Final Report
March 31, 2007

Center for Programming Models for Scalable Parallel Computing (UIUC subgroup) DOE DEFC 02-01 ER25508 September 15, 2001 --- September 14, 2006

Marianne Winslett
Department of Computer Science
University of Illinois
201 N. Goodwin Avenue
Urbana, IL 61801
Work: (217) 333-3536
FAX: (217) 265-6494
winslett@uiuc.edu

Michael Folk
National Center for Supercomputing Applications
University of Illinois
605 East Springfield Avenue
Champaign, IL 61820
Work: (217) 244-0647
mfolk@ncsa.uiuc.edu

1 Summary

The mission of the Center for Scalable Programming Models (Pmodels) was to create new ways of programming parallel computers that are much easier for humans to conceptualize, that allow programs to be written, updated and debugged quickly, and that run extremely efficiently---even on computers with thousands or even millions of processors. At UIUC, our work for Pmodels focused on support for I/O in a massively parallel environment, and included both research and technology transfer activities:

1. Research
    a. Speeding up writes through active buffering
    b. Data-format-independent data management
    c. Data-format-independent buffer management
    d. Bitmap indexes
    e. Log-based I/O
    f. I/O-aware compilation

2. Technology transfer: making new I/O techniques available to a wider audience.
    a. Log-based I/O (published on SourceForge)
    b. Data migration (incorporated into the ROMIO parallel I/O library)
    c. Bitmap indexes (available to HDF users)
    d. Metadata handling (incorporated in HDF)
    e. Arithmetic operations during I/O (incorporated in HDF), and support for formulas in HDF

3. Synergistic activities with DoE

We describe each of these activities in more detail below, including lists of the publications that resulted from each activity.

2 Active Buffering

This grant supported our work on *active buffering*, a technique that exploits idle resources on parallel computers to hide the cost of writes from parallel application programs. We developed active buffering in response to the I/O needs of GENx, a state of the art parallel rocket simulation code. Active buffering reduces or even eliminates the application-visible cost of I/O, by taking advantage of available idle memory on processors. Experiments with GENx and other applications showed that active buffering is very effective, as documented in the papers listed below.

Working with the HDF developers in a technology transfer activity, we concluded that the best place to put active buffering was in the ROMIO layer that underlies HDF. Because ROMIO's architecture is significantly different from Panda's, we reimplemented active buffering in ROMIO and evaluated its performance. The results of the evaluation, as well as other recent results on active buffering, were presented in papers listed below. Overall, we found that active buffering provides significant potential performance upside for ROMIO clients, with little overhead. Our implementation resided in the platform-independent layer of ROMIO, thus making active buffering available on all ROMIO platforms without extra effort.

To make these facilities available in a released version of ROMIO, active buffering had to work with *all* the calls in ROMIO's API---not just with collective writes, as in our earlier work. For example, what happens if a processor writes a piece of data collectively, then reads it back in a non-collective call, and the data are still in the active buffers at the time of the second call? We determined that active buffering violated several undocumented concurrency control assumptions embedded in the ROMIO library, regarding the semantics of possible sequences of collective and non-collective calls that write and then read the same piece of data. For that reason, active buffering will not be incorporated in the released version of ROMIO.

Publications:

X. Ma, M. Winslett, J. Lee, and S. Yu. Faster Collective Output through Active Buffering. In Proceedings of the 2002 International Parallel and Distributed Processing Symposium. April 2002. (Introduces active buffering)

J. Lee, X. Ma, M. Winslett, and S. Yu. Active Buffering Plus Compressed Migration: An Integrated Solution to Parallel Simulations' Data Transport Needs. In Proceedings of the 16th ACM International Conference on Supercomputing, June 2002. (Active buffering combined with data migration)

Xiaosong Ma. Hiding Periodic I/O Costs in Parallel Applications. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 2003.

X. Ma, J. Jiao, M. Campbell, and M. Winslett. Flexible and Efficient Parallel I/O for Large-Scale Multi-component Simulations. Proceedings of The 4th Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications, in conjunction with the 2003 International Parallel and Distributed Processing Symposium, April 2003. (Application experience with active buffering)

X. Ma, M. Winslett, J. Lee, and S. Yu. Improving MPI-IO Output Performance with Active Buffering Plus Threads. Proceedings of the 2003 International Parallel and Distributed Processing Symposium, April 2003. (Active buffering in a server-less architecture, such as ROMIO's)

X. Ma, J. Lee, M. Winslett, "High-level buffering for hiding periodic output cost in scientific simulations," *IEEE Transactions on Parallel and Distributed Systems*, volume 17, issue 3, March 2006, pp. 193-204.

