Optimization and Parallelization of the Boundary Element Method for the Wave Equation in Time Domain

Bérenger Bramas

Inria, Bordeaux - Sud-Ouest

PhD defense - Feb. 15th 2016

Advisor: Oliver Coulaud (Inria)

Industrial co-advisor: Guillaume Sylvand (Airbus)

Context

Wave Equation Problems

- Study the wave propagation in acoustics or electromagnetism
- Critical in several industrial fields (design, robustness study)

In our case: **antenna placement**, electromagnetic compatibility, furtivity, lightning, ...

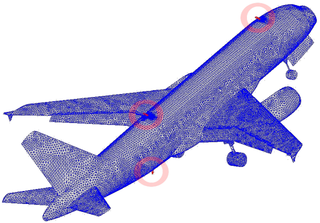


Image from Airbus Group.

Wave Equation Simulations

Boundary element method (BEM): integral equation over a discretized mesh

Interest of BEM compared to other approaches

- Better accuracy
- Surfacic mesh (easier to produce)

Disadvantages of BEM

- Dense matrices (specific solvers)

BEM

Boundary Element Method (BEM) for the Wave Equation

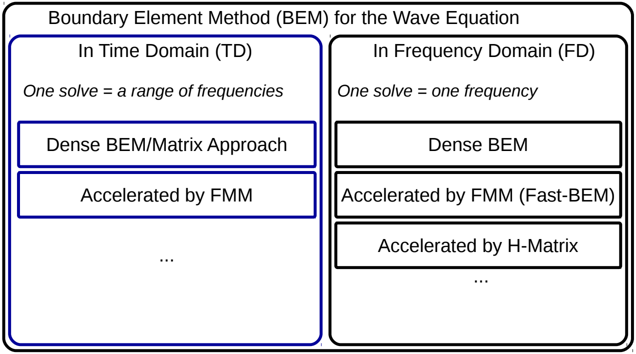
In Time Domain (TD)

One solve = a range of frequencies (Fourier Transform)

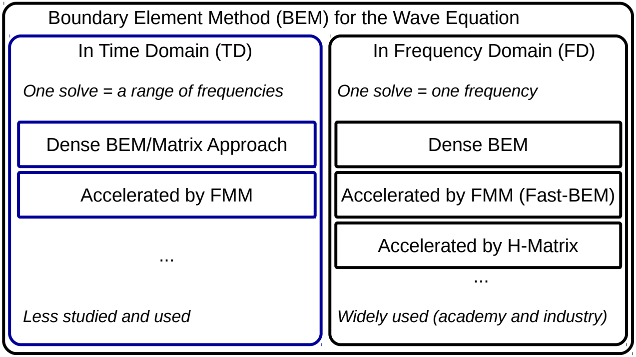
In Frequency Domain (FD)

One solve = one frequency

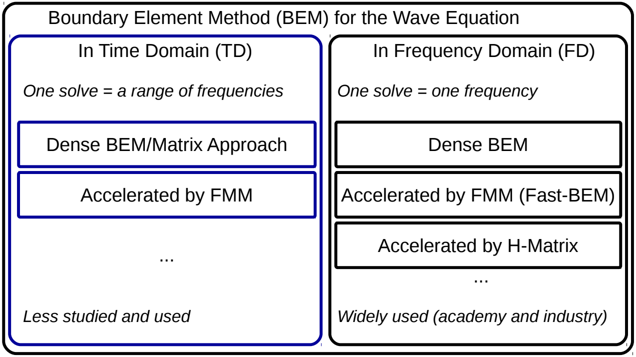
BEM


Boundary Element Method (BEM) for the Wave Equation
In Time Domain (TD)
One solve = a range of frequencies
Dense BEM/Matrix Approach
Accelerated by FMM
...
In Frequency Domain (FD)
One solve = one frequency
Dense BEM
Accelerated by FMM (Fast-BEM)
Accelerated by H-Matrix
...

BEM


Boundary Element Method (BEM) for the Wave Equation
In Time Domain (TD)
One solve = a range of frequencies
Dense BEM/Matrix Approach
Accelerated by FMM
...
Less studied and used
In Frequency Domain (FD)
One solve = one frequency
Dense BEM
Accelerated by FMM (Fast-BEM)
Accelerated by H-Matrix
...
Widely used (academy and industry)

BEM



Advantages/disadvantages depend on the application/configuration

Industrial Context

- In partnership with Airbus Group Innovation (financed jointly with Region Aquitaine)
- Airbus solvers:
  - FD-BEM
    - Accelerated by FMM or H-Matrix techniques
  - TD-BEM (experimental)
    - No stability problem (formulation based on a full Galerkin discretization unconditionnaly stable from [Terrasse, 1993])
    - With FMM [Ergin et al., 2000] (trial)

Industrial Context

- In partnership with Airbus Group Innovation (financed jointly with Region Aquitaine)
- Airbus solvers:
  - FD-BEM
    - Accelerated by FMM or H-Matrix techniques
  - TD-BEM (experimental)
    - No stability problem (formulation based on a full Galerkin discretization unconditionnaly stable from [Terrasse, 1993])
    - With FMM [Ergin et al., 2000] (trial)

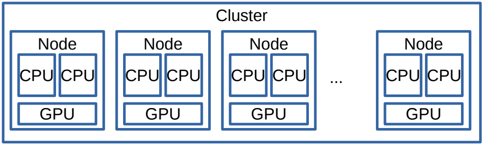
Objective:

- Reduce the performance gap between FD and TD approaches

HPC

Super-computers are mandatory to solve large problems

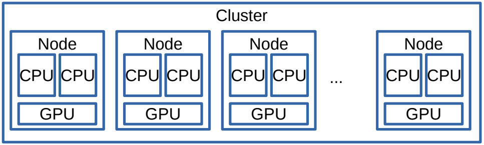
- Shared/Distributed memory
- Heterogeneous (one or more GPU per node)



HPC

Super-computers are mandatory to solve large problems

- Shared/Distributed memory
- Heterogeneous (one or more GPU per node)



Some of the challenges

- Efficient computational algorithm/kernel
- Parallelization
- Balancing
- Hardware abstraction, *portable* implementation, long-term development, ...

