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

Kim LIEGEOIS

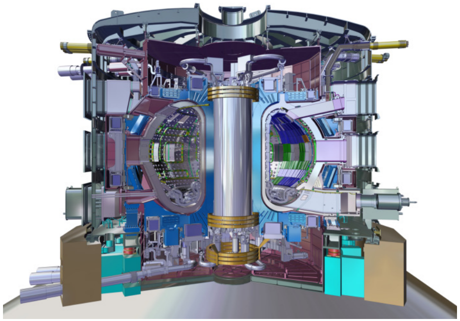
Computational and Stochastic Modeling, Université de Liège, Belgium

PhD Public Defense Sart-Tilman (Liège) Septembre 10, 2020

Outline

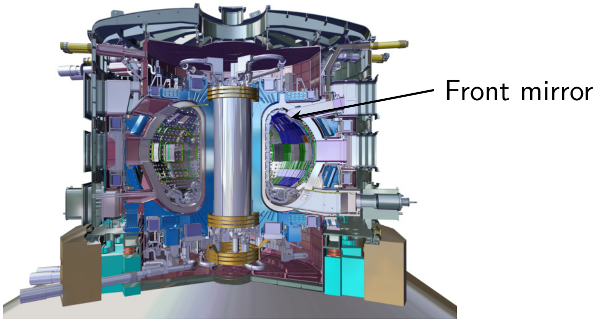
Context: ITER front mirror

Multiphysics models



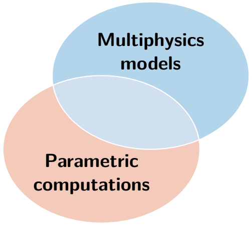
Context: ITER front mirror

Multiphysics models



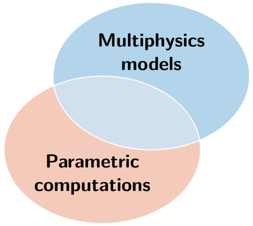
Front mirrors of ITER diagnostic systems are subject to high thermal loads emitted by the plasma which thermally deform the mirror and can reduce the optical quality of the measures.

Context: Parametric computations

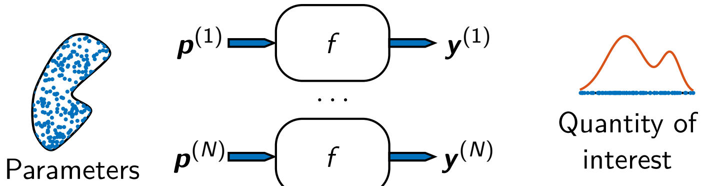


In order to quantify the robustness of the design to uncertainties in the parameters, the numerical model f of the front mirror must be evaluated a large number of times N:

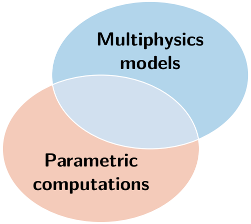
Context: Parametric computations



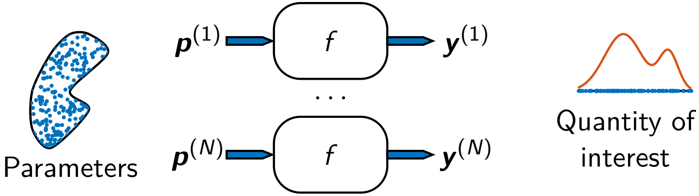
In order to quantify the robustness of the design to uncertainties in the parameters, the numerical model f of the front mirror must be evaluated a large number of times N:



Context: Parametric computations

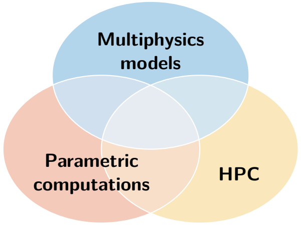


In order to quantify the robustness of the design to uncertainties in the parameters, the numerical model f of the front mirror must be evaluated a large number of times N:



As N is typically large and f costly to evaluate, the total CPU cost of the parametric computation is large.

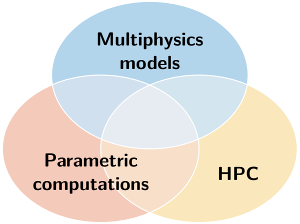
Context: High Performance Computing



High Performance Computing is an opportunity for parametric computation:

- Clusters will allow to run larger and larger number of evaluations concurrently,

Context: High Performance Computing



High Performance Computing is an opportunity for parametric computation:

- Clusters will allow to run larger and larger number of evaluations concurrently,

- The high performance computing library **Trilinos** provides embedded capabilities targeting parametric computations.

Context: Trilinos

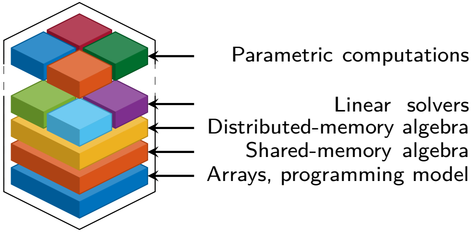
Trilinos is:

- A C++ library which targets the solution of PDE problems;

Context: Trilinos

Trilinos is:

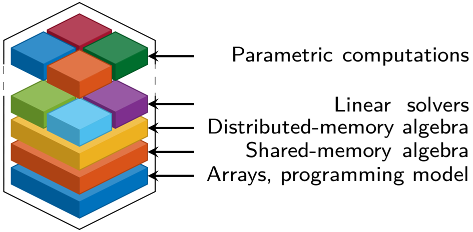
- A C++ library which targets the solution of PDE problems;
- A collection of packages whose purpose ranges from the definition of arrays to the solution of nonlinear transient problems;



Context: Trilinos

Trilinos is:

- A C++ library which targets the solution of PDE problems;
- A collection of packages whose purpose ranges from the definition of arrays to the solution of nonlinear transient problems;
- Open source and hosted on github: `https://github.com/trilinos/Trilinos`;
- Developed by 225 contributors.

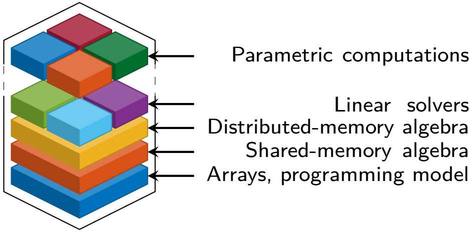


Context: Trilinos

Trilinos is:

- A C++ library which targets the solution of PDE problems;
- A collection of packages whose purpose ranges from the definition of arrays to the solution of nonlinear transient problems;
- Open source and hosted on github: `https://github.com/trilinos/Trilinos`;
- Developed by 225 contributors.

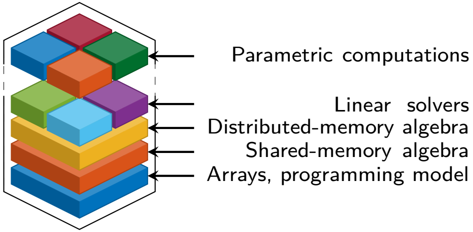
Trilinos provides a solver stack which includes definitions of matrices, linear solvers, nonlinear solvers, preconditioners, . . . **templated on the data type**.



Context: Trilinos

Trilinos is:

- A C++ library which targets the solution of PDE problems;
- A collection of packages whose purpose ranges from the definition of arrays to the solution of nonlinear transient problems;
- Open source and hosted on github: `https://github.com/trilinos/Trilinos`;
- Developed by 225 contributors.



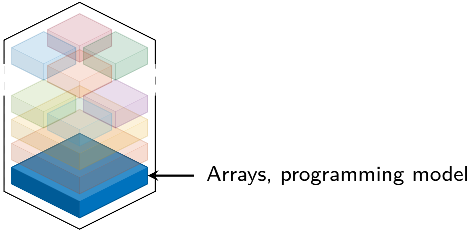
Trilinos provides a solver stack which includes definitions of matrices, linear solvers, nonlinear solvers, preconditioners, . . . **templated on the data type**.

This feature is a key feature for the work presented today.

Context: Kokkos

Kokkos:

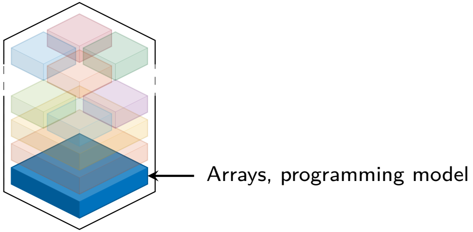
- Is a C++ Performance portability library;
- Enables single source performance portable codes;



Context: Kokkos

Kokkos:

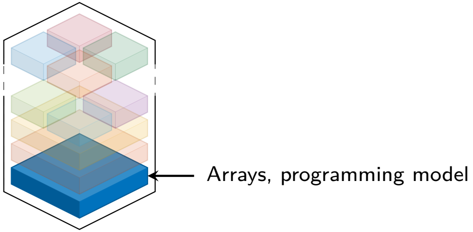
- Is a C++ Performance portability library;
- Enables single source performance portable codes;
- Provides programming model for shared-memory parallelism;
- Provides data abstractions critical for performance portability;



Context: Kokkos

Kokkos:

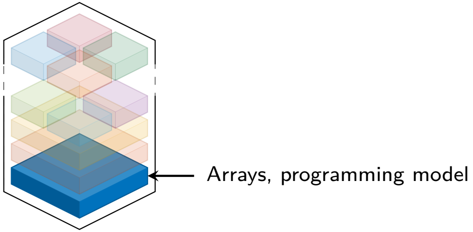
- Is a C++ Performance portability library;
- Enables single source performance portable codes;
- Provides programming model for shared-memory parallelism;
- Provides data abstractions critical for performance portability;
- Is open source and hosted on github: `https://github.com/kokkos/kokkos`;
- Has dedicated developer staff at 5 US National Labs.



Context: Kokkos

Kokkos:

- Is a C++ Performance portability library;
- Enables single source performance portable codes;
- Provides programming model for shared-memory parallelism;
- Provides data abstractions critical for performance portability;
- Is open source and hosted on github: `https://github.com/kokkos/kokkos`;
- Has dedicated developer staff at 5 US National Labs.

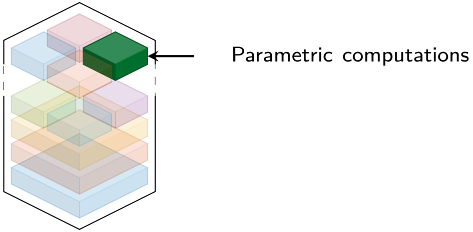


Those **data abstractions** and **programming models** are used in the work presented today to implement the **efficient GEMV** discussed later.

Context: Stokhos

Stokhos:

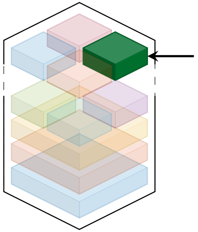
- Is the Trilinos package for embedded uncertainty quantification methods;



Context: Stokhos

Stokhos:

- Is the Trilinos package for embedded uncertainty quantification methods;
- Provides an implementation of the Stochastic Galerkin method;
- Provides an implementation of the embedded ensemble propagation method;

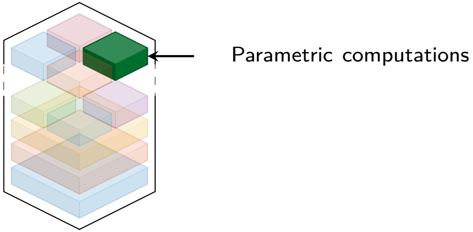


Parametric computations

Context: Stokhos

Stokhos:

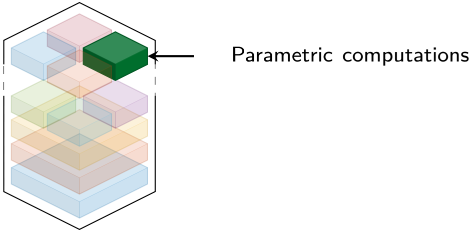
- Is the Trilinos package for embedded uncertainty quantification methods;
- Provides an implementation of the Stochastic Galerkin method;
- Provides an implementation of the embedded ensemble propagation method;
- Is developed by Eric T. Phipps.



Context: Stokhos

Stokhos:

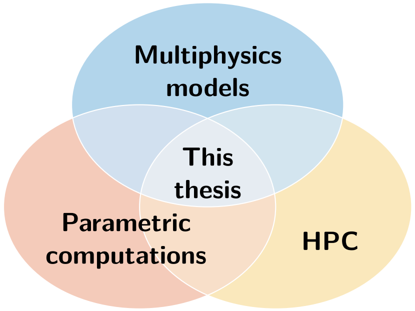
- Is the Trilinos package for embedded uncertainty quantification methods;
- Provides an implementation of the Stochastic Galerkin method;
- Provides an implementation of the embedded ensemble propagation method;
- Is developed by Eric T. Phipps.



The work presented today **uses** and **contributes** to the **embedded ensemble propagation** method.

