The Simulation Platform ParFlow

Carsten Burstedde[1,2], Jose A. Fonseca[1,2,*], Stefan Kollet[2,3]
[1]Institut für Numerische Simulation and Hausdorff Center for Mathematics,
Rheinische Friedrich-Wilhelms-Universität Bonn, Germany
[2]SFB/TR 32 "Patterns in Soil-Vegetation-Atmosphere-Systems," Universität Bonn
[3]Agrosphere (IGB-3), Forschungszentrum Jülich GmbH, Germany

Description of the Code

ParFlow [1–4] is an integrated hydrology model that simulates saturated and variably saturated subsurface flow in heterogeneous porous media. ParFlow is written mainly in `C`, with the exception of the land surface model CLM which is a `FORTRAN 90/95` code. ParFlow is built using distributed MPI parallelism and suitable to solve large scale, high resolution problems. It provides a solver for the three dimensional Richards equation [5] based on a cell centered finite difference (FD) scheme on regular Cartesian meshes. Time integration is performed with an Euler implicit method. The resulting system of algebraic equations is solved by a Newton-Krylov nonlinear solver that employs a multigrid preconditioned conjugate gradient solver in the linear step. The non-linear solver and preconditioners are provided by the libraries `KINSOL` [6] and `hypre` [7], respectively.

Visualization of output data sets is possible with VisIt [8]. To this end, the output should be written using the SILO format, which is available as direct output option when compiling the code against the external dependency SILO [9]. SILO is effectively a serial library, which implies that a workaround has to be activated to write output in parallel. With ParFlow, it is possible to divide the MPI processes into N groups and write a separate SILO file for each group. Within each group, the processes write to a single file, where one and only one process performs write access to the group's file at any given time. Hence, I/O is serial within a group and parallel across groups. This technique is called Poor Man's Parallel I/O (PMPIO).

ParFlow also supports distributed output of data sets in a binary format, which can be translated to `vtk` or SILO formats using post-processing tools. Since these tools work in serial, the approach is not advised when the number of processes is large.

The ParFlow code has been proved to execute on machines up to 32k processes (MPI ranks), and good parallel efficiency (without I/O) has been reported for uses up to 16k processes [10]. The version of ParFlow that we have been developing in the DFG SFB/TR32 project D8 can be run to the full size of JUQUEEN at 458k cores with good parallel efficiency [11]. The main change introduced in our modified version is the integration of the parallel adaptive mesh refinement library `p4est` [12, 13] as its new mesh manager.

Results

Our main goal for the Extreme Scaling Workshop 2017 was to investigate scalability and speed of the SILO output for a large scale and high resolution simulation. The numerical

*Corresponding author: `fonseca@ins.uni-bonn.de`

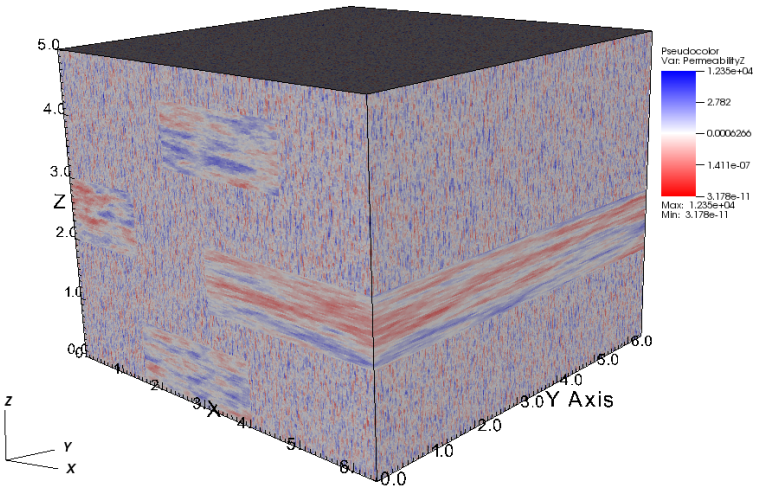


Figure 1: Output permeability field from ParFlow's built-in parallel random field generator for a domain of $6.4 \times 6.4 \times 5.0$ cubic meters. Color code scale with values ranging from $3.17 \cdot 10^{-11}$ (red) to $1.23 \cdot 10^{4}$ (blue).

experiment chosen solves an infiltration problem on a Cartesian domain. The initial water table is implemented as constant head boundary at the bottom of the domain with a five meter unsaturated zone on top of it. The heterogeneous permeability parameter is simulated with a spatially correlated log-transformed Gaussian random field. Realizations of this field are obtained with a parallel random field generator implemented in ParFlow. We show an example result in Figure 1. We aimed at testing the performance of the aforementioned random field generator and additionally to gain knowledge about how we would benefit from parallel visualization of our data sets using the JURECA visualization nodes.