Outline

- **Problem Formulation**
- BEM Solver (Matrix Approach)
- Fast-Multipole Method Approach
    - FMM Algorithm & Parallelization
    - FMM BEM Solver (Experimental Implementation)
- Conclusion & Perspectives

TD-BEM Application Stages

User inputs, simulation parameters

$\downarrow$

Mesh generator, configuration

$\downarrow$

Solver

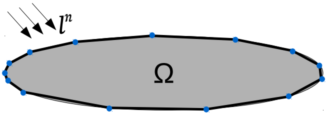
$\downarrow$

Post-processing (TD $\rightarrow$ FD)

Linear Formulation

Notations:

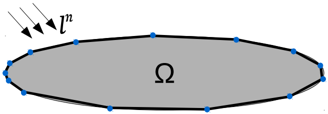
- $\delta\Omega$ discretized in N unknowns/degrees of freedom



Linear Formulation

Notations:

- $\delta\Omega$ discretized in N unknowns/degrees of freedom
- M^k: the convolution matrices (dimension $N \times N$) - input
- I^n: the incident wave emitted by a source on the unknowns at time step n - input
- a^n: the state of the system at time step n - to compute



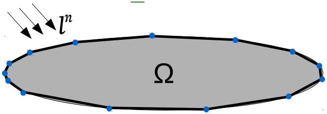
Linear Formulation

Notations:

- $\delta\Omega$ discretized in N unknowns/degrees of freedom
- M^k: the convolution matrices (dimension $N \times N$) - input
- I^n: the incident wave emitted by a source on the unknowns at time step n - input
- a^n: the state of the system at time step n - to compute

Convolution system:

$$M^0 \cdot a^n + \sum_{k \geq 1}^{K^{max}} M^k \cdot a^{n-k} = I^n \quad (1)$$



Linear Formulation

Notations:

- $\delta\Omega$ discretized in N unknowns/degrees of freedom
- M^k: the convolution matrices (dimension $N \times N$) - input
- I^n: the incident wave emitted by a source on the unknowns at time step n - input
- a^n: the state of the system at time step n - to compute

Convolution system:

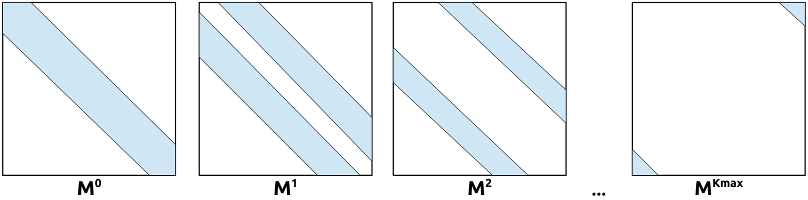
$$M^0 \cdot a^n + \sum_{k \geq 1}^{K^{max}} M^k \cdot a^{n-k} = I^n \quad (1)$$

Solve at each time step:

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

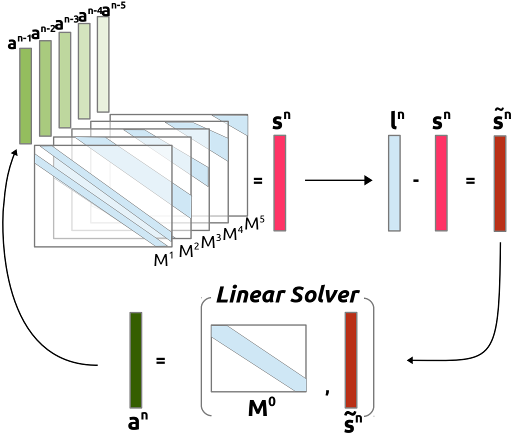
Interaction/Convolution Matrices (M^k)

- Interactions between unknowns
- Symmetric and sparse, $M^k(i,j) \neq 0$ if $distance(i,j) \approx k.c.\Delta t$
- Pre-computed (external tool)



Solve (Schematic View)

$$a^n = (M^0)^{-1} \left(I^n - \sum_{k=1}^{K_{max}} M^k \cdot a^{n-k} \right) \quad (3)$$



SpMV (sparse matrix/vector product)

Summation stage $\rightarrow K^{max}$ SpMVs

- Permutation, advanced storages/kernels, blocking [White III and Sadayappan, 1997, Pinar and Heath, 1999, Pichel et al., 2005, Vuduc and Moon, 2005]
- Auto-tuning [Im and Yelick, 2001, Vuduc et al., 2005]

SpMV (sparse matrix/vector product)

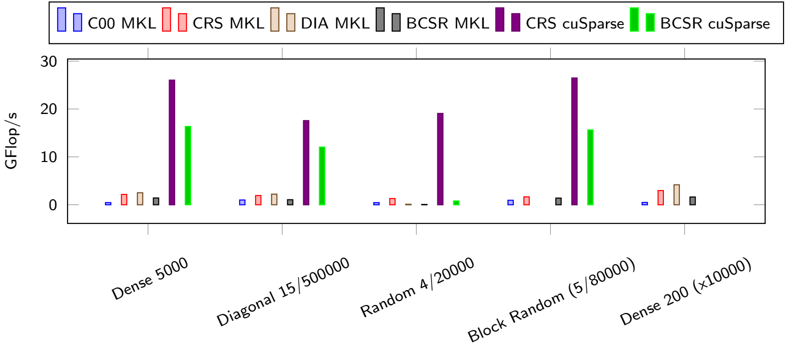
Summation stage $\rightarrow K^{max}$ SpMVs

- Permutation, advanced storages/kernels, blocking
  [White III and Sadayappan, 1997, Pinar and Heath, 1999, Pichel et al., 2005, Vuduc and Moon, 2005]
- Auto-tuning
  [Im and Yelick, 2001, Vuduc et al., 2005]

Low Flop-rate:

- Memory bound operation
  - Flop/Word hardware limit
- Irregular/not contiguous memory accesses
  - Instruction (pipelining, vectorization)
- Not appropriate for GPUs
  [Garland, 2008, Baskaran and Bordawekar, 2008, Bell and Garland, 2009]

SpMV (Performance)



SpMVs MKL/cuSparse (double precision)
Peak performance: CPU Haswell Intel Xeon E5-2680 2,50 GHz core $20 GFlop/s$, and K40-M GPU $1.43 TFlop/s$.

TD-BEM Application Stages

User inputs, simulation parameters

↓

Mesh generator, configuration,
interaction matrices pre-computation

↓

Solver

- Summation stage
- M^0 Linear Solver (external tool)

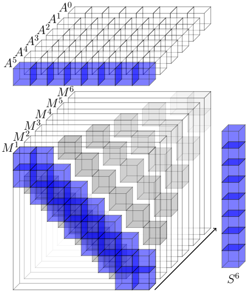
↓

Post-processing (TD $\rightarrow$ FD)

Outline

- Problem Formulation
- **BEM Solver (Matrix Approach)**
- Fast-Multipole Method Approach
  - FMM Algorithm & Parallelization
  - FMM BEM Solver (Experimental Implementation)
- Conclusion & Perspectives

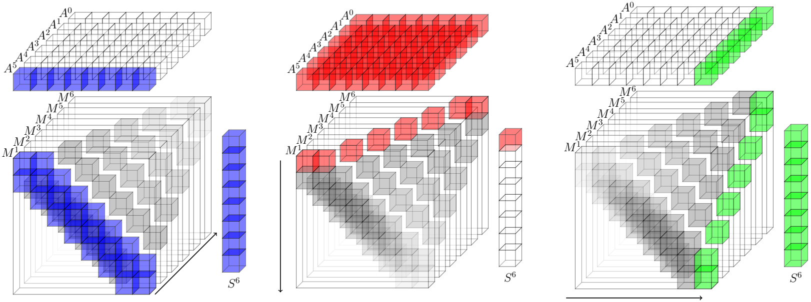
Computational Ordering



Front (k)/ SpMV

$$s^n(i) \quad = \quad \sum_{k=1}^{K_{max}} \sum_{j=1}^{N} M^k(i,j) \times a^{n-k}(j) \, , 1 \leq i \leq N \, . \qquad (4)$$