Objective of the thesis

For a given set of samples, to **reduce** the **wall-clock time** to evaluate multiphysics models on high performance clusters.



This has been done using and contributing to the **embedded ensemble propagation** one of the embedded strategies implemented in **Stokhos**.

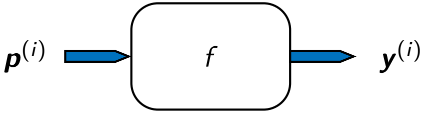
Outline

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.

Given N samples and an ensemble size s, instead of looping over the N samples:

```
for i in range(0,N):
  y[i] = f(p[i])
```

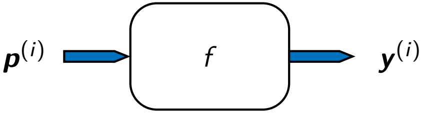


Ensemble propagation

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.

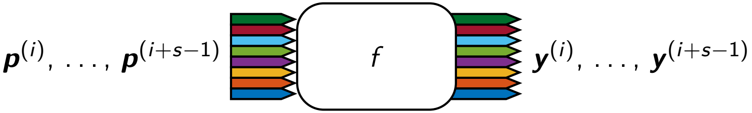
Given N samples and an ensemble size s, instead of looping over the N samples:

```
for i in range(0,N):
  y[i] = f(p[i])
```



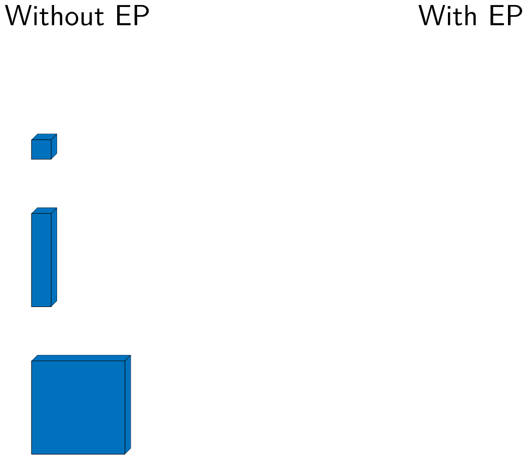
Ensemble propagation loops over sets of size s of the N samples:

```
for i in range(0,N,s):
  y[i:i+s] = f(p[i:i+s])
```


$\boldsymbol{p}^{(i)}, \ldots, \boldsymbol{p}^{(i+s-1)}$
f
$\boldsymbol{y}^{(i)}, \ldots, \boldsymbol{y}^{(i+s-1)}$

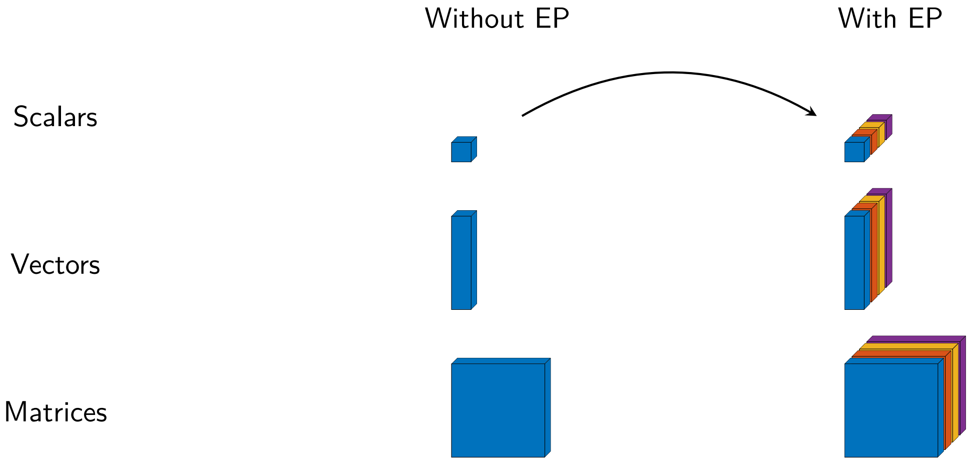
Ensemble propagation

Using EP increases the order of sample-dependent tensors by one:



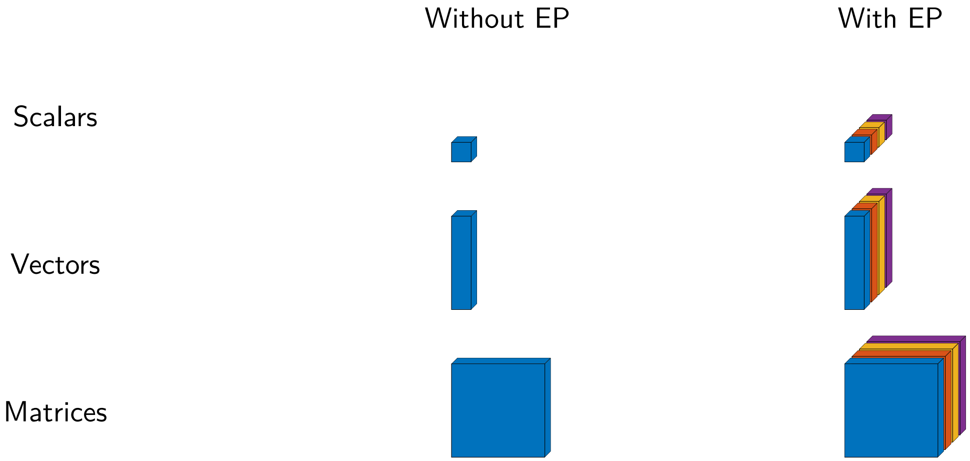
Ensemble propagation

Using EP increases the order of sample-dependent tensors by one:



Ensemble propagation

Using EP increases the order of sample-dependent tensors by one:



Algorithms should be adapted accordingly.

Ensemble propagation

EP was introduced by [Phipps, 2017], made available in **Stokhos**, 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);
  //...
}
```

and providing template specializations for some of the Trilinos functions and classes for this new data type.

Ensemble propagation

EP was introduced by [Phipps, 2017], made available in **Stokhos**, 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);
  //...
}
```

and providing template specializations for some of the Trilinos functions and classes for this new data type.

This implementation strategy allows to use EP in the full solver stack of **Trilinos** supporting templated data types.

Ensemble propagation

Advantages of the EP:

- Reuse of common variables;
- More opportunities for vectorization (more data parallelism);
- Improved memory access pattern;
- Reduction of Message Passing Interface (MPI) latency per sample.

Ensemble propagation

Advantages of the EP:

- Reuse of common variables;
- More opportunities for vectorization (more data parallelism);
- Improved memory access pattern;
- Reduction of Message Passing Interface (MPI) latency per sample.

Challenges of the EP:

- Increased memory usage compared to a single computation;
- Ensemble divergence:
  - control flow divergence:
    - If-then-else divergence;
    - Loop divergence;
  - function call divergence:
    - Missing BLAS functions;
    - What is an inner product of two ensemble typed vectors?

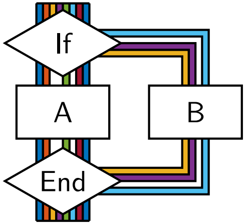
Ensemble propagation

Advantages of the EP:

- Reuse of common variables;
- More opportunities for vectorization (more data parallelism);
- Improved memory access pattern;
- Reduction of Message Passing Interface (MPI) latency per sample.

Challenges of the EP:

- Increased memory usage compared to a single computation;
- Ensemble divergence:
  - control flow divergence:
    - If-then-else divergence;
    - Loop divergence;
  - function call divergence:
    - Missing BLAS functions;
    - What is an inner product of two ensemble typed vectors?



Outline

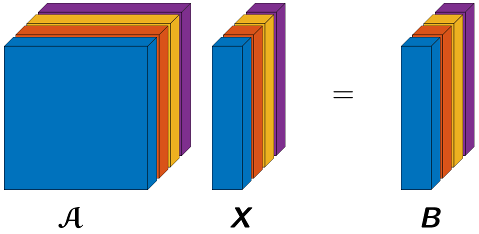
Parametric linear systems

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 sparse and not symmetric or not positive definite.

Representation of a system for $s = 4$:



How to solve the sparse parametric linear system efficiently with EP?

Right-preconditioned GMRES

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

Right-preconditioned GMRES

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

The GMRES method iteratively creates an **orthonormal basis** $\boldsymbol{v}_{:1}, \ldots, \boldsymbol{v}_{:j}$ for the vector space:

$$\mathrm{span}\left\{\boldsymbol{r}^{(0)}, \boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}, \ldots, \left(\boldsymbol{A}\boldsymbol{M}^{-1}\right)^{(j-1)}\boldsymbol{r}^{(0)}\right\}$$

using the orthonarmal basis of the current iteration $\boldsymbol{v}_{:1}, \ldots, \boldsymbol{v}_{:j}$, applying $\boldsymbol{A}\boldsymbol{M}^{-1}$ to the last vector $\boldsymbol{v}_{:j}$ and **orthonormalize** it with $\boldsymbol{v}_{:1}, \ldots, \boldsymbol{v}_{:j}$.

This process continues up to the point where the basis allows to have a sufficiently small error to the solution of the linear system.

Right-preconditioned GMRES

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

The GMRES method iteratively creates an **orthonormal basis** $v_{:1}, \ldots, v_{:j}$ for the vector space:

$$\operatorname{span}\left\{r^{(0)}, AM^{-1}r^{(0)}, \ldots, \left(AM^{-1}\right)^{(j-1)} r^{(0)}\right\}$$

using the orthonarmal basis of the current iteration $v_{:1}, \ldots, v_{:j}$, applying AM^{-1} to the last vector $v_{:j}$ and **orthonormalize** it with $v_{:1}, \ldots, v_{:j}$.

This process continues up to the point where the basis allows to have a sufficiently small error to the solution of the linear system.

What is an **inner product** of two **ensemble typed vectors**?

Reduced inner product

First approach [Phipps, 2017]: equivalent to gathering the sample matrices $\boldsymbol{A}_{::1}, \ldots, \boldsymbol{A}_{::s}$ into a block diagonal matrix

$$\begin{bmatrix} \boldsymbol{A}_{::1} & & \\ & \ddots & \\ & & \boldsymbol{A}_{::s} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{:1} \\ \vdots \\ \boldsymbol{x}_{:s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{:1} \\ \vdots \\ \boldsymbol{b}_{:s} \end{bmatrix},$$

and to applying an iterative method on the block diagonal system.

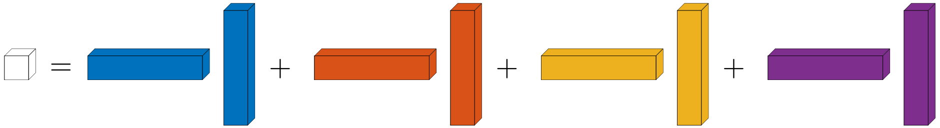
Reduced inner product

First approach [Phipps, 2017]: equivalent to gathering the sample matrices $\boldsymbol{A}_{::1}, \dots, \boldsymbol{A}_{::s}$ into a block diagonal matrix

$$\begin{bmatrix} \boldsymbol{A}_{::1} & & \\ & \ddots & \\ & & \boldsymbol{A}_{::s} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{:1} \\ \vdots \\ \boldsymbol{x}_{:s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{:1} \\ \vdots \\ \boldsymbol{b}_{:s} \end{bmatrix},$$

and to applying an iterative method on the block diagonal system.

This is mathematically equivalent to defining a **reduced inner product**:



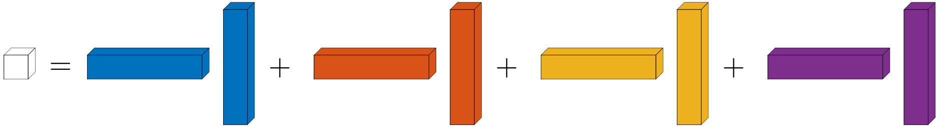
Reduced inner product

First approach [Phipps, 2017]: equivalent to gathering the sample matrices $\boldsymbol{A}_{::1}, \ldots, \boldsymbol{A}_{::s}$ into a block diagonal matrix

$$\begin{bmatrix} \boldsymbol{A}_{::1} & & \\ & \ddots & \\ & & \boldsymbol{A}_{::s} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{:1} \\ \vdots \\ \boldsymbol{x}_{:s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{:1} \\ \vdots \\ \boldsymbol{b}_{:s} \end{bmatrix},$$

and to applying an iterative method on the block diagonal system.

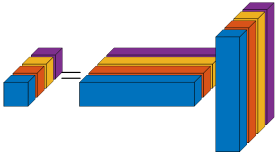
This is mathematically equivalent to defining a **reduced inner product**:



Advantages: No ensemble divergence and possibility to use efficient BLAS implementations,
Challenges: The samples are coupled together, the spectra of $\boldsymbol{A}_{::1}, \ldots, \boldsymbol{A}_{::s}$ are gathered, and the condition number increases and is larger than the ones of $\boldsymbol{A}_{::1}, \ldots, \boldsymbol{A}_{::s}$. The number of iterations increases.

