Available on-line at www.prace-ri.eu
Partnership for Advanced Computing in Europe

Parallel IO performance and scalability study on the PRACE CURIE supercomputer.

Philippe Wautelet[a,*], Pierre Kestener[a]

[a]IDRIS-CNRS, Campus universitaire d'Orsay, rue John Von Neumann, Bâtiment 506, F-91403 Orsay, France
[b]CEA Saclay, DSM / Maison de la Simulation, centre de Saclay, F-91191 Gif-sur-Yvette, France

Abstract

The results of two kinds of parallel IO performance measurements on the CURIE supercomputer are presented in this report. In a first series of tests, we use the open source IOR benchmark to make a comparative study of the parallel reading and writing performances on the CURIE Lustre filesystem using different IO paradigms (POSIX, MPI-IO, HDF5 and Parallel-netCDF). The impact of the parallel mode (collective or independent) and of the MPI-IO hints on the performance is also studied. In a second series of tests, we use a well known scientific code in the HPC astrophysics community: RAMSES, which a grid-based hydrodynamics solver with adaptive mesh refinement (AMR). IDRIS added support for the 3 following parallel IO approaches: MPI-IO, HDF5 and Parallel-netCDF. They are compared to the traditional one file per MPI process approach. Results from the two series of tests (synthetic with IOR and more realistic with RAMSES) are compared. This study could serve as a good starting point for helping other application developers in improving parallel IO performance.

Project ID:

1. Introduction

Data accesses in large scale supercomputers tend to be a huge challenge for both scientists and engineers. Understanding and tuning parallel I/O are necessary to leverage aggregate communication and I/O bandwidth of client machines. Finding usable, efficient and portable programming interfaces and portable file formats are also of crucial importance. Ideally, the use of high-level I/O libraries (HDF5, PnetCDF...) or I/O middleware (MPI-IO) should reduce the need for optimizations in application codes. However, to obtain good I/O performance, it is absolutely essential to investigate how to set MPI-IO hints in an application code to make an appropriate use of the underlying I/O software stack (MPI-IO, parallel file system) on top of the actual hardware. More precisely, MPI-IO hints are parameters that help tuning the MPI-IO middleware for facilitating, by example, concurrent access by a group of processes (collective I/O, atomicity rules...) and efficiently mapping high-level operations into a dedicated parallel file system flavour (GPFS, Lustre...).

The PRACE CURIE supercomputer is the French Tier-0 system based on x86 architecture CPUs with a total 92,160 processing cores with 4 GB memory per core. At the time the present study was made, only 360 fat nodes (equipped with four 8-cores Intel Nehalem processor per node) were available; this amounts to 11,520 CPU cores. The site-wide global storage subsystem uses the LUSTRE parallel filesystem and provides an aggregate 5.4 PBytes storage with a peak I/O bandwidth of 150 GB/s. This LUSTRE file system is made of several sub-domains or logical volumes(e.g. SCRATCH, WORK and STORE) with different LUSTRE characteristics, goals and performance. For example, on the CURIE machine, the SCRATCH space is made of 120 OSTs (Object Based Devices), each of which contains a 6.9 TBytes space.

2. IOR benchmark

The IOR (Interleaved or Random) micro-benchmark [1] provides a flexible tool to perform reading/writing throughputs measurements by varying different parameters like access pattern (single file per MPI process or collective IO), transaction size, file size, concurrency rate and programming interface (POSIX, MPI-IO, HDF5 or

[*]Corresponding author.
tel. +33-1-69358576 fax. +33-1-69853775 e-mail. philippe.wautelet@idris.fr
[1]IOR was developped by Lawrence Livermore National Laboratory; source code available at http://sourceforge.net/projects/ior-sio/

Table 1. List of MPI-IO hints values used in the collective mode IOR study.

