Trace-based Visualization as a Tool to Understand Applications' I/O Performance in Multi-Core Machines

Rodrigo Virote Kassick[1], Francieli Zanon Boito[1], Matthias Diener[1], Philippe O. A. Navaux[1], Yves Denneulin[2], Claudio Schepke[1], Nicolas Maillard[1], Carla Osthoff[3], Pablo Grunmann[3], Pedro Dias[3], Jairo Panetta[4]

[1]Instituto de Informática, Universidade Federal do Rio Grande do Sul - Porto Alegre, RS, Brazil
{*rvkassick, fzboito, mdiener, navaux, cschepke, nicolas*}*@inf.ufrgs.br*

[2] Laboratoire d'Informatique de Grenoble (LIG), ENSIMAG - Montbonnot-Saint Martin, France
yves.denneulin@imag.fr

[3] Laboratório Nacional de Computação Científica (LNCC) - Petrópolis, RJ, Brazil
{*osthoff, pablojg, pldsdias*}*@lncc.br*

[4] Centro de Previsão de Tempo e Estudos Climáticos (CPTEC/INPE) - Cachoeira Paulista, SP, Brazil
jairo.panetta@cptec.inpe.br

Abstract

This paper presents the use of trace-based performance visualization of a large scale atmospheric model, the Ocean-Land-Atmosphere Model (OLAM). The trace was obtained with the libRastro library, and the visualization was done with Pajé. The use of visualization aimed to analyze OLAM's performance and to identify its bottlenecks. Especially, we are interested in the model's I/O operations, since it was proved to be the main issue for the model's performance. We show that most of the time spent in the output routine is spent in the close operation. With this information, we delayed this operation until the next output phase, obtaining improved I/O performance.

1 Introduction

The input/output (I/O) performance observed by an application when accessing a parallel file system (PFS) depends on several factors, such as its data access pattern, concurrency on the file system, number of data servers, I/O libraries, etc. Since there are a lot of impacting factors, it is often very difficult to understand why the performance is poor. Because of that, it is also difficult to identify the changes that can be made in order to improve the I/O performance.

This paper proposes the use of trace-based visualization in order to understand the I/O performance of applications. As an example, we used the *Ocean-Land-Atmosphere Model (OLAM)* [13]. Execution traces were obtained with a trace library, libRastro [5], which we integrated into the source code of OLAM. The traces were then processed and visualized with Pajé [6]. The visualization allowed us to identify a routine with low performance. OLAM's code was then modified, increasing the model's performance for large numbers of processes.

OLAM is a numerical model for weather and climate prediction. It is an important example because high performance implementations of these models are fundamental for these activities, since increasing performance results in higher achievable resolution and accuracy. Such applications are frequently executed on distributed architectures with multi-core processing elements. It's thus important to understand how to make application use these available cores in an efficient manner without overloading the underlying I/O system.

The remainder of the paper is organized as follows: Section 2 presents OLAM and its I/O properties. Section 3 describes the tools used in this work and their integration with OLAM. The results of the visualization are shown and

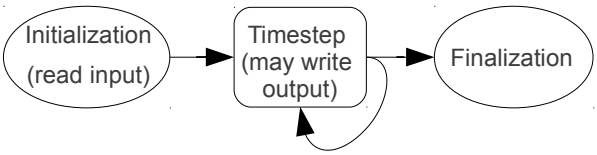


Figure 1. OLAM's iterative organization.

analyzed in Section 4. The modifications done in OLAM's code and their consequences in the model's performance are presented in Section 5. Section 6 discusses some related works, and Section 7 summarizes conclusions and presents future directions.

2 Ocean-Land-Atmosphere Model (OLAM)

This section presents the *Ocean-Land-Atmosphere Model* (OLAM) [13], developed at Duke University. The model aims to represent both global and local scale phenomena, through a global grid that can be refined on points of interest. OLAM was developed in FORTRAN 90 and parallelized using the Message Passing Interface (MPI).

