ECP Data & Vis SDK Documentation

Release 1.1

Chuck Atkins, Kitware, Inc.

Jul 24, 2020

Introduction

1 What is it? 3

2 What's in it? 5
- 2.1 ecp-io-sdk . . . 5
- 2.2 ecp-viz-sdk . . . 5

3 Spack Packages 7
- 3.1 ecp-io-sdk . . . 7
- 3.2 ecp-viz-sdk . . . 8

4 I/O and Data Services 9
- 4.1 ADIOS2 . . . 9
- 4.2 HDF5 . . . 10
- 4.3 PNetCDF . . . 12
- 4.4 VeloC . . . 14
- 4.5 UnifyFS . . . 15
- 4.6 Mercury . . . 15
- 4.7 Darshan . . . 15

5 Visualization, Analysis, and Data Reduction 17
- 5.1 ParaView . . . 17
- 5.2 Catalyst . . . 17
- 5.3 VTK-m . . . 17
- 5.4 SZ . . . 18
- 5.5 ZFP . . . 19

6 Indices and tables 23

Funded by the Exascale Computing Project (ECP), U.S. Department of Energy

CHAPTER 1

What is it?

The Data and Vis SDK is a collection of software packages being developed under the DOE's Exascale Computing Program with a focus on I/O, compression, and visualization. The SDK provides a mechanism to bring common capabilities with respect to development practices, software quality, installation, and usability.

CHAPTER 2

What's in it?

The Data and Vis SDK is currently broken into two distinct packages containing the following products:

2.1 ecp-io-sdk

I/O and data management

HDF5 A data model, library, and file format for storing and managing data.

ADIOS A framework designed for scientific data I/O to publish and subscribe (put/get) data when and where required.

PNetCDF A parallel I/O library for NetCDF file access.

Darshan A scalable HPC I/O characterization tool.

Mercury A C library for implementing Remote Procedure Call, optimized for High-Performance Computing Systems.

UnifyFS A user-level burst buffer file system under active development. UnifyFS supports scalable and efficient aggregation of I/O bandwidth from burst buffers while having the same life cycle as a batch-submitted job.

VeloC A multi-level checkpoint/restart runtime that delivers high performance and scalability for complex heterogeneous storage hierarchies without sacrificing ease of use and flexibility.

2.2 ecp-viz-sdk

Visualization, analysis, and data reduction

ParaView A multi-platform data analysis and visualization application.

Catalyst An in-situ use case library for ParaView.

VTK-m A toolkit of scientific visualization algorithms for emerging processor architectures.

ZFP A library for compressed numerical arrays that support high throughput read and write random access.

SZ An error-bounded lossy data compressor.

CHAPTER 3

Spack Packages

Spack, a source-based package manager for supercomputing environments, is the chosen deployment mechanism for the Data and Vis SDK. For both the `ecp-io-sdk` and `ecp-viz-sdk` packages, each currently supported product is enabled through a spack variant.

3.1 ecp-io-sdk

The I/O and data services products are available through the `ecp-io-sdk` spack meta-package. As of the publishing of this guide all variants are on by default allowing all products to be installed simultaneously:

```
$ spack info ecp-io-sdk
...
Description:
    ECP I/O Services SDK
...
Variants:
...
    adios2 [on]                  True, False            Enable ADIOS2
    darshan [on]                 True, False            Enable Darshan
    hdf5 [on]                    True, False            Enable HDF5
    mercury [on]                 True, False            Enable Mercury
    pnetcdf [on]                 True, False            Enable PNetCDF
    unifyfs [on]                 True, False            Enable UnifyFS
    veloc [on]                   True, False            Enable VeloC
...

$ spack install ecp-io-sdk
```

3.2 ecp-viz-sdk

The visualization and data reduction products are available through the `ecp-viz-sdk` spack meta-package. As of the publishing of this guide all variants are off by default allowing each product to be installed one at a time:

```
$ spack info ecp-viz-sdk
...
Description:
    ECP Viz & Analysis SDK
...
Variants:
...
    catalyst [off]                True, False             Enable Catalyst
    paraview [off]                True, False             Enable ParaView
    sz [off]                      True, False             Enable SZ
    vtkm [off]                    True, False             Enable VTK-m
    zfp [off]                     True, False             Enable ZFP
...

$ spack install ecp-viz-sdk
```

CHAPTER 4

I/O and Data Services

4.1 ADIOS2

Note: for more detailed examples and documentation see the ADIOS2.

