WP7/JRA1
Deliverable 7.3 – Reference implementations of Parallel I/O and of I/O Server

Abstract:

Climate simulation and weather prediction models have to write out field data for later analysis regularly. The amount of data for forecast and for high resolution climate models is very significant. The common method for writing this data was (and partially still is) to collect all data on a single processor of the model and write it to hard disk from there in serial. For today's high resolution climate models writing in serial is a huge bottleneck as while the data is being written the model cannot progress its simulation. Strategies to reduce or remove the bottleneck are by convention referred to either as parallel I/O or as I/O server techniques. Parallel I/O typically involves writing out data in parallel from each processor, either to one file or to many files, which reduces the time to write each field. I/O server techniques involve gathering data to one or more processors that are not involved in running the model, which means the model can continue while data is being packed, processed and written to disk. In a workshop on this issue that was organised in the context of the IS-ENES project[1] several groups presented their developments for the I/O problem. Their results seem to be very promising. As part of IS-ENES deliverable D7.3, a workshop was held to discuss developments done within the IS-ENES project as well as developments done by others. An overview on these solutions, their results and a discussion on the general approach is given here.

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 Parallel I/O and of I/O Server			**Deliverable**	D 7.3
Document Id N°:		**Version:**	1	**Date:**	March 4th 2013
Status:	Final				
Filename:					
Project Classification:	Public				

Document Authors	
Moritz Hanke	DKRZ (4)
Joachim Biercamp	DKRZ (4)
Carlos Osuna Escamilla	Eidgenössische Technische Hochschule Zürich
Thomas Jahns	DKRZ (4)
Deike Kleberg	Max Planck Institute for Meteorology
Paul Selwood	METOFFICE (10)
Steve Mullerworth (Executive	METOFFICE (10)

[1] https://is.enes.org/

Executive Summary

This document describes delivery of D7.3 of the IS-ENES project: *Reference implementations of Parallel I/O and of I/O server.* The main document is a summary of work on parallel I/O and I/O server techniques described at the IS-ENES IO workshop which was held in February 2012 at DKRZ, Hamburg. The workshop included discussions of work from both IS-ENES contributors as well as from the wider community. The contributions of the IS-ENES project to this document include results from prototype studies with CDI-pio undertaken at DKRZ (4), and development of the XIOS server undertaken at CNRS-IPSL (1). This document also includes discussions of approaches being developed outside of IS-ENES, including PIO (which is a different application from CDI-pio), the COSMO I/O solution, and the Met Office I/O server.

Introduction

Writing intermediate data to hard disk for later analysis poses a problem for today's earth system climate and weather prediction models. If high write frequencies (e.g. every simulated hour) and high grid resolutions (e.g. 1*10^6 points) are being used huge amounts of data are generated. The common method that collects all data on a single process, which then writes the data in serial to disk, can consume a significant part of the overall runtime. Meanwhile, all other processes are idle waiting for the single process to finish writing.

One solution to alleviate the problem is to use an I/O library like NetCDF4 [2] or PIO[3] that supports parallel writing. With these, all processes can concurrently write their part of the data to disk. It removes the need for collecting the data on a single node. In theory, the parallel writing should allow a good utilisation of the I/O bandwidth of the computing system. Experience of various groups shows that it is very hard to tune a system such that a high utilisation of the available bandwidth is achieved. In addition, the model still needs some time to write out the data. For a more detailed introduction to parallel I/O please see "Parallel I/O – Review on information on techniques for parallel I/O in High Performance Computing"[4].

A better solution is to use a few processes that are dedicated to writing out to hard disk. On today's massively parallel systems where the computing power of the overall system is generated by the huge number of CPUs instead of by a few very powerful CPUs, it is no problem to use a few CPUs for special tasks like I/O. These dedicated processes can write the data to disk, while the rest of the processes can continue their computation simultaneously. Theoretically, this approach should hide the latency of the writing completely from the rest of the processes.

In the next section this solution is described in general. This is followed by a presentation of a range of IO solutions: CDI-pio from MPI-M, XIOS from CNRM-IPSL, The Met Office IO server, the COSMO IO system and PIO.

I/O server

The concept of dedicating a few processes for doing the writing is commonly referred to as the I/O server approach. Figure 1 shows a general overview on this concept.