Configuration	MPI-IO Hints values			
number	romio_cb_write	romio_ds_write	romio_cb_read	romio_ds_read
1	enable	enable	enable	enable
2	enable	disable	enable	disable
3	enable	auto	enable	auto
4	disable	enable	disable	enable
5	disable	disable	disable	disable
6	disable	auto	disable	auto
7	auto	enable	auto	enable
8	auto	disable	auto	disable
9	auto	auto	auto	auto

ParallelNetCDF). As most other available IO benchmarks, IOR is good at tuning a specific operation, validating a newly installed system but might be not as useful when focusing on modeling or predicting application performance (using a real case application is the purpose of the second part of this study). IOR benchmarks consists in reading or writing files organized as following [2]. In the particular case of collective IO, IOR handles a single file with a specific layout consisting in a sequence of *segments*, which could correspond for example to a set of data variables (e.g. velocity, pressure, energy) associated to a simulation time step. Each processor (i.e. MPI process) handles a *segment* called *blockSize* which is actually read or written from disk by chunks of size *transferSize* which directly correspond to the I/O transaction size. In the one-file-per-process mode, each MPI processor performs I/O operations on its own file.

Building IOR is straighforward on the CURIE machine, but just requires to install high-level parallel IO libraries: HDF5 (version 1.8.7) and ParallelNetCDF (version 1.1.1). As an exhaustive study was impracticable because there are too many parameters that can be tuned and also by lack of time, we limited ourself to the following setup: we used a fixed transfer size of 2MiB, and a block size of 1024 MiB per process. A typical IOR input file would be:

```
IOR START
  api=HDF5
  testFile=fichier
  repetitions=1
  readFile=1
  writeFile=1
  filePerProc=0
  keepFile=0
  blockSize=1024M
  transferSize=2M
  verbose=0
  numTasks=0
  collective=1
IOR STOP
```

where the important parameters to our study will be the API (POSIX, MPI-IO, HDF5 or ParallelNetCDF) and `collective` (to activate operation mode where all processes coordinates to read/write from/to the same file). Finally, let us mention that IOR implements a simple mecanism which allows the user to set MPI-IO hints at runtime by parsing environment variables whose name must have the following prefix `IOR_HINT__MPI__` and as suffix the name of a valid MPI-IO hint. For example we will use `IOR_HINT__MPI__romio_cb_write=enable` to activate MPI-IO hint `romio_cb_write` (collective buffer in writing mode).

Table 1 specifies the different configurations used in the collective mode IOR study by varying the values of four different MPI-IO hints controlling data sieving and collective buffering. Data sieving is an optimization technique used for efficiently accessing noncontiguous regions of data in files when noncontiguous accesses are not provided as a file system primitive [3]. Collective buffering is another optimization technique which consists in using a two-stage approach; for example, in the case of a reading operation, data buffers are first split up amongst a set of aggregator processes that will then actually perform I/O operations through filesystem calls.

In table 2 are shown some reading and writing throughputs mesured in MiB/s for all the MPI-IO hints specified in table 1. Please note that these measurements where repeated only five times; we noticed variations in measurements of the order of 10 to 20% and once a drop of performance maybe due to a high load on the machine (all the tests were performed in normal production). From the table 2, one can make the following comments:

- Changing the MPI-IO hint configuration has a large impact on throughput: a factor of 24 can be seen by comparing HDF5 reading throughput of configuration 4 to configuration 1 and a factor of 7 in HDF5 writing throughput;

Table 2. IOR Reading/Writing throughputs in MiB/s measured with different MPI-IO configurations listed in table 1 using 256 MPI processes/cores, each reading/writing a block of 1GiB with a transfert size of 2MiB. In configuration 0, IOR is used in a non-collective mode while all other configurations correspond to a collective mode.