Computational Ordering



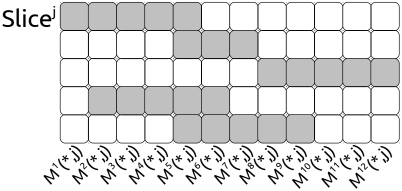
Front (k)/ SpMV Top (i) Side (j)

$$s^n(i) = \sum_{k=1}^{K_{max}} \sum_{j=1}^{N} M^k(i,j) \times a^{n-k}(j) \, , 1 \leq i \leq N \, . \qquad (4)$$

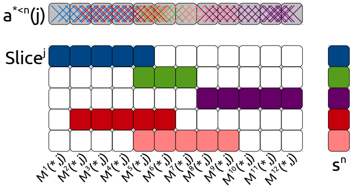
Structure of a Slice Matrix

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 dense vector per row**
- $Slice^j(i,k) = M^k(i,j) \neq 0$
  with $k_s = d(i,j)/(c\Delta t)$ and $k_s \leq k \leq k_s + p$



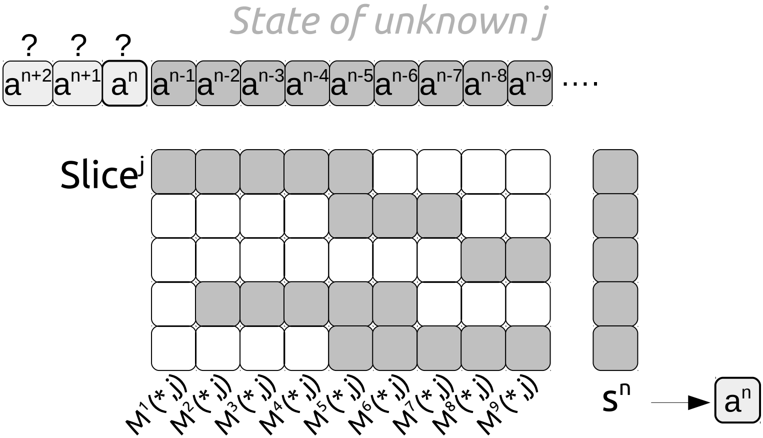
Computing with a Slice Matrix



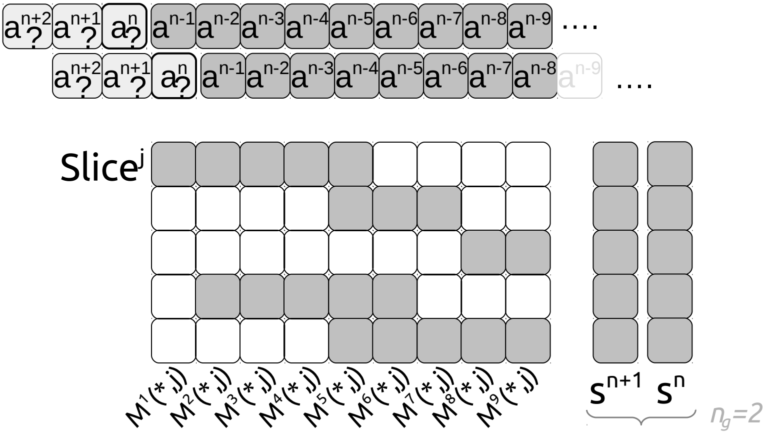
Computation with N vector/vector products (one per line):

- Regular memory access (vectorization, pipelining)
- Low Flop/word ratio (same as SpMV)

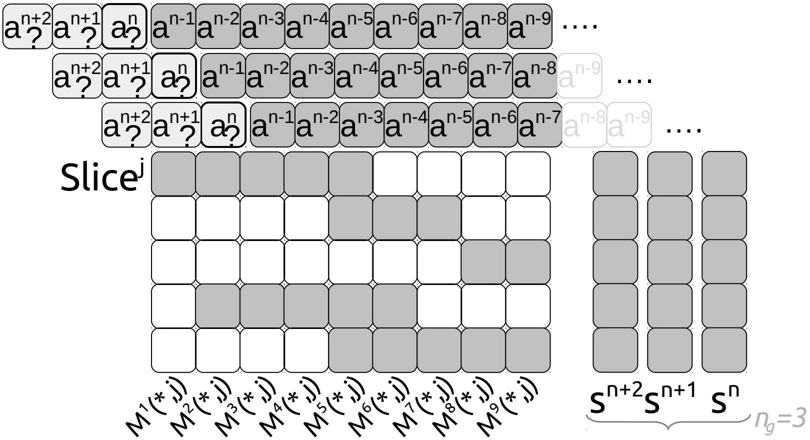
Improving the Flop/Word Ratio



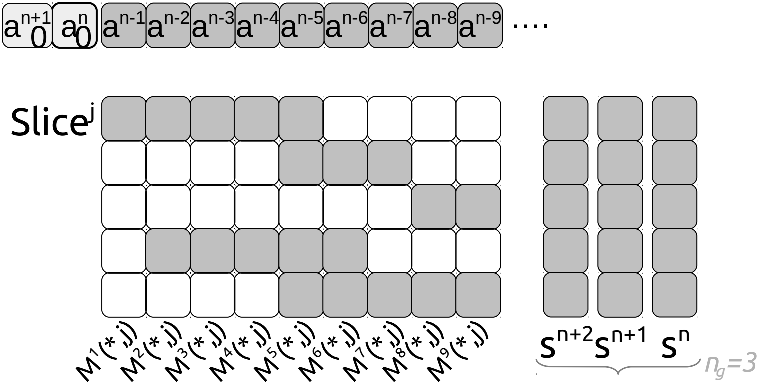
Improving the Flop/Word Ratio



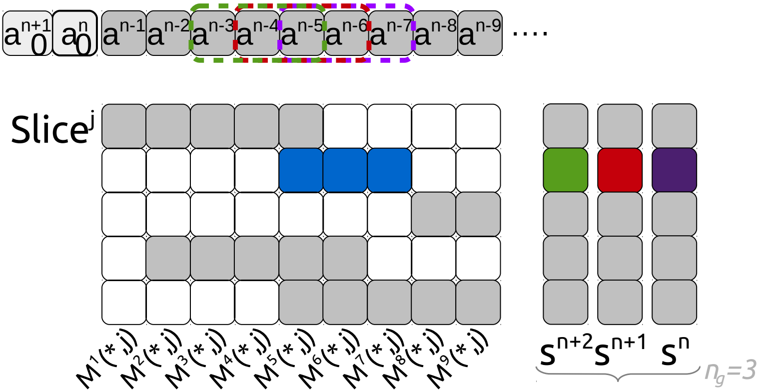
Improving the Flop/Word Ratio



Improving the Flop/Word Ratio

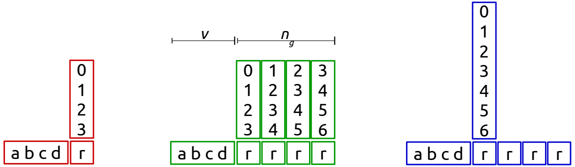


Improving the Flop/Word Ratio



Flop/Word Ratio

Vector length $v = 4$, group size $n_g = 4$ ($v \times n_g \times 2$ Flops):



