Porting the hydrologic model ParFlow to accelerator architectures using eDSL and Kokkos

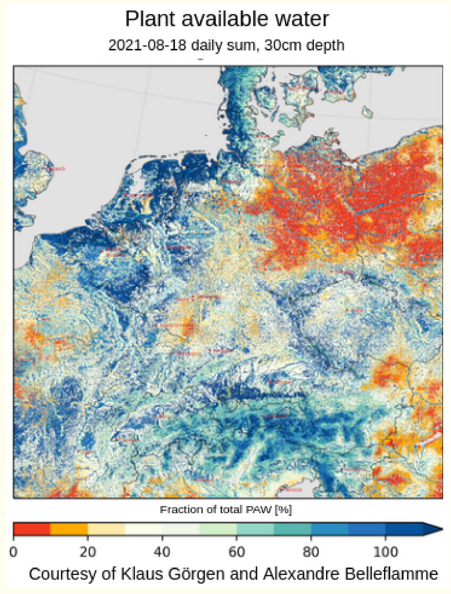
Jörg Benke[1], Muhammad Fahad[2], Daniel Caviedes-Voullieme[1,2], Andreas Herten[1], Stefan Kollet[2]

[1] Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich GmbH, Jülich, Germany
[2] Institut für Bio- und Geowissenschaften - Agrosphäre (IBG-3), Forschungszentrum Jülich GmbH, Jülich, Germany

Introduction and Background

ParFlow (https://github.com/parflow) is an open-source, modular, parallel watershed flow model which is written in C. ParFlow includes fully-integrated overland flow, the ability to simulate complex topography, geology and heterogeneity and coupled land-surface processes including the land-energy budget, biogeochemistry and snow (via CLM). ParFlow solves the 3D Richards equation and the 2D kinematic/diffusive surface flow.

ParFlow has been abstracted early on with an so-called embedded Domain Specific Language (eDSL) approach leading to a best-practice separation-of-concerns, which means the domain scientist/developer does not see for example a single MPI call when using distributed memory parallelization in ParFlow. In the recent past, ParFlow's eDSL has been expanded to incorporate backends utilizing GPUs; one backend with native CUDA support and one backend incorporating Kokkos-CUDA. During the natESM and EoCoE-III projects [2,3] Kokkos-HIP was also incorporated to achieve a high degree of performance portability.



Courtesy of Klaus Görgen and Alexandre Belleflamme

embedded Domain Specific Language (eDSL) in ParFlow - The general concept

A typical loop in 3D space spanning indices i,j,k

```
double *fp;
double *fp;
double value;
Subvector *f_sub;

/* some code missing here*/

for(k = iz; k < iz + nz; k++)
    for(j = iy; j < iy + ny; j++)
        for(i = ix; j < ix + nx; i++)
        {
            int ip = SubvectorEltIndex(f_sub, i, j, k);
            fp[ip] = pp[ip] - value;
        }
```

Same loop, abstracted into the BoxLoopI0 macro

```
double *fp;
double *fp;
double value;
Subvector *f_sub;

/* some code missing here*/

BoxLoopI0(i, j, k, ix, iy, iz, nx, ny, nz,
        {
            int ip = SubvectorEltIndex(f_sub, i, j, k);
            fp[ip] = pp[ip] - value;
        });
```

eDSL macro definition for BoxLoopI0

```
#define BoxLoopI0(i, j, k, ix, iy, iz,
  nx, ny, nz, loop_body)
{
  for (k = iz; k < iz + nz; k++)
    for (j = iy; j < iy + ny; j++)
      for (i = ix; i < ix + nx; i++)
      {
        loop_body;
      }
}
```

- ParFlow is using an embedded Domain Specific Language (eDSL).
- This approach results in a best-practice separation-of-concerns, which means that the developer does not see for example complex loops, allocation of memory or a single MPI call when programming ParFlow.