Config number	POSIX		HDF5		MPI-IO		PnetCDF	
	R	W	R	W	R	W	R	W
0 (non-collective)	235396	10057	5299	938	59887	1102	57593	959
1	-	-	252	192	664	363	851	265
2	-	-	257	191	569	363	849	264
3	-	-	340	182	230	363	627	285
4	-	-	6083	931	3867	984	9412	867
5	-	-	3779	1148	4151	1005	6150	1328
6	-	-	5251	891	4671	999	12722	847
7	-	-	269	176	170	361	354	269
8	-	-	357	177	407	348	467	267
9	-	-	369	172	404	337	475	273

Table 3. IOR Reading/Writing throughputs in MiB/s measured with two MPI-IO configurations listed in table 1 using 1024 MPI processes/cores, each reading/writing a block of 256MiB with a transfer size of 2MiB. In configuration 0, IOR is used in a non-collective mode while the other configuration correspond to a collective mode.

Config number	POSIX		HDF5		MPI-IO		PnetCDF	
	R	W	R	W	R	W	R	W
0 (non-collective)	610370	16310	51789	3660	43984	3841	49114	2732
6	-	-	37717	1610	17768	3890	14939	1466

- Regarding reading/writing performances, configurations 4, 5 and 6 provide the best throughputs. This means that in this particular case of quite large files (256GiB, each process accessing 1GiB), disabling collective buffering for both reading and writing is a good option. This is clearly not true when each process handles smaller block sizes;
- Regarding API impact on performance, one can notice the overall performance of PnetCDF is better than HDF5 by roughly a factor of 1.5;
- The best writing performance is obtained using PnetCDF API in configuration 5; that is by disabling both data sieving and collective buffering.

Table 3 shows some mesurements using 1024 processes for the same total file size of 256GiB in the shared file mode. Please notice that, we were not able to run all configurations when using 1024 processes by lack of time and also because we observed that those jobs have a tendency to terminate prematurely. We notice that we have a much higher writing throughput by using 1024 processes for the same total file size (256 GiB) in shared file mode. We can also observe a much narrower gap in reading throughput between collective and non-collective IO. This is a non-exhaustive study, that should be repeated and enlarged to other MPI-IO hints configurations, and towards larger numbers of processes.

3. RAMSES benchmark

In this section, we ran I/O performance tests on a real application. Our choice was to use the astrophysics code RAMSES. It is well known in this community and has proven to be scalable to tens of thousands of processes. Moreover, different parallel I/O approaches (MPI-IO, HDF5 and Parallel-netCDF) have been implemented by IDRIS in the previous years allowing to compare them on the CURIE supercomputer.

3.1. Presentation of RAMSES

RAMSES[4] is an astrophysics application originally developed by Romain Teyssier (CEA) under the CeCILL software license. Its usage is free for non-commercial use only.

It is available at http://web.me.com/romain.teyssier/Site/RAMSES.html.

RAMSES contains various algorithms designed for:

- Cartesian AMR (Adaptive mesh refinement) grids in 1D, 2D or 3D with load balancing and dynamic memory defragmentation;
- Solving the Poisson equation with a Multi-grid and a Conjugate Gradient solver;
- Using various Riemann solvers (Lax-Friedrich, HLLC, exact) for adiabatic gas dynamics;
- Computing collision-less particles (dark matter and stars) dynamics using a PM code;

Table 4. Size in GiB of the output files (for HDF5 on 1024 cores)

File	1024^3	2048^3
AMR	25.860	191.813
hydro	52.848	393.246

Table 5. MPI-IO hints tested with RAMSES

Hint	Possible values (default in bold)
direct_read	**false**, true
direct_write	**false**, true
romio_cb_read	**automatic**, enable, disable
romio_cb_write	**automatic**, enable, disable
romio_ds_read	**automatic**, enable, disable
romio_ds_write	**automatic**, enable, disable
romio_lustre_ds_in_coll	**enable**, disable

- Computing the cooling and heating of a metal-rich plasma due to atomic physics processes and an homogeneous UV background (Haardt and Madau model);
- Implementing a model of star-formation based on a standard Schmidt law with the traditional set of parameters;
- Implementing a model of supernovae-driven winds based on a local Sedov blast wave solution.

The version used in this paper has been modified by IDRIS. The three following parallel IO approaches have been implemented: MPI-IO, HDF5 and Parallel-netCDF. They complement the traditional "one file per process" approach, that we will call the POSIX approach.

