BIL: A Block I/O Layer for Analysis and Visualization Applications

Wes Kendall

May 30, 2010

1 Introduction

When using parallel I/O, some limitations are currently present in MPI-I/O and higher level data access libraries like Parallel netCDF. One of these limitations is I/O across multiple files. Another limitation in netCDF is not being able to request disjoint regions of a file in one I/O request. The Block I/O Layer (BIL) is designed to specifically handle situations like this and uses state of the art approaches to do so. BIL operates under the general concept of performing I/O on distributed arrays (blocks). To use BIL, each process adds all the blocks they wish to read in by calling $BIL\_Add\_block\_\{raw, nc\}$. The user may request multiple blocks from a file and the blocks may span multiple files. Figure 1 shows an example of an access pattern like this where processes request blocks from files in a round robin ordering. Once all of the blocks have been added, BIL_Read is then called and an array is returned with a pointer to the data of each block. BIL handles all of the aggregation of reading in the background and uses MPI-IO for raw datasets and Parallel netCDF for netCDF datasets.

The API specification is listed in Section 2. Example code is shown in Section 3. The code is freely available at the Seelab website. Further questions and bug reports about the interface may be sent to Wes Kendall. Suggestions for future direction of the interface are also welcome. If anyone intends to use this interface for their research, please reference [1] as it uses some of the concepts in BIL.

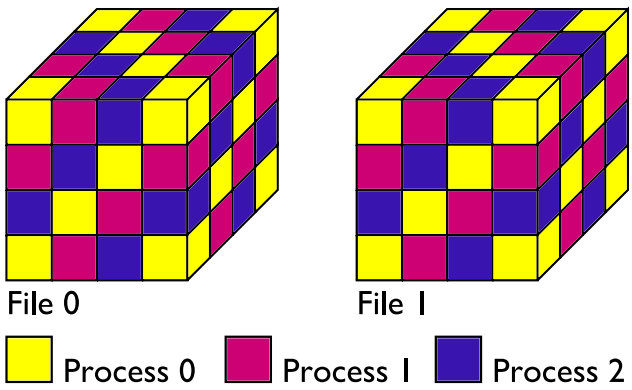

Figure 1: A sample access pattern of three processes requesting 3D blocks in a round robin ordering across two files.

2 Interface

2.1 BIL_Init

```
void
BIL_Init
  (MPI_Comm world_comm)
```

Synopsis Initializes the interface and allocates all necessary internal structures.

Parameters

world_comm
: The MPI communicator that BIL will operate in.

2.2 BIL_Finalize

```
void
BIL_Finalize
  ()
```

Synopsis Finalizes the interface and cleans up all internal structures.

2.3 BIL_Add_block_raw

```
void
BIL_Add_block_raw
  (const int num_dims,
   const int *file_dims,
   const int *block_start,
   const int *block_size,
   const char *file_name,
   const MPI_Datatype var_type)
```

Synopsis Adds a raw block to a future read request. The block is described by the beginning and sizes of each dimension. The user specifies the type of variable in the file. Currently, only elementary MPI variables or contiguous datatypes are supported for the variable type. Other MPI datatypes will results in undefined behavior.

Parameters

num_dims
: The number of dimensions of the variable.

file_dims
: The sizes of each dimension of the entire variable.

block_start
: The start of each dimension of the block to read.

block_size
: The size of each dimension of the block to read.

file_name
: The file name that the variable is located in.

var_type
: The type of variable to read in. Currently, only elementary MPI datatypes are supported along with contiguous datatypes.

2.4 BIL_Add_block_nc

```
void
BIL_Add_block_nc
  (int num_dims,
   int *block_start,
   int *block_size,
   char *file_name,
   char *var_name)
```

Synopsis Adds a netCDF block to a future read request. The block is described by the beginning and sizes of each dimension. The user specifies the name of the variable to be read in.

Parameters

num_dims
: The number of dimensions of the variable.

block_start
: The start of each dimension of the block to read.

block_size
: The size of each dimension of the block to read.

file_name
: The file name that the variable is located in.

file_name
: The name of the variable to be read.

2.5 BIL_Read

```
void **
BIL_Read
  ()
```

Synopsis Issues a read request to all the blocks that were added. The blocks may span multiple files and multiple variables. If processes only requested at most one block each, then BIL performs a simple one-phase I/O request. If processes requested more than one block each, then two phase I/O is used. The result is stored in an array of pointers to the blocks of data. The array is ordered in the order that the blocks were added.

