Portable data format by example: `netcdf`

Latin American Introductory School on Parallel Programming and Parallel Architecture for High Performance Computing

William Oquendo, woquendo@gmail.com

Outline

A simple example : a 2D matrix

Saving simulation state to future use

- Printing to binary

Portability

Implementing netcdf for our example

Post-processing: Paraview and netcdf

Beyond : Parallel netcdf

Topic

A simple example : a 2D matrix

Saving simulation state to future use

Printing to binary

Portability

Implementing netcdf for our example

Post-processing: Paraview and netcdf

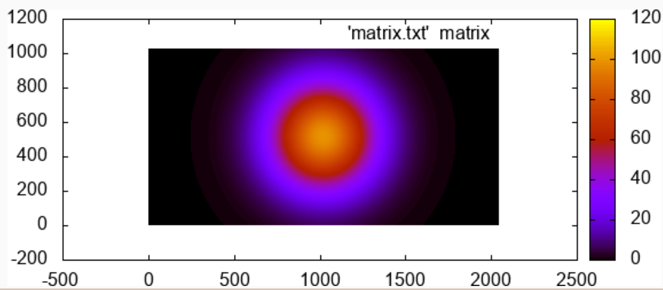
Beyond : Parallel netcdf

Matrix simple creation and printing

```
#include "matrix_io_txt.h"
#include "matrix_util.h"
#include <cmath>

const int NX = 1024;
const int NY = 2048;

int main(void)
{
  double * A = new double [NX*NY] {0.0}; // compile with -std=c++11 or -std=c++0x
  fill(A, NX, NY);
  write_to_txt(A, NX, NY, "matrix.txt");

  return 0;
}
```

Routine to fill the matrix

```
#include "matrix_util.h"
void fill(double *A, int nx, int ny)
{
  double x, y;
  for(int ii = 0 ; ii < nx; ii++) {
    for(int jj = 0 ; jj < ny; jj++) {
      x = (nx/2 - ii); y = (ny/2 - jj);
      A[ii*ny + jj] = 100.032*std::exp(-1.0e-5*(+x*x + y*y));
    }
  }
}
```



```
#include "matrix_io_txt.h"

void write_to_txt(const double * matrix, int nx, int ny, const std::string & fname)
{
  auto t1 = std::chrono::high_resolution_clock::now();
  std::ofstream fout(fname);
  fout.precision(16); fout.setf(std::ios::scientific);
  for(int ii = 0; ii < nx; ++ii) {
    for(int jj = 0; jj < ny; ++jj) {
      fout << matrix[ii*ny + jj] << "  ";
    }
    fout << "\n";
  }
  fout.close();
  auto t2 = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> elapsed = t2 - t1;
  std::printf("out-txt(s): %.4lf\n", elapsed.count());
}

void read_from_txt(double * matrix, int nx, int ny, const std::string & fname)
{
  auto t1 = std::chrono::high_resolution_clock::now();
  std::ifstream fin(fname);
  for(int ii = 0; ii < nx; ++ii) {
    for(int jj = 0; jj < ny; ++jj) {
      fin >> matrix[ii*ny + jj];
    }
  }
  fin.close();
```

How much type to print? How large is a typical file?

We compile and run it like

```
g++ -std=c++11 main_matrix_txt.cpp matrix_io_txt.cpp matrix_util.cpp
./a.out
```

```
out-txt(s): 2.9394
```

And the size of the written file is

```
ls -sh matrix.txt
```

```
49M matrix.txt
```

Topic

A simple example : a 2D matrix

Saving simulation state to future use

Printing to binary

Portability

Implementing netcdf for our example

Post-processing: Paraview and netcdf

Beyond : Parallel netcdf

Why saving intermediate states is important?

- Maybe the simulation takes several days/weeks/months.
- Maybe the initialization is costly.
- Sometimes accidents happen: power grid failure, people just turn off computers, etc.
- Maybe you want to perform intermediate post-processing.
- etc

Therefore . . .

- It is advisable to be able to restart the simulation.
- We need to read back the data at the point previous to failure!

