Analyzing the I/O Scalability of a Parallel Particle-in-Cell Code

Sandra Mendez(✉), Nicolay J. Hammer, and Anupam Karmakar

High Performance Systems Division, Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities, 85748 Garching bei München, Germany
sandra.mendez@lrz.de

Abstract. Understanding the I/O behavior of parallel applications is fundamental both to optimize and propose tuning strategies for improving the I/O performance. In this paper we present the outcome of an I/O optimization project carried out for the parallel astrophysical Plasma Physics application ACRONYM, a well-tested particle-in-cell code for astrophysical simulations. ACRONYM is used on several different supercomputers in combination with the HDF5 library, providing the output in form of self-describing files. To address the project, we did a characterization of the main parallel I/O sub-system operated at LRZ. Afterwards we have applied two different strategies that improve the initial performance, providing a solution with scalable I/O. The results obtained show that the total application time is 4.5x faster than the original version for the best case.

1 Introduction

The Leibniz Supercomputing Centre (LRZ) operates a Top50 HPC system, SuperMUC [1] accessible for users in Germany and Europe. SuperMUC has a total peak performance of 6.8 Petaflops, 500 Terabyte main memory, 20 Petabyte external data storage, and a high speed Infiniband interconnect. LRZ has a strong focus on user support, in order to enable users to efficiently use all compute resources offered. As the installation of SuperMUC has increased LRZ's compute capacities, users require more efforts for the parallel I/O of their applications. Therefore, we provide support for parallel I/O optimization to enable high I/O scalability of applications that requires to analyze the application I/O characteristics and its interaction with the I/O system. In the case of SuperMUC, an additional complexity arises from the large diversity of scientific applications which are actively used on the system.

The authors thank the ACRONYM developer team for their contributions and the good teamwork. Special thanks goes to Gerald Mathias for reading the manuscript and providing valuable feedback. Computations for this project were done on SuperMUC at LRZ, a member of the Gauss Centre for Supercomputing (GCS).

R. Yokota et al. (Eds.): ISC 2018 Workshops, LNCS 11203, pp. 9–22, 2018.
https://doi.org/10.1007/978-3-030-02465-9_1

Since 2016 LRZ has build up structures to focus on domain specific community-oriented support and research infrastructure, to strengthen its support commitment. These so called *Application Labs* take the lead in supporting scientific communities in the following research areas: Astrophysics and Plasma Physics (AstroLab), Biology and Life Sciences (BioLab), Computational Fluid Dynamics (CFDLab) and Geosciences (GeoLab). Aside from first level support offered by the LRZ application support for technical problems with I/O implementations in scientific applications, the *Application Labs* offer project based high level support for tuning, optimization and refactoring I/O implementations for user applications.

In this paper, we present the work done during a project in the 2nd LRZ AstroLab Support Call. Its objective was to improve the I/O scalability of the astrophysical particle-in-cell Plasma Physics code Acronym [2], where scaling problems were observed for a larger number MPI tasks ($>10^4$, one SuperMUC's Island). It is a typical example for implementation related problems reported by SuperMUC users and how these problems can be overcome and can be seen as a blueprint for tackling and improving similar problems for other applications as well. Similar challenges were tackled in [3] using a different approach for the parallel Particle-in-Cell application VPIC.

The project consisted of three parts: (1) performance characterization of the I/O sub-system of SuperMUC with respect to I/O parameters at application user level; (2) Access pattern analysis of Acronym, to gain understanding of the application behavior and (3) Refactoring of Acronym's I/O implementation to enable data aggregation. First and second parts are the basis to explain the limitations of Acronym's I/O capabilities and to develop data aggregation strategies for performance improvement. We create I/O communicators for aggregation based on the application access pattern and system characterization. An own implementation for aggregation is proposed because the Two-Phase I/O technique reported poor performance in our GPFS file system and it does not support all two-phase I/O hints provided by ROMIO [4]. Tessier et al. [5] describes the limitations of two-phase I/O related with the problem of mapping aggregators in the topology and the data access pattern of the application.

The paper is organized as follows: in Sect. 2, we present a study of the I/O capabilities of the two SuperMUC GPFS file systems, Sect. 3 describes the methodology for analyzing the I/O scalability of parallel applications and Sect. 4 reviews the experimental evaluation. Finally, we present our conclusions in Sect. 5.

