Parallel SDDS: A Scientific High-Performance I/O Interface *

Hairong Shang†, Yusong Wang, Robert Soliday, Michael Borland, ANL, Argonne, IL 60439, USA

Abstract

Use of SDDS, the Self-Describing Data Sets file protocol and toolkit, has been a great benefit to development of several accelerator simulation codes. However, the serial nature of SDDS was found to be a bottleneck for SDDS-compliant simulation programs such as parallel elegant. A parallel version of SDDS would be expected to yield significant dividends for runs involving large numbers of simulation particles. In this paper, we present a parallel interface for reading and writing SDDS files. This interface is derived from serial SDDS with minimal changes, but defines semantics for parallel access and is tailored for high performance. The underlying parallel IO is built on MPI-IO. The performance of parallel SDDS and parallel HDF5 are studied and compared. Our tests indicate better scalability of parallel SDDS compared to HDF5. We see significant I/O performance improvement with this parallel SDDS interface.

INTRODUCTION

SDDS[1] is a self-describing data file protocol developed at Argonne National Laboratory's Advanced Photon Source (APS). It is a standardized way to store and access data, and is the basis of a toolkit[2] of interoperable accelerator physics programs. Over the years, several SDDS-compliant accelerator programs (e.g, `clinchor` [3], `elegant` [4], and `shower`[5]) have been developed at the APS. Also, many existing accelerator design tools for which the source code is available have been converted to read and write SDDS files. This allows physicists to readily use several codes in combination, with greater speed, flexibility, and accuracy than otherwise possible. In addition to requiring accelerator codes to read and write SDDS files, we created a suite of generic data processing and display tools that work with SDDS files. In effect, we created a common pre- and postprocssing toolkit that is used by our codes and codes we have modified. This set of approximately 80 generic programs is referred to as the SDDS Toolkit[2].

A major advantage of using SDDS files is that data from one code can more readily be used by another. The self-describing nature of the files makes this robust, meaning that one code can be upgraded without requiring a change of the other code. In addition, with SDDS it is straightforward to process and display data from several codes together, which blurs the line between the separate codes and allows them to be used as a unit [7]. The SDDS toolkit also provides the ability to make transformations of data, which is useful when codes have different conventions (e.g. for phase-space quantities). Finally, using SDDS means that adding capabilities to a simulation code is faster and easier. The new data is simply placed in SDDS files where it can be accessed with the existing suite of tools[2].

In addition to the SDDS Toolkit, users can import SDDS data directly into programming environments like C/C++,, FORTRAN, IDL, Java, MATLAB, and Tcl/Tk, using libraries created and supported by APS. These libraries, like the rest of the SDDS software and our simulation codes, are covered by an Open Source license and available for download from our web site. The codes discussed are all available for UNIX environments, including LINUX, Solaris, and MAC OS-X, and also (usually) for Microsoft Windows. The program `elegant` [4] was the first of the SDDS-compliant accelerator codes, and it is widely used for accelerator design and simulation. It is at the center of the SDDS-compliant accelerator simulation codes. The computing power of `elegant` has been enhanced significantly because of recent parallelizations and optimizations [6]. However, the SDDS tools with sequential execution are a bottleneck for both memory and I/O operations. Therefore, parallel SDDS is required for large simulations, as well as analysis and visualizations of the resulting large data sets. This paper introduces the design, implementation, and performance study on parallel SDDS. Since HDF5[10] is another popular scientific data format, the performance of parallel HDF5 is also studied on Jazz for comparison. Although HDF5 already supports parallel I/O, it is not necessarily beneficial to switch from SDDS to HDF5, given the large number of programs and applications that already use SDDS. Only if HDF5 offers a significant performance advantage over parallel SDDS would such a conversion be considered.

