Efficient Execution of Multiphysics Simulation Assembly Using Kokkos::Graph

Arnst M., Tomasetti R.

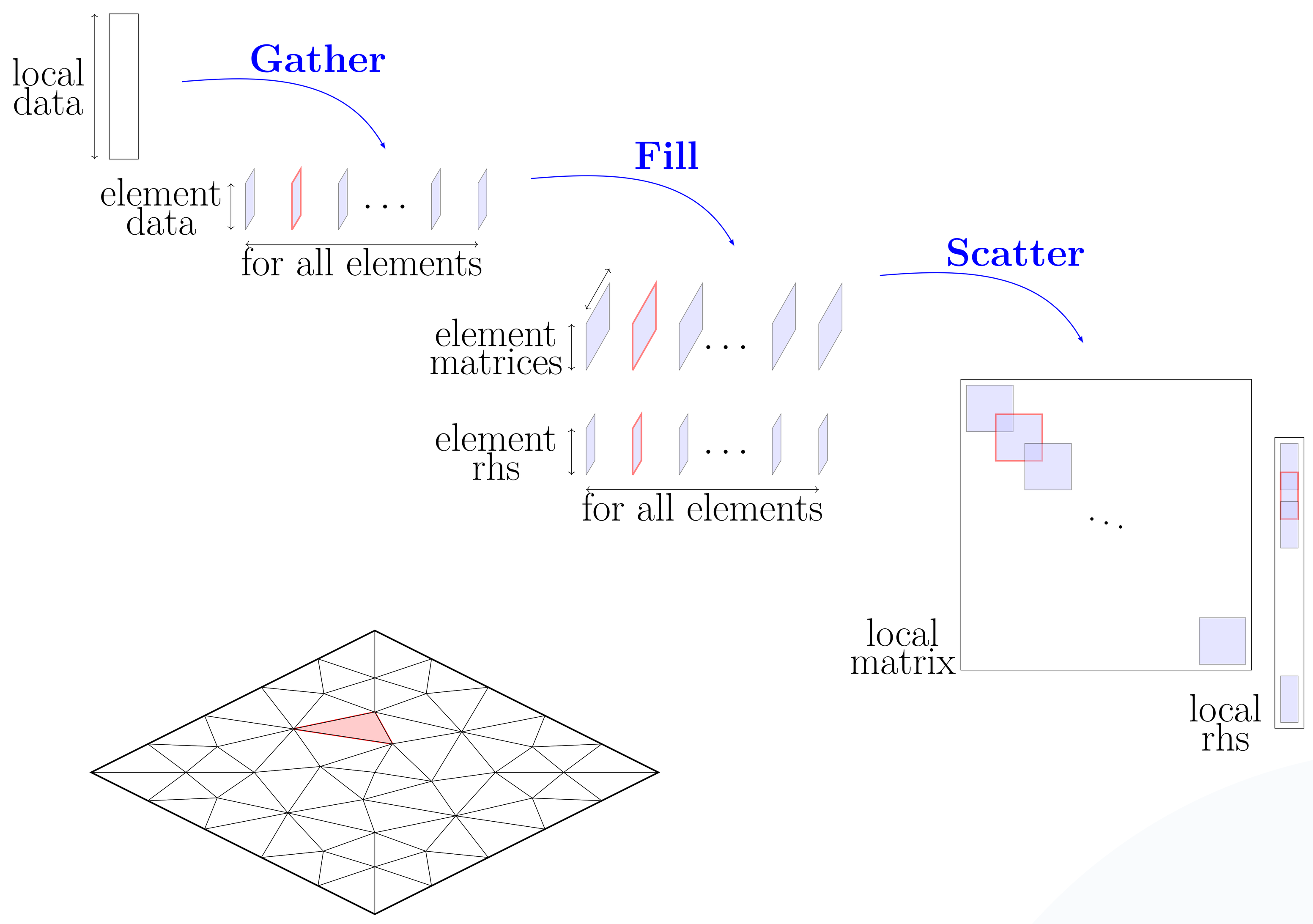
Computational and stochastic modeling, University of Liège

maarten.arnst@uliege.be

Performance portable FE code

In the `Kokkos` programming model for performance portability, parallel work is described by a functor, a pattern, and a policy.

