New Computational Ordering to Reach High Performance in the Time-domain BEM for the Wave Equation

Berenger Bramas [a], Olivier Coulaud[a], Guillaume Sylvand[b]

Sparse Days 2014

[a] Inria Bordeaux – Sud-Ouest, 33405 Talence, France

[b] Airbus Group Innovations, Applied Mathematics and Simulation, Toulouse, France

Introduction

Introduction

Time Domain Boundary Element Method (TD-BEM) for wave equation

- Simulates the propagation of a wave on a mesh
- In Electromagnetism/Acoustics for example
- TD-BEM less studied than frequency-domain BEM
- One time-domain computation is equivalent to many frequency-domain computations

Context

- Airbus Group and Inria collaboration
- We focus on the implementation/optimizations
- The resulting software:
  - will replace an old implementation
  - is a layer of a computational work-flow
  - uses black boxes all around
  - should run industrial simulations

Interaction Matrices, Formulation & Algorithm

Interaction/Convolution matrices

Properties of the system:

- a wave w (with a velocity c and a wavelength λ)
- a boundary Ω of N unknowns

Interaction/Convolution matrices

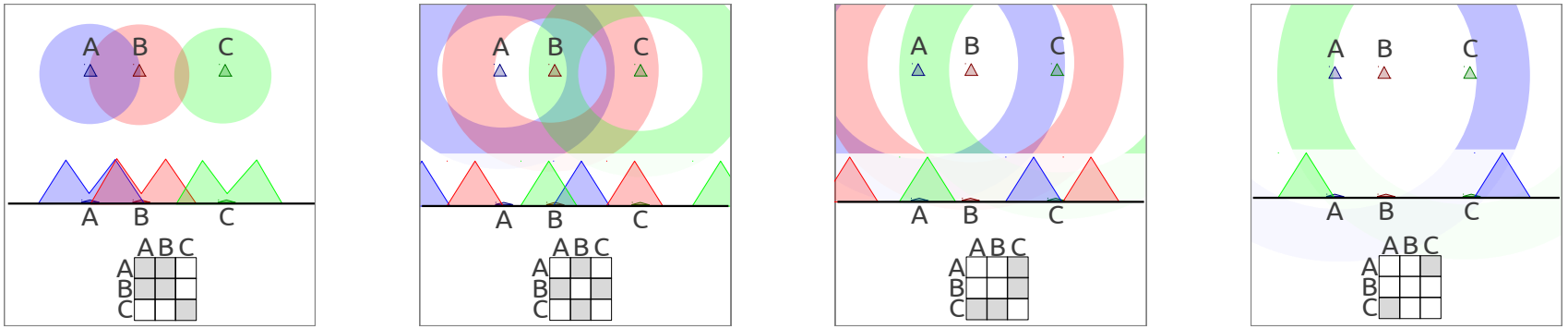
Properties of the system:

- a wave w (with a velocity c and a wavelength λ)
- a boundary Ω of N unknowns

Interaction/Convolution Matrix M^k:

- Size is $N \times N$
- Has a NNZ value at (i, j) if unknowns are far from $\approx k.c.\Delta t$
  The number of non-zero values depends on the distance between the unknowns and the physical properties
- Positive definite and sparse
- For $k > K_{max}$ the matrices M^k are null
  ($K_{max} = 2 + \ell_{max}/(c\Delta t)$, with $\ell_{max} = max_{(x,y)\in\Omega\times\Omega}(|x-y|)$ the maximum distance between two unknowns)
- Computed once if the mesh is static

Interaction/Convolution matrices - Example



- In the example: three unknowns A, B, C in $1D$
- A wave emitted from each unknown is represented at every time steps
- All matrices M^k with $k > 3$ are zero since the highest distance between elements is $\leq 3.c.\Delta t$

Formulation

Convolution system.

$$\sum_{k\geq 0}^{K^{max}} M^k \cdot a^{n-k} = l^n \tag{1}$$

- n: the time step
- M^k: the convolution matrices
- l^n: the incident wave emitted by a source on the unknowns
- a^n: the state of the system at time n

Formulation

Convolution system.

$$\sum_{k\geq 0}^{K^{max}} M^k \cdot a^{n-k} = l^n \tag{1}$$

- n: the time step
- M^k: the convolution matrices
- l^n: the incident wave emitted by a source on the unknowns
- a^n: the state of the system at time n

The objective is to compute a^n:

$$a^n = (M^0)^{-1} \left(l^n - \sum_{k=1}^{K_{max}} M^k \cdot a^{n-k} \right) \tag{2}$$

Algorithm - Formal view

a^n is calculated in two steps:

- First step: the summation stage using the past

$$s^n = \sum_{k=1}^{K_{max}} M^k \cdot a^{n-k} \tag{3}$$

$$\widetilde{s}^n = l^n - s^n \tag{4}$$

- Second step: the factorization

$$M^0 a^n = \widetilde{s}^n \tag{5}$$

Algorithm - Formal view

a^n is calculated in two steps:

- First step: the summation stage using the past

$$s^n = \sum_{k=1}^{K_{max}} M^k \cdot a^{n-k} \tag{3}$$

$$\widetilde{s}^n = l^n - s^n \tag{4}$$

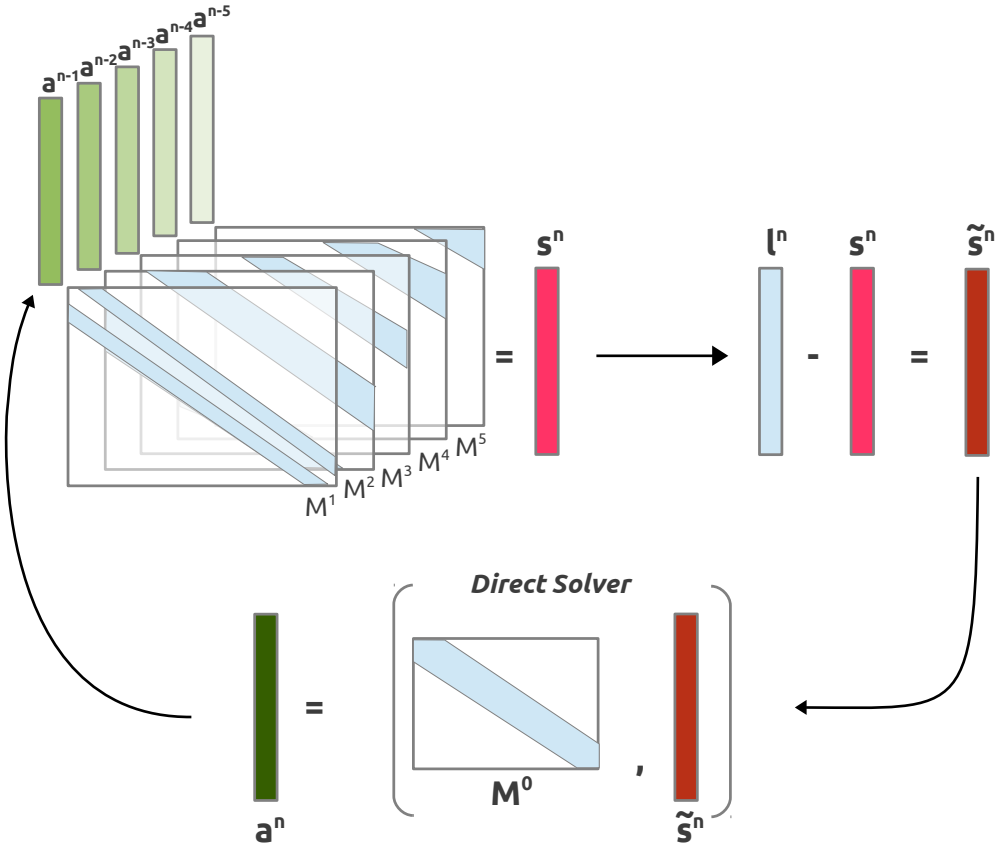
- Second step: the factorization

$$M^0 a^n = \widetilde{s}^n \tag{5}$$

The summation is the most expensive part!

