Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Porting and Evolution of THETIS on the Curie Supercomputer

Stéphane Glockner[b], N. Audiffren[a,*], H. Ouvrard[b,a]

[a] CINES, Centre Informatique National de l'Enseignement Supérieur, Montpellier, France
[b] I2M (Institut de Mécanique et d'Ingénierie de Bordeaux)
*Corresponding author : audiffren@cines.fr

Abstract

It has already been shown that the numerical tool Thétis based on the resolution of the Navier-Stokes equation for multiphase flows gives accurate results for coastal applications, e.g. wave breaking, tidal bore propagation, tsunamis generation, swash flows, etc. [1,2,3,4,5,6]. In this study our goal is to improve the time and memory consumption in the set-up phase of the simulation (partitioning and building the computational mesh), examining the eventual benefits of an hybrid approach of the Hypre library, and doing fine tuning in implementation of the code on Curie Tier-0. We also implement parallel POSIX VTK and HDF5 I/O. Thétis is now able to run efficiently up to 1 billion mesh nodes at 16384 cores on CURIE in a production context.

1. Introduction

Numerical simulation of air/water coastal applications are still a very challenging goal to achieve since small interface deformations, air entrainment, very active turbulence and vorticity generation are involved. The accuracy of the whole physical process description is closely linked to the mesh grid size. Parallel computing will enhance access to a better level of description of the turbulent behaviour of the entrained and rising air bubbles, provided refined mesh grids are used to ensure an accurate free surface description.

The Thétis code has been parallelized a few years ago using domain decomposition and MPI libraries. It is linked to the massively parallel solver and preconditioner Hypre library. Weak scalability is good up to 1 billion mesh points at 16 384 cores (see Fig. 1), but before this Preparatory Access Project PA0937, some developments were still needed before envisaging to use the code in a production context. In this project, we improved the partitioning step that was too memory and CPU time consuming. We also improved the I/O thanks to parallel POSIX and HDF5 approaches, and did fine tuning in the implementation of the code on the Curie Tier-0 system.

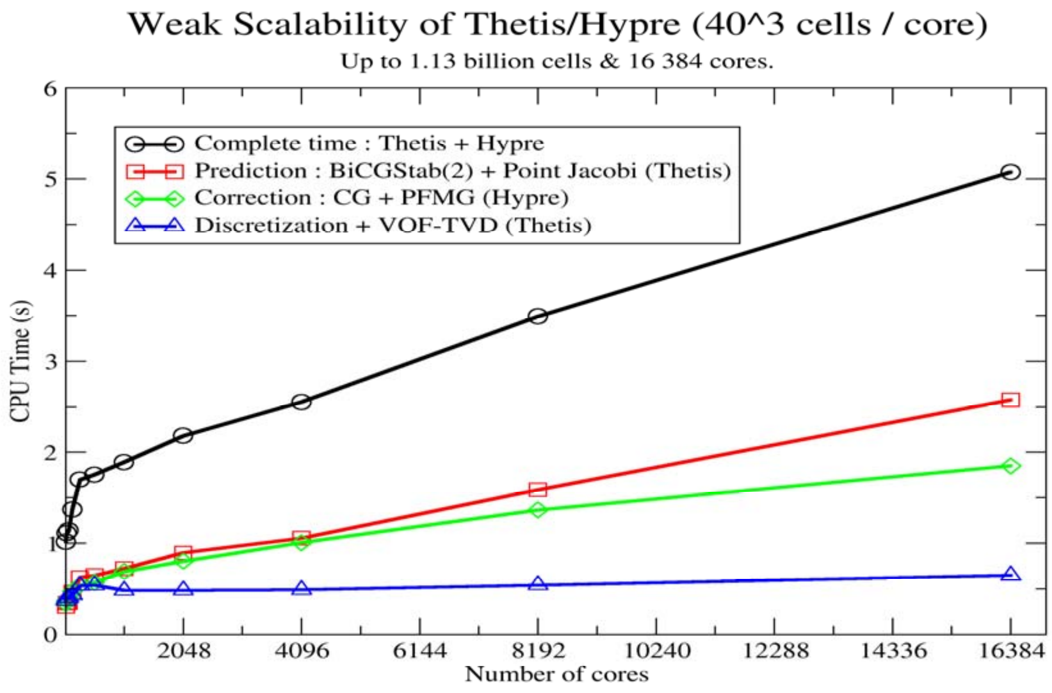


Fig. 1: weak scalability study of Thetis and Hypre with 40x40x40 mesh cells per core (up to 1.1 billion cells at 16 384 cores on Curie). The times given in the table are the maximum (among all the cores) elapsed wall-clock given by MPI_WTIME in different part of the code : (1) the time loop (Thetis+Hypre) ; (2) the prediction step of the time loop (Hypre) ; (3) the correction step of the time loop (Hypre) ; (4) other parts of the time loop (Thetis). We have (1)=(2)+(3)+(4)