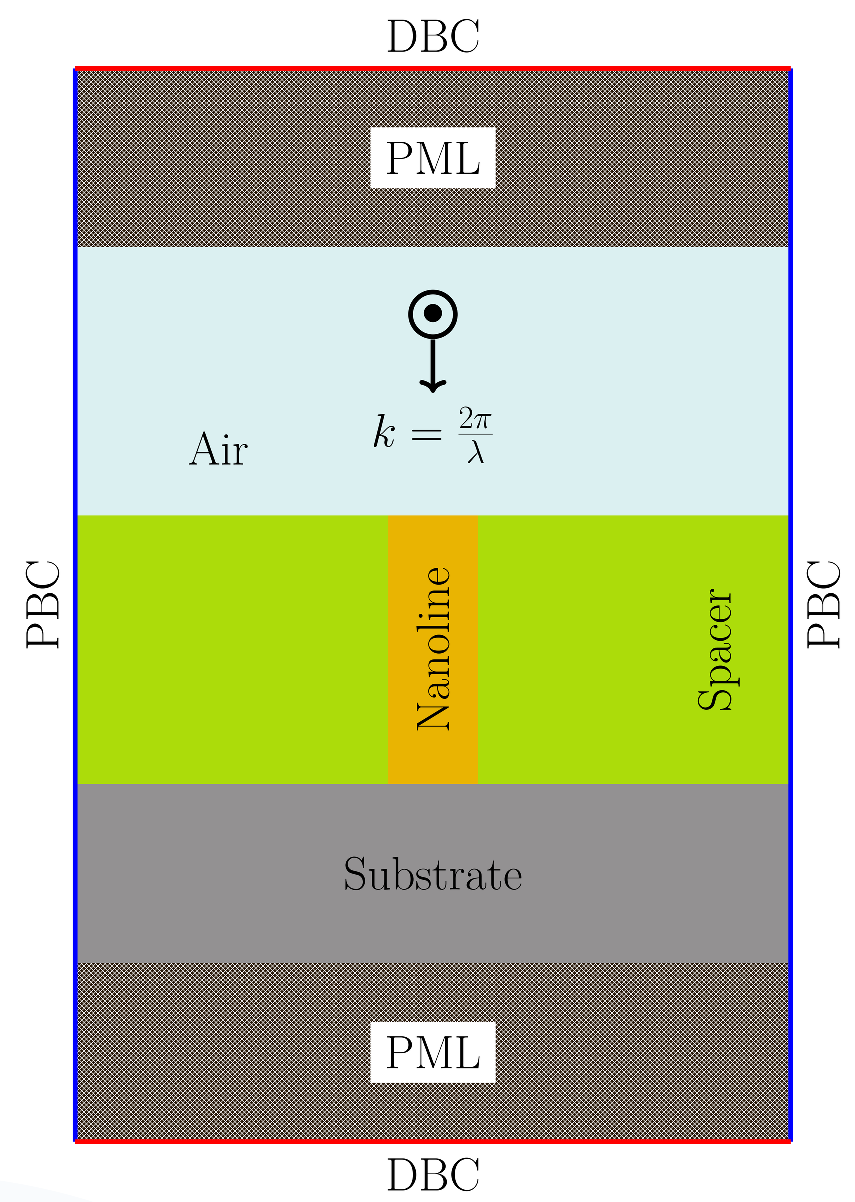
```
Kokkos::parallel_for (
    Kokkos::RangePolicy(numWorkItems) , functor
);
```



