WP7/JRA1

Deliverable 7.5 – Reference implementations of IO server combined with Parallel I/O

Abstract:

This document describes the development and testing of two approaches to improve the performance of I/O for climate and weather models. The first development is called XIOS which has been released with the latest release of the NEMO ocean model. The second development is called CDI-pio and has been demonstrated within an early version of the ECHAM6 model. For both developments, work is underway to demonstrate that they can be used in a range of different models.

Grant Agreement Number:	**228203**	**Proposal Number:**	**FP7-INFRA-2008-1.1.2.21**
Project Acronym:	**IS-ENES**		
Project Co-ordinator:	**Dr Sylvie JOUSSAUME**		

Document Title:	*Reference implementations of IO server combined with Parallel I/O*			**Deliverable:**	D 7.5
Document Id N°:		**Version:**	1	**Date:**	March 4th 2013
Status:	Final				
Filename:					
Project Classification:	Public				

Document Authors	
Yann Meurdesoif	CNRS-IPSL(1)
Thomas Jahns	DKRZ (4)
Luis Kornblueh	DKRZ (4)
Steve Mullerworth (Executive	METOFFICE (10)

Executive Summary

Work on IO within JRA1 of the IS-ENES project envisaged a review of a range of existing IO technology (D7.3) and, additionally, development of existing and new technology to meet existing and future needs of climate models. At the beginning of the project, the two main techniques were referred to as parallel IO (writing multiple files in parallel, or writing an individual file using output from multiple processors), and IO server (writing data from the model to independent IO processes, which allows the model to continue its integration while the file contents are still being written to disk). In principle the two techniques can beneficially be used together.

At the beginning of the project, different partners were researching and implementing different techniques and it was envisaged that within the lifetime of the project, aspects of the two techniques could have been combined together. In practice, two main approaches have been considered under the umbrella of this work package both including parallel IO and IO server but responding to different needs. These are:

- The further development of an IO server system, using parallel IO functionality of NetCDF, at CNRS-IPSL which has generated the XIOS system that was released with the latest version of NEMO
- A review and trialling of existing parallel IO packages at DKRZ (D7.1), which has resulted in extending the existing CDI serial IO library for parallel IO with IO servers.

At the same time, the Met Office was implementing its own IO server system and envisaged sharing its experiences with the other partners and trialling the software that was being developed.

To this end, two activities took place to try and combine techniques:

- An IO workshop took place, and work plans to combine approaches were discussed.
- The Met Office explored the use of XIOS as a complement to the Met Office's own IO server developments.

In practice, developing and demonstrating the approaches chosen at each individual partner institution was significantly more time-consuming than was envisaged at the start of the project, which reduced resources available for collaboration. Furthermore, at the time XIOS was explored at the Met Office, the application was not fully developed, so implementing it in the Met Office model turned out to be too difficult.

That said, the XIOS server can be considered as fulfilling the deliverable objectives, as it is an IO server implementing parallel IO. The XIOS server is being tested within the NEMO model at a number of different institutions and on a number of different computing platforms."

This document briefly discusses the potential complementarities between the two approaches being taken. Then two further main sections describe each approach (XIOS IO server followed by CDI-pio parallel IO based system) in more detail as well as the results of tests.

Are XIOS and CDI-pio complementary?

There is some agreement among the partners that the CDI-PIO integration developed at DKRZ can complement the XIOS server developed at CNRS-IPSL by being integrated at the lowest level of the XIOS system, and this combination may be very beneficial in future computing platforms.

However, the two systems are targeting some different requirements. For example, CDI-pio favours GRIB format files. When combined with compression, this format typically results in a $1/6^{th}$ reduction in storage when compared to similar NetCDF output files and is therefore suited for archival purposes. However, it is not well adapted for application-level restart. XIOS favours NetCDF format files, which are better suited for sharing of individual data sets and more widely used within the climate community, but it does not yet support any succinct format like GRIB directly. NetCDF is also better supported by many post-processing applications.

As emphasized in D7.3, only XIOS uses external configuration files. This makes the Application Programming Interface (API) that is used by the climate models simpler. In addition, it makes changing of the output configuration easier and possibly faster, because the model might not have to be recompiled, if only the external files have to be adjusted.

In terms of implementing a new IO system within an existing model, both XIOS and CDI-pio require changes to the workflow of an experiment besides changes to the model source code (setup of the XML description files for XIOS for example). However, the CDI-pio API is a compatible extension of the CDI API and thus requires very little change to existing workflows including CDI. XIOS also does support some post-processing for which there are currently no plans within CDI-pio.

XIOS

Yann Meurdesoif: CNRS-IPSL

Motivation

Management of output files and diagnostics from climate simulations is becoming a source of growing concern. The increase in complexity of models, resolution of models and number of variables output by models results in the writing and storing of growing data volumes and challenges model performance.

