Adaptive I/O System (ADIOS)

Michael Neff

Agenda

- Key features
- Basic usage
- More advanced usage
- Converters
  - To HDF5, Netcdf, …

What is ADIOS?

- An I/O abstraction framework
- Provides a portable, fast, scalable, easy to use, metadata rich output
- One API regardless of the method used for I/O
  - Almost as easy as F90 write statements
- Change I/O method on the Fly

- Adios uses a common XML file for:
  - Data grouping
  - Selection of I/O method
  - Buffer-sizes
- Common Tools
  - Buffering
  - Scheduling

Key Features

- Ease of use
- High performance
- Scalable
- Portable
- Easy to master

Design Goals

- **Data Grouping: Multiple, independently controlled IO settings**
  - diagnostics, restarts, analysis, vizualization
- **Optional data items: Different data from different processes**
  - Single group write has header from a single proc and data from all
- **Data sizes change dynamically: Datasets vary in size**
  - Run size changes data sizes; also codes using aggregators
- **Constructed output is special: Reused buffers must be handled properly**
  - stack temporaries...
- **IO memory is second to science: Buffer space for IO is strictly limited**
  - respect the memory needs of the scientific codes

ADIOS API

How to use: Startup

- **Adios_init**
  - parse the xml config file for each process
  - Setup transport method
  - Has to be specified after MPI_Init
  - For a serial program a special library is available

- **C/C++**
  - int adios_init (const char * xml_fname , MPI_Comm comm)
- **Fortran**
  - call adios_init ("config.xml", comm , ierr)

How to use: Cleanup

- **Adios_finalize**
  - Give each transport method opportunity to cleanup
  - Particularly important for asynchronous methods to make sure they have completed before exiting
  - Has to be specified before MPI_finalize

- **C/C++**
  - int adios_finalize (int rank)

- **Fortran**
  - call adios_finalize (rank , ierr)

How to use: Open a file

- adios_open(handle, ‘group name’, ‘file name’, mode, MPI_comm comm)
  - Handle used for subsequent calls of read/write/close
  - ‘group name’ is the same as in the XML file
  - ‘mode’ is one of “r” read, “w” write, “a” append or “u” update
- Fortran
  - call adios_open (handle , ‘group name’, ‘file name’, mode, comm , ierr)
  - handle is either integer*8 or pointer type
- For closing the file:
  - int adios_close (handle)
  - call adios_close (handle , ierr)

How to use: read/write

- **adios_write (handle, ‘name’, data)**
  - handle from open
  - name of variable or attribute var in XML for this group
  - data reference
- **adios_read (handle, ‘name’, size, buffer)**
  - handle from open
  - name of variable in XML for this group
  - buffer to store read value
- **Fortran**
  - call adios_write (handle , ‘name’, var, ierr)
  - call adios_read (handle , ‘name’, 8, var, ierr)

Must specify one per variable written or read

XML config file

XML config file

- **Usually the adios_write and adios_read commands are not specified in the source code.**
- **The gpp.py python script processes XML file**
  - creates include files
  - These include files contain the adios_write and adios_read statements

Basic XML file

```
<? xml version="1.0 "?>
<adios-config >
<adios-group >
        <var ... />
        <attribute .../ >
</adios-group >
<method ... />
<buffer ... />
</adios-config >
```

XML: adios-config

- **Container for all IO**
- **Only one option: host-language**
  - Can be either C or Fortran

```
<? xml version ="1.0 "?>
<adios-config host-language=“Fortran>
...
</adios-config >
```

XML: adios-group

- **Container for a set of variables**
  - Sharing a common IO pattern
- **As many groups as needed available**
- **Options:**
  - host-language – language the program part is written in. (required)
  - name – descriptive string to name the group. (optional)

```
<adios-config …>
<adios-group name=“restart” host-language=“C”>
...
</adios-group >
</adios-config >
```

XML: var

- **Variables var**
  - (nested) element for adios-group
  - Can be array or primitive datatype
    - Determined by the attribute dimension

- **Attributes**
  - name – string name of the variable stored in the output file (required)
    - can be arbitrary strings or full paths containing "\"
  - type – data type of the variable (required)
  - gwrite – name of the variable in the source code (optional)
    - used to generate adios_write calls
    - if not specified name will be used as default
    - C or Fortran expressions allowed
  - gread – as gwrite but for adios_read routines(optional)