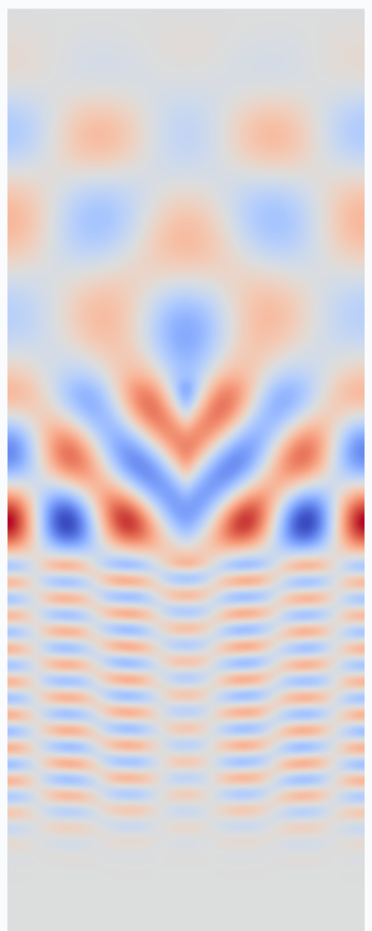
We develop a FE code based on `Kokkos` and `Trilinos`. The FE assembly involves a hierarchy of functors that carry out the elementary operations. Efficiently scheduling many parallel dispatches on device is a key challenge.

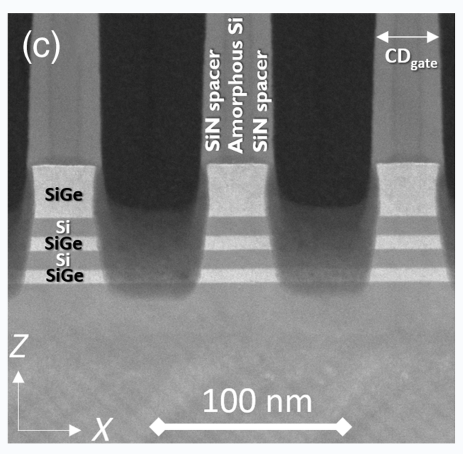
Asynchronous workloads

Simulations involve multiple types of governing equations that are specific to subdomains (heterogeneous coefficients, different source terms, ...).



- spatially dependent stretch function in PML
- source term related to incident field in nanoline
- homogeneous coefficients in air, spacer and substrate
- Dirichlet and periodic boundary conditions