OLAM is an iterative model, where each timestep may result in the output of data as defined in its parameters. The model's workflow is illustrated in Figure 1. In order to simulate a typical analysis environment for this case study, a resolution of 280.5 km was used, with the atmosphere divided into 28 layers. The simulation was organized in timesteps of integration of 60 seconds, simulating 24 hours.

For this case study, OLAM requires reading 4.6 MB of input files. In the output phase, each process creates a history file, writes its state and closes the file. Therefore, the number of independent output files increases with the the number of processes. The history files are considered to be of small size for the standards of scientific applications: file size ranges from 100 to 600 KB, depending on the grid definition and the number of processes. This leads to an access pattern of "large amount of small files" that comes with a great cost in terms of I/O performance: the overhead of file creation and the small size of the writes cause the file system to perform poorly.

In order to avoid the overhead imposed by the large number of files, a version of OLAM was developed using OpenMP in addition to MPI. In this version, the MPI processes create OpenMP threads to process each timestep. Therefore, this version uses a different level of parallelism and generates less files, resulting in 20 times better I/O performance for a reduction of 8 times in the number of files [2].

However, despite the I/O performance increase in the hybrid version of the model, scalability issues still remain [2, 10]. This led to the current approach of using visualization to analyze the application's performance.

3 Trace-based Visualization of OLAM

To obtain the trace of OLAM we used the libRastro tracing library [5]. LibRastro is used to generate execution traces of applications with a minimal impact on its behavior. To use the library, the source code of the target application must be modified at the points of interest in order to generate events. Beginning and end of these events are specified by two subroutine calls: `IN` and `OUT`, respectively. Each event has a name and optional parameters.

In the case of I/O operations of OLAM, the parameters are the name of the file, amount of data written/read, among others. Besides the I/O operations, we created events in all of the most important subroutines of OLAM, with the goal of identifying portions of the execution which are impaired by the I/O operations or other factors. Moreover, the detailed analysis of the application can identify the parts that do not scale.

During execution with libRastro, each process of the application generates a binary trace file. These trace files must be merged and converted to a higher level language by an application-specific tool, because the semantics of the events change from one application to the other.

One high-level event description language is Pajé [6]. Pajé allows the developer to describe events, states and messages between distinct containers (a container being any element that may have states, events or be source or destination of a message). The developer has a great flexibility to create containers and the associated events in a way which best describes his code.

In our OLAM's modeling, each MPI Rank was represented by one Pajé container. The states of this container are the events obtained from the trace. Therefore, there are states inside other states (when one subroutine calls another).

Each event must be of a predefined *type*. Pajé groups events of the same type in an execution flow and automatically stacks one state inside the other as in the case of function calls. We defined the `APP_STATE` type in which we map events related to the OLAM application. There are also the `P_STATE` type, which corresponds to I/O utility functions, and the `MPI_STATE` type to which MPI events are mapped.

The visualization was done via the Pajé Visualization Tool. It allows for a *gantt-chart* style, time based visualization of the events and states of the containers. The next section presents the results obtained.

4 Results of the Performance Visualization

In order to obtain traces from OLAM, we executed the instrumented version of the application on the clusters *Adonis* and *Edel* of Grid5000 [3]. The tests were executed with

8 nodes using either the local file system or the shared NFS volume to store the execution output. We tested OLAM with and without OpenMP threads.

Figure 2 presents part of the Pajé visualization for the execution of OLAM with 8 processes, each with 8 threads, over local files and NFS. The rectangles on the left of the graph show the *APP_STATE* and *P_STATE*, as discussed in Section 3. When the application enters the *OLAM_OUTPUT* state, the underlying I/O functions are presented in the process container below. This can be more easily observed in Figure 2(a) at around 11 seconds: when the application enters the *O_OUPUT* state, the process calls a sequence of HDF5 helper functions, of which *SHDF5_OREC* takes the longest. This functions is responsible for writing the variables describing the atmospheric conditions to the output file of the process.

