# A Performance Assessment of the Unified Model

Dale Roberts and Mark Cheeseman

National Computational Infrastructure, Canberra, Australia
{ds.roberts,mark.cheeseman}@anu.edu.au

**Abstract.** The Unified Model (UM) is a model produced by the UK MetOffice for Numerical Weather Prediction (NWP) and climate simulation. It is used extensively by various university, government and other research organizations on the large supercomputer hosted at the National Computing Infrastructure (NCI). A 3-year collaboration between NCI, the Australian Bureau of Meteorology and Fujitsu is underway to address performance and scalability issues in the UM on NCI's supercomputer, Raijin.

IO performance in the UM is the most dominant factor in its overall performance. The IO server approach employed is sophisticated and requires proper calibration to achieve acceptable performance. Global synchronization and file lock contention is a problem that can be remedied with simple MPI global collective calls. Complimentary IO strategies, such as MPI-IO and directed IO, are being investigated for implementation.

The OpenMP implementation employed in the UM is investigated, and is found to have inefficiencies that are detrimental to the load balance of the model. Only loop-wise parallelism is employed. Due to the inherently imbalanced nature of the model, a task-wise approach could yield improved threading efficiency.

**Keywords:** unified model, numerical weather prediction, performance analysis, high performance computing.

## 1 Introduction

The Unified Model, produced by the UK MetOffice, is a model used for atmospheric simulation over time scales ranging from a few hours for numerical weather prediction (NWP) to decades for climate modeling. An overview of its development can be found in [1]. It is used in the Australian Bureau of Meteorology's (BoM) operational NWP forecasts at both a global and regional resolution. It also serves as the atmosphere component of the Australian Community Climate and Earth System Simulator (ACCESS) coupled climate model. Its complexity and scale demand the use of supercomputers. For this paper, we focus on a global configuration of the UM on the National Computational Infrastructure's supercomputer, Raijin. Raijin is a Fujitsu Primergy cluster consisting of 3592 nodes. Each node possesses two Intel Xeon E5-2670 8 core CPUs giving Raijin a total of 57,472 CPU cores and a theoretical peak performance of 1.2 Petaflops. The nodes are connected by a fast Infiniband network, capable of 56GBit/s between any two nodes. Raijin also provides a fast lustre-based

R. Denzer et al. (Eds.): ISESS 2015, IFIP AICT 448, pp. 552–560, 2015.

[2] file system with approximately 10PB of storage capacity and a peak write performance in excess of 150GB/s. For this paper Intel FORTRAN and C compilers are used along with OpenMPI [3] for MPI (Message Passing Interface) support.

The UM configuration of interest for the presented performance assessment and optimization work is the global N512L70 configuration. It possesses an approximately 25km horizontal resolution (making for a horizontal grid of dimensions 1024 by 769) at mid latitudes and contains 70 model vertical levels. Optimization of configuration is important to BoM's future operational weather forecasting suite, as it will lead to greater accuracy in both weather and climate prediction. BoM has mandated a strict performance target of 10 model days per wallclock hour if this global configuration is to be included in its operational forecast suite. We suspect that performance issues observed at N512L70 (such as I/O and efficient OpenMP use) may be more severe at higher resolutions, such as N768L85, the global resolution of the next BoM operational forecast suite. Thus any solutions implemented at the N512L70 configuration should be applicable and valuable for them as well.

## 2 Initial Scaling Tests

Using version 8.4 of the Unified Model [4] a strong scaling analysis of the global N512L70 configuration is performed. Each run lasts 24 model hours with a time step of 10 minutes. The UM is capable of using a multi-threaded model, in that each MPI task is able to spawn multiple OpenMP threads. When this multi-threaded mode is enabled, the IO can be performed using the IO Server asynchronous dump feature. In this feature a number of MPI tasks are set aside to perform the expensive process of writing model output to disk as the remaining MPI tasks perform the normal computational work in the atmosphere model run. If this feature is not activated, all output is passed to and written by a single MPI task while all other MPI tasks sit idle – thereby interrupting the normal model calculations.

Figure (1) shows observed strong scaling of the model with the IO Server feature disabled and with all output switched off. It's clear that the single task writing approach severely limits scalability. There is no observed benefit in using more than 512 CPU cores with output enabled in this case. With an averaged runtime of 984 seconds at 2048 CPU cores (over 5 runs), it will not be possible to meet BoM's operational performance requirement with this configuration.