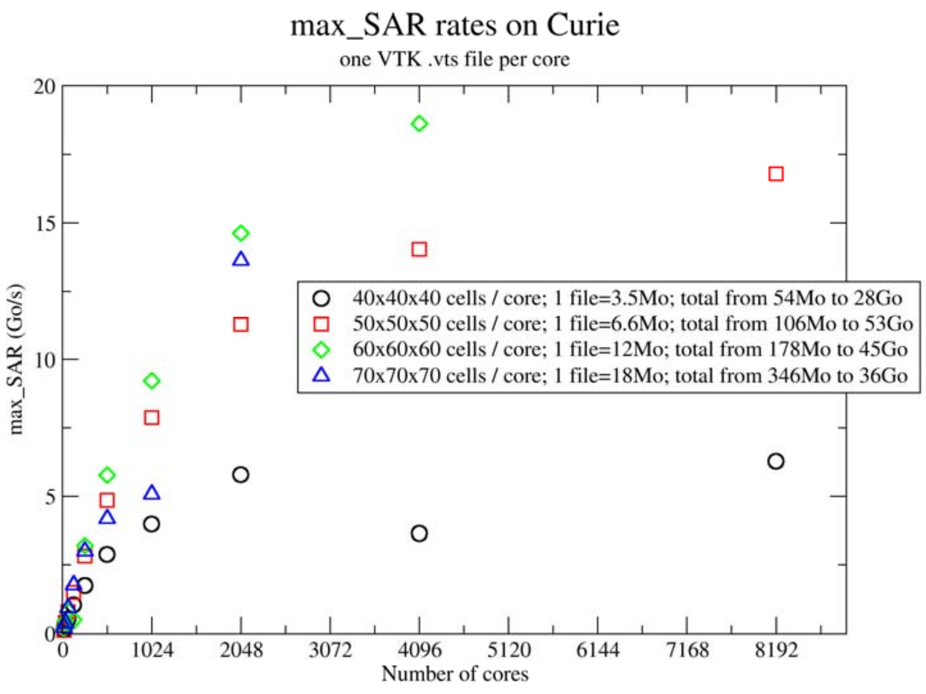
2. Partitioning

As a starting point the Scalasca tool was used to analyse the code. It confirms good load equilibrium between process and scalability measured by mpi_wtime. It was helpful to improve CPU time for three time-consuming subroutines.

We first optimized the partitioning step of the code, which, for some parts of it, is still sequential. Improvements have been achieved to decrease memory usage and CPU time. This was realized up to 40% and 20 % regarding CPU time and memory respectively. Moreover, a new special check point / restart procedure has been created which skips the partitioning step for a new job when the mesh size and the number of cores do not change. We also improved 3 subroutines regarding CPU time: resolution of the advection equation thanks to the VOF-TVD method (20%), VOF-PLIC (5%) and the subroutine of the matrix preparation for the Navier-Stokes equations (2%).

3. Parallel I/O

We first perform parallel I/O in a POSIX mode for Paraview/Visit thanks to VTK .pvts and .vts files for structured grids. Figure 2 shows the good performance rates obtained in such manner, from 40^3 to 70^3 grid cells per core and from 16 to 8192 cores. The values min_SAR and max_SAR refer to the rates defined in [7]. Max_SAR is the maximum sustained aggregate rate when *p* processors are doing I/O and min_SAR refers to the minimum sustained aggregate rate which can reflect the slowest processor to commit I/O.