In Figure 2(a) we can see the first 9 seconds of execution for some of the processes of OLAM. In the *APP_STATE* flow, we can observe the execution of a sequence of *timesteps* (event TIMESTEP) after which the *olam_output* (event O_OUTPUT) function is called. At around 6.5 and 11 seconds of execution, we can observe that one O_OUTPUT event that takes longer to complete, something that can be observed in other parts of the execution and in other processes. The execution over NFS (Figure 2(b)) has a similar behavior, but the divergence between the normal I/O phases and the long ones is smaller. In these cases, we can observe that the function responsible for the divergence is *hdf5_close_write*, called from within *SHDF5_OREC*.

Figure 3 presents the visualization of traces generated when executing 64 processes of OLAM with no threads (classic MPI). In this case, since the data division becomes more fine-grained process-wise, the time of each timestep is smaller and there are more frequent I/O calls. Despite the overhead associated with creating more files for the same work, the performance was not penalized in the execution with local files (Figure 3(a)) due to the use of a fast local disk shared by each 8 processes. This overhead was made clear when output files are stored in the shared NFS volume (Figure 3(b)). In this case, most of the time spent in the P_STATE flow is in closing the file (SHDF5_CLOSE): small writes end up being delegated to the write-back mechanism in kernel, but the requests must be flushed by the time the file is closed.

Figure 4 presents the time spent on the I/O states of the application for a process from the trace of Figure 3(b). We have observed that I/O functions occupy around 25% of the execution time. Most of this time is spent on the *SHDF5_CLOSE_WRITE* function. This is due to the write-back mechanism, as said before, indicating that this is a good target to optimizations.

