natESM

Sprint documentation #4

ParFlow Sprint

Contact: info@nat-esm.de

Published on 29.08.23 on https://www.nat-esm.de/services/accepted-sprints.

1 Summary

The main idea of this sprint was to port and to enable HIP as a Kokkos backend of ParFlow using the eDSL (embedded Domain Specific Language) of ParFlow. After laying this foundation, a second goal was to address the memory pooling problem which is solved by the Rapids Memory Manager (RMM) in the CUDA backend of Kokkos and to do further optimizations. We found out that the seemingly easiest part of the sprint was very challenging because of build problems with Kokkos and ParFlow. After fixing these issues, ParFlow started to run but was hanging during runtime without any output or indication of failure. The application of different problem-solving attempts wasn't successful. Consequently, the sprint had to be stopped because of reaching the sprint deadline and it wasn't terminated successfully.

2 General information

Start and end date:	12/22 – 04/23
Intended period:	5 months
Real period:	6 months
Responsible RSE:	Jörg Benke (JSC, FZJ)
Responsible scientist:	Daniel Caviedes-Voulliéme (IBG-3 / JSC, FZJ)

The ParFlow hydrologic model is an integrated variably saturated groundwater, surface water flow simulator that incorporates subsurface energy transport and land surface processes through the integration of the Common Land Model (CLM). Applications of ParFlow range from fundamental science to socio-economically relevant applied studies. Spatial and temporal scales range from watershed to continental (and recently global proof-of-concept) simulations and from the event to the climate timescale. ParFlow is written in C (with additional Fortran 90 parts (especially if CLM is enabled)) and uses the parallelization methods MPI, OpenMP, CUDA and the programming model Kokkos (with the backend CUDA or OpenMP).

3 Sprint objectives

In the recent past, ParFlow's eDSL has been expanded to incorporate backends utilizing GPUs; one backend with native CUDA support [1] and one backend incorporating Kokkos [2]. ParFlow reached a higher degree of performance portability via the Kokkos backend and the additional usage of the eDSL extension. Performance and scalability of ParFlow using these two concepts have been demonstrated on JUWELS utilizing the cluster-booster next-generation hardware. However, current hardware is characterized by varying architectures. For example, the pre-exascale HPC system LUMI-G of EuroHPC is based on AMD GPU accelerators; in the US, there

are two exascale systems based on AMD GPUs (Frontier & El Capitan). The main idea of this sprint was to port ParFlow to the AMD architecture via HIP as a backend of Kokkos and using the eDSL. The three main tasks in this sprint were:

1. Porting ParFlow to AMD GPUs based on Kokkos and the eDSL (Compilation of Kokkos with HIP backend).
2. Inspection of the RAPIDS Memory Manager (RMM) and HIPification (Inspection of pool allocator for HIPification; outline of HIPification and implementation). Since the RAPIDS memory manager isn't available for HIP, other alternatives must be tested. Candidates are the Kokkos memory manager and the HIP one.
3. Performance analysis / optimization and performance and scaling tests; proof-of-concept (exascale readiness).

4 Procedure and insights

4.1 Technical Approach / procedure

The foundation was the porting of ParFlow from CUDA to HIP using the performance portability library Kokkos and the built-in eDSL of ParFlow (and the understanding of the underlying build architecture / scripts). In the first step Kokkos and ParFlow had to be built using HIP / ROCm for the AMD GPUs. This in turn first required building Kokkos with the HIP backend, after which ParFlow could be built on top. During the Kokkos build process many challenges were encountered. The build process failed for several different reasons, which partly had to do with the configuration of Kokkos, as well as our initial use of Kokkos v3.7. Adapting ParFlow to Kokkos v4.0, which now fully incorporates HIP support (i.e., not under the experimental Kokkos namespace), using a self-compiled version of HIP Fortran and manually including the path to specific include files solved the main problems. The last two points especially suggests that the build system and its interaction with the underlying software stack was not completely robust.

After successfully building Kokkos ParFlow had to be compiled. The most challenging (and time consuming) efforts were the identification of linker warnings and again problems during the compilation process. The source of the main problem was the HIP Fortran compiler and the compilation of the Fortran source within ParFlow (mostly a C code). One solution approach was to skip the Fortran part of ParFlow and to compile ParFlow only with the C compiler. But this didn't solve the problem, nor did using different pre-implemented HIP / ROCm capable Fortran compilers. The solution required us to build the HIP Fortran compiler on our own, thus avoiding implicit dependencies, after which we were able to compile ParFlow and link it to Kokkos with a HIP backend.

