HPSC TerrSys universitätbonn JÜLICH FORSCHUNGSZENTRUM

Implementation of parallel NetCDF in the ParFlow hydrological model: A code modernisation effort as part of a big data handling strategy

Lukas Poorthuis[1,2], Klaus Goergen[1,2,3], Wendy Sharples[1,2], Stefan Kollet[2,4]

(1) Simulation Laboratory Terrestrial Systems, Jülich Supercomputing Centre, Jülich Research Centre, Germany; (2) Centre for High Performance Scientific Computing in Terrestrial Systems, Geoverbund ABC/J, Germany, (3) Meteorological Institute, University of Bonn, Germany, (4) Agrosphere(IBG-3), Jülich Research Centre, Germany

1. Background and motivation

State-of-the-art geoscience simulations are tending towards ever **increasing model complexity**. Due to the incorporation of **multi-physics**, fully coupled model systems with **higher spatial resolutions, larger model domains** and simulations running for **longer time periods** this leads to a **big data challenge**.

This data challenge is typically characterised by TB-scale data volumes, **namely I/O**, where data variety, velocity and complexity are less relevant issues.

Within the **NIC Scientific Big Data Analytics project** "Towards a high-performance big data storage, handling and analysis framework for Earth science simulations" work has concentrated on a code modernisation effort as a best-practice example, towards "big data readiness" of geo-science simulation codes (Overpeck et al., 2011), focusing on the massively MPI-parallel hydrological model ParFlow. Here we present work that thus far has centered around the optimisation of ParFlow's **parallel I/O by implementing a NetCDF4 API**. NetCDFhas evolved as a quasi-standard in computational geosciences.

In step one a standalone C-code was used to access and test the pNetCDF and HDF5-based NetCDF4 I/O libraries, features and their parallel read and write performance. In the ongoing step two a parallel NetCDF4 API is implemented in ParFlow.

2. ParFlow model system

Integrated parallel watershed model, fully coupled dynamic 2D/3D hydrological, groundwater and land surface processes. It is written in C and FORTRAN and parallelised with MPI.

It supports:
- 3D variably saturated subsurface flow and energy transport (Jones & Woodward, 2001)
- Integrated overland flow (Kollet & Maxwell, 2006)
- Part of fully coupled multi-physics TerrSysMP via external OASIS-3 MCT coupler
- Usage across spatial scale from catchment to continent (Maxwell & Kollet, 2015)

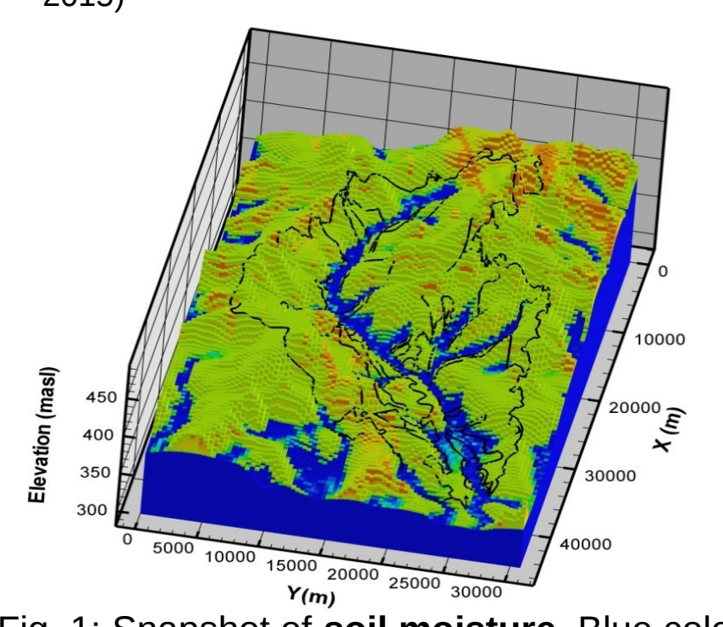


Fig. 1: Snapshot of **soil moisture**. Blue colors indicate wetter conditions along e.g. river corridors.