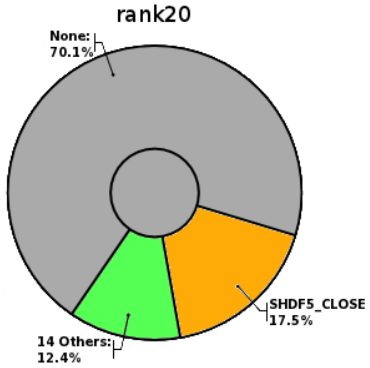


Figure 4. Relative time spent on I/O states

5 Improving OLAM's Performance

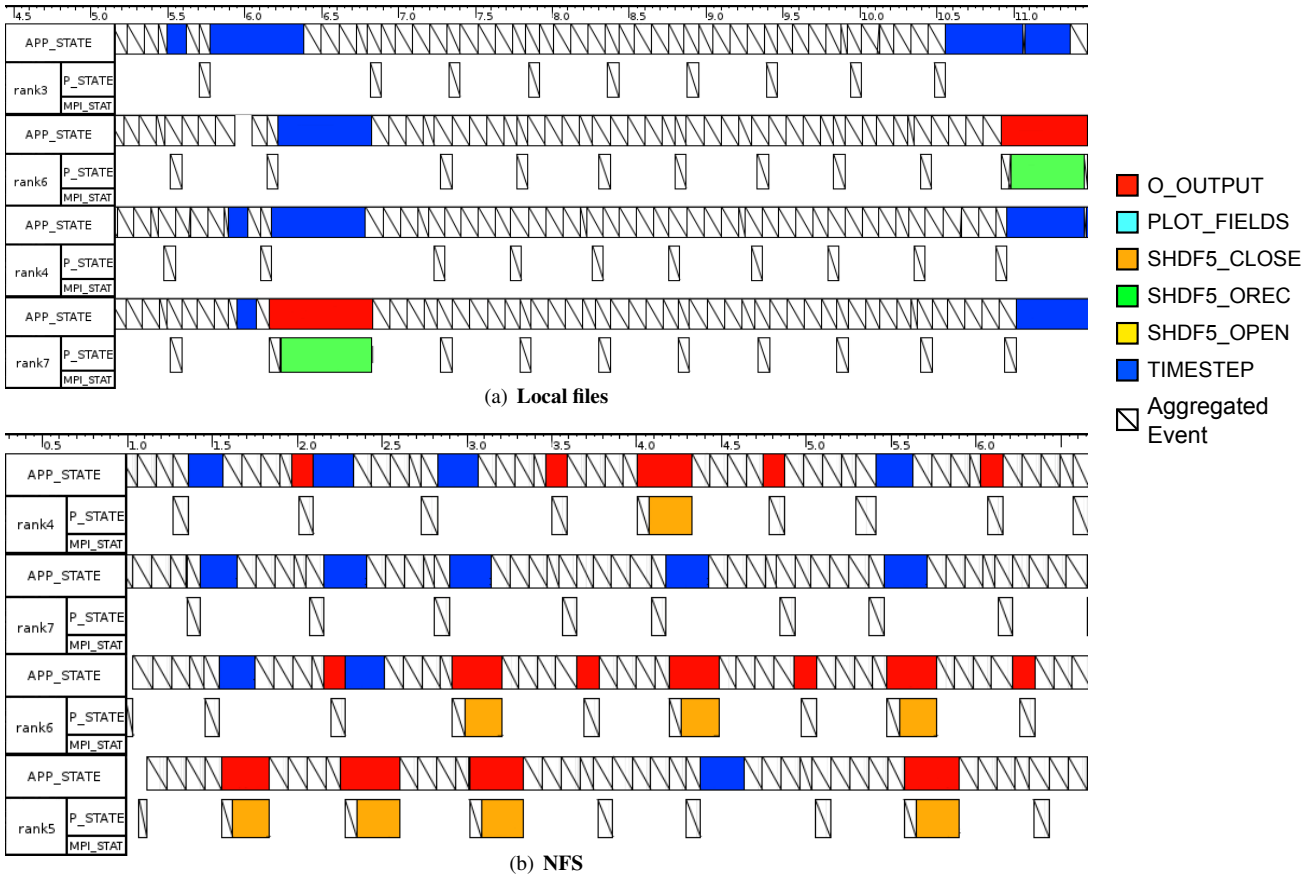
OLAM does not profit from the write-back mechanism due to the small size of it's I/O operations: by the time the process calls *SHDF5_CLOSE*, it's previous write requests have not been sent to the server. One way of forcing OLAM to profit from the background I/O operations offered by the file system is to change the order that the files are opened and closed. Instead of calling the close function after each process's I/O phase, the file can be closed before a new one is opened – i.e. during an *SHDF5_OPEN*.

This way, instead of having an order of events of *Computation – Open File – Write Data – Close File – Computation* ..., the order is chained to *Computation – Close Previously Opened File – Open File – Write Data – Computation* ...,

This change was evaluated in the *Adonis* cluster of Grid5000. We used the MPI+OpenMP version of OLAM and the pure MPI implementation, adjusting the number of OpenMP threads so that the number of cores used was the same as in the MPI version. OLAM was executed with three different output intervals: 10, 30 and 60 minutes of simulation. The smaller the intervals, the higher the number of generated files.

Figure 5 presents the execution time for this set of tests. The continuous lines are the execution times for OLAM with the proposed Close-on-open optimization, while the dashed lines represent the execution time with the standard implementation.

In previous works [10] we have shown that the OpenMP implementation performs better than the pure MPI one due to the smaller amount of files to be created. Because of this, the optimization did not provide performance gains for this version. We can observe that, as the number of processes increases, the difference between the times with and without optimization decreases. We expect, therefore, an improvement in performance for larger number of files with the MPI+OpenMP version of OLAM.



(a) **Local files**

(b) **NFS**

Figure 2. OLAM execution with 8 Processes, 8 Threads each (64 cores)

With pure MPI the file system is stressed with the larger amount of files. With 80 cores the proposed optimization performed 15% better than the original one for the 10min and 37% better in the 30min interval. Gains were also seen for the 60min interval configuration. The smaller the interval, and therefore the larger the number of generated files, the sooner the gain was observed. This indicates that, as previously stated, the improvement obtained by the optimization increases with the number of generated files and, therefore, we can expect to observe these gains in the OpenMP version too (note that more processes incur in more files).