2 Characterization of the I/O System

In this section, we present the throughput evaluation of the SuperMUC file systems. Our aim is to evaluate the I/O performance behavior of the two file system under normal operation. We define two kinds of experiments for:

– *Throughput Evaluation as a Function of Request Sizes*: we evaluate request sizes which are `aligned/non-aligned` with file system block size (See Fig. 1). This decision is based on the experience that data sizes per MPI task in scientific applications are in most cases of arbitrary size and cannot be expressed as 2^n bytes. We use request sizes of $2^n \times 32$ kiB with $n \in (1..16)$ for `aligned` tests. For `non-aligned`, we use these `(1.5 GiB, 729 MiB, 243 MiB, 81 MiB, 27 MiB, 9 MiB, 3 MiB, 1.4 MiB, 459kiB, 153kiB, 51kiB, 17kiB)` request sizes.

– *Throughput Evaluation as a Function of the Number of Nodes*: we evaluate I/O aggregation using 32 to 2048 compute nodes for three request sizes: (a) `6.8 MiB`, a request sizes slightly below the file system block size; (b) `13.6 MiB`, a request sizes slightly above the file system block size and (c) `2 GiB`, as maximum request size since the MPI I/O implementation based on 4 byte integers.

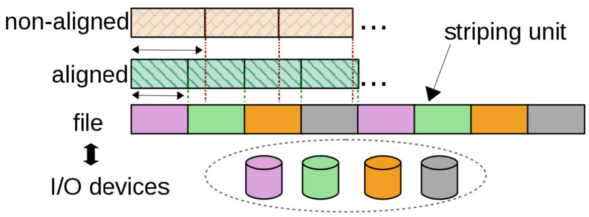


Fig. 1. Representation of request sizes `aligned`/`non-aligned` with the file system block size

Table 1 shows the compute system and I/O system description used for the experiments. Each measurement was rerun three times. We used an own benchmark based on MPI-I/O, in which every MPI task writes/reads its data consecutively in one block. The I/O tests was implemented in FORTRAN using MPI (IBM Parallel Environment 1.4.04). This benchmark has a long usage history on our system, which makes it easy to analyze the measured results. In several comparison tests our benchmark delivered consistent results with Livermore's IOR (see https://github.com/hpc/ior).

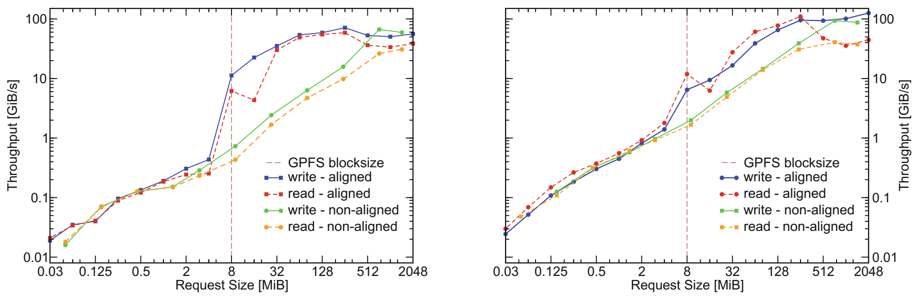
2.1 Throughput Evaluation as a Function of Request Sizes

In Fig. 2 we present the MPI-I/O throughput for the two available parallel filesystems of SuperMUC as a function of the request size. The measurements was done during user operation of SuperMUC, which may result in a "noisy" measurement due to I/O of other applications.

As can be observed, a general trend is the strong continuous increase of the throughput with growing request size. This is expected, since every I/O systems has a maximum number of IOPS, which naturally leads to smaller throughput at small request sizes and a higher throughput at larger request sizes.

Table 1. SuperMUC supercomputer

Compute system	Description	
Number of nodes	9216	
Nodes per Island	512	
Sockets per node	2	
Cores per node	16	
Memory per node (GByte)	32 (Usable 26)	
Communication network	FDR10 IB	
Intra-Island topology	Non-blocking tree	
Inter-Island topology	Pruned tree 4:1	
I/O system	`WORK`	`SCRATCH`
Parallel filesystem	IBM spectrum scale	
Network shared disk (NSD)	80 (DDN based)	16 (GSS based)
Stripe/block size	8 MiB	8 MiB
Filesystem capacity	12 PiB	5.2 PiB
Max. I/O performance		
Write(GiB/sec)	≈180	≈130
Read(GiB/sec)	≈200	≈150
Compute node	≈4.5 GiB/sec	



(a) `SCRATCH` file space (b) `WORK` file space

Fig. 2. Throughput of our MPI-I/O benchmark as a function of request size. The job was executed on 512 compute nodes of the SuperMUC sandy bridge system with 1 MPI task per node. There are two cases shown, one (blue/red) for `aligned` requests and a second one (yellow/green) for `non-aligned`. Each point is the average of 3 independent measurements (Color figure online).