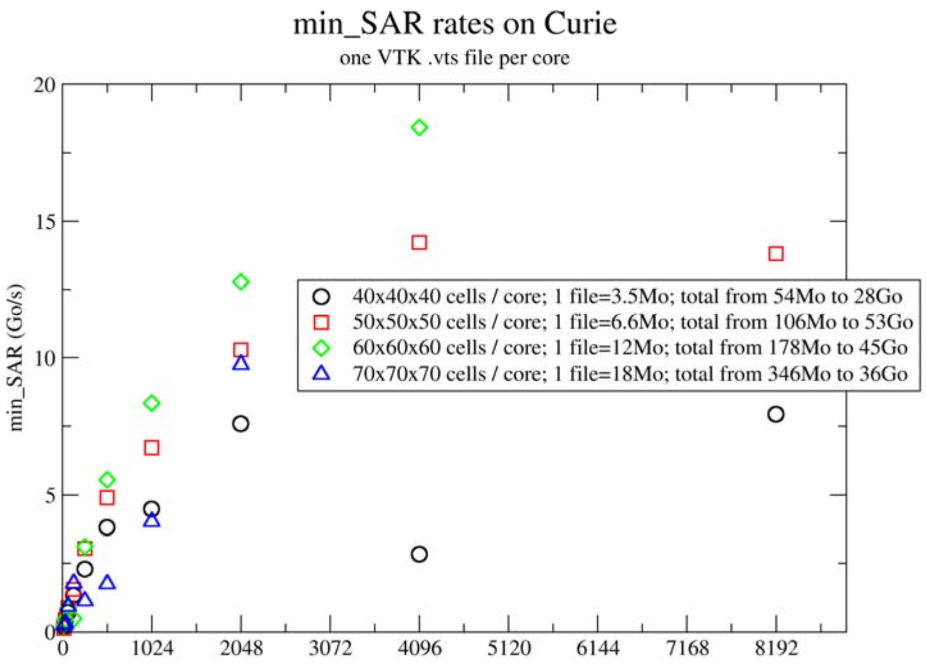


Fig. 2: maximum and minimum sustainable rates in POSIX I/O of THETIS.

But some users of Thétis code also need to visualize results using the Tecplot software which do not support VTK files. Since this software can read HDF5 files, we have implemented such an output. The measured rates in the MPI I/O writing case are still min_SAR and max_SAR. When best tuning is achieved, both are nearly identical. However, we aimed to test whether writing a shared file in HDF5 format composed of 8 to 11 scalar fields would be useful and tractable. We used the HDF5 interface and MPI collective I/O. We expect some lack of performance compared to the POSIX approach as pointed out by several publications including [8]. We face that a small amount of data has to be written by each process (from 2MB (40^3 case) to 11MB (70^3 case)) and that numerous processes have to do that (1024 to 8192). The total file size varies according to the number of cores because we are in conditions of weak scaling. The range goes from 2GB (1024 cores) to 32GB (4096). In fact, the file size also depends on the amount of mesh points per tasks. We focus on the configurations given in Tab. 2 representative of production runs.

Mesh points per core /nb of cores	1024	2048	4096	8192
40x40x40	2GB	4GB	8 GB	16 GB
50x50x50	/////////	8 GB	16 GB	/////////////
60x60x60	6,6 GB	13 GB	/////////////////	/////////////
70x70x70	11 GB	22 GB	/////////////	/////////////

Tab. 1: HDF5 shared-file size

During collective writes a two-phase I/O is performed. The first one is communication between processes making offsets, strides and counts specific to a process known to all others. The filling of the temporary buffer is also done during this phase. The second phase is I/O.

The default mode for collective buffering is automatic which means that ROMIO will use heuristics to determine when to enable optimization. The buffer size is 4MB by default. The advantage of this method is to avoid locks on the file [9].

Although collective I/O was our choice we also tuned the user-controllable hints concerning data sieving because we perform small I/O amounts per I/O request and that data sieving is used in the second phase of collective I/O in order to decrease the number of requests [10].

Our first guess of ROMIO hints were derived from a previous study [8] but some large differences between their work and ours clearly appeared: their numbers of cores does not go over 1024 cores while we run much higher numbers of cores, and moreover their throughput per each core is very large since their shared file size is 250 GB. The reason for that is that RAMSES code used in their study owns one million of points per task whereas our range of number of mesh points per task is [40^3 -70^3] which is far less. Hence, the best settings of ROMIO hints for our code obviously differ from theirs. In particular, they do not use collective buffering.