The motivation for the development of XIOS arose from experiences with the IO libraries used by the IPSL models. During CMIP5 (Climate Model Intercomparison Project 5), these Earth system models had to deliver more than 800 output variables, each one associated with a large amount of metadata (name, description, unit, associated grid, associated files, etc.). Diagnostic operations on these variables also needed to be performed (such as time average). The management of metadata within the code unnecessarily overloaded I/O library calls, affecting the clarity and readability of the code. For performance reasons it was also necessary to keep many indices (handles) related to variable definitions or files, and organise the code so that I/O calls are concentrated in few portions of code. All these aspects harmed the modularity of the models. In addition, each change in the definition of

the choice of outputs or metadata involved recompiling the model. All this demonstrated the lack of flexibility of I/O libraries used for the outputs of our models.

In addition, the available libraries did not offer parallelism management. Each MPI process over a computational subdomain wrote its output in a separate file, corresponding to its portion of domain. The reconstruction of the global file was done later in the post-processing stage. At the resolutions of our models, the time needed to generate local files during the simulation and to reconstruct the global file post-processing was becoming large, increasing the cost of the post-processing workflow. This method remains usable for simulations using up to O (100) processes. Beyond, the overload of the file system results in prohibitive rebuild times. Beyond O (1000) process, it becomes impossible to manage our outputs by this method.

XIOS (XML-IO-Server) is a new tool developed at CNRS-IPSL dedicated to managing files and output diagnostics climate simulation models. It addresses the problems of flexibility and performance mentioned above.

To improve flexibility, XIOS greatly simplifies the method for defining outputs from a model by describing them in an XML file, parsed at runtime. By implementing concepts of inheritance hierarchy within the syntax of the XML, definitions are much more compact and non-redundant. Sending fields at each time step from code then requires no more than 2 arguments: the identifier field (string) name which relates to an externally held field definition such as those defined by the CF convention, and the address of the field data in memory. The metadata for the resulting output field can be determined from the identifier field combined with the post-processing choices requested in the XML file.

To improve performance, a proportion of the processors used to run the model are set up to run as “IO servers” exclusively dedicated to file operations. Data are transferred from client processes (models) to server processes through asynchronous communications, allowing concurrency of computing, data transfer and write access to the file system. In addition, through the sequence of layers NETCDF4/HDF5/MPI-IO libraries, we exploit the parallelism of the computer file system. First, by aggregating the bandwidth of the system file, we are able to write a much larger volume of data for the same period of time. Second, server processes simultaneously write their data into a single parallel file, avoiding the costly rebuild phase in post-processing.

XIOS is written in C + + and is currently made of about 35,000 lines of code available in SVN from the site: http://forge.ipsl.jussieu.fr/ioserver.

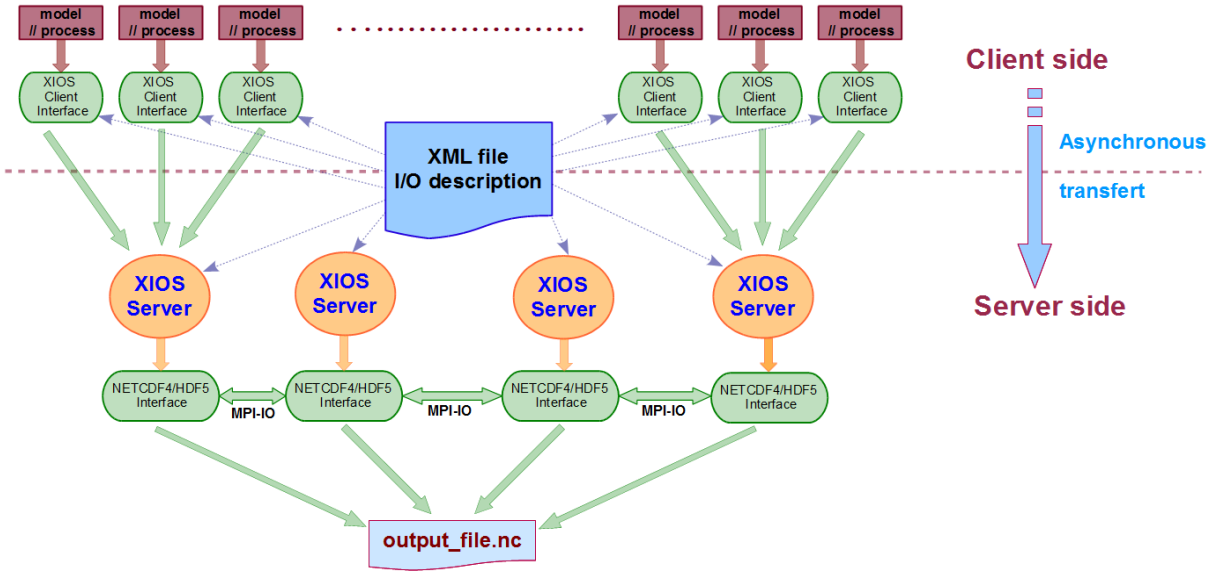

Figure 1 : XIOS clients / servers sketch

XML features

The XML file, written by the model user, contains the description of all fields to be written and their associations to different output files. As the XML file is parsed at runtime, changing an XML definition does not require a model modification or recompilation. We present here a brief description of the various features of XML definitions.