3 Data-format-independent Data Management

Dramatic improvements in scientific instruments, as well as increasingly realistic simulations on massively parallel computers, have resulted in massive amounts of data and their associated data management problems. The large amounts of data necessitate specialized data storage formats to store and retrieve data in reasonable amounts of time. Several generic storage formats (e.g., HDF, netCDF, CGNS) and systems specialized for particular scientific realms (e.g., ROOT, FITS) have been developed to solve this problem. To analyze and query data stored in such formats, the application developer (e.g., visualization tool developer or simulation code writer) has to write functions and programs using the interfaces available for those formats. This is a very time consuming process, and the resulting programs are specific to the format used. Furthermore, usually the application developer is a scientist who would rather spend time investigating scientific questions instead of dealing with data management headaches. Such developers need a system that provides data-format-independent, application-tailorable, loosely-coupled facilities for buffering, caching, indexing, querying, concurrency control and metadata management. The goal of our Maitri data management system is to provide such facilities.

Maitri provides a set of lightweight, mix-and-match components that developers can combine as needed to provide holistic data management facilities for a particular application. To provide format independence, Maitri introduces the concept of user-defined logical blocks. In traditional data management systems, a block consists of one or more logically consecutive blocks of a file; but in Maitri a block is a user-defined aggregate of data. The user defines the functions required to extract data from these logical blocks. The internals of the Maitri system operate on these blocks and hence require no knowledge of the format of storage. This provides the user with format independence, at the cost of having him or her write the interfaces to access data from the logical blocks. User-defined logical blocks were used to provide format independence in the GODIVA buffer manager developed under this grant in previous years, and an extended version of GODIVA is the buffer manager for Maitri, as discussed below. Maitri also uses the bitmap indexing facilities discussed below. The publications listed below focus on the overall architecture of Maitri; publications that focus specifically on buffering or indexing in Maitri are discussed in subsequent sections of this report.

Publications:

R. R. Sinha, S. Mitra, and M. Winslett, Maitri: Format Independent Data Management System for Scientific Data. *Proceedings of the 3rd International Workshop on Storage Network Architecture and Parallel I/O*, 2005.

R. Sinha, A. Termehchy, and M. Winslett, Maitri: Managing Large Scale Scientific Data, demonstration and paper at *Conference on Innovative Database Research*, January 2007.

4 Data-format-independent Buffer Management

We designed, implemented, and evaluated light-weight buffering facilities that are intended for use in I/O-intensive programs such as visualization facilities. These buffering facilities make no assumptions about the format of the data being buffered; instead, the application developer writes a small amount of code that describes how to read and write data in the underlying storage format. In return, the application developer has at their disposal a library of traditional buffering commands and hints, allowing them to describe which data should be fetched or prefetched into memory and when. Hints also allow the application developer to describe which data can or must be written back to disk, and when. With such a library, an application developer need not develop his or her buffering facilities.

The resulting buffering library, GODIVA, has been evaluated in the context of a parallel visualization tool, ROCKETEER, used by the Center for Simulation of Advanced Rockets at UIUC. GODIVA simplified data management for the ROCKETEER authors and improved I/O performance by 25-45% in early tests. GODIVA was released in a subsequent version of ROCKETEER. In addition, an extended version of GODIVA forms the buffer management and block management layers of Maitri.

Publication:

X. Ma, M. Winslett, J. Norris, J. Jiao, and R. Fiedler. GODIVA: Lightweight Data Management for Scientific Visualization. *Proceedings of the 20th International Conference on Data Engineering*, 2004.

5 Bitmap indexes

Bitmap indexes were developed to meet the need for quick data retrieval in data warehousing applications. Somewhat surprisingly, bitmap indexes also hold great potential for indexing scientific data---much more so than KD trees, R trees, quad trees, B-trees, and so on. These traditional indexes are not efficient in the high-dimensional data sets that are common in scientific research, as even multidimensional indexes like R-Trees degrade in performance if the number of dimensions is too high.

We created a new kind of bitmap index, called a multiresolution bitmap index, and adopted it as the indexing technology in Maitri. We showed that multiresolution bitmap indexes are very effective at reducing the run times for real and simulated applications, using data from the Sloan Digital Sky Survey and other sources. Further, multiresolution bitmap indexes occupy only about twice the space of the data that they index; thus they are much smaller than traditional tree-based indexes.