Return An array of pointers to each block in the order that they were added.

2.6 BIL_Set_io_hints

```
void
BIL_Set_io_hints
  (const MPI_Info io_hints)
```

Synopsis Sets the I/O hints to be used when BIL issues an I/O request.

Parameters

io_hints
: The MPI_Info struct containing the I/O hints.

3 Example Code

3.1 test_bil_raw.c

Synopsis This program uses BIL to read in random blocks of raw files. This is a test program that is included with the BIL release. The files that BIL reads in are randomly generated before the program starts.

```
#include "bil.h"

// number of blocks each process reads in per variable
#define NUM_BLOCKS 2

int main(int argc, char **argv)
{
  if (argc < 3 || argc > 4)
  {
    fprintf(stderr, "usage %s dim_size num_files <file_dir>\n", argv[0]);
    exit(1);
  }
  const int dim_size = atoi(argv[1]);
  const int num_files = atoi(argv[2]);
  const char *file_dir = (argc == 4) ? argv[3] : "./";

  // do some checking of arguments
  assert(strlen(file_dir) < 512);
  assert(dim_size <= 512 && dim_size > 0);
  assert(num_files <= 16 && num_files > 0);

  MPI_Init(NULL, NULL);
  // Initialize BIL for MPI_COMM_WORLD
  BIL_Init(MPI_COMM_WORLD);

  // normally we would create an MPI_Info structure that contains
  // hints to pass to BIL. This call is not necessary but is only
  // here as an example
  BIL_Set_io_hints(MPI_INFO_NULL);

  int rank;
  int h, i, j, k, l;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  int t = time(NULL);
  srand(rank + t);

  int data_dims[3] = {dim_size, dim_size, dim_size};
  char file_names[num_files][1024];
  for (i = 0; i < num_files; i++)
  {
    memset(file_names[i], 0, 1024);
    sprintf(file_names[i], "%s/%d_%d_%d.%d.raw", file_dir,
```

```
      dim_size, dim_size, dim_size, i);
  }

  int block_starts[NUM_BLOCKS * num_files][3];
  int block_sizes[NUM_BLOCKS * num_files][3];
  // randomly generate block bounds
  int block = 0;
  for (h = 0; h < num_files; h++)
  {
    for (i = 0; i < NUM_BLOCKS; i++, block++)
    {
      block_starts[block][0] = ((float)rand() / RAND_MAX) * (dim_size - 1);
      block_starts[block][1] = ((float)rand() / RAND_MAX) * (dim_size - 1);
      block_starts[block][2] = ((float)rand() / RAND_MAX) * (dim_size - 1);

      block_sizes[block][0] =
        ((float)rand() / RAND_MAX) * (dim_size - block_starts[block][0]) + 1;
      block_sizes[block][1] =
        ((float)rand() / RAND_MAX) * (dim_size - block_starts[block][1]) + 1;
      block_sizes[block][2] =
        ((float)rand() / RAND_MAX) * (dim_size - block_starts[block][2]) + 1;
      BIL_Add_block_raw(3, data_dims, block_starts[block], block_sizes[block],
          file_names[h], MPI_INT);
    }
  }

  // read all the blocks
  int **block_data = (int **)BIL_Read();

  // the blocks are stored as a two dimensional array of pointes to the
  // data in the order it was added. now the user is able to process the
  // data however they want

  ...

  // Finalize BIL
  BIL_Finalize();
  MPI_Finalize();

  return 0;
}
```

4 Future Work / Known Limitations

In the future, HDF support will be added to BIL along with the ability to write blocks of data. More advanced functionality will be added to support hints being passed to BIL along with handling basic I/O requests without using two-phase I/O.

BIL has some known limitations and there are circumstances when using BIL will not give you good I/O performance. First, BIL does not have any special support for noncontiguous file access. BIL will simply use whatever mechanisms are implemented in the underlying MPI-IO implementation for file access of this kind. Second, BIL will aggregate the block requests into larger read requests when possible. If the global I/O request contains noncontiguous file access, BIL may read more data than is necessary. More information on this will come in future releases of this manual. If you believe this may be a problem, please email Wes Kendall for more details on this issue.

References

[1] W. Kendall, M. Glatter, J. Huang, T. Peterka, R. Latham, and R. Ross. Terascale data organization for discovering multivariate climatic trends. In *SC '09: Proceedings of the ACM/IEEE Supercomputing Conference*, 2009.