With the IO Server feature enabled, observed strong scaling improves substantially as illustrated in Figure (2). The UM must be run in multithreaded mode here. (I.e. at least 2 OpenMP threads need to be spawned and available to each MPI task). This requirement comes from the IO Server design that uses two threads per IO Server MPI task. Thus, the use of IO servers will be an integral part in meeting operational performance requirement.

The two sets of points at 512 cores in Fig. (2) denote two different runs performed on Raijin, one with 2 OpenMP threads per MPI task, and one with 4 OpenMP threads per MPI task. It is clear that the best utilization of 512 cores is to have 256 MPI tasks each with 2 OpenMP threads. This type of observation leads us to believe that improved performance in the UM could be achieved through additional OpenMP optimization. In spite of the IO now residing on separate MPI tasks, Figure (2) shows that the performance with IO disabled altogether is still better than when IO is enabled on the IO servers.

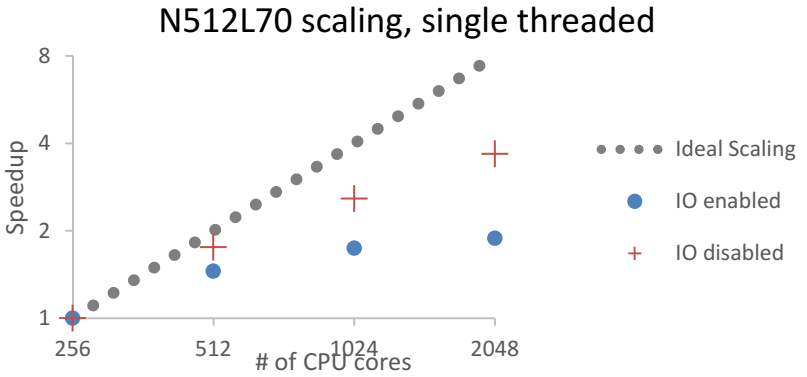


**Fig. 1.** Observed strong scaling of the global N512L70 configuration using the single MPI task I/O approach. We define 'Speedup' as the ratio of the runtime at 256 cores compared to that at larger stated core counts. Ideal scaling would see a job run on twice as many cores take half the time, a speedup factor of 2.

It is possible for an IO server to possess more than 1 MPI task. For the 'IO enabled' runs shown in Figure (2), we used IO servers containing 8 MPI tasks each. Write performance can be enhanced by fine-tuning the layout and structure of the IO servers used –including the number of MPI tasks allocated to each. As an example, in Figure (2), we display the difference in observed strong scaling at 4096 cores using 8 and 16 MPI tasks per IO Server (e.g. the green cross). While IO Server tuning does improve observed runtimes, it is not enough to significantly change the overall scalability of the global configuration.

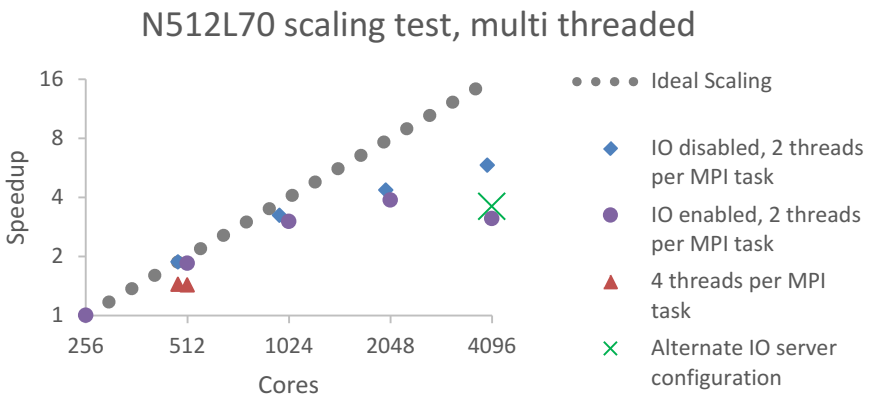


**Fig. 2.** Observed strong scaling of global N512L70 configuration with IO Servers enabled

## 3 Threading Performance

Investigations showed that, while the UM can be run with up to 16 OpenMP threads per MPI task on Raijin, performance decreases significantly when more than two threads per MPI task are employed. A detailed analysis of the OpenMP performance was completed using the Score-P [5] profiling and tracing tool, in conjunction with the Vampir trace viewer [6]. For this particular analysis, the UM N512L70 configuration was run in a 16×30 decomposition, with 4 IO servers, each comprising 8 tasks; giving a total of 512 MPI tasks, each with 2 OpenMP threads. Score-P adds approximately 20% to the model run time for a full performance trace. Vampir categorizes all function calls, and displays each category in with different colors according to the legend in Figure (3).

