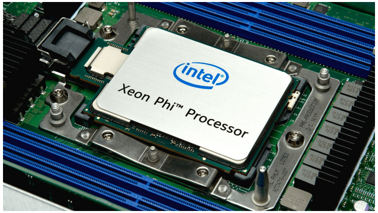
Ensemble Propagation for Efficient Uncertainty Quantification of Mechanical Contact Problems



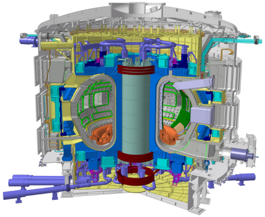
K. Liegeois[1], R. Boman[1], E. T. Phipps[2] and M. Arnst[1]

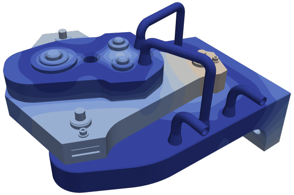
[1]Aerospace and Mechanical Engineering, Université de Liège, Belgium
[2]Sandia National Laboratories, USA

2019 SIAM Conference on Computational Science and Engineering
MS332 SIMD Approaches for Achieving Performance and Portability on Emerging Computational Architectures
Spokane, USA
February 28, 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



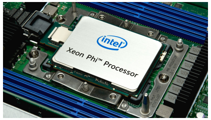


Ph. Mertens, A. Panin, FZ. Jülich

High performance computing library

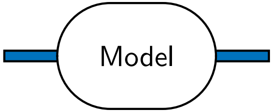
Clusters

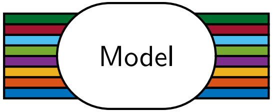
Emerging architectures



Ensemble propagation

In sampling-based uncertainty quantification (UQ), 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);
    //...
}
```

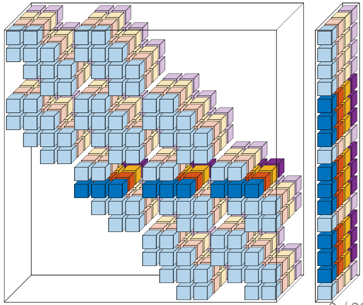
Ensemble propagation

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.

Example sparse matrix vector product:

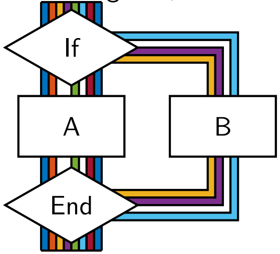
```
// CRS matrix-vector product z = A*x for arbitrary floating-point type T
template <typename T>
void crs_mat_vec(const CrsMatrix<T>& A, const T *x, T *z) {
  for (int row =0; row<A.num_rows; ++row) {
    const int entry_begin = A.row_map[row];
    const int entry_end = A.row_map[row+1];
    T sum = 0.0;
    for (int entry = entry_begin; entry<entry_end; ++entry) {
      const int col = A.col_entry[entry];
      sum += A.values[entry] * x[col];
    }
    z[row] = sum;
  }
}
```

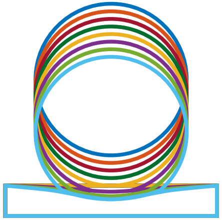


