PARALLEL SDDS: A SCIENTIFIC HIGH-PERFORMANCE I/O INTERFACE*

Hairong Shang†, Yusong Wang, Robert Soliday, Michael Borland, Louis Emery,
Argonne National Laboratory, 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 I/O is built on MPI-I/O. 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 in the other code. 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 are available for download from our web site. The codes discussed are all available for UNIX environments, including LINUX, Solaris, and MAC OS-X, and (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, and is at the center of the SDDS-compliant accelerator simulation codes. The computing power of `elegant` has been enhanced significantly through 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 for analysis and visualizations of the resulting large data sets. This paper introduces the design, implementation, and performance study on parallel SDDS. Since HDF5 [7] is another popular scientific data format, the performance of parallel HDF5 is also studied on Jazz [8] 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, single-/double-precision floating point, single character, and character string. The names, units, data types, and other descriptions 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. Ar-

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

† shang@aps.anl.gov

rays 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.

Obviously a column-major ordered data file would be read and written faster since the data is stored as column-major order in the memory. We will soon release in [9] a column-ordered serial SDDS library, which has much better performance over row-ordered serial SDDS library. For parallel I/O, the relative advantage of column-major ordering in data files is not a given since the MPI-I/O can be executed in two modes: independent and collective [10]. In collective mode, all processors pause until they are ready to execute the I/O together. We expect that independent MPI I/O benefits the row-major ordered SDDS files, while collective MPI I/O benefits the column-major ordered SDDS files.

We built parallel SDDS libraries in the four mode combinations of independent I/O or collective I/O and row-major ordered files or column-major ordered files for study and comparison with other implementations of parallel I/O, say, that of HDF5 [7]. In this reference collective I/O has been found to be much more effective than independent I/O for non-contiguous storage, though the authors didn't specify whether HDF files were column- or row-major ordered.

PARALLEL SDDS IMPLEMENTATION

Parallel SDDS is built on top of MPI-I/O in either independent I/O or collective I/O modes, 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.

In parallel SDDS, each processor holds the SDDS header data and the column data for only part of the rows, the total number of 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, we first read the header using the serial SDDS functions and then close the file. Depending on the input request, either all processors read the header or the master processor reads the header and then broadcasts it to the other processors, which reduces the file I/O load. Next, we use MPI-I/O to open the file and to read the page title information, which are the parameters (if any), arrays (if any), and the total number of rows (n_t) in the current page. Again, either all processors (n_p in number) read the title information or the master processor reads it and then broadcasts it to other processors if requested. Finally, each processor reads $n_\mathrm{t}/n_\mathrm{p} + r$, where r is 1 if the processor ID is less than or equal to $n_\mathrm{t} \bmod n_\mathrm{p}$, otherwise, 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-I/O. 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.

PERFORMANCE COMPARISON

In this section we look primarily at row-major order SDDS library performance compared with row-major order HDF5, thus for clarity the term row-major order for HDF and SDDS is dropped.

In the SDDS test code 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 version 1 file system. There are 8 PVFS parallel file systems on Jazz running over 10/100 Ethernet. The theoretical peak I/O rate is 10 MB/sec per node.

In order to fairly compare parallel SDDS with HDF5, the parallel HDF5 write/read code (ph5example.c), which comes along with the parallel HDF5 package, was compiled with the same compiler used for parallel SDDS.

Reading Performance

Two HDF5 row-major-ordered data files were generated using ph5example, with sizes of 1.2GB (1245710336B) and 600MB (622856192B). Each file has one two-dimensional dataset, with dimension 811008x384 for the 1.2GB file and 811008x192 for the 600MB file. The dimensions are chosen by the requirement of ph5example that all dimensions must be a multiple of the number of processors. Here, 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 were 1245722085B and 622861029B respectively, which are slightly bigger than the HDF5 files because the SDDS header is written in ASCII, and there are many columns in both files (making the header large). But in actual applications such as Pelegant [6], the SDDS files have only 8 columns, which produces less overhead than a HDF5 file would. 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 the two files are shown in Figure 1.