Discussion. For analyzing results, we sub-divided the graph in three regions, which mark different characteristics of the I/O sub-systems. We refer to the first region at the left end with limit in 4 MiB, as region of `small scale I/O`. Here the request sizes are smaller than the GPFS blocksize. To the third region at the right end, we refer as region of `large scale I/O` with request sizes larger than 512 MiB. To the second region located in between, we refer as `intermediate` region.

In the `large scale I/O` region, the throughput is given by the design specifications of the I/O sub-systems. Compared to that, in `small scale I/O` region the throughput is significantly smaller, because the filesystem was tuned for large throughput using large files and therefore a large blocksize is required and the IOPS limits of the I/O sub-system. In this region the `WORK` throughput is clearly higher than on the `SCRATCH`. The difference in throughput directly below the blocksize (8 MiB) is approximately a factor of 5 (`WORK` compared to `SCRATCH`). We associate this behavior with the larger number of NSD servers and the larger number of controller machines available on the I/O sub-system of `WORK`, compared to the `SCRATCH`. We therefore conclude that codes making I/O in that region, should rather use the `WORK` filesystem.

In the `intermediate` region, the throughput is strongly dependent on alignment with filesystem blocksize. If the data size can be expressed as an integer multiple of the blocksize, the throughput is high and, with increasing request size, it approaches the system maximum quite fast. If not, the throughput is significantly lower and straightly connects the `small` and `large` scale I/O regions. Only for request sizes in the order of GiB the throughput is on the same level for both cases. On both filesystems a dip in the read throughput curve at 16 MiB request size is visible which is connected to caching effects. However, currently we do not understand the root cause of this behavior.

2.2 Throughput Evaluation as a Function of the Number of Nodes

Fig. 3 shows the measured throughput as a function of the utilized number of nodes. The three measurements corresponds to request sizes slightly below and

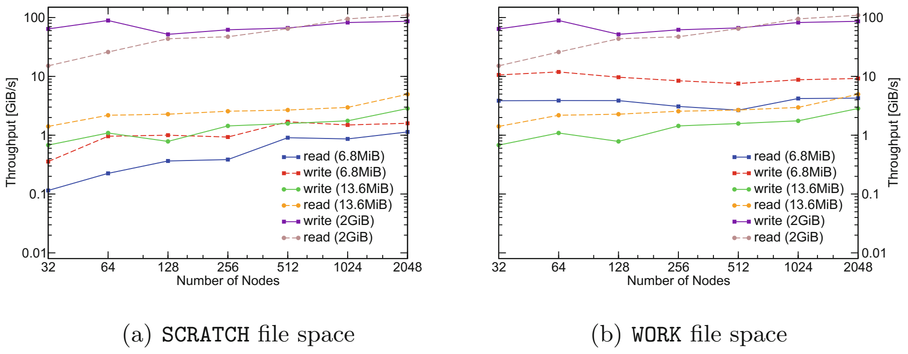


Fig. 3. Throughput of our MPI-I/O benchmark on SuperMUC as a function of the number of I/O nodes (i.e. number of compute nodes). The benchmark was executed on SuperMUC Sandybridge system partion with 2 MPI task per node. The plot shows the write and read performance for a request size of 6.8 MiB (blue/red), 13.6 MiB (green/yellow) and 2 GiB (purple/brown) per task, respectively. Each point is the avarage of 3 independent measurements (Color figure online).

above the GPFS block size and a maximum request size. The general trend of all three measurements is a moderate increase of the throughput with increasing number of nodes (moderate compared to the increase of throughput with request size).

Discussion. The system behaviour observed explains quantitatively why aggregation is useful under certain conditions to improve the I/O performance of HPC applications. As can be seen in Figs. 2 and 3, the throughput improves strongly increases with request size in the region below and around the block size of the SuperMUC filesystems. At the same time the throughput does not strongly decrease with decreasing number of I/O nodes being used. This is true as long as the used number of nodes is large enough, i.e. greater/equal 256 nodes. An example to illustrate it is provided by the write values from Figs. 2 and 3(b). To make a conservative estimation, we take the read/write performance for a request size not matching a 2^n value, because the data size per MPI task in a lot of applications is defined by a dynamic domain decomposition scheme and it does not correspond to 2^n bytes.