Reading data back in text mode

Writing and reading using text mode

```
// compile with -std=c++11 or -std=c++0x
#include "matrix_io_txt.h"
#include "matrix_util.h"
#include <cmath>
#include <iostream>

const int NX = 1024;
const int NY = 2048;

int main(void)
{
  double * A = new double [NX*NY] {0.0};
  fill(A, NX, NY);

  write_to_txt(A, NX, NY, "matrix.txt");
  read_from_txt(A, NX, NY, "matrix.txt");

  return 0;
}
```

Results

out-txt(s):	2.2384
in-txt(s):	3.9741

Remarks

- This is taking a lof of time. How to solve it?
- The solution might be to print to a binary file.

Reading data back in text mode

Writing and reading using text mode

```
// compile with -std=c++11 or -std=c++0x
#include "matrix_io_txt.h"
#include "matrix_util.h"
#include <cmath>
#include <iostream>

const int NX = 1024;
const int NY = 2048;

int main(void)
{
  double * A = new double [NX*NY] {0.0};
  fill(A, NX, NY);

  write_to_txt(A, NX, NY, "matrix.txt");
  read_from_txt(A, NX, NY, "matrix.txt");

  return 0;
}
```

Results

out-txt(s):	2.2384
in-txt(s):	3.9741

Remarks

- This is taking a lof of time. How to solve it?
- The solution might be to print to a binary file.

Saving simulation state to future use

Printing to binary

```
#include "matrix_io_bin.h"

void write_to_bin(const double * matrix, int nx, int ny, const std::string & fname)
{
  auto t1 = std::chrono::high_resolution_clock::now();
  std::ofstream fout(fname, std::ios::binary);
  for(int ii = 0; ii < nx; ++ii) {
    for(int jj = 0; jj < ny; ++jj) {
      fout.write((char *)&matrix[ii*ny + jj], sizeof(double));
    }
  }
  fout.close();
  auto t2 = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> elapsed = t2 - t1;
  std::printf("out-bin(s): %.4lf\n", elapsed.count());
}

void read_from_bin(double * matrix, int nx, int ny, const std::string & fname)
{
  auto t1 = std::chrono::high_resolution_clock::now();
  std::ifstream fin(fname, std::ios::binary);
  for(int ii = 0; ii < nx; ++ii) {
    for(int jj = 0; jj < ny; ++jj) {
      fin.read((char *)&matrix[ii*ny + jj], sizeof(double));
    }
  }
  fin.close();
  auto t2 = std::chrono::high_resolution_clock::now();
```

Writing/reading in binary mode

Main function

```
// compile with -std=c++11 or -std=c++0x
#include "matrix_io_txt.h"
#include "matrix_io_bin.h"
#include "matrix_util.h"
#include <cmath>
#include <iostream>
const int NX = 1024;
const int NY = 2048;
int main(void)
{
  double * A = new double [NX*NY] {0.0};

  fill(A, NX, NY);

  write_to_txt(A, NX, NY, "matrix.txt");
  write_to_bin(A, NX, NY, "matrix.dat");
  read_from_txt(A, NX, NY, "matrix.txt");
  read_from_bin(A, NX, NY, "matrix.dat");
  return 0;
}
```

Results

out-txt(s):	2.6566
out-bin(s):	0.2082
in-txt(s):	3.8641
in-bin(s):	0.1252

16M	matrix.dat
49M	matrix.txt

- This is very good. Printing is faster and produces smaller files, but ...

Writing/reading in binary mode

Main function

```
// compile with -std=c++11 or -std=c++0x
#include "matrix_io_txt.h"
#include "matrix_io_bin.h"
#include "matrix_util.h"
#include <cmath>
#include <iostream>
const int NX = 1024;
const int NY = 2048;
int main(void)
{
  double * A = new double [NX*NY] {0.0};

  fill(A, NX, NY);

  write_to_txt(A, NX, NY, "matrix.txt");
  write_to_bin(A, NX, NY, "matrix.dat");
  read_from_txt(A, NX, NY, "matrix.txt");
  read_from_bin(A, NX, NY, "matrix.dat");
  return 0;
}
```

Results

out-txt(s):	2.6566
out-bin(s):	0.2082
in-txt(s):	3.8641
in-bin(s):	0.1252

16M	matrix.dat
49M	matrix.txt

- This is very good. Printing is faster and produces smaller files, but ...