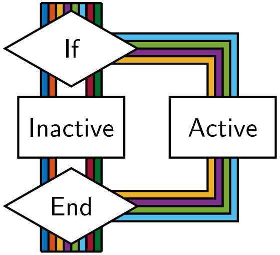
Ensemble propagation

Challenges of the EP:

- Increased memory usage,
- Ensemble divergence:
  - if-then-else divergence,







  - loop divergence,
  - function call divergence.

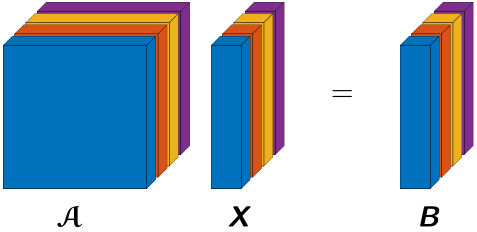
Parametric linear systems

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

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

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

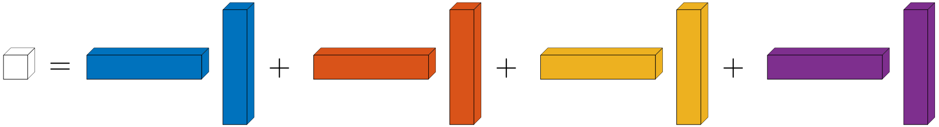
```
$r^{(0)} = b - A\, x^{(0)}$
$\beta = \|r^{(0)}\|$
$v_{:1} = r^{(0)}/\beta$
for $j = 1, \ldots, m$ do
    $w = AM^{-1}\, v_{:j}$
    $h_{(1:j)j} = V^{\mathrm{T}}_{:(1:j)}\, w$
    $v_{:(j+1)} = w - V_{:(1:j)}\, h_{(1:j)j}$
    $h_{(j+1)j} = \|v_{:(j+1)}\|$
    if $h_{(j+1)j} \neq 0$ then
        $v_{:(j+1)} = v_{:(j+1)}/h_{(j+1)j}$
    else
        $m = j$
        break
    if $q^{\mathrm{T}}_{:(j+1)} e_1 \leq \varepsilon$ then
        $m = j$
        break
$y = \arg\min_z \|\beta\, e_1 - H_{(1:m+1)(1:m)}\, y\|$
$x^{(m)} = x^{(0)} + M^{-1} V_{:(1:m)}\, y$
```
Algorithm 1: GMRES for one sample

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**.

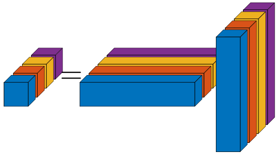
Reduced and ensemble-typed inner products

- **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**, was 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 **tensors contractions** as follows:

$$\mathbf{y}_{:\ell} = \beta_\ell \, \mathbf{y}_{:\ell} + \alpha_\ell \, \mathbf{A}_{::\ell} \, \mathbf{x}_{:\ell} \quad \text{for all} \quad \ell = 1, \ldots, s, \qquad (2)$$

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

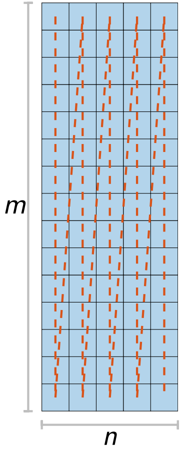
$$a_{ij\ell} \leftarrow\!\!\dashv a\left[(i-1)\, s + (j-1)\, m\, s + (\ell - 1)\right], \qquad (3)$$

i.e.