Vector/vector product

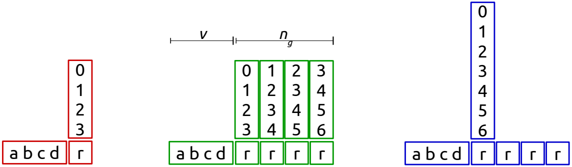
Vector/matrix product

Multi-vectors/vector product

- Vectors product (≈ SpMV) : $n_g(2v + 1)$
- Vector/matrix product : $v + n_g(v + 1)$
- Multi-vectors/vector product : $(v + n_g - 1) + (v) + (n_g)$

Flop/Word Ratio

Vector length $v = 4$, group size $n_g = 4$ ($v \times n_g \times 2$ Flops):

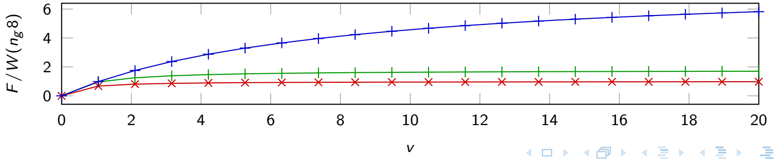


Vector/vector product

Vector/matrix product

Multi-vectors/vector product

- Vectors product ($\approx$ SpMV) : $n_g(2v+1)$
- Vector/matrix product : $v + n_g(v+1)$
- Multi-vectors/vector product : $(v + n_g - 1) + (v) + (n_g)$



Multi-vectors/vector Product (CPU)

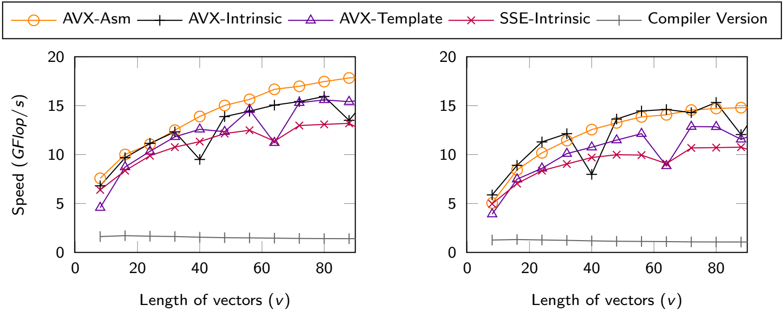


Figure : $N_r = 1\,024$

Figure : $N_r = 20\,480$

Plots show the $GFlop/s$ with $n_g = 8$ for test cases of dimension $N_r \times v$ (in double precision).

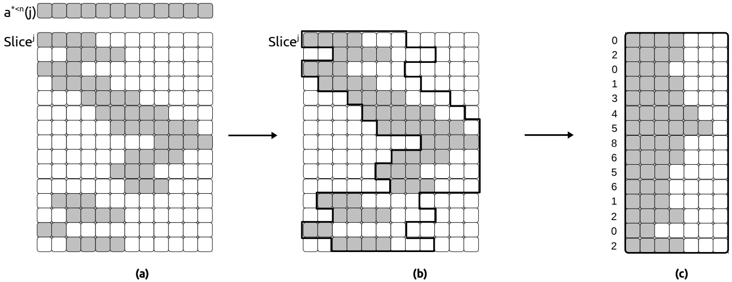
Haswell Intel Xeon E5-2680 at $2,50GHz$ $(20GFlop/s)$

GPUs Slice Storages

- Blocking scheme (small conversion overhead)
- Data access appropriate for SIMT/SIMD
- Memory accesses (coalesced, low bank conflicts)
- Data re-use (shared memory)
- CPU/GPU Balancing

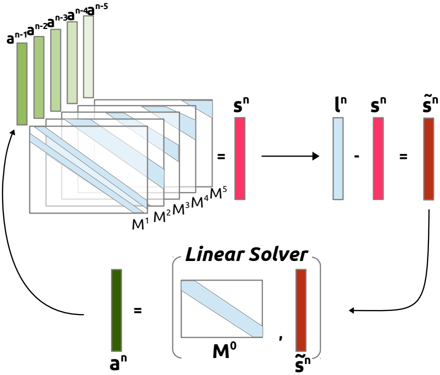
GPUs Slice Storages

- Blocking scheme (small conversion overhead)
- Data access appropriate for SIMT/SIMD
- Memory accesses (coalesced, low bank conflicts)
- Data re-use (shared memory)
- CPU/GPU Balancing

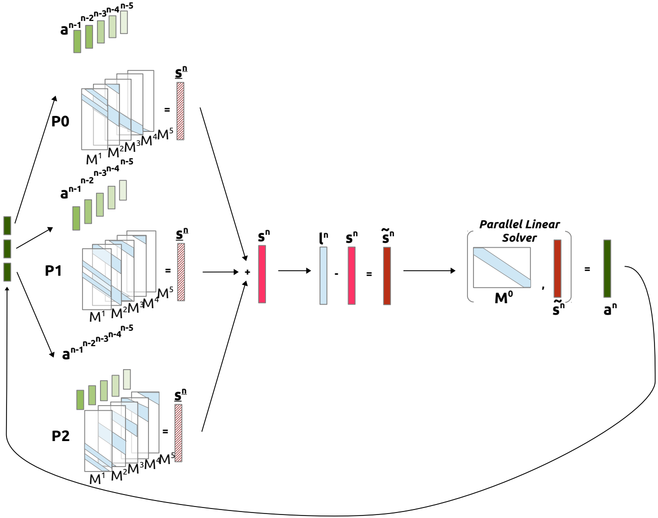


Parallelization

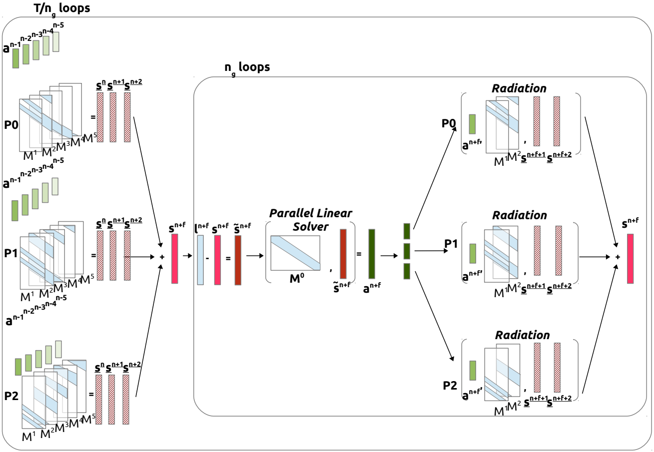
Sequential algorithm:



Parallel Solver (Schematic View)