- In ParFlow the eDSL is implemented via C macro definitions
- In the left example the eDSL abstracts from a three level for loop (see left hand side box in the figure) ...
- via the C macro definition *BoxLoopI0* (see middle box).
- As a result *BoxLoopI0* with the loop body of the for loop as an argument will be called (see figure on the right).

embedded Domain Specific Language (eDSL) in ParFlow - Abstraction of CUDA/HIP calls

eDSL macro definition - sequential (host)

```
#define BoxLoopI0(i, j, k, ix, iy, iz,
  nx, ny, nz, loop_body)
{
  for (k = iz; k < iz + nz; k++)
    for (j = iy; j < iy + ny; j++)
      for (i = ix; i < ix + nx; i++)
      {
        loop_body;
      }
}
```

eDSL macro definition - CUDA (device)

```
#define BoxLoopI0(i, j, k, ix, iy, iz,
  nx, ny, nz, loop_body)
{
  auto lambda_body = [=] __host__ __device__
    (const int i, const int j, const int k)
     loop_body;

  /* some code missing for grid & block sizes */

  BoxKernelI0<<<grid, block>>>(lambda_body,
    ix, iy, iz, nx, ny, nz);
}
```

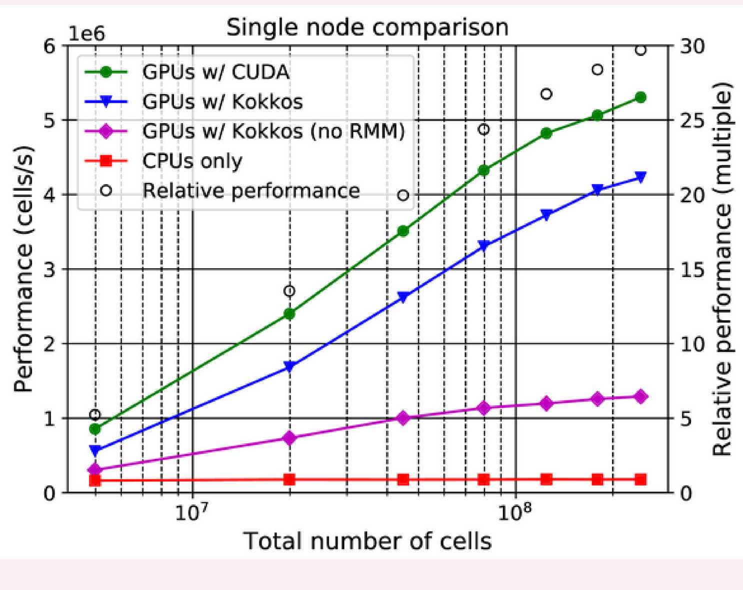
eDSL macro definition - Kokkos (host & device / sequential, parallel)

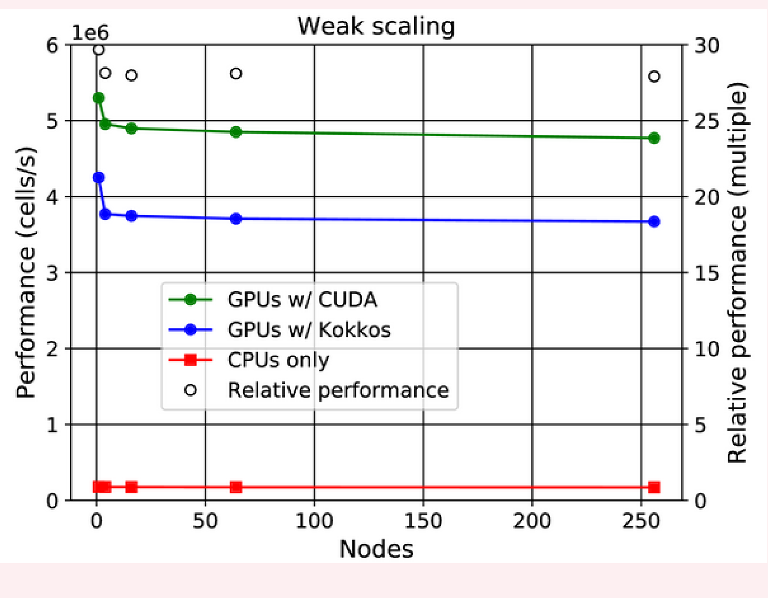
```
#define BoxLoopI0(i, j, k, ix, iy, iz,
  nx, ny, nz, loop_body)
{
  auto lambda_body = KOKKOS_LAMBDA(int i, int j, int k)
  {
    i += ix; j += iy; k += iz;
    loop_body;
  }

  MDPolicyType_3D mdpolicy_3d({{0, 0, 0}},{{nx, ny, nz}});
  Kokkos::parallel_for(mdpolicy_3d, lambda_body);
}
```

- The above shown example is also extendable to MPI calls, memory allocation and offloading.
- During the last few years ParFlow's eDSL has been expanded to incorporate backends utilizing GPUs.
- One backend with native CUDA support and one backend incorporating Kokkos with CUDA and Kokkos with HIP to achieve a high degree of performance portability.

