ARMY RESEARCH LABORATORY

The Battlefield Environment Division Modeling Framework (BMF) Part II: Serial and Parallel Output Enhancements

by Benjamin MacCall and Yansen Wang

ARL-TN-0646 November 2014

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Adelphi, MD 20783-1138

ARL-TN-0646 **November 2014**

The Battlefield Environment Division Modeling Framework (BMF) Part II: Serial and Parallel Output Enhancements

Benjamin MacCall and Yansen Wang
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2014		10/2013–06/2014

4. TITLE AND SUBTITLE
The Battlefield Environment Division Modeling Framework (BMF) Part II: Serial and Parallel Output Enhancements

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Benjamin MacCall and Yansen Wang

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CIE
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0646

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The treatment of input/output (IO) is critically important in computational fluid dynamics (CFD) for scalable high-performance computing (HPC) and overall data longevity. The Battlefield Environment Division Modeling Framework (BMF) v0.90 was developed for the purpose of reducing source code complexity and development time by reducing repetitive, error prone or tedious operations in source code through the use of object-oriented program (OOP) design. Here we extend BMF to include IO functionality for serial and distributed compute configurations. The Atmospheric Boundary Layer Environment (ABLE) model has been built using BMF, and ABLE now uses the IO enhancements to BMF to enable serial and parallel output, and an output buffering mechanism using dedicated output processes. Using the parallel, buffered output features, ABLE performed lid-driven cavity flow simulations and shear instability simulations saving approximately 1 GB of model state and analysis data every output time step. There was no appreciable delay when comparing these output time steps to time steps with no output.

15. SUBJECT TERMS
parallel output; high-performance computing; HDF5; NetCDF

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Benjamin MacCall
Unclassified	Unclassified	Unclassified	UU	18	19b. TELEPHONE NUMBER (*Include area code*) 301-394-1463

Contents

List of Tables

Acknowledgments

The High-Performance Computing Modernization Office (HPCMO) and the Defense Shared Resource Centers (DSRC) at the US Army Research Laboratory (ARL) and Naval Research Laboratory (NRL) supported this project with computational resources.

INTENTIONALLY LEFT BLANK.

1. Introduction

The treatment of input/output (IO) is critically important in computational fluid dynamics (CFD) for high-performance computing (HPC), inter-application communication, and overall data longevity. In HPC environments, extreme amounts of data can be generated depending on the phenomena being simulated and analysis needs. Inter-application communication is most easily accomplished using intermediate data files, and standardized IO file formats reduce the special effort required to ensure both applications are properly exchanging data. The shift towards self-describing binary data storage formats (e.g., Hierarchal Data Format v5 [HDF5], Network Common Data Format [NetCDF]) was an important transition for data longevity, and such output libraries often include features, such as on-the-fly compression, to ensure efficient use of hardware and network resources. Incorporating libraries, such as NetCDF or HDF5, into model source code—especially in HPC environments—requires significant time investment; the flexibility afforded by the output libraries creates opportunities for trivial errors that may lead to unusable data or impact performance.

The Battlefield Environment Division Modeling Framework (BMF) v0.90 was developed for the purpose of reducing source code complexity and development time by reducing repetitive, error prone, or tedious operations in source code by using object-oriented program (OOP) design. Here we extend BMF to include IO functionality for serial and distributed compute configurations. The Atmospheric Boundary Layer Environment (ABLE) model (MacCall et al., 2014) has been built using BMF; ABLE now uses the IO enhancements to BMF to enable serial and parallel output, and an output buffering mechanism using dedicated output processes. One example application is the inclusion of restart functionality, where periodically the model state is saved in order to restart integration when dividing large jobs into smaller pieces, or in the case of an interruption or unanticipated need for continued model integration.

Currently, BMF is implemented in Fortran 2008. Using the new output functionality simply requires a '`use BMF`' statement at the beginning of the module or procedure. The actual routines defined in the output libraries (e.g., HDF5 v1.8.12) are accessed via wrapper functions that employ the Fortran-C language bridge defined in Fortran 2003.