In Fig. 2(b) it can be seen that the throughput increases by roughly a factor of 20 when scaling from a request size near 2 MiB to near 64 MiB. If the number of I/O nodes is down-scaled by a factor of 32, e.g. from 1024 nodes to 64 nodes, the I/O performance drops only by roughly a factor of 2 to 3 as seen in Fig. 3(b), so we "gain" a factor of 8. This example is not precise in it's numbers, but rather demonstrates the basic principles behind I/O aggregation. Moreover, one must take into account that the aggregation traffic imposes additional overhead to the communication costs of the application. However, due to the rather large gain factors (in our example approximately $8\times$) the I/O aggregation is still beneficial.

3 Analyzing the Application's I/O Scalability

In Sect. 2, we have presented a system I/O characterization for different I/O request sizes and compute nodes in a normal operation of the SuperMUC. This shows to users an initial idea about the I/O performance behavior.

In this section, we describe the proposed methodology to analyze the I/O scalability of parallel applications that is composed by three steps : (1) I/O Pattern Analysis, (2) Evaluation of the weight of I/O operations; and (3) Evaluation of I/O Strategies.

3.1 I/O Pattern Analysis

We perform two steps to analyze the I/O pattern with focus on the scalability:

1. Extracting temporal/spatial patterns to identify the dominant I/O phases.
2. Identifying the relation between application parameters and I/O phases.

Acronym made use of parallel HDF5 library to regularly write three-dimensional field datasets using collective operations with all processes participating. The Acronym users reported slow performance when using more than 10k MPI processes on SuperMUC. Initial tests revealed runtimes increasing superlinearly with the number of MPI ranks suggesting a growing communication overhead. Moreover, the data volume written by each process was small, exacerbating the problem. To understand the I/O pattern that produce that behavior, we evaluated the Acronym's I/O kernel, named `IOSTEST`, by using different number of MPI processes to extract the main properties of the I/O operations to analyze the scalability behavior. Acronym was evaluated by using a global simulation size $(52, 52, 66560)$ that means 52 cells along the x and y-direction and 66560 cells along the z-direction; 10 simulation steps and 6 fields.

Extracting Temporal/Spatial Patterns to Identify the Dominant I/O Phases. To extract the I/O pattern we use PIOM-MP (former knows as PAS2P-IO [6,7]) a tool that isolates the influence of the underlying I/O system allowing it to obtain the patterns in I/O phases and focus on the I/O routines. PIOM-MP depicts the global I/O pattern of a parallel application in two dimensions: the spatial and temporal patterns. The temporal pattern represents the order which the I/O operations are performed by the MPI processes and the spatial pattern represents the file logical view for each MPI process. Furthermore, a third dimension is added to show the weight of the I/O operations.

In Fig. 4 it is depicted the global I/O pattern by using 320 (small case) and 5120 (medium case) MPI processes. We can observe the same pattern in both cases, where the red point represent the I/O operations with bigger requests. The 3D picture depicts in the x-axis the process rank, in the y-axis the ticks that corresponds to the communication and I/O events of MPI, and in z-axis the file offset for each process in each tick. A heat map depicts the request size of each I/O operation in the 3D picture. Read operations in blue corresponds to small requests less than 512 Bytes that are independent on the application parameters. Write operations are in red and their request size is variable depending on the number of processes. As can be seen in Fig. 4, there are ten phases of writing operations, in which red points correspond to larger requests.

Additional read and view operations are done at MPI level, because Acronym uses parallel HDF5 that is built on top of MPI. Count of read operations is mainly related to the metadata operations of a HDF5 file. Additionally, the following metadata complements the pattern provided in Fig. 4: (a) Access type is shared that means that a file is accessed by all writer processes. The number of writer processes is equal to the number of compute nodes; (b) Access mode is strided that means each writer process accesses to non-contiguous positions of the file; (c) Read operations are independent, blocking and use explicit offset; (d) Write operations with more data to move are collective, blocking and use explicit offset; (e) Two view operations are called when a field is written to the file.