Nevertheless, running ParFlow on AMD GPUs at JSC's experimental hardware in JURECA-DC (with a well-established testing scenario) was unsuccessful. ParFlow failed at runtime in an apparent deadlock, i.e. the job started and initialized, but then hung at some point during the solver execution. Since JURECA-DC AMD GPU configuration is experimental, we also requested access to the LUMI-G partition which operates AMD GPUs for production, in the expectation of testing whether the problems were generated by the experimental software stack in JURECA-DC. After successfully building in LUMI-G (which also required substantial effort to configure all the necessary software), we verified that ParFlow would still hang at runtime with no apparent reason. This suggests that the problems come either from Kokkos or ParFlow, but not from the software stack.

But despite the usage of AMD tools and enabling a low-level debugging environment, we were unable to pinpoint the source of the runtime problem, since no significant indication was obtained from these tools. We hypothesize that one possible reason might be a non-closing fence in Kokkos. To solve the problem, one should use the HIP debugger as a next step which wasn't used until now because of time constraints.

Consequently, a full and successful porting to the AMD architecture is still pending. It is planned to retake this work with the debugging step and go along the original plan if the porting was successful.

4.2 General Insights

The usage of the eDSL and Kokkos allows in ParFlow in principle a very easy adaptation to different architectures and parallel models. This was the case when porting ParFlow to CUDA with Kokkos and this was also shown in principle in this sprint. But the problems arose in very technical details, which often were related to the ROCm software stack and the change from Kokkos v3.7 to v4.0. Several low-level challenges occurred which required a significant amount of time to solve. We will list these as bullet points, and possible solutions will be added in parentheses:

1. Kokkos v4.0 breaks backwards compatibility with Kokkos v3.7. The update to Kokkos v4.0 required some updates in the Kokkos API in ParFlow. This was solved.
2. Using the different tools of the Kokkos ecosystem (e.g., kernel logger of Kokkos) or AMD debugging flags in ParFlow. The Kokkos tools are very helpful in general and a lean addition to Kokkos. Unfortunately, in our case they were not working as expected since no additional output was generated. The same occurred in the case of the AMD debugging flags.
3. There were problems with the local ROCm installation. The issues were not obvious, and they required a deep dive supported by experts and very experienced users of the ROCm environment. We were able to continue the work with a workaround, but we were not able to solve the fundamental problem (both because of time reasons, and because they should be solved at a lower level).
4. HIPification via eDSL and Kokkos is in principle relatively easy. We've modified the ParFlow eDSL to enable the Kokkos API to reach the HIP backend, mimicking the approach already enabled for CUDA [2]. We were able to avoid using the HIPification tools provided with HIP which would entail hundreds of substitutions in our case. These tools are powerful, but their usage may be a bit cumbersome for large codes (especially with hipify-perl). We still consider HIPification of ParFlow with a HIP backend directly in the eDSL, but we did not address this.
5. There seem to be problems in the HIP environment because compilation with the standard installed HIPFortran wasn't possible. This was the case both at JURECA-DC and in LUMI-G, for which we had to clone HIP Fortran and needed to compile it on our own (which led us to success).

5 Results

We were able to build and install Kokkos, in particular with version 4.0 with full HIP support. We were also able to build ParFlow on top of the Kokkos-HIP dependency. We implemented the necessary code in the ParFlow eDSL to inform the Kokkos API in ParFlow with the appropriate HIP choices. But ParFlow failed at runtime with an undetermined problem (no progress in the run of ParFlow, no error report, no crash). We were able to build in both JSC and LUMI-G and reproduce the runtime problems in both systems. Despite the usage of Kokkos tools, and the AMD runtime debugging environment, we were not able to pinpoint the origin of the problem, with several candidate issues remaining to be investigated.

6 Conclusions and Outlook

The outcome of this sprint is that, in theory, the combination of Kokkos and the eDSL of ParFlow makes it relatively easy to port ParFlow to AMD with HIP as the backend. No specific issues were encountered in implementing the changes to the eDSL to reach the Kokkos-HIP backend. However, major technical challenges were encountered at several levels in the use of the ROCm

software stack. It seems that, partly, the AMD environment and the ROCm software stack is not as mature and robust as needed for our use case at the moment. This was especially notorious with the experimental hardware and software stack in JURECA-DC during the initial phase of the sprint, where we encountered issues at the time unknown by the maintainers of these resources.

To solve the above problems and to get the porting effort to a successful end, a follow-up natESM sprint application is envisaged to solve the standing issues and fully port ParFlow to AMD / HIP and to complete the announced tasks. We will not follow up immediately as we wish to gather some additional information on the software stack and potential pitfalls with Kokkos.

7 References

[1] Hokkanen, J., Kollet, S., Kraus, J. et al. Leveraging HPC accelerator architectures with modern techniques — hydrologic modeling on GPUs with ParFlow. Comput Geosci 25, 1579–1590 (2021). https://doi.org/10.1007/s10596-021-10051-4

[2] Zbigniew P. Piotrowski, Jaro Hokkanen, Daniel Caviedes-Voullieme, Olaf Stein, and Stefan Kollet. Lightweight embedded DSLs for geoscientific models. Geoscientific Model Development. Submitted (2023).