Standalone tests with parallel HDF5 were performed prior to implementation of collective writings of unique shared file in Thétis. Hints for these tests of I/O amounts of 15MB per task allowed improving the writing rate from default (few hundreds of MB/s to 3GB/s). The best ROMIO settings for Thétis are seen on Table 2.

Table 3 sums up the results. Cb_nodes are the number of cores working as aggregators in collective I/O. By default, its value is set to the number of unique hosts in the communicator used in opening the file. Setting its value to half the number of cores or even the number of cores were tried giving no better results.

In conclusion, for moderate number of cores (1024 ore 2048) parallel HDF5 can provide a shared file at writing rates that are tractable. Tuning appeared to be essential because without it, for example in the case of 40^3 mesh points per cell and 1024 cores, min_SAR and max_SAR values go down to 152 MB/s and 256 MB/s respectively .

Among the ROMIO hints, we also confirm that the striping unit and factor are important hints as said in [8] for example. I/O chunks in Thetis are small. Even though collective MPI I/O functions allow to aggregate I/O in order to better exploit bandwith, it does not scale. MPI overheads seem to impact performance significantly with increasing number of cores by a higher amount of interprocess communication. Further investigation with additional hints specific to Lustre file system could be examined: romio_lustre_co_ratio, romio_lustre_threshold, romio_lustre_cb_ds_threshold, romio_lustre_ds_in_coll as indicated in [11].

Striping factor	Striping unit	Data sieving	Independent Write buffer size	Collective buffering (CB)	CB_block_size	CB_buffer_size	Cb_nodes
40*	16 MB**	enable	16 MB	**automatic**	No value assigned	No value assigned	No value assigned ***

- * Striping factor is 80 for the case 70^3 with 2048 cores
- ** 8 MB is used for the smallest number of mesh points i.e. case 40^3
- *** Usually the default (cb_nodes=number of nodes) gives best results except for 8192 tasks.

Tab. 2: ROMIO hints used for Thétis parallel HDF5 collective I/O.

Configuration	**1024**	**2048**	**4096**	**8192**
40^3	2.1 GB/s	1.15 GB/s	0.484 GB/s	0.362 GB/s
50^3	////////	2.1 GB/s	0.895GB/s	////////
60^3	0.770 GB/s	2.6 GB/s	///////	/////////
70^3	3.04 GB/s	2.6 GB/s	////////	/////////

Tab. 3: Thétis I/O with parallel HDF5 and ROMIO tuning as indicated in Tab2.

4. Use of shared memory Hypre library

We used three tools to analyze the code: scalasca, tau and hpctoolkit, all based on the PAPI tools. They showed that 43% of the total cycles are spent in the solver BICGSTAB associated with the multigrid preconditioner called in the correction step of the predictor-corrector method used by the pressure correction method to solve the velocity pressure coupling of the Navier-Stokes equations.

Given 1024 cores we compare the case of a pure MPI run with the case of an hybrid run with 4 threads and 256 MPI tasks. Tab. 4 shows the results.

Mode	**Duration of the time loop**	**Correction step**	**Time spent outside Hypre subs**
Pure MPI (1024 tasks)	2.48 s	0.69 s	1.78 s
Hybrid mode (256 tasks, 4threads per task)	2.80 s	0.69 s	2.2 s

Tab 4: Comparison of MPI and MPI/OpenMP Hypre runs

5. Study of the placement of MPI tasks across the nodes

Placement of MPI tasks can be derived directly from SLURM using the command ccc_mprun which distributes mpi tasks by node and binds them to core. This approach results in a placement of consecutive mpi-rank tasks on different nodes. On the other hand, one can leave aside this SLURM-derived placement of tasks and choose to perform the placement of the tasks using the BullxMPI mpirun command. The next figure shows the different possibilities of distribution and binding up to 4096 cores and provides the comparison to ccc_mprun - derived placement.

Only beyond 8192 cores is the SLURM placement valuable. At current number of cores for Thétis runs it is worthy to choose a distribution of task “by core” and binding their memory to core or to socket.