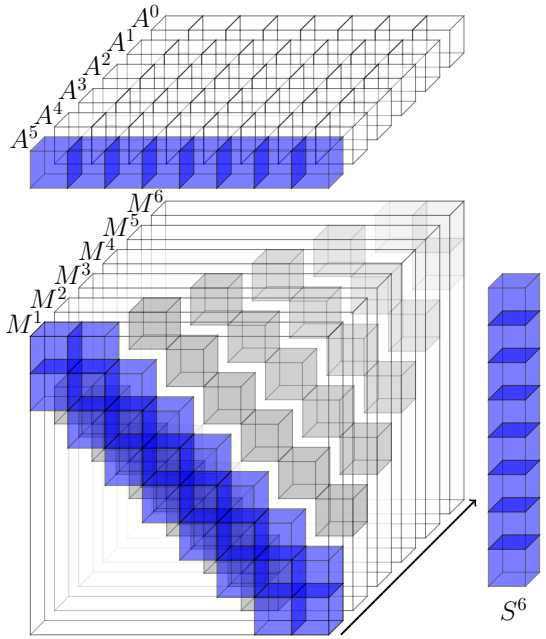
Algorithm - Schematic view

$$a^n = (M^0)^{-1} \left(l^n - \sum_{k=1}^{K_{max}} M^k \cdot a^{n-k} \right) \tag{6}$$



Computational order of the summation

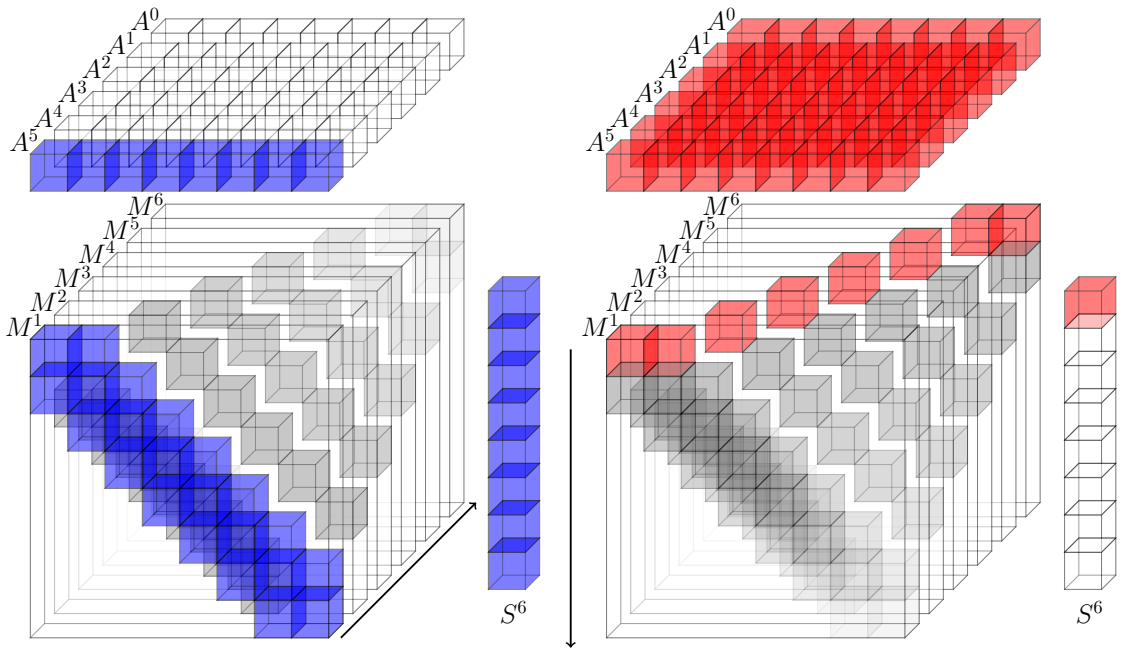
Possible order of computation



Front (k)/ SpMV

$$1 \leq i \leq N, s^n(i) \quad = \quad \sum_{k=1}^{k_{max}} \sum_{j=1}^{N} M^k(i,j) \times a^{n-k}(j) \qquad (7)$$

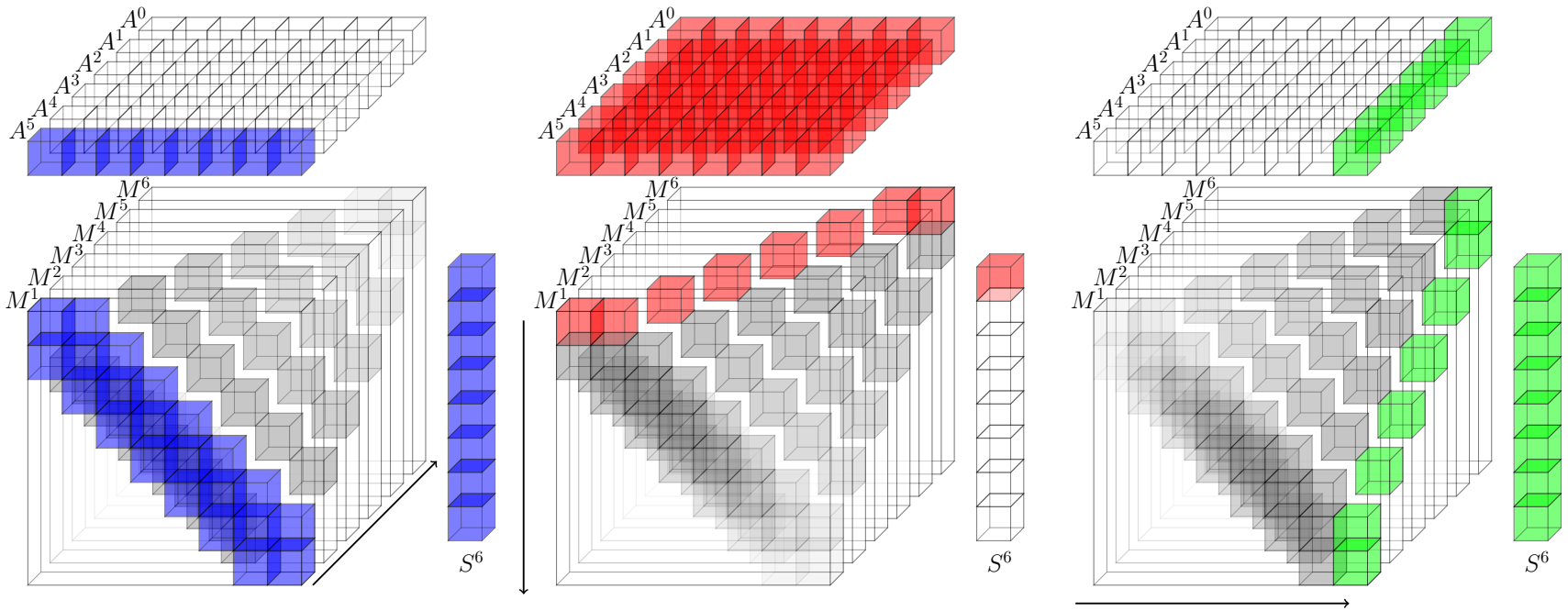
Possible order of computation



Front (k)/ SpMV Top (i)

$$1 \leq i \leq N, s^n(i) = \sum_{k=1}^{k_{max}} \sum_{j=1}^{N} M^k(i,j) \times a^{n-k}(j) \qquad (7)$$

Possible order of computation



Front (k)/ SpMV Top (i) Slice (j)

$$1 \leq i \leq N, s^n(i) = \sum_{k=1}^{k_{max}} \sum_{j=1}^{N} M^k(i,j) \times a^{n-k}(j) \qquad (7)$$

Structure of a slice

A $Slice^j$:

- When outer loop index is j
- The concatenation of column j of the interaction matrices M^k (except M^0)
- Size $(N \times (K_{max} - 1))$
- There is one vector per line
- $Slice^j(i,k) = M^k(i,j) \neq 0$
  with $k_s = d(i,j)/(c\Delta t)$ and $k_s \leq k \leq k_s + p$