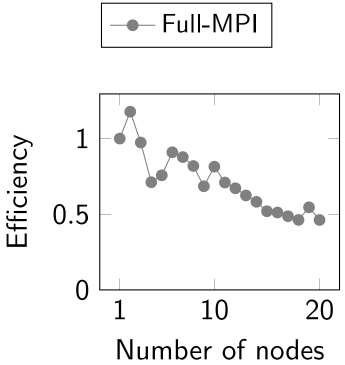
Parallel Solver with $n_g > 1$ (Schematic View)



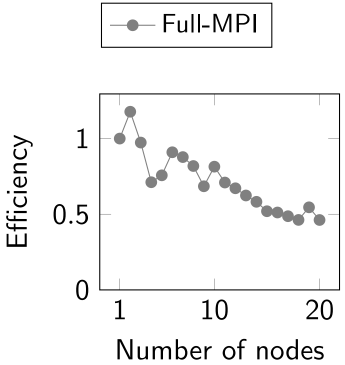
Airplane Simulation

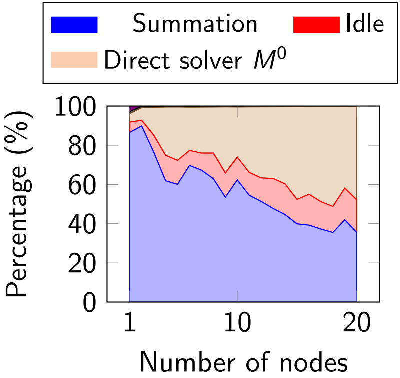
- Acoustics
- $N = 23\,962$
- $10\,823$ time iterations
- $K^{max} = 341$ interaction matrices M^k
- $n_g = 8$
- $70GB$ of data
- double precision
- Homogeneous node: 24 Cores CPU ($128GB$ memory)
- Heterogeneous node: 24 Cores CPU ($128GB$ memory) and 4 K40M GPUs ($12GB$ memory)

Parallel Efficiency/Percentage (Homogeneous)

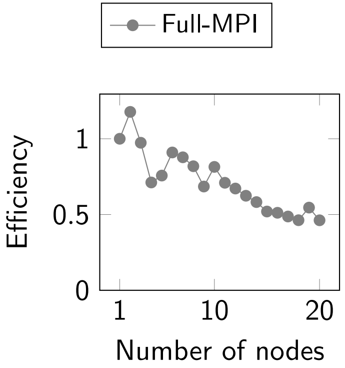


Parallel Efficiency/Percentage (Homogeneous)

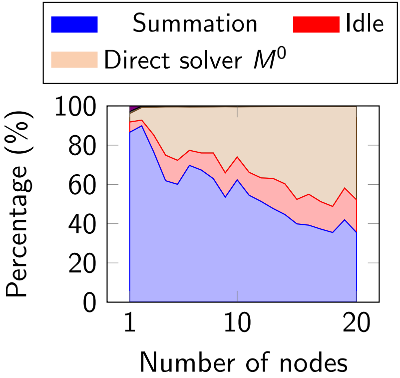




Parallel Efficiency/Percentage (Homogeneous)



Summation stage ↘ M^0 Solve →



With GPUs

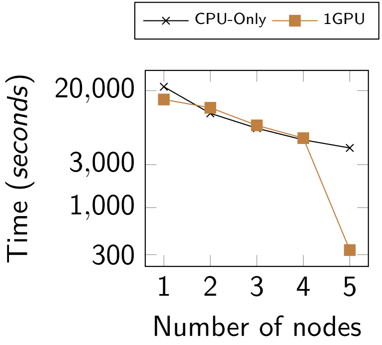


Figure : Execution time

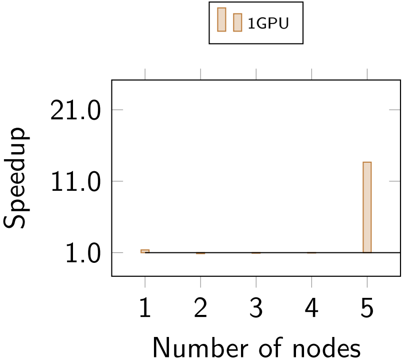


Figure : Speedup against CPU-Only

Problem ≈ $70GB$/GPU $12GB$ memory

With GPUs

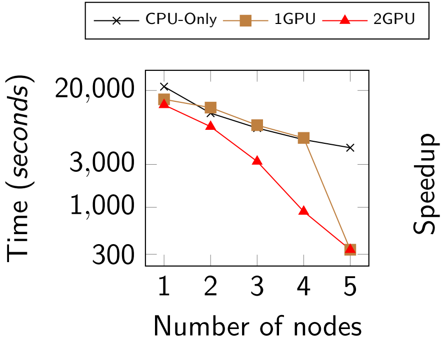


Figure : Execution time

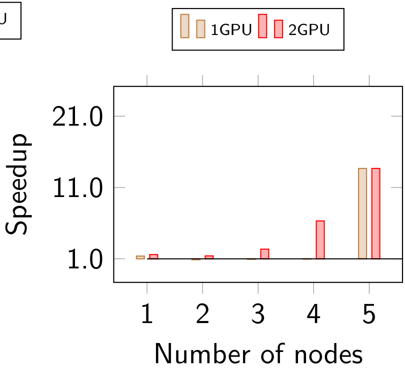


Figure : Speedup against CPU-Only

Problem $\approx 70GB$/GPU $12GB$ memory

With GPUs

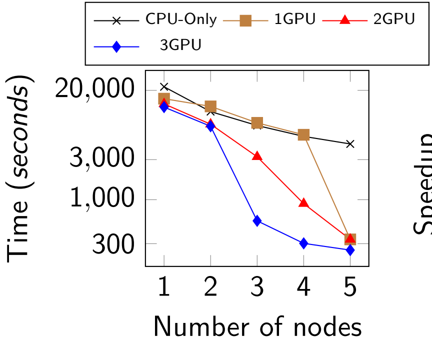


Figure : Execution time

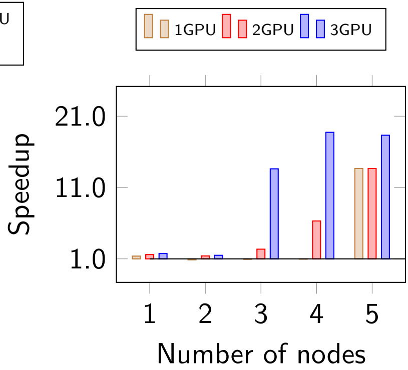


Figure : Speedup against CPU-Only

Problem ≈ $70GB$/GPU $12GB$ memory

With GPUs

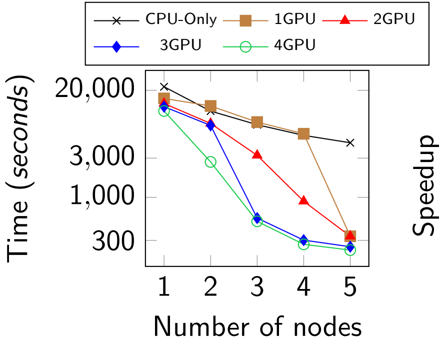


Figure : Execution time

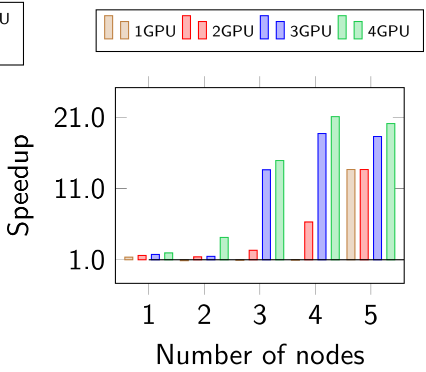


Figure : Speedup against CPU-Only

Problem ≈ $70GB$/GPU $12GB$ memory

Summary:

- New computational ordering [Bramas et al., 2014]
- Solver with few communication points

Additional contributions:

- Permutations/SpMV
- Efficient SIMD kernel CPU
- Efficient blocking scheme/kernel for GPU [Bramas et al., 2015]
- Dynamic balancing (CPU/GPU)

Limits:

- M^0 Linear solver
- GPUs' memory
- Interaction matrices construction
- Complexity $\rightarrow O(N^2)$ for each iteration

Outline

- Problem Formulation
- BEM Solver (Matrix Approach)
- **Fast-Multipole Method Approach**
  - **FMM Algorithm & Parallelization**
  - FMM BEM Solver (Experimental Implementation)
- Conclusion & Perspectives

FMM Operators (1D)

- Spatial decomposition → Potential decomposition

$$f_i = f_i^{near} + f_i^{far}$$

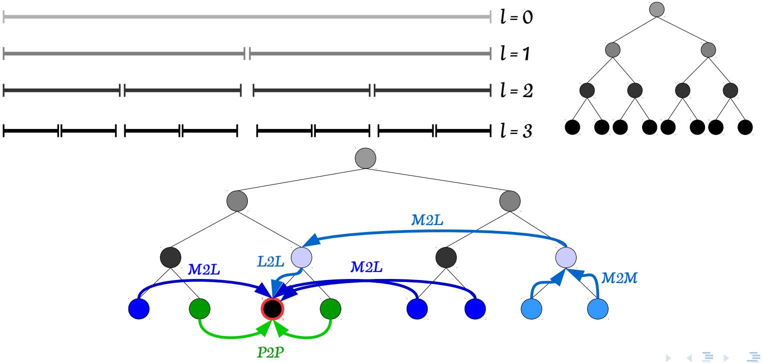
- Near field by direct interactions (leaves)
- Far field with FMM operators (tree)

FMM Operators (1D)

- Spatial decomposition → Potential decomposition

$$f_i = f_i^{near} + f_i^{far}$$

- Near field by direct interactions (leaves)
- Far field with FMM operators (tree)



Related work:

- Multicore study [Chandramowlishwaran et al., 2010]
- NVidia GPU [Yokota and Barba, 2011]
- Distributed GPU [Hamada et al., 2009]
- Distributed CPU/GPU [Hu et al., 2011, Lashuk et al., 2012, Malhotra and Biros, 2015]
- Using a runtime system (multicore) [Ltaief and Yokota, 2014]

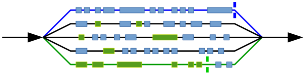
Paradigms

- Fork-join
- Parallel-for (OpenMP)

Paradigms

- Fork-join
- Parallel-for (OpenMP)

- Task-based
- Tasks pool (OpenMP 3.1) [Agullo et al., 2014][1]

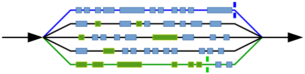


[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2014). Task-based fmm for multicore architectures. SIAM Journal on Scientific Computing, 36(1):C66C93.*

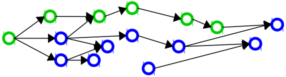
Paradigms

- Fork-join
- Parallel-for (OpenMP)

- Task-based
- Tasks pool (OpenMP 3.1) [Agullo et al., 2014][1]

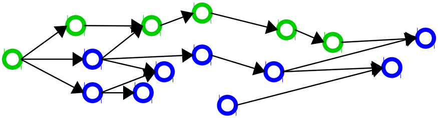


- Tasks-and-dependencies (runtime systems, OpenMP 4)

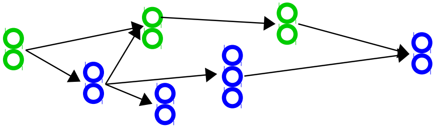


[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2014). Task-based fmm for multicore architectures. SIAM Journal on Scientific Computing, 36(1):C66C93.*

Tasks-and-Dependencies Model (OpenMP 4, StarPU)



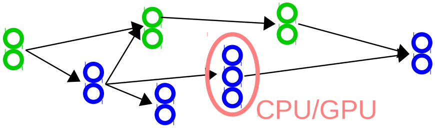
Tasks-and-Dependencies Model (OpenMP 4, StarPU)



Challenges

- Granularity

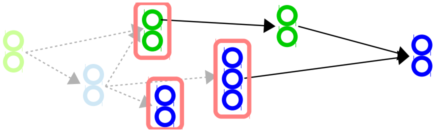
Tasks-and-Dependencies Model (OpenMP 4, *PU)



Challenges

- Granularity
- Computational kernels

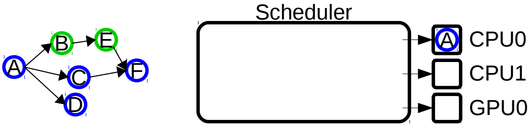
Tasks-and-Dependencies Model (OpenMP 4, *PU)



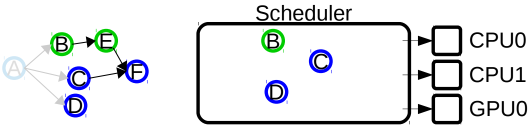
Challenges

- Granularity
- Computational kernels
- Scheduling

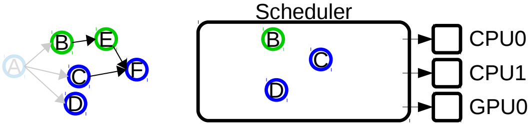
Scheduling



Scheduling

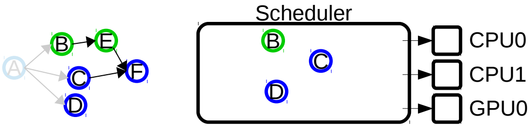


Scheduling



- Priority
- Work stealing [Blumofe and Leiserson, 1999]
- Heterogeneous Earliest Finish Time (Heft) [Topcuouglu et al., 2002]

Scheduling



- Priority
- Work stealing [Blumofe and Leiserson, 1999]
- Heterogeneous Earliest Finish Time (Heft) [Topcuouglu et al., 2002]

Drawbacks:

- Calibration
- Overhead
- Ready-tasks view

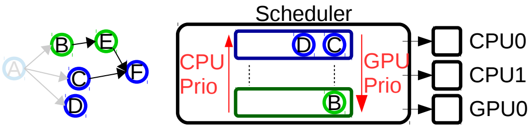
Heteroprio

- Heteroprio [Agullo et al., 2015][1]
  - Steady-state : execute tasks where they have the best acceleration factor
  - Critical-state : execute a task by a worker if it does not delay the hypothetical end