Writing simulation results:

```
#include <mpi.h>
#include <adios2.h>
...

int main(int argc, char **argv);
{
  MPI_Init(&argc, &argv);

  int size, rank;
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // Number of elements per rank
  size_t N = 10000;

  try
  {
    // Initialize output
    adios2::ADIOS adios(MPI_COMM_WORLD);
    auto io = adios.DeclareIO("sim_output");

    // Define a 1D temperature array
    auto varT = io.DefineVariable<double>("t",
      {size*N}, {rank*N}, {N}, adios2::ConstantDims);

    // Define a 2D 3xN pressure array
    auto varP = io.DefineVariable<double>("p",
```

(continues on next page)

(continued from previous page)

```
      {3, size*N}, {0, rank*N}, {3, N}, adios2::ConstantDims);

    // Open file for writing
    auto writer = io.Open("output.bp", adios2::Mode::Write);

    std::vector<double> dataT(N);
    std::vector<doublt> dataP(3*N);

    // Main simulation loop
    for(size_t step = 0; step = 100; ++step)
    {
      //
      // Step the simulation and populate the dataT and dataP arrays
      //

      // Begin the output step
      writer.BeginStep();

      // Write the arrays
      writer.Put(varT, dataT.data());
      writer.Put(varP, dataP.data());

      // Finish the output step
      writer.EndStep();
    }
    writer.Close();
  }
  catch(const std::exception &ex)
  {
      std::cerr << "Error: " << ex.what() << std::endl;
  }

  MPI_Finalize();
  return 0;
}
```

4.2 HDF5

Note: for more detailed examples and documentation see the HDF5 Support Portal.

Write out dataset using dynamic array:

```
#include "hdf5.h"
#include <stdlib.h>

#define FILE        "dyn.h5"
#define DATASETNAME "IntArray"
#define NX     5                      /* dataset dimensions */
#define NY     6
#define RANK   2

int
main (void)
{
```

(continues on next page)

(continued from previous page)

```
    hid_t       file, dataset;          /* file and dataset handles */
    hid_t       datatype, dataspace;    /* handles */
    hsize_t     dimsf[2];               /* dataset dimensions */
    herr_t      status;
    int         **data;
    int         rows, cols;
    int         i, j;


    rows = NX;
    cols = NY;

/************************** BEGIN ******************************/

/* Allocate memory for new integer array[row][col]. First allocate
   the memory for the top-level array (rows).  Make sure you use the
   sizeof a *pointer* to your data type. */

    data = (int**) malloc(rows*sizeof(int*));

/* Allocate a contiguous chunk of memory for the array data
   values.  Use the sizeof data type */

    data[0] = (int*)malloc( cols*rows*sizeof(int) );

/* Set the pointers in the top-level (row) array to the correct
   memory locations in the data value chunk. */

    for (i=1; i < rows; i++) data[i] = data[0]+i*cols;

/************************** END ******************************/

    /* Data and output buffer initialization. */
    for (j = 0; j < NX; j++) {
        for (i = 0; i < NY; i++)
            data[j][i] = i + j;
    }
    /*
     * 0 1 2 3 4 5
     * 1 2 3 4 5 6
     * 2 3 4 5 6 7
     * 3 4 5 6 7 8
     * 4 5 6 7 8 9
     */

    file = H5Fcreate(FILE, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);

    dimsf[0] = NX;
    dimsf[1] = NY;
    dataspace = H5Screate_simple(RANK, dimsf, NULL);

    datatype = H5Tcopy(H5T_NATIVE_INT);
    status = H5Tset_order(datatype, H5T_ORDER_LE);

    dataset = H5Dcreate(file, DATASETNAME, datatype, dataspace,
                        H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
```

(continues on next page)

(continued from previous page)

```
    status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
                      H5P_DEFAULT, &data[0][0]);

    free(data[0]);
    free(data);

    H5Sclose(dataspace);
    H5Tclose(datatype);
    H5Dclose(dataset);
    H5Fclose(file);

    return 0;
}
```

4.3 PNetCDF

Note: for more detailed examples and documentation see PNetCDF.