SDDS File Format And Data Storage

An SDDS file is referred to as a "data set". Each data set consists of an ASCII header describing the data that is stored in the file, followed by zero or more "data pages". The data may be in ASCII or unformatted (i.e., "binary"). Each data page is an instance of the structure defined by the header. That is, while the specific data may vary from page to page, the structure of the data may not. Three types of entities may be present in each page: parameters, arrays, and columns. Each of these may contain data of a single data type, with the choices being long and short integer,

* Work supported by the U.S. Department of Energy, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.

† shang@aps.anl.gov

single and double precision floating point, single character, and character string. The names, units, data types, and so forth of these entities are defined in the header. Parameters are scalar entities. That is, each parameter defined in the header has a single value for each page. Arrays are multidimensional entities with potentially varying numbers of elements. While there is no restriction on the number of dimensions an array may contain, this quantity is fixed throughout the file for each array. However, the size of the array may vary from page to page. All columns in a data set are organized into a single table, called the "tabular data section." Thus, all columns must contain the same number of entries, that number being the number of rows in the table. There is no restriction on how many rows the tabular data may contain, nor on the mixing of data types in the tabular data. The tabular data is stored in the file by row major order, which is partly a legacy of SDDS's origins in the APS control system, where it is used to collect time-series data.

PARALLEL SDDS IMPLEMENTATION

Parallel SDDS is built on top of MPI-IO, and derived from serial SDDS with minimal changes. In parallel SDDS, a file is opened, operated on, and closed by the participating processors in a communication group defined by the user interface. Other memory access functions are retained from serial SDDS. Collective IO has been found to be much more effective than independent IO for non-contiguous storage[10], both independent and collective MPI-IO is implemented in parallel SDDS and the performance was studied too. In order to study the performance of collective IO, column majored parallel SDDS was also implemented.

In parallel SDDS, each processor holds the SDDS header data and the column data for only part of the rows, the total rows being the sum of the row numbers of all processors.

Similar to serial SDDS, parallel SDDS reads or writes a file page by page. For parallel SDDS page reading, first read the header using the serial SDDS functions and then close the file. Depends on input request, either all processors read the header or the master processor reads the header and then broadcast to other processors, this reduces the file IO load. Next, using MPI file open to open the file and then read the page title information which includes the parameters (if any), arrays (if any), and the total number or rows (total_rows) in the current page. Either all processors read the title information or the master processor reads it and then broadcast to other processors if requested. Finally, each processor reads total_rows/n_processors + left_row, the left_row is 1 if the processor ID is less than or equal to total_rows % n_processors, otherwise, it is 0.

For parallel SDDS page writing, all processors hold the layout information that is defined by the existing serial SDDS functions, and part of the tabular data partitioned by row. The file is opened for write with MPI IO. Only the master processor writes the ASCII layout, parameters (if any), arrays (if any), and the number of total rows, and then its own part of tabular data into the file. Other processors write their own part of the tabular data into the file at the same time.

PARALLEL SDDS AND PARALLEL HDF5 PERFORMANCE COMPARISON

Since the structure of current SDDS is in row majored order, collective IO would not improve the performance of row-majored parallel SDDS. Initially, independent parallel SDDS was implemented and compared with parallel HDF5. And to be simple, all processors read the header, number of rows, parameters and arrays in each page. Parallel SDDS was compiled with MPICH1 on ANL Jazz and the performance was studied with PVFS version1 file system. There are 8 PVFS parallel file system on Jazz, and the PVFS file system on Jazz is running over 10/100 ethernet. The theoretical peak I/O rate is 10 MB/sec per node on Jazz.

Since HDF5[10] is another popular scientific data format, a parallel HDF5 write/read code (ph5example.c) which comes along with the parallel HDF5 package was compiled with the same compiler used by parallel SDDS, and the performance of parallel HDF5 is studied on Jazz for comparison.