[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2015). Task-based fmm for heterogeneous architectures. Concurrency and Computation: Practice and Experience.*

Heteroprio

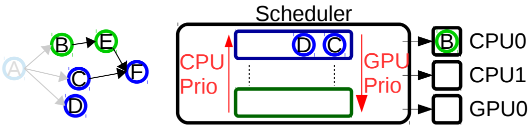
- Heteroprio [Agullo et al., 2015][1]
  - Steady-state : execute tasks where they have the best acceleration factor
  - Critical-state : execute a task by a worker if it does not delay the hypothetical end



[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2015). Task-based fmm for heterogeneous architectures. Concurrency and Computation: Practice and Experience.*

Heteroprio

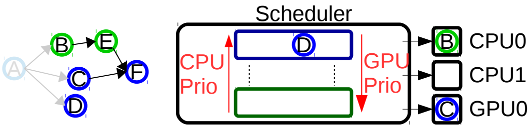
- Heteroprio [Agullo et al., 2015][1]
  - Steady-state : execute tasks where they have the best acceleration factor
  - Critical-state : execute a task by a worker if it does not delay the hypothetical end



[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2015). Task-based fmm for heterogeneous architectures. Concurrency and Computation: Practice and Experience.*

Heteroprio

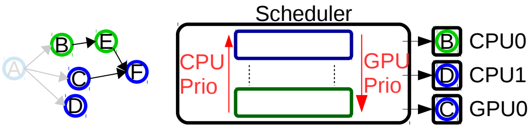
- Heteroprio [Agullo et al., 2015][1]
  - Steady-state : execute tasks where they have the best acceleration factor
  - Critical-state : execute a task by a worker if it does not delay the hypothetical end



[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2015). Task-based fmm for heterogeneous architectures. Concurrency and Computation: Practice and Experience.*

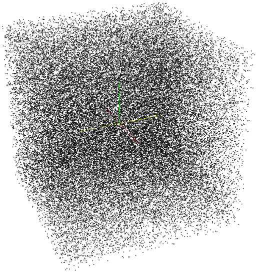
Heteroprio

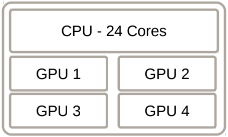
- Heteroprio [Agullo et al., 2015][1]
  - Steady-state : execute tasks where they have the best acceleration factor
  - Critical-state : execute a task by a worker if it does not delay the hypothetical end



[1] *Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2015). Task-based fmm for heterogeneous architectures. Concurrency and Computation: Practice and Experience.*

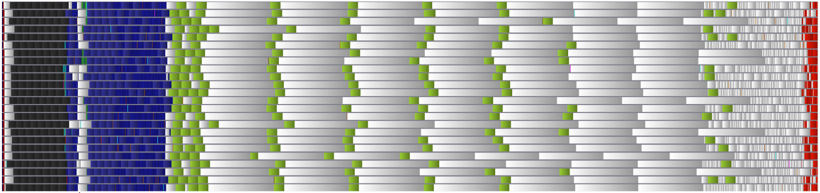
Test Case





- $N = 30$ millions particles
- Spherical Expansion/Rotation Kernel
- $Acc = 10^{-3}$, $h = 7$ and $Granularity = 1500$

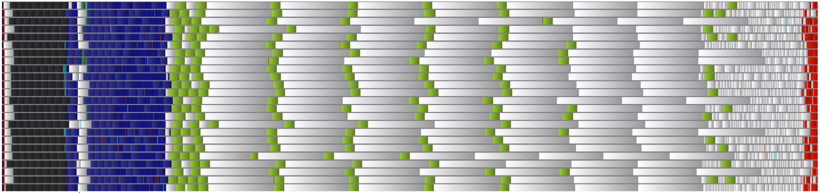
Trace - Heterogeneous (24CPUs)



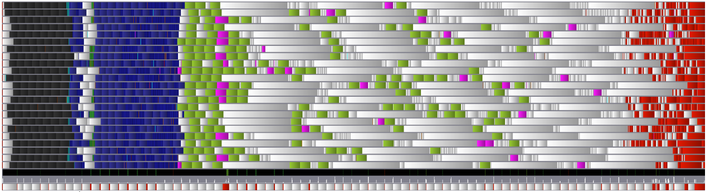
0GPU/15.5s

Legend: P2P ( ), P2M ( ) , M2M ( ) , M2L ( ), L2L ( ), L2P ( ) and Idle ( )

Trace - Heterogeneous (1GPU/23CPUs)



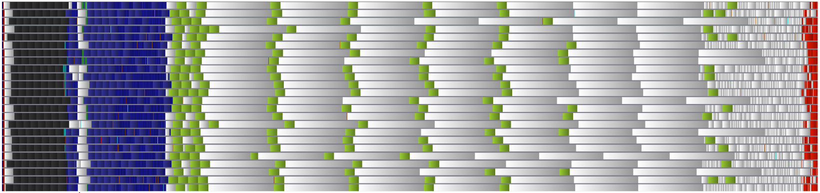
0GPU/15.5*s*



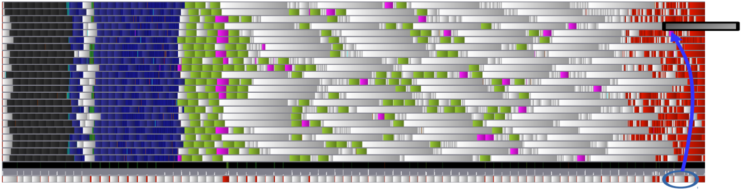
1GPU/13.4*s*

Legend: P2P ( ), P2M ( ) , M2M ( ) , M2L ( ), L2L ( ), L2P ( ) and Idle ( )

Trace - Heterogeneous (1GPU/23CPUs)



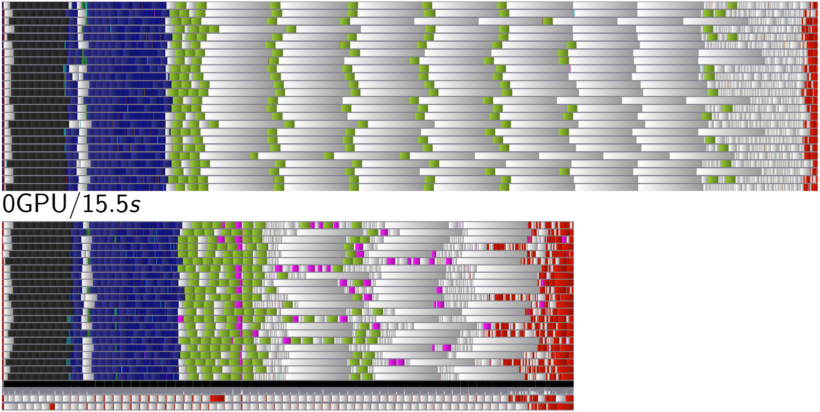
0GPU/15.5s



1GPU/13.4s

Legend: P2P ( ), P2M ( ) , M2M ( ) , M2L ( ), L2L ( ), L2P ( ) and Idle ( )

Trace - Heterogeneous (2GPUs/22CPUs)