[2] http://www.unidata.ucar.edu/software/netcdf/

[3] http://code.google.com/p/parallelio/

[4] https://is.enes.org/documents/ISENES_D7.1_ParallelIOReview.pdf/view

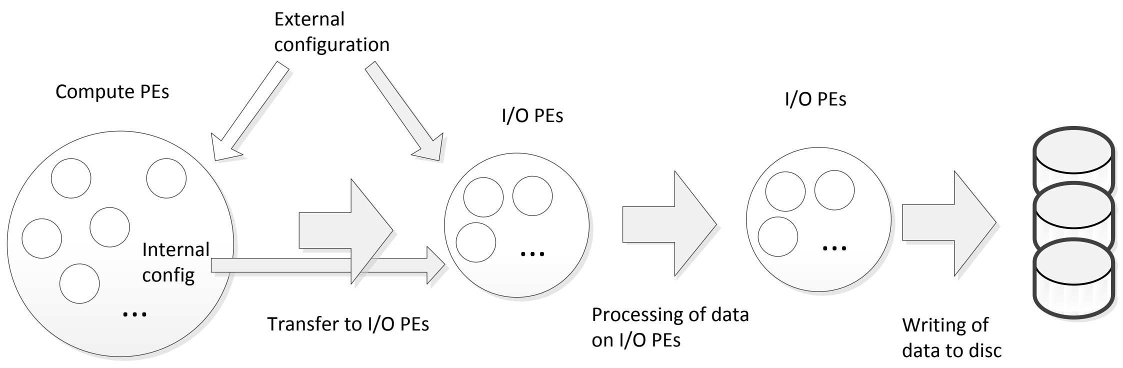

Figure 1: General overview on I/O servers

The I/O processes need to be initialised with metadata information on the data to be written to hard disk and on possible additional operations that are to be performed on this data. This metadata can contain information such as the number of fields to be written, the name and dimension of fields, the decomposition of data on the compute processes and information on additional operations like computation of mean, minimum or maximum values that can be performed on the data. This meta data can be hardcoded, be provided through an API (Application Programming Interface) by the computing processes and/or by external configuration files.

Once the I/O processes are configured and the compute processes have generated data that needs to be written, this is transferred to the I/O processes. The usual method of transferring data in these kinds of applications is through the Message Passing Interface (MPI). MPI provides multiple methods for transferring this data. These transfer operations can be synchronous or asynchronous, one- or two-sided (only one or both sides are directly involved in the transfer).

There are multiple ways to distribute the data on the I/O processes. All data of a single field or file can be aggregated on a single process or distributed among the processes.

When the data of a field is collected on the I/O processes, they can carry out post-processing on the data, for example computing min/max or mean values. In addition, the data might have to be converted into a certain data format before writing. Compression of the data is also possible and useful since it allows a better utilisation of the I/O system bandwidth.

Finally, the data can be written to disk based on the metadata provided before. This can again be done using different methods. A single I/O process could collect the data and write it in serial. In case the fields are distributed among the I/O processes all process could write their part in parallel to the same file or different files. If each process has the complete data from one or more whole fields it can write its data to an individual file or to one that is shared with the other I/O processes.

I/O developments

CDI-pio

CDI[5] (Climate Data Interface) is an I/O library that is being developed at the Max Planck Institute for Meteorology. It supports serial writing using various data formats. Based on this, CDI-pio is currently being developed. It extends the CDI library with I/O server capabilities.

CDI-pio and PIO (described later in this document) are two different and independent applications.

CDI-pio is configured using the CDI API, which is extended by a small set of additional routines.

In order to reduce the impact of writing on the overall run time as much as possible, CDI-PIO uses a one-sided communication scheme to transfer the data from the compute processes to the I/O processes. For this, the models copy the data that needs to be written into a special buffer, which supports RDMA (remote direct memory access). The I/O processes can then collect the data using the MPI_Win_get operation. Depending on the MPI implementation and underlying hardware this get operation can be implemented such that the transfer can be done without interfering with the computing resource of the compute processes.

The data fields of the simulation, typically in 3D, are distributed such that each field is assigned to a single I/O process. The I/O processes have to collect the data of the fields assigned to them, from all compute processes before writing them to hard disk.