In subsequent conversations with researchers pursuing “big science”, we learned that the observational science with the highest data rates cannot afford to add more than 20-30% of the space occupied by data, in order to support indexing. In collaboration with Arie Shoshani’s group at LBL, we developed a way to cut the footprint of a bitmap index down to 25% of the size of the data being indexed, while still offering excellent data retrieval performance for real-world workloads. We dubbed the resulting technique *adaptive bitmap indexes*.

The code for adaptive and multiresolution bitmap indexes is being released through The HDF Group’s web site, and will be incorporated into HDF in the future.

Publications:

R. R. Sinha, S. Mitra, and M. Winslett, Bitmap Indexes for Large Scientific Data Sets: A Case Study. *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium, 2006*.

R. R. Sinha and M. Winslett, Multi-resolution Bitmap Indexes for Large Scientific Data Sets. Submitted to *ACM Transactions on Database Systems* in 2006.

R. R. Sinha, M. Winslett, J. Wu, K. Stockinger, A. Shoshani, Adaptive Bitmap Indexes for Space-Constrained Systems, submitted for conference publication. (Introduces adaptive bitmap indexes.)

6 Log-based I/O

The ever-increasing gap between the processor and I/O speeds of computer systems has become a problem for I/O intensive applications like scientific simulations. While the growing computation power has enabled scientists to simulate larger phenomena, storing the data to disk for subsequent visualization is increasingly problematic. The challenge is especially acute for cluster computers, which lack the kind of I/O and optimizations found in traditional supercomputers.

Within the Pmodels project, we worked to find ways to improve the I/O performance of these simulation applications by exploiting their special I/O characteristics. These simulations write a lot of data but do not read them back during the course of the simulation. Furthermore, different processes of the simulation either write to different files or disjoint locations of the same file, so concurrency control (order of writing) is also not an issue.

Our past work on active buffering, also supported by this grant, exploited these characteristics and the presence of idle resources on processors to *hide* the cost of I/O from the application. Our work on *log-based I/O* extends that idea in a new direction. Our approach is to asynchronously defer the file system write calls made by the application. We create a log of all the write calls executed by the simulation, at the C library level. We store this log on each processor's local disk, and asynchronously transfer/replay the log to the file server that is the ultimate destination of the data. The asynchronous copy operation prevents the file server from getting overloaded from simultaneous calls by the application. Our performance results with synthetic and real applications were quite good (see publication below). We incorporated log-based I/O into a small stand-alone library and released on SourceForge. The library is extremely easy to install and use; we have used it in our own subsequent projects as a quick-and-easy way to improve performance for write-intensive parallel applications.

Publications:

S. Mitra, R. R. Sinha, and M. Winslett, X. Jiao, An Efficient, Non Intrusive, Log Based I/O Mechanism for Scientific Simulations on Clusters. *Proceedings of the 21st International Conference on Cluster Computing*, 2005.

Log-based I/O (LBIO) release: http://sourceforge.net/projects/lbio/

7 I/O-aware Compilation

The idea behind *I/O-aware compilation* is to expand a parallel programming language to include simple high-level I/O commands (e.g., "output this array to this data repository"), and then make use of a compiler for that language that knows how to map those calls to library-specific I/O calls (e.g., calls to HDF or netCDF to write out the specified array), and also knows how to optimize such calls. For example, the compiler can turn such calls into asynchronous I/O calls that can be started as soon as possible and terminated as late as possible, to maximize the overlap between I/O activities and other program activities. Our group and the Pmodels Co-Array Fortran compiler group led by John Mellor-Crummey developed a plan for how to realize such calls in Co-Array Fortran, using Panda and HDF as the underlying I/O libraries.

In subsequent experiments, we determined that I/O-aware compilation does not offer performance benefits beyond those obtained by the use of active buffering, in typical scientific codes. The reason is that I/O-aware compilation moves all I/O-related activities to a separate thread, which runs in the background. However, many I/O activities in a high-performance code require communication between processors, typically because an array's in-memory distribution across processors is different from the desired distribution on disk. Moving these communication activities to the background hurts typical applications, because they have to wait for their turn to use the communication facilities. The resulting switching between threads can slow down overall application run time. We found that I/O-aware compilation did tend to increase run times slightly. In contrast, active buffering performs communication-intensive activities in the foreground, avoiding this slowdown problem.

As these results were negative, we did not seek to publish them. However, we recognized that I/O-aware compilation could still be helpful for out of core codes, and marked that as a possible future direction for research. Further, and with the potential for greater impact, we recognized that today's hyperthreaded architectures may make it possible for communication-intensive I/O activities to run in the background, without causing significant slowdowns for applications.