```
Kokkos::View< Ensemble<double,s>**,
  Kokkos::LayoutLeft, Kokkos::Device,
  Kokkos::MemoryTraits>
```

Challenge: the **memory layout** prevents us from using efficiently a **scalar-typed GEMV** implementation sequentially s times.

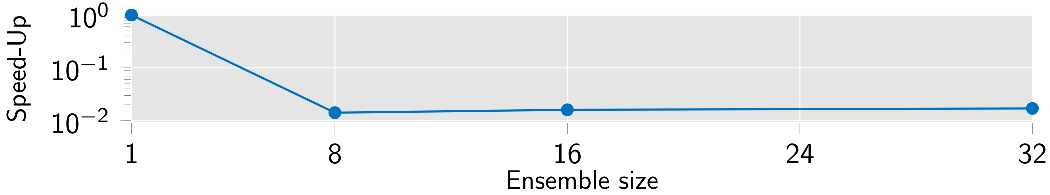
Tall skinny matrices $\mathbf{A}_{::\ell}$ with left layout and row stride of s



GEMV with Ensemble propagation

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

The throughput of this computation is therefore limited by the **memory bandwidth** on standard architectures. The speed-up of this tensors contraction versus s GEMV with unit stride **cannot be greater than 1**.
Unoptimized implementations of the contraction lead to **a big slowdown** of the GMRES:



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

GEMV and GEMM in the literature

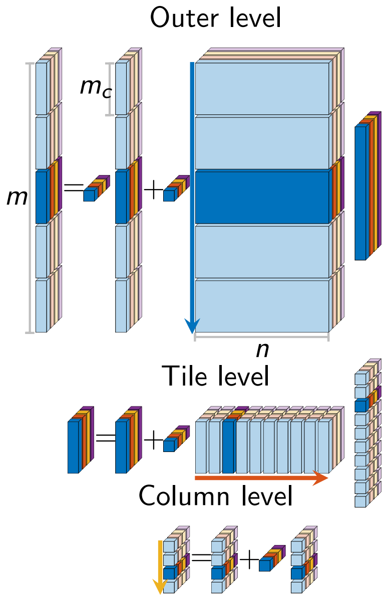
To reach full bandwidth, we have to:

- **Exploit the parallelism of the architecture:**
  - Use every physical cores as much as possible.
- **Transfer data efficiently through the memory hierarchy:**
  - Keep reusable data in cache.
  - Use unit stride loads.
- **Exploit CPU power:**
  - Keep reusable data in registers.
  - Use vector load and store, avoid vector gather.

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:
  - As usual,
  - Each thread applies a tile at a time,
  - Cache blocking of **Y**.
- Vectorization:
  - Different,
  - Vectorization of the loops over the samples,
  - Intel Intrinsics, overloaded operators.



GEMV: results - KNL

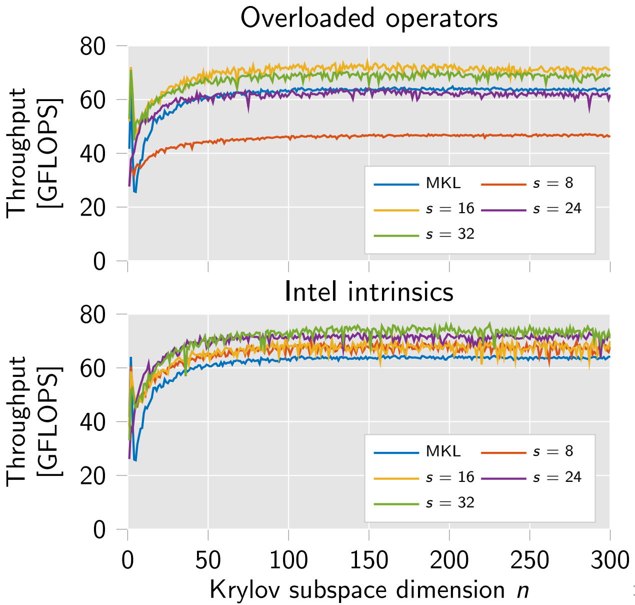
Xeon Phi KNL in quadrant cache mode
Measured bandwidth:
320 GB/s

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.



GEMV: results - Skylake