XML: var - continued

- **More attributes**
  - path – hdf5 style path, Obsolete since path can be in name(optional)
  - dimensions – comma separated list (optional)
    - corresponding to integer values
    - variable scalar if not specified
  - read – either yes or no (optional)
    - if no gread will not be created
    - default yes

```
<adios-group …>
<var name="z- plane ion particles"
gwrite="zion "
gread="zion_read "
type="adios_real "
dimensions="7, mimax "
read="yes"/>
</adios-group >
```

XML: attributes

- **Attributes attribute**
  - Ability to specify more descriptive information about variable or group
  - Can be defined either static or dynamic
  - Only scalar definition possible
  - Only root process writes the attribute

- **Options**
  - name – name of the attribute
  - path - hierarchical path inside the file for the attribute
  - value – attribute has static value of the attribute
    - mutually exclusive with the attribute var
  - type - string or numeric type, paired with attribute value
    - mutually exclusive with the attribute var also
  - var - attribute has dynamic value
    - defined by a variable in var

XML: attributes examples

- **static declaration:**

```
<attribute name =" experimental date "
path ="/zion "
value ="Sep-19-2008 "
type =" adios_real "/>
```

- **dynamic declaration:**

```
<attribute name =" experimental date "
path ="/zion "
var =" time"/>
```

XML: gwrite source

- **Contents of src will be copied to the include file.**
- **Write a subset of variables conditionally**
  - depending on a logical expression in the source code

```
<gwrite src="if ( have_ions ==1) then "/>
...
<gwrite src=" endif "/>
```

XML: global-bounds

- **Optional nested element for adios group**
- **Specifies the global space and offsets for the variables within that space.**
- **Options:**
  - dimensions - the dimension of global space
  - offsets - the offset of the data set in global space

```
<global - bounds dimensions ="nx_g , ny_g " offsets ="nx_o ,0"/>
... variable declarations ...
</ global - bounds >
```

XML: Time

- **Dataset can be expanded in space domain as well as in time domain.**
  - Space domain is specified by global bounds
  - File contains a group of time-based variables with undetermined dimension on the time axis
- **Writing: just append to the file.**
  - No need to specify anything (since ADIOS 1.6)
- **Reading: all timesteps will be visible**

XML: Transport group

- **Specifies the mapping of the transport method used.**
  - Either *transport* or *method* can be used
- **Options:**
  - group – corresponds to an adios-group specified earlier in the file.
  - method – a string indicating a transport method to use
    - applies to the associated adios-group
  - priority – a numeric priority for the I/O method
    - used to better schedule this write with others that may be pending currently
  - iterations – a number of iterations between writes of this group
    - used to gauge how quickly this data should be evacuated from the compute node

```
<transport group =" restart "
method ="MPI"
priority ="1"
iteration ="100"/>
```

Transport methods

- NULL
- POSIX
- MPI
- MPI_LUSTRE
- MPI_AGGREGATE (or MPI_AMR)
- VAR_MERGE
- PHDF5 (for Parallel HDF5)
- NC4PAR (for Parallel NetCDF4)
- DATASPACES (or DART)
- DIMES
- FLEXPATH

- Use *adios_config –m* to check for available methods

XML: Buffer size

- Specifies the size and the time of internal buffer creation
- Options:
  - size-MB - the user-defined size of buffer in megabytes
    - Maximum ADIOS can allocate from compute nodes.
    - Exclusive with free-memory-percentage.
  - free-memory percentage – user-defined percentage from 0 to 100% of free memory available on the machine.
    - Exclusive with size-MB.
  - allocate-time - indicates when the buffer should be allocated
    - either “now” or “on call”

```
<buffer size -MB="100"
allocate - time ="now" />
```

Example config file

```
<adios-config host-language="C">
        <adios-group name=" temperature" coordination-communicator =" comm ">
                <var name="NX" type=" integer "/>
                <var name="t" type="double" dimensions="NX"/>
                <attribute name=" recorded date " path="/" value="Sep 19, 2008 "
                        type =" string "/>
        <! -- conditional writing of a variable -->
                <gwrite src="if ( want_x ) {" />
                <var name ="x" type =" integer " dimensions ="NX"/>
                <gwrite src="}" />
        </adios-group >
        <method group =" temperature " method =" MPI"/>
        <buffer size-MB="1" allocate - time ="now "/>
        <analysis adios-group=" temperature " var="t"
                break-points="0, 100 , 200 , 300 "/>
</adios - config >
```

Testcase

- A little Fortran program running on 24 cores.
- Writing 27 GB of data.
- Timings will be compared