Topic

A simple example : a 2D matrix

Saving simulation state to future use

Printing to binary

Portability

Implementing netcdf for our example

Post-processing: Paraview and netcdf

Beyond : Parallel netcdf

Sharing results

1. Now I (proudly) send the final result to my supervisor.
2. But he works on windows and strangely he cannot read the data!
3. What happened? Now I am in trouble. Binary formats are not portable!

This could happen if:

- You are using platforms with different endianess
- Embedded/exotic platforms
- You are not using standard IEEE754 datatypes

How to solve this? Find a binary portable data format. So you need to go to serialization $\rightarrow$ Lot of work!

Finding the right data format

Let me google that for you

Scientific_data

Finding the right data format

Let me google that for you

Scientific_data

Portable data formats

- xdmf (wrapper to hdf5 with lightweight metadata)

What is netcdf? (`module load netcdf`)

From unidata site
NetCDF is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data (Latest version 4.6.0).

Self-describing It has metadata about the data it contains.

Portable *Can be accessed by different platforms!*

Scalable Small subsets can be accessed efficiently.

Appendable Data may be appended without redefining the structure.

Sharable Multiple access to the same file.

Bindings You can use it from `c`, `c++`, `python`, `fortran`

HDF5 Already uses hdf5 underlying, but much more easy to handle.

Criticism Not a database system, no transactions, parallel io through another package (no longer true).

Topic

A simple example : a 2D matrix

Saving simulation state to future use

- Printing to binary

Portability

Implementing netcdf for our example

Post-processing: Paraview and netcdf

Beyond : Parallel netcdf

For our simple example, this is a simple routine to write a `netcdf` file. You can create compound types or just write arrays.

```
// based on https://www.unidata.ucar.edu/software/netcdf/examples/programs/simple_xy_wr.c
#include "matrix_io_netcdf.h"
void write_to_netcdf(const double *A, int NX, int NY, const std::string & fname,
                     int deflate_level)
{
  const int NDIMS = 2;
  int ncid, dimids[NDIMS], varid;
  auto t1 = std::chrono::high_resolution_clock::now();
  nc_create(fname.c_str(), NC_CLOBBER | NC_NETCDF4, &ncid);
  /* Define the dimensions. NetCDF will hand back an ID for each. */
  nc_def_dim(ncid, "x", NX, &dimids[0]);
  nc_def_dim(ncid, "y", NY, &dimids[1]);
  nc_def_var(ncid, "data", NC_DOUBLE, NDIMS, dimids, &varid);
  /* set COMPRESSION!!!! This works better for non-contiguous data*/
  int shuffle = 0, deflate = 1;
  nc_def_var_deflate(ncid, varid, shuffle, deflate, deflate_level);
  nc_enddef(ncid); // done defining data

  nc_put_var_double(ncid, varid, &A[0]); // write all data
  nc_close(ncid);
  auto t2 = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> elapsed = t2 - t1;
  std::printf("out-netcdf(s): %.4lf\n", elapsed.count());
}
```

Now read it by using the same interface,

```
#include "matrix_io_netcdf.h"

void read_from_netcdf(double *A, const std::string & fname)
{
  int ncid, varid;
  auto t1 = std::chrono::high_resolution_clock::now();
  nc_open(fname.c_str(), NC_NOWRITE, &ncid);
  nc_inq_varid(ncid, "data", &varid); // variable name
  nc_get_var_double(ncid, varid, &A[0]); // A must have the right size
  nc_close(ncid);
  auto t2 = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> elapsed = t2 - t1;
  std::printf("in-netcdf(s): %.4lf\n", elapsed.count());
}
```