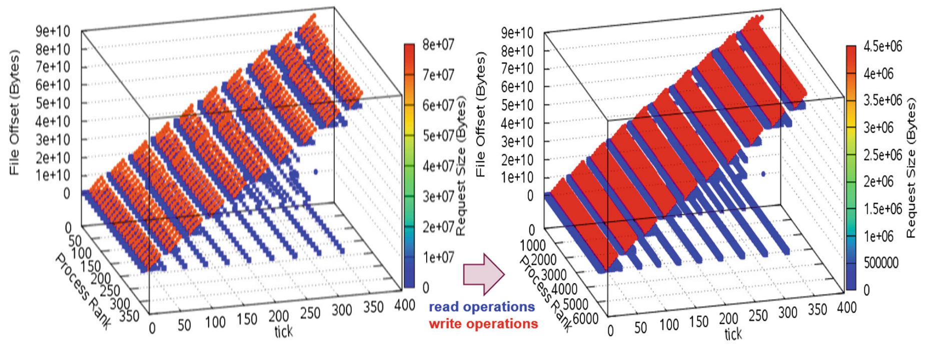


Fig. 4. Global I/O pattern of the `Acronym's IOTEST` at MPI-IO level using 320 (left) and 5120 (right) MPI processes. Write and read operations are represented in a 3D picture. x-axis corresponds to the MPI rank, y-axis represents calls to MPI-IO operations and z-axis represents the offset in the file for each MPI process. A heat map depicts the request size of each I/O operation. Read operations are represented in blue and write in red. Plots obtained with PIOM-MP [6,7] (Color figure online)

Identifying the Relation Between Application Parameters and I/O Phases. Focusing on a simulation step, we analyze its call tree that is shown in Table 2. Using the information of the global pattern and call tree we identify and define relation between the parameters of the application and the I/O pattern properties.

Table 2. Call tree for a simulation step

Order	MPI-I/O operation	Data access aspect
1	`MPI_File_open`	
2	Once only by rank 0 `MPI_File_get_size`	
3	From seven to twelve times `MPI_File_read_at`	Blocking, noncollective, explicit offset

4	Six times (once for each field)
	`MPI_File_set_view`
	`MPI_File_write_at_all`
		Blocking, collective, explicit offset

	`MPI_File_set_view` `MPI_File_read_at`	Blocking, noncollective, explicit offset
5	Only for the first seven I/O ranks `MPI_File_write_at`	Blocking, noncollective, explicit offset
6	`MPI_File_set_size`	
7	`MPI_File_close`	

The application parameters are shown in Table 3. As can be observed the request size (rs) depends on local simulation size that is based on the number of MPI processes (np). File size increases as writer processes (wp), rs, simulation steps (st) and fields (fi) increase. Each simulation step that corresponds with an I/O phase moves D_{st} bytes, which means that if we have a writer process per compute node, each compute node will write to the file system $rs \times fi$ bytes.

Table 3. I/O parameters of ACRONYM's I/O kernel.

I/O parameter	Values
Global simulation size	(x, y, z)
Local simulation size	$(x\_loc = x, y\_loc = y, z\_loc = \frac{z}{np})$
Compute nodes	cn
Simulation step	st
Fields	fi
Writer processes	$wp = cn$
Data size (bytes)	ds
RequestSize(bytes)	$rs = x\_loc \times y\_loc \times z\_loc \times ds$
FileSize(bytes)	$fz = cn \times rs \times st \times fi$
Data per st (Bytes)	$D_{st} = cn \times rs \times fi$
Data per 1 cn per st (Bytes)	$D_{cnxst} = rs \times fi$

In Table 4 we define the number of I/O operations based on the parameters that is performed to determine the scalability capacity of the I/O kernel. Request size corresponds to `write_at_all` operations, because they compose the I/O phase with more weight. Count of read operations is dependent upon the st, wp and fi. Furthermore, the rs of read operations is not dependent upon the parameters of the simulation and it is less than 512 Bytes.

The parameters and formula defined in Table 3 are applied to experimental design in Sect. 3.2 to evaluate I/O time impact on SuperMUC.

3.2 Evaluation of the Weight of I/O Operations

A strong scaling test is performed where the global simulation size stays fixed as the number of compute nodes grows. In this case, the request size is decreased and the count of I/O operations increases as the number of compute nodes increases. The used parameters were provided by the ACRONYM developers: the global simulation size in cells, with 52 cells along the x- and y-direction and 66560 cells along the z-direction $(52, 52, 66560)$; 10 simulation steps (st) and 6 fields (fi). The size of data (ds) is 128 Bytes. By using these values we determine the rs and D_{cnxst} (Data per compute node per simulation step). Table 5 presents the values for the experiments, which were calculated with the formulae shown in Table 3.