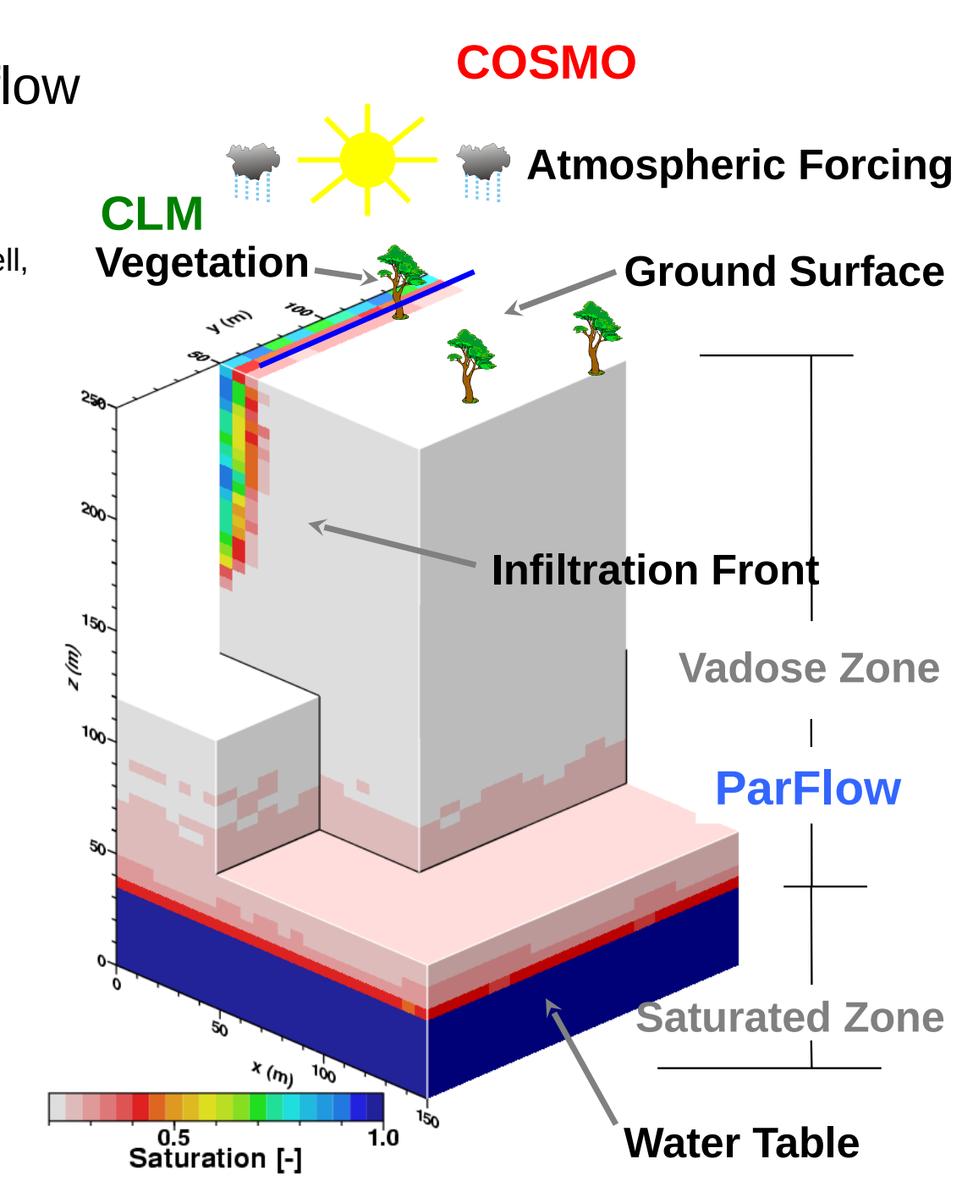


Fig. 2: Model grid showing the underground structure of ParFlow. Also shown is the impact of CLM and COSMO when coupled via OASIS 3 as part of TerrSysMP.