Using them

```
#include <iostream>
#include "matrix_util.h"
#include "matrix_io_txt.h"
#include "matrix_io_bin.h"
#include "matrix_io_netcdf.h"

const int NX = 1024;
const int NY = 2048;

int main(void)
{
  double * data = new double [NX*NY] {0.0}; // compile with -std=c++11 or -std=c++0x

  fill(data, NX, NY);

  write_to_txt(data, NX, NY, "matrix.txt");
  write_to_bin(data, NX, NY, "matrix.dat");
  write_to_netcdf(data, NX, NY, "matrix-deflate1.nc", 1);
  write_to_netcdf(data, NX, NY, "matrix-deflate9.nc", 9);

  read_from_txt(data, NX, NY, "matrix.txt");
  read_from_bin(data, NX, NY, "matrix.dat");
  read_from_netcdf(data, "matrix-deflate1.nc");
  read_from_netcdf(data, "matrix-deflate9.nc");
```

Results

Compiling and running

```
g++  -std=c++11 main_netcdf.cpp matrix_out_netcdf.cpp matrix_in_netcdf.cpp \
 matrix_io_txt.cpp matrix_io_bin.cpp matrix_util.cpp -lnetcdf && ./a.out
ls -sh matrix-deflate*.nc matrix.{txt,dat} | column -c 1
```

out-txt(s):	3.1977
out-bin(s):	0.1248
out-netcdf(s):	1.2359
out-netcdf(s):	0.933
in-txt(s):	3.0
in-bin(s):	0.0656
in-netcdf(s):	0.0886
in-netcdf(s):	0.0865
14M	matrix-deflate1.nc
14M	matrix-deflate9.nc
16M	matrix.dat
49M	matrix.txt

Netcdf tools to process data

Command line tools

- `nccopy` Copy a netCDF file, optionally changing format, compression, or chunking in the output.
- `ncdump` Convert netCDF file to text form (CDL).
- `ncgen` From a CDL file generate a netCDF-3 file, a netCDF-4 file or a C program.

```
ncdump -h matrix-deflate1.nc
```

```
netcdf matrix-deflate1 {
dimensions:
        x = 1024 ;
        y = 2048 ;
variables:
        double data(x, y) ;
}
```

Example of a CDL file

```
netcdf co2 {
dimensions:
T = 456 ;
variables:
float T(T) ;
T:units = "months since 1960-01-01" ;
float co2(T) ;
co2:long_name = "CO2 concentration by volume" ;
co2:units = "1.0e-6" ;
co2:_FillValue = -99.99f ;

// global attributes:
:references = "Keeling_etal1996, Keeling_etal1995"
```

