DAMSEL - A Data Model Storage Library for Exascale Science (API and Use cases)

(This work is supported by Office of Advanced Scientific Computing Research under the program of X-stack Software Research)

Saba Sehrish
CSciADS 2012
July 31, 2012
Outline

- Project Team
- Motivation
- Damsel I/O Library
- Usecases: FLASH, GCRM
- Data layout (In Progress)
Project Team

- **Northwestern University**: Alok Choudhary, Wei-keng Liao, Saba Sehrish, Sueng Woo Son
- **Argonne National Laboratory**: Rob Ross, Rob Latham, Tim Tautges
- **The HDF Group**: Quincey Koziol, Ben Clifford, Peter Cao
- **NC State University**: Nagiza Samatova, Sriram Lakshminarasimhan
1 Motivation

- Existing I/O Libraries
- Goals
Existing I/O Libraries

- Storage data models developed in the 1990s; Network Common Data Format (netCDF) and Hierarchical Data Format (HDF)
- I/O library interfaces still based on low-level vectors of variables
- Lack of support for sophisticated data models, e.g. AMR, unstructured Grids, Geodesic grid, etc
- Require too much work at application level to achieve close to peak I/O performance
Example: Lower Triangle Matrix

- **netCDF**: fixed dimensions
- **HDF5**: Potential for odd interactions between application data layout and chunk allocation
- **Lower-triangular aware storage mode and layout**
Motivation

Example: FLASH

FLASH - AMR Grid

- Red boxes are cells
- Black boxes are blocks

Morton order

Each block in AMR grid corresponds to a tree node

Existing I/O Libraries
Goals
Example: FLASH

- Parallel adaptive-mesh refinement (AMR) code; Block structured - a block is the unit of computation
- **Tree information**: FLASH uses tree data structure for storing grid blocks and relationships among blocks, including lrefine, which_child, nodetype and gid.
- **Per-block metadata**: FLASH stores the size and coordinates of each block in three different arrays: coord, bsize and bnd_box
- **Solution Data**: Physical variables i.e. located on actual grid are stored in a multi-dimensional (5D) array e.g. UNK
Goals

- Provide higher-level data model API to describe more sophisticated data models, e.g. structured AMR, geodesic grid, etc
- Enable exascale computational science applications to interact conveniently and efficiently with storage through the data model API
- Develop a data model storage library to support these data models, provide efficient storage data layouts
- Productizing Damsel and working with computational scientists to encourage adoption of this library by the scientific community
2 Damsel I/O Library
  - Data Model
  - API
Proposed Approach

- A set of data models I/O APIs relevant to computational science applications
- A data layout component that maps these data models onto storage efficiently,
- A rich metadata representation and management layer that handles both internal metadata and that generated by users and external tools,
- I/O optimizations: adaptive collective I/O, request aggregation, and virtual filing,
Data Model Components

- Describe structural/(hierarchical) and solution information through API
- To describe the structural information, i.e. Grid data Entity, Collections, Structured Blocks
- To describe the solution variable, i.e. Solution data Tags on Entities, Collections, Structured Blocks
Example: Entity and Tags

Entities: Vertex, Edge, Rectangle, Hex

Tags: Solution data at vertices, edges, centers, etc
Example: Sequence of entities and Tags

Step 1: Create a sequence of entities (QUADS)

Step 2: Define an entity by specifying vertices

start_coord[2] = {0.0, 0.0}

Step 3: Set coordinates of vertices

Step 4: Define tags with name (temp) and type (float)

Step 5: Map tags to the entities

Step 6: Iterate through all tags and write to file
Example: Lower Triangle Matrix

- **netCDF**: fixed dimensions
- **HDF5**: Potential for odd interactions between application data layout and chunk allocation
- **Lower-triangular data model and layout**

**A QUAD entity in Damsel**

**A sequence of quads**

Damsel I/O Library

Data Model

API

1. **Today's I/O Software Stack**

I/O systems on modern HPC hardware is actually a stack of components, consisting of disk and network hardware, high-level I/O libraries, and I/O middleware that link the two. A depiction of this I/O stack is shown in Figure 2. At the bottom of this stack is the storage layer, appearing as a parallel file system connected to disk and network hardware. I/O middleware, such as an MPI-IO implementation, sits on top of the parallel file system and handles communication between parallel compute nodes and I/O nodes, including management of both concurrency and locality of accessing data.