Elements

There are several types of tags "elements" used to define the properties of different objects handled:

- <context>: This element defines the context of outputs. The definitions within each context are independent from the definitions in other context, for example, enabling the separation of the definitions for two different models. Identical identifiers can be reused in different contexts.
- <axis>, <domain>, <grid>: These elements define respectively the vertical axes, the horizontal areas and grids that are associated with fields. Horizontal areas contain information on the global domain and on local domains. A 3D grid is defined by the combination of a horizontal area and a vertical axis. Currently, only regular longitude-latitude grid or curvilinear grids are supported. In both cases, indexed grids (for instance to write only land points) are also supported.
- <field>: This element, through its attributes, sets the properties of an output field. Values relating to the field id are output from the model. XIOS processes the values in accordance with the field request, and outputs the result to a file or to multiple files.
- <file>: This item defined an output file, with a given output frequency, containing the various fields that are included as child elements of the XML hierarchy.

```
<simulation>
  <context id="hello_word" calendar_type="Gregorian" start_date="2012-02-27 15:00:00">

    <axis_definition>
      <axis id="axis_A" value="1.0" size="1" />
    </axis_definition>

    <domain_definition>
      <domain id="domain_A" />
    </domain_definition>

    <grid_definition>
      <grid id="grid_A" domain_ref="domain_A"  axis_ref="axis_A" />
    </grid_definition>

    <field_definition >
      <field id="field_A"  operation="average" freq_op="1h" grid_ref="grid_A" />
    </field_definition>

    <file_definition type="one_file" output_freq="1d" enabled=".TRUE.">
      <file id="output" name="output_file">
        <field field_ref="field_A" />
      </file>
    </file_definition>

  </context>
</simulation>
```

Figure 2 : Minimalist XML file example: output the netcdf file "output_file.nc" contains daily averaging of the field "field_A"

Inheritance

Each family member is organized hierarchically; each child inherits the attributes of its parents, while having the opportunity to redefine the values of inherited attributes for its own use.

- root element (eg <field_definition>) : entry point definitions for an item type. May contain elements or groups of simple elements.
- Item group (eg <field_group>): allows the user to define attributes that will be transmitted to child elements. May contain groups of elements or single elements of the same family.
- Single element (eg <field>): Set an element, which inherits the attributes of its parents.

```
<field_definition level="1" prec="4" operation="average" enabled=".TRUE.">
  <field_group id="grid_W" domain_ref="grid_W">
    <field_group axis_ref="depthw">
      <field id="woce"       long_name="vertical velocity"           unit="m/s" />
   </field_group>
  </field_group>
</field_definition>

==>    <field id="woce" long_name="vertical velocity" unit="m/s" axis_ref="depthw"
         domain_ref="grid_W" level="1" prec="4" operation="average" enabled=".TRUE." />
```

Figure 3 : example of inheritance from parents elements to child

Management schedules and durations

XIOS supports different calendars: Gregorian, Julian, 360 days, 365 days, 366 days. Date format is (example): "2012-02-7 15:30: 00." XIOS also supports several time units that can be combined: year (y), months (mo), day (d), minute (mi) and second (s). Example: "1mo 1.5h 30s 2d".

Post-processing operation

XIOS can perform diagnostics such as time averaging and extremes over a given period. These operations are defined through the field attribute "operation" that allows the values "once, instant, average, maximum, minimum". Field values are extracted at each time step, averaged and then written to the file depending on the output frequency (file attribute "freq_output").

XIOS can also combine different field values using standard arithmetic operations to create new fields. These combinations also work on averaged fields.

All these operations usually attributed to post-processing are now made during the simulation and benefits from the parallelism of the models.

```
<field id="A" />
<field id="B" />
<field id="C" > (A + B) * exp (1. - A / B )    </field>
```

Figure 4 : example of a new field "C" created by combination of 2 fields "A" and "B"

Fortran interface

XIOS has an API that allows complementing of the XML definition within Fortran codes. The interoperability between Fortran and C + + is based on the Fortran 2003 that standardizes exchanges with C. Attributes of the different elements can be added, especially when their value is known only at runtime (resolution, details of longitudes, latitudes mesh, etc.).

```
CALL xios_set_field_attribut(id="toce",long_name="Temperature", unit="deg C", enabled=".TRUE.")
```

Figure 5 : example of a call setting attribute of a field element

The whole tree of the XML file can also be created or complemented from Fortran, adding groups of parents or child elements.

```
CALL xios_get_handle("field_definition", field_group_handle)
CALL xios_add_child(field_group_handle,field_handle,id="toce")
```

At each time step, the field values are transferred to XIOS through the Fortran interface. XIOS offers a minimalist interface that requires two arguments: a string and the address field of the table, such as:

```
CALL xios_send_field(id="toce",toce)
```

Client – server feature

XIOS has the ability to dedicate MPI processes (servers) to the I/O tasks. This option is activated dynamically by setting the parameter "using_server = true" in the parameter list. To provide this functionality, we chose to provide an independent binary process so as to preserve the independence of the model components and as a way to be less intrusive. So, the binary "xios_server.exe" must be started as MPMD with other binaries, as a component model. The main advantages of operating in server mode are, i) to collect data to be written on a small number of servers in order to reduce the number of requests to the file system, and avoid saturation. Ii) servers

are asynchronous : they operates the writing tasks in parallel to model processes, minimizing the impact on simulation time.