3.2. Testcases and run conditions

The RAMSES testcase we used in all the tests is Sedov3D (explosion in a cubic box). Several domain sizes have been chosen (1024^3 or 2048^3). There is no AMR during execution (fixed mesh size) and the load is perfectly balanced between the MPI processes. Each run generate two series of files:

- *AMR* files (one per process for the POSIX approach, one shared for all processes for the MPI-IO, HDF5 or Parallel-netCDF approaches);
- *hydro* files (one per process for the POSIX approach, one shared for all processes for the MPI-IO, HDF5 or Parallel-netCDF approaches).

Their main differences are:

- the *AMR* files are more complex and contains more variables;
- the first 2 sections (containing respectively parameters identical on all processes and small datasets) are much bigger in the *AMR* files (but still relatively small in absolute size);
- the *hydro* files are about 2 times larger (320 bytes per cell instead of 156 for the third and main section);
- the granularity of the entries is significantly larger in the *hydro* files and therefore better I/O performances are expected for them.

Typical file sizes are given in the Table 4. Small variations exist depending on the fileformat (POSIX separated files, MPI-IO, HDF5 or Parallel-netCDF) and on the number of processes (some data structures have size proportional to the number of MPI processes).

All runs have been done on CURIE in normal production. In addition, due to lack of time and computing resources, most measurements have been done only one time. Therefore, perturbations in the measured performances are likely.

3.3. Effect of the MPI-IO hints

MPI-IO hints can have a dramatic effect on the I/O performances (see [5] by example). Default system values are not always the most adapted ones to the application and can give very poor results. It is therefore strongly recommanded to test several of them on realistic testcases (in terms of problem size and number of processes).

The MPI-IO hints tested with RAMSES on the Lustre filesystem of CURIE are given in the Table 5. 7 hints of the 23 available on the system have been benchmarked.

We chose the Sedov3D testcase with a 1024^3 mesh. All runs were done on 256 cores.

The obtained results are shown in the Table 6 for the writing tests and in the Table 7 for the reading tests. It can be seen that the most adequate MPI-IO hints for this testcase and with this number of processes are the default ones for writing and *romio_cb_read = disable* for reading. These values are used in the scalability tests of the following sections. Different combinations of *romio_cb_read = disable* with other hints have been tested but have not given improved performances.

Table 6. Throughput in writing with different MPI-IO hints (in MiB/s, Sedov3D 1024^3 on 256 cores)

Hint	Value	POSIX		HDF5		MPI-IO		PnetCDF	
		AMR	hydro	AMR	hydro	AMR	hydro	AMR	hydro
default	-	1154	2627	187	402	273	1113	230	943
romio_cb_write	enable	-	-	183	390	269	1090	245	955
romio_cb_write	disable	-	-	37	373	181	701	59	479
romio_ds_write	enable	-	-	185	391	259	828	235	770
romio_ds_write	disable	-	-	183	391	193	421	182	391
romio_lustre_ds_in_coll	disable	-	-	181	393	193	420	181	399
direct_write	true	-	-	147	238	157	286	137	255

Table 7. Throughput in reading with different MPI-IO hints (in MiB/s, Sedov3D 1024^3 on 256 cores)

Hint	Value	POSIX		HDF5		MPI-IO		PnetCDF	
		AMR	hydro	AMR	hydro	AMR	hydro	AMR	hydro
default	-	6511	7507	378	977	412	946	385	793
romio_cb_read	enable	-	-	367	942	451	991	436	815
romio_cb_read	disable	-	-	1718	1611	2142	1572	1922	1670
romio_ds_read	enable	-	-	437	969	432	1002	342	815
romio_ds_read	disable	-	-	414	986	430	1017	424	820
romio_lustre_ds_in_coll	disable	-	-	434	996	454	986	451	805
direct_read	true	-	-	152	203	164	209	255	361

