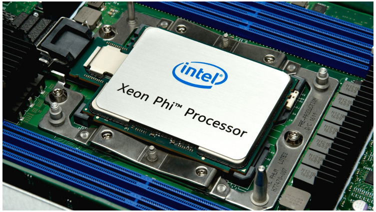
Efficient parametric computations using ensemble propagation for high dimensional finite element models



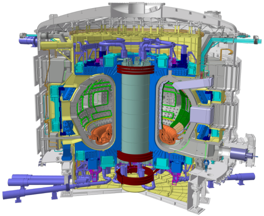
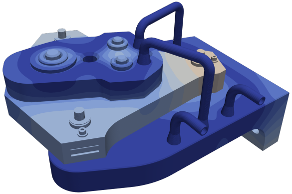
K. Liegeois[1], R. Boman[1], E. T. Phipps[2] and M. Arnst[1]

[1]Aerospace and Mechanical Engineering, Université de Liège, Belgium
[2]Sandia National Laboratories, USA

11th CÉCI Scientific Meeting
Université Libre de Bruxelles
April 25, 2019
`https://kliegeois.github.io/`

Ongoing PhD: New methods for parametric computations with multiphysics models on HPC architectures with applications to design of opto-mechanical systems

ITER

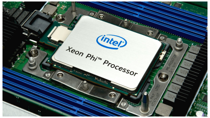
Ph. Mertens, A. Panin, FZ. Jülich

High performance computing library

Clusters



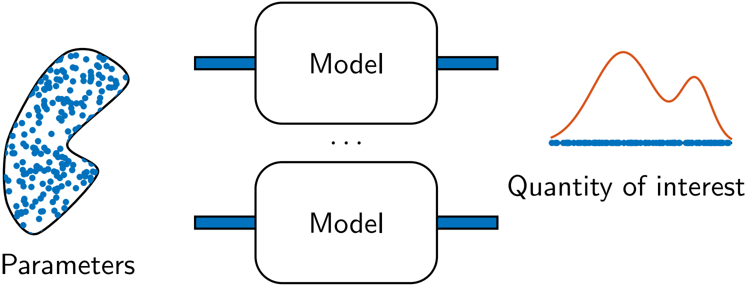
Emerging architectures



Parametric computations

Sampling-based parametric computations typically require numerous calls of potentially **costly models**.

Example: Monte Carlo for uncertainty quantification.



Goal of the work: to **reduce the CPU cost** to evaluate a given set of samples or **increase the number of evaluated samples** for a given CPU cost.

Ensemble propagation

In sampling-based parametric computation, instead of individually evaluating each instance of the model, Ensemble propagation (EP) consists of **simultaneously evaluating** a **subset of samples** of the model.

EP was introduced by [Phipps, 2017], made available in **Stokhos** a package of **Trilinos**, and implemented using a **template-based generic-programming** approach:

```
template <typename T, int ensemble_size>
class Ensemble{
    T data[ensemble_size];
    Ensemble<T,ensemble_size> operator+ (const Ensemble<T,ensemble_size> &v);
    Ensemble<T,ensemble_size> operator- (const Ensemble<T,ensemble_size> &v);
    Ensemble<T,ensemble_size> operator* (const Ensemble<T,ensemble_size> &v);
    Ensemble<T,ensemble_size> operator/ (const Ensemble<T,ensemble_size> &v);
    //...
}
```

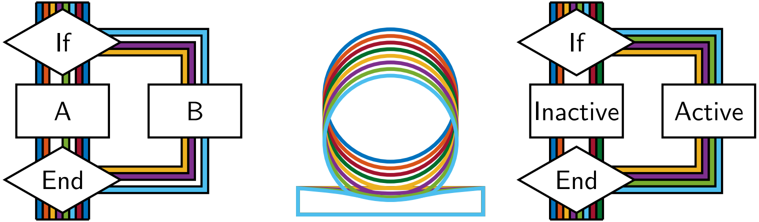
Ensemble propagation

Advantages of the EP:

- Reuse of common variables,
- More opportunities for SIMD (more data parallelism),
- Improved memory usage,
- Reduction of Message Passing Interface (MPI) latency per sample.