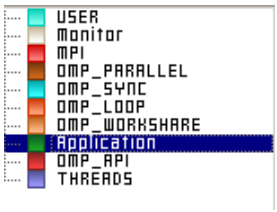


**Fig. 3.** The function group legend used by Vampir. An ideal trace will spend the majority of its time in the 'Application', 'OMP_LOOP' or 'OMP_WORKSHARE' states. These categories denote function calls in which useful computational work is occurring. Time spent in 'OMP_SYNC' and 'MPI' denotes points where one OpenMP thread is waiting for another to compete a task, or waiting for data to be received from another process, and no useful computation is occurring.

Fig. (4) shows a selection of atmosphere tasks in the 'Master Timeline' view from Vampir version 8.3.0. The view has been zoomed in on a single atmosphere time step, without radiation calculations, that is performed every 6 time steps.

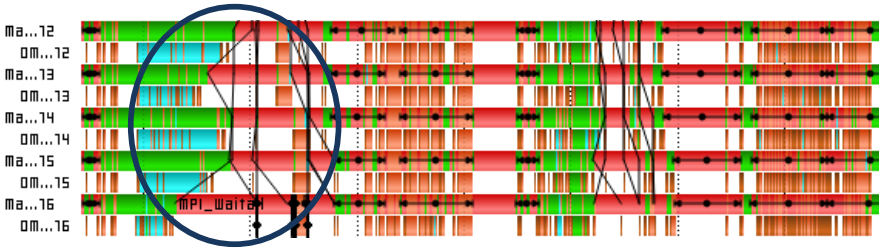


**Fig. 4.** The 'Master Timeline' view of 5 MPI processes performing a single UM atmosphere step. Function groups are colored as per Fig. (3). Note the significant load imbalance in the circled region.

The vast majority of time between the start of the atmosphere step and the first 'MPI_WaitAll' call (the circled section of the trace) is spent in the 'OMP_SYNC' category of calls. This category denotes every time an OpenMP thread needs to wait

for another thread to complete. In this case, the large scale precipitation calculation in the microphysics section of the atmosphere step contains an `!$OMP PARALLEL` declaration, which then proceeds to be executed in serial, whilst the newly created OpenMP thread waits for the completion of the main thread. This section of the code is potentially taking twice as long as necessary, thereby increasing the load imbalance among the tasks. Load imbalance comes about when each MPI task performs a different amount of work on the data it has. When this occurs, the tasks that complete that section first need to wait for the final tasks to complete that section. This can be seen in the circled region of Fig. (4). During this wait period, they will not be doing any useful work, and are considered wasted. One of key activities in improving the performance of this UM configuration will be eliminating this type of observed load imbalance. Fig. (5) illustrates a similar scenario. Here we display the work done by two OpenMP threads assigned to a single MPI task running an atmosphere step.

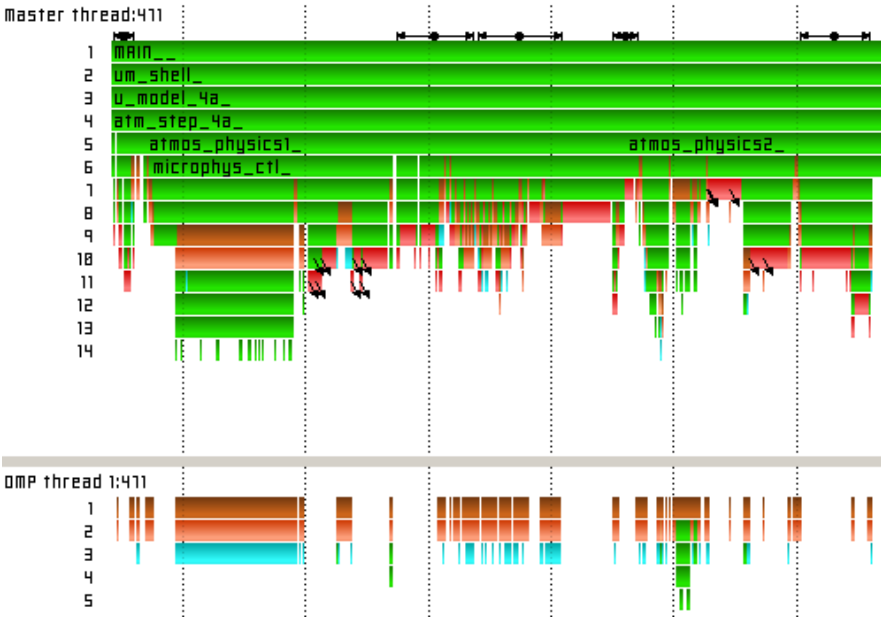


**Fig. 5.** Vampir's view of the call stack of 2 OpenMP threads allocated to an MPI task running a portion of one atmosphere step. Function groups are colored as per Fig (3).