3.4. Scalability tests

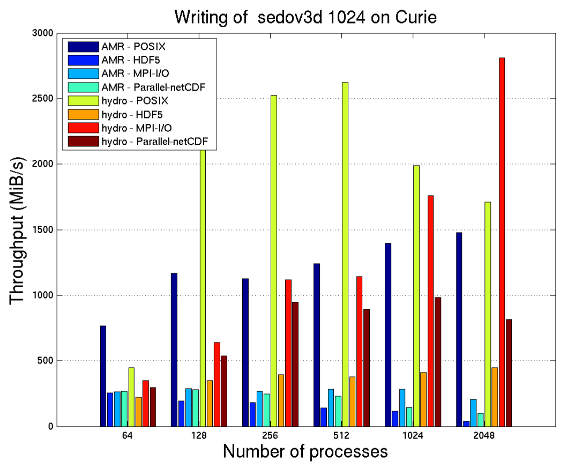
Once the MPI-IO hints were chosen, scalability benchmarks have been realized with 2 different problem sizes: 1024^3 and 2048^3. They were run from the minimum possible number of processes (due to memory limitations) to a relatively high number of cores (2048 for the 1024^3 testcase and 4096 for the 2048^3 testcase). The Parallel-netCDF approach was not used with the biggest testcase due to known problems with the use of it in RAMSES and with these filesizes.

The figures Fig. 1 and Fig. 2 show the throughputs obtained with the different parallel I/O approaches and with different numbers of processes.

The POSIX separate files approach is nearly always the fastest. Even with a very big testcase, the peak throughput of the file system can not be approached (less than 20GiB/s in the best case compared to a theoretical maximum throughput of 150GiB/s). Throughput increazes with the number of process up to around 1024 processes, stabilizes up to 2048 and seems to decreaze with more processes.

For the three other approaches (MPI-IO, HDF5 and Parallel-netCDF), MPI-I/O is usually the most efficient. Its throughput increazes with the number of processes, except when writing the *AMR* file. With a high number of cores, its performance become comparable to the POSIX approach.

Parallel-netCDF is close to (and slighly slower than) MPI-I/O up to 512 processes. This seems logical because the overlay on MPI-I/O is not very important. However, when the number of processes becomes



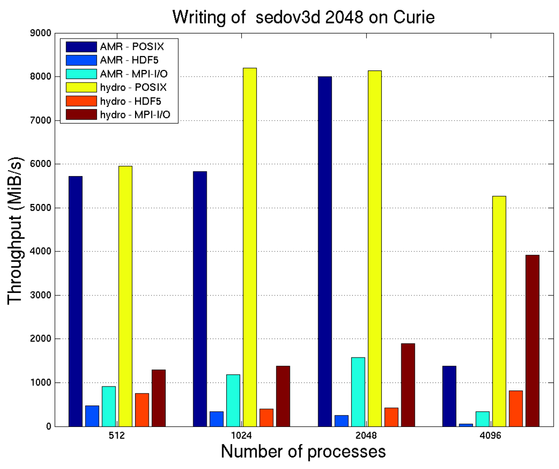


Fig. 1. Writing throughput of Sedov3D on CURIE for different I/O approaches; (a) 1024^3 meshsize; (b) 2048^3 meshsize