Table 4. Count of the MPI-IO operations for np processes and $cn = np/16$ based on ACRONYM's I/O parameters.

I/O operation	Count
`MPI_File_open`	$st \times cn$
`MPI_File_write_at_all`	$st \times fi \times cn$
`MPI_File_write_at`	$(fi + 1) \times st$
`MPI_File_set_view`	$st \times fi \times cn \times 2$
`MPI_File_read_at`	$2 \times fi \times st \times cn + 23 \times cn$
`MPI_File_get_size`	st
`MPI_File_set_size`	$st \times cn$
`MPI_File_close`	$st \times cn$

Table 5. ACRONYM IOTEST parameters. Global simulation size is (52,52,66560), File size = 82 GiB, 16 processes per compute node, 8.05 GiB per simulation step with a writer process per compute node.

Compute node (cn) or writer processes	Number of processes (np)	Local simulation size	Request size rs (MiB)	Data per 1 cn per st D_{cnxst} (MiB)
1	16	(52,52,4160)	1373.13	8238.75
5	80	(52,52,832)	274.63	1647.75
10	160	(52,52,416)	137.31	823.88
20	320	(52,52,208)	68.66	411.94
40	640	(52,52,104)	34.33	205.97
80	1280	(52,52,52)	17.16	102.98
160	2560	(52,52,26)	8.58	51.49
320	5120	(52,52,13)	4.29	25.75

To evaluate the I/O time for the different I/O operations of the call tree we use Darshan. Darshan [8] is a profiling tool for characterizing I/O workloads on the petascale systems.

Figure 5 shows the I/O time and the count of I/O operations per type of operation at MPI-IO level. In Fig. 5(a), the count of I/O operations is shown in a stacked histogram. It can be observed as `write_at_all` operations (green), `read_at` operations (cyan) and `set_view` operations (magenta) increase as the number of compute node grows. The percentage of operations is mainly represented by `write_at_all` with 17%, `read_at` with 40% and `set_view` with 34%. Although, count of `write_at_all` operations is less than `read_at`, the data transferred in write phases is more than 99% of the data transferred during the execution of the `IOTEST`.

The I/O operation time is shown in Fig. 5(b). `write_at_all` operations are representing more than 80% of the run time from 20 compute nodes. Despite `read_at` is representing 40% of the count of I/O operations in Fig. 5(a), this has less impact on the execution time, although its influence increases from 80 compute nodes. In the case of `set_view` (time included in the META time), that represent 34% in Fig. 5(a), it does not significantly increase running time.

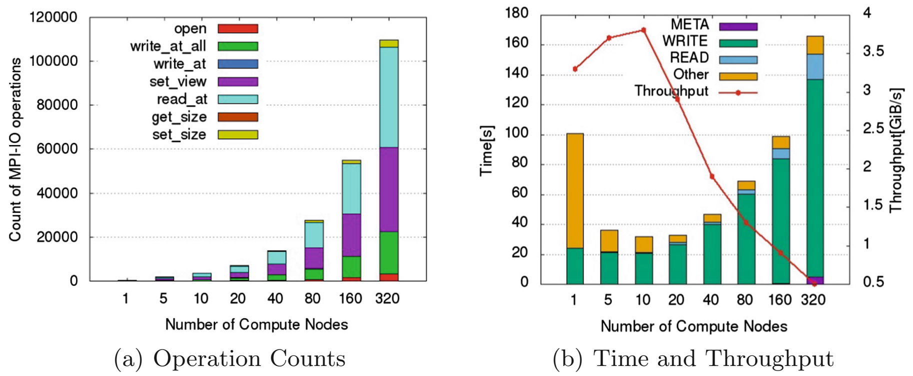
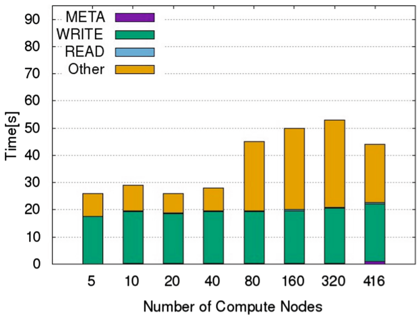