A simple demonstration of pnetcdf

- text attribute on dataset
- write out rank into 1-d array collectively.
- The most basic way to do parallel i/o with pnetcdf

```
/* This program creates a file, say named output.nc, with the following
   contents, shown by running ncmpidump command .
    % mpiexec -n 4 pnetcdf-write-standard /orangefs/wkliao/output.nc
    % ncmpidump /orangefs/wkliao/output.nc
    netcdf output {
    // file format: CDF-2 (large file)
    dimensions:
            d1 = 4 ;
    variables:
            int v1(d1) ;
            int v2(d1) ;
    // global attributes:
                :string = "Hello World\n",
        "" ;
    data:
         v1 = 0, 1, 2, 3 ;
         v2 = 0, 1, 2, 3 ;
    }
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
#include <pnetcdf.h>

static void handle_error(int status, int lineno)
{
    fprintf(stderr, "Error at line %d: %s\n", lineno, ncmpi_strerror(status));
```

(continues on next page)

(continued from previous page)

```
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int main(int argc, char **argv) {

    int ret, ncfile, nprocs, rank, dimid, varid1, varid2, ndims=1;
    MPI_Offset start, count=1;
    char filename[256], buf[13] = "Hello World\n";
    int data;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

    if (argc > 2) {
        if (rank == 0) printf("Usage: %s filename\n", argv[0]);
        MPI_Finalize();
        exit(-1);
    }
    if (argc > 1) snprintf(filename, 256, "%s", argv[1]);
    else          strcpy(filename, "testfile.nc");

    ret = ncmpi_create(MPI_COMM_WORLD, filename,
                       NC_CLOBBER|NC_64BIT_OFFSET, MPI_INFO_NULL, &ncfile);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    ret = ncmpi_def_dim(ncfile, "d1", nprocs, &dimid);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    ret = ncmpi_def_var(ncfile, "v1", NC_INT, ndims, &dimid, &varid1);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    ret = ncmpi_def_var(ncfile, "v2", NC_INT, ndims, &dimid, &varid2);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    ret = ncmpi_put_att_text(ncfile, NC_GLOBAL, "string", 13, buf);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    /* all processors defined the dimensions, attributes, and variables,
     * but here in ncmpi_enddef is the one place where metadata I/O
     * happens.  Behind the scenes, rank 0 takes the information and writes
     * the netcdf header.  All processes communicate to ensure they have
     * the same (cached) view of the dataset */

    ret = ncmpi_enddef(ncfile);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    start=rank, count=1, data=rank;

    /* in this simple example every process writes its rank to two 1d variables */
    ret = ncmpi_put_vara_int_all(ncfile, varid1, &start, &count, &data);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    ret = ncmpi_put_vara_int_all(ncfile, varid2, &start, &count, &data);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);
```

(continues on next page)

(continued from previous page)

```
    ret = ncmpi_close(ncfile);
    if (ret != NC_NOERR) handle_error(ret, __LINE__);

    MPI_Finalize();

    return 0;
}
```

4.4 VeloC

Note: for more detailed examples and documentation see the Veloc.

Write a checkpoint file every *K* iteratons.

```
MPI_Init(&argc, &argv);
VELOC_Init(MPI_COMM_WORLD, argv[2]); // (1): init

// further initialization code
// allocate two critical double arrays of size M
h = (double *) malloc(sizeof(double *) * M * nbLines);
g = (double *) malloc(sizeof(double *) * M * nbLines);

// (2): protect
VELOC_Mem_protect(0, &i, 1, sizeof(int));
VELOC_Mem_protect(1, h, M * nbLines, sizeof(double));
VELOC_Mem_protect(2, g, M * nbLines, sizeof(double));

// (3): check for previous checkpoint version
int v = VELOC_Restart_test("heatdis", 0);

// (4): restore memory content if previous version found
if (v > 0) {
    printf("Previous checkpoint found at iteration %d, initiating restart...\n", v);
    assert(VELOC_Restart_begin("heatdis", v) == VELOC_SUCCESS);
    char veloc_file[VELOC_MAX_NAME];
    assert(VELOC_Route_file(veloc_file) == VELOC_SUCCESS);
    int valid = 1;
    FILE* fd = fopen(veloc_file, "rb");
    if (fd != NULL) {
        if (fread(&i, sizeof(int),            1, fd) != 1)         { valid = 0; }
        if (fread( h, sizeof(double), M*nbLines, fd) != M*nbLines) { valid = 0; }
        if (fread( g, sizeof(double), M*nbLines, fd) != M*nbLines) { valid = 0; }
        fclose(fd);
    } else
        // failed to open file
        valid = 0;
        assert(VELOC_Restart_end(valid) == VELOC_SUCCESS);
} else
    i = 0;

while (i < n) {
    // iteratively compute the heat distribution

    // (5): checkpoint every K iterations
```

(continues on next page)