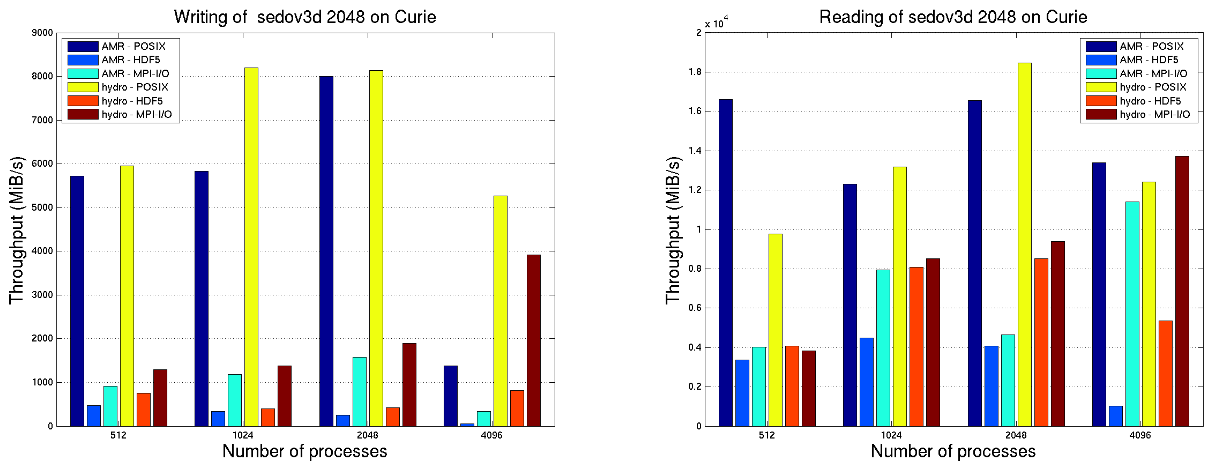


Fig. 2. Reading throughput of Sedov3D on CURIE for different I/O approaches; (a) 1024^3 meshsize; (b) 2048^3 meshsize

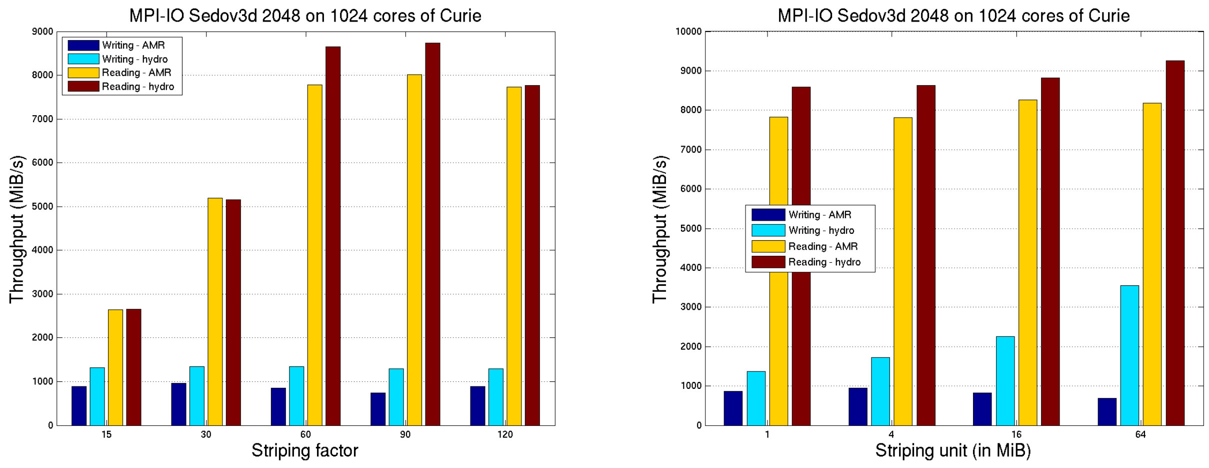


Fig. 3. Throughput of Sedov3D 1024^3 on CURIE for different striping parameters; (a) striping_factor; (b) striping_unit

important, the gap increazes dramatically.

The HDF5 approach is the slowest. This is probably due to the fact that the HDF5 interface is more complex. The throughput for the *hydro* file is problematic and decreazes when there are more and more cores.

It can also be noted than, for a fixed-size problem, the differences between the different paradigms tend to increase with the number of processes.

3.5. Additional tests: effects of the striping

How the files are positioned on the disks can have a big influence on the performance. On the Lustre filesystem, this can be controlled in two different ways: by setting the number of OSTs (Object Storage Targets, which can be seen as different "disks") on which a file is spanned and by defining the maximum size of each block for one OST (by example, if set to 1MiB, a file of 4.5Mio will be separated in 5 blocks).