For the entire atmosphere step, OMP thread 1 is active (i.e. the program is inside an `!$OMP PARALLEL` region) for approximately one third of the time and spends over a quarter of its time in in the `OMP_SYNC` category. Though we have selected an arbitrary task for this comparison, this behavior is observed in all non-IO MPI tasks.

Consequently, one OpenMP thread on each atmosphere MPI task spends approximately 30% of its time in `MPI_WaitAll` calls. This load imbalance is further exacerbated as the halo exchange immediately follows, resulting in more idle CPU time as neighbor tasks wait for threads to synchronize.

OpenMP use in the UM atmosphere code is overwhelmingly data-based parallelization, in which separate iterations of a 'DO' loop are executed on individual OpenMP threads. For a model as inherently load imbalanced as the UM, this may not be sufficient. An alternative approach could be to employing work-share OpenMP constructs where separate independent tasks are allocated to different OpenMP threads. A potential target for this approach would be the convection control subroutines. Every atmosphere tile may undergo one of deep, shallow or congestus convection, as well as mid-level convection. This routine is particularly load imbalanced, as the entire model area on a particular MPI task may only undergo mid-level convection, whereas other MPI tasks may spend a significant amount of time in one of the other convection routines as well as mid-level convection. One could separate the deep, shallow and congestus convection routines on to different OpenMP tasks, such that they are calculated simultaneously, and then revert to data-based parallelism for the mid-level convection calculation.

## 4 IO Performance

Jobs ran at the global N512L70 configuration run for 1 model day and generate approximately 151GB of output data to disk. If one ignores the UM code completely and just performs a single low-level sequential write to Raijin's high-speed filesystem, then it takes about 220 seconds to perform the write. BoM's current operational forecast suite requires the global UM configuration to run for 10 model days and generate an accordingly ten-fold increase in output. If one ignores the UM code and gain just performs a single low-level sequential write, the time required for generating the output jumps to approximately 36 minutes. BoM's operational forecast suite cannot last longer than one wallclock hour. Thus, one has (at best) 24 minutes to complete all computations in a N521L70 run which is not feasible currently. (Both I/O approaches, IO Server and single MPI task output, use sequential writes for data output.)

In the IO server implementation, model output can be written concurrently with normal atmospheric calculations. At each write, each MPI task performing atmosphere computations sends relevant output data to the nearest IO task (by MPI rank) of each IO server. The first IO server to receive all necessary output data commences the writing of the first output/dump file. The remaining IO servers will flush out all data pertaining to that output file and focus on the next output file.

The 1 model-day runs of the N512L70 configuration produces fifteen dump files in total. Testing showed that the optimal IO Server setting for this configuration varies depending on the number of MPI tasks assigned to the atmosphere model. Fig. (6) shows that for fewer than 1024 or fewer MPI tasks (2048 total cores), 8 MPI tasks per IO server and up to 4 IO servers. Moving to 4096 total cores, gives 8 IO servers in total, which is seen to harm performance. In the 8 IO server setup, several servers seem to remain idle as fewer than 8 files are being written to simultaneously for the majority of the time. Furthermore, the overhead associated with synchronizing all files across 8 IO servers will ultimately increase the time spent in IO, to the point where the IO will take longer than the model run when a large number of MPI tasks are utilized. Reducing the number of IO servers by increasing the number of tasks per server but retaining the total number of MPI tasks for IO is also seen to be detrimental