(continued from previous page)

```
    if (i % K == 0) {
        assert(VELOC_Checkpoint_wait() == VELOC_SUCCESS);
        assert(VELOC_Checkpoint_begin("heatdis", i) == VELOC_SUCCESS);
        char veloc_file[VELOC_MAX_NAME];
        assert(VELOC_Route_file(veloc_file) == VELOC_SUCCESS);
        int valid = 1;
        FILE* fd = fopen(veloc_file, "wb");
        if (fd != NULL) {
            if (fwrite(&i, sizeof(int),            1, fd) != 1)          { valid = 0; }
            if (fwrite( h, sizeof(double), M*nbLines, fd) != M*nbLines) { valid = 0; }
            if (fwrite( g, sizeof(double), M*nbLines, fd) != M*nbLines) { valid = 0; }
            fclose(fd);
        } else
            // failed to open file
            valid = 0;
        assert(VELOC_Checkpoint_end(valid) == VELOC_SUCCESS);
    }
    // increment the number of iterations
    i++;
}
VELOC_Finalize(0); // (6): finalize
MPI_Finalize();
```

4.5 UnifyFS

Note: for more detailed examples and documentation see the UnifyFS.

4.6 Mercury

Note: for more detailed examples and documentation see Mercury.

4.7 Darshan

Note: for more detailed examples and documentation see Darshan.

CHAPTER 5

Visualization, Analysis, and Data Reduction

5.1 ParaView

Note: for more detailed examples and documentation see ParaView.

5.2 Catalyst

Note: for more detailed examples and documentation see Catalyst User's Guide.

5.3 VTK-m

Note: for more detailed examples and documentation see VTK-m User's Guide.

A simple example of using VTK-m to load a VTK image file, run the Marching Cubes algorithm on it, and render the results to an image:

```
vtkm::io::reader::VTKDataSetReader reader("path/to/vtk_image_file");
vtkm::cont::DataSet inputData = reader.ReadDataSet();
std::string fieldName = "scalars";

vtkm::Range range;
inputData.GetPointField(fieldName).GetRange(&range);
vtkm::Float64 isovalue = range.Center();

// Create an isosurface filter
vtkm::filter::Contour filter;
filter.SetIsoValue(0, isovalue);
filter.SetActiveField(fieldName);
vtkm::cont::DataSet outputData = filter.Execute(inputData);
```

(continues on next page)

(continued from previous page)

```
// compute the bounds and extends of the input data
vtkm::Bounds coordsBounds = inputData.GetCoordinateSystem().GetBounds();
vtkm::Vec<vtkm::Float64,3> totalExtent( coordsBounds.X.Length(),
                                        coordsBounds.Y.Length(),
                                        coordsBounds.Z.Length() );
vtkm::Float64 mag = vtkm::Magnitude(totalExtent);
vtkm::Normalize(totalExtent);

// setup a camera and point it to towards the center of the input data
vtkm::rendering::Camera camera;
camera.ResetToBounds(coordsBounds);

camera.SetLookAt(totalExtent*(mag * .5f));
camera.SetViewUp(vtkm::make_Vec(0.f, 1.f, 0.f));
camera.SetClippingRange(1.f, 100.f);
camera.SetFieldOfView(60.f);
camera.SetPosition(totalExtent*(mag * 2.f));
vtkm::cont::ColorTable colorTable("inferno");

// Create a mapper, canvas and view that will be used to render the scene
vtkm::rendering::Scene scene;
vtkm::rendering::MapperRayTracer mapper;
vtkm::rendering::CanvasRayTracer canvas(512, 512);
vtkm::rendering::Color bg(0.2f, 0.2f, 0.2f, 1.0f);

// Render an image of the output isosurface
scene.AddActor(vtkm::rendering::Actor(outputData.GetCellSet(),
                                      outputData.GetCoordinateSystem(),
                                      outputData.GetField(fieldName),
                                      colorTable));
vtkm::rendering::View3D view(scene, mapper, canvas, camera, bg);
view.Initialize();
view.Paint();
view.SaveAs("demo_output.pnm");
```

5.4 SZ

Note: for more detailed examples and documentation see SZ User Guide.

An example to illustrate compression in C code.

```
int main (int argc , char * argv [])
{
    size_t r5 =0 , r4 =0 , r3 =0 , r2 =0 , r1 =0;

    /* Get the dimension information and set configuration file - cfgFile */
    .....

    /* Initializing the compression environment by loading the configuration file .
    SZ_Init ( null ) will adopt default setting in the compression .*/
    int status = SZ_Init ( cfgFile ) ;
    if( status == SZ_NSCS )
        exit (0) ;
    sprintf ( outputFilePath , "%s.sz", oriFilePath ) ; // specify compression file␣
↪path
```

(continues on next page)