Ensemble-typed inner product

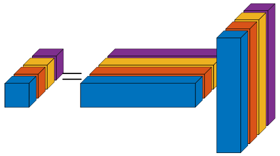
Second approach [D'Elia, 2020]: to avoid the coupling of the samples together using an **ensemble-typed inner product**:



It was first introduced for grouping purpose.

Ensemble-typed inner product

Second approach [D'Elia, 2020]: to avoid the coupling of the samples together using an **ensemble-typed inner product**:



It was first introduced for grouping purpose.

Advantage: No coupling: each sample converges as fast as if it was propagated alone,
Challenge: Every ensemble divergence has to be managed explicitly.

Advantages and challenges of both approaches

With ensemble reduction:

Advantages:

- **No control flow divergence**;
- Use of **standard libraries** such as the Intel Math Kernel Library.

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.

Advantages and challenges of both approaches

With ensemble reduction:

Advantages:

- **No control flow divergence**;
- Use of **standard libraries** such as the Intel Math Kernel Library.

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.

Without ensemble reduction:

Advantages:

- Convergence of an iterative method implies the convergence for every sample;
- The spectra **are not** gathered;
- Global convergence rate **controlled** by the slowest sample.

Challenges:

- **Control flow divergence** has to be treated explicitly;
- **No current** implementation of the needed BLAS routines in the Intel Math Kernel Library.

Outline

Occurrence of ensemble divergence in GMRES

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

Ensemble divergence in GMRES:

1. a vector can require a normalization or not: **if-then-else divergence**;

Occurrence of ensemble divergence in GMRES

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

Ensemble divergence in GMRES:

1. a vector can require a normalization or not: **if-then-else divergence**;
2. different samples may require different numbers of iterations to converge: **loop divergence**;

Control flow divergence: example of the normalization

The ensemble class comes with some **utility functions** which allow to access its *i*-th element. Therefore, it is feasible to **explicitly loop over the samples** of the ensemble and evaluate the if-statement for each sample one at the time in the normalization process.

```
  typedef EnsembleTrait<T> ET;
  const int s = ET::ensemble_size;
  bool all_zeros = true;
```

Control flow divergence: example of the normalization

The ensemble class comes with some **utility functions** which allow to access its *i*-th element. Therefore, it is feasible to **explicitly loop over the samples** of the ensemble and evaluate the if-statement for each sample one at the time in the normalization process.

```
typedef EnsembleTrait<T> ET;
const int s = ET::ensemble_size;
bool all_zeros = true;

T norm_inv;
for (int l = 0; l < s; ++l)
  if (ET::coeff(norm, l) > 0) {
    ET::coeff(norm_inv, l) = 1. / ET::coeff(norm, l);
    all_zeros = false;
  }
  else
    ET::coeff(norm_inv, l) = 0.;
```

Control flow divergence: example of the normalization

The ensemble class comes with some **utility functions** which allow to access its *i*-th element. Therefore, it is feasible to **explicitly loop over the samples** of the ensemble and evaluate the if-statement for each sample one at the time in the normalization process.

```
typedef EnsembleTrait<T> ET;
const int s = ET::ensemble_size;
bool all_zeros = true;

T norm_inv;
for (int l = 0; l < s; ++l)
  if (ET::coeff(norm, l) > 0) {
    ET::coeff(norm_inv, l) = 1. / ET::coeff(norm, l);
    all_zeros = false;
  }
  else
    ET::coeff(norm_inv, l) = 0.;

if (all_zeros) return has_converged;

for (int i = 0; i < n; ++i)
  v[i] *= norm_inv;
```

Control flow divergence: example of the normalization

The ensemble class comes with some **utility functions** which allow to access its *i*-th element. Therefore, it is feasible to **explicitly loop over the samples** of the ensemble and evaluate the if-statement for each sample one at the time in the normalization process.

```
typedef EnsembleTrait<T> ET;
const int s = ET::ensemble_size;
bool all_zeros = true;

T norm_inv;
for (int l = 0; l < s; ++l)
  if (ET::coeff(norm, l) > 0) {
    ET::coeff(norm_inv, l) = 1. / ET::coeff(norm, l);
    all_zeros = false;
  }
  else
    ET::coeff(norm_inv, l) = 0.;

if (all_zeros) return has_converged;

for (int i = 0; i < n; ++i)
  v[i] *= norm_inv;
```

This is **correct**, but **not concise** and relies on the **compiler** to optimize the code.

Control flow divergence: Mask class

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

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

which stores the result of any comparison of ensembles sample-wise.

Control flow divergence: Mask class

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

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

which stores the result of any comparison of ensembles sample-wise.

This mask can then used for **masked assignments** and **logical reductions**:

```
T norm_inv;

if (AND(norm == 0)) return has_converged;

MaskAssign(norm > 0, norm_inv) /= {1., norm, 0.};

for (int i = 0; i < n; ++i)
  v[i] *= norm_inv;
```

Control flow divergence: Mask class

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

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

which stores the result of any comparison of ensembles sample-wise.

This mask can then used for **masked assignments** and **logical reductions**:

```
T norm_inv;

if (AND(norm == 0)) return has_converged;

MaskAssign(norm > 0, norm_inv) /= {1., norm, 0.};

for (int i = 0; i < n; ++i)
  v[i] *= norm_inv;
```

This second implementation is **more concise**, potentially **more readable**, and **helps** the optimization performed by the compiler.

Occurrence of ensemble divergence in GMRES

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

Ensemble divergence in GMRES:

1. a vector can require a normalization or not: **if-then-else divergence**;
2. different samples may require different numbers of iterations to converge: **loop divergence**;
1. called BLAS functions, such as GEMV for the dense matrix-vector operations in the orthogonalization process (inner products and update), may not support ensemble-typed inputs, leading to **function call divergence**.

GEMV with ensemble propagation for the update of GMRES

The **GEMV** with ensemble propagation takes the form of **tensors contractions** as follows:

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

GEMV with ensemble propagation for the update of GMRES

The **GEMV** with ensemble propagation takes the form of **tensors contractions** as follows:

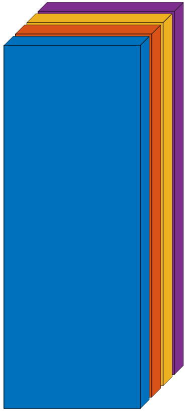
$$\boldsymbol{y}_{:\ell} = \beta_\ell \, \boldsymbol{y}_{:\ell} + \alpha_\ell \, \boldsymbol{A}_{::\ell} \, \boldsymbol{x}_{:\ell} \quad \text{for all} \quad \ell = 1, \dots, s,$$

Interleaved memory layout of the $n \times j \times s$ third-order tensor $\mathcal{A}$ due to EP:

$$a_{ik\ell} \leftarrow\!\!\shortmid a\left[(i-1)\,s + (k-1)\,n\,s + (\ell-1)\right],$$

with n the number of degrees of freedom per sample, j the Krylov subspace dimension, and s the ensemble size;

Tall skinny third-order tensor $\mathcal{A}$



GEMV with ensemble propagation for the update of GMRES

The **GEMV** with ensemble propagation takes the form of **tensors contractions** as follows:

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

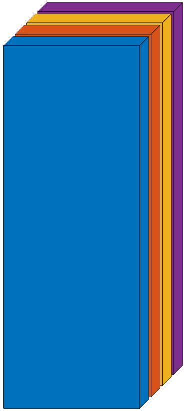
Interleaved memory layout of the $n \times j \times s$ third-order tensor $\mathcal{A}$ due to EP:

$$a_{ik\ell} \leftarrow\!\shortmid \, a\left[(i-1)\,s + (k-1)\,n\,s + (\ell-1)\right],$$

with n the number of degrees of freedom per sample, j the Krylov subspace dimension, and s the ensemble size;

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

Tall skinny third-order tensor $\mathcal{A}$



GEMV with ensemble propagation for the update of GMRES

Such an operation has a **low arithmetic intensity** as, for every $a_{ik\ell}$ 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** providing that both implementation reach maximal throughput.

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 core 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.

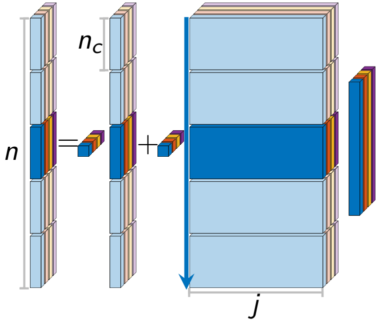
GEMV with ensemble propagation for the update of GMRES

```
parfor $t = 1$ to $n - n_c + 1$ by $m_c$ do
    for $i = t, \dots, t + n_c - 1$ do
        $y_{i\ell} = \beta_\ell \, y_{i\ell}$   for all   $\ell = 1, \dots, s$
    for $k = 1, \dots, j$ do
        $\gamma_\ell = \alpha_\ell \, x_{k\ell}$   for all   $\ell = 1, \dots, s$
        for $i = t, \dots, t + n_c - 1$ do
            $y_{i\ell} = y_{i\ell} + \gamma_\ell \, a_{ik\ell}$   for all   $\ell = 1, \dots, s$
```

GEMV with ensemble propagation for the update of GMRES

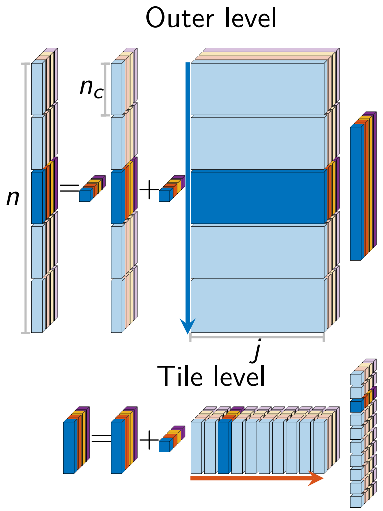
```
parfor $t = 1$ to $n - n_c + 1$ by $m_c$ do
    for $i = t, \dots, t + n_c - 1$ do
        $y_{i\ell} = \beta_\ell \, y_{i\ell}$   for all   $\ell = 1, \dots, s$
    for $k = 1, \dots, j$ do
        $\gamma_\ell = \alpha_\ell \, x_{k\ell}$   for all   $\ell = 1, \dots, s$
        for $i = t, \dots, t + n_c - 1$ do
            $y_{i\ell} = y_{i\ell} + \gamma_\ell \, a_{ik\ell}$   for all   $\ell = 1, \dots, s$
```

Outer level



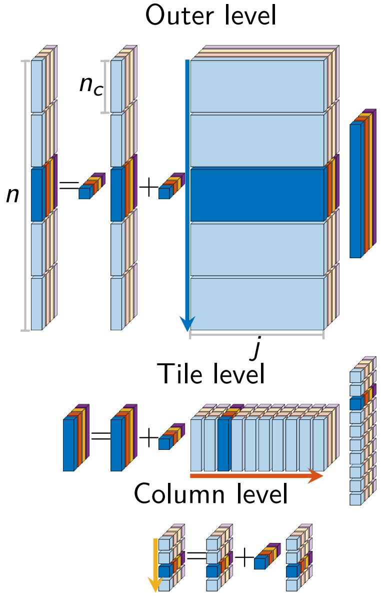
GEMV with ensemble propagation for the update of GMRES

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



GEMV with ensemble propagation for the update of GMRES

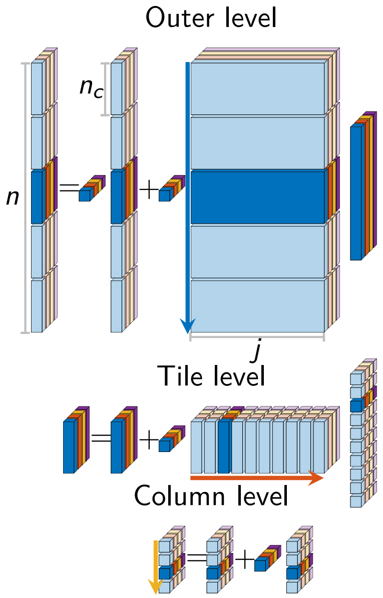
```
parfor $t = 1$ to $n - n_c + 1$ by $m_c$ do
    for $i = t, \ldots, t + n_c - 1$ do
        $y_{i\ell} = \beta_\ell \, y_{i\ell}$   for all   $\ell = 1, \ldots, s$
    for $k = 1, \ldots, j$ do
        $\gamma_\ell = \alpha_\ell \, x_{k\ell}$   for all   $\ell = 1, \ldots, s$
        for $i = t,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



GEMV with ensemble propagation for the update of GMRES