Fotran Code 1

```
program adiostest
    use adios_write_mod

    implicit none

    ... Variable Declaration ...

    ... MPI initialisation ...

    irdim=300000000
    allocate(drank(myrank*irdim+1:myrank*irdim+irdim))
    rank=myrank
    gsize=mysize*irdim
    ierr=0

    do i=1,irdim,2
      drank(myrank*irdim+i)=myrank
      drank(myrank*irdim+i+1)=-myrank
    end do

    offset=irdim*myrank+1
```

Fotran Code 2

```
  call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!! Initialize adios
  ierr=0
  call adios_init("config.xml",MPI_COMM_WORLD,ierr)
!! Open file
  call adios_open(adios_handle,"writer","restart.bp","w",MPI_COMM_WORLD,ierr)
  include "gwrite_writer.fh"
!!close file
  call adios_close(adios_handle,ierr)
  ierr=0
  call adios_finalize(myrank,ierr)
  deallocate(drank)
  call MPI_FINALIZE(ierr)

end
```

XML File

```
<?xml version="1.0"?>
<adios-config host-language="Fortran">
 <adios-group name="writer">
  <var name="offset" type="integer" read="yes"/>
  <var name="mysize" type="integer" read="yes"/>
  <var name="irdim" type="integer"  read="yes"/>
  <var name="gsize" type="integer"  read="yes"/>
  <global-bounds dimensions="gsize" offsets="offset">
   <var name="drank" type="integer" dimensions="irdim" read="yes"/>
  </global-bounds>

 </adios-group>

 <transport group="writer" method="MPI">
  num_aggregators=24,num_ost=4
 </transport>
 <buffer size-MB="1224" allocate-time="now"/>

</adios-config>
```

Makefile

```
ADIOS_DIR = /univ_1/ws1/ws/hpcmneff-ADIOS-0/adios-install
MXML_DIR = /univ_1/ws1/ws/hpcmneff-ADIOS-0/mxml-install
ADIOS_INC=-I${ADIOS_DIR}/include
ADIOS_FLIB= -L${ADIOS_DIR}/lib -ladiosf -ladios -ladiosread -ladiosreadf -L${MXML_DIR}/lib -lmxml

FC     = ftn
FFLAGS  = $(ADIOS_INC)
LDFLAGS = $(ADIOS_FLIB)
GPP     = ${ADIOS_DIR}/bin/gpp.py

OBJ     = test1.o

default: test1

test1: $(OBJ)
	$(FC) $(FFLAGS) -o test1 $(OBJ) $(LDFLAGS)

clean:
	rm *.o test1 *.fh

config.fh: config.xml
	${GPP} config.xml

test1.o: test1.f90 config.fh
	$(FC) $(FFLAGS) -c test1.f90
```

Include File gwrite_writer.fh

created by gpp.py

```
adios_groupsize = 4 &
                + 4 &
                + 4 &
                + 4 &
                + 4 * (irdim)
call adios_group_size (adios_handle, adios_groupsize, adios_totalsize,
adios_err)
call adios_write (adios_handle, "offset", offset, adios_err)
call adios_write (adios_handle, "mysize", mysize, adios_err)
call adios_write (adios_handle, "irdim", irdim, adios_err)
call adios_write (adios_handle, "gsize", gsize, adios_err)
call adios_write (adios_handle, "drank", drank, adios_err)
```

Timings

- **POSIX**
  - Runtime: 26.5 s
  - Without Buffering: 44.38s
- **MPI**
  - Runtime: 40.64 s
  - Without Buffering: 39.69s
- **MPI_AGGREGATE**
  - Runtime: 23.90 s
  - Without Buffering: 30.9 s

Further information

- For further information see:
- https://github.com/ornladios/ADIOS/blob/master/tutorial/adios_api_scidac_tutorialv2.pptx

- http://adiosapi.org/index.php5?title=Main_Page

Adios finalize!

Summary

I/O performance: to keep in mind

- There is no “one size fits all” solution to the I/O problem
- Many I/O patterns for well for some range of parameters
  - When scaling out you might want to change your implemention
- Bottlenecks in performance can occur in different places
  - application level
  - File system
- Going to extremes with an I/O pattern will typically lead to problems

- I/O is a shared resource: except timing variation

Summary

- **I/O is always a bottleneck when scaling out**
  - You may have to change your I/O implementation when scaling up
- **Take-home messages on I/O performance**
  - Performance is limited for single process I/O
  - Parallel I/O utilizing a file-per-process or a single shared file is limited at large scales
  - Potential solution is to utilize multiple shared file or a subset of processes which perform I/O
  - A dedicated I/O Server process (or more) might also help
  - Use MPI I/O and/or high-level libraries (HDF5, NETCDF, ADIOS ..)
- **Select the best Lustre striping parameters for your case!**

Thank You !!!