The current implementation of CDI-pio only supports writing of GRIB[6] formatted files. GRIB uses domain specific compression methods, which are applied level-wise. Therefore, each process has to convert its levels into this format. In addition to the intrinsic compression of the GRIB data format, CDI-PIO also supports the SZIP[7] compression. Once the data is converted accordingly, it can be written. Multiple methods for writing are available. Tests on an IBM Power6 system achieved the highest performance when each I/O process used POSIX I/O to write out its part of the data into a common file. In GRIB formatted files whole levels of fields (compressed or uncompressed) are always written into a contiguous block of memory within the file. Due to the characteristics of the compression algorithm compressed different levels of the same field can have different sizes in memory. Therefore, it not trivial to determine the start addresses for each level within the file. This problem is overcome by using a dedicated processor that manages the offsets for the levels in the file

XIOS

The XML-I/O-Server (XIOS) is an I/O library developed at the Institute Pierre Simon Laplace. It will initially be integrated into the NEMO ocean model and later integrated in other component models.

As the name suggests, XIOS uses XML files to do most of the configuration of the output data requests. This allows for changes to be made to the output of a run without the need to modify and

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

[6] http://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/Introduction_GRIB1-GRIB2.pdf

[7] http://public.ccsds.org/publications/archive/121x0b1c2.pdf

recompile the model code. Another advantage is that this simplifies the API that needs to be called from within the code. However, some parts of the configuration like the domain decomposition still need to be provided by the model itself in the initialisation process.

XIOS uses an RPC-like (Remote Procedure Call) approach for the communication of the computing processes with the I/O ones. Different types of messages are packed into uniform buffers on the computing processes and are then sent to the I/O processes using asynchronous send methods. On the I/O side the messages are unpacked and according to their content the respective actions are executed.

Data fields are evenly distributed among all I/O processes. The user can specify, via the external configuration files, that the I/O processes apply post-processing operations to the collected data. XIOS allows the user to choose between two different modes for writing to disk. One method is to write one file per I/O process. The performance of this method is good. However, after the run an additional post-processing step is required to merge these files. The second option is to write all I/O processes into the same file using NetCDF4.

A special feature of XIOS is the integrated support for the OASIS coupling software[8].

Met Office Unified Model I/O Server

The Met Office Unified Model (UM) is used for operational numerical weather forecasting as well as climate research. Its use as a forecast model adds further key requirements to the I/O solution on top of the climate requirements. Since the results of operational forecast simulations are time critical, the I/O implementation needs to be very robust; if a forecast needs to be re-run due to failures, the Met Office misses its targets for the customers that rely on the forecast’s outputs. In contrast to climate simulations where the amount of data being output is relatively constant throughout the run, the I/O solution in a forecast model has to deal with changing workloads over the time of the simulation: for example, it is common for data to be output more frequently near the beginning of the simulation. For climate jobs, as long as the I/O server keeps up with the model simulation, it is acceptable. But for a forecast model the earlier a file is written and closed the better, as the results in the file can be sent to the customer sooner.

Similar to other models, the Met Office Unified Model has a long history. Because of this, it uses a unique file format and can therefore not resort to I/O libraries like NetCDF. The I/O server implementation is tightly integrated into the model itself and can therefore not be used as a generic I/O library.

The configuration of the I/O is split into two separate parts; diagnostics to be performed on the I/O processes and restart files to be written on the one hand and the set up of the I/O servers like number of I/O processes and which I/O process writes which file on the other. Both parts are controlled via namelist input files. These configuration files are generated by tools with graphical user interfaces that simplify this task.

[8] http://www.cerfacs.fr/3-26568-OASIS.php

To minimise the latency impact of I/O operations on the compute processes, the Met Office uses asynchronous transfer methods for both: metadata and actual field data. To further improve the transfer, multiple messages are aggregated before sending and a double buffering schema for the messages is used. This reduces the overall latency, improves overall memory utilisation and allows for a better bandwidth utilisation.

A user of the UM selects the fields to output, the processing operations to be done on each field, and can also direct each output field to one of a multiple of files,. These output files are distributed among the I/O processes. Each of the I/O processes then uses serial I/O operations to write out the data. For example, if a user wants to drive one or more external models or wants to generate different datasets for two different customer's requirements, a set of diagnostics required for each purpose can be sent to a separate file dedicated for that purpose.