3. HPC system: JSC/JURECA

Fig. 3: JURECA at JSC

JURECA configuration (02/2016)
- 1,872 nodes (45,216 cores)
- Node: 24 cores with up to 48 threads
- Main memory: 271 TB
- Peak performance: 1.8 (CPU) + 0.44 (GPU) PFLOPS
- Interconnect: InfiniBand non-blocking fat tree
- CPU: Intel Xeon E5-2680 v3, 2.5GHz
- RAM: 128 GB DDR4 RAM per node

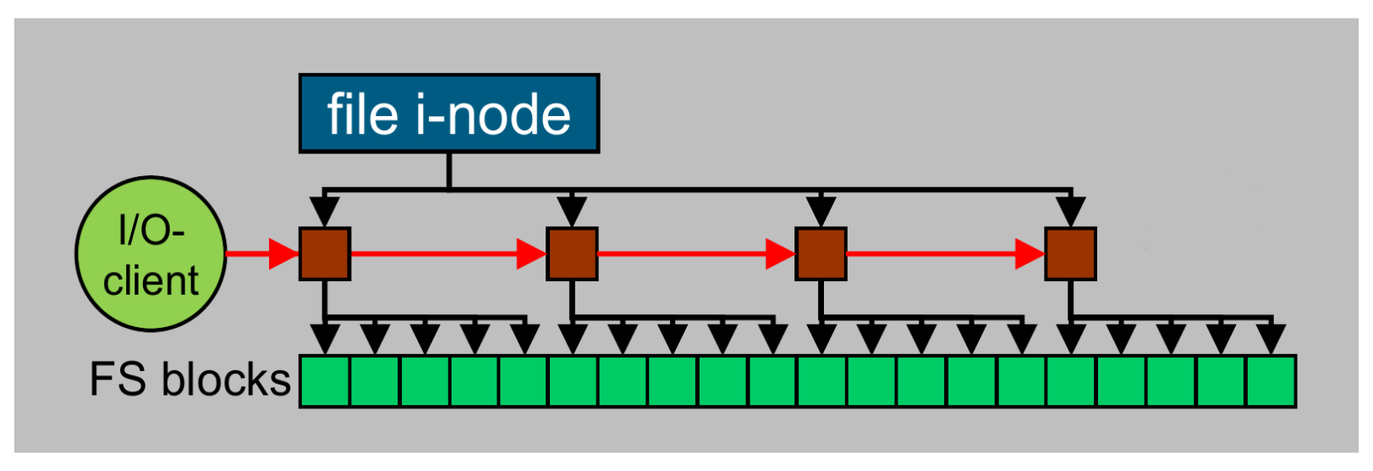


Fig. 4: Writing to GPFS from JURECA. Every node (I/O-client) has one lane to the GPFS. Communication with multiple MPI processes over this one lane will cause locks in the file system (FS) blocks which reduce I/O performance. (Courtesy: S. Lührs et al., JSC)

4. Parallel I/O libraries, implementation and usage

Tab. 1: Feature comparison between pNetCDF and NetCDF4.

	pNetCDF	NetCDF4 (only HDF5 features)
File version support	CDF-1, CDF-2, CDF-5	HDF5
API write/read modes	nonblocking, independent, collective	collective, independent
MPI I/O hints	x	x
buffering	x	x
compression		x
chunking		x

Both **pNetCDF** and **NetCDF4** provide **high-performance parallel I/O**.

The proper and efficient usage of the performance tuning features constitutes the challenge of these libraries. Especially the **chunking** feature, a process of storing multidimensional data in rectangular chunks to speed up slow file access, can offer substantial performance improvements for multi-dimensional variables.

Important considerations for the implementation and usage:
- Small write/read operations should be conducted with independent/buffered APIs to avoid wait time across MPI processes. In addition, the API should make one big I/O operation out of many small ones.
- Big chunks of data should be written in a collective fashion.
- Supercomputing resources: ParFlow is run predominantly on massively parallel HPC systems, such as JURECA or JUQUEEN.
- Limiting **I/O to one data stream per node**, accounts for network's topology, i.e., how JURECA's compute nodes are connected to the centralized storage system (here: GPFS).

5. Chunking with NetCDF4

The advantages of chunking with NetCDF4 (i.e., data structure of the NC-file during write-access) are a higher I/O performance when specific spatial subsets or time ranges are read from a multidimensional dataset.

Choosing the correct chunk size and respecting the access patterns of 1D and 2D data is most important.