I/O middleware and lower layers are designed to maximize I/O throughput, primarily as a linear stream of bytes. Computational science codes, in contrast, understand the semantics of those bytes, as grids, fields on the grids, and metadata annotations to both. High level I/O libraries are designed to translate between the semantic and storage representations. HDF5 and PnetCDF are the two most popular options in HPC, supporting the management and organization of semantic information, as well as the mechanics of I/O storage operations. Currently, these high-level I/O libraries present a data model based on multi-dimensional arrays of typed elements, with annotations for timestamps, runtime parameters, or other provenance. In addition to multi-dimensional arrays and attributes, the libraries also define a portable, self-describing on-disk file format, making it easier to exchange data with colleagues.

However, even a fairly simple example of solution data on a structured grid can map to I/O libraries in less than ideal ways. Figure 3 illustrates the mapping from a conceptually straightforward lower triangular matrix to several storage layouts. The netCDF layout, based on fixed-dimensional arrays, results in an array that wastes just under half its allocated space. HDF5 supports multi-dimensional arrays and chunk-based allocators, which alleviate this problem somewhat. However, the application must specify the chunk size, and coordinate matching the matrix structures to those chunks. This can add development complexity, and is even more difficult for unstructured data types. A better approach would take advantage of the data model's semantic information and avoid allocating space that will not be used.

2. **Data Models and Layouts**

A data model describes how simulation data is represented and accessed. For discretization-based solutions of PDEs, the data model includes a description of the discretized domain (space and time, and sometimes other dimensions like energy), and field data computed by the simulation over those discretizations. The data model is a key part of HPC codes and strongly influences the efficiency of both computation and communication. At their core, scientific codes usually store the model as multi-dimensional arrays, since...
Damsel Program Flow

- `damsel_library lib = DMSLlib_Init();`
- Create a model
  `DMSLmodel_create(DAMSEL_TYPE_HANDLE_64);`
- Fill in the application specific model details e.g. number of entities, types of entities, etc
  1) `damsel_handle my_handle = {12, 45, 67, 89 };`
  2) `damsel_container my_container = DMSLcontainer_create_vector(model, my_handle, 4);`
  3) `damsel_collection my_coll = DMSLcoll_create(model, my_handle, my_container, DAMSEL_HANDLE_COLLECTION_TYPE_VECTOR);`
Fill in the application specific variables and solution data e.g. tags (coordinates, solution data)
1) `damsel_handle tag_handle = 10001;`
2) `DMSLtag_define(model, "temperature", DAMSEL_TYPE_FLOAT);`
3) `DMSLmodel_map_tag(data, my_coll, &tag_handle);`
- `DMSLExecute(model);`
- `DMSLLib_finalize(lib);`
3 Usecases

- Usecase I: FLASH
- Usecase II: GCRM
Use cases

Usecase I: FLASH
Usecase II: GCRM

Introduction

FLASH - AMR Grid

- Red boxes are cells
- Black boxes are blocks

Morton order

Each block in AMR grid corresponds to a tree node
Introduction

- The FLASH is a modular, parallel multi-physics simulation code capable of handling general compressible flow problems found in many astrophysical environments.
- Parallel adaptive-mesh refinement (AMR) code; Block structured - a block is the unit of computation
- **Tree information:** FLASH uses tree data structure for storing grid blocks and relationships among blocks, including lrefine, which_child, nodetype and gid.
- **Per-block metadata:** FLASH stores the size and coordinates of each block in three different arrays: coord, bsize and bnd_box
- **Solution Data:** Physical variables i.e. located on actual grid are stored in a multi-dimensional (5D) array e.g. UNK
FLASH using existing I/O Libraries

FLASH in PnetCDF

/*Step 1: Create data set*/
cmpi_create_data()