- A successful porting to Nvidia-GPUs was done in 2021 with CUDA and Kokkos-CUDA [1].
- End of 2022 the porting to Kokkos-HIP was started in the natESM project and was completed in the EoCoE-III project.
- Several challenges occured during the natESM sprint (ROCm, Kokkos, machines)
- Solving the conflicts with internal Desul atomics and multiple calls to finalize() made ParFlow running with Kokkos-HIP.

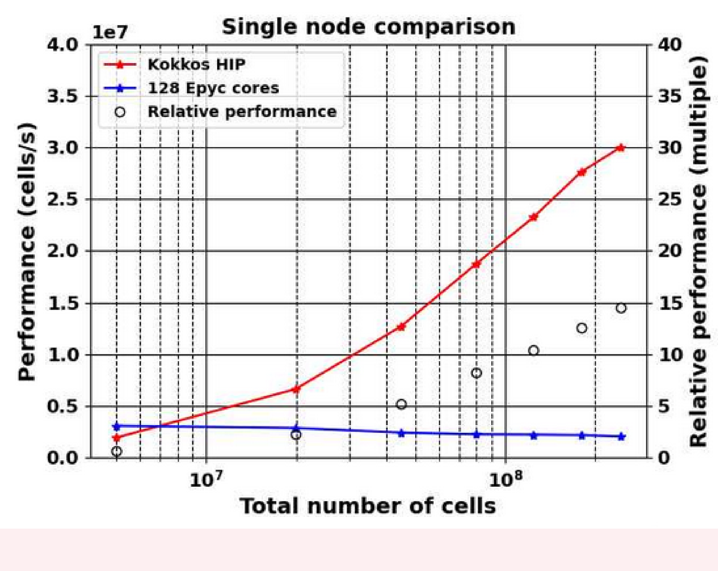
Scaling tests with ParFlow (CUDA/Kokkos)

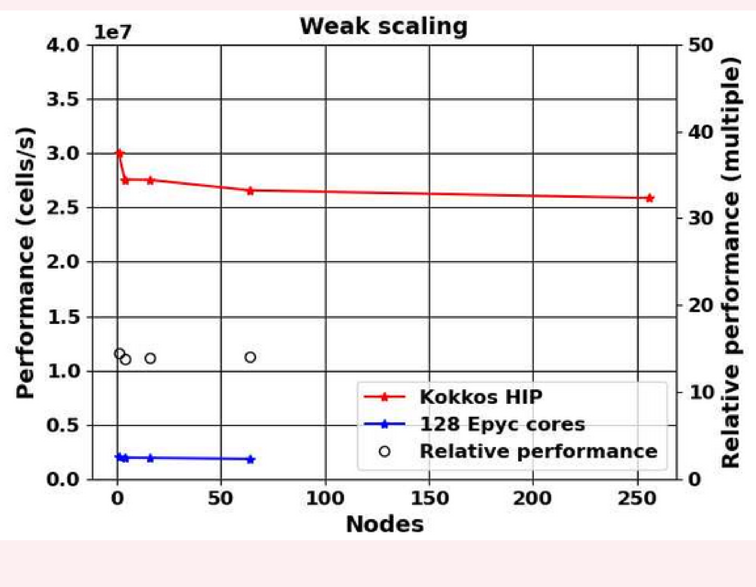




- Above strong and weak scaling plots are showing the results of CUDA and Kokkos-CUDA runs (see figure above; performed on JUWELS Booster (FZJ) with NVIDIA A100 GPUs [1]) and Kokkos-HIP runs (see right figure; performed on LUMI (Finland), AMD MI250)
- Mesh sizes (single node): 144*144*240 up to 1008*1008*240 (2.43×10^8) mesh points.

Scaling tests with ParFlow (HIP/Kokkos)





- The performance (cells per second) is calculated as total number of cells (problem size) divided by the execution time for a given datapoint.
- The relative performance is calculated by dividing the execution time on CPU by the execution time on GPU
- Using Kokkos with CUDA and HIP led to a significant speed-up with only limited costs of changing the code.

References

[1] Hokkanen, J., Kollet, S., Kraus, J., Herten, A., Hrywniak, M. and Pleiter, D.: Leveraging HPC accelerator architectures with modern techniques — hydrologic modeling on GPUs with ParFlow, Comp. Geosciences, Vol. 25, 1579 - 1590, https://link.springer.com/article/10.1007/s10596-021-10051-4, 2021.
[2] natESM (national Earth System Modeling strategy project): https://www.nat-esm.de
[3] EoCoE-III (Energy-oriented Centre of Excellence for Exascale HPC applications): https://www.eocoe.eu