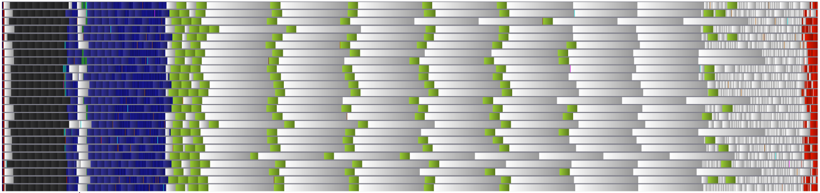


0GPU/15.5s

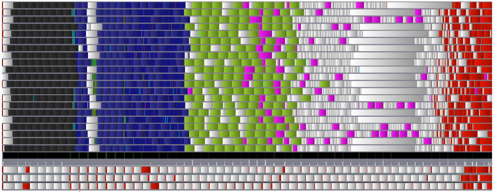
2GPU/10.9s

Legend: P2P ( ), P2M ( ) , M2M ( ) , M2L ( ), L2L ( ), L2P ( ) and Idle ( )

Trace - Heterogeneous (3GPUs/21CPUs)



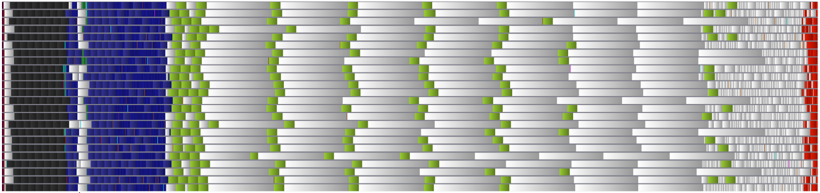
0GPU/15.5*s*



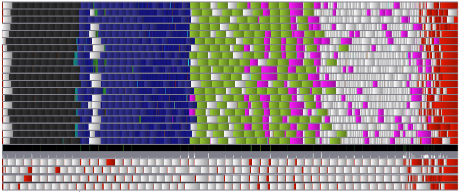
3GPU/9.4*s*

Legend: P2P (■), P2M (■) , M2M (■) , M2L (■), L2L (■), L2P (■) and Idle (■)

Trace - Heterogeneous (4GPUs/20CPUs)



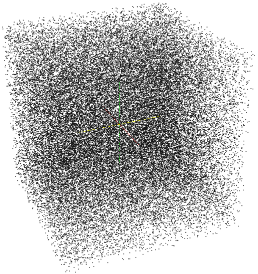
0GPU/15.5s

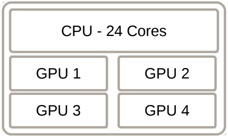


4GPU/8.7s

Legend: P2P (■), P2M (■) , M2M (■) , M2L (■), L2L (■), L2P (■) and Idle (■)

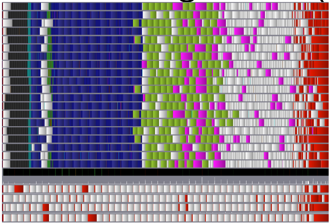
Test Case





- $N = 30$ millions particles
- **Uniform/Lagrange kernel**
- **Acc** $= \{\mathbf{10^{-5}}, \mathbf{10^{-7}}\}$, $h = 7$ and $Granularity = 1500$

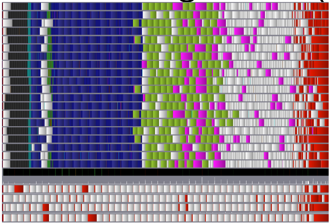
Trace - Heterogeneous (4GPUs)



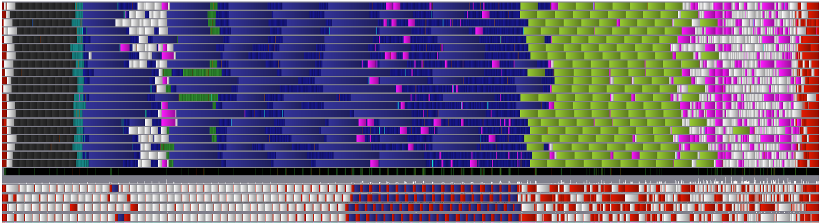
$Acc = 10^{-5}/7.9s$

Legend: P2P (■), P2M (■) , M2M (■) , M2L (■), L2L (■), L2P (■) and Idle (■)

Trace - Heterogeneous (4GPUs)



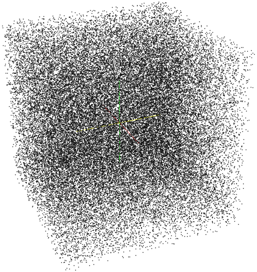
$Acc = 10^{-5}/7.9s$

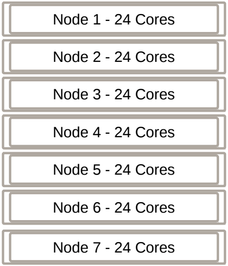


$Acc = 10^{-7}/17s$

Legend: P2P ( ), P2M ( ) , M2M ( ) , M2L ( ), L2L ( ), L2P ( ) and Idle ( )

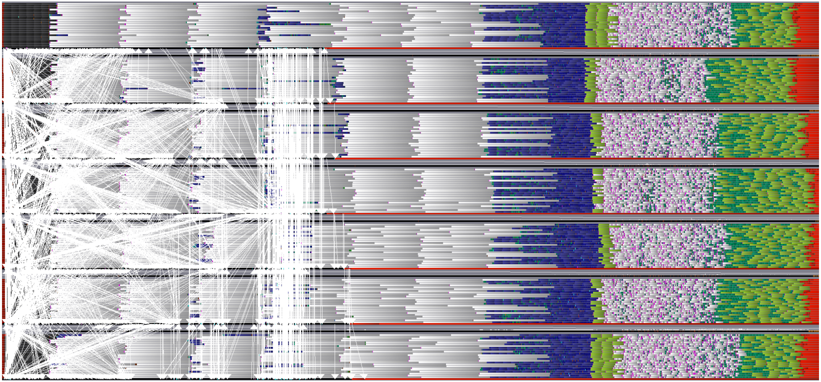
Test Cases





- $N = 200$ millions particles
- Spherical Expansion/Rotation Kernel
- $Acc = 10^{-3}$, $h = 8$ and $Granularity = 2000$

Trace - 7 nodes × 24CPUs



Legend: P2P ( ), P2M ( ) , M2M ( ) , M2L ( ), L2L ( ), L2P ( ) and Idle ( ) .

Summary:

- Generic
- Kernel independent
- Architecture independent
- Performance portability

Summary:

- Generic
- Kernel independent
- Architecture independent
- Performance portability

Additional contributions:

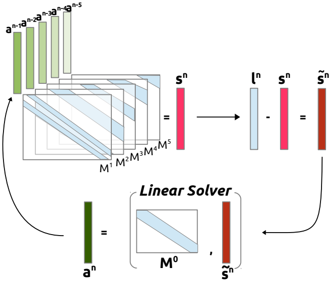
- Commutativity expression in FMM
- MPI/OpenMP implementation

All included in ScalFMM (C++/HPC library)

