performance. In addition, the data mining algorithm cannot start until the conversion is done. For the second strategy, it involves more work to implement the existing data mining algorithms using a specific data structure. Essentially, the choice between the two strategies would be application specific. The challenge is to determine which strategy is more effective at run time. Currently, we compare the performance results manually.

In our example, the output data of the cosmological simulation is grid, while the data structure used in the clustering algorithm, HOP, is KD tree. To embed HOP clustering within the simulation cycle, we experimented with two methods: one is to convert grid to KD tree before HOP clustering begins; the other one is to perform HOP clustering algorithm using grid structure. We found that the performance of the simulation with embedded HOP using grid deteriorates when the number of grids increases dramatically. As discussed before, the grids' unbalanced nature results in a huge amount of unnecessary neighborhood searching.

For future work, we plan to explore the methods that can automatically determine which strategy is more effective at run time. In that case, no manual work would be needed within the entire simulation cycle, eliminating the need for scientists to supervise the simulation cycles. We also would like to investigate additional applications that are more data-intensive, as such applications are expected to benefit further from our on-line processing model.

5 Conclusions.

We proposed an on-line data mining model to automatically connect data producer and data consumer in the scientific simulation cycle. In our model, run-time libraries extract useful data for data mining applications, and then distribute data across multiple processors if necessary, and send mining results to the subsequent stage in the simulation cycle. Using our model, I/O cost and initialization overhead are reduced by connecting data producer and data consumer. In addition, the simulation can automatically repeat for a number of cycles until the pre-defined stopping criterion is satisfied.

We evaluated our on-line processing model by embedding HOP clustering algorithm within a cosmological simulation, ENZO. Two data sets were examined, both of which exhibited reduction in the execution time when scaled from 1 to 16 processors, achieving a 57% I/O time reduction with 8 processors. In addition, our experiments indicated that the number of clusters identified by HOP algorithm converges at some point of time during the simulation.

Although we applied HOP clustering to a cosmological simulation, the model for connecting data producer and data consumer in the scientific simulation cycle is applicable to most large-scale scientific simulations that involve data mining applications. Execution time can be reduced by avoiding data movement, data storage, and initialization overhead. For applications that are more data-intensive, the impact of I/O reduction provided by our on-line model is expected to be more significant. Scientists can benefit from the ease of use, portability and scalability provided by the on-line model.

Acknowledgement

We thank Dr. Michael Norman from UCSD and Dr. Greg Bryan from University of Oxford for providing the ENZO source code. We also thank Dr. Daniel Eisenstein and Dr. Piet Hut for the HOP source code and documentation.

References

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

[2] G. Bryan, T. Abel and M. Norman, *Achieving Extreme Resolution in Numerical Cosmology Using Adaptive Mesh Refinement: Resolving Primordial Star Formation*, Proceedings of the SuperComputing Conference, Nov. 2001.

[3] A. Choudhary, M. Kandemir, J. No, G. Memik, X. Shen, W. Liao, H. Nagesh, S. More, V. Taylor, R. Thakur and R. Stevens, *Data Management for Large-Scale Scientific Computations in High Performance Distributed Systems*, Cluster Computing: The Journal of Networks, Software Tools, and Applications. Baltzer Science Publishers, 3(2000), pp. 45–60.

[4] D. J. Eisenstein and P. Hut, *Hop: A New Group Finding Algorithm for N-body Simulations*, J. Astrophysics, 498 (1998), pp. 137–142.

[5] J. Li, W. Liao, A. Choudhary and V. Taylor, *I/O Analysis and Optimization for an AMR Cosmology Application*, in Proceedings of Cluster Computing, Sep. 2002.

[6] W. Liao, G. K. Thiruvathukal and A. Choudhary, *A Framework for Large-Scale Scientific Data Management and Knowledge Discovery*, Technical Report CPDC-TR-2002-02-001, Northwestern University, 2002.

[7] Y. Liu, W. Liao and A. Choudhary, *Design and Evaluation of a Parallel HOP Clustering Algorithm for Cosmological Simulation*, in Proceedings of International Parallel and Distributed Processing Symposium, Nice, France, Apr. 2003.

[8] Z. Lan, V. Taylor and G. Bryan, *Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications*, in Proceedings of the 30th Internal Conference on Parallel Processing, 2001.

[9] M. Norman, J. Shalf, S. Levy and G Daues, *Diving Deep: Data-Management and Visualization Strategies for Adaptive Mesh Refinement Simulations*, Computing in Science and Engineering, 1(4), pp. 36–47, Jul./Aug., 1999.