8 Data Migration

As part of the Pmodels project, we worked on technology transfer of our results on how to migrate data long distances effectively, using on-the-fly parallel data compression and other techniques. Through a postdoc shared with the ROMIO group at Argonne National Laboratory, we incorporated those facilities into ROMIO. These facilities form a new module at the ADIO level in ROMIO and are available to all ROMIO users.

Publication:

J. Lee, M. Winslett, X. Ma, and S. Yu. Enhancing Data Migration Performance via Parallel Data Compression. In Proceedings of the 2002 International Parallel and Distributed Processing Symposium. April 2002. (original data migration paper)

J. Lee, X. Ma, M. Winslett, and S. Yu. Active Buffering Plus Compressed Migration: An Integrated Solution to Parallel Simulations' Data Transport Needs. In Proceedings of the 16th ACM International Conference on Supercomputing, June 2002. (Active buffering combined with data migration)

Jonghyun Lee. Supporting I/O for Remote Visualization of High-Performance Scientific Simulations. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, May 2003.

J. Lee, X. Ma, R. Ross, R. Thakur, and M. Winslett, "RFS: Efficient and flexible remote file access for MPI-IO," Proceedings of the International Conference on Cluster Computing, San Diego, September 2004. (Data migration in ROMIO)

9 Metadata Handling in HDF

Traditionally, HDF has stored metadata in the same file with data. This architecture has its advantages, but it also has several limitations. For example, the collocation of data and metadata in a single file may lead to increased numbers of seeks as applications write out data and the associated metadata. HDF users are storing more and more metadata, often describing smaller and smaller objects. This proliferation of metadata has led to scalability problems with the original implementation of metadata handling in HDF 5. The collocation of data and metadata may make it more difficult to treat metadata and data differently (e.g., collective write operations for data, coupled with non-collective reads and writes on metadata). We have determined that in the long run, it is better to allow metadata to reside in a separate file if desired, or in the same file with the data, as appropriate.

To include this flexibility in HDF, we rearchitected the virtual file driver by separating logical Virtual File Device operations from physical VFD operations. For example, the split file driver can write metadata to one file and raw data to another, but this is a logical operation. The decision on how to actually perform this write operation (e.g., raw data in parallel and metadata in serial) is made at the physical level of HDF. The resulting new facilities eliminate performance problems associated with metadata handling, and provide the basis for scalable metadata handling in HDF for some time to come. These facilities are available in the current release of HDF.

10 Arithmetic Operations During I/O, and Support for Formulas

HDF and DRA users have been requesting facilities that allow them to perform simple arithmetic operations on data during I/O. As perhaps the simplest example, suppose that weather data is stored using the Fahrenheit scale, but is needed in an application in the Centigrade format. We developed facilities that allow HDF users to have the conversion applied when the data is read and written.

As the first step, we determined an architecture for incorporating such a facility into HDF, through a user-supplied callback function that performs the arithmetic operations. We determined the best place for such a facility to reside in HDF, in the lower levels of the HDF library. This positioning raised certain other issues, such as how to pass the needed data type information down to that level of the library, which normally views the data as a byte stream. We addressed those issues and incorporated the new facilities into HDF.

Next, we built on the new facilities to develop an approach to support “formula” datasets. The idea is that a user can define a data object of type *formula*, store it in an HDF file, retrieve it when needed, and apply the formula to HDF data values to create new data values on the fly. These facilities are currently under development.

11 Synergistic Activities with DoE

Our portion of the Pmodels project included funds for approximately one HDF developer and one PhD student, along with partial support for their supervisors. The HDF Group has had tight ties to the DoE Laboratories for many years, as HDF is widely used inside DoE; we do not elaborate on those ties here. In addition, the PhD students supported by this work have developed close ties with the Department of Energy.

- Xiaosong Ma, the student who developed active buffering and data-format-independent buffer management, has had a 50% position at ORNL since graduation. (She also has a 50% appointment in the Department of Computer Science at North Carolina State University.) Xiaosong received a DoE Early Career Award in 2006.
- Jonghyun Lee, who developed facilities for data migration and incorporated them into ROMIO, was a postdoc at Argonne National Laboratory before assuming his current position at Oracle.
- Rishi Sinha, the student who has been working on bitmap indexing, visited Arie Shoshani’s group at Lawrence Berkeley Laboratory for an extended period. During this time he developed adaptive bitmap indexing, in collaboration with LBL researchers.