This is the most general formula for optimal chunk sizes with an access pattern that is uniform for 2D and 1D data within the 3D NetCDF variable (nx*ny*nz shape):

$$nx/N^2 \text{ by } C*ny/N \text{ by } (1/C)*nz/N$$

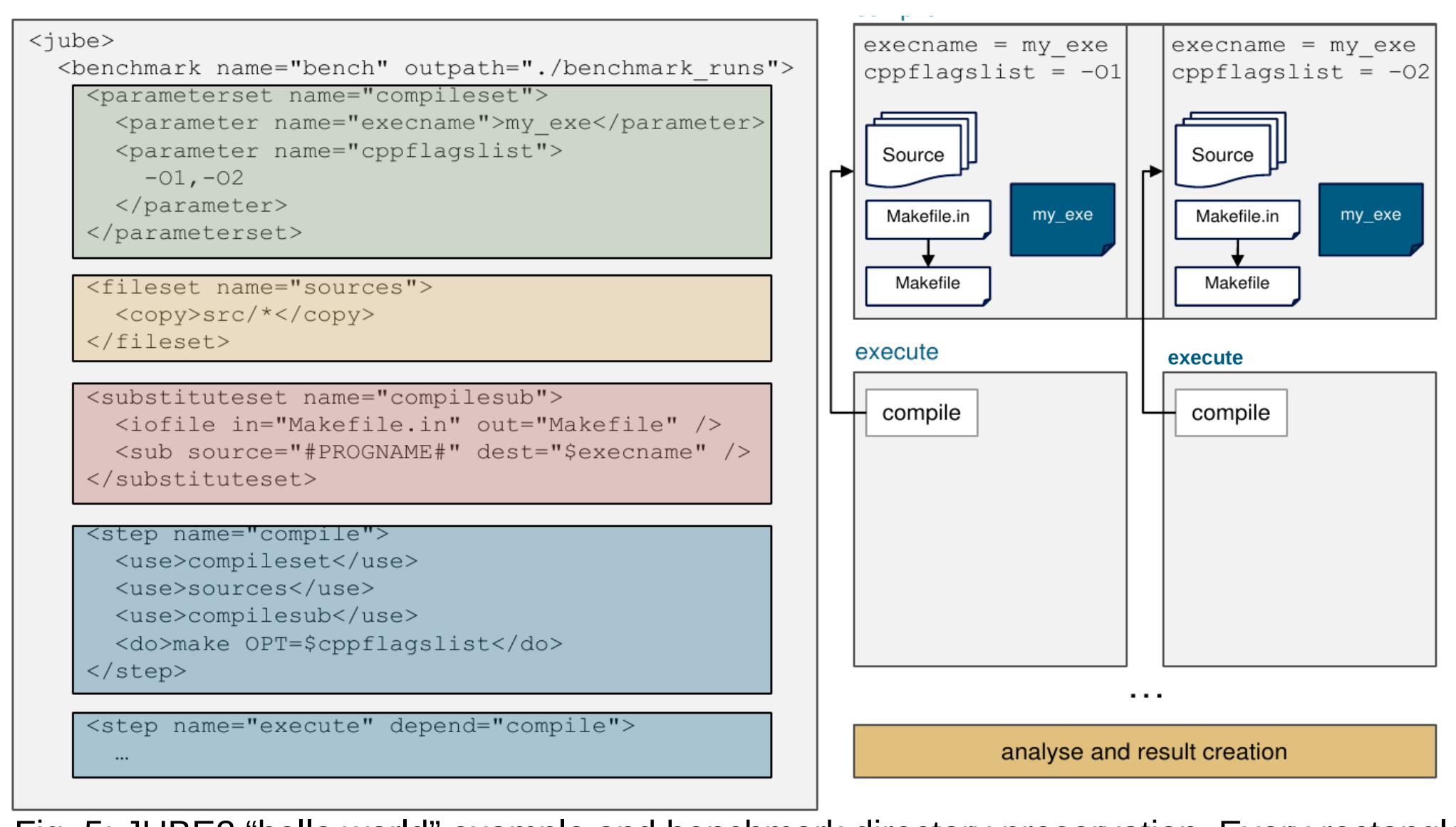
N^4=total amount of chunks
C =arbitrary positive number

Depending on the kind of read accesses to be optimised, this formula can be adapted or extended for additional dimensions.