```
parfor $t = 1$ to $n - n_c + 1$ by $m_c$ do
    for $i = t, \dots, t + n_c - 1$ do
        $y_{i\ell} = \beta_\ell \, y_{i\ell}$   for all   $\ell = 1, \dots, s$
    for $k = 1, \dots, j$ do
        $\gamma_\ell = \alpha_\ell \, x_{k\ell}$   for all   $\ell = 1, \dots, s$
        for $i = t, \dots, t + n_c - 1$ do
            $y_{i\ell} = y_{i\ell} + \gamma_\ell \, a_{ik\ell}$   for all   $\ell = 1, \dots, s$
```

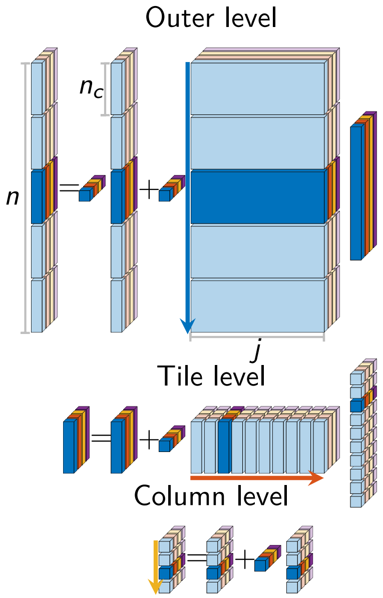
- Tiling:
  - Each thread applies a tile at a time;
  - Cache blocking of $\boldsymbol{Y}$.
- Vectorization:
  - Vectorization of the loops over the samples;
  - Intel Intrinsics or overloaded operators.



GEMV with ensemble propagation for the update of GMRES