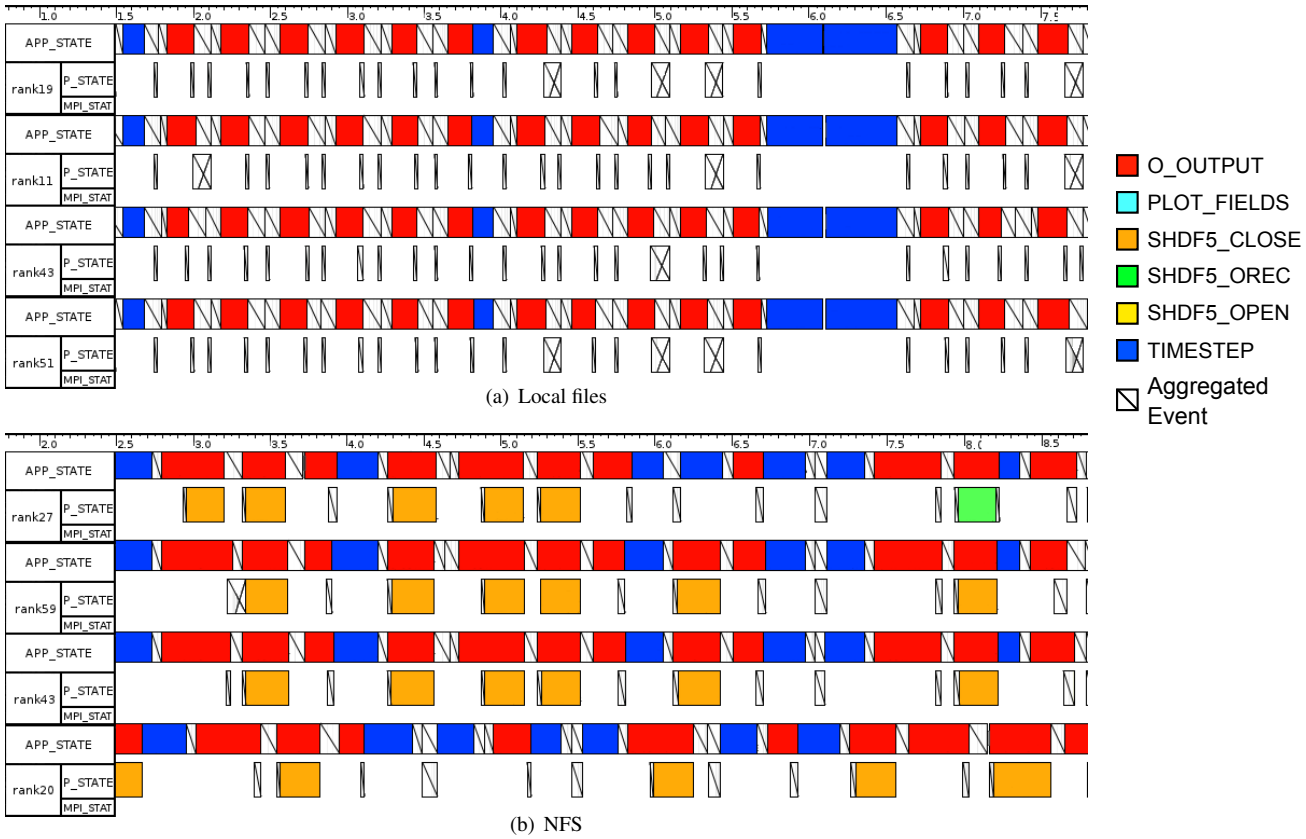
6 Related Work

Scalability of atmospheric models is the focus of several papers In [7] the authors execute the high resolution WRF weather forecast code on 15k cores and conclude that one of the greatest bottlenecks was data storage.

I/O contention on multi-core systems is also a known issue in the literature and a few strategies to mitigate the performance loss can be found. In [9], the authors improve the performance of multiple concurrent I/O streams on multi-core nodes using data aggregation, forcing contiguous access on the storage nodes. A similar strategy is used in IOFSL [1], an initiative from the PVFS team. Carns et al. [4] employed five techniques on PVFS in order to improve its performance for small-files scenarios: precreation of objects, stuffing of files, coalescing metadata commits, eager I/O and POSIX extensions for directory access.