Fig. 5. Time and count of operations at MPI-IO level for the I/O kernel of the ACRONYM application. Mapping corresponds to 16 MPI processes per compute node. Access mode is a shared file and file size is 82 GiB. META time corresponds to cumulative time spent in open, view and close. WRITE is cumulative time spent in write and sync. Global simulation size is x = 52, y = 52 and z = 66,560) (Color figure online)

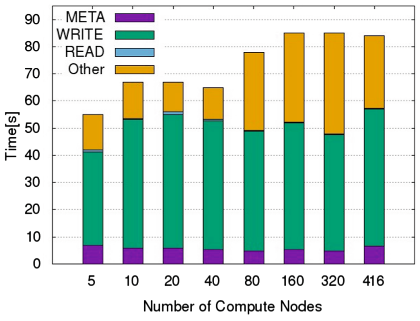
These results show that ACRONYM is not scaling because the number of operations grows and the request size decreases, that means more I/O requests for the file system. Additionally, if the number of writer processes grows then the degree of I/O concurrency increases by generating more load for the file system. Experimental results achieve a transfer rate between 500 MiB/sec and 3.8 GiB/sec, the best result is using an I/O rank per 10 compute nodes where the request size is 137 MiB and the worst corresponds to 360 compute nodes with a request size of 4 MiB. These results affect the performance and it can be observed on the execution time. Using the parameters and formula defined in Table 3 it is possible to select the number of writer processes taking into account the global simulation size to have an appropriate request size to scale.

3.3 Evaluation of I/O Strategies

The original aggregation strategy implemented by ACRONYM developers created a new I/O communicator per compute node based on colors and keys to select the writer processes. Colors and keys are calculated by using the identifiers of island, rack and node where each MPI rank is running. This strategy allows to have an I/O communicator considering closeness of the MPI ranks. We have to mention that in SuperMUC by default the number of I/O aggregators is 1 per compute node when collective operations are used. Therefore the user could



(a) Disabled Chunking



(b) Enabled Chunking

Fig. 6. Time at MPI-IO level for the revised I/O kernel of the ACRONYM application. Mapping corresponds to 16 MPI processes per compute node. Access mode is a shared file and file size is 82 GiB. META time corresponds to cumulative time spent in open, view and close. WRITE is cumulative time spent in write and sync. 5 writer processes for 5 compute nodes and 10 writer processes are setup for the rest of the experiments. Global simulation size is x = 52, y = 52 and z = 66,560) (Color figure online)

have used that strategy and not implement an own solution. However, we have observed the collective buffering report poor performance in SuperMUC.

Using the pattern analysis done we revised the original aggregation strategy removing unnecessary open calls and two aggregation strategies are implemented: (i) a small number of computational ranks also act as designated writer processes. These aggregate data from neighboring processes in simulated 3D space via MPI communication and the data is rearranged in memory; (ii) Enable HDF5 chunking based on a manual size selection or on an automatic selection where chunk size is equal to the output block size of writer processes. In both approaches, the data is subsequently written to disk, fewer processes are participating and larger blocks are sent to the filesystem. Moreover, the scheme is setup such that the number of writer along each dimension is configurable via simulation parameters.

Results for the first strategy is shown in Fig. 6(a). In this case the number of writer processes stays fixed to 10, except for 5 compute nodes where the writer processes are five. We can observe that the I/O time does not increase as increase the number of compute nodes which reduces the total execution time. Figure 6(b) shows execution time for the modified version with chunking strategy enabled and the same number of writers to Fig. 6(a). Chunking produces a fixed overhead for metadata and the I/O time does not grow as in original version.

By comparing the two strategies, we can observe better results for the first where the user can select the number of writers. This number of writers was selected considering the request sizes presented in Table 5 where the request sizes by using 5 and 10 writers obtain more performance in Fig. 2(b). In this case, we have selected the option with more writers because this value does not impact on the total time and provides a higher I/O parallelism degree.

4 Experimental Evaluation

In this section, we present the result of applying the modified I/O strategy on the Acronym for a weak scaling case. In order to avoid the overhead of thousands of processes accessing a single file, a small number of designated writer processes is chosen at the beginning of a simulation run. Through a setting in the configuration file, the number of MPI ranks taking part in the field output operation is selected. Communication occurs only between the computational nodes and their designated writer process as well as among the small number of writer processes.