Tab. 2: Chunk shape results (i.e., input to the NetCDF4 API) for an arbitrary 3D variable with the dimensions 98128*277*349. A shape documents how many elements from each dimension are used for the I/O operation, similar to a subvector.
The desired chunk size is chosen with respect to the filesystem characteristics. The actual chunk size approximates the desired chunk size. By setting the chunk shape, e.g. 1D as well as 2D data retrievals from the NetCDF4 file are optimized at the same time.

Desired chunk size (bytes)	Actual chunk size (bytes)	Chunk shape (values)	Number of chunks per access (time series, spatial slice)
4096	3960	33*5*6	2974, 3304
8192	7728	46*6*7	2134, 2850
16384	16384	64*8*8	1533, 1540
1048576	1032000	516*20*25	191, 196
4194304	4189920	1032*29*35	96, 100

6. Parallel I/O tests using the JUBE2 environment



```
<jube>
  <benchmark name="bench" outpath="./benchmark_runs">
    <parameterset name="compileset">
      <parameter name="execname">my_exe</parameter>
      <parameter name="cppflagslist">
        -O1,-O2
      </parameter>
    </parameterset>
    <fileset name="sources">
      <copy>src/*</copy>
    </fileset>
    <substituteset name="compilesub">
      <iofile in="Makefile.in" out="Makefile" />
      <sub source="#PROGNAME#" dest="$execname" />
    </substituteset>
    <step name="compile">
      <use>compileset</use>
      <use>sources</use>
      <use>compilesub</use>
      <do>make OPT=$cppflagslist</do>
    </step>
    <step name="execute" depend="compile">
      …
```

Fig. 5: JUBE2 "hello world" example and benchmark directory preservation. Every rectangle on the right side of the figure represents a subdirectory. For every parameter set permutation and the total amount of steps subdirectories are created that "auto-document" stderr, stdout and the explicit parameter set for the current step or test.

JUBE2 is an essential tool that allows one to execute and document a variety of tests quickly and easily, in order to benchmark code changes and assist with on going code development (Lührs et al., 2015).

Originally the JUBE2 framework was just used to conduct the I/O scaling tests and was then further extended to also support the ParFlow code base.

The currently implemented JUBE2 framework for ParFlow supports:
- Real data/idealised test cases
- Weak/strong scaling experiments
- Customisable compiler options
- Common HPC profiling tools (e.g., Score-P/Scalasca, Paraver, and Darshan I/O profiling)

7. I/O scaling test

This scaling study was conducted with ROMIO hints that enable collective buffering for the MPI I/O library, disable the built-in heuristics and set the collective buffering cache appropriate for the GPFS file system.

The results from the study show that both libraries demonstrate good scaling behaviour. However, NetCDF4 shows more consistent read performance and has a more linear scaling behaviour than pNetCDF. In addition, the data format produced by NetCDF4 is in fact NetCDF4/HDF-5 opposed to the NetCDF/CDF-5 format of pNetCDF. This makes the data produced by NetCDF4 easier to handle during post-processing. Finally, NetCDF4 supports all currently available NetCDF file formats. Therefore, it is logical to choose NetCDF4 as our parallel I/O library.