Challenges of the EP:

- Increased memory usage,
- Ensemble divergence:
  - control flow divergence: if-then-else divergence and loop divergence,



  - function call divergence.

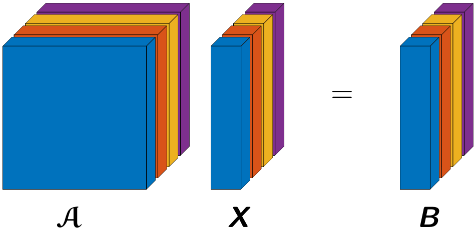
Parametric linear systems

We want to solve a **parametric linear system** for a subset of s samples of the parameters together:

$$\boldsymbol{A}_{::\ell}\,\boldsymbol{x}_{:\ell} = \boldsymbol{b}_{:\ell} \quad \text{for all} \quad \ell = 1, \ldots, s,$$

where matrices $\boldsymbol{A}_{::1}, \ldots, \boldsymbol{A}_{::s}$ are not necessarily symmetric positive definite (SPD).

Representation of a system for $s = 4$:



As the matrices are not SPD, we cannot use conjugate gradient methods.

GMRES and ensemble divergence

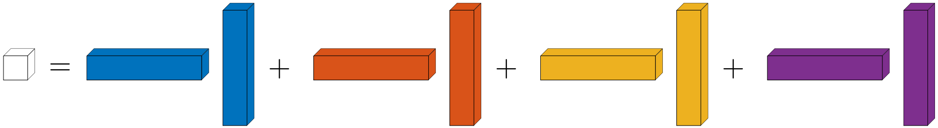
$$
\begin{aligned}
&\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}^{(0)}\\
&\beta = \|\boldsymbol{r}^{(0)}\|\\
&\boldsymbol{v}_{:\,1} = \boldsymbol{r}^{(0)}/\beta\\
&\textbf{for } j = 1, \ldots, m \textbf{ do}\\
&\quad \boldsymbol{w} = \boldsymbol{A}\boldsymbol{M}^{-1}\,\boldsymbol{v}_{:\,j}\\
&\quad \boldsymbol{h}_{(1:j)j} = \boldsymbol{V}^{\mathrm{T}}_{:(1:j)}\,\boldsymbol{w}\\
&\quad \boldsymbol{v}_{:(j+1)} = \boldsymbol{w} - \boldsymbol{V}_{:(1:j)}\,\boldsymbol{h}_{(1:j)j}\\
&\quad h_{(j+1)\,j} = \|\boldsymbol{v}_{:\,(j+1)}\|\\
&\quad \textbf{if } h_{(j+1)\,j} \neq 0 \textbf{ then}\\
&\quad\quad \boldsymbol{v}_{:\,(j+1)} = \boldsymbol{v}_{:\,(j+1)}/h_{(j+1)\,j}\\
&\quad \textbf{else}\\
&\quad\quad m = j\\
&\quad\quad \textbf{break}\\
&\quad \textbf{if } \boldsymbol{q}^{\mathrm{T}}_{:(j+1)}\boldsymbol{e}_1 \leq \varepsilon \textbf{ then}\\
&\quad\quad m = j\\
&\quad\quad \textbf{break}\\
&\boldsymbol{y} = \arg\min_{\boldsymbol{z}} \|\beta\,\boldsymbol{e}_1 - \boldsymbol{H}_{(1:m+1)(1:m)}\,\boldsymbol{y}\|\\
&\boldsymbol{x}^{(m)} = \boldsymbol{x}^{(0)} + \boldsymbol{M}^{-1}\boldsymbol{V}_{:(1:m)}\,\boldsymbol{y}
\end{aligned}
$$

Ensemble divergence in the GMRES:

1. an Arnoldi vector can require a normalization or not: **if-then-else divergence**,
2. different samples may require different numbers of iterations to converge: **loop divergence**,
3. called BLAS functions, such as GEMV for the dense matrix-vector operations, may not support ensemble-typed inputs, leading to **function call divergence**.