Structure of a slice

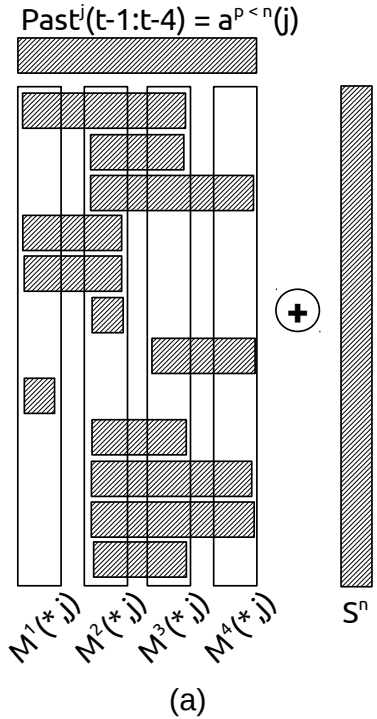
A $Slice^j$:

- When outer loop index is j
- The concatenation of column j of the interaction matrices M^k (except M^0)
- Size $(N \times (K_{max} - 1))$
- There is one vector per line
- $Slice^j(i,k) = M^k(i,j) \neq 0$
  with $k_s = d(i,j)/(c\Delta t)$ and $k_s \leq k \leq k_s + p$

One vector per row!

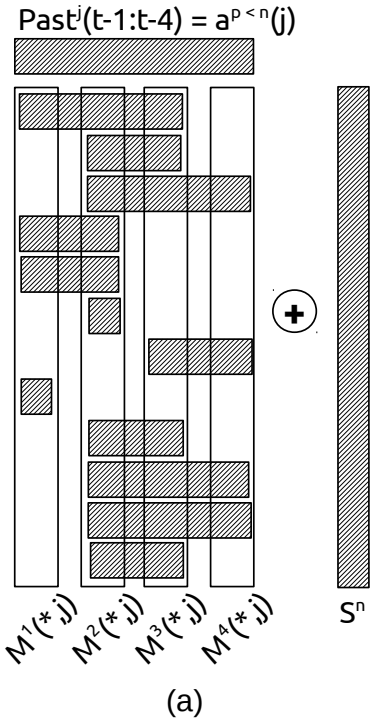
Slice Computation

Computation with slices



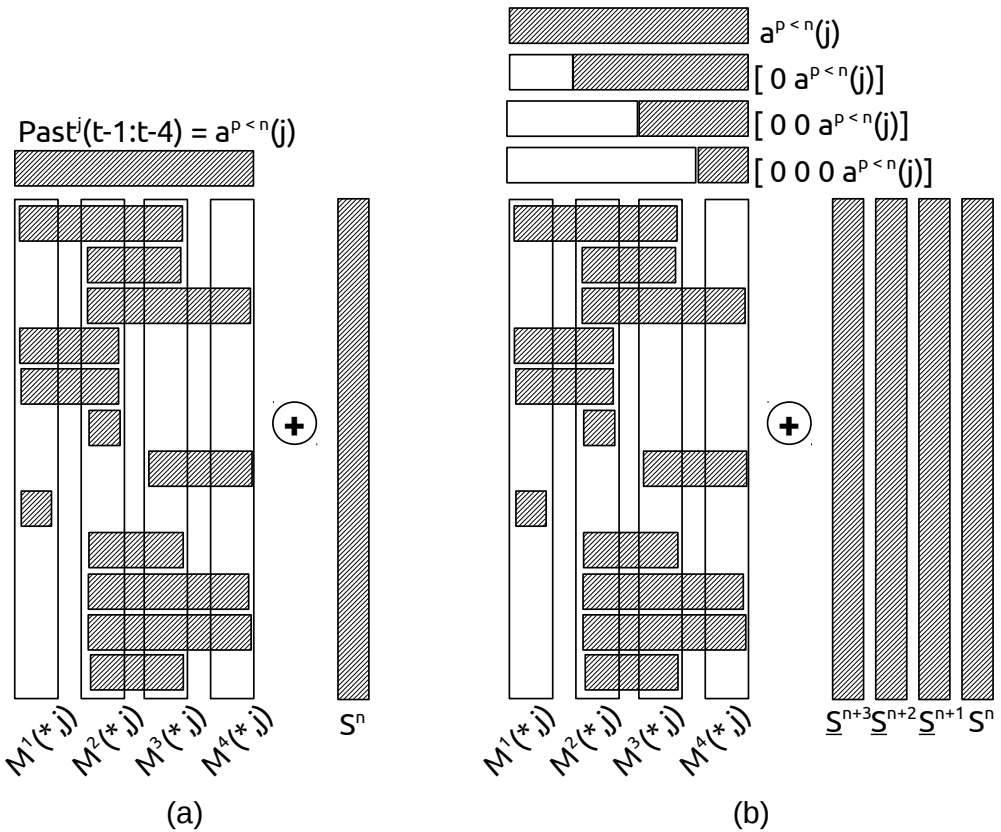
(a)

Several Summations Per Iteration

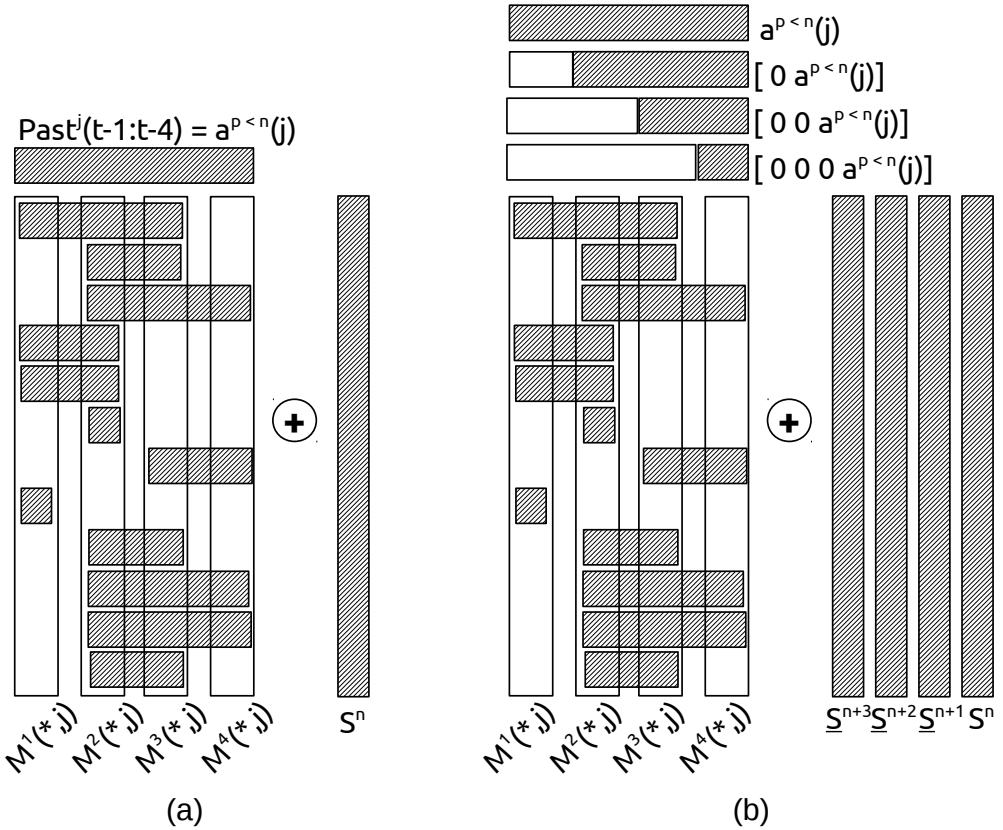


(a)