Interaction with model components

XIOS has been designed for use as part of a coupled model. Each component of the model, identified by a string, initializes the XIOS library through the Fortran interface. XIOS sets the global MPI communicator (MPI_COMM_WORLD) and sends back a new communicator to each model. XIOS can inter-operate with the OASIS coupler, which is considered as a component model. Then, at any time, each code can initialize one or more contexts. Each context will be associated with an inter-communicator enabling it to exchange messages with the "pool" of XIOS servers.

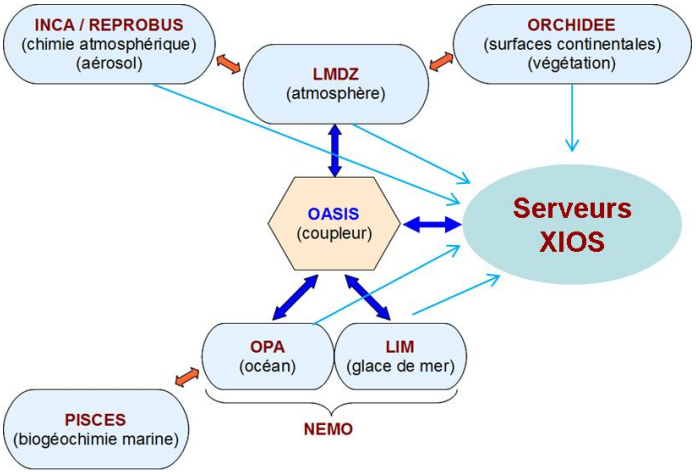

Figure 6 : interaction between the XIOS library, the OASIS coupler and the component models in the IPSL Earth system model.

Data distribution

On the client side, the field distribution of each process comes from the domain decomposition set by the model parallelization. On the server side we chose to distribute the data fields to optimize the disk operation. So data are organized as shown in Figure 7, i.e. contiguous in memory so as to have larger block to write on disk, and evenly distributed for a good load balancing. Each client sends its data to one or more servers, each server receiving data from one or more clients, while ensuring the transfer of asynchronous messages.

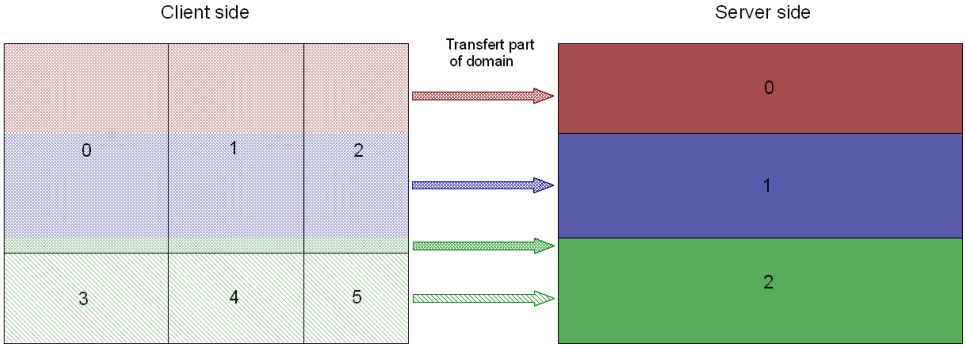

Figure 7 : data distribution between model domains of clients and servers. On the client side, data distribution is driven by the model and, on the server side, the distribution is driven by optimal disk operation, i.e. are evenly distributed and memory contiguous.

Transport layer

Traditionally, models require writing outputs at regular intervals (hourly outputs, daily, monthly ...). The pattern of access to the file system therefore presents very marked peaks following output frequencies. To smooth these peaks that periodically saturate the file system, XIOS buffers data sent to servers in order to distribute the writing operations uniformly in time throughout the simulation. The data are sent from clients to servers without synchronization; only calls MPI non-blocking point to point communications (MPI_Issend, MPI_IRecv MPI_Test, MPI_IProbe) are used when transferring messages. On the client side, a double buffer mechanism is used: while messages are being transferred from the first buffer, it stores the data in the second buffer, then interchange roles of the two buffers once the data from the first buffer is completely transmitted.

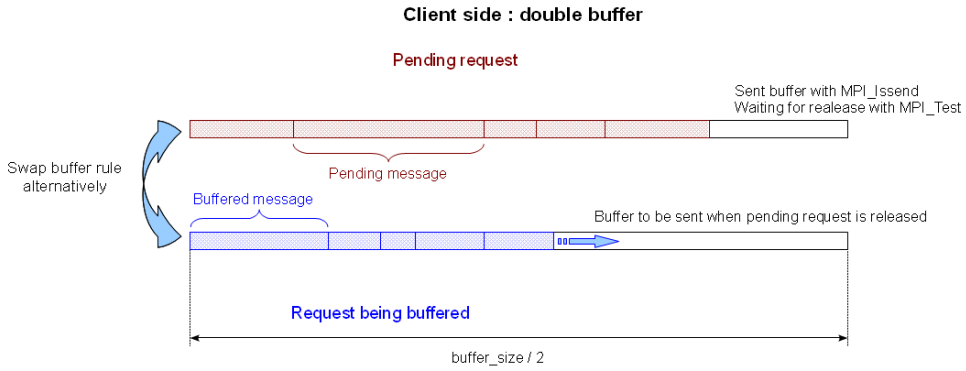

Figure 8 : client side buffer management

On the server side, circular buffers are used. MPI transfers data from the client to the server. Then while the server is processing this data and writing it to disk, further transfers of data can be received from the client.