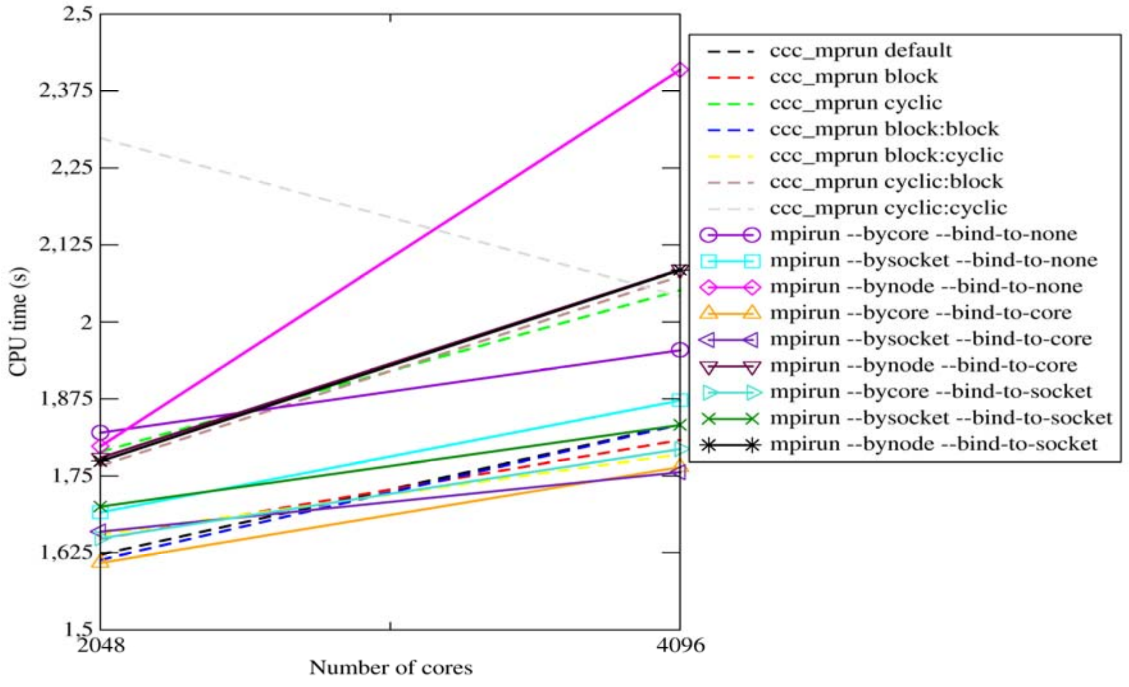


Fig 3a: Several methods of MPI tasks placement for THETIS runs of 2048 or 4096 cores

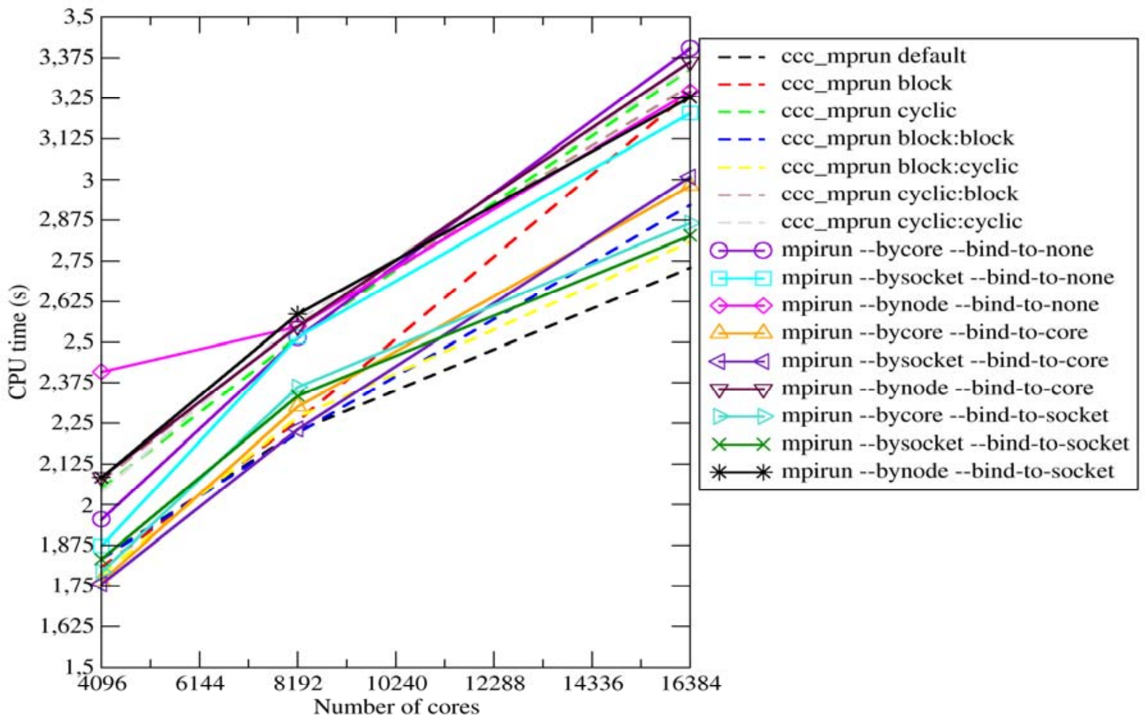


Fig 3b: Several methods of MPI tasks placement for THETIS runs of more than 4096 cores

6. Conclusion

Significant improvements have been achieved in the partitioning phase which is the most time consuming part. This PA achieved its goal on this point.

We studied the OpenMP/MPI approach which is implemented in the Hypre library. Various tests were done on fat and thin nodes of CURIE without any real improvement compared to the pure MPI approach. MPI runs are much better on the thin nodes. Moreover, the overheads induced by OpenMP combined with the fact that less cores are available for the part of the code out of Hypre lead us to continue on the pure MPI code version. However, with future version of Hypre and on future architectures as Xeon-phi, it would be worth while to look at it again.

We also demonstrated that one must pay attention to the placement of tasks. For moderate numbers of tasks, SLURM-derived placement is not the best one can get but it is optimal at very high numbers of tasks. However, production runs with this code will use moderate number of cores.

Finally we also improved the I/O part of the code thanks to parallel POSIX VTK files or HDF5 files. The code is now able to run efficiently up to 1 billion mesh nodes at 16 384 cores in a production context on Curie supercomputer.

References

[1] P. Lubin, S. Glockner, O. Kimmoun, H. Branger, Numerical study of the hydrodynamics of regular waves breaking over a sloping beach. Accepté dans European Journal of Mechanics B/Fluids, 2011.

[2] P. Lubin, H. Chanson, S. Glockner, Large Eddy Simulation of turbulence generated by a weak breaking tidal bore, Environmental Fluid Mechanics, Volume 10, Number 5, pp587-602, 2010.

[3] P. Lubin, S. Glockner, H. Chanson, Numerical simulation of a weak breaking tidal bore, Mechanics Research, 37 1: 119-121, 2010.

[4] M. Mory, S. Abadie, S. Mauriet, P. Lubin, Run-up flows of collapsing bores over a beach. European Journal of Mechanics B/Fluids, 30 (6), pp 565-576, 2011.

[5] P. Lubin, S. Vincent, S. Abadie, J.-P. Caltagirone, Three-dimensional Large Eddy Simulation of air entrainment under plunging breaking waves. Coastal Engineering, 53 (8), pp 631-655, 2006.

[6] S. Abadie, D. Morichon, S. Grilli & S. Glockner, Numerical simulation of waves generated by landslides using a multiple-fluid Navier–Stokes model Coastal Engineering, Volume 57, Issue 9, September 2010, pp779-794.

[7] James C. French, Terrence W. Pratt and Mriganka Das **“**Performance Measurement of a Parallel Input/Output System for the Intel iPSC/2 Hypercube”, IPC-TR-91-002. *IN Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, May, 1991.

[8] P. Wautelet and P. Kestener., “Parallel IO performance and scalability study on the PRACE CURIEsupercomputer”, White paper, Prace 2011. http:// www.prace-ri.eu/IMG/pdf/Parallel_IO_performance_and_scalability_study_on_the_PRACE_CURIE_supercomputer-2.pdf

[9] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO”. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 182-189, February 1999.

[10] R. Thakur, W. Gropp, and E. Lusk “ A Case for Using MPI's Derived Datatypes to Improve I/O Performance”. In *Proceedings of SC98: High Performance Networking and Computing*, November 1998.

[11] R. Thakur, R. Ross, E. Lus, W. Groppp, and R. Latham "Users guide for ROMIO: A High performance, portable MPI-IO Implementation", Argonne National Laboratory, May 2004, revised in April 2010.

[12] A. Baker, R.D. Falgout, T.V. Kolev and U. Meier Yang " Scaling hypre's multigrid solvers to 100,000 cores", from https://computation.llnl.gov/casc/linear_solvers/pubs/Baker-2010-scaling-hypre.pdf.

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-283493.