Reading Performance

Two HDF5 data files were generated in row majored order by ph5example, whose size is 1.2GB (1245710336B) and 600MB (622856192B) respectively. Each file has one two-dimensional dataset, and the dataset dimension sizes are 811008x384 for the 1.2GB file, and 811008x384 for the 600MB file, respectively. We choose row majored order because currently SDDS files are stored in row majored order. The dimensions are chosen by the requirement of ph5example that all dimensions must be a multiple of the number of processors, and 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64 processors are used for performance study. However, SDDS does not have any limitations on the dimension sizes. The two HDF5 files were converted into two SDDS files using our hdf2sdds toolkit program. The SDDS file sizes are 1245722085B, and 622861029B respectively. The sizes of the two SDDS files are bigger than those of HDF5, because SDDS header is written in ascii, and there are many columns in both files. The more columns there are, the larger the SDDS header will be. If we instead convert a 150000000x8 float (with actual data size of 1.2GB) HDF5 data set into an 8-column SDDS file with 150000000 rows, the SDDS file size is 1200000632B with a overhead of 632 bytes. The corresponding HDF5 data file size is 1200002048B with 2048 bytes overhead. In this case, SDDS has less overhead than HDF5. The read performance of both parallel SDDS and parallel HDF5 was studied with the PVFS version 1 file system on Jazz. The results of reading two files are shown in Figures 1.

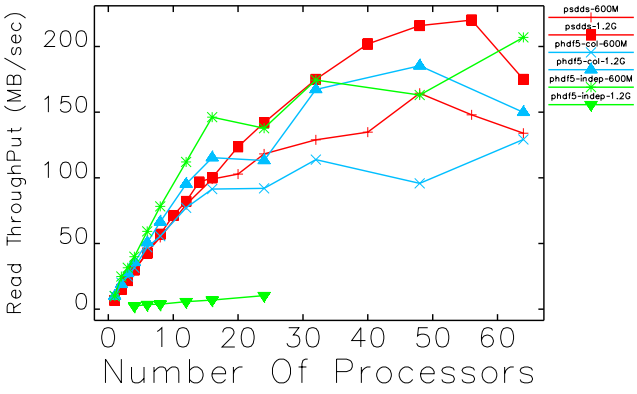

Figure 1: psdds and phdf I/O performance of reading 600MB and 1.2GB files on Jazz.

Note that the number of processors used in HDF studies are 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 56, and 64, while for parallel SDDS we chose finer spacing of the number of processors. This is because parallel HDF5 requires the dimension sizes be multiples of the number of processors, while parallel SDDS has no such limitations. Figure 1 shows that independent HDF5 I/O has better performance than collective HDF5 for reading the 600MB file. In addition, independent HDF5 has better performance than parallel SDDS when the number of processors is less than 32, however, the speed of HDF5 starts drop after 32, and its performance is similar to parallel SDDS after that. The speed of parallel SDDS continues increasing until the number of processors reaches 48 and then starts to drop. It indicates that parallel SDDS has better scalability than collective HDF5. Apparently, collective HDF5 is not a good choice for reading such a 600MB row majored file.

However, the performance of independent HDF5 in reading a 1.2G file is so poor that our performance study could not be completed with available sources. It is much worse than collective HDF5 and parallel SDDS. The performance of collective HDF5 is slightly better than parallel SDDS when the number of processors is less than 20. However, the performance of SDDS is consistently better than collective HDF5 when the number of processors is greater than 20.