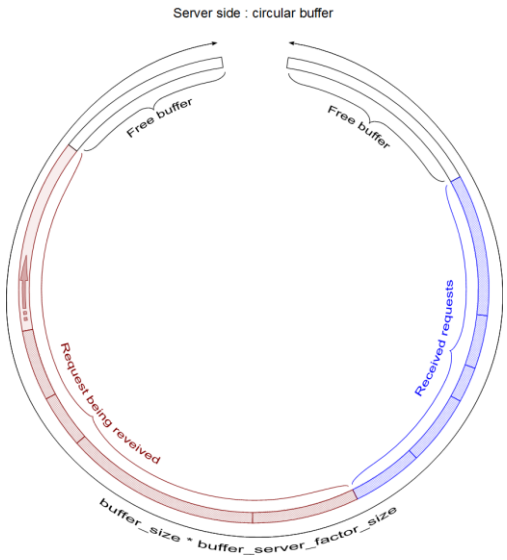

Figure 9 : server side buffer management

If the data stream sent is greater than the processing capacity of a server, the buffer fills up first on the server side. Once full, it will no longer accept data from the clients until some data has been processed, freeing some space in the buffer. On the client side, if the buffer becomes full, XIOS

enters a blocking mode, and will only stop blocking requests when some space is released in the buffers. In this case it is necessary to add servers to ensure the concurrency of writing and calculation. A set of diagnosis allows the user to properly set the number of servers.

I/O layer

The format of the output files XIOS currently relies on NETCDF4 format that builds itself on HDF5 format. However, the output layer in XIOS is highly modular, and another output format can be easily implemented. There are two output modes. In mode "multiple_file", each server writes the data of its own domain. The use of a library of parallel writing is therefore not required but a phase of post-processing is needed to rebuild the global file. The mode "one_file" uses parallel versions of HDF5 libraries and NETCDF4 (based on MPI-IO) to simultaneously write the output to a unique file, deleting the rebuild phase. The aggregated bandwidth of the file system is less effective than when writing files independently as in "multiple_file".

Reference implementation and results

The NEMO consortium (6 laboratories in France and Europe) has officially adopted XIOS. XIOS will be part of the next official release 3.5 of NEMO, currently in beta version. The integration of XIOS in all components of the IPSL Earth system model IPSL (NEMO, LMDZ, ORCHIDEE, INCA) has begun and will take effect in mid-2013. CNRM (Météo-France) has also initiated a collaboration to bring XIOS in all components of their Earth system model (NEMO, ARPEGE, GELATO, SURFLEX) (IS-ENES 2 EU, CONVERGENCE French ANR). Finally, the regional atmospheric model MAR (developed by LGGE, Grenoble) uses XIOS to manage its outputs.

To validate the client / server functionalities and performances, XIOS has been extensively tested during the PULSATION Project (http://www.locean- ipsl.upmc.fr/ ~pulsation/anr/welcome.html). This project sets up a coupling between NEMO (for the ocean) and WRF (for the atmosphere) at a very high resolution in the equatorial zone (9km), with nested models. This project received a grant of 22 million hours in the call for PRACE Tier-0 on machine. This allowed us to test the performance of NEMO-XIOS on very high resolution (1/12th ° overall) – see Figure 10:

- Tier0 Curie (1.6 PFlops) Lustre file system (150 Gbyte/s peek- theorically)
- Gyre 144, 4000x3000x31 grid on 8160 cores
- 6 days simulated, i.e. 2880 time steps: ~ 300 s
- Hourly outputs: 1.1 Tera byte for 6 days simulated, in 3 files

The configuration chosen for testing is voluntarily ambitious and constitutes an extreme case in terms of output volume. In fact, such a simulation produces 312 Tb per day, or 9.7 Tb per month.

In mode "multiple_file", i.e. a file server XIOS with 128 servers, scalability is perfect up to more than 8000 cores and allows even a super-linear regime due to cache effects.