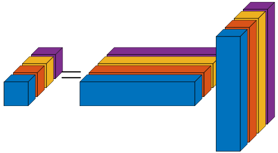
Reduced and ensemble-typed inner products used in CG

- **Reduced inner product** and its associated norm were the first ones introduced, implemented, and tested in the EP [Phipps, 2017]:



Fully remove every ensemble divergence **coupling** the samples together.

- **Ensemble-typed inner product** was first introduced for grouping purpose [D'Elia, 2017]:



This approach requires to **manage** every ensemble divergence **explicitly**.

Advantages and challenges of both approaches

Reduced inner product:

Advantages:

- **No control flow divergence**.
- Use of **standard libraries** such as MKL.

Challenges:

- Convergence in the least-squares sense.
- The spectrum of the ensemble matrix **is the union** of the spectra of the sample matrices: having a good preconditioner is more complex.
- **Increased** number of iterations.

Ensemble-typed inner product:

Advantages:

- Convergence for every sample.
- The spectra **are not** gathered.
- Convergence rates **controlled** by the slowest sample.

Challenges:

- **Control flow divergence** has to be treated explicitly.
- **No current** implementation of the needed BLAS routines in the MKL.

Control flow divergence

The control flow divergence, both the **if-then-else divergence** and the **loop divergence**, has been solved by defining a Mask class equivalent to:

```
template <int ensemble_size>
class Mask{
    bool data[ensemble_size];
    //...
}
```

which is returned by any comparison of ensembles.

This mask is then used for masked assignments and logical reductions:

```
Ensemble<double,8> a,b;
b = 3.; b[3] = -5.; b[7] = 5.;
mask_assign(b>=0.,a) = {b,1.};
cout << a << endl;                  // Print: [3.,3.,3.,1.,3.,3.,3.,5.]
mask_assign(a>=5.,a) /= {a, 5.,-1.};
cout << a << endl;                  // Print: [-1.,-1.,-1.,-1.,-1.,-1.,-1.,1.]
bool test_a = AND(a==1.);
cout << test_a << endl;             // Print: 0
bool test_a = OR(a==1.);
cout << test_a << endl;             // Print: 1
```

Those operations are enough to safely implement the GMRES.

GEMV with Ensemble propagation

The **GEMV** with EP takes the form of a **tensors contraction** as follows:

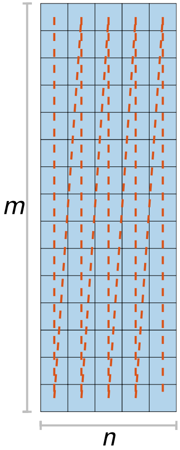
$$\mathbf{y}_{:\ell} = \beta_\ell \, \mathbf{y}_{:\ell} + \alpha_\ell \, \mathbf{A}_{::\ell} \, \mathbf{x}_{:\ell} \quad \text{for all} \quad \ell = 1, \dots, s,$$

Such an operation has a **low arithmetic intensity** as, for every $a_{ij\ell}$ loaded from memory only two operations are performed.

Interleaved memory layout of the $m \times n \times s$ third-order tensor $\mathcal{A}$:

$$a_{ij\ell} \hookleftarrow a\left[(i-1)\,s + (j-1)\,m\,s + (\ell-1)\right].$$

Tall skinny matrices $\mathbf{A}_{::\ell}$ with left layout and row stride of s



GEMV with Ensemble propagation

Challenge: the **memory layout** and the fact that the operation is memory bound prevent us from using efficiently a **scalar-typed GEMV** implementation sequentially s times.

How should we implement the contraction such that theoretical performance is achieved?