Several Summations Per Iteration

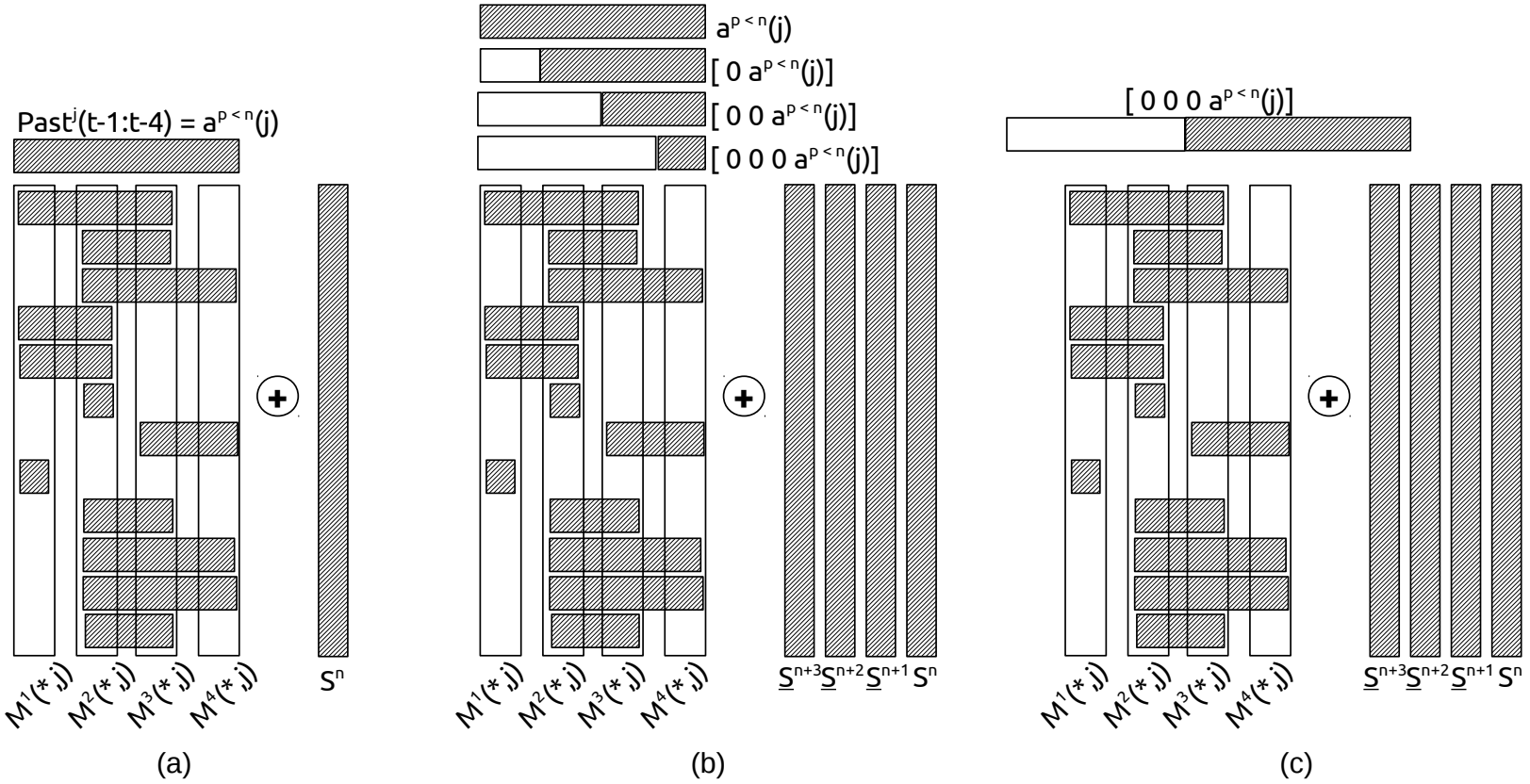


Several Summations Per Iteration



When grouping we need a radiation stage (because the future summation $s^{n<m}$ vectors depends on not yet computed $a^{n\leq p'<m}$)

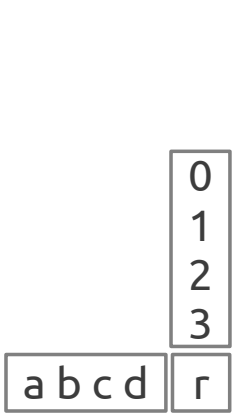
Several Summations Per Iteration



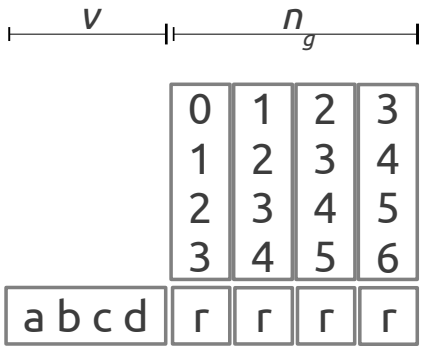
When grouping we need a radiation stage (because the future summation $s^{n<m}$ vectors depends on not yet computed $a^{n\leq p'<m}$)

Multi-vectors/vector product

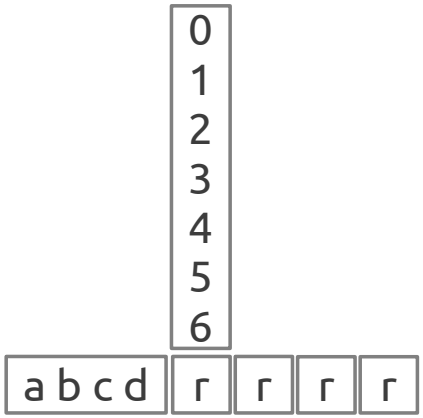
With vector length $v = 4$ and group size $n_g = 4$:



Vector/vector product



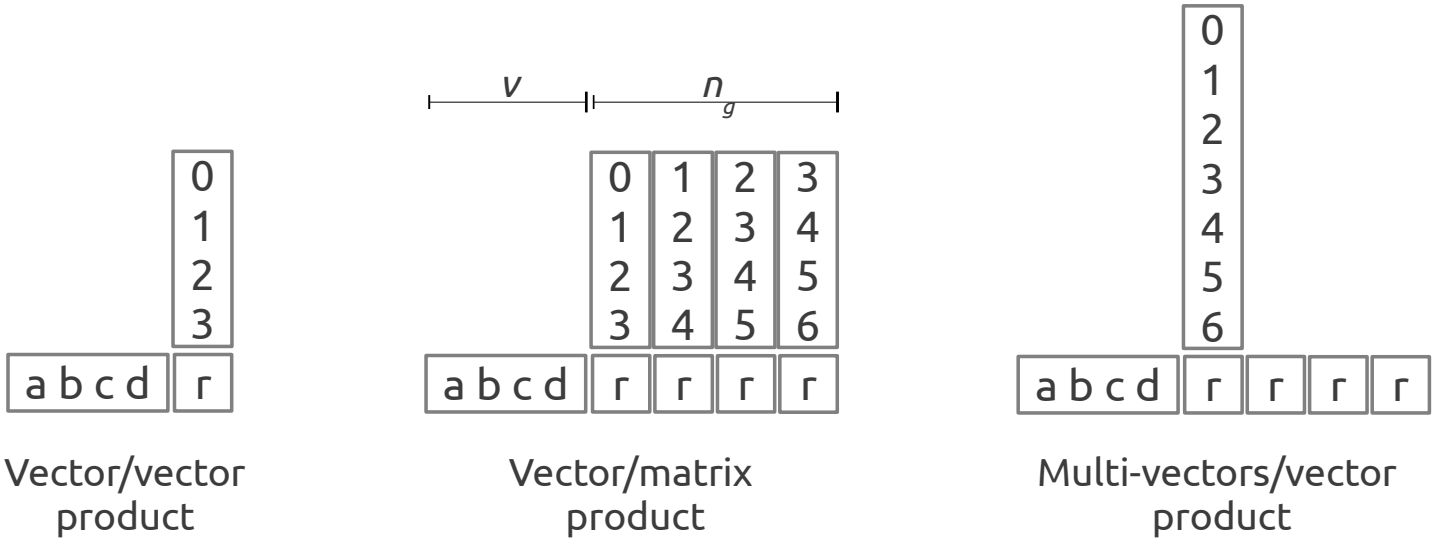
Vector/matrix product



Multi-vectors/vector product

Multi-vectors/vector product

With vector length $v = 4$ and group size $n_g = 4$:



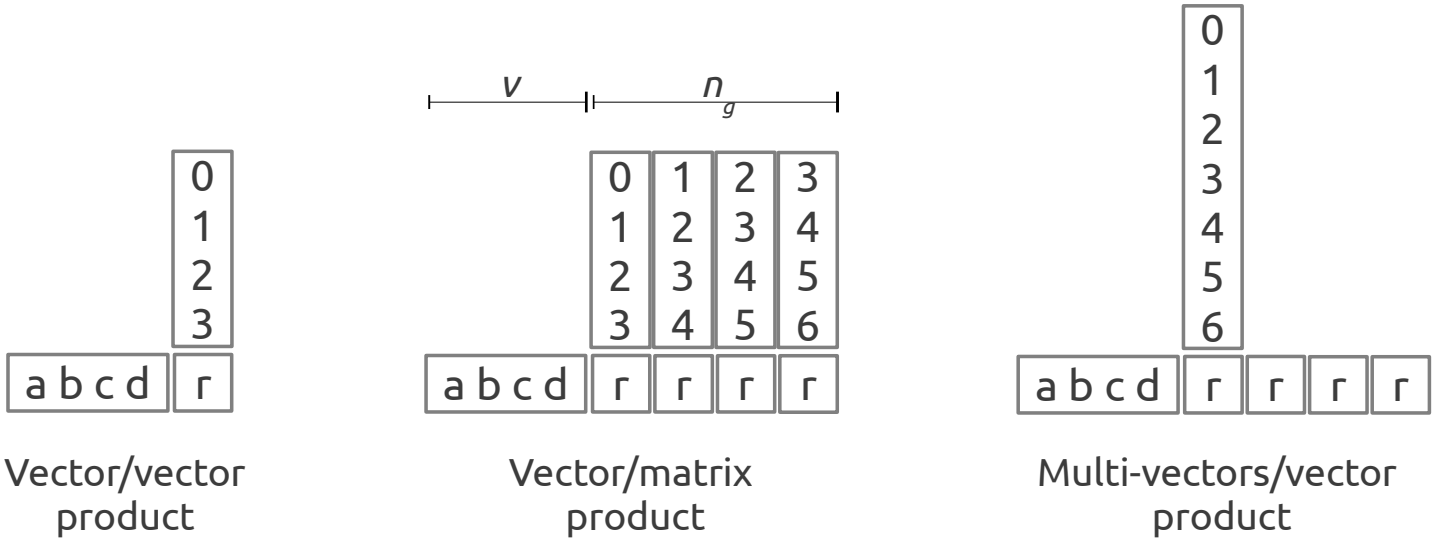
Vector/vector product | Vector/matrix product | Multi-vectors/vector product

To perform $v \times n_g \times 2$ Flop:

- Vectors product loads
  - $2v + 1$ per vector product, total : $n_g(2v + 1)$

Multi-vectors/vector product

With vector length $v = 4$ and group size $n_g = 4$:



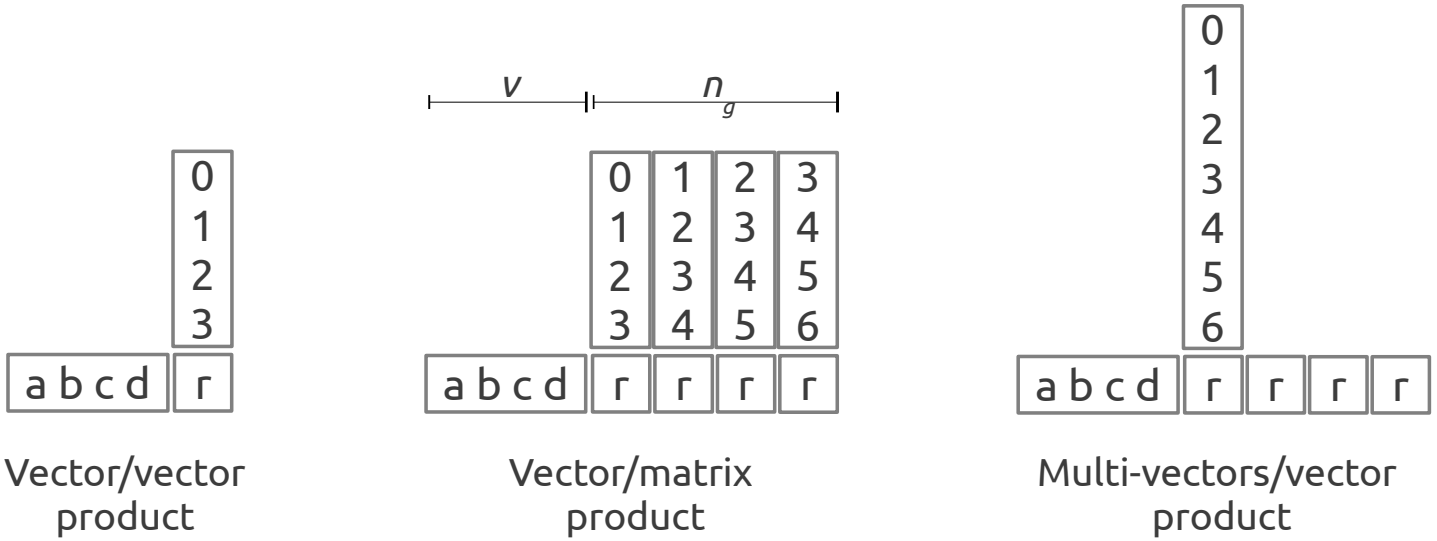
Vector/vector product | Vector/matrix product | Multi-vectors/vector product

To perform $v \times n_g \times 2$ Flop:

- Vectors product loads
  - $2v + 1$ per vector product, total : $n_g(2v + 1)$
- Vector/matrix product loads
  - $v + n_g(v + 1)$

Multi-vectors/vector product

With vector length $v = 4$ and group size $n_g = 4$:



Vector/vector product | Vector/matrix product | Multi-vectors/vector product

To perform $v \times n_g \times 2$ Flop:

- Vectors product loads
  - $2v + 1$ per vector product, total : $n_g(2v + 1)$
- Vector/matrix product loads
  - $v + n_g(v + 1)$
- Multi-vectors/vector product loads
  - $(v + n_g - 1) + (v) + (n_g)$

Multi-vectors/vector product Algorithm

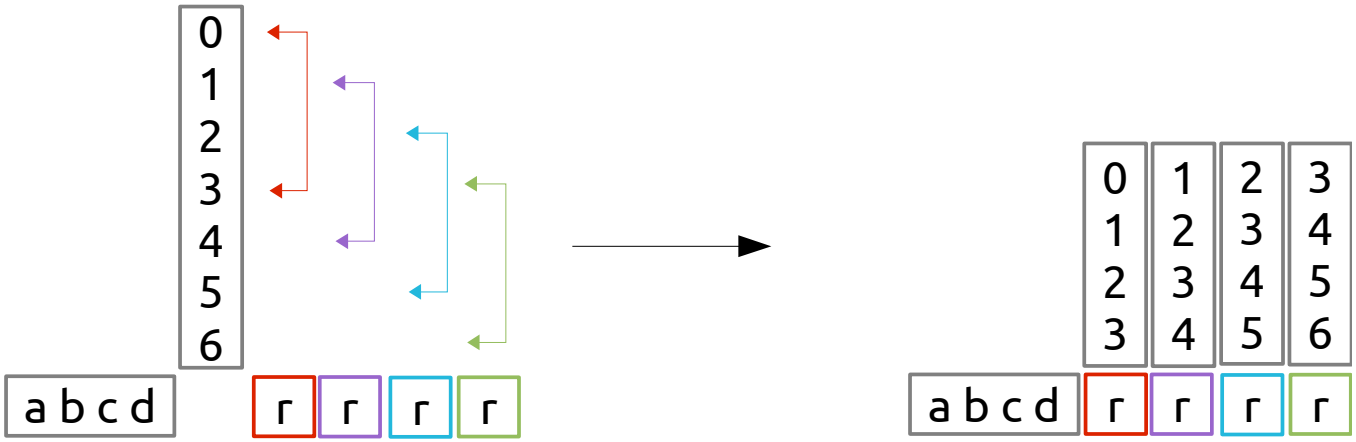
Example using Axpy:

---

for k from 0 to n_g-1 do

$$res(i,k) = v * past(starts(i) + k)$$

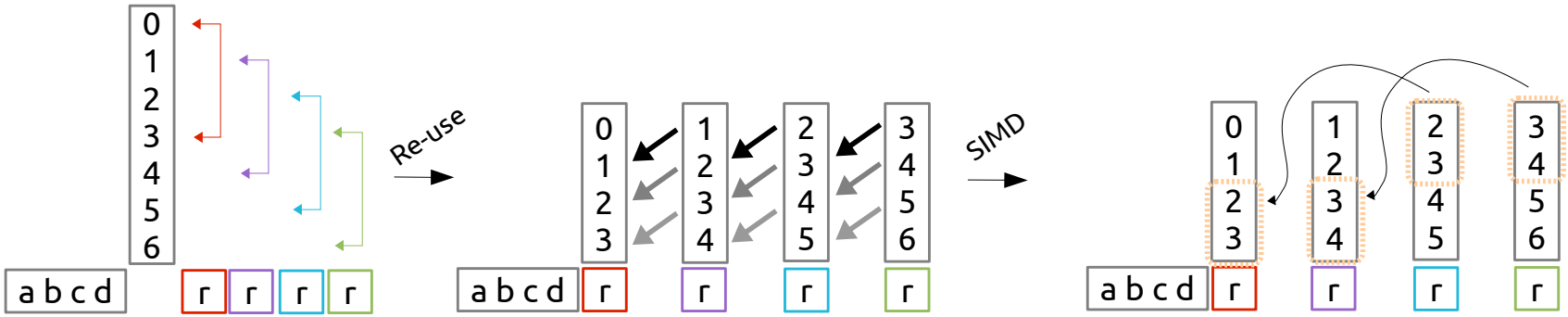
endfor



Multi-vectors/vector product Algorithm

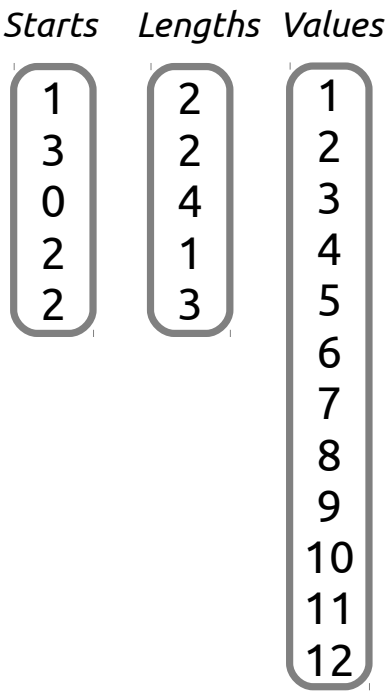
Optimizations:

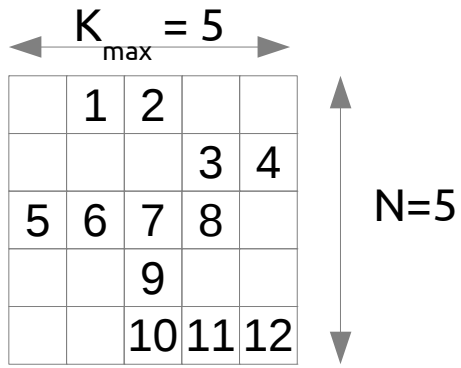
- Re-use the past values
- Minimize the load of the slices values
- Use SIMD (SSE, AVX) by hand



Slice data structure

- $starts$: a vector of size N to store the starting column of each slice-vector
- $lengths$: a vector of size N to store the length of each slice-vector
- $values$: A block of values to store the slice-vector values in row major





Multi-vectors/vector product

The Multi-vectors/vector product operator:

- Has a good ratio of data loaded against flop
- Is implemented and optimized (C, SSE, AVX, Assembly, etc.)
- But needs a radiation stage if $n_g > 1$ because we pad the past vector with 0

Results & Performance study

Configuration

Hardware:

- SSE-Host
  - 2 Quad-core Nehalem Intel Xeon X5550 $(2,66GHz)$
  - $24GB$ (DDR3) of shared memory
- AVX-Host
  - 2 Deca-core Ivy-Bridge Intel Xeon E5-2670 v2 $(2,50GHz)$
  - $128GB$ (DDR3) of shared memory

Software:

- Gcc 4.7.2 compiler + Aggressive compilation flags
- In double + single precision floating point numbers
- Open-MPI 1.6.5 + State of the art direct solver Mumps 4.10.0

Performance evaluation of the multi-vectors/vector

SSE-Host, in Double precision, with $n_g = 8$, slices have dimensions $N_r \times v$.