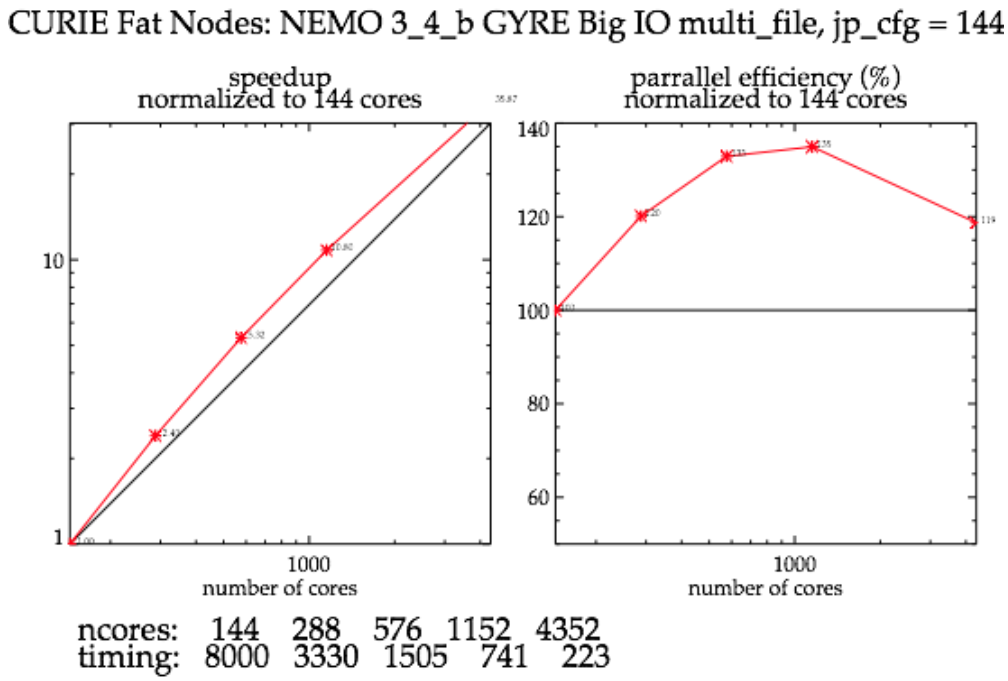

Figure 10 : Performance data obtained on the Curie supercomputer

In mode "one_file" (a single global file), the main challenge is to aggregate enough bandwidth from the parallel file system to have a writing time less than the time of calculation. On the Lustre file system, and using netcdf4 in parallel, we were able to aggregate approximately 5 Gb / s.

Depending on different output frequencies, we obtain the following results:

- 8160 NEMO,16 XIOS, daily output: : + 1.5% due to I/O
- 8160 NEMO, XIOS 16, 6 hours outputs : + 5% due to I/O
- 8160 NEMO, 128 XIOS, hourly outputs : + 15% due to I/O

For six-hour and daily outputs, the impact of I/O is less than or close to the normal fluctuations of the calculation time of the machine. Analysis of test cases with hourly outputs shows that we are very close to the maximum bandwidth obtained in simple NetCDF parallel outputs benchmarks. In addition, the file system shares the same network bandwidth as MPI communications, so we suspect an impact of file operations on the model communications. Nevertheless, a 15% impact on the computation time is quite reasonable compared to the amount of data written in this extreme scenario.

This study of extreme cases shows that XIOS is a viable solution for managing large amounts of data for future releases of high-resolution Earth system models.

CDI-pio

Thomas Jahns and Luis Kornblueh: DKRZ

1 Introduction

Based on the analyses of parallel computing systems conducted within the IS-ENES project and prior work, a number of challenges for current and potential future I/O patterns used in parallel climate simulation codes (referred to as model later) were identified.

A common architecture for parallel I/O implementations is a functional decomposition, where some processes compute (parts of) the simulation data (compute tasks), while others are tasked with handling I/O (I/O servers). This kind of architecture avoids implementation issues that occur, when too many processes open files simultaneously and access very small amounts of data. Since the optimal data decompositions for I/O and computation are usually very different, it also offloads the necessary transposition from the compute tasks, freeing up time to progress the simulation. And last, since data usually needs some conversion before writing, this can also be removed from the main loop of compute tasks. Since the number of I/O servers can typically be determined at program start, this also allows for tuning of an unchanged simulation binary relative to the level of detail of written simulation data (e.g. using fewer I/O server processes, when only monthly and more, if daily or even shorter interval value snapshots are required).

CDI-pio's design realizes the above transposition while attending to additional requirements and retaining close compatibility with the established CDI API (**C**limate **D**ata **I**nterface). CDI so far supported multi-thread, single-process semantics only, i.e. multi-threaded operations are possible, but MPI-I/O is not.

The current state of CDI-pio is an evolution of work begun by Deike Kleberg and Luis Kornblueh in the context of the ScalES project at Max-Planck-Institute for Meteorology in Hamburg.

2 The design of CDI-pio

2.1 Requirements

A survey of current practices and promising technologies available in modern HPC systems led to the following list of requirements:

- Data must be written in formats suitable for long term archival:
  - File formats need to be well standardized.
  - Data must be compressed for economical storage.
- The computation must progress in balance.[1]

[1] Load-imbalance is too broad a topic to cover here, therefore it must suffice to state that it has to be avoided when possible.

- Hence the computation must not be disturbed by the communication with I/O.
- Parallel writing must match the operating system software requirements. This especially means data must be decomposed in a fashion matching the file layout.