Topic

A simple example : a 2D matrix

Saving simulation state to future use

Printing to binary

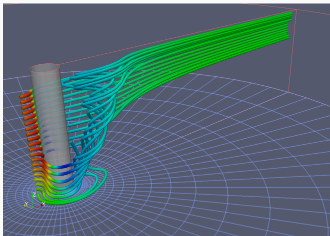
Portability

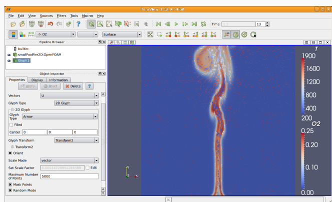
Implementing netcdf for our example

Post-processing: Paraview and netcdf

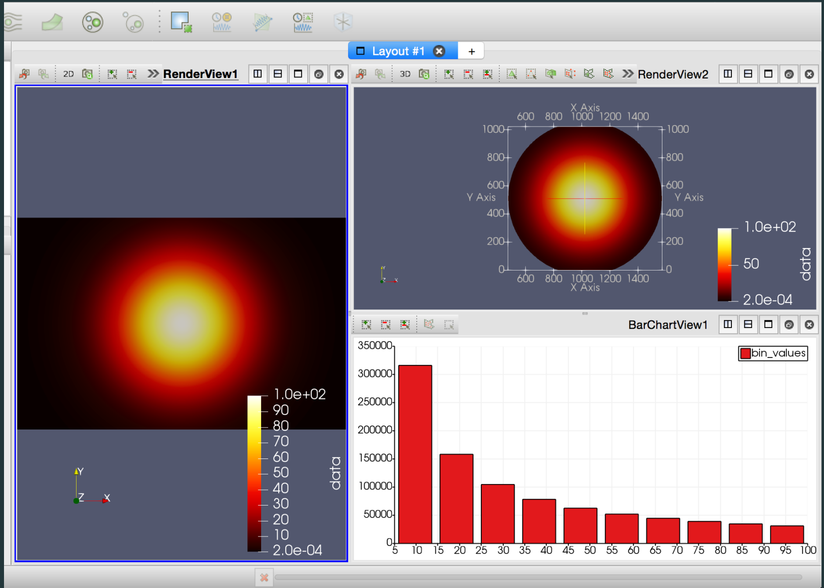
Beyond : Parallel netcdf

What is Paraview? https://www.paraview.org





- ParaView is an open-source, multi-platform data analysis and visualization application.
- Support distributed computation models to process large data sets, including both cpu and gpu.
- Many file readers, including netcdf!
- Allows post-processing and data analysis.
- Python scripting to create powerful visualizations.
- Deployed on Windows, Mac OS X, Linux, SGI, IBM Blue Gene, Cray and various Unix workstations, clusters and supercomputers



Layout #1
RenderView1
RenderView2
BarChartView1
X Axis
Y Axis
data
1.0e+02
50
2.0e-04
bin_values

Topic

A simple example : a 2D matrix

Saving simulation state to future use

Printing to binary

Portability

Implementing netcdf for our example

Post-processing: Paraview and netcdf

Beyond : Parallel netcdf

Direct support in netcdf-4

See https://www.unidata.ucar.edu/software/netcdf/docs/netcdf__par_8h.html#details

- nc_var_par_access : Sets parallel access to collective or independent.
- nc_create_par : Creates a new netCDF file for parallel I/O (NC_NETCDF4|NC_MPIIO) access.

```
int nc_create_par (const char * path, int cmode, MPI_Comm comm,
                   MPI_Info info, int * ncidp);
```

- nc_open_par() : Opens a file in parallel mode
- There is another available project: https://trac.mcs.anl.gov/projects/parallel-netcdf

Example in the cluster

Example from: https://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf-c/parallel-access.html

```
MPI_Init(&argc,&argv);
...
nc_create_par(FILE, NC_NETCDF4|NC_MPIIO, comm,
                              info, &ncid);
...
nc_var_par_access(ncid, v1id, NC_COLLECTIVE);
...
nc_put_vara_int(ncid, v1id, start, count,
                &data[mpi_rank*QTR_DATA]);
```

To test it on Abacus-I:

```
module load netCDF/c/4.4.1-rc2
module load netCDF/parallel/1.7.0
mpicc parallel_netcdf.c  -lpnetcdf -lnetcdf -lhdf5 -lhdf5_hl
mpiexec.hydra -np 4 -ppn 4 ./a.out
```

Results

```
$ mpiexec.hydra -np 4 -ppn 4 ./a.out # run it in the cluster, not the login node
mpi_name: service0 size: 4 rank: 2
mpi_name: service0 size: 4 rank: 0
mpi_name: service0 size: 4 rank: 3
mpi_name: service0 size: 4 rank: 1
mpi_rank=3 start[0]=18 start[1]=0 count[0]=6 count[1]=24
mpi_rank*QTR_DATA=432 (mpi_rank+1)*QTR_DATA-1=576
mpi_rank=2 start[0]=12 start[1]=0 count[0]=6 count[1]=24
mpi_rank*QTR_DATA=288 (mpi_rank+1)*QTR_DATA-1=432
mpi_rank=1 start[0]=6 start[1]=0 count[0]=6 count[1]=24
mpi_rank*QTR_DATA=144 (mpi_rank+1)*QTR_DATA-1=288
mpi_rank=0 start[0]=0 start[1]=0 count[0]=6 count[1]=24
mpi_rank*QTR_DATA=0 (mpi_rank+1)*QTR_DATA-1=144
```

```
-rw-r--r-- 1 user6  19K Feb 18 22:18 test_par.nc
```

Conclusions

The `netcdf4` file format offers you a tool to write files that are

- binary portable
- compressed
- self-describing
- easy to use from c, python, fortran, R, ...
- can be directly visualized in paraview (and xmgrace)
- can write in parallel (work in progress)

Thank you