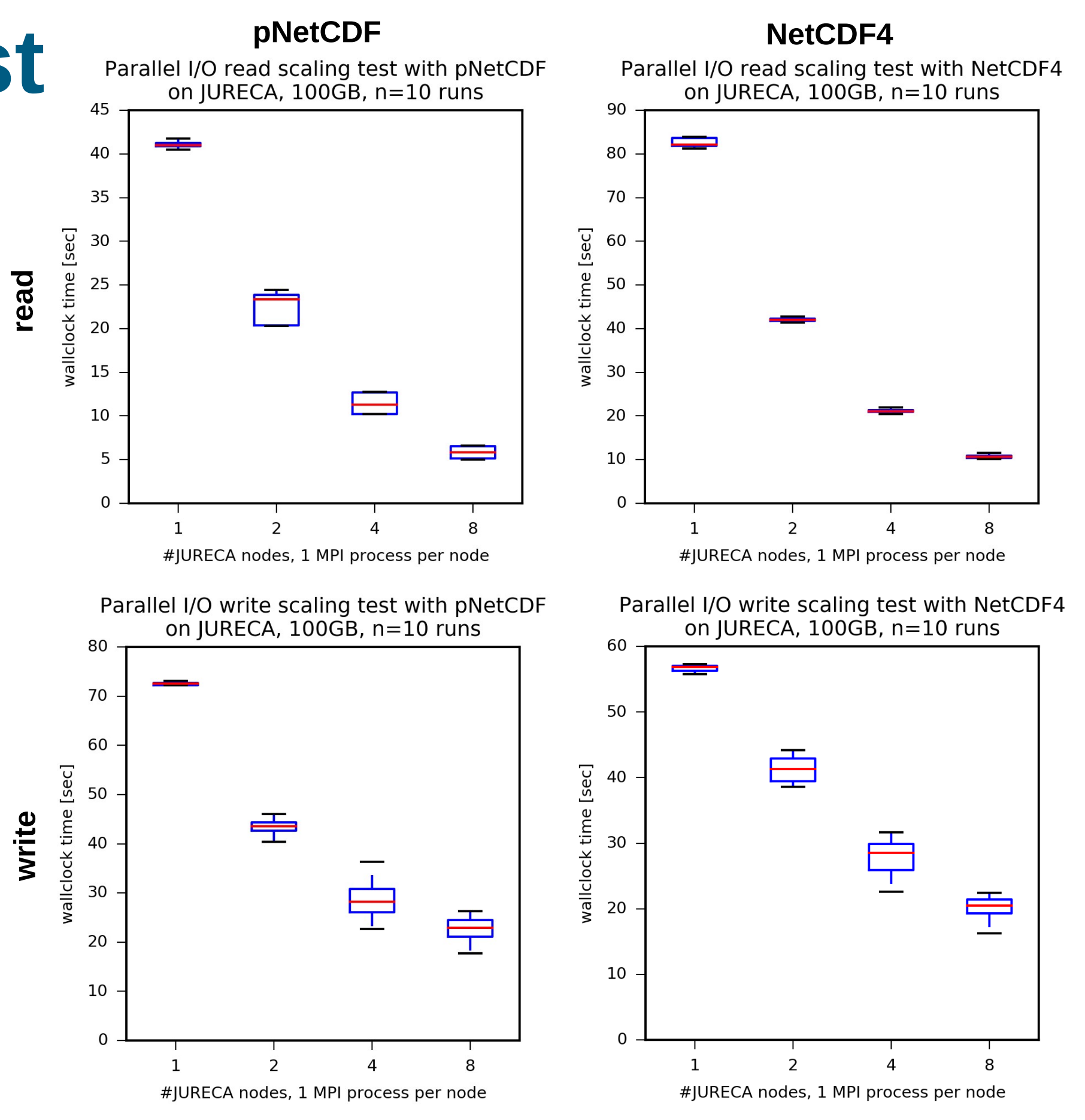


Fig. 6: Scaling plots comparing read/write performance of pNetCDF and NetCDF4 for different JURECA node counts with pI/O test framework. One I/O stream per node. Shown are the minimum and maximum I/O times (Whiskers), the median (red line) and the interquartile range (blue box). Sample size is 10 realisations. Please note the different axes ranges.