Data access performance is affected by many factors, including caching, network bandwidth, and latency. Jazz has two kinds of nodes, large memory nodes which have 2.4GB memory, and smaller memory node which have 1.2GB memory. The network bandwidth is 10 MB/s. The bandwidth per processor achieved by parallel HDF5 is close to 10 MB/s with a small number of processors. However, it drops quickly to 3 MB/s as number of processors increases. The bandwidth of parallel SDDS is about 6 MB/s from 1 processors to 56 processors, and drops at 64 processors. The relatively low efficiency of SDDS at low number of processors compared to parallel HDF5 may have two reasons: First, reading SDDS data requires at least two times as much memory as the data size; this results from the way SDDS encapsulates the data. Therefore the nodes may not have enough memory to hold the data and swap space may be needed when the number of processors is small. Second, all processors read the SDDS layout at the same time. Therefore, the time spent in layout reading increases as number of processors increases, which reduces the speed when the file header is big (as in our test files) and the number of processors is large. For example, the time to read the 1.2GB file header with one processor is 0.01 seconds, but increases to 2 seconds with 64 processors, while the data access time is only 3 seconds. The layout reading can be improved in the future.

Still, the results indicate that parallel SDDS has better scalability than parallel HDF, and has better performance with large files.

Writing Performance

The writing performance of parallel HDF5 and sdds was studied for writing a 811008x192 and 811008x384 two dimensional dataset into HDF5 files or SDDS files. Both collective and independent HDF5 writing were tested. The performance of parallel SDDS writing was studied by reading a previously generated SDDS file of 811008x192 data or 11008x384 data into an SDDS dataset, copying it into a new dataset in memory, and then writing the new dataset into an SDDS file. This doubles the memory size for storing two SDDS datasets in the memory, so that memory requirements are more than four times of size of the data file. The purpose of using copying in testing parallel SDDS writing is to verify that the write operation produces a file that is identical to the original (which was the case in all tests). The performance of parallel SDDS writing may be improved when writing data generated internally into an SDDS file.

Since only one processor writes the layout, the time spent in writing layout does not increase as the number of processors increases. However, the layout writting can be improved by buffered IO, since right now each definition uses a separate write operation and the I/O times are the sum of number of parameters, arrays and columns, plus the time required to write other (generally small) parts of the SDDS header. Buffering could reduce this by a significant factor. The results of writing files are shown in Figure 2.

Different from reading, Figures 2 shows that independent HDF has better performance for writing row-majored ordered HDF5 file, and that the performance of collective HDF5 writing is worse than both independent HDF and parallel SDDS. Similar to reading, independent HDF5 performs better than parallel SDDS with a small number of processors, but as the number of processors increases, parallel SDDS performs better than independent HDF5 for writing both 600MB and 1.2GB file. The results again indicate that parallel SDDS has better scalability than HDF5, and better performance with large files.

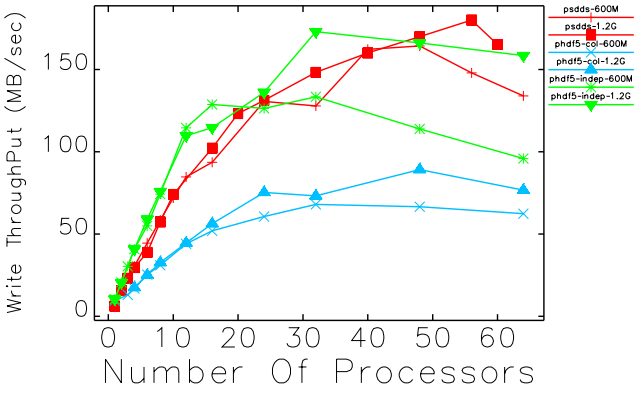

Figure 2: psdds and phdf I/O performance of writing 600MB and 1.2GB file on Jazz

We made further improvements in parallel SDDS which include 1) changing the header reading strategy so that only one processor reads the layout information, parameters and total number of rows, and then broadcasts this information. 2) using buffered IO for writing the layout, parameters, arrays, and the number of rows and for reading parameters, arrays, and the total number of rows 3) parallel reading and writing of SDDS in column majored order. Since the collective IO seems to have better performance on GPFS file system, we also implemented collective parallel SDDS. The performance was studied on Intrepid (IBM Blue Gene/P) GPFS file system[11] with reading/writing 2.4GB file. But we did not have enough time to study the performance of HDF5 on intrepid, so there is no comparison here. The results are:

- As expected, collective IO does not benefit row majored SDDS data. But it does benefit the column majored SDDS data especially in writing. The writing performance of column majored SDDS data is 1GB/sec with 350 processors, this is close to the thearetical throughput 1GB/sec with 320 processors since the bandwidth of one IO node which has 32 processors is 100MB/s.
- Similar to the Jazz PVFS system, independent IO shows good performance on GPFS in both reading and writing row majored SDDS data. And the throughput reaches the measurement of blue gene GPFS system where the reading throughput is 240MB/sec and the writing throughput is 250MB/sec[12].

CONCLUSION

In this work, we implemented a parallel SDDS interface with independent IO and completed a performance study of parallel SDDS and parallel HDF5 on Jazz with PVFS version 1 file system based MPICH1 MPI-IO. Parallel SDDS turned to have better scalability than HDF5 on PVFS file system and better performance with large files. We also implemented parallel SDDS with independent IO and collective IO for row majored and column majored SDDS data, and studied the performance on intrepid (blue gene P) GPFS file system, and the results show that the collective writing of column majored SDDS data reaches the theoretical throughput of IO nodes. Indepent parallel SDDS, which is currently being used in parallel applications such as Pelegant[13], shows good performance for both reading and writing row ordered SDDS data.

ACKNOWLEDGMENTS

We thank Robert Latham and Sam Lang from MCS ANL for providing helpful information on Jazz MPICH1 compiler, and instructions on how to run jobs with PVFS on jazz. Also we appreciate their useful information about Jazz hardware and valuable suggestions. We thank for the 2007 INCITE workshop organized by IBM and ANL ALCF division.

REFERENCES

[1] M. Borland, “A Self-Describing File Protocol for Simulation Integration and Shared Postprocessors,” 1995 PAC, Dallas, Texas, 2184(1996)

[2] M. Borland, L. Emery, H. Shang, and B. Soliday, “SDDS-Based Software Tools for Accelerator Design”, 2003 PAC, Portland, Oregon.

[3] L. Emery, “Required Cavity HOM deQing Calculated from Probability Estimates of Coupled Bunch Instabilities in the APS Ring,” Proceedings of PAC93, 3360-3362, www.jacow.org.

[4] M. Borland, “`elegant`: A Flexible SDDS-Compliant Code for Accelerator Simulation,” Advanced Photon Source LS-287, September 2000.

[5] L. Emery, “Beam Simulation and Radiation Dose Calculation at the Advanced Photon Source with `shower`, an Interface Program to the EGS4 Code System,” Proceedings of PAC96, 2309-2311, www.jacow.org.

[6] Y. Wang, M.Borland, Proceedings of PAC07, Albuquerque, New Mexico, USA June 25-29, 2007.

[7] M. Borland, Y. C. Chae, P. Emma, J. W. Lewellen, V. Bharadwaj, W. M. Fawley, P. Krejcik, C. Limborg, S. V. Milton, H.-D. Nuhn, R. Soliday, M. Woodley, “Start-to-end simulation of self-amlified spontaneous emission free electron lasers from the gun through the undulator,” NIM A 483 (2002) 268-272.

[8] W. Gropp, E. Lusk, and R. Thakur, “Using MPI-2: Advanced Features of the Mesaage-Passing Interface”, MIT press, Cambridge, MA, 199.

[9] http://www.lcrc.anl.gov/jazz/Documentation/index.php

[10] C. Chilan, M. Yang, A. Cheng, L. Arber, “Parallel I/O Performance Study with HDF5, A Scientific Data Package”, www.spscicomp.org/ScicomP12/Presentations/User/Yang.pdf

[11] https://wiki.alcf.anl.gov/index.php/File_Systems/

[12] http://www.redbooks.ibm.com/abstracts/redp4168.html?Open

[13] Y. Wang, *et al.*, these proceedings.