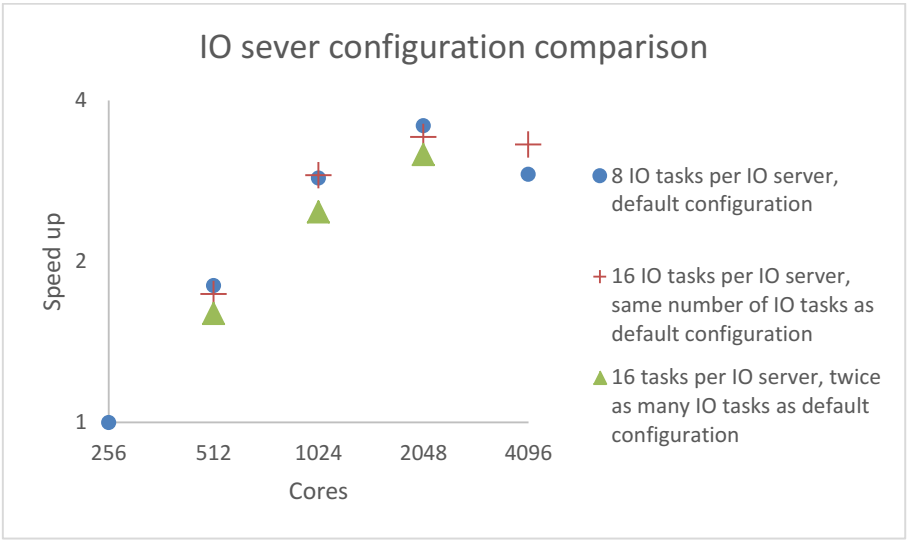


**Fig. 6.** Comparison of different IO server configurations with the same number of total MPI tasks

to performance for 1024 or fewer MPI tasks, as this limits the ability of the IO servers to write multiple files in parallel. Furthermore, increasing the number of IO tasks and retaining the same number of IO servers is also seen to reduce performance, as in this case, fewer MPI tasks are being allocated to the model itself.

Even when IO servers are employed, there is still a significant overhead due to IO. At 4096 cores, it takes approximately 185 seconds to complete the 24-model hour run without IO, and 305 seconds to complete it with IO enabled. Fig. (7) shows a time-sampled profile of the speed of UM output writes during a run with IO servers enabled.

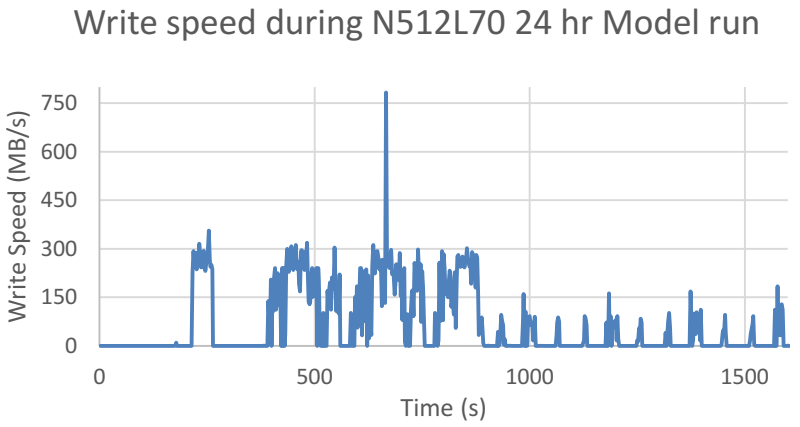


**Fig. 7.** Observed write speeds for a model day run of the N512L70 configuration. The model produces output at approximately 200, 400, 600 and 800 seconds.

Raijin's high-speed file system is capable of sequentially writing to a single file at around 700MB/s. The observed average write speed for a UM model dump is around 300MB/s. This is due to the overhead of the IO servers moving all data across to the main IO server MPI task in 4MB segments, and writing each segment as it is received. Fig. (8) shows the effect this has on atmosphere tasks.

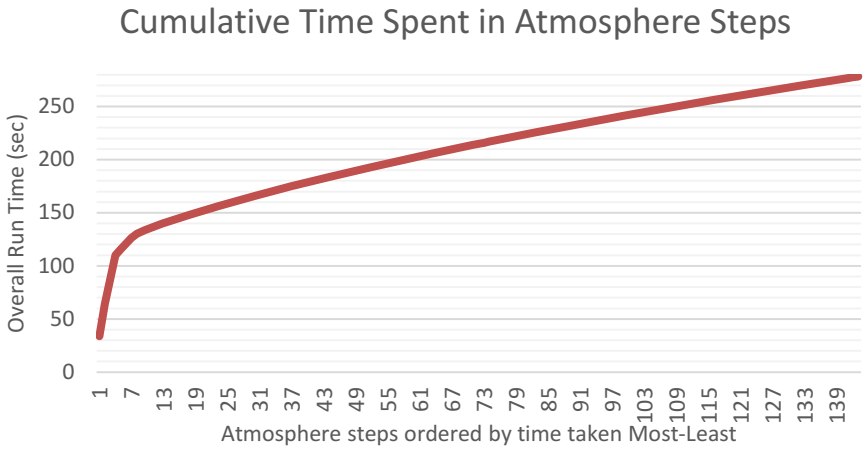


**Fig. 8.** Cumulative time spent in UM atmosphere steps, ordered by longest time step to shortest. This was performed with a 32×62 decomposition using 4 IO servers each with 16 MPI tasks, for a total of 4096 cores.