```
parfor $t = 1$ to $n - n_c + 1$ by $m_c$ do
    for $i = t, \dots, t + n_c - 1$ do
        $y_{i\ell} = \beta_\ell \, y_{i\ell}$   for all   $\ell = 1, \dots, s$
    for $k = 1, \dots, j$ do
        $\gamma_\ell = \alpha_\ell \, x_{k\ell}$   for all   $\ell = 1, \dots, s$
        for $i = t, \dots, t + n_c - 1$ do
            $y_{i\ell} = y_{i\ell} + \gamma_\ell \, a_{ik\ell}$   for all   $\ell = 1, \dots, s$
```

- Tiling:
  - Each thread applies a tile at a time;
  - Cache blocking of $\boldsymbol{Y}$.
- Vectorization:
  - Vectorization of the loops over the samples;
  - Intel Intrinsics or overloaded operators.
- Choice of the tile size n_c to keep $\boldsymbol{Y}_{(t:t+n_c-1):}$ in cache.



Choice of the tile size n_c: example on Intel 8100-Series (Skylake)

Measured bandwidth (1 NUMA region):

101.2108 GB/s

Deduced maximal throughput:

25 GFLOPS

Parameters:

- Third order tensor $\mathcal{A}$ of size $n \times j \times s$;
- $n = 768000$, $j = 300$, s is the ensemble size;
- only evaluated n_c such that n is a multiple of Nn_c where $N = 48$ is the number of threads.

The highlighted value of $n_c^{\max} s = 32768$ corresponds to the case where $\boldsymbol{Y}_{(t:t+n_c-1):}$ takes at most half of the L2 cache [Goto, 2008].

Choice of the tile size n_c: example on Intel 8100-Series (Skylake)

Measured bandwidth (1 NUMA region):

101.2108 GB/s

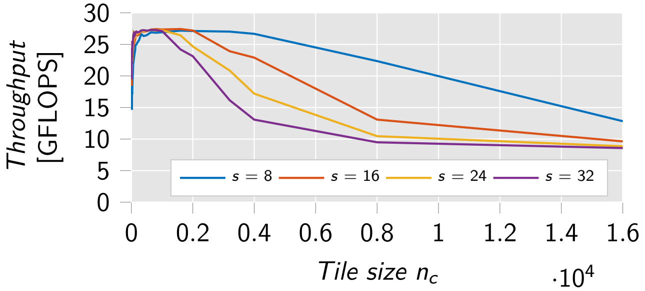
Deduced maximal throughput:

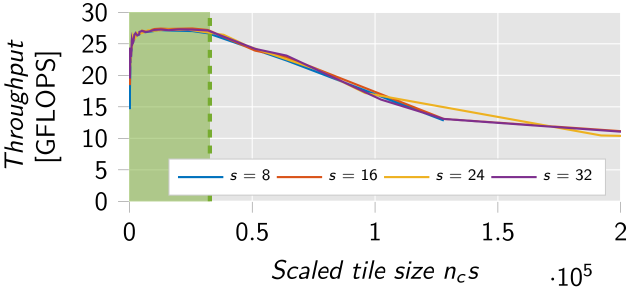
25 GFLOPS

Parameters:

- Third order tensor $\mathcal{A}$ of size $n \times j \times s$;
- $n = 768000$, $j = 300$, s is the ensemble size;
- only evaluated n_c such that n is a multiple of Nn_c where $N = 48$ is the number of threads.

The highlighted value of $n_c^{\max} s = 32768$ corresponds to the case where $\boldsymbol{Y}_{(t:t+n_c-1):}$ takes at most half of the L2 cache [Goto, 2008].





GEMV: results - Intel(R) Xeon(R) Platinum 8160 CPU

The tile size is chosen based on n and $n_c^{\max}$ to have evenly distribute work among the threads:

$$n_c = \left\lceil \frac{n}{N \left\lceil \frac{n}{N\, n_c^{\max}} \right\rceil} \right\rceil.$$

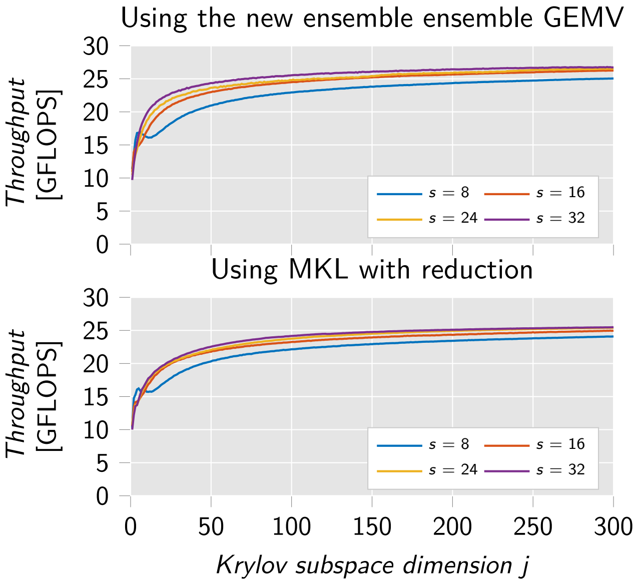
GEMV: results - Intel(R) Xeon(R) Platinum 8160 CPU

The tile size is chosen based on n and $n_c^{\max}$ to have evenly distribute work among the threads:

$$n_c = \left\lceil \frac{n}{N \left\lceil \frac{n}{N\, n_c^{\max}} \right\rceil} \right\rceil.$$

Parameters:

- Threads $N = 48$;
- $n = 30000$;
- $n_c^{\max} = 4096, 2048, 1365,$ and 1024 for $s = 8, 16, 24,$ and 32 respectively.



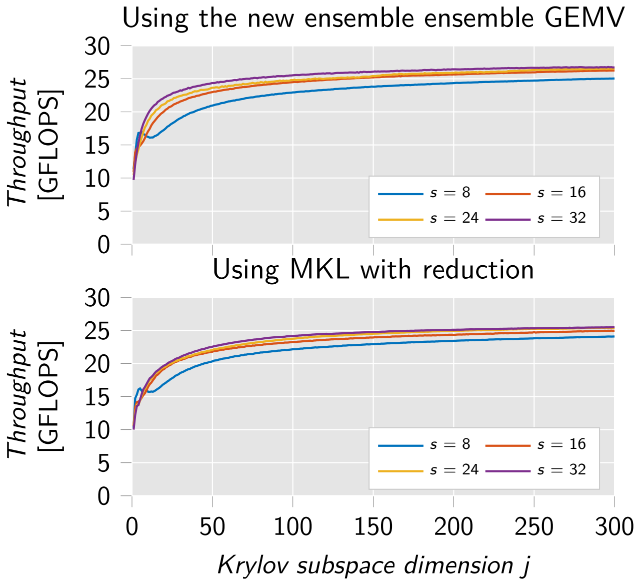
GEMV: results - Intel(R) Xeon(R) Platinum 8160 CPU

The tile size is chosen based on n and $n_c^{\max}$ to have evenly distribute work among the threads:

$$n_c = \left\lceil \frac{n}{N \left\lceil \frac{n}{N\, n_c^{\max}} \right\rceil} \right\rceil.$$

Parameters:

- Threads $N = 48$;
- $n = 30000$;
- $n_c^{\max} = 4096, 2048, 1365,$ and 1024 for $s = 8, 16, 24,$ and 32 respectively.



GEMV: results - Intel(R) Xeon(R) Platinum 8160 CPU

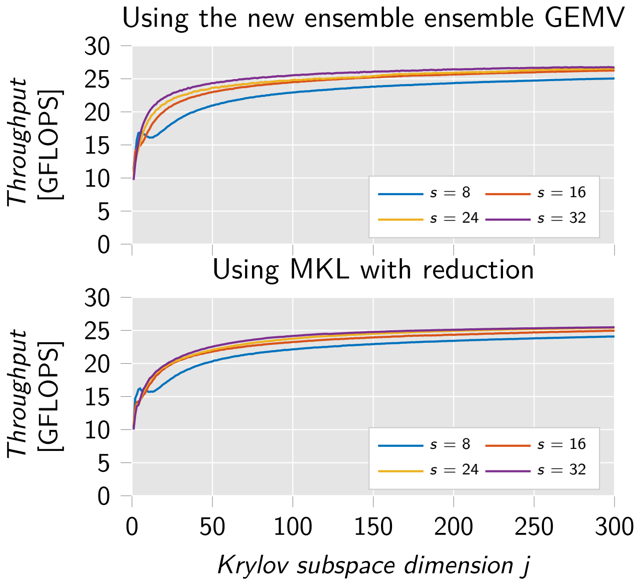
The tile size is chosen based on n and $n_c^{\max}$ to have evenly distribute work among the threads:

$$n_c = \left\lceil \frac{n}{N \left\lceil \frac{n}{N\, n_c^{\max}} \right\rceil} \right\rceil.$$

Parameters:

- Threads $N = 48$;
- $n = 30000$;
- $n_c^{\max} = 4096, 2048, 1365,$ and 1024 for $s = 8, 16, 24,$ and 32 respectively.

Performance similar to the ensemble reduction with MKL used to compute the update.



K. Liegeois, R. Boman, E. T. Phipps, T. Wiesner, M. Arnst, GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models, *Computer Methods in Applied Mechanics and Engineering*, 2020

In [Liegeois, 2020], we describe and implement an efficient ensemble GMRES without ensemble reduction.

K. Liegeois, R. Boman, E. T. Phipps, T. Wiesner, M. Arnst, GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models, *Computer Methods in Applied Mechanics and Engineering*, 2020

In [Liegeois, 2020], we describe and implement an efficient ensemble GMRES without ensemble reduction.
The implementation is available in **Stokhos**. This implementation includes **template specialization** of GMRES related classes and of the GEMV function for ensemble type and the definition of the Mask. The implementation of the GEMV function relies on the **Kokkos** programming model.

K. Liegeois, R. Boman, E. T. Phipps, T. Wiesner, M. Arnst, GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models, *Computer Methods in Applied Mechanics and Engineering*, 2020

In [Liegeois, 2020], we describe and implement an efficient ensemble GMRES without ensemble reduction.
The implementation is available in **Stokhos**. This implementation includes **template specialization** of GMRES related classes and of the GEMV function for ensemble type and the definition of the Mask. The implementation of the GEMV function relies on the **Kokkos** programming model.

The control flow divergence and the function call divergence have been solved by:

- Implementing a **Mask** class which is used to apply masked assignment and logical reduction;
- Implementing an efficient **ensemble GEMV** for the orthogonalization process.

K. Liegeois, R. Boman, E. T. Phipps, T. Wiesner, M. Arnst, GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models, *Computer Methods in Applied Mechanics and Engineering*, 2020

In [Liegeois, 2020], we describe and implement an efficient ensemble GMRES without ensemble reduction.
The implementation is available in **Stokhos**. This implementation includes **template specialization** of GMRES related classes and of the GEMV function for ensemble type and the definition of the Mask. The implementation of the GEMV function relies on the **Kokkos** programming model.

The control flow divergence and the function call divergence have been solved by:

- Implementing a **Mask** class which is used to apply masked assignment and logical reduction;
- Implementing an efficient **ensemble GEMV** for the orthogonalization process.

Those two contributions lead to:

- An equivalent cost per iteration of ensemble GMRES with and without reduction;
- A safe implementation which is able to deal with early-converged samples.

Impact of the reduction on the convergence of ensemble GMRES

Example using a discretized Dirichlet problem for the 1D Laplacian with $n = 6$:

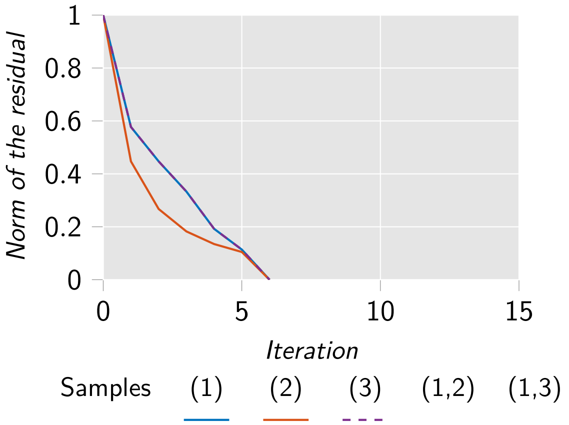
$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b},$$

where:

$$\boldsymbol{A} = \kappa \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Samples are considered:

1. $\kappa = 1$, $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$,
2. $\kappa = 1$, $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$,
3. $\kappa = 1.5$, $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$.



Impact of the reduction on the convergence of ensemble GMRES

Example using a discretized Dirichlet problem for the 1D Laplacian with $n = 6$:

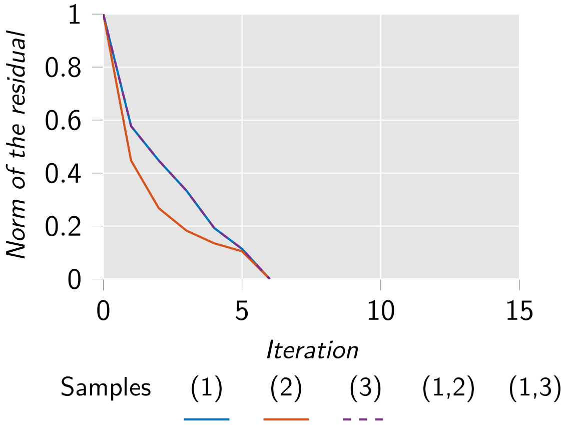
$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b},$$

where:

$$\boldsymbol{A} = \kappa \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Samples are considered:

1. $\kappa = 1$, $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$,
2. $\kappa = 1$, $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$,
3. $\kappa = 1.5$, $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$.



Impact of the reduction on the convergence of ensemble GMRES

Example using a discretized Dirichlet problem for the 1D Laplacian with $n = 6$:

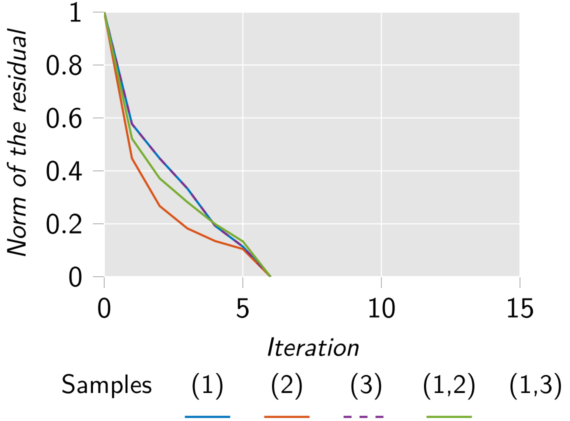
$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b},$$

where:

$$\boldsymbol{A} = \kappa \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Samples are considered:

1. $\kappa = 1,\ \boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix},$
2. $\kappa = 1,\ \boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$
3. $\kappa = 1.5,\ \boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$



Coupling 1 and 2 changes the convergence but not the number of iterations,

Impact of the reduction on the convergence of ensemble GMRES

Example using a discretized Dirichlet problem for the 1D Laplacian with $n = 6$:

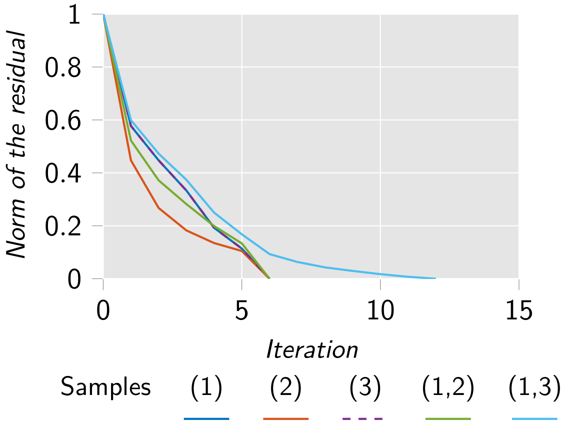
$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b},$$

where:

$$\boldsymbol{A} = \kappa \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Samples are considered:

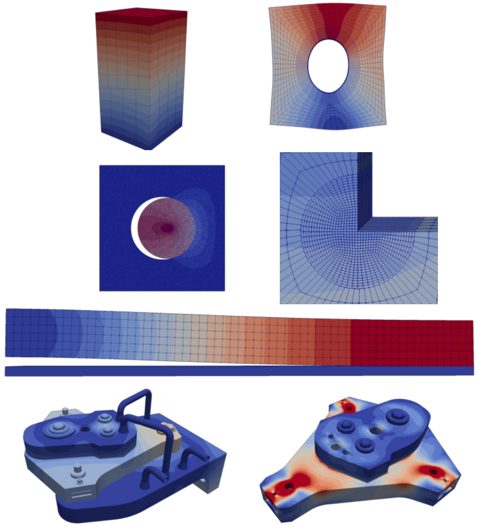
1. $\kappa = 1,\ \boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix},$
2. $\kappa = 1,\ \boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$
3. $\kappa = 1.5,\ \boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$



Coupling 1 and 2 changes the convergence but not the number of iterations,
Coupling 1 and 3 multiply by 2 the number of iterations.

Katoptron

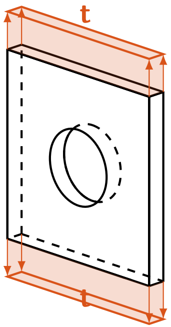
- **Fully templated C++** code heavily based on **Trilinos** which provides a fully templated solver stack;
- Embedded in a **Python** interface. This eases the looping around samples, the grouping of samples together, etc;
- **Hybrid parallelism** based on **Tpetra** with **MPI** for distributed memory and **Kokkos** with **OpenMP** for shared memory;
- Uses **Gmsh** to import 3D meshes and **VTK** to write the output files;
- **Open source** and freely available on `https://gitlab.uliege.be/am-dept/waves.`

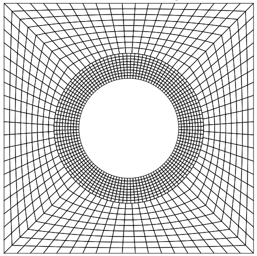


Outline

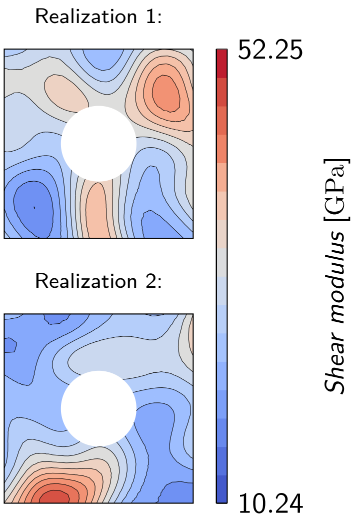
Mesh-tying problem

- Plate with a hole pulled on two opposite sides;
- Two meshes glued with the **Mortar finite element method** in saddle point formulation;
- Lamé parameters represented as a **lognormal random field**;
- Multigrid preconditioner with saddle point matrix on each multigrid level;
- 32088 degrees of freedom per sample;
- 1 Intel Skylake CPU.





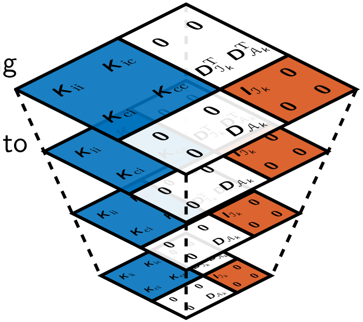
Lamé parameters:



Preconditioners: Full multigrid approach

Introduced in [Wiesner, 2015] for **contact problem**.

- Main idea: use **coarser representations** of fine level problems in order to **speed up** the solution process;
- Uses the multigrid approach on **the full matrix**, preserving the **saddle-point structure** on all levels;
- Algebraic multigrid: **no special information** is necessary to build the multigrid **hierarchies**;
- Mutligrid hierarchies are **independent of the activity** of the Lagrange multipliers;
- Implemented in **MueLu** with already existing codes and contributions of this thesis;
- Due to EP, **level matrices** are now **third-order tensors** but **prolongation** and **restriction** operators are **sample independent**.



Mesh-tying example: speed-up of one GMRES iteration

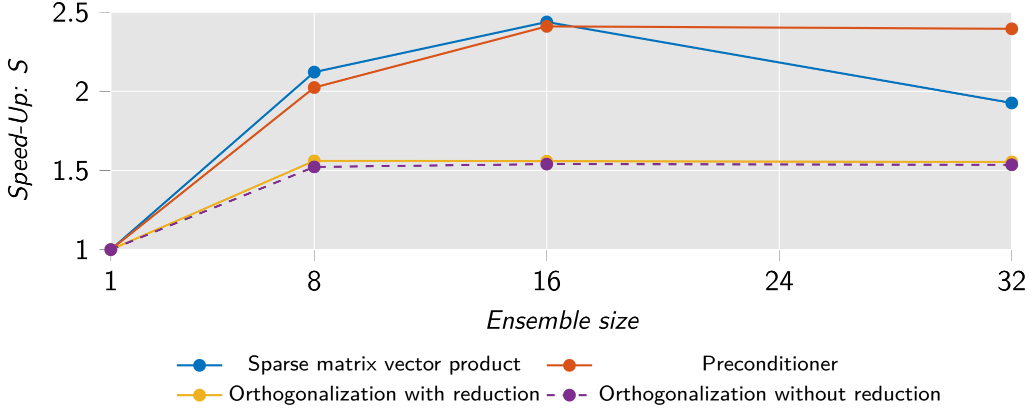
Speed-up: relative gain in wall-clock time (architecture dependent):

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

Mesh-tying example: speed-up of one GMRES iteration

Speed-up: relative gain in wall-clock time (architecture dependent):

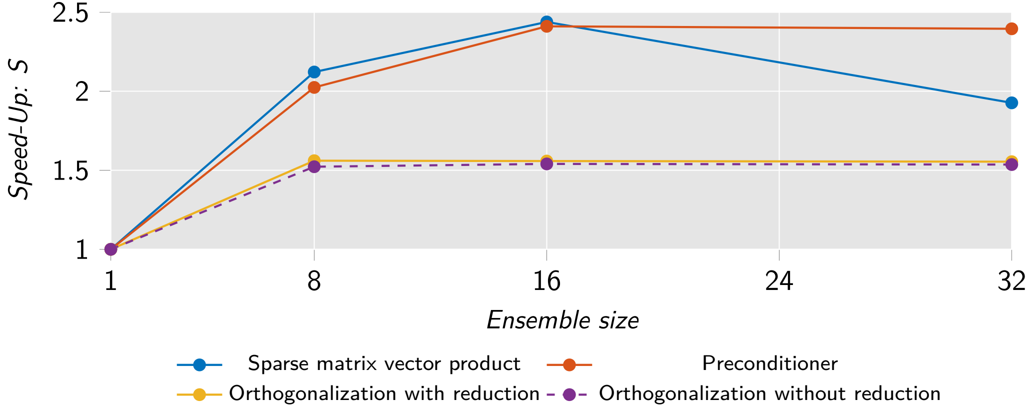
$$\mathsf{S}(e) = \frac{\sum_{\ell \in e} \mathsf{Time}_\ell}{\mathsf{Time}_e}.$$



Mesh-tying example: speed-up of one GMRES iteration

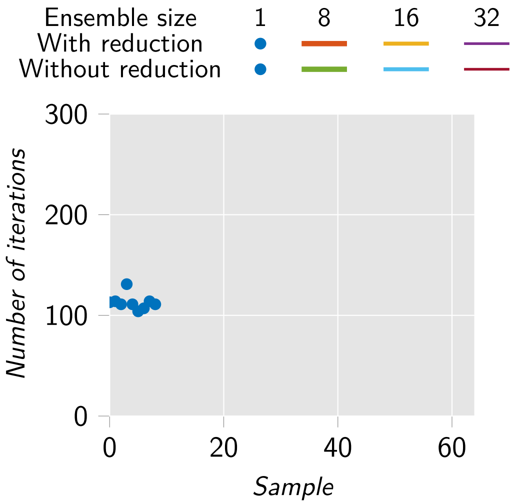
Speed-up: relative gain in wall-clock time (architecture dependent):

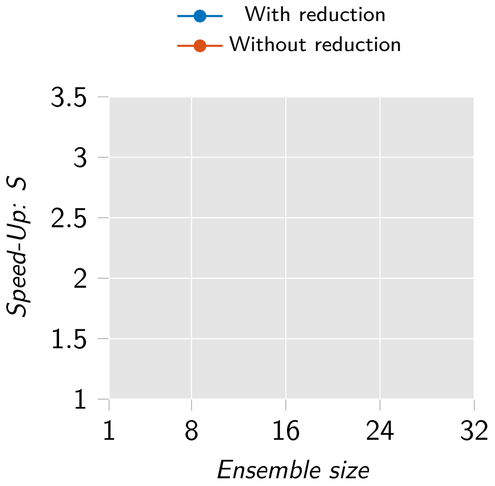
$$S(e) = \frac{\sum_{\ell \in e} \mathrm{Time}_\ell}{\mathrm{Time}_e}.$$



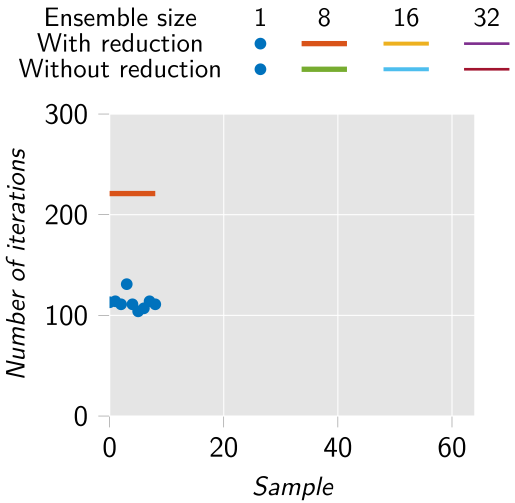
The speed-up per iteration is nearly independent of the use of ensemble reduction.

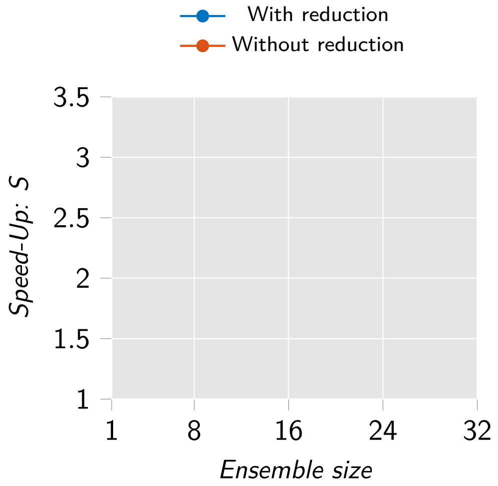
Mesh-tying example: convergence and total speed-up



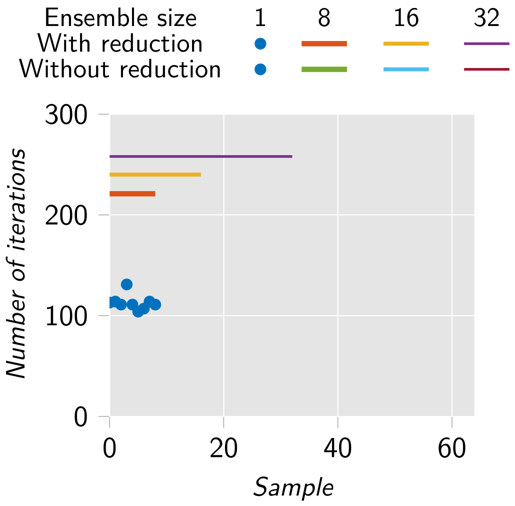


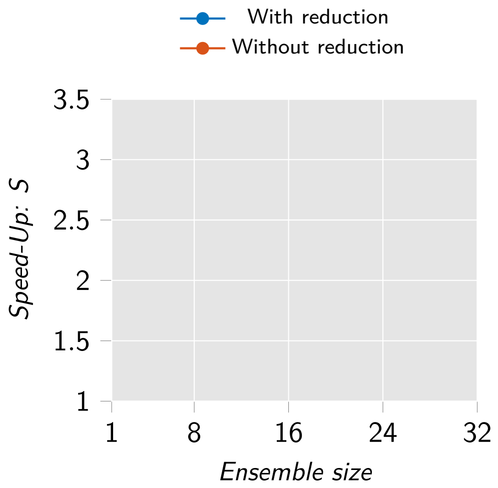
Mesh-tying example: convergence and total speed-up



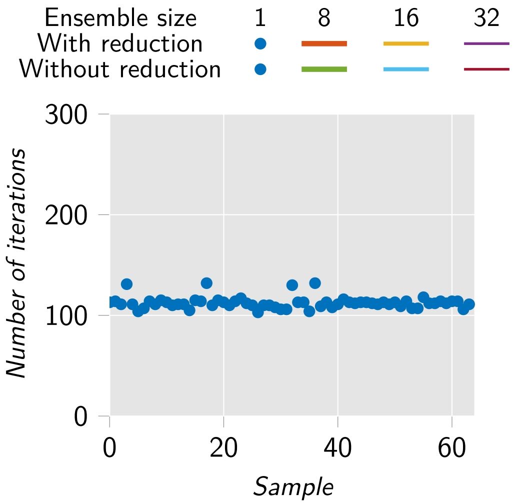


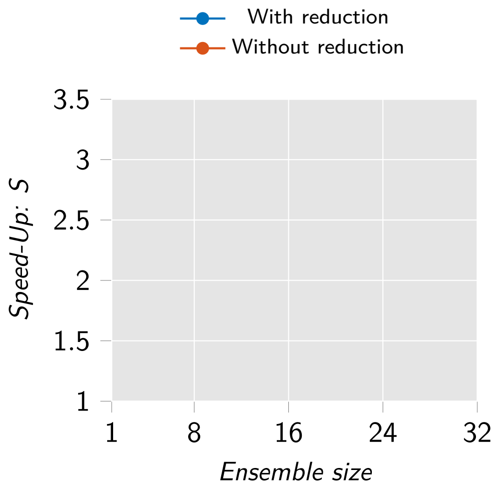
Mesh-tying example: convergence and total speed-up



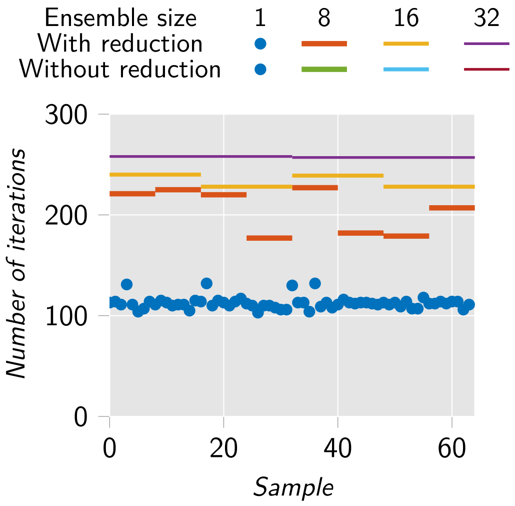


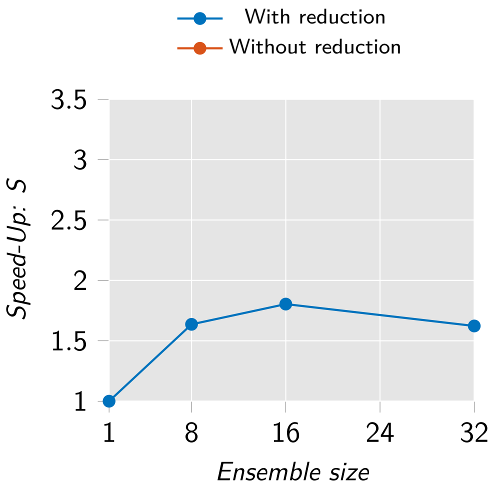
Mesh-tying example: convergence and total speed-up



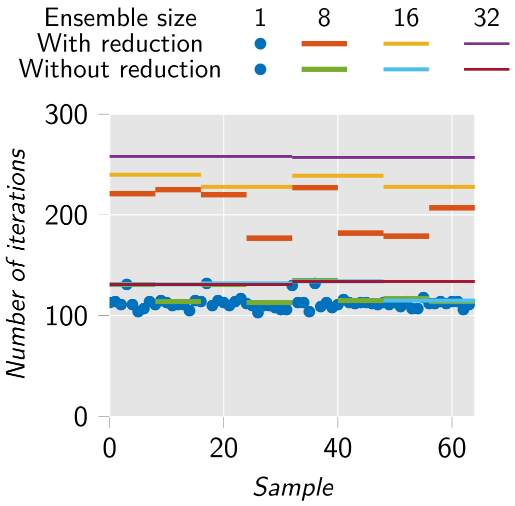


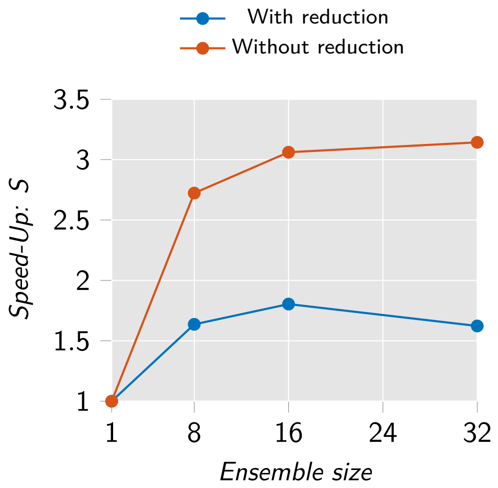
Mesh-tying example: convergence and total speed-up





Mesh-tying example: convergence and total speed-up



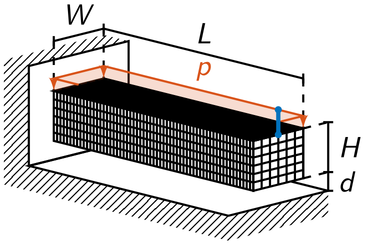


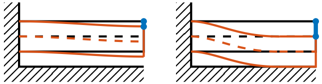
The reduced number of iterations to converge without ensemble reduction improves the speed-up compared to ensemble reduction.

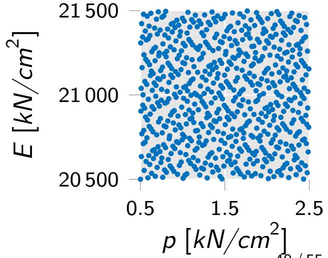
Outline

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;
- Depending on the pressure p, the contact is fully open or partially closed;
- Uncertainties:
  - Young's modulus: $E \sim \mathcal{U}(205, 215)$ [GPa];
  - Pressure: $p \sim \mathcal{U}(5, 25)$ [MPa].
- Quantity of Interest: displacement along z at point $(L, 0, H/2)$;
- 640 Halton Quasi Monte Carlo samples;
- Solved on 1 Intel Skylake CPU.







Ensemble propagation for mechanical contact problem

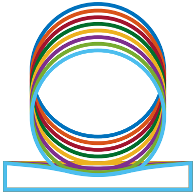
Instead of individually solving the mechanical contact problem for each instance of the model, we have to **solve simultaneously** the mechanical contact problem for **a subset of samples** of the model.

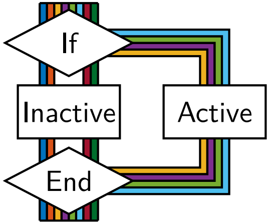
Ensemble propagation for mechanical contact problem

Instead of individually solving the mechanical contact problem for each instance of the model, we have to **solve simultaneously** the mechanical contact problem for **a subset of samples** of the model.

Challenges of the EP for mechanical contact problem:

- Different samples can have **different active Lagrange multipliers**,





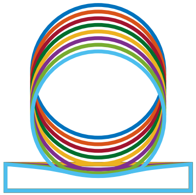
- Samples may require a **different number** of **active set iterations**.

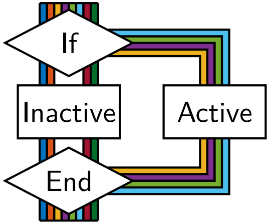
Ensemble propagation for mechanical contact problem

Instead of individually solving the mechanical contact problem for each instance of the model, we have to **solve simultaneously** the mechanical contact problem for **a subset of samples** of the model.

Challenges of the EP for mechanical contact problem:

- Different samples can have **different active Lagrange multipliers**,



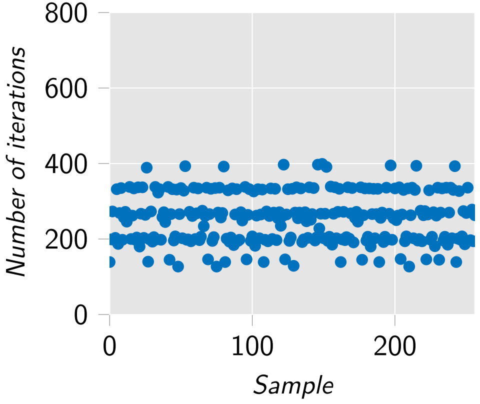


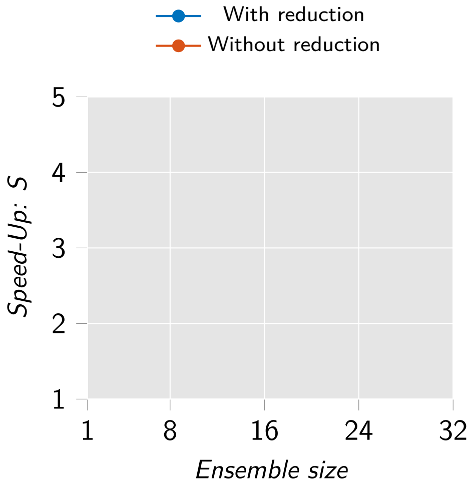
- Samples may require a **different number** of **active set iterations**.

Those challenges have been solved using the developed **Mask class** and waiting for the **convergence of all samples** in the active set strategy.

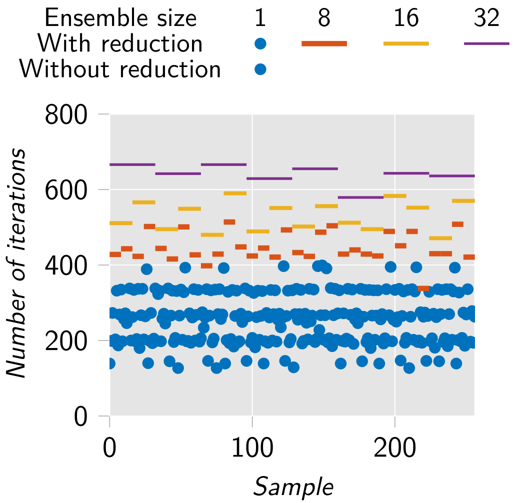
Contact example: convergence and total speed-up

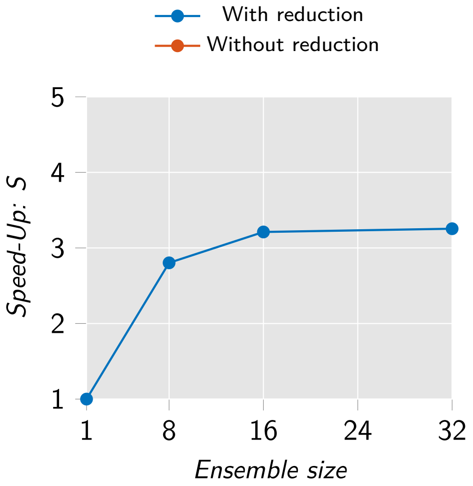
Ensemble size	1	8	16	32
With reduction	●			
Without reduction	●			



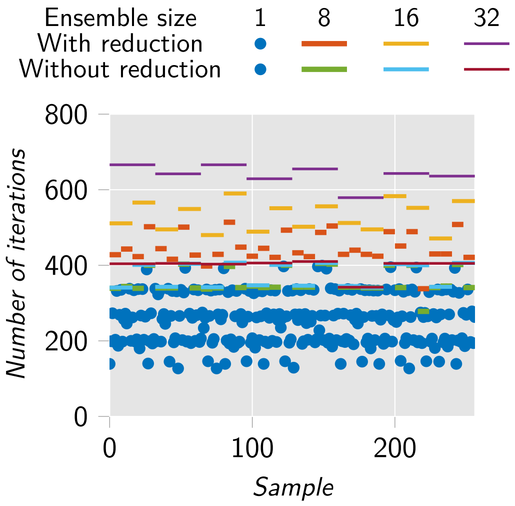


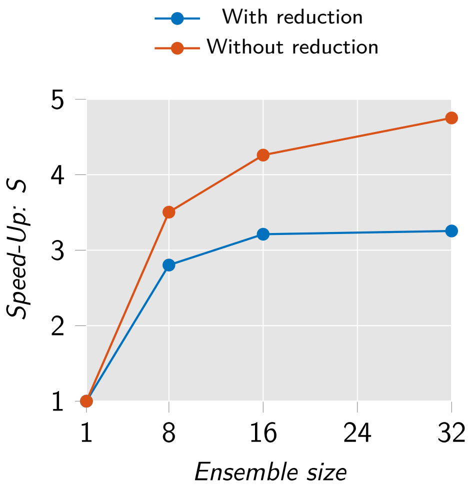
Contact example: convergence and total speed-up





Contact example: convergence and total speed-up





The reduced number of iterations to converge without ensemble reduction improves the speed-up compared to ensemble reduction.

Outline

Front mirror context

First mirrors of optical diagnostics in ITER:

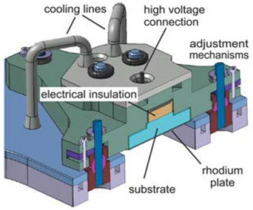
- are exposed to **high radiation** and **fluxes** of particles which escape the plasma;
- are the **most vulnerable** in-vessel optical components, being subject to erosion or to deposition of impurities.

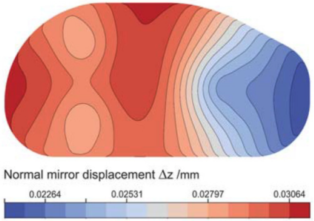
Material selected for the reflecting surface must combine:

- a **high optical reflectivity** in a wide spectral range;
- a sufficient **resistance** to physical sputtering.

Rhodium[a] is identified as a promising candidate, due to:

- low sputtering in most cases;
- high optical reflectance;
- optical reflectance insensitive to large temperature changes.

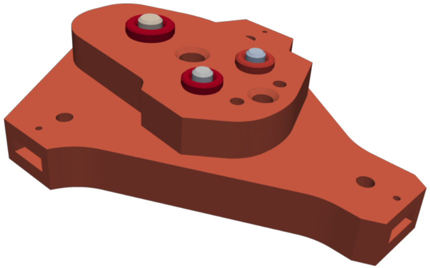




[a] P. Mertens, R. Boman, S. Dickheuer, Y. Krasikov, A. Krimmer, D. Leichtle, **K. Liegeois**, C. Linsmeier, A. Litnovsky, O. Marchuk, M. Rasinski, M. De Bock, On the use of rhodium mirrors for optical diagnostics in ITER, *Fusion Engineering and Design*, 146:2514–2518, 2019.

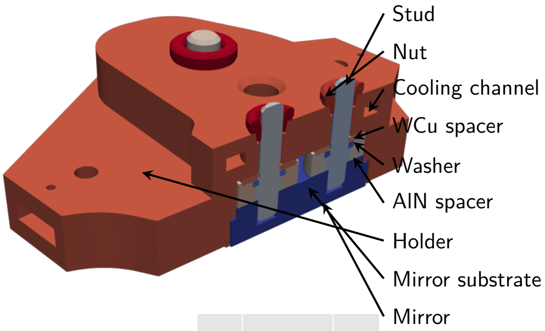
ITER test case

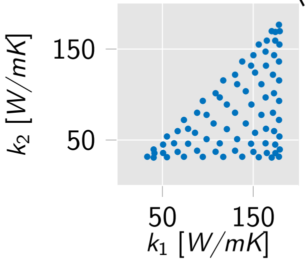
- **Thermomechanical** problem;
- The contact interfaces are **not modeled**, the mesh is fused at the common interface of the components;
- The assembly is heated by **surface and volumetric loads**;
- Rigid body motions are prevented by setting zero displacements close to the bolt holes;
- Temperature at the cooling channel is set to $70\,^{\circ}$C;
- $1.7\,10^6$ elements and $1.31\,10^6$ **degrees of freedom** per sample;
- 3-level multigrid preconditioner with block Gauss-Seidel level smoother and Klu as smoother on the coarsest level;
- Solved on **4 Intel Skylake CPUs**.



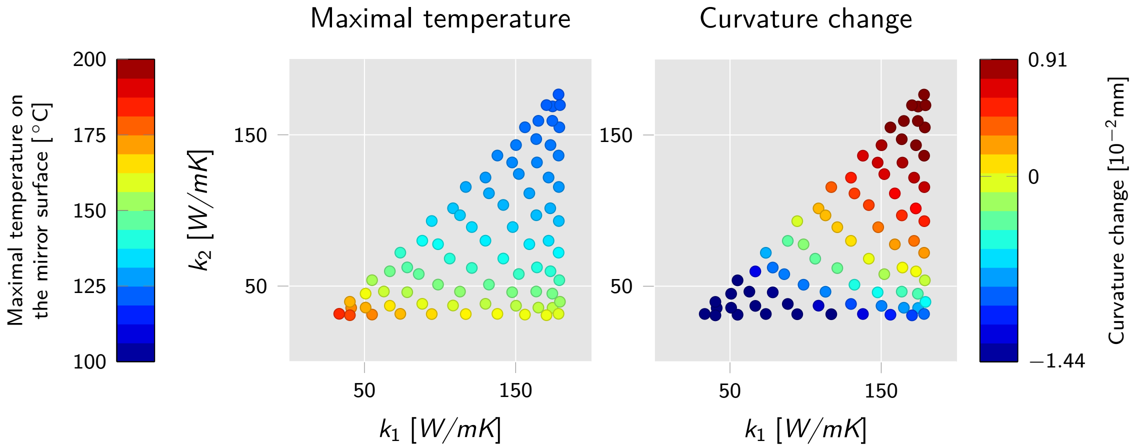
ITER test case: Uncertainties and quantities of interest