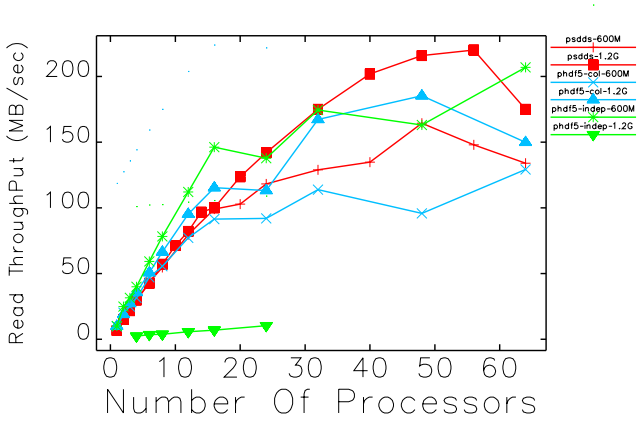

Figure 1: Parallel SDDS (psdds) and parallel HDF5 (phdf) I/O performance of reading 600MB and 1.2GB files on Jazz.

The figure includes a comparison of collective and independent I/O for HDF5. We didn't have the collective-I/O SDDS library run available for comparison. 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 independent-I/O SDDS continues increasing until the number of processors reaches 48 and then starts to drop. This may indicates that independent-I/O SDDS has better scalability than collective-I/O HDF5. Apparently, collective-I/O HDF5 is not a good choice for reading such a 600MB row-major ordered file.

However, the performance of independent-I/O 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-I/O HDF5, which is consistent with the results of parallel HDF5 [7]. The performance of collective-I/O HDF5 is slightly better than independent-I/O SDDS when the number of processors is less than 20. However, the performance of this SDDS library is consistently better than collective-I/O 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 collective-I/O HDF5 is close to 10 MB/s with a small number of processors. However, it drops quickly to 3 MB/s as the 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 a low number of processors compared to parallel HDF5 may have two causes: First, reading SDDS data requires at least two times as much memory as the data size because of 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 using serial code. Therefore, the time spent in layout reading increases as the 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 could be improved in the future.

Still, the results indicate that independent-I/O SDDS has better scalability than HDF of either I/O mode, and has better performance with large files. This may be due to the relatively simpler structure of SDDS data compared to HDF5.

Writing Performance

The writing performance of parallel HDF5 and SDDS was studied when writing 811008x192 and 811008x384 two-dimensional datasets into HDF5 files or SDDS files. Both collective-I/O and independent-I/O HDF5 writing were tested. Again results for collective-I/O SDDS was not available. 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 memory, so that memory requirements are more than four times the size of the data file. The purpose of 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 that is generated internally.

The results of writing files are shown in Figure 2. Unlike what we found for reading, independent-I/O HDF has better performance that collective-I/O HDF for writing the row-major ordered HDF5 file. Independent-I/O SDDS is also better than collective-I/O HDF. Similar to reading, independent-I/O HDF5 performs better than independent-I/O SDDS with a small number of processors, but as the number of processors increases, independent-I/O SDDS starts to perform better than independent-I/O HDF5 for writing both 600MB and 1.2GB files. The results again indicate that parallel SDDS has better scalability than HDF5 and better performance with large files in our tests.

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 writing can be improved by buffered I/O, since right now each definition uses a separate write operation, and the I/O times are the

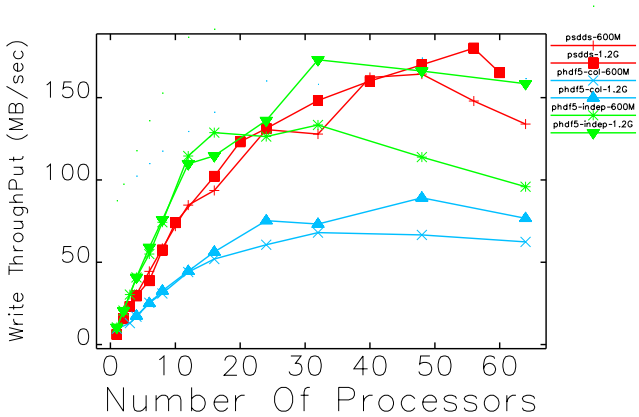

Figure 2: Parallel SDDS and parallel HDF5 I/O performance of writing 600MB and 1.2GB files on Jazz.