2. Methodology

The IO implementation continues the OOP design began in MacCall et al. (2013), which used several classes to reduce the complexity of employing distributed parallel computing and overlapping computation and inter-process communication via the Message Passing Interface (MPI). These classes and the classes designed for multidimensional variables on structured grids

were used in the development of the new family of IO controller classes to enable single process (serial) or multiprocess (parallel) output, using either the HDF5 or NetCDF output formats. For parallel IO, all processes can be used for output or a subset of processes can be dedicated to output via a buffering mechanism implemented via a heterogeneous linked-list container. Having all processes take part in IO reduces the interprocess communication; however, the actual writing of data to disk becomes a significant bottleneck as the number of processes increase to encompass more than a few nodes. Furthermore, all processes must complete before calculations resume. For simulations requiring more than a few nodes worth of processes, dedicated IO nodes with their associated data buffers allow compute processes to send their data to the IO node and then continue with the calculation. As long as output is not frequent enough to fill the buffers, then the bottleneck is eliminated and IO will not significantly affect model integration time.

The IO controller classes (an abstract unifying class and subclasses for each format) have been built in a modular manner to ease addition of new file formats. The basic interface for these separate use cases is contained in the abstract *OutputController* class; thus, most interactions with an IO controller will be with a generic pointer of this type. The pointer will point to an instantiated subclass of one of the file format subclasses (e.g., *HDF5Controller*, *BufferedOutputController*). In Fortran 2008, abstract classes such as the *OutputController* class can define a minimal interface for all subclasses. Thus, determining the actual type of a generic pointer is not necessary for basic usage. For special features built into a specific library, the actual type needs to be determined (e.g., using a `select type` block in Fortran 2008).

2.1 Object Instantiation and Initialization

In the following, we demonstrate the use of abstract *OutputController* interface. Instantiation requires specifying the type of output controller object during allocation and initializing the controller for further usage. In Fortran 2008:

```
class(OutputController), pointer :: outputfile
allocate(HDF5Controller::outputfile)
call outputfile%init(<filepath>, <mpi_controller>)
```

For the init routine, `<filepath>` is a character string specifying the relative path of the output file, and the `<mpi_controller>` argument is an object of *MPICartesianController* type. For non-MPI runs, the MPI object is not used.

2.2 Opening and Closing IO Files

Before performing disk IO, a file needs to be opened, and after performing IO a file must be closed to avoid data corruption, flush disk caches, etc.

```
class(OutputController), pointer :: outputfile
call outputfile%openfile([readOnly],[now])
! perform IO operations
call outputfile%closefile()
```

Both arguments are optional and are type *logical*. The `[now]` argument is necessary when performing buffered input operations to ensure files are available for immediate read operations. The `closefile()` subroutine flushes all internal IO caches and prevents further file access operations without a new open operation. For buffered output, `closefile()` will flush buffered variables to disk removing them from the IO queue. When using buffered output, the close file operation will loop through the output buffer elements and write any data buffers specified for the file associated with that particular *OutputController* object.

There is an additional issue when using buffered output. Once a `closefile()` operation is called, the processes dedicated to output will be performing queued write operations and, thus, unavailable to receive additional commands. For optimal performance, consequently, `closefile()` calls should be done before a significant amount of non-IO work will be done by the compute processes while the output-dedicated processes are busy. When attempting to close multiple files, the first `closefile()` call would proceed normally, but the compute processes would be idle until they could send the next `closefile()` message. For this case, an external subroutine `closeMultipleFiles()` is provided. For example,

```
class(OutputController), pointer :: file1, file2
call closeMultipleFiles(file1, file2, ...)
```

This subroutine accepts any output controller class not just *BufferedOutputController* objects.

2.3 Reading and Writing Data

Once a file is opened, IO operations can proceed. The type-bound procedure `writeVariable` is overloaded to include multiple data types. The routine accepts scalar or arrays of default data types (e.g., real, integer) and also accepts the BMF data objects, such as objects of type *RealVariable* or *IntegerVariable*, or any of their subclasses.