8. NetCDF4 pI/O implementation into ParFlow

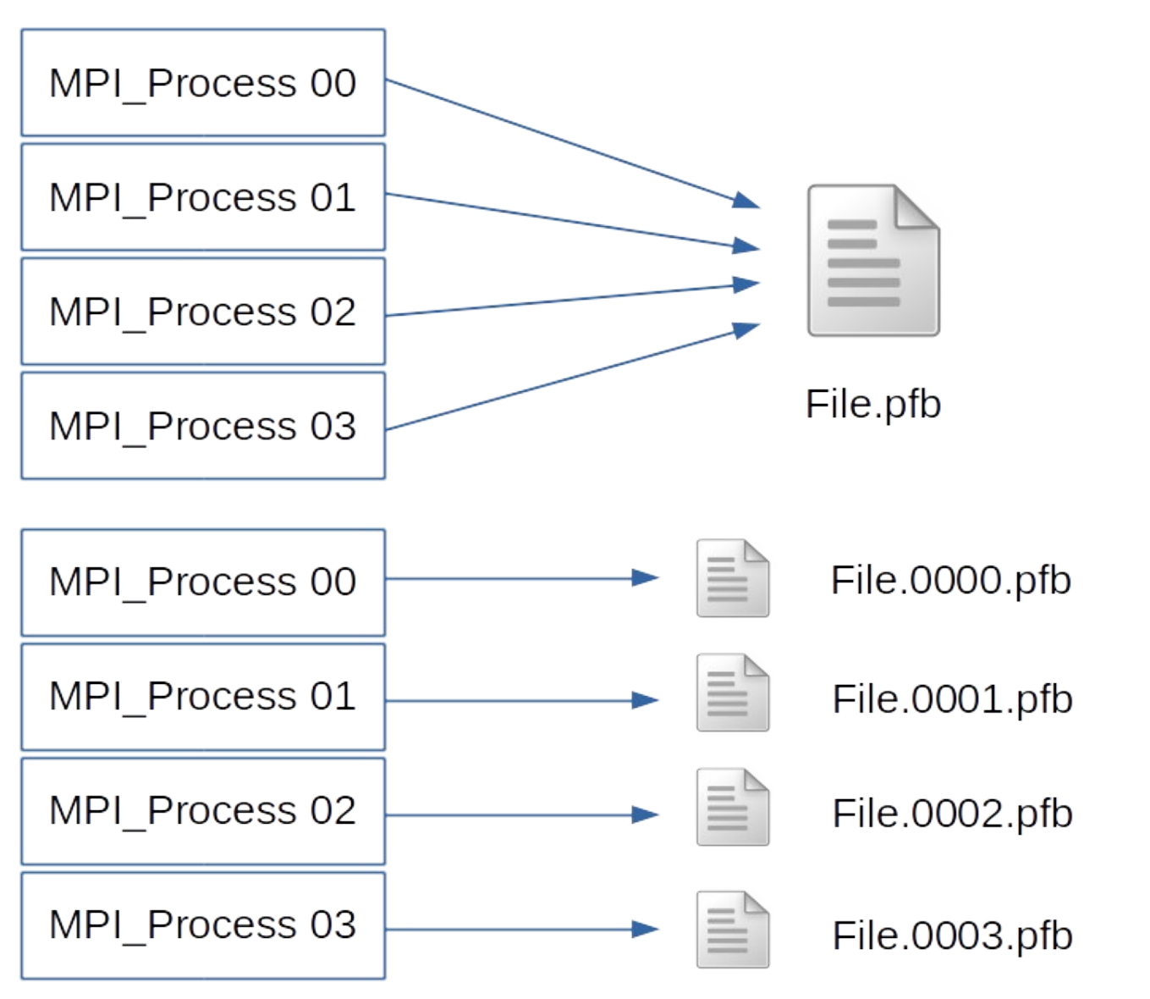


Fig. 7: Both currently available ParFlow binary output methods. Either every MPI process writes the data it produced to one shared file or every MPI process generates its own file (task local). The second step also needs post processing to merge the distributed data.

To enable I/O handling with one I/O stream per node the data management of ParFlow has to be extended. To gather the data, an additional MPI communicator has to be introduced, which only communicates on the specific hardware node.