Over half of the total run time was spent in the twelve longest atmosphere steps. The long duration of these steps is caused by IO servers being unable to complete some given request. There are a variety of reasons for this, for instance, an insufficient amount of memory has been allocated to the IO servers, which, when full, will cause the model to stop whilst the data in memory on the IO servers is written to disk. Many of these issues could be alleviated with tuning of the IO server parameters. However, as one increase the model resolution, this type of fine-tuning will not prevent such stalling. The number of output files will not increase as resolution increase. Thus one will need to either a) increase the amount of memory to each IO server task so that it can hold more output data, or b) increase the number of MPI tasks dedicated to IO server activity (eg. Increase number of IO servers and/or increase number of MPI tasks belonging to each IO server). Each alternative will suffer from the same stalling behavior. Thus alternate I/O methods should be explored. One possible approach is MPI-IO [7] where multiple IO server tasks can write to files concurrently thereby achieving greater write speeds. The overhead of sending entire files across to a single MPI task would also be eliminated.

## 5 Conclusions and Future Work

We have investigated the overall performance of the UM MetOffice's Unified Model focusing on the OpenMP and IO server implementation. We found inefficiency in the

threading, with the second OpenMP thread of each MPI task spending approximately 75% of the time idle. OpenMP parallelism has been predominantly loop/data-wise. A more task-wise OpenMP parallelism strategy could improve the treading efficiency significantly. In particular, the sections of code identified in Section 3, namely, the different convection subroutines, will be the first target for optimization.

IO is a significant factor in the run time of the UM, even with the IO server approach. Tuning of available IO server parameters can improve the performance at N512L70, but it is unlikely that parameter tuning will have sufficient performance for higher resolution configurations. One possible way to provide such speeds is to make use of Raijin's high-speed parallel file systems by employing a parallel IO standard, such as MPI-IO. In particular, the large dump and post-processing files produced, at several gigabytes each, rely on accumulating data from each IO server process before writing the data from a single MPI task. This accumulation step can be replaced by MPI-IO calls that allow each IO process to write to the file in lustre-aware manner.

## References

1. Brown, A., Milton, S., Golding, B., Mitchell, J., Shelly, A.: Unified Modeling and Prediction of Weather and Climate A 25-Year Journey. American Meteorological Society, 1865-1877 (December 2012)
2. Braam, P.: The Lustre Storage Architecture (2004), `ftp://ftp.uni-duisburg.de/pub/linux/filesys/Lustre/lustre.pdf`
3. Gabriel, E., et al.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 97–104. Springer, Heidelberg (2004)
4. Wood, N., Staniforth, A., White, A., Allen, T., Diamantakis, M., Gross, M., Melvin, T., Smith, C., Vosper, S., Zerroukat, M., Thuburn, J.: An inherently mass-conserving semi-implicit semi-Lagrangian discretization of the deep-atmosphere global non-hydrostatic equations. Q.J.R. Meteorol. Soc. 140, 1505–1520 (2014), doi:10.1002/qj.2235
5. Schlutter, M., Philippen, P., Morin, L., Geimer, M., Mohr, B.: Profiling Hybrid HMPP Applications with Score-P on Heterogeneous Hardware. In: Bader, M., Bode, A., Bungartz, H.-J., Gerndt, M., Joubert, G.R., Peters, F.J. (eds.) Parallel Computing: Accelerating Computational Science and Engineering (CSE), vol. 25, pp. 773–782. IOS Press (2014)
6. Knüpfer, H., Brunst, J., Doleschal, M., Jurenz, M., Lieber, H., Mickler, M., Müller, S., Nagel, W.E.: The Vampir Performance Analysis Tool-Set, pp. 139–155. Springer, Heidelberg (2008)
7. Thakur, R., Gropp, W., Lusk, E.: Data sieving and collective I/O in ROMIO. In: The Seventh Symposium on the Frontiers of Massively Parallel Computation, Frontiers 1999, pp. 182–189. IEEE (February 1999)