To ensure asynchronous behaviour of the I/O processes, the implementation uses two threads per I/O process. One thread takes the role of a "Listener" that communicates with the compute processes and receives the data. When all data has been received, it passes the data to the "Writer" thread, which does the actual writing to hard disk.

I/O solution developed for COSMO climate model

For the COSMO regional climate model an integrated I/O server solution was developed by CSCS/ETH that successfully improved the overall performance of the model significantly.

Since the I/O solution is an integrated part of the model, the I/O processes can directly take part in the initialisation phase of the model. This information (e.g. number of time steps, variable definitions, etc.) is used for the configuration of the I/O processes.

The compute processes pack metadata and field data into a single message and send it to an I/O process using an asynchronous transfer mechanism. The I/O processes process the received data based on the metadata packed into the message only.

The climate models discretise the simulation into time steps, which typically have a constant length. In COSMO all fields written to hard disk in a single time step get written into one file. Different time steps have different files. All data for a time step is collected on a single process, which then writes out the data in a serial manner using the NetCDF library. This makes writing itself easy and does not require difficult tuning of the system. If an I/O process has not finished its write operation when another write request of a new time step arrives, another I/O process is selected to process the data of that time step. Evidently, this is only possible if there are idling I/O processes.

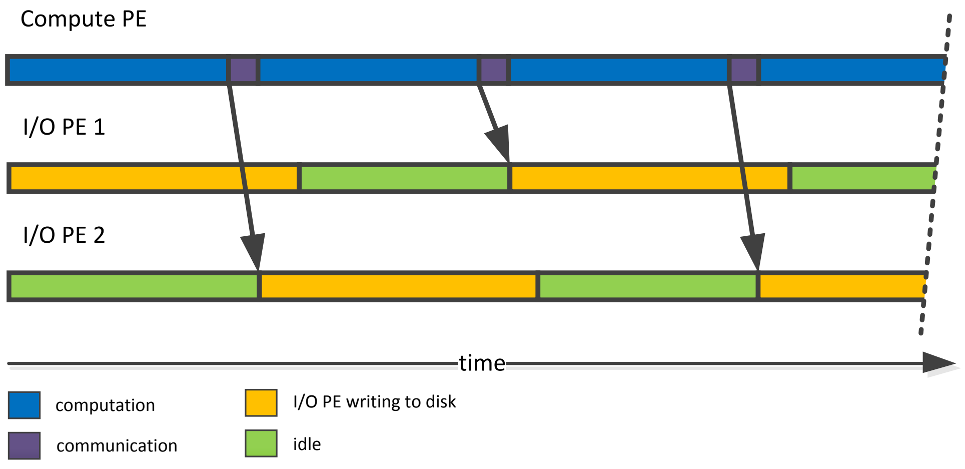

Figure 2: Workflow in COSMO

PIO

The Parallel I/O (PIO) library developed in the US for the Community Climate System Model (CCSM) is a meta-I/O library (Dennis et al., 2012[9]). It does not implement its own I/O operations but uses other I/O libraries for this instead. One of its main tasks is to provide a simple interface for the user to a number of backend libraries (MPI-IO, NetCDF3, NetCDF4, pNetCDF and NetCDF+VDC). In contrast to the other solutions presented above it does not have dedicated processes for I/O that to not take part in the computation of the model.

Depending on the system hardware, the optimal I/O pattern can vary significantly. However, the data decomposition within the models is mostly fixed. PIO allows the use of independent computational and I/O decompositions. To support this, it provides the necessary mechanisms to automatically redistribute the data according to the two decompositions before writing the data to the file. The redistribution is done on the computed processes. This allows tuning the I/O decomposition for the respective system without the need to change the model itself.