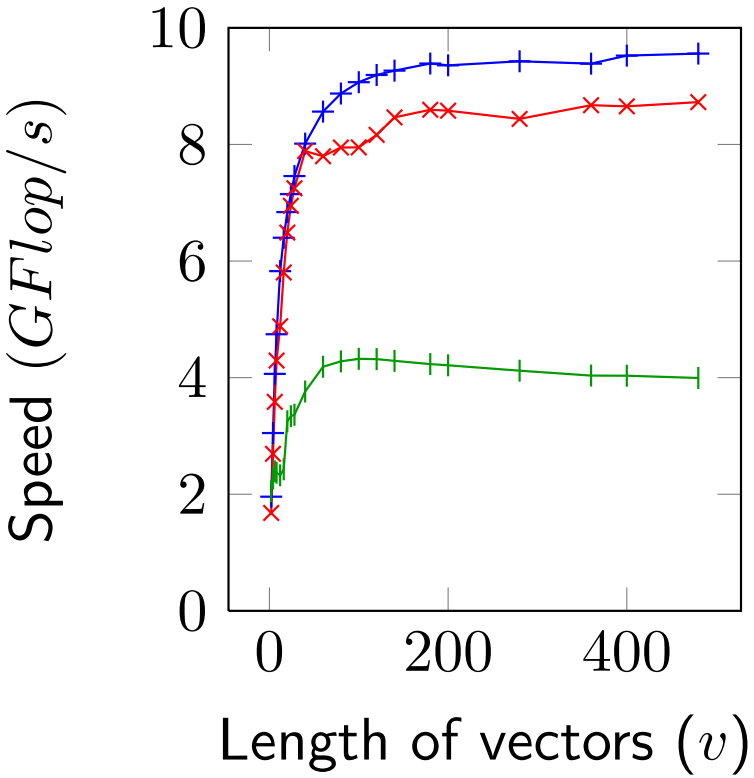


Figure: $N_r = 1\,000$

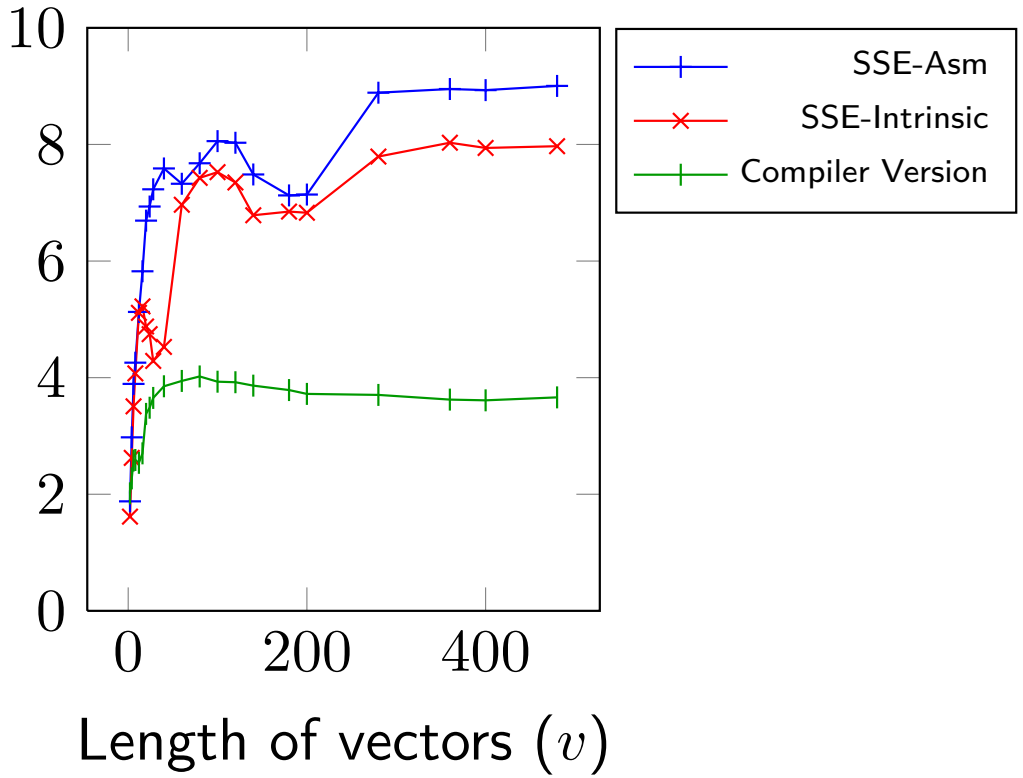


Figure: $N_r = 20\,000$

Performance evaluation of the multi-vectors/vector

AVX-Host, with $N_r = 1\,000$, $n_g = 8$, slices have dimensions $N_r \times v$.

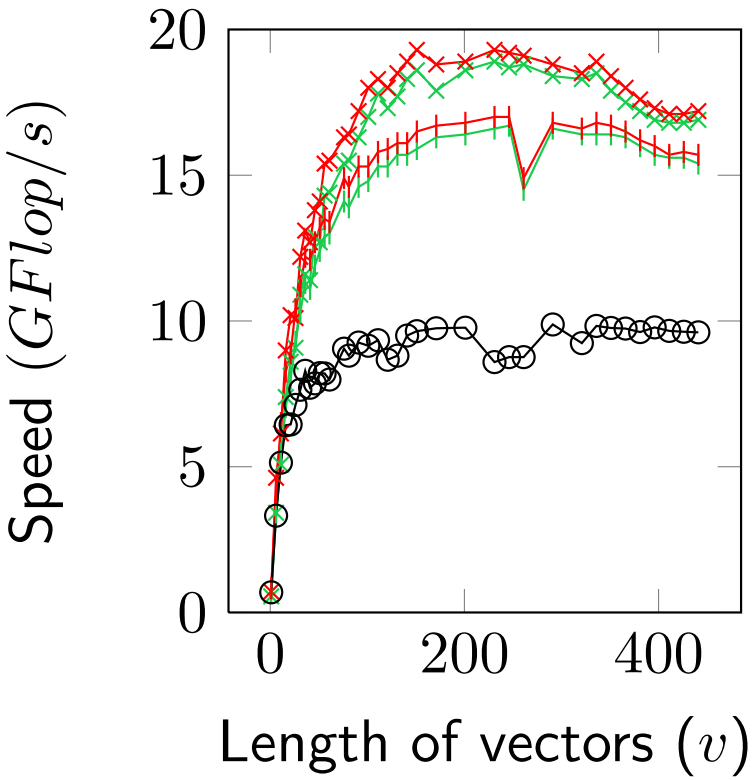


Figure: Single Precision

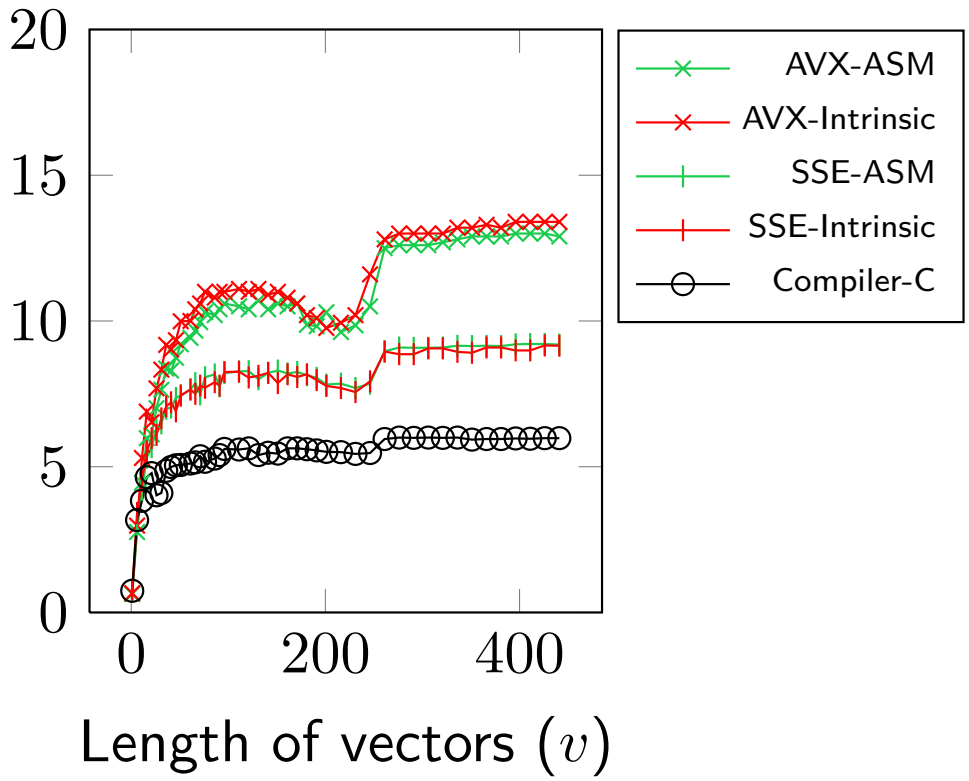


Figure: Double Precision

Performance evaluation of the multi-vectors/vector

AVX-Host, with $N_r = 20\,000$, $n_g = 8$, slices have dimensions $N_r \times v$.