/*Step 2: Define dimension*/
status = cmpi_def_dim(ncid, "dim_tot_blocks", (MPI_Offset) (*total_blocks), &dim_tot_blocks);

/*Step 3: Define variables*/
Status = cmpi_def_var (ncid, "runtime_parameters", NC_INT, rank, dimids, &varid[id]);
status = cmpi_def_var (ncid, "lrefine", NC_INT, rank, dimids, &varid[id]);

/*Step 4: Create attributes for some variables*/
status = cmpi_put_att_int(ncid, 1, intScalarNames[i], NC_INT, 1, &intScalarValues[i]);

/*Step 5: Write structural & solution data*/
/* Write data from memory to file */
    err = cmpi_put_vara_all(fileID, varID, diskStart, diskCount, pData, memCountScalar, memType);

/*Step 6: Close the dataset/file*/
cmpi_close(fileID);
FLASH using DAMSEL data model

- **Goal**: to describe hierarchical/structural and solution information through API

- **Entity**
  - FLASH blocks as a sequence of entities

- **Collections**
  - Blocks assigned to collections to define hierarchical/structural information

- **Tags**
  - coordinates, size, bounding box
  - UNK (temperature, pressure, etc)
FLASH using DAMSEL API

**Step 1: Define sequence of block entities**
1. `damsel_handle block_id [17]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17};`
2. `damsel_container block_collection_id = DMSLcontainer_create_vector(model, block_id, 17);`
3. `Damsel_collection DMSLcoll_create();`
4. `DMSLentity_define(block_collection_id, , ,);`

**Step 2: Defining block metadata using tags**
1. `damsel_handle coord_tag_handle = 10004;`
2. `DMSLtag_define(model, &coord_tag_handle, coords_array_type, "coordinates");`
3. `DMSLmodel_map_tag(block_coords, block_collection_id, &coord_tag_handle);`

// Same procedure for bounding box, size, etc
**FLASH using DAMSEL API**

**Step 3: Define hierarchy through collections**

1. `damsel_handle temp_cont[5] = {3, 4, 5, 6, 7};`
2. `damsel_container c31 = DMSLcontainer_create_vector(model, temp_cont, 5);`
3. `damsel_handle parent_tag_handle = 10023;`
4. `DMSLtag_define(model, &parent_tag_handle, TYPE_HANDLE, "Parent_b3");`
5. `DMSLmodel_map_tag(2, c31, &parent_tag_handle);`

**Step 4: Defining Solution data using tags**

1. `damsel_handle unk_tag_handle = 10004;`
2. `DMSLtag_define(model, &unkd_tag_handle, unk_array_type, "UNK");`
3. `DMSLmodel_map_tag(unk_data, block_collection_id, &unk_tag_handle);`
FLASH using DAMSEL API

Step 5: Mapping to file handles

1. DMSLmodel_attach(model, "test-flash.h5", MPI_COMM_WORLD, NULL);
2. DMSLmodel_map_handles_inventing_file_handles(block_collection_id);
3. DMSLmodel_map_handles_inventing_file_handles(unk_tag_handle);
4. ...
5. DMSLmodel_transfer_async(model, DAMSELTRANSFER_TYPE_WRITE, &req);
6. Finalize lib instance
Introduction
**Introduction**

- **Grid data**
  - Cell corners (2/cell)
  - Cell edges (3/cell)
  - Layers and interfaces

- **Solution data at both interfaces and layers**
  - Cell centers,
  - corners, edges
GCRM using existing I/O Libraries

PNetCDF

- **Grid Data:**
  - Dimensions: Cells, edges, interfaces, etc
  - Variables: grid_center_lat(cells), grid_corner_lat(corners), cell_corners(cells, cellcorners)

- **Solution Data:**
  - float pressure(time, cells, layers)
  - float u(time, corners, layers)
  - float wind(time, edges, layers)
GCRM using DAMSEL

- A Hexagonal Prism entity to describe a cell
- An unstructured mesh to describe GCRM grid (no hierarchical information)
- Or a structured mesh to describe GCRM grid
Summary

- Motivation
- DAMSEL Data Model
- API Implementation
- Usecases: FLASH and GCRM
- Data layout work is in progress