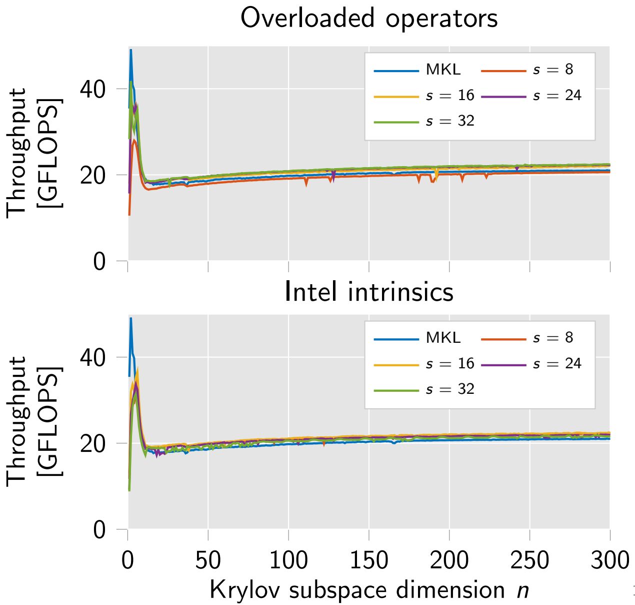
Xeon Skylake
Measured bandwidth:
88 GB/s

Deduced maximal throughput:
22 GFLOPS

Parameters:

- Threads $N = 24$
- $m_c = 1024$ for $s = 8$, $m = 8\,N\,m_c$,
- for a given n, data size independent of s.

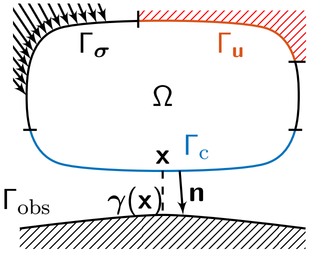
Performance similar to the MKL,
Performance similar to the theoretical limit,
Less sensitive to the Intel Intrinsics.



Code

- We have implemented a **fully templated** code heavily based on **Trilinos** which provides a fully templated solver stack.
- The **C++** code is embedded in a **Python** interface [Boman]. This eases the looping around samples, the grouping of samples together, etc.
- The software has **hybrid parallelism** based on **Tpetra** with **MPI** for distributed memory and **Kokkos** with **OpenMP** for shared memory.
- It uses **Gmsh** [Geuzaine, 2009] to import 3D meshes and **VTK** to write the output files.
- The code has already generated preliminary results for **industrial thermomechanical contact problems**.

Mechanical contact problem



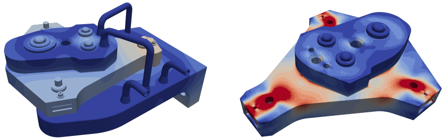
```
$k \leftarrow 0$
Choose an initial guess for the active set $\mathcal{A}_k$
do
    Given $\mathcal{A}_k$, compute the solution of
    $$\left[\begin{array}{cc|cc} \mathbf{K}_{\mathrm{ii}} & \mathbf{K}_{\mathrm{ic}} & \mathbf{0} & \mathbf{0} \\ \mathbf{K}_{\mathrm{ci}} & \mathbf{K}_{\mathrm{cc}} & \mathbf{D}_{\mathcal{I}_k}^{\mathrm{T}} & \mathbf{D}_{\mathcal{A}_k}^{\mathrm{T}} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I}_{\mathcal{I}_k} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{\mathcal{A}_k} & \mathbf{0} & \mathbf{0} \end{array}\right] \left[\begin{array}{c} \mathbf{u}_{\mathrm{i}}^{k+1} \\ \mathbf{u}_{\mathrm{c}}^{k+1} \\ \hline \boldsymbol{\lambda}_{\mathcal{I}_k}^{k+1} \\ \boldsymbol{\lambda}_{\mathcal{A}_k}^{k+1} \end{array}\right] = \left[\begin{array}{c} \mathbf{f}_{\mathrm{i}} \\ \mathbf{f}_{\mathrm{c}} \\ \hline \mathbf{0} \\ \mathbf{g}_{0,\mathcal{A}_k} \end{array}\right]$$
    $\mathcal{A}_{k+1} \leftarrow \left\{ q \in P_{\mathrm{c}}^{h,\mathrm{s}} : \lambda_q^{k+1} + c\,\mathbf{e}_q^{\mathrm{T}} \left(\mathbf{D}\mathbf{u}_{\mathrm{c}}^{k+1} - \mathbf{g}_0\right) > 0 \right\}$
    $k \leftarrow k+1$
while $\mathcal{A}_k \neq \mathcal{A}_{k-1}$
```

Algorithm 2: Active set strategy

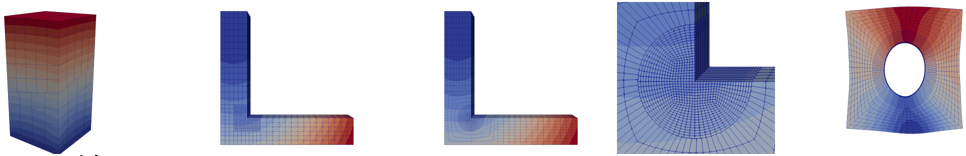
Inner nodes: i, potential contact nodes: c, at iteration k, inactive set: $\mathcal{I}_k$, and active set: $\mathcal{A}_k$.

Code capabilities

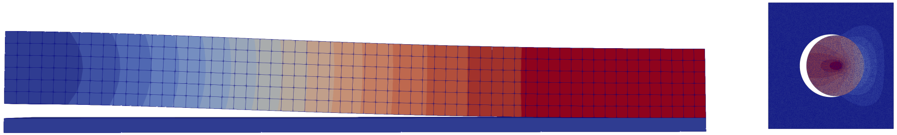
- Monolithic thermoelasticity problems,



- Mesh tying problems,

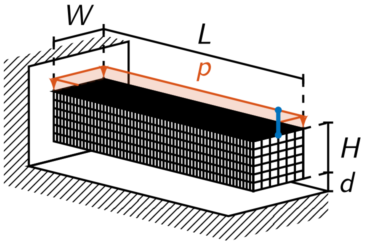


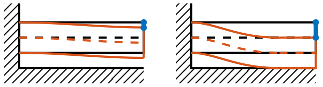
- Contact problems,

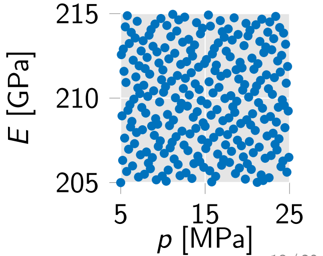


Test case: beam contact problem

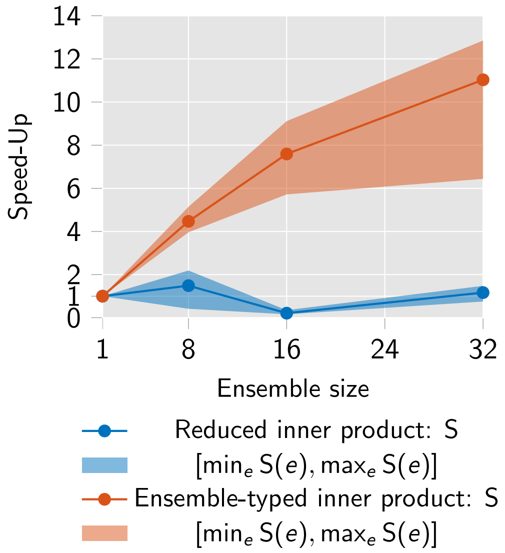
- Size: $L = 50\,\text{cm}, W = 5\,\text{cm}, H = 5\,\text{cm}, d = 1\,\text{cm}$,
- Elements: $60 \times 6 \times 6$ hexahedra,
- Number of Dofs: $9\,394 = 3 \times 61 \times 7^2 + 61 \times 7$,
- Depending on the pressure $p \sim \mathcal{U}(5, 25)$ [MPa], the contact is fully open or partially closed.
- Material:
  - Young's modulus: $E \sim \mathcal{U}(205, 215)$ [GPa].
  - Poisson coefficient: 0.29.
- Quantity of Interest: displacement along z on the center point of the face $x = L$,
- 256 Halton Quasi Monte Carlo samples,
- One MPI process on a Xeon Phi KNL with 256 OpenMP threads.

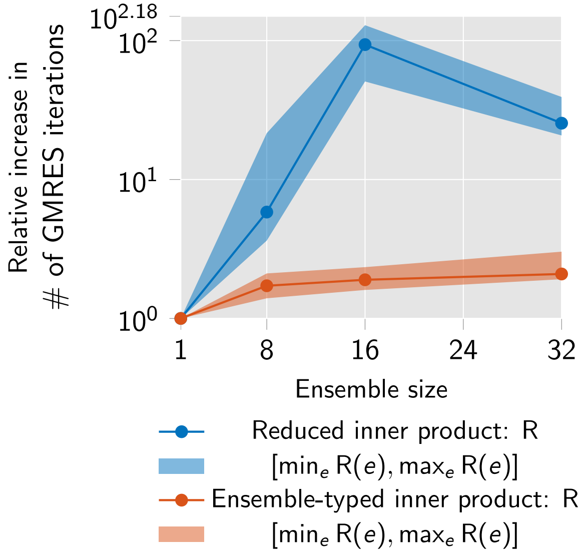