For parallel output, especially over a Cartesian domain decomposition, the varying location and uniqueness of data to be output requires special treatment. If the data (scalar or array) is equal in size and value over all processes—for example, the current time step in the simulation—then passing in the default data type (e.g., an integer) is preferred. If the data has a process-dependent structure, such as the temperature field, which may vary over different processes, the preferred treatment is to create an object of the appropriate type (e.g., an object of class *RealVariable*), which has additional features for storing process dependent information.

Another example is a 2-dimensional array of data points that is only relevant along the lower bound of the domain, so only processes that contain the surface points will have data. Again, the preferred manner to treat such cases is to use the BMF classes of the appropriate type, shown in the below pseudo-Fortran code:

```
class(RealVariable), pointer :: myvar
allocate(myvar)
call myvar%init(name="varname", &
      extent=Coordinate(XX_val=NX, YY_val=NY, ZZ_val=1), &
      mpiController=cartesianController, & ! MPI configuration
      processDependent= .True., & ! all directions are true
      dataOnProcess= processContainsLowerBound )

...calculate variable data...

class(OutputController), pointer :: outputfile
allocate(outputfile)
call outputfile%init("filename", cartesianController)

call outputfile%openfile()
call outputfile%writeVariable(path="aGroup/", &
      variable=myvar, &
      precision=real8, &
      timeDependent=.True. )
call outputfile%closefile()
```

This code first creates a real variable object, with name "varname", that has *NX* points in the x-direction, *NY* points in the y-direction, and 1 point in the z-direction. The MPI configuration is embedded in the *cartesianController* object, and the data is specified as being process dependent in all 3 directions. We finish the initialization by specifying that only processes that contain the domain lower bound will contain data. On the other processes, the *RealVariable* object exists, but contains no data. Saving the variable data to the file, "filename," requires opening the file, executing the `writeVariable` procedure, and closing the file.

Reading data from a file follows a similar procedure. In the case of process dependent data, the *RealVariable* object needs to already be initialized as was done in the previous example, because the process dependent details stored in the *RealVariable* object are used to determine the subset of the data in the file that will be read. In this way, data written to a file using 1 MPI decomposition—say, 2 processes in each direction—can be read when using a different decomposition (say, 2 processes in the vertical direction and 4 processes in each horizontal direction). Below is pseudo-Fortran code reading the data saved to file in the previous example:

```
...same RealVariable call to init as above...
class(OutputController), pointer :: outputfile
allocate(outputfile)
call outputfile%init("filename", cartesianController)

call outputfile%openfile(now=.True.)
call outputfile%readVariable(path="aGroup/varname", &
      variable=myvar, &
      timeRecord=3 )
call outputfile%closefile()
```

The same initialization is performed for `myvar` as above. The *OutputController* object `outputfile` still needs to be initialized with the file system path. The data file is made accessible to the program by calling `openfile`. When calling the read subroutine, the full path, including the variable name, for the variable in the data file must be specified. If the variable is time-dependent, a time record can be specified. If the variable is not time-dependent, the `timeRecord` argument can be specified, but the value must be set to 1.

3. Discussion

The aforementioned code was implemented in BMF v0.90 and employed by the ABLE model v0.90 during several simulations of lid-driven cavity flow and shear instability. The nature of the integration scheme and the simulations required a large number of grid points and many small time steps when integrating the governing equations. The MPI configuration employed 512 process cores in a 3-D decomposition (8 × 8 × 8). A single node with 16 cores was dedicated to performing output, and the *OutputController* object was instantiated with a type of *BufferedOutputController*. Because of the small time steps, output was performed every 5000 time steps; approximately 500 GB of data was generated for each simulation, which took about 18 h on the US Army Research Laboratory (ARL) cluster Pershing.

With the buffered output strategy, time steps with output operations had a negligibly longer duration than time steps with no output. The compute time for data transfer to the output node for buffering was significantly smaller than the time spent integrating the governing equations.

The HDF5 libraries come with a tool to test performance characteristics on a cluster. Using this tool on the ARL cluster Pershing is detailed in the Table.