Specific goals

In summary, the points we were planning to investigate during the scaling workshop are:

1. Evaluate performance of the parallel random permeability field generator.

2. Measure parallel performance of the SILO output system.

3. Find optimal choices for N in the PMPIO setting for JUQUEEN.

4. If the schedule allows it, consider a simulation domain of $50\,\mathrm{m} \times 50\,\mathrm{m} \times 5\,\mathrm{m}$ discretized at 1 cm. We estimate the output file of such a simulation to have a size of 100 GB. It is unclear at this point how this can be visualized efficiently.

In all our runs we will consider 16 cores per node without multi-threading. Concerning item 4., we would set up a series of experiments with increasing domain size, until we hit a practical limit or the target simulation domain is reached.

bg_size	rpn	MPI ranks	Random generator run time (s)
1024	16	16384	122.33
2048	16	32768	167.28
4096	16	65536	151.25

Table 1: Strong scaling exercise for ParFlow's parallel random generator routine. The first and second columns show the number of JUQUEEN nodes and ranks per node requested, respectively.

Outcome of our tests

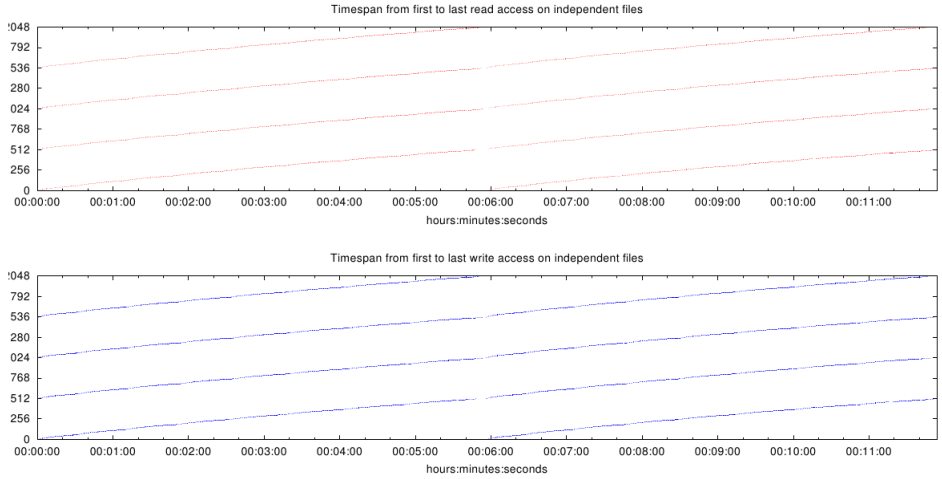
We evaluate the performance of the parallel random field generator in order to decide if we should keep it enabled for the tests of the SILO output. We executed a short strong scaling exercise with a Cartesian domain of dimensions $6.4 \times 6.4 \times 5$ cubic meters discretized at 2 centimeter scale using one, two and four JUQUEEN racks. The results are summarized in Table 1. They show that the run time does not benefit from increasing the core count.

The results shown in Table 1 motivated us to set up a new configuration to evaluate the performance of the SILO output in which the parallel random generator is disabled. For the rest of the workshop we worked with a simplified test case in which two time steps from ParFlow's Richards solver are executed and the Gaussian random field is not required. Realistic simulations require hundreds to thousands of time steps. The output is usually written at the very end of the simulation or every N time steps in order not to overload the file system.

During the workshop we noticed that the overall performance of ParFlow degraded when the SILO output was enabled. To investigate this further, we instrumented the code with the I/O characterization tool Darshan [14]. A first test of the PMPIO feature using 2048 processes and $N = 4$ showed that the code was reading as much data as it was writing. This behavior was unexpected, since the only data that ParFlow should read in this case is a small configuration file at the beginning of the simulation.

The explanation that emerged after discussing with the workshop staff was that the serialization within the $N = 4$ groups was responsible for this behavior. The additional overhead in the overall run time seems to stem from the fact that within each group, the last process is allowed to access the file in a time proportional to the number of processes in the group. Within each group, once a process finishes writing data, immediately after the subsequent process executes a read operation and then writes its own data. Figure 2 shows an extract from the Darshan report that drove us to this conclusion. A heuristic suggested by the organizers was then to increase the number of groups to match the number of requested nodes. With this idea in mind, we performed a weak scaling exercise with process counts from 4096 up to 131072 (8 racks). The results are summarized in Table 2, showing that I/O consumes over 80% of the overall run time. The I/O timing results are far from the expected weak scaling behavior and grow with the number of procesess, this is particulary evident when running from 2 to 8 racks. Increasing the number of groups seems to improve the performance compared to the initial test with $N = 4$ in which the I/O execution took 98% of the total run time. The I/O subsystem has previously been executed on at most 32k processes. Since it is only now possible to experiment with larger processes counts, it will be important to work on its scalability. On the other hand, the code has the potential to write more data than can possibly be analyzed, so the user is generally prompted to use the I/O capabilities mindfully.