These two parameters can be set by using the Lustre command *lfs* when creating the output directory (all files in a directory inherit by default of its properties) or by using the following MPI-IO hints: *striping_factor* and *striping_unit*.

For these tests, we chose to run the Sedov3D testcase with a resolution of 2048^3 on 1024 cores. Only the MPI-IO approach was tested. The POSIX approach has no sense here because the created files are relatively small and therefore won't be striped on a lot of OSTs. The HDF5 and Parallel-netCDF fileformats should give conclusions close to the MPI-IO one because they are based on it.

The obtained results are shown on the figure Fig. 3. We can observe that the *striping_factor* (the number of used disks) has an impact on the reading performance. If it is too small, it is reduced. From a certain level (here between 30 and 60), there are no more gains. The effect of this parameter on the writing performance seems negligible. However, it is likely that a very small value of the *striping_factor* would lead to a reduced writing throughput.

If we set it to the maximum value (120 that corresponds to the number of OSTs of the studied filesystem), the performance degrade slightly. This is probably a consequence of the shared ressources between all the concurrent running jobs.

The filesystem blocksize (set with the *striping_unit* parameter) has a strong influence on the writing performance. Depending on the filetype, the effects differ. For the *AMR* file, a value of 4MiB seems optimal (for this testcase). For the *hydro* file, if the value of the *striping_unit* increazes, the performances too. The granularity of the I/O operations being smaller in the *AMR* file than in the *hydro* file (a factor 8 for the big data structures), the optimal value is probably shifted of this same factor. Increazing further the *striping_unit* would very probably degrade the throughput of writing the *hydro* file.

For reading, the *striping_unit* parameter shows no influence. However, low values of it could have a negative impact.

In conclusion, we saw that the way the file are striped can have a big impact on the performances. New scalability tests with different values of them would be interesting. These tests were not conducted here by lack of computational resources and human time.

4. Conclusion

In this study, we performed two kinds of parallel I/O mesurements on the Tier-0 CURIE supercomputer. In a first series of tests, by using the IOR micro-benchmarks we have studied the impact of API (MPI-IO, HDF5, PnetCDF) and of MPI-IO hints on measured reading and writing throughputs. We have shown that in the particular case of a quite large shared file of size 256 GiB, it is best to disable both data sieving and collective buffering MPI-IO optimization techniques. In a second series of tests, we used a real scientific application from the astrophysics community (code RAMSES) where the I/O pattern is much more complex due to the use of an adaptive mesh refinement algorithm. Theses two series of tests demonstrate that for a high number of cores (espacially in the RAMSES case), MPI-IO performances become comparable to the POSIX approach (one file per process). This last observation is less true for HDF5 and PNetCDF. Finally, we study the impact of some LUSTRE filesystem parameters. First, the *striping_factor* should be large enough, up to the number of LUSTRE OSTs, to have a good reading bandwidth, and then the *striping_unit* size is demonstrated to be important to the writing performance.

This study, despite being far from exhaustive, should be considered as a starting point to a better understanding of tuning parallel IO on current PRACE Tier-0 supercomputers.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resources CURIE at TGCC, France.

References

1. IOR benchmark source code repository, http://ior-sio.sourceforge.net
2. H. Shan and J. Shalf, "Using IOR to analyze the I/O Performance for HPC Platforms", Cray User Group Conference 2007, Seattle, WA, May7-10, 2007.
3. Romio User's Guide, http://www.mcs.anl.gov/research/projects/romio/doc/users-guide
4. R. Teyssier, Cosmological hydrodynamics with Adaptive Mesh Refinement: a new high-resolution code called RAMSES, Astronomy and Astrophysics, 2002, volume 385, page 337.
5. P. Wautelet, "Parallel I/O experiments on massively parallel computers", ScicomP 17 conference, Paris, May 2011, http://www.idris.fr/docs/docu/IDRIS/IDRIS_io_scicomp2011.pdf