As explained before, OLAM's scalability is an ongoing research problem. [10] presents several results regarding the model scalability on multi-core clusters and identifies points of contention. More recently, [11] presents an evaluation of OLAM with the Intel Vtune Analyzer and identifies a large amount of cache misses when using 8 cores.

Performance visualization is currently being studied to make it easier to analyze performance issues in parallel programs. Muelder et al. [8] created *IOVIS*, which is a system to analyze I/O system behavior. It consists of two parts: a data collection tool which generates traces by hooking into I/O system software, and a data analysis and visualization tool to present the I/O behavior of an application.



(a) Local files

(b) NFS

Figure 3. OLAM execution with 64 Processes, 1 Thread each (64 cores)

Another approach was taken by Uselton et al. [12]. They extended an existing performance monitoring tool to include I/O tracing and used the generated I/O statistics to find performance problems in several HPC applications. By fixing these problems, they achieved a speedup of up to 4x.

7 Conclusions and Future Work

This paper presented a trace-based visualization of I/O performance of a large atmospheric model, the Ocean-Land-Atmosphere Model (OLAM). OLAM has a well-known scalability issue, and its performance is proved to be limited by its I/O operations. The analysis was done with the intent to understand the model's performance and how it is impaired by I/O.

The libRastro library was used to instrument and obtain the traces of the application. The traces were analyzed and visualized with the Pajé. We have shown that, when using the shared file system to store the results, most of the time spent in the output routines was spent in the *close* operation. Then, we proposed a modification to delay this operation, obtaining an increase in performance of up to 37%.

As future work, we intend to obtain a visualization of the file system's performance as well. Since PVFS has optimizations for small-file scenarios, we intend to instrumentate PVFS's code. This will help us to obtain more details about the flow of application's requests, pointing more optimizations to improve the model's and the file system's performance. We also intend to evaluate the optimization proposed in Section 5 with a larger number of processes in the OpenMP version.

