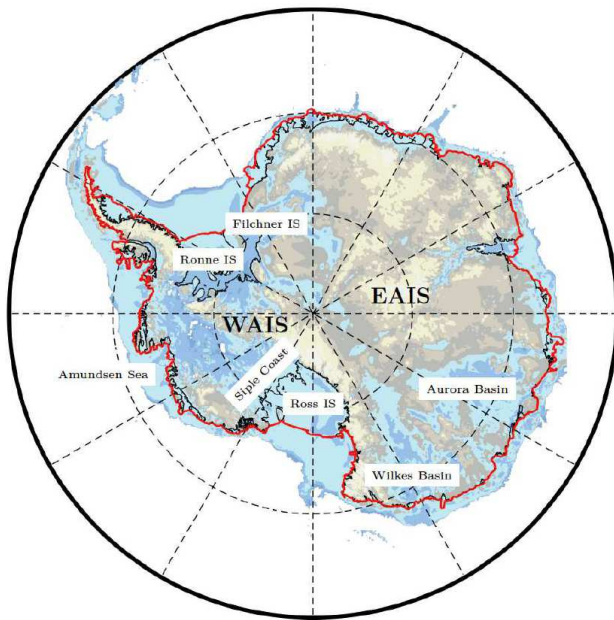


---

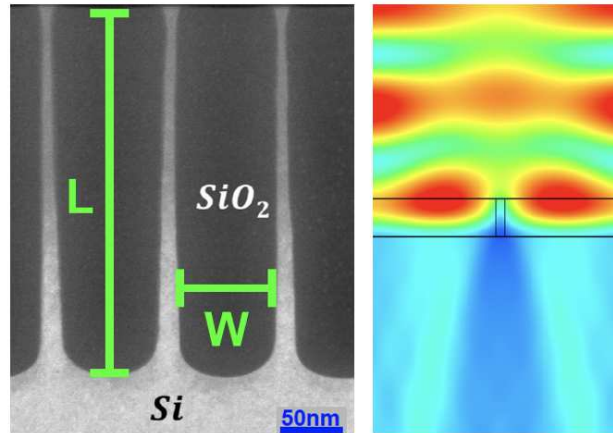
## Use of Trilinos and Stokhos ensembles in a multiphysics application code with embedded UQ

Maarten Arnst, Thomas Gregov, and Romin Tomasetti

September 13, 2022



Ice sheet.



Inverse problems  
in optical metrology.

Computational modeling.  
Augmented simulation.  
Uncertainty quantification.



Heterogeneous architectures.

**Contribute to the development of new methods, algorithms, and software  
for augmented simulations on heterogeneous architectures.**

# Team and collaborations



E. Phipps

K. Liegeois



M. Arnst

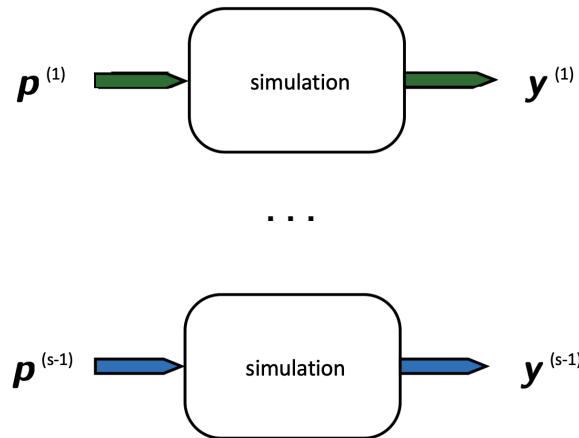
R. Tomasetti

T. Gregov

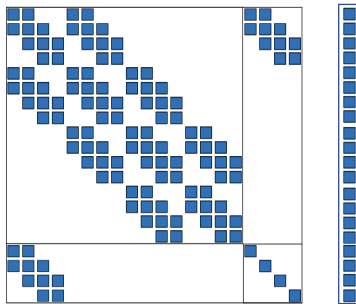
- Context.
- Team and collaborations.
- Plan.
- Stokhos ensembles:
  - ▷ Overview.
  - ▷ Advantages and challenges.
  - ▷ Katoptron.
- Multiphysics code:
  - ▷ Applications.
  - ▷ Overview.
  - ▷ Polymorphism.
  - ▷ Scatter computational kernel.
  - ▷ NOX Tpetra implementation.
  - ▷ trilinos-container.
- Directions for future work and conclusion.

---

## Stokhos ensembles

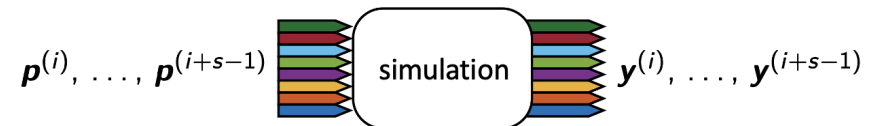


Samples  $\rightarrow$  outermost loop  
outside of the simulation code.

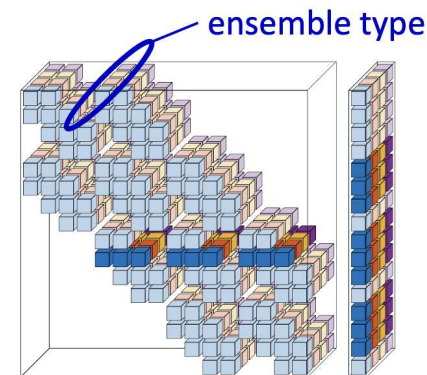


Scalar type: double.

**Nonintrusive UQ.**



Samples  $\rightarrow$  innermost unit of computation  
inside the simulation code.



Scalar type: ensemble type.

**Embedded UQ with ensembles.**

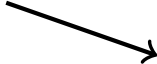
**Main idea: embedded UQ with an explicit SIMD scalar type [Phipps et al. SISC, 2017].  
Stokhos package in Trilinos (E. Phipps, K. Liegeois, ...).**

```
template <int s>
class Ensemble
{
    double val[s];

public:
    ///! Data accessor
    double &fastAccessCoeff(int i) {
        return val[i];
    }

    ///! Addition-assignment operator
    Ensemble &operator += (const Ensemble &x) {
        for (int i=0; i<s; i++)
            val[i] += x.fastAccessCoeff(i);
        return *this;
    }
};
```

instantiate for ensemble type

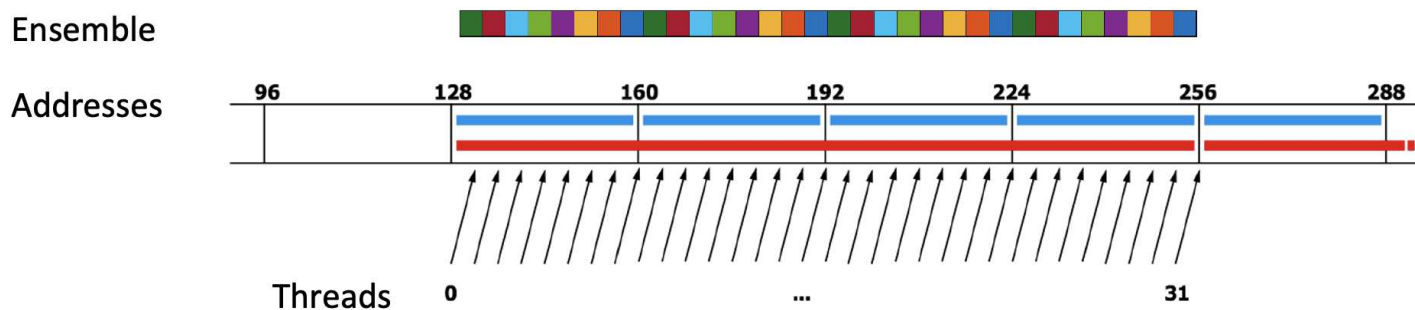


```
template <typename scalar_t>
class NonLinearSystem {
public:
    Tpetra::Vector<scalar_t> sol;
};
```

- Ensemble type = short vector of scalars (sample values). Compile-time sized, best fit for hardware.
- Mathematical operations are defined for the ensemble type. Found by overload resolution.
- For host code, compiler autovectorization maps math operations to hardware vector parallelism.

# Advantages and challenges

## Advantages



- Creates an innermost unit of computation naturally amenable to hardware vector parallelism.
- Sample-independent computations need to be carried out only once per ensemble.

## Challenges

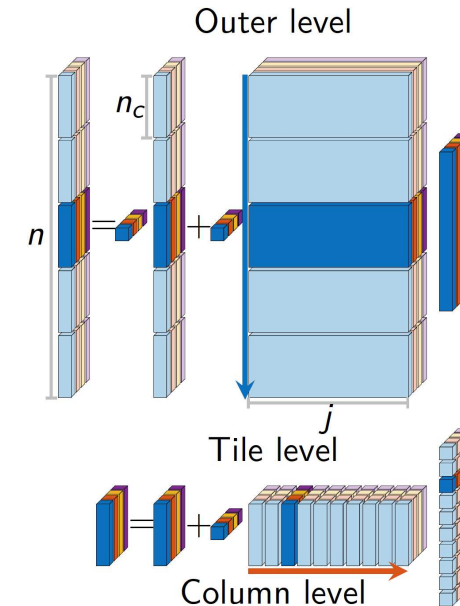
- Divergence: computations for different samples taking distinct code paths (e.g. conditionals).
- Functions called from optimized (e.g. vendor) libraries may not support the ensemble type.
- For device code, the overloaded operators do not map to warp-level parallelism. The issue is that the overloaded operators create loops in each thread; but, the lanes of a device warp are distinct threads. Stokhos has device template specializations of key computational kernels.
- Specializations can lead to significant code duplication, thus posing maintenance challenges.



```

 $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A} \mathbf{x}^{(0)}$ 
 $\beta = \|\mathbf{r}^{(0)}\|$ 
 $\mathbf{v}_{:1} = \mathbf{r}^{(0)} / \beta$ 
for  $j = 1, \dots, m$  do
     $\mathbf{w} = \mathbf{A} \mathbf{M}^{-1} \mathbf{v}_{:j}$ 
     $\mathbf{h}_{(1:j)j} = \mathbf{V}_{:(1:j)}^T \mathbf{w}$  Inner products
     $\mathbf{v}_{:(j+1)} = \mathbf{w} - \mathbf{V}_{:(1:j)} \mathbf{h}_{(1:j)j}$  Update
     $h_{(j+1)j} = \|\mathbf{v}_{:(j+1)}\|$ 
    if  $h_{(j+1)j} \neq 0$  then
         $\mathbf{v}_{:(j+1)} = \mathbf{v}_{:(j+1)} / h_{(j+1)j}$ 
    else
         $m = j$ 
        break
    if  $\mathbf{q}_{:(j+1)}^T \mathbf{e}_1 \leq \varepsilon$  then
         $m = j$ 
        break
 $\mathbf{y} = \arg \min_{\mathbf{z}} \|\beta \mathbf{e}_1 - \mathbf{H}_{(1:m+1)(1:m)} \mathbf{z}\|$ 
 $\mathbf{x}^{(m)} = \mathbf{x}^{(0)} + \mathbf{M}^{-1} \mathbf{V}_{:(1:m)} \mathbf{y}$ 

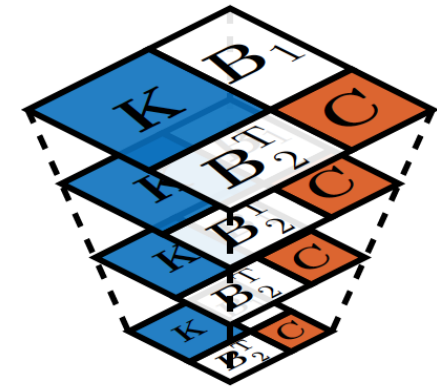
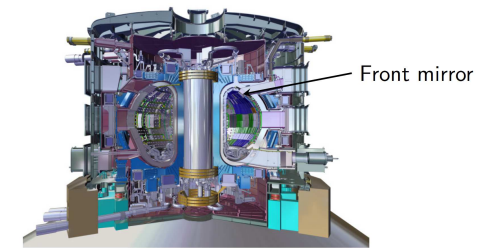
```



- Addressed divergence in GMRES by using masking.
- Template specialization of GEMV computational kernel for the ensemble type.

Kim Liégeois. GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models with application to the thermomechanical simulation of an ITER front mirror. PhD Thesis, University of Liège, 2020.

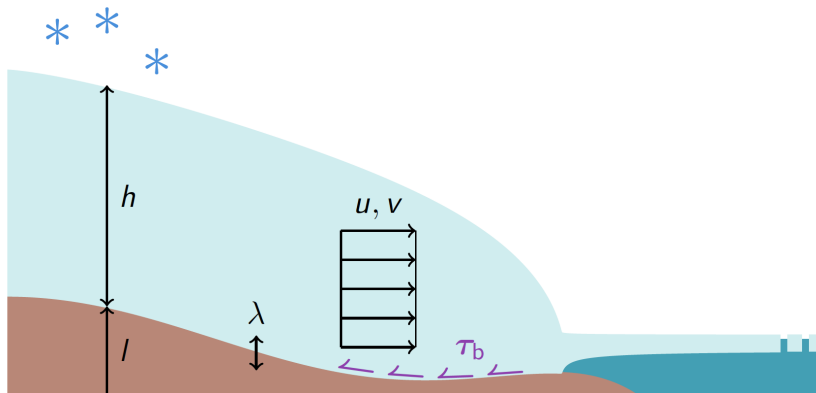
	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
Total	II	1.	4.072	5.844	7.031	3.982	5.769	7.143
Preconditioner setup								
Total	III	1.	7.015	12.776	20.409	7.022	12.936	20.536
GMRES								
Orthogonalization	IV.X	1.	0.628	0.506	0.444	1.050	1.049	0.975
Matrix-vector product	IV.IV	1.	1.095	1.048	0.954	1.359	1.424	1.331
Preconditioner	IV.III	1.	1.517	1.382	1.147	1.910	1.880	1.598
Level 0 smoother		1.	1.342	1.198	0.972	1.694	1.635	1.359
Level 1 smoother		1.	4.232	4.609	4.502	5.362	6.251	6.275
Level 2 smoother		1.	1.035	0.983	0.939	1.249	1.262	1.225
Total	IV	1.	1.428	1.288	1.078	1.842	1.819	1.560
Total		1.	1.583	1.453	1.225	2.021	2.032	1.763



Kim Liégeois. GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models with application to the thermomechanical simulation of an ITER front mirror. PhD Thesis, University of Liège, 2020.

---

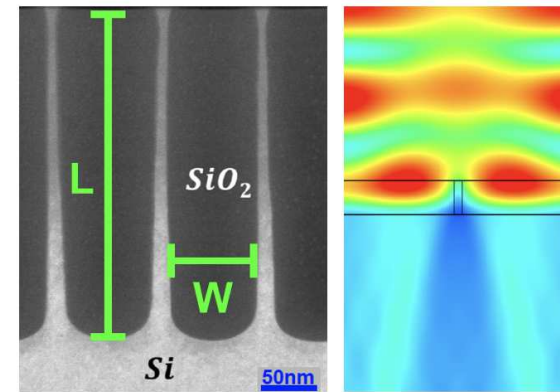
Multiphysics code



Ice sheet.

Collaboration with ULB, Belgium.

Thomas Gregov. PhD Thesis, ULiège, 2020-...



Inverse problems  
in optical metrology.

Collaboration with IMEC, Belgium.

Romin Tomasetti. PhD Thesis, ULiège, 2022-...

- We develop a multiphysics code, currently driven by two applications, the first relevant to ice sheets, and the second relevant to inverse problems in optical metrology.



```
template <typename scalar_t, typename execution_space>
class NonLinearSystem {
public:
    using node = Kokkos::Compat::KokkosDeviceWrapperNode<execution_space>;

    Tpetra::Vector<scalar_t, local_ordinal_t, global_ordinal_t, node> sol;
};
```

All classes in our multiphysics code are templated on the scalar type and the execution space.

mesh	foundations	dofs	basis and quadrature	solvers	augmented simulation
gmsh	kokkos tpetra	panzer	intrepid2	amesos2 belos ifpack2 NOX tempus cusolver	stokhos loca

Our multiphysics code is built by making extensive use of components from Trilinos.



physics-agnostic  
generic components



physics-specific  
components

Our multiphysics code provides an abstract base class for the elemental computation on a cell. The physical model provides an implementation.

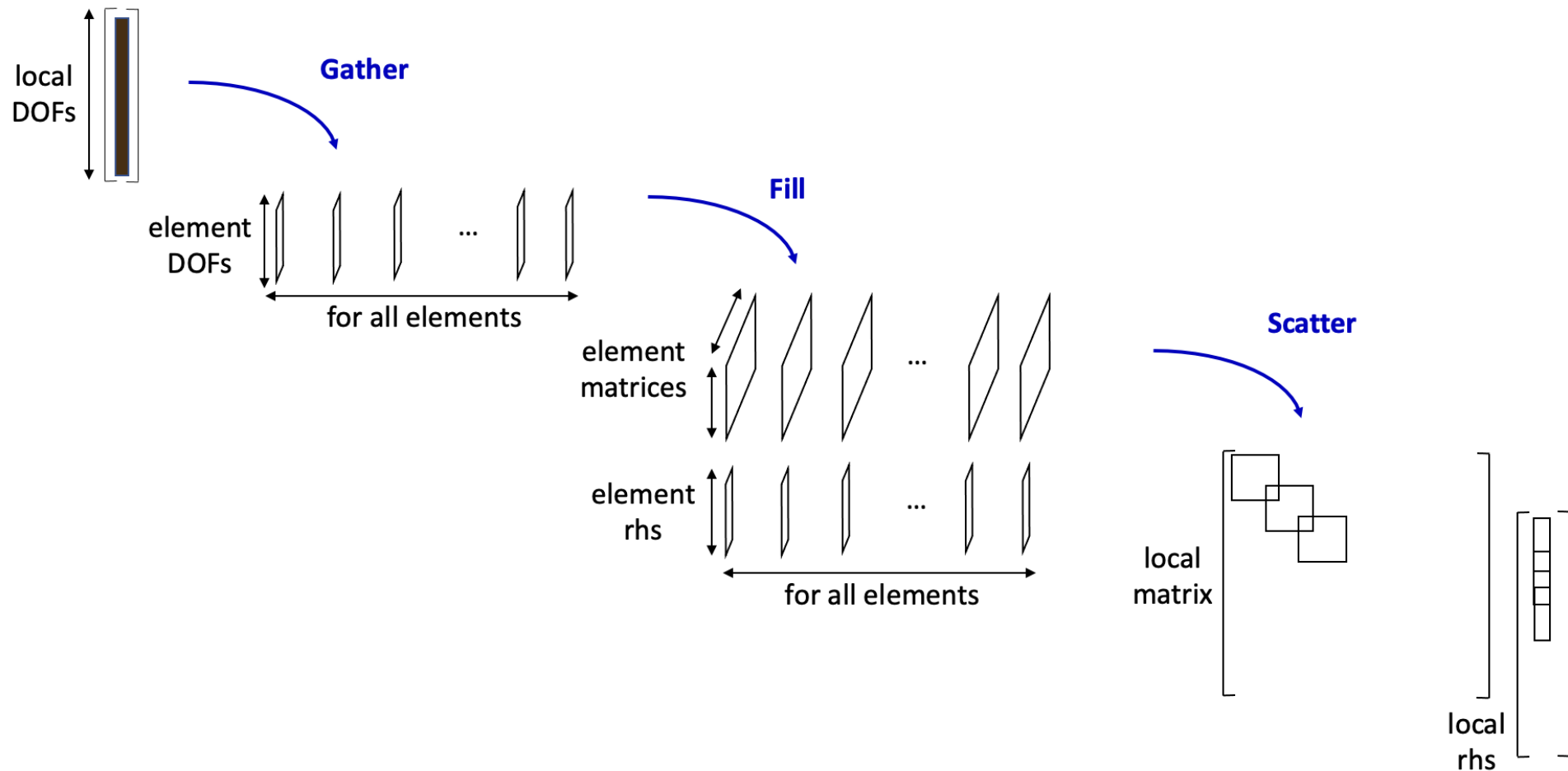
- We rely on patterns for polymorphism in Kokkos applications:

- ▷ `copy_for_device`.
- ▷ `Dispatcher`.

These patterns were proposed in:

- ▷ V. Brunini et al. Runtime polymorphism in Kokkos applications. Presentation. SAND2019-0279PE, 2019
- ▷ M. Howard et al. Towards a performance portable compressible CFD code. AIAA 2017–4407. SAND2017-5941C, 2017.

# Scatter computational kernel



# Scatter computational kernel

```
template <typename scalar_t, typename execution_space>
struct Scatter
{
    KOKKOS_INLINE_FUNCTION
    void operator()(
        const ordinal_t          elm,
        const const_scalar_2d_view_t &stackedRhs,
        const local_mv_view_t      &localRhs
    ) const {
        for (ordinal_t dof = 0; dof < numDofs; dof++) {
            local_ordinal_t lcl_row_id = localOwnedMap.getLocalElement(indexer(elm, dof));
            if (lcl_row_id != LO_INVALID) {
                Kokkos::atomic_add(
                    &localRhs(lcl_row_id, 0),
                    stackedRhs(elm, dof)
                );
            }
        }
    }
};

KOKKOS_PARALLEL_FOR(
    Kokkos::RangePolicy<execution_space>(0, numRlvntElms),
    KOKKOS_LAMBDA (const ordinal_t elm) { scatter->operator()(elm, stackedRhs, localRhs); }
);
```

index lookups

atomic adds

(Alternative implementation: Kokkos::ScatterView)



# Scatter computational kernel

```
template <typename scalar_t, typename execution_space>
struct Scatter
{
    KOKKOS_INLINE_FUNCTION
    void operator()(
        const ordinal_t          elm,
        const const_scalar_2d_view_t &stackedRhs,
        const local_mv_view_t      &localRhs
    ) const {
        for (ordinal_t dof = 0; dof < numDofs; dof++) {
            local_ordinal_t lcl_row_id = localOwnedMap.getLocalElement(indexer(elm, dof));
            if (lcl_row_id != LO_INVALID) {
                Kokkos::atomic_add(
                    &localRhs(lcl_row_id, 0),
                    stackedRhs(elm, dof)
                );
            }
        }
    }
};

KOKKOS_PARALLEL_FOR(
    Kokkos::RangePolicy<execution_space>(0, numRlvntElms),
    KOKKOS_LAMBDA (const ordinal_t elm) { scatter->operator()(elm, stackedRhs, localRhs); }
);
```

index lookups  
→ once per ensemble

atomic adds  
→ memory access fills cache lines  
→ mapped to vector parallelism (AVX)

**Host code:** instantiate for the **ensemble type** and a host execution space.

# Scatter computational kernel

```
template <>
struct Scatter<Ensemble<ENSEMBLE_SIZE>, Kokkos::Cuda>
{
    KOKKOS_INLINE_FUNCTION
    void operator()(
        const member_t          teamMember,
        const int                sample,
        const const_scalar_2d_view_t &stackedRhs,
        const local_mv_view_t      &localRhs
    ) const {
        const ordinal_t ielm = teamMember.league_rank() * teamMember.team_size() + teamMember.team_rank();
        for (ordinal_t dof = 0; dof < numDofs; dof++) {
            Kokkos::single(Kokkos::PerThread(teamMember), [&] (local_ordinal_t &lcl_row_id_tmp) {
                lcl_row_id_tmp = localOwnedMap.getLocalElement(Indexer(elm, dof));
            }, lcl_row_id);
            if (lcl_row_id != LO_INVALID) {
                Kokkos::atomic_add(
                    &localRhs(lcl_row_id).fastAccessCoeff(sample),
                    stackedRhs(elm, dof).fastAccessCoeff(sample)
                );
            }
        }
    }
};

KOKKOS_PARALLEL_FOR(teamPolicy, KOKKOS_LAMBDA (const member_t &teamMember) {
    Kokkos::parallel_for(Kokkos::TeamThreadRange(teamMember, teamSize), [=] (const int /* */) {
        Kokkos::parallel_for(Kokkos::ThreadVectorRange(teamMember, ENSEMBLE_SIZE), [=] (const int sample) {
            scatter->operator()(teamMember, sample, stackedRhs, localRhs);
        });
    });
});
```

index lookups  
→ once per ensemble

atomic adds  
→ coalesced memory access  
→ mapped to warp-level parallelism

**Device code:** template specialization for **ensemble type** and device execution space to lay out explicitly how the computation maps to warp-level parallelization.

# NOX tpetra implementation

- NOX is a nonlinear solver package in Trilinos.
- NOX defines abstract base classes for the main components involved in the nonlinear solution process (vectors, ...).

NOX provides epetra, lapack, petsc and thyra implementations for the abstract base classes. Wrapping tpetra objects in thyra objects is one way of using NOX with the tpetra stack.

- ```
template <typename Scalar, typename LO, typename GO, typename Node>
class Vector : public NOX::Abstract::Vector {
public:
    using vector_type = Tpetra::Vector<Scalar, LO, GO, Node>;
private:
    Teuchos::RCP<vector_type> tpetraVec;
};
```

We developed a tpetra implementation for the NOX abstract base classes. This implementation avoids the use of thyra wrappers when using NOX with the tpetra stack. Our implementation covers all functionalities and tests, except those relevant to the JFNK method.

- We work with Docker containers that package Trilinos and dependencies. We use these containers in our day-to-day coding, as well as in GitHub actions that run tests for our multiphysics code.
- We have developed a toolbox, `trilinos-container`, that can automatically build a collection of Docker images and publish them in a private repository. It is made up of dockerfiles, Python scripts, and GitHub action files.



The configuration of the Docker images is controlled through CMakePresets and other .json files.

- We regularly build Docker images for OpenMP builds; for CUDA builds, with and without UVM, for the Volta and Ampere architectures; and for ROCM builds.

## Summary

- Trilinos Stokhos package: ensembles for embedded UQ with an explicit SIMD type.
- Multiphysics code.
- NOX tpetra implementation.
- trilinos-container.

## Current focus

- Modeling and formulation of the ice-sheet and photonics application in a manner that facilitates ensemble propagation (reduces divergence, ...).
- Performance evaluation and optimization. GPU.
- LUMI supercomputer (Zen3/MI250X).

**Thank you for your attention!**