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.

IMPROVEMENTS

We made further improvements in parallel SDDS that 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 I/O for writing the layout, parameters, arrays, and the number of rows, and for reading parameters, arrays, and the total number of rows; and 3) parallel reading and writing of SDDS in column-major order.

Since the collective I/O seems to have better performance on the GPFS file system, we also implemented collective-I/O row-major SDDS. The performance was studied on the Intrepid (IBM Blue Gene/P) GPFS file system [11] with reading/writing a 2.4GB file. We have no performance of HDF5 on Intrepid due to lack of time. The results are as follows.

As expected, collective I/O does not benefit row-major SDDS data. But it does benefit the column-major SDDS data, especially in writing. The writing performance of column-major SDDS data is 1GB/s with 350 processors, which is close to the theoretical throughput (1GB/s for 320 processors).

Similar to the Jazz PVFS system, independent I/O row-major SDDS shows good performance on GPFS in both reading and writing. The maximum reading throughput is 600MB/s and the writing throughput is 370MB/s.

CONCLUSION

In this work, we implemented a parallel SDDS interface with independent I/O and completed a performance study of parallel SDDS and parallel HDF5 on Jazz with PVFS version 1 file system based MPICH1 MPI-I/O. Parallel SDDS (for row-major ordered files) was found to have better scalability than HDF5 on a PVFS file system and better performance with large files. We also implemented parallel SDDS with independent I/O and collective I/O for row-major and column-major SDDS data, and studied the performance on the Intrepid (Blue Gene P) GPFS file system. The results show that collective writing of column-major ordered SDDS data reaches the theoretical throughput of the I/O nodes. Independent-I/O SDDS, which is currently being used in parallel applications such as Pelegant [12], shows good performance for both reading and writing row-major-ordered SDDS data.

ACKNOWLEDGMENTS

The ComPASS project is supported under the SciDAC program by the U.S. Department of Energy Office of High Energy Physics, Office of Nuclear Physics, Office of Basic Energy Sciences, and Office of Advanced Scientific Computing Research.

REFERENCES

[1] M. Borland, "A Self-Describing File Protocol for Simulation Integration and Shared Postprocessors," Proc. of PAC95, Dallas, Texas, 2184 (1996); www.jacow.org.

[2] M. Borland et al., "SDDS-Based Software Tools for Accelerator Design", Proc. of PAC03, Portland, Oregon, 3461 (2003); www.jacow.org.

[3] L. Emery, "Required Cavity HOM deQing Calculated from Probability Estimates of Coupled Bunch Instabilities in the APS Ring," Proc. of PAC93, Dallas, Texas, 3360 (1993), www.jacow.org.

[4] M. Borland, "`elegant`: A Flexible SDDS-Compliant Code for Accelerator Simulation," Advanced Photon Source Note 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," Proc. of PAC96, Vancouver BC, 2309 (1996); www.jacow.org.

[6] Y. Wang, M.Borland, Proc. of PAC07, Albuquerque, New Mexico, 3444 (2007); www.jacow.org.

[7] C. Chilan et al., "Parallel I/O Performance Study with HDF5, A Scientific Data Package," www.spscicomp.org/-ScicomP12/Presentations/User/Yang.pdf

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

[9] http://www.aps.anl.gov/Accelerator_Systems_Division/Operations_Analysis/oagSoftware.shtml

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

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

[12] Y. Wang et al., these proceedings.