2.2 Solutions in CDI-pio

CDI-pio supports the NetCDF and GRIB1 file formats, GRIB2 and several legacy formats currently supported by the serial CDI API are to be ported later.

The preferred data format is WMO GRIB standard, a well-documented, very concise, record-structured data format.

GRIB features lossy lowest entropy data subsampling, a compression scheme which occurs in two stages: lossy entropy based reduction of data (i.e. only a specified number of significant binary digits is preserved) and subsequent lossless entropy compression following the method of Rice[2].

NetCDF is supported in all possible installation variants (single task, MPI parallel, classic and HDF5 disk formats) but must rely on site-specific NetCDF-tuning for performance (e.g. must be linked to an I/O extension library matching the parallel filesystem).

Data is transferred to I/O server processes via RDMA. See Figure 11. This mechanism has two benefits:

1. Congestion of the communication system is avoided as data is transferred to I/O servers only as fast as the servers can receive it.
2. RDMA can potentially avoid disturbance of the CPUs serving compute tasks, because the transfer can be offloaded to the communication network hardware.

[2] R. F. Rice (1971) and R. Plaunt, , "Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data, " IEEE Transactions on Communications, vol. 16(9), pp. 889–897, Dec. 1971.

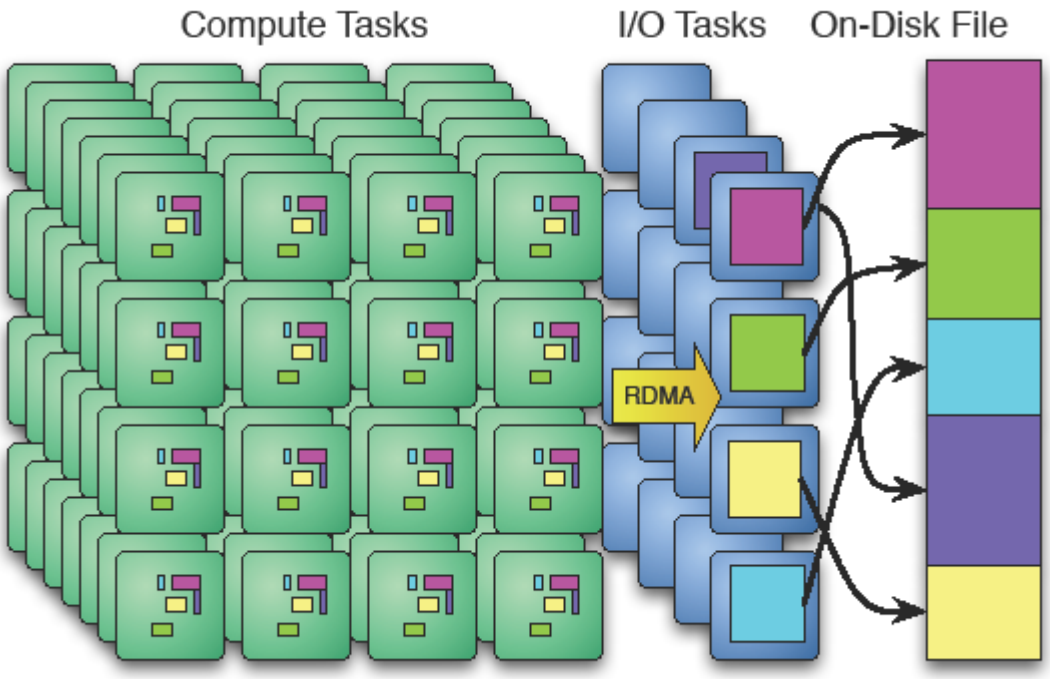

Figure 11: Data transfer in a parallel program employing CDI-pio I/O server tasks for output. The data decomposed into small chunks on the compute tasks is gathered by variable on I/O tasks with one task gathering one or more variables. All I/O then collectively write to the resulting on-disk file.

3 State of implementation

CDI is a C library with a thin Fortran interface layer (i.e. without dependence on the Fortran run-time library). All the CDI-pio additions required for parallel I/O were implemented in C. The corresponding extensions to the Fortran interface functions were introduced in a manner that retains compatibility with the interface for single-task programs, which is under simultaneous active development. Hence CDI-pio can be seamlessly used in C/C++ and Fortran programs, with little migration effort required for programs already adapted for the CDI API.

CDI-pio solves the transposition problem described in the introduction via the generic YAXT[3] data redistribution library and thus supports arbitrary data decompositions on the compute tasks. Since data prepared for output on the model tasks is only copied to a local RDMA-enabled buffer at first, no extraneous communication or synchronization and consequently no loss in parallel efficiency is introduced. Currently the relevant long-term archive formats NetCDF and GRIB are supported, where parallel I/O for NetCDF files is contingent on the provision of corresponding NetCDF libraries (parallel NetCDF for classic XDR-based formats and parallel HDF5 + parallel NetCDF 4.x for the more recent HDF5-based format). Support for parallel I/O with legacy formats available via single-task CDI is pending.

[3] YAXT is a library developed at DKRZ based on work from the ScalES project. See https://www.dkrz.de/redmine/projects/yaxt/ for developer access and https://redmine.dkrz.de/doc/yaxt/html/index.html for documentation.