Table Example performance benchmarks on the ARL cluster Pershing using the HDF5 h5perf tool. Configuration used the parallel HDF5 IO interface with the MPI POSIX driver. The tool wrote and then read an 8 MB file with a single dataset per process. The use of multiple nodes causes a significant reduction in IO performance; however, the larger available memory for use by the output buffers may reduce the impact of IO on overall runtime.

		8 MB/file		16 MB/file	
# of Nodes	**# of Processes**	**Peak Write (MB/s)**	**Peak Read (MB/s)**	**Peak Write (MB/s)**	**Peak Read (MB/s)**
1	1	4253	9738	3249	6873
1	2	4886	11,790	1800	8869
1	4	5717	12,524	2618	12,856
1	8	5383	12,903	3366	13,368
1	16	5235	16,837	4095	19,051
2	32	3279	3560	3320	5548
4	64	3443	4168	3594	6211

Generally, using additional processes is advantageous to IO performance up to the point of using multiple nodes. Thus, on hardware with high-performance disks (e.g., RAID arrays or solid state drives), MPI-based simulations using a small number of processes (especially if only a single node is available) should generally use all processes when performing output rather than using a master process that collects the data from the others. As the number of processes used in the calculation increases, a set of dedicated output processes will allow the simulations to run more efficiently unless the data buffers on the output processes become full, which requires immediate writing of the data to disk. In this case, the recommended solution is to use multiple nodes for output, thereby increasing the total size of the output buffers. When running a simulation, if the output buffers become full, a message is written to the program log. Peak performance may require testing a few different configurations.

4. Conclusion

The aforementioned examples condense several thousand lines of source code (not counting the source code for classes like *RealVariable*) for various operations that ensure data is properly collected from multiple processes and written—in an efficient manner with basic error checking—into a standard-format, binary data file. With BMF and a relatively small number of commands, powerful, flexible output libraries, such as NetCDF and HDF5, can be implemented in (currently) Fortran source code using MPI parallelism and optionally dedicated output processes. Using BMF for output does not require all the features of the framework, especially for serial output. Additionally, the object-oriented design provides design strategies for adding new file formats without requiring a significant modification to existing model code.

Managing output and the bottleneck associated with slow disk performance when conducting IO operations is critical for efficient, scalable computing. Using the parallel, buffered output

features, the lid-driven cavity flow simulations and shear instability simulations were able to save the model state and analysis data with no appreciable delay when compared to time steps where no output occurred. The compute clusters (at the ARL and NRL DSRC) employed 16-core nodes with 29 GB of usable memory. For the simulations with 200^3 and 304^3 data points per 3-D variable, one node dedicated to output was enough to ensure that the processes responsible for computation spent minimal time preparing for output. Obviously, with 3-D simulations and fields being updated rapidly, overwhelming the disk controllers and buffering system is always possible and will cause model integration to slow. However, if there are intervals with a large amount of output interspersed with intervals with little to no output, overwhelming the buffers can be mitigated by increasing the number of nodes dedicated to output, which increases the size of the output buffers.

The upcoming refinement to the above output routines will entail the implementation of dimensional scales to the written quantities. For example, time dependent variables will have a time attached for each output, and spatial arrays will have coordinates associated with each point. The procedure for including physical dimensions will be implemented for BMF v1.0.

5. References

MacCall B, Huynh G, Wang Y. The Battlefield Environment Division Modeling Framework (BMF) part I: Optimizing the atmospheric boundary layer environment model for cluster computing. Adelphi (MD): US Army Research Laboratory (US); 2014. Report No.: ARL-TR-6813.

List of Symbols, Abbreviations, and Acronyms

ABLE	Atmospheric Boundary Layer Environment
ARL	US Army Research Laboratory
BMF	The Battlefield Environment Division Modeling Framework
CFD	computational fluid dynamics
DSRC	Defense Shared Resource Centers
HDF5	Hierarchal Data Format v5
HPC	high-performance computing
IO	input/output
MPI	Message Passing Interface
NetCDF	Network Common Data Format
NRL	Naval Research Laboratory
OOP	object-oriented programming

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDFS)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
2 (PDF)	US ARMY RESEARCH LAB ATTN RDRL CIE M B MACCALL ATTN RDRL CIE M Y WANG