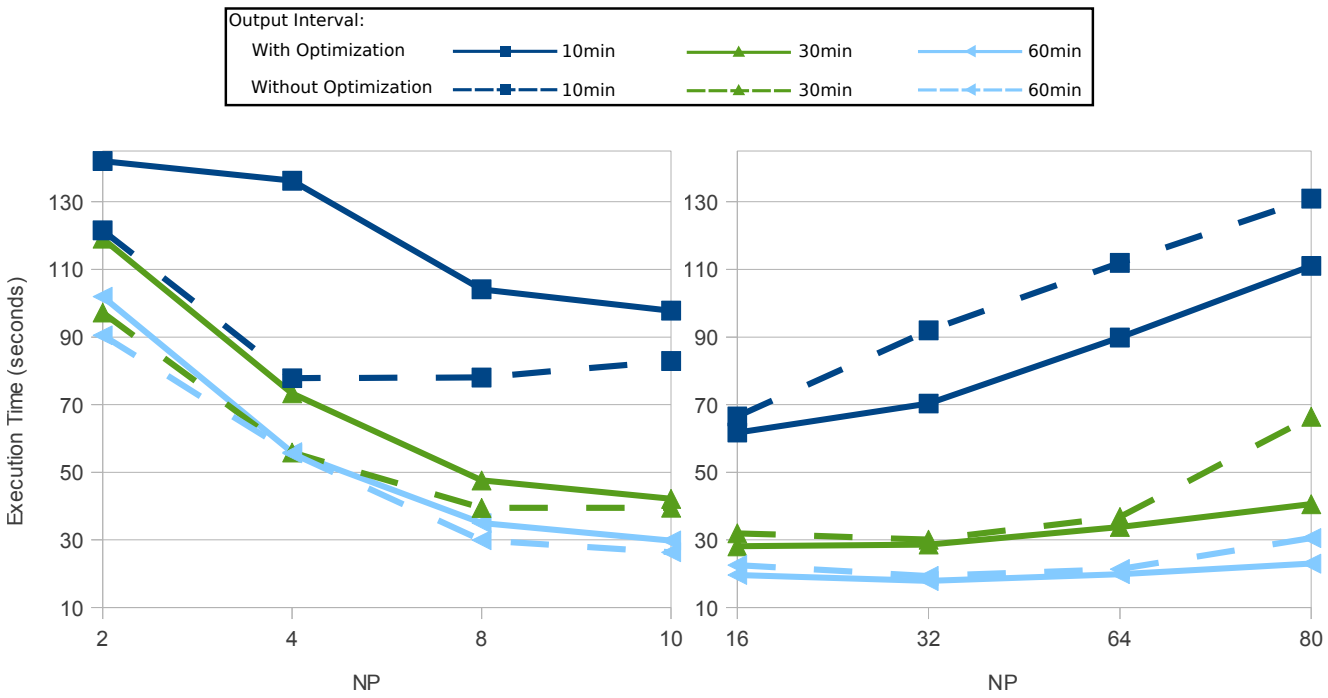
Acknowledgment

This work was partially supported by CNPq, CAPES, FAPERGS and FINEP.

References

[1] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, 2009.

[2] F. Boito, R. Kassick, L. Pilla, N. Barbieri, C. Schepke, P. Navaux, N. Maillard, Y. Denneulin, C. Osthoff, P. Grun-



(a) MPI+OpenMP

(b) Pure MPI

Figure 5. Results for Close-on-Open optimization for MPI+OpenMP and pure MPI

mann, P. Dias, and J. Panetta. I/O performance of a large atmospheric model using PVFS. In *Rencontres francophones du Parallélisme (RenPar'20)*, 2011.

[3] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[4] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig. Small-file Access in Parallel File Systems. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–11, Washington, DC, USA, 2009.

[5] G. J. da Silva, L. M. Schnorr, and B. de Oliveira Stein. Jrastro: A trace agent for debugging multithreaded and distributed java programs. *Computer Architecture and High Performance Computing, Symposium on*, 0:46, 2003.

[6] J. C. de Kergommeaux, B. Stein, and P. E. Bernard. Pajé, an interactive visualization tool for tuning multi-threaded parallel applications. *Parallel Computing*, 26(10):1253 – 1274, 2000.

[7] J. Michalakes, J. Hacker, R. Loft, M. O. McCracken, a. Snavely, N. J. Wright, T. Spelce, B. Gorda, and R. Walkup. WRF nature run. *Journal of Physics: Conference Series*, 125:012022, July 2008.

[8] C. Muelder, C. Sigovan, K.-l. Ma, J. Cope, S. Lang, K. Iskra, P. Beckman, and R. Ross. Visual Analysis of I/O System Behavior for High – End Computing. In *Proceedings of the third international workshop on Large-scale system and application performance (LSAP '11)*, pages 19–26, New York, 2011. ACM.

[9] K. Ohta, H. Matsuba, and Y. Ishikawa. Improving Parallel Write by Node-Level Request Scheduling. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 196–203. Ieee, 2009.

[10] C. Osthoff, P. Grunmann, F. Boito, R. Kassick, L. Pilla, P. Navaux, C. Schepke, J. Panetta, N. Maillard, P. L. S. Dias, and R. Walko. Improving Performance on Atmospheric Models through a Hybrid OpenMP / MPI Implementation. In *Proceedings of the 9th IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2011.

[11] C. Schepke, N. Maillard, C. Osthoff, and P. Dias. Performance Evaluation of an Atmospheric Simulation Model on Multi-Core Environments. In *Proceedings of the Latin American Conference on High Performance Computing 2010*, 2010.

[12] A. Uselton, M. Howison, N. J. Wright, D. Skinner, N. Keen, J. Shalf, K. L. Karavanic, and L. Oliker. Parallel I/O Performance : From Events to Ensembles. *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010.

[13] R. Walko and R. Avissar. The Ocean–Land–Atmosphere Model (OLAM). Part I: Shallow-Water Tests. *Monthly Weather Review*, 136:4033–4044, 2008.