4 Results

Measurements of run-time were (among others) taken for ECHAM6, running for 1 year of simulation, with a T127L95 grid on DKRZ blizzard using 32 IBM p575 nodes in ST mode with 16 MPI tasks per node and 2 OpenMP threads per task. For the decomposition[4] with nproca=32 and nprocb=16, nproma=72 was chosen.

The removal of work from the main loop of the compute tasks results in an almost total reduction of time spent on waiting on I/O by compute tasks as shown in figure 12. The total run-time is reduced from 12875 seconds to 8173 s and while computation time is slightly increased from 7500 s to 8053 s (supposedly because of IBM PE requiring some CPU action for RDMA to occur, but some increase is to be expected because of increased memory congestion caused by interfering RDMA access) time spent in the output routines shrinks almost 45-fold from 5375 s to 120 s.

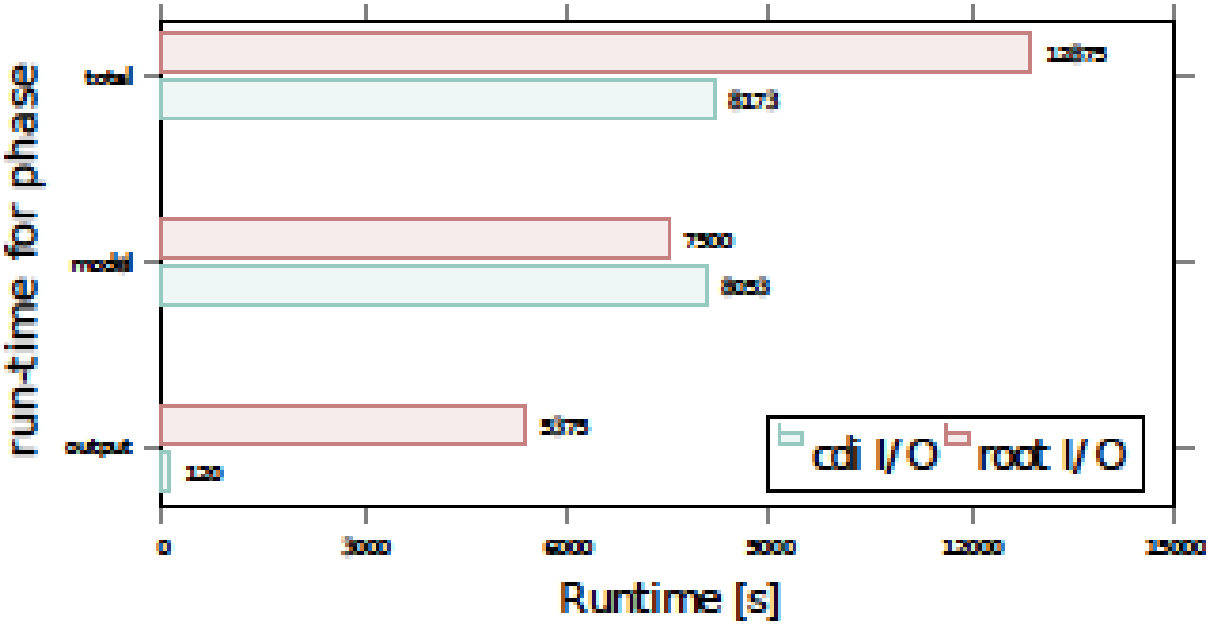

Figure 12: Employing parallel I/O via CDI-pio significantly reduces total run-time by virtue of almost complete elimination of waiting on I/O as previously witnessed for single-task root I/O for ECHAM6 (see text for specifics of configuration)

5 Applying CDI-pio

CDI-pio is part of recent CDI distributions (versions 1.5.6 and up) available for download from the CDI site.[5]

The documentation in the CDI C/Fortran manuals at:

https://code.zmaw.de/projects/cdi/documents

[4] nproc[ab] [ab]-division of earth, nproma: internal vector length; See ECHAM6 user's guide for description of technical settings.

[5] https://code.zmaw.de/projects/cdi/files

shows how to program the core data model of CDI. The CDI-pio manual also details necessary extensions to perform parallel I/O.

Users not yet using CDI would add, for example, calls to declare output meta-data (like grid sizes and types), variables to write, and file open/write just like is currently done for serial output with CDI. CDI-pio fits within the SPMD model; therefore the application must initiate parallel I/O with a call (pioInit) to declare the number of tasks to use as I/O servers.

Later running requires no changes to pre-existing workflows involving the adapted model with the possible exception to use more tasks then.

First attempts are underway to include CDI-pio in other models of the European community. The Irish Centre for High-End Computing is trying to introduce first the serial CDI interface into EC-Earth. In a second step this should be extended to the parallel version, which requires a few additional calls only.

A prototype version of ECHAM6 that uses a preliminary version of CDI-pio exists. Efforts to port all of MPI-ESM (of which ECHAM is a component model) and the ICON model to the most recent version of CDI-pio for inclusion into the released versions are on-going.