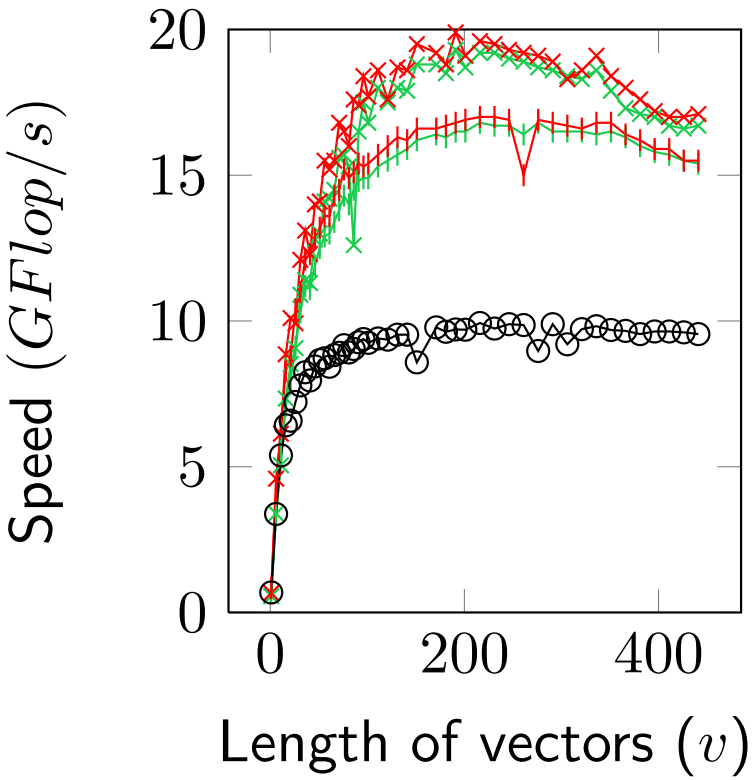


Figure: Single Precision

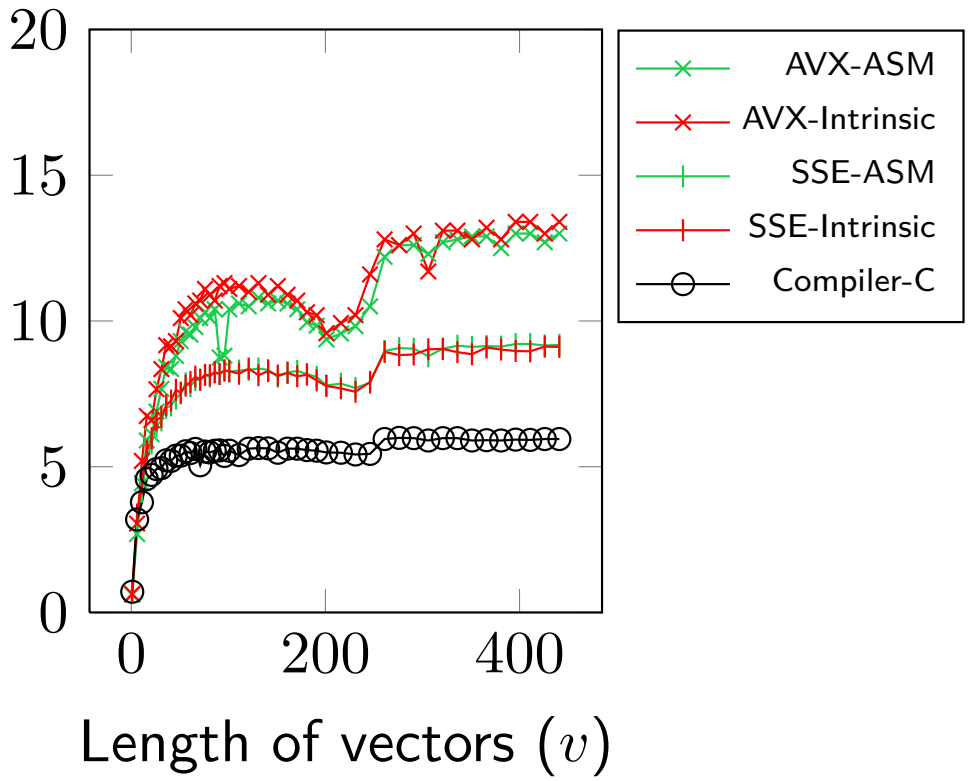


Figure: Double Precision

Test case

We use an airplane test case:

- Composed of $23\,962$ unknowns
- With $10\,823$ time iterations
- There are 341 interaction matrices M^k ($\approx 5.5 \times 10^9$ NNZ)
- Computing one summation s^n requires $11\,GFlop$
- The total simulation costs $130\,651\,GFlop$
- In Double

Parallel Efficiency - Definition

All the simulation data takes $70\,GB$, in order to stay in-core we use at least 4 nodes.

We use a modified version of the parallel efficiency:

$$\widetilde{e}_n = \frac{(r * T_r)}{(p * T_p)} \tag{8}$$

- T^p: the time taken by p Cores
- r: the minimum number of Cores (used as reference)

(original formula is $e_n = T_1/(T_p * p)$)

Parallel Efficiency

In the airplane test case

- v is between 1 and 15 (average is 9.5)
- SSE-Asm $\rightarrow 3.9\,GFlop/s$ / Compiler Version $\rightarrow 1.7\,GFlop/s$

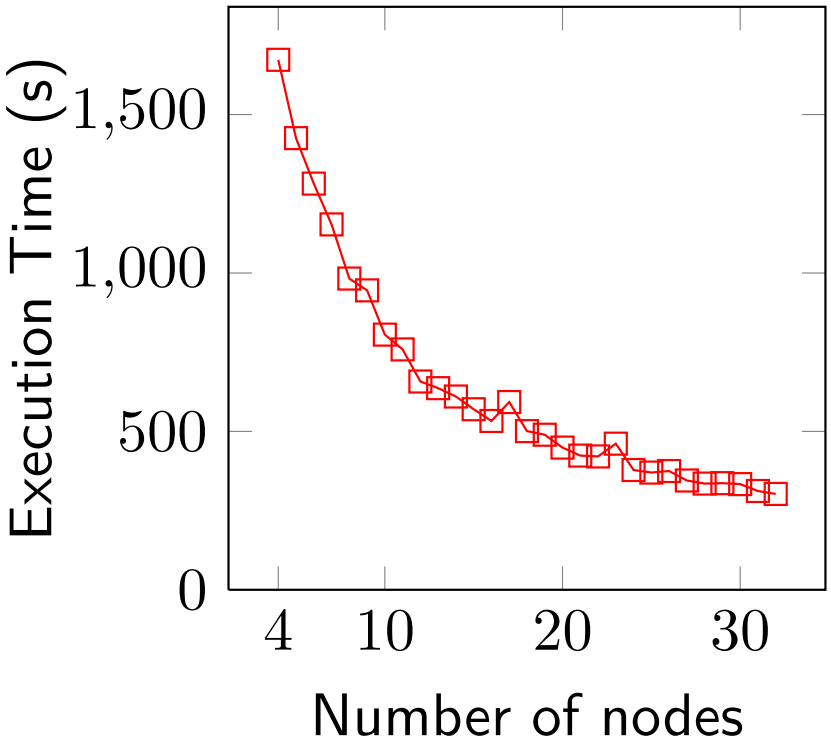


Figure: Execution time

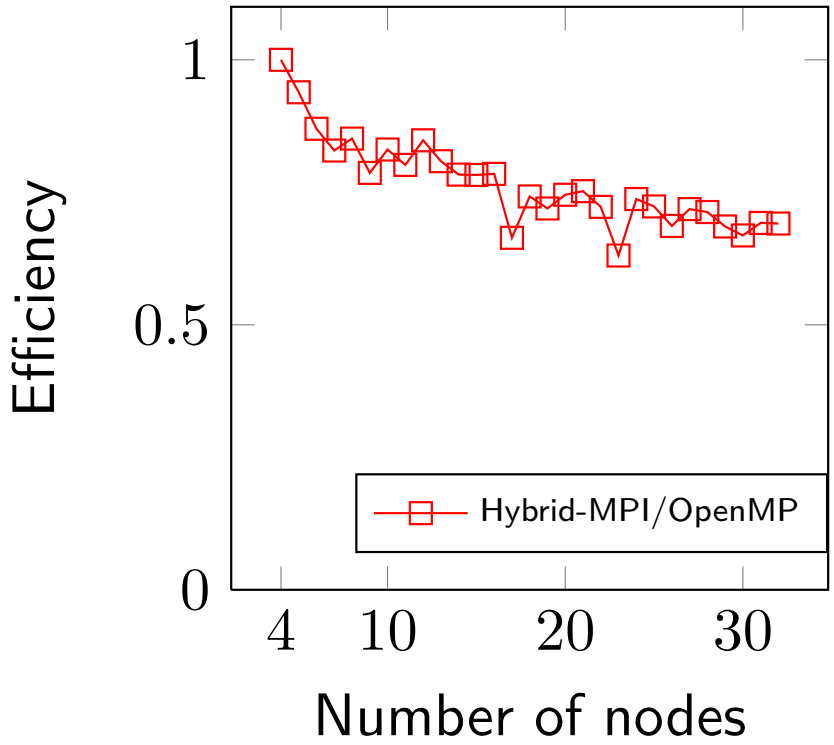


Figure: Parallel efficiency $\widetilde{e}_n$

(SSE-Host, using 4 to 32 nodes, 8 CPU per node and $n_g = 8$)

Conclusion

- High flop-rate
- Good efficiency
- Nice improvement against the previous implementation

Conclusion

- High flop-rate
- Good efficiency
- Nice improvement against the previous implementation

Perspective:

- Run big simulations

Acknowledgement

- Experiments using the PLAFRIM experimental test bed.
- This work is supported by the Airbus Group Defense & Space, Airbus Group Innovations, Inria and Conseil Régional d'Aquitaine initiative.

Thanks - Questions?