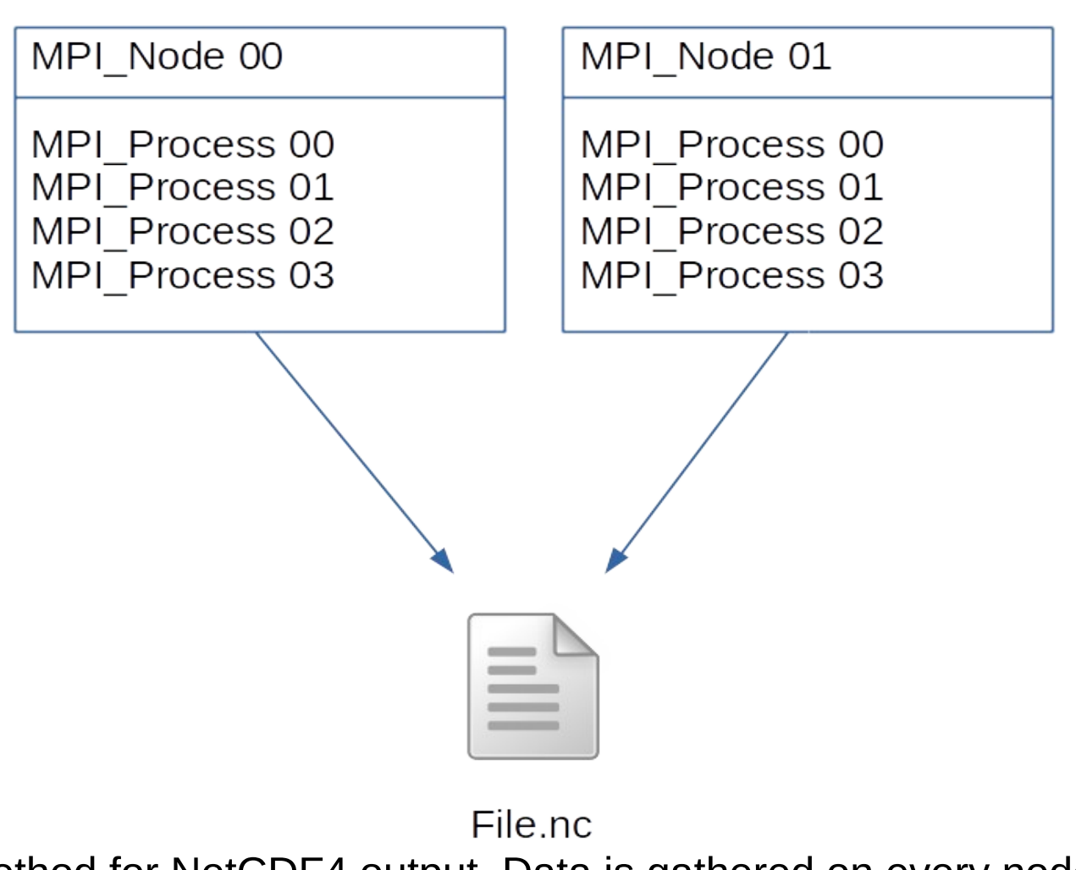


Fig. 8: Desired method for NetCDF4 output. Data is gathered on every node and is written with one I/O stream per node to one shared NetCDF4 file.

9. Outlook

- Implementation of a MPI-parallel data management to only have one data stream per node (with several MPI tasks per node)
- Replacement of the ParFlow binary output module with a NetCDF4 I/O module
- ParFlow I/O optimisation with the profiling tools
- Compression of NetCDF output
- Implementation of in-situ processing in ParFlow using VisIt on JURECA to reduce total processing time and model output data volumes

Acknowledgements

GCS/NIC project ID SBDA007 (Kollet and Goergen): The authors gratefully acknowledge the computing time granted by the John von Neumann Institute for Computing (NIC) and provided on the supercomputer JURECA at Jülich Supercomputing Centre (JSC).

References

Lührs, S., Rohe, D., Schnurpfeil, A., Thust, K., & Frings, W. (2015) Flexible and Generic Workflow Management *(in proceeding of ParCo 2015)*

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

Kollet, S. J., & Maxwell, R. M. (2006). Integrated surface–groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model. Advances in Water Resources, 29(7), 945–958. http://doi.org/10.1016/j.advwatres.2005.08.006

Kollet, S. J., & Maxwell, R. M. (2008). Demonstrating fractal scaling of baseflow residence time distributions using a fully-coupled groundwater and land surface model. Geophysical Research Letters, 35(7). http://doi.org/10.1029/2008GL033215

Overpeck, J. T., Meehl, G. A., Bony, S., & Easterling, D. R. (2011). Climate Data Challenges in the 21st Century. SCIENCE, 331(6018), 700–702. http://doi.org/10.1126/science.1197869

Maxwell, R. M., Condon, L., & Kollet, S. (2015). A high-resolution simulation of groundwater and surface water over most of the continental US with the integrated hydrologic model ParFlow v3. Geoscientific Model Development, 8(3), 923–937. http://doi.org/10.5194/gmd-8-923-2015

Contact

Lukas Poorthuis
l.poorthuis@fz-juelich.de

SimLab Terrestrial Systems (SLTS)
Jülich Supercomputing Centre (JSC)
Forschungszentrum Jülich (FZJ)
D-52425 Jülich, Germany
http://www.fz-juelich.de/ias/jsc/slts