Figure 7 shows the results of Acronym, the black continuous line represents the ideal scaling, the red line represent the time without I/O. The blue line represent the original strategy. We can clearly see the improvements with stronger aggregation factors, i.e. larger number of nodes sharing one writer process in yellow line with inverted-triangle. Results showed total time 4.5x faster than the original version for the best case.

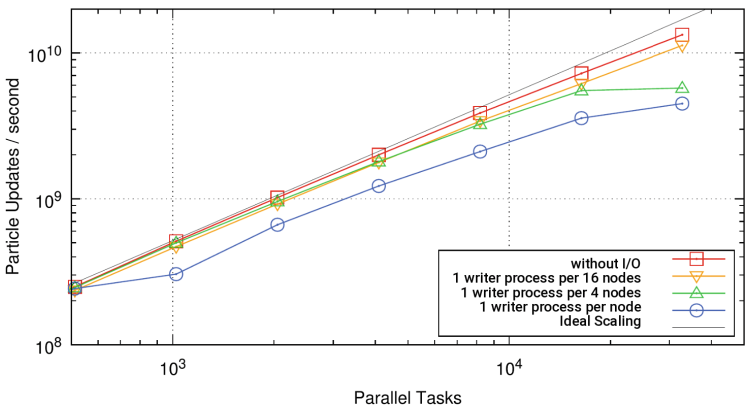


Fig. 7. Weak scaling of the Acronym PiC-Code with and without I/O by using the optimized I/O implementation (plot provided by Acronym developer team)

5 Conclusions

We have presented an analysis of the I/O scalability of Acronym parallel code and have shown promising results by using an I/O aggregation strategy taking into account the system topology of SuperMUC (i.e. island and node configuration of the system). We have defined the request size considering the simulation parameters and the I/O pattern that allows us to select an appropriate number of writer processes in the experimental design time.

We did a characterization of the I/O system and used the results to explain the behavior of the original I/O implementation of Acronym. Furthermore, we discussed a suitable range of aggregation factor for the implemented I/O aggregation scheme based on the characterization results. Moreover, the presented I/O characterization can provide guidelines for other users of SuperMUC encountering problems with I/O scalability.

Initial scaling tests of Acronym showed a sub-linear scaling with the number of MPI ranks suggesting a significantly growing communication overhead. Moreover, the data volume written by each MPI task was small by increasing the count of I/O operations, exacerbating the problem. This brought a complete redesign of the I/O routines into play. In latest code version, a small number of computational ranks act as designated I/O agents. This newly implemented method provides much better scaling even for simulations up to 32k cores by showing a total time 4.5x faster than the original version.

References

1. SuperMUC: Leibniz supercomputing centre (LRZ). Technical report, Bayerischen Akademie der Wissenschaften (2014)
2. Kilian, P., Burkart, T., Spanier, F.: The influence of the mass ratio on particle acceleration by the filamentation instability. In: Nagel, W.E., Kröner, D.B., Resch, M.M. (eds.) High Performance Computing in Science and Engineering 2011, pp. 5–13. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-23869-7_1
3. Byna, S., et al.: Parallel I/O, analysis, and visualization of a trillion particle simulation. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis SC 2012, vol. 59, pp. 1–12. IEEE Computer Society Press, Los Alamitos (2012)
4. Thakur, R., Gropp, W., Lusk, E.: Data sieving and collective I/O in ROMIO. In: Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation FRONTIERS 1999, pp. 182–189. IEEE Computer Society, Washington (1999)
5. Tessier, F., Malakar, P., Vishwanath, V., Jeannot, E., Isaila, F.: Topology-aware data aggregation for intensive I/O on large-scale supercomputers. In: Proceedings of the First Workshop on Optimization of Communication in HPC COM-HPC 2016, pp. 73–81. IEEE Press, Piscataway (2016)
6. Mendez, S., Rexachs, D., Luque, E.: Modeling parallel scientific applications through their input/output phases. In: 2012 IEEE International Conference on Cluster Computing Workshops (CLUSTER WORKSHOPS), pp. 7–15, September 2012
7. Mendez, S., Panadero, J., Wong, A., Rexachs, D., Luque, E.: A new approach for analyzing I/O in parallel scientific applications. In: CACIC 2012, Congreso Argentino de Ciencias de la Computación, pp. 337–346 (2012)
8. Carns, P., et al.: Understanding and improving computational science storage access through continuous characterization. Trans. Storage **7**(3), 8:1–8:26 (2011)