[9] John M. Dennis, Jim Edwards, Ray Loy, Robert Jacob, Arthur A. Mirin, Anthony P. Craig and Mariana Vertenstein, An application-level parallel I/O library for Earth system models, International Journal of High Performance Computing Applications 2012 26: 43

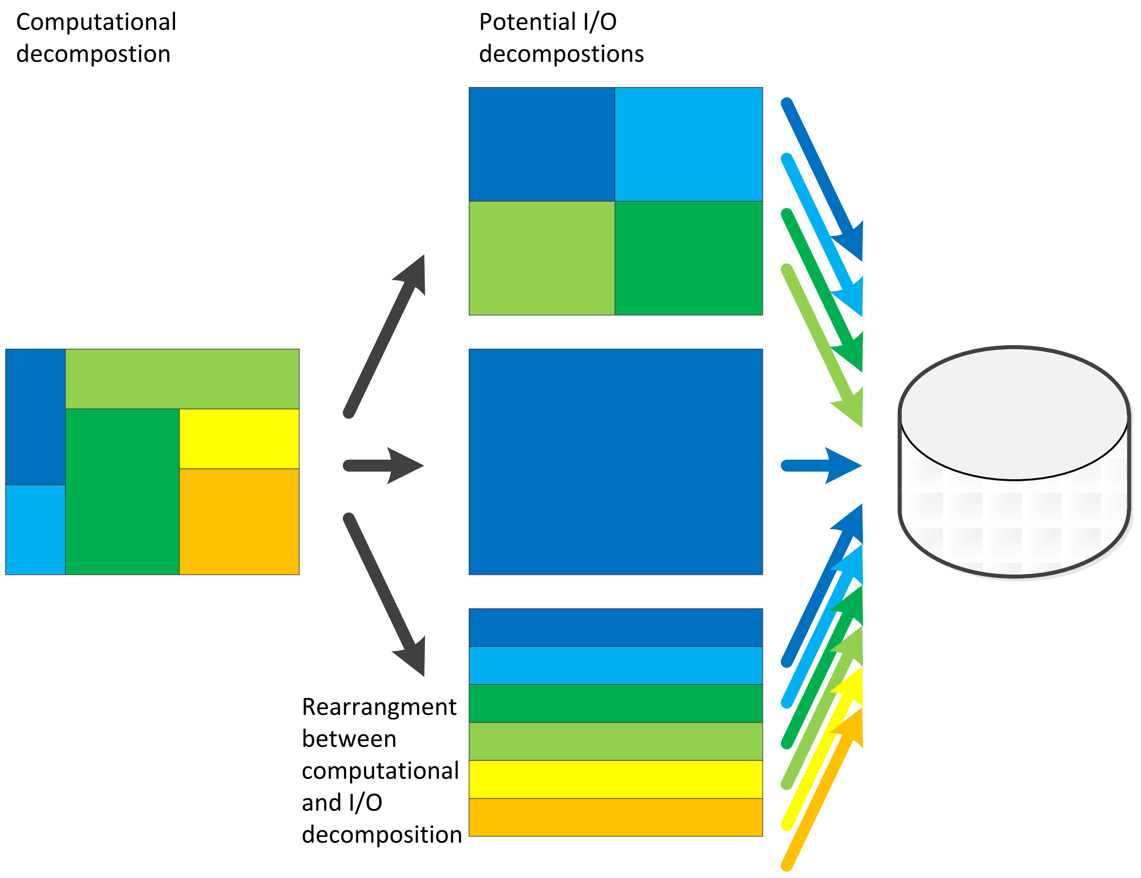
Computational decompostion
Potential I/O decompostions
Rearrangment between computational and I/O decomposition

Figure 3: Separation between computational and I/O decomposition in PIO (the different colours represent the computed processes on which the respective data resides)

Discussion

All presented I/O server solutions except for PIO follow the same general concept presented in section “I/O server”.

XIOS and the Met Office I/O server use external configuration files. In both cases it makes changing of the output configuration easier and possibly faster, because the model does not have to be recompiled, as only external files have to be adjusted. On the other hand, it might increase complexity of the I/O library as it adds one or more files that need to be provided and maintained for the model run.

Splitting the communication between compute and I/O processes into metadata and field data is the most common approach between all solutions. Only the I/O solution in COSMO is an exception to this. In addition it is the only implementation in which the I/O process handling the actual field data is not known a priori, because in case a process is still busy processing the data of a previous time step, the writing request might be forwarded to another I/O process. Therefore, in contrast to the other I/O implementations, it requires a synchronous communication scheme for the selection of the responsible I/O process. This can be a potential problem in case of load balancing issues on the computing processes. Using a round robin approach to assign different time steps to the I/O processes could avoid this problem.