- **Uncertainties:** the irradiation due to neutron damage of the mirror modifies the material properties of the used material; we consider the **heat conductivity of the AlN (Aluminum nitride) ceramic spacers as random variables**; k_1 is the thermal conductivity of the two small AlN spacers and k_2 is the thermal conductivity of the largest AlN spacer.
- **Quantities of interest:**
  - **maximal temperature** reached on the mirror surface;
  - **deformation** of the mirror surface;
- **Uncertainty quantification strategy:** evaluate the model at some samples to build a surrogate model using the **nonintrusive spectral projection method**.



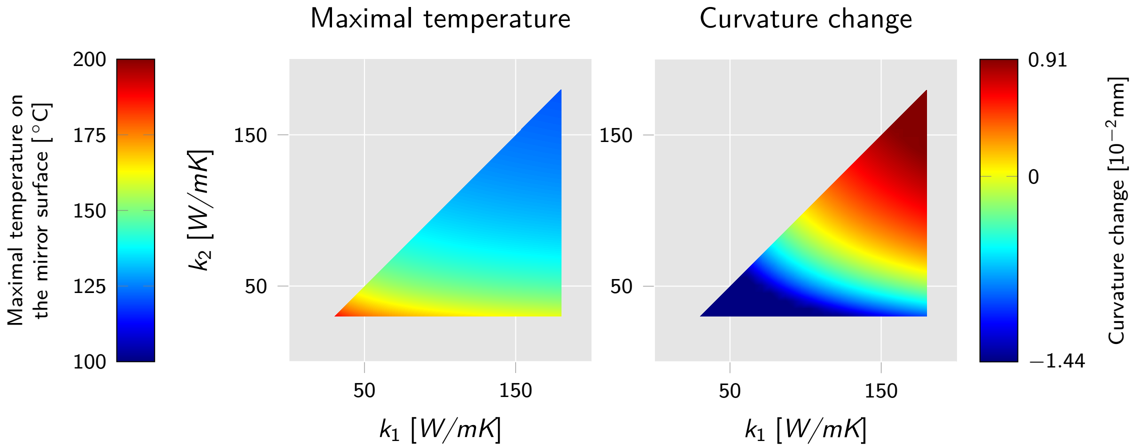


ITER test case: Quantities of Interest



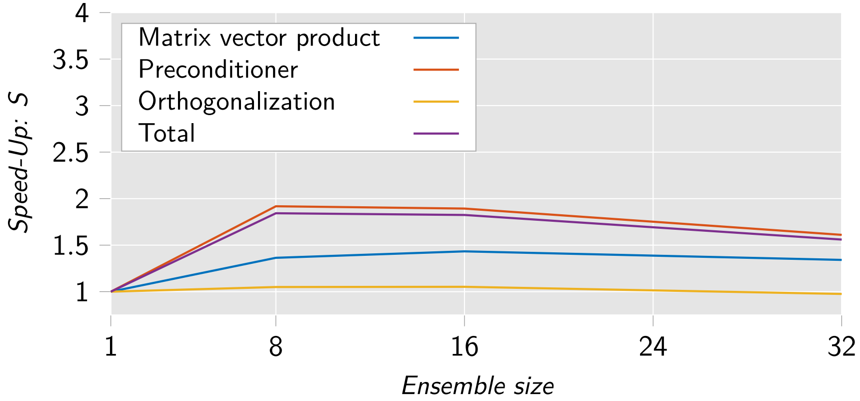
As the thermal conductivities influence the temperature distribution, they impact the deformation of the mirror surface and its curvature.

ITER test case: Surrogate models



As the thermal conductivities influence the temperature distribution, they impact the deformation of the mirror surface and its curvature.

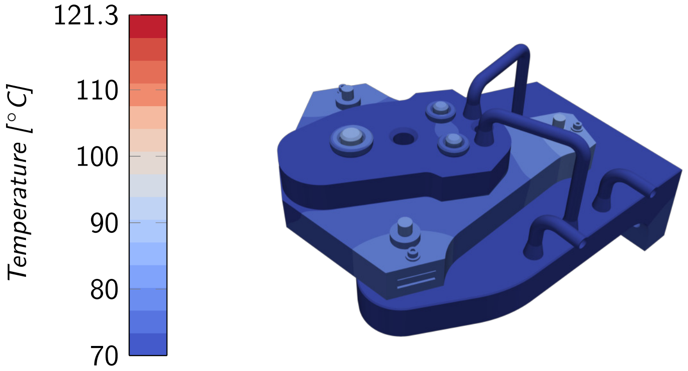
ITER test case: total speed-up of ensemble GMRES without reduction



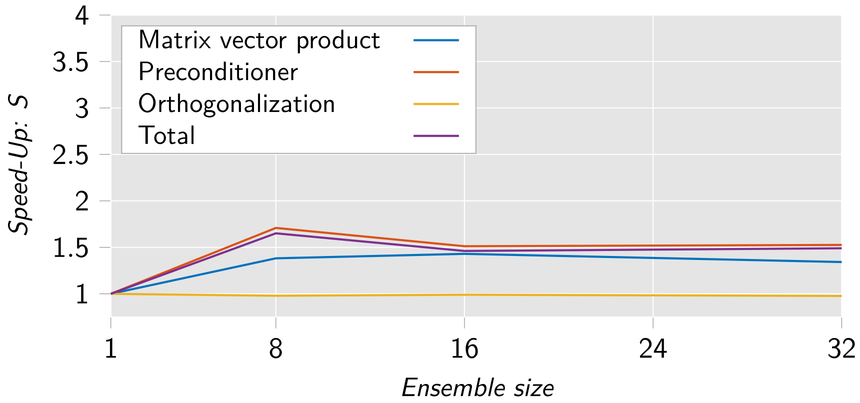
The total speed-up of ensemble GMRES is between 1.5 and 2 and is smaller than the speed-up of the sparse matrix vector product or the preconditioner due to the othogonalization process.

ITER test case: full assembly

- **Thermomechanical** problem on the full assembly;
- The contact interfaces are **not modeled**;
- $12.8\,10^6$ elements and $9.15\,10^6$ **degrees of freedom** per sample;
- 3-level multigrid preconditioner with block Gauss-Seidel level smoother and Klu as smoother on the coarsest level;
- Solved on **32 Intel Skylake CPUs**.



ITER test case: full assembly



The total speed-up of ensemble GMRES is about 1.5 and is smaller than the speed-up of the preconditioner due to the othogonalization process.

Outline

Conclusion and contributions

Extension of the embedded ensemble propagation **method from CG to GMRES**:

Conclusion and contributions

Extension of the embedded ensemble propagation **method from CG to GMRES**:

- Two variants of ensemble GMRES can currently be used: **with** ensemble reduction **and without ensemble reduction**;

Conclusion and contributions

Extension of the embedded ensemble propagation **method from CG to GMRES**:

- Two variants of ensemble GMRES can currently be used: **with** ensemble reduction **and without ensemble reduction**;
- Implementation of an **efficient ensemble GEMV**;
- **Cost per iteration** of ensemble GMRES is **independent** of coupling the samples together with ensemble reduction;

Conclusion and contributions

Extension of the embedded ensemble propagation **method from CG to GMRES**:

- Two variants of ensemble GMRES can currently be used: **with** ensemble reduction **and without ensemble reduction**;
- Implementation of an **efficient ensemble GEMV**;
- **Cost per iteration** of ensemble GMRES is **independent** of coupling the samples together with ensemble reduction;
- Ensemble GMRES without reduction is faster due to an **improved convergence** compared to ensemble GMRES with reduction;

Conclusion and contributions

Extension of the embedded ensemble propagation **method from CG to GMRES**:

- Two variants of ensemble GMRES can currently be used: **with** ensemble reduction **and without ensemble reduction**;
- Implementation of an **efficient ensemble GEMV**;
- **Cost per iteration** of ensemble GMRES is **independent** of coupling the samples together with ensemble reduction;
- Ensemble GMRES without reduction is faster due to an **improved convergence** compared to ensemble GMRES with reduction;
- Implementation of a **Mask class** for ensemble types which is now used in ensemble GMRES with ensemble reduction;

Conclusion and contributions

Extension of the embedded ensemble propagation **method from CG to GMRES**:

- Two variants of ensemble GMRES can currently be used: **with** ensemble reduction **and without ensemble reduction**;
- Implementation of an **efficient ensemble GEMV**;
- **Cost per iteration** of ensemble GMRES is **independent** of coupling the samples together with ensemble reduction;
- Ensemble GMRES without reduction is faster due to an **improved convergence** compared to ensemble GMRES with reduction;
- Implementation of a **Mask class** for ensemble types which is now used in ensemble GMRES with ensemble reduction;
- The **implementation** related to GMRES without reduction has been **merged into the official Trilinos repository on github**.

Conclusion and contributions

- Handling of **contact constraints** with embedded **ensemble propagation**;

Conclusion and contributions

- Handling of **contact constraints** with embedded **ensemble propagation**;

- An open-source software, **Katoptron**, has been developed to test ensemble GMRES;

Conclusion and contributions

- Handling of **contact constraints** with embedded **ensemble propagation**;

- An open-source software, **Katoptron**, has been developed to test ensemble GMRES;

- Application to thermomechanical problems relevant for the **front mirrors** in ITER;

Conclusion and contributions

- Handling of **contact constraints** with embedded **ensemble propagation**;
- An open-source software, **Katoptron**, has been developed to test ensemble GMRES;
- Application to thermomechanical problems relevant for the **front mirrors** in ITER;
- Observed total speed-up of **about 2** in all cases.

Directions for future work

- Investigate the impact of the **preconditioners** on the influence of the reduction on the convergence;
- Investigate and apply the strategy on the **mirror problem with contact** losses;
- Investigate the use of ensemble GMRES on other problems such as **fluid mechanics problems** or **ice sheet problems**;
- Study **grouping** strategies **for non-linear problems**;
- Consider **adaptive ensemble sizes**;
- Test the method on **transient problems**.

References

- M. D'Elia, E. T. Phipps, A. Rushdiz, and M. S. Ebeida, Surrogate-based ensemble grouping strategies for embedded sampling-based uncertainty quantification, Quantification of Uncertainty: Improving Efficiency and Technology, pp. 41-66, 2020.
- K. Goto and R. A. Geijn. Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS), 34(3):12, 2008.
- K. Liegeois, R. Boman, E.T. Phipps, T. Wiesner, and M. Arnst, GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models, Computer Methods in Applied Mechanics and Engineering, 2020.
- E.T. Phipps, M. D'Elia, H C. Edwards, M. Hoemmen, J. Hu, and S. Rajamanickam, Embedded ensemble propagation for improving performance, portability, and scalability of uncertainty quantification on emerging computational architectures. SIAM Journal on Scientific Computing, vol. 39, no 2, p. C162-C193, 2017.
- T. Wiesner, Flexible aggregation-based algebraic multigrid methods for contact and flow problems, PhD Thesis, 2015.

Acknowledgement

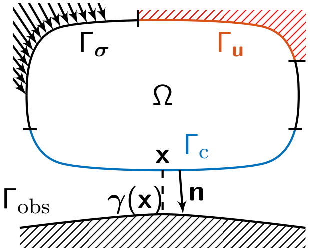
The first author, Kim Liegeois, would like to acknowledge the Belgian National Fund for Scientific Research (FNRS-FRIA), the Federation Wallonia-Brussels (FW-B), and the Faculty of Applied Sciences for their financial support.

fnrs
FREEDOM TO RESEARCH

Contributions to the Trilinos repository

- Implemented from scratch:
  - Mask class, mask assignments, logical reduction in Stokhos;
  - Ensemble GEMV;
  - Coordinate type in Teuchos;
  - Map from Lagrange multipliers to DOF, and relative checks in Moertel;
  - Interface aggregation strategy, Interface mapping transfer.
- Implemented relying on specializations:
  - Ensemble GMRES without reduction;
  - Orthogonalization manager: DGKS, ICGS, IMGS;
  - Status tests: explicit and implicit;
  - ROTG, TRSM.
- Corrections of small bugs;
- Inclusion of new tests;
- Update the Trilinos automatic tests to compile EP with and without reduction;
- Packages impacted: Stokhos, Belos, Ifpack2, KokkosKernels, MueLu, Teuchos, Moertel, Xpetra, Galeri.

Mechanical contact problem



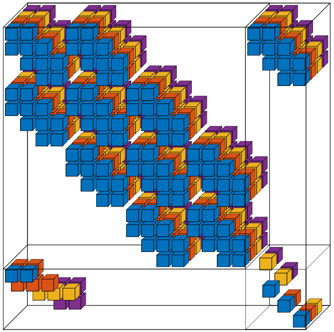
```
$k \leftarrow 0$
Choose an initial guess for the active set $\mathcal{A}_k$
do
    Given $\mathcal{A}_k$, compute the solution of
        $$\left[\begin{array}{cc|cc} \boldsymbol{K}_\mathrm{ii} & \boldsymbol{K}_\mathrm{ic} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{K}_\mathrm{ci} & \boldsymbol{K}_\mathrm{cc}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