(continued from previous page)

```
    // read the binary data
    size_t nbEle ;
    float * data = readFloatData ( oriFilePath , & nbEle , & status ) ;
    if( status != SZ_SCES )
    {
        printf (" Error : data file %s cannot be read !\n", oriFilePath ) ;
        exit (0) ;
    }

    /* Perform compression . r5 , .... , r1 are sizes at each dimension . The size of␣
↪a
       nonexistent dimension is 0. For instance , for a 3D dataset (10 x20x30 ), the
       setting is r5 = 0 , r4 = 0 , r3 = 10 , r2 = 20 , r3 = 30. SZ_FLOAT indicates␣
↪single
       - precision . */
    size_t outSize ;
    unsigned char * bytes = SZ_compress ( SZ_FLOAT , data , & outSize , r5 , r4 , r3 ,
↪ r2 , r1 ) ;

    // write the compression bytes to ' outputFilePath '
    writeByteData ( bytes , outSize , outputFilePath , & status ) ;
    if( status != SZ_SCES )
    {
        printf (" Error : data file %s cannot be written !\n", outputFilePath ) ;
        exit (0) ;
    }

    /* Do not forget to free the memory of compressed data if they are not useful any␣
↪more .*/
    free ( bytes ) ;
    free ( data ) ;
    SZ_Finalize () ;
    return 0;
}
```

5.5 ZFP

Note: for more detailed examples and documentation see *ZFP <https://zfp.readthedocs.io/>*

Minimal code example showing how to call the zfp (de)compressor:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "zfp.h"

/* compress or decompress array */
static int
compress(double* array, int nx, int ny, int nz, double tolerance, int decompress)
{
  int status = 0;    /* return value: 0 = success */
  zfp_type type;     /* array scalar type */
```

(continues on next page)

(continued from previous page)

```
  zfp_field* field;  /* array meta data */
  zfp_stream* zfp;   /* compressed stream */
  void* buffer;      /* storage for compressed stream */
  size_t bufsize;    /* byte size of compressed buffer */
  bitstream* stream; /* bit stream to write to or read from */
  size_t zfpsize;    /* byte size of compressed stream */

  /* allocate meta data for the 3D array a[nz][ny][nx] */
  type = zfp_type_double;
  field = zfp_field_3d(array, type, nx, ny, nz);

  /* allocate meta data for a compressed stream */
  zfp = zfp_stream_open(NULL);

  /* set compression mode and parameters via one of three functions */
/*  zfp_stream_set_rate(zfp, rate, type, 3, 0); */
/*  zfp_stream_set_precision(zfp, precision); */
  zfp_stream_set_accuracy(zfp, tolerance);

  /* allocate buffer for compressed data */
  bufsize = zfp_stream_maximum_size(zfp, field);
  buffer = malloc(bufsize);

  /* associate bit stream with allocated buffer */
  stream = stream_open(buffer, bufsize);
  zfp_stream_set_bit_stream(zfp, stream);
  zfp_stream_rewind(zfp);

  /* compress or decompress entire array */
  if (decompress) {
    /* read compressed stream and decompress array */
    zfpsize = fread(buffer, 1, bufsize, stdin);
    if (!zfp_decompress(zfp, field)) {
      fprintf(stderr, "decompression failed\n");
      status = 1;
    }
  }
  else {
    /* compress array and output compressed stream */
    zfpsize = zfp_compress(zfp, field);
    if (!zfpsize) {
      fprintf(stderr, "compression failed\n");
      status = 1;
    }
    else
      fwrite(buffer, 1, zfpsize, stdout);
  }

  /* clean up */
  zfp_field_free(field);
  zfp_stream_close(zfp);
  stream_close(stream);
  free(buffer);
  free(array);

  return status;
}
```

(continues on next page)

(continued from previous page)

```
int main(int argc, char* argv[])
{
  /* use -d to decompress rather than compress data */
  int decompress = (argc == 2 && !strcmp(argv[1], "-d"));

  /* allocate 100x100x100 array of doubles */
  int nx = 100;
  int ny = 100;
  int nz = 100;
  double* array = malloc(nx * ny * nz * sizeof(double));

  if (!decompress) {
    /* initialize array to be compressed */
    int i, j, k;
    for (k = 0; k < nz; k++)
      for (j = 0; j < ny; j++)
        for (i = 0; i < nx; i++) {
          double x = 2.0 * i / nx;
          double y = 2.0 * j / ny;
          double z = 2.0 * k / nz;
          array[i + nx * (j + ny * k)] = exp(-(x * x + y * y + z * z));
        }
  }

  /* compress or decompress array */
  return compress(array, nx, ny, nz, 1e-3, decompress);
}
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search