Speed-Up of the full simulation and increased computational work





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:

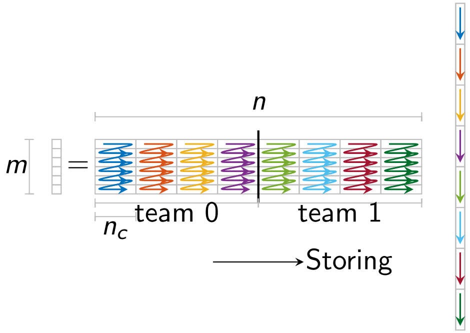
- Profiling study of the EP on mesh tying problem,
- Applying the method on **engineering problems** relevant for **ITER** in collaboration with FZ. Jülich,
- Testing on more than one computational node to leverage the increased memory usage,
- Studying how to use this method in **uncertainty quantification** 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.

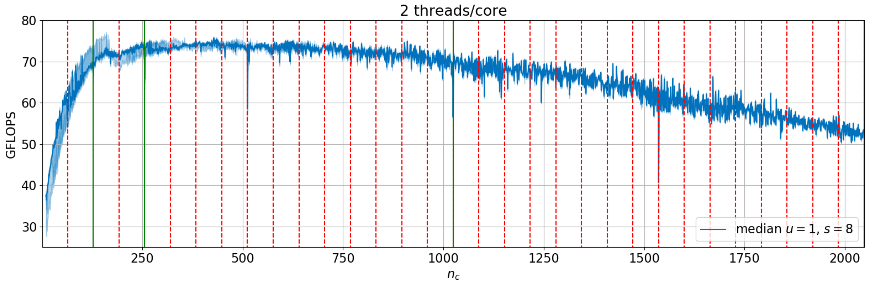
fnrs
FREEDOM TO RESEARCH

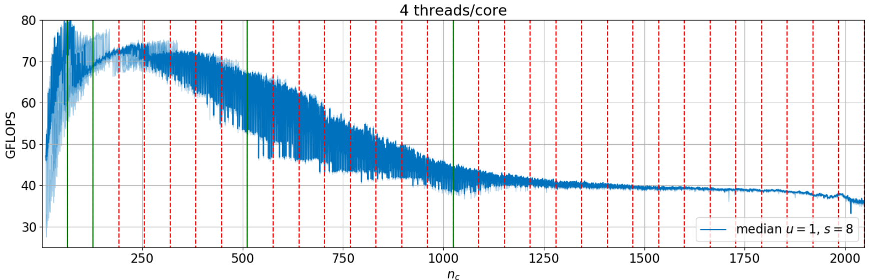
Inner product case



- The atomic adds introduced a fixed cost linked to the desynchronization of the threads that all want to access the first entries of the left-hand side vector at the same time.
- We used a cycling technique such that the threads start at different rows evenly distributed among m. This reduces the desynchronization cost for larger m.
- To reduce the fixed cost for small m, we gather threads per team of 4, do a parallel reduction per team and then do the atomics.

Choice of m_c (or n_c) on KNL

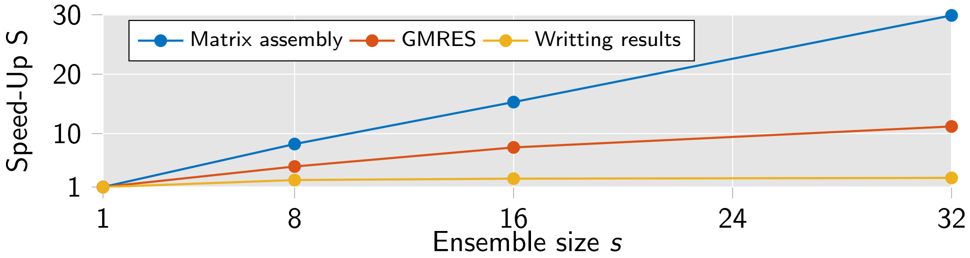




Speed-Up and R

- **Speed-Up:** relative gain in CPU cost (architecture dependent):

$$\mathsf{S}(e) = \frac{\sum_{\ell\in e} \mathsf{Time}_\ell}{\mathsf{Time}_e}, \quad \mathsf{S} = \frac{\sum_e \sum_{\ell\in e} \mathsf{Time}_\ell}{\sum_e \mathsf{Time}_e}.$$



- **R:** relative increase in computational work (architecture independent):

$$\mathsf{R}(e) = \frac{s\,\#\mathsf{iterations}_e}{\sum_{\ell\in e} \#\mathsf{iterations}_\ell}, \quad \mathsf{R} = \frac{s\sum_e \#\mathsf{iterations}_e}{\sum_e \sum_{\ell\in e} \#\mathsf{iterations}_\ell}.$$