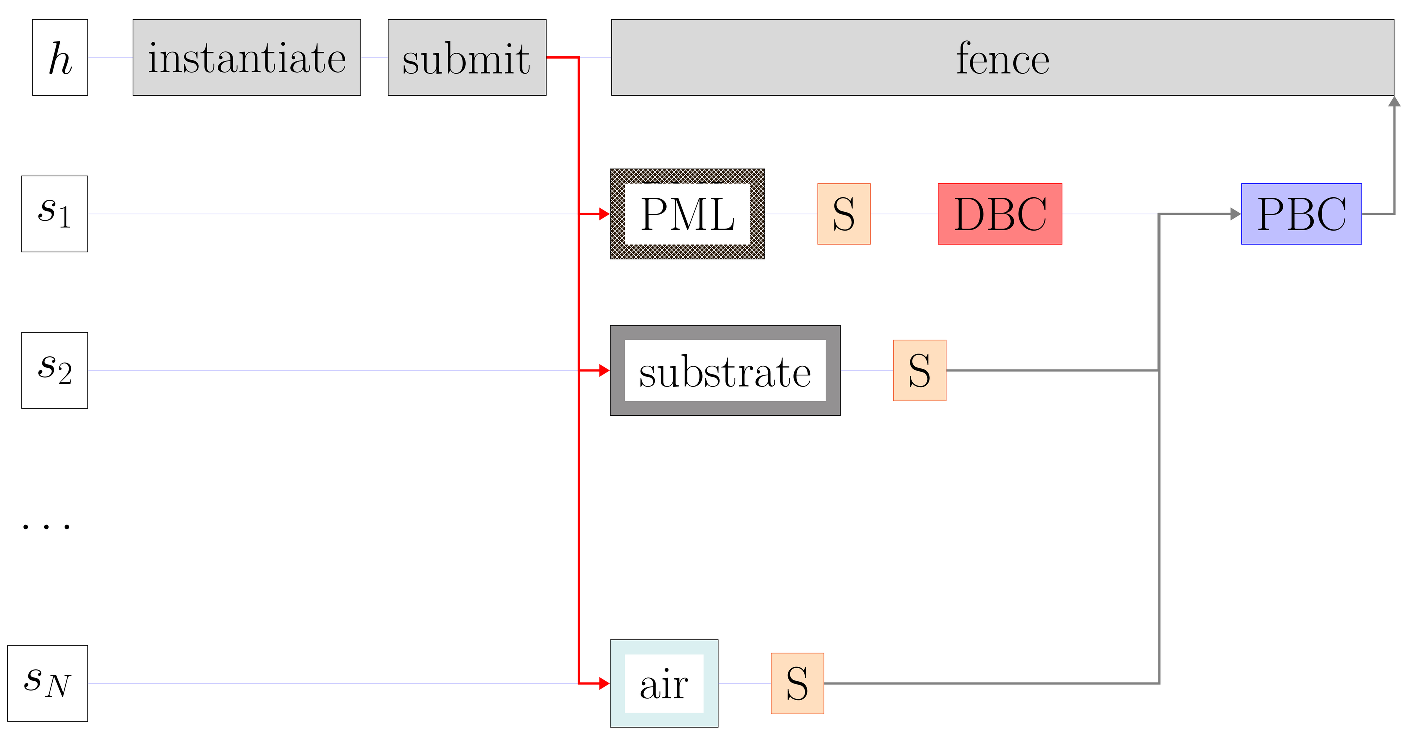


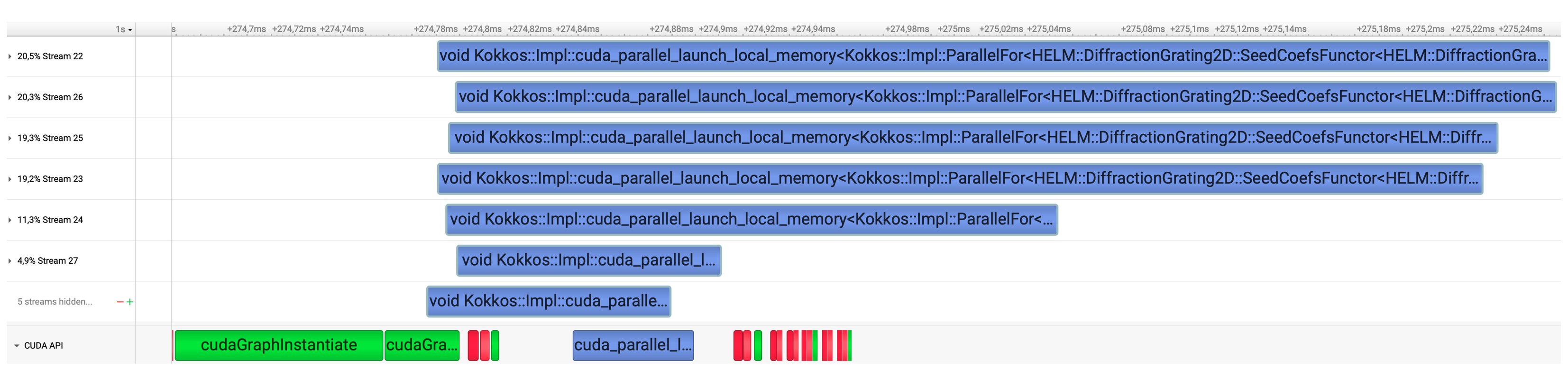
Tailoring functors to subdomain, type of BC, ... helps optimize resource usage and instructions. This leads to many workload dispatches that must be efficiently scheduled.

Runtime scheduling

For the `Cuda`, `HIP` and `SYCL` backends, the asynchronous workloads are executed concurrently when permitted by their dependencies and available resources.

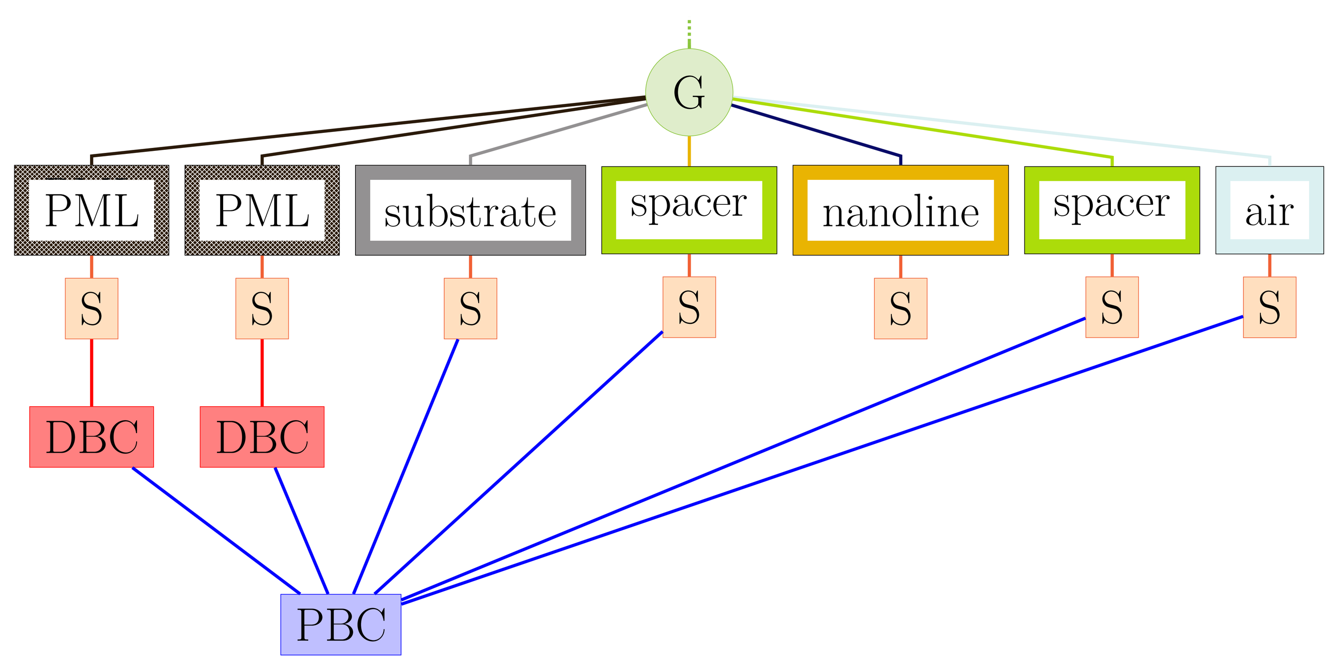
The `Nsight Systems` tracing shows the `cudaGraphInstantiate` and `cudaGraphLaunch` API calls on the host thread and multiple asynchronous workloads running concurrently in distinct device streams.





Mapping to Kokkos::Graph

`Kokkos::Graph` allows a Direct Acyclic Graph (DAG) of asynchronous workloads to be dispatched. For the `Cuda`, `HIP` and `SYCL` backends, it maps to optimized vendor graph constructs that allow the entire DAG to be dispatched efficiently.




```
auto graph = Kokkos::create_graph([&](const auto& root) {
    auto pml_then_s_then_dbc = root
        .then_parallel_for(policy_pml,       functor_pml)
        .then_parallel_for(policy_s_pml,     functor_s_pml)
        .then_parallel_for(policy_dbc_s_pml, functor_dbc_s_pml);
    auto air_then_s = root
        .then_parallel_for(policy_air,   functor_air)
        .then_parallel_for(policy_s_air, functor_s_air);
    auto agg = Kokkos::when_all(pml_then_s_then_dbc, ..., air_then_s);
    agg.then_parallel_for(policy_pbc, functor_pbc);
});
graph.instantiate();
graph.submit(exec);
```

Acknowledgements Tomasetti R. is a FRIA grantee of the Fonds de la Recherche Scientifique – FNRS.

uliegecsm.github.io

kokkos.org/kokkos-core-wiki