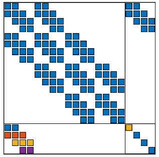
Algorithm 1: Active set strategy

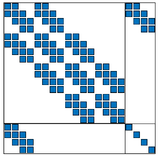
Inner nodes: i, potential contact nodes: c, at iteration k, inactive set: $\mathcal{I}_k$, and active set: $\mathcal{A}_k$.

The algebraic full form as a way to handle activities

The matrix of the system:





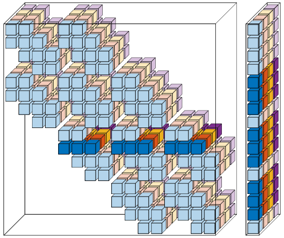


- has a **constant size** but its **graph varies** with the active set,
- can be stored using an **extended graph** which is the union of all the possible graphs,
- has a **saddle-point** structure,
- is **not positive definite** (if at least one Lagrange multiplier is active).

Ensemble propagation

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 e = entry_begin; e<entry_end; ++e) {
      const int col = A.col_entry[e];
      sum += A.values[e] * x[col];
    }
    z[row] = sum;
  }
}
```



GEMV: results - Intel(R) Xeon(R) Phi Knights Landing (KNL)

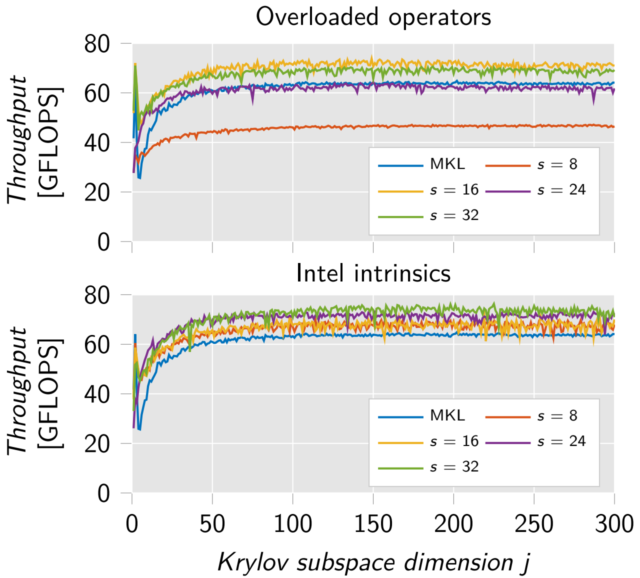
Xeon Phi KNL in quadrant cache mode
Measured bandwidth of the MCDRAM:
320 GB/s

Deduced maximal throughput:
80 GFLOPS

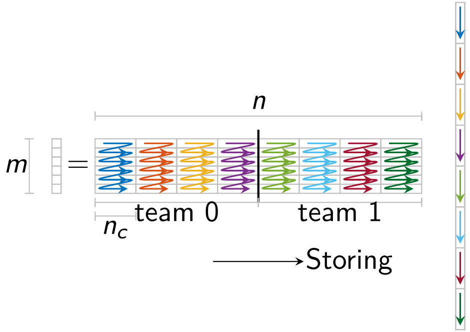
Parameters:

- Threads $N = 128$
- $n_c = 1024$ for $s = 8$, $n = 8\,N\,n_c$,
- 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.



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.