Since the transfer of field data to the I/O process can be time consuming due to the size of the data, asynchronous transfer methods are used for this. Here one can distinguish two ways. One is the use of asynchronous send operations, as is done in XIOS. This allows the simulation to continue computing while the data is being sent in the background. However, depending on the system hardware, the send might still require some CPU time, which can impair the performance of the simulation. On the other hand, one can use one-sided RDMA transfers, as implemented in CDI-pio. Once the I/O processes know that new data that needs to be written to disk is available on the computing processes, they can execute a get operation. This is only possible if the data on the computing processes is in a special part of the memory that is accessible through RDMA. Theoretically, this makes it possible to transfer the data to the I/O processes without usage of the computing resources of the compute processes. Unfortunately, depending on the software and hardware of the respective system this transfer can be implemented differently using asynchronous send operations. Tuning of the software stack to enable real RDMA transfers can be difficult, for example the developers of CDI-PIO reported that user generated MPI data types were not supported by RDMA (IBM Power6 system running AIX). In summary, one-sided transfers have the potential to deliver the best performance, but asynchronous send operations seem easier to handle and can be completely sufficient.

An optimisation applied by the Met Office to improve the performance of the data transfer is the aggregation of messages before sending them to the I/O processes. This is done in order to reduce the overall latency and to improve the bandwidth utilisation.

The decomposition of data on the I/O processes depends on the application field of the respective I/O solution. Integrated solutions like the one from the Met Office and the one for COSMO have internal knowledge of the files that need to be written and can therefore easily parallelise over the different output streams (files). Currently they use serial writing for each output stream. This is much simpler to tune than a parallel write operation. Performance measurements done for the COSMO model even showed worse performance for parallel writing. But for serial writing the parallelisation of writing to hard disk is limited to the number of output streams, which can be a problem in case not enough output streams are available. However, other operations like packing of data can still run on more I/O processes and streams. The actual I/O bandwidth of a system typically limits scaling of writing to a very limited number of streams. Writing to too many output streams at once might even swamp the I/O subsystem of the system and therefore decrease writing performance. On machines with limited amounts of memory per node (like Blue Gene) it can be a problem to collect too much data per I/O process. Problems like this might be overcome by the introduction of additional caching processes, whose only task is to collect the data from the compute processes and cache it for the I/O processes. Research in this issue is currently conducted at the Irish Centre for High-End Computing in the frame of the PRACE[10] project. Its results are supposed to be integrated into CDI and XIOS.

General libraries like CDI-pio and XIOS cannot make assumptions about the number of output streams, which can potentially be only one. Therefore, they usually distribute the field data evenly among the I/O processes and use parallel writing operations. This can be done by usage of the NetCDF4 library, which is done by XIOS. Tuning of NetCDF4 however is difficult. It is highly system

[10] http://www.prace-project.eu

dependent and might hence be a problem when used on different computing sites. CDI-pio uses a file format that is not supported by any other parallel I/O library. Hence, they implemented their own writing mechanisms.

The asynchronous data transfer between the compute and I/O processes and the writing of the data to hard disk can consume parts of the network bandwidth. This, on the other hand, can have an effect on the performance of the simulation, for example if the model does a halo exchange. So, even though the data transfer might not directly have an impact on the runtime of the compute processes, it can indirectly decrease the performance of the model itself. Using hints for the I/O library in the model code that mark compute intensive sections that do not use the system network might make it possible to avoid this issue. But this again makes the implementation more complex.

If the writing to disk does not take as long as the computing processes required to generate new data that needs to be written, the I/O processes can be idle for some time. This idle time can be used to perform some post-processing operations such as is done in XIOS.

Conclusion

For many climate modelling groups, writing the intermediate results of their results to hard disk is a paramount issue, which limits the overall performance and scaling of their model. With future computing systems being most probably even more massively parallel than today's machines, this problem will become even more significant.

The use of the I/O server approach that moves the actual writing of data from the computing processes to dedicated I/O processes can be a solution for this. The implementations presented here and their results indicate that this is a feasible way. However, the development of such a solution is much more difficult than the use of a standard parallel I/O library like NetCDF4. While I/O servers have the potential to hide nearly all latency of the I/O operations, parallel I/O without I/O servers will always delay the models by the writing time.