Regarding parallel visualization of our data sets, a JURECA account for visualization has been activated and we had productive conversations with the staff at the supercomputing center. We learned about the workflow of using the visualization nodes with Paraview. We



Data Transfer Per Filesystem

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/work	1082.34070	1.00000	865.19118	1.00000

Figure 2: Snapshot from a Darshan report of the workload for a ParFlow test case with SILO output enabled for two timesteps and using 4 PMPIO groups. The diagrams on top show the timespan from last read (top, red colored lines) and write (bottom, blue colored lines) access on independent files, for 2048 MPI processes arranged on the vertical axes. The table on the bottom shows that the application wrote around 1 GB and read 865 MB of data.

bg_size	rpn	MPI ranks	Total Time (s)	Solver Time (s)	I/O Time (s)
128	16	2048	26.8746	2.97640	22.89820
256	16	4096	26.5785	3.03480	22.58120
512	16	8192	30.6826	3.55589	26.19600
1024	16	16384	30.7140	3.50900	26.26570
2048	16	32768	30.8680	3.89190	26.03880
4096	16	65536	45.7541	4.29419	40.42300
8192	16	131072	87.5470	5.01709	81.38050

Table 2: Weak scaling exercise for ParFlow with SILO output enabled. The scaling behavior of the solver is affected by I/O initilization, which makes the scaling look less ideal than the pure solver performance reported in [10, 11].

could not yet test the remote use of VisIt as required for the SILO file format written by ParFlow.

Conclusions

This workshop allowed us to test the performance of components of the ParFlow code that had not been evaluated previously for high process counts. With the information collected from the Darshan profiler, we conclude that additional effort should be invested into improving the I/O performance of the code, since the options currently available produce excessive overhead and limit the size of simulations that can be analyzed visually. Supposing we will be able to write large data sets with acceptable overhead in the future, we believe that remote parallel visualization of the data is a promising technology to analyze our results.

References

[1] Ashby, S. F., and R. D. Falgout (1996), A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations, *Nuclear Science and Engineering*, *124*(1), 145–159.

[2] Jones, J. E., and C. S. Woodward (2001), Newton-Krylov-multigrid solvers for large-scale, highly heterogeneous, variably saturated flow problems, *Advances in Water Resources*, *24*(7), 763–774, doi:10.1016/S0309-1708(00)00075-0.

[3] Kollet, S. J., and R. M. Maxwell (2006), Integrated surface-groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model, *Advances in Water Resources*, *29*, 945–958.

[4] Maxwell, R. M. (2013), A terrain-following grid transform and preconditioner for parallel, large-scale, integrated hydrologic modeling, *Advances in Water Resources*, *53*, 109 – 117, doi:10.1016/j.advwatres.2012.10.001.

[5] Richards, L. A. (1931), Capillary conduction of liquids through porous media, *Physics*, *1*, 318–33.

[6] Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (2005), SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers, *ACM Transactions on Mathematical Software (TOMS)*, *31*(3), 363–396.

[7] The Hypre Team (2016), ***hypre*** – *High Performance Preconditioners Users Manual*, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, software version 2.11.1.

[8] `http://wci.llnl.gov/simulation/computer-codes/visit/`

[9] `http://wci.llnl.gov/simulation/computer-codes/silo/`

[10] Kollet, S. J., R. M. Maxwell, C. S. Woodward, S. Smith, J. Vanderborght, H. Vereecken, and C. Simmer (2010), Proof of concept of regional scale hydrologic simulations at hydrologic resolution utilizing massively parallel computer resources, *Water Resources Research*, *46*, W04,201, doi:10.1029/2009WR008730.

[11] Burstedde, C., Fonseca, J. A., and Kollet, S. "Enhancing speed and scalability of the ParFlow simulation code". `http://arxiv.org/abs/1702.06898`, 2017.

[12] Burstedde, C., L. C. Wilcox, and O. Ghattas (2011), `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, *SIAM Journal on Scientific Computing*, *33*(3), 1103–1133, doi:10.1137/100791634.

[13] Isaac, T., C. Burstedde, L. C. Wilcox, and O. Ghattas (2015), Recursive algorithms for distributed forests of octrees, *SIAM Journal on Scientific Computing*, *37*(5), C497–C531, doi:10.1137/140970963.

[14] Darshan: HPC I/O characterization tool, Argonne National Laboratory `http://www.mcs.anl.gov/research/projects/darshan/`