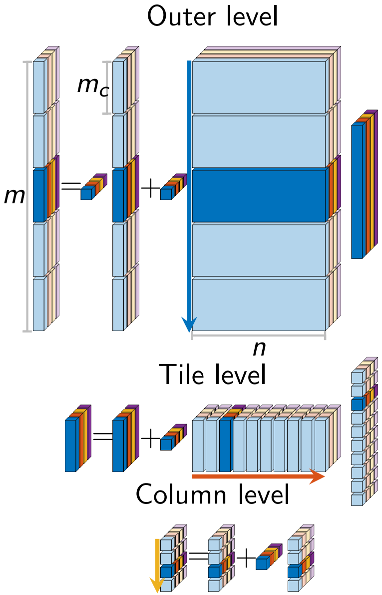
In order to use efficiently the **memory bandwidth** here, it is important to:

- Reuse reusable data from cache,
- Use all the physical cores,
- Load data with unit stride,
- Use vector instructions while avoiding gather vector loads.

GEMV with Ensemble propagation

parfor $t = 1$ **to** $m - m_c + 1$ **by** m_c **do**
  for $i = t, \ldots, t + m_c - 1$ **do**
    $y_{i\ell} = \beta_\ell \, y_{i\ell}$ for all $\ell = 1, \ldots, s$
  for $j = 1, \ldots, n$ **do**
    $\gamma_\ell = \alpha_\ell \, x_{j\ell}$ for all $\ell = 1, \ldots, s$
    for $i = t, \ldots, t + m_c - 1$ **do**
      $y_{i\ell} = y_{i\ell} + \gamma_\ell \, a_{ij\ell}$ for all $\ell = 1, \ldots, s$

- Tiling:
  - Each thread applies a tile of $\mathcal{A}$ at a time,
  - Cache blocking of $\mathbf{Y}$.
- Vectorization:
  - Vectorization of the loops over the samples,
  - Intel Intrinsics, overloaded operators.



GEMV: results - KNL

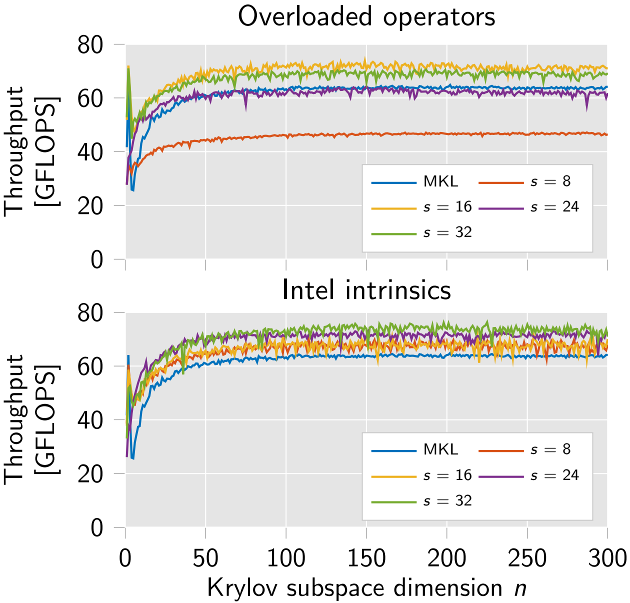
Xeon Phi KNL in quadrant cache mode
Measured bandwidth:
320 GB/s

Deduced maximal throughput:
80 GFLOPS

Parameters:

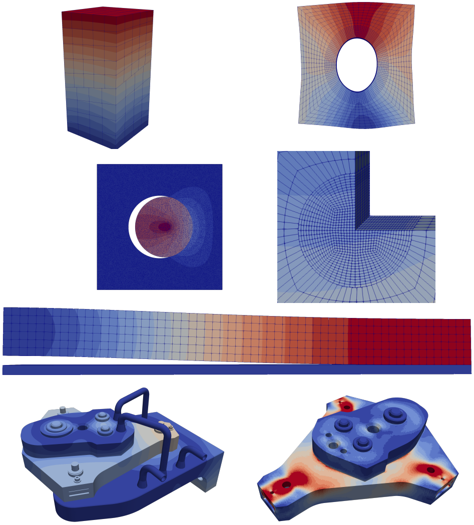
- Threads $N = 128$
- $m_c = 1024$ for $s = 8$, $m = 8\,N\,m_c$,
- for a given n, data size independent of s.

Performance greater than the MKL,
Performance similar to the theoretical limit,
Sensibility to the order of the operations.

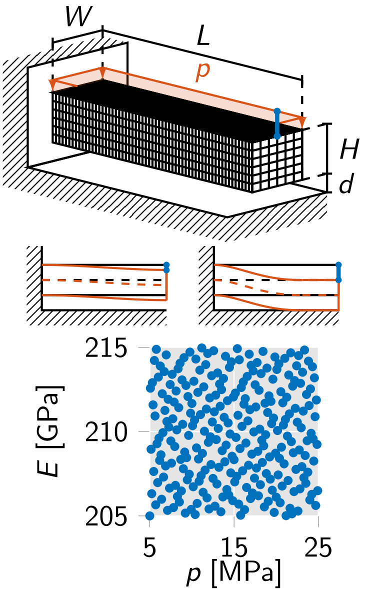


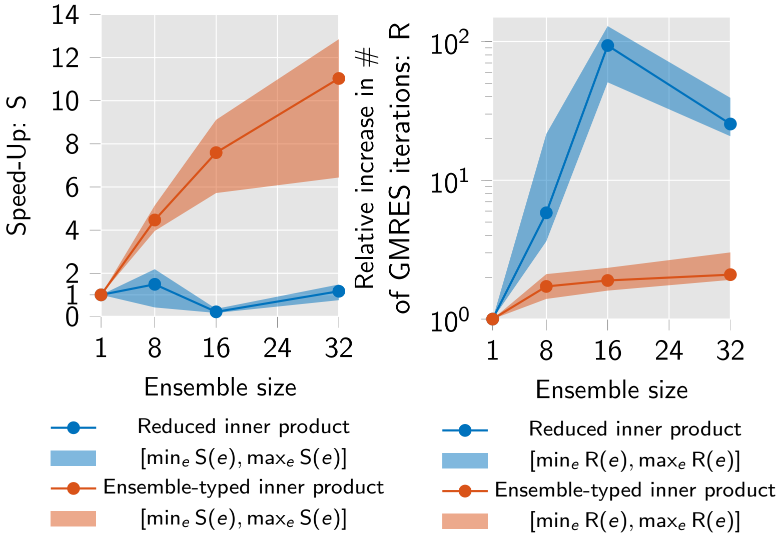
Implemented code and its capabilities

- **Fully templated C++** code heavily based on **Trilinos** which provides a fully templated solver stack.
- Embedded in a **Python** interface. This eases the looping around samples, the grouping of samples together, etc.
- **Hybrid parallelism** based on **Tpetra** with **MPI** for distributed memory and **Kokkos** with **OpenMP** for shared memory.
- Uses **Gmsh** [Geuzaine, 2009] to import 3D meshes and **VTK** to write the output files.
- Has already generated preliminary results for **industrial thermomechanical contact problems**.



Test case: beam contact problem on Xeon Phi KNL





Conclusion

Conclusion and contributions:

- Contributions towards EP applied to the GMRES,
- Implementation of the mask and the masked assignments,
- Implementation of the GEMV for ensemble type that reaches performance similar to the MKL,
- Two variants of the GMRES can currently be used: with reduced inner product and with ensemble-typed inner product,
- First results that suggest that the GMRES with ensemble-typed inner product is faster than the GMRES with reduced inner product.

Future work:

- Applying the method on **engineering problems** relevant for **ITER** in collaboration with FZ. Jülich,
- Testing on more than one computational node to leverage the increased memory usage,
- Studying how to use this method in **uncertainty quantification** of contact problems with **local surrogate model** and **grouping**,

Acknowledgement

The first author, Kim Liegeois, would like to acknowledge the Belgian National Fund for Scientific Research (FNRS-FRIA) and the Federation Wallonia-Brussels (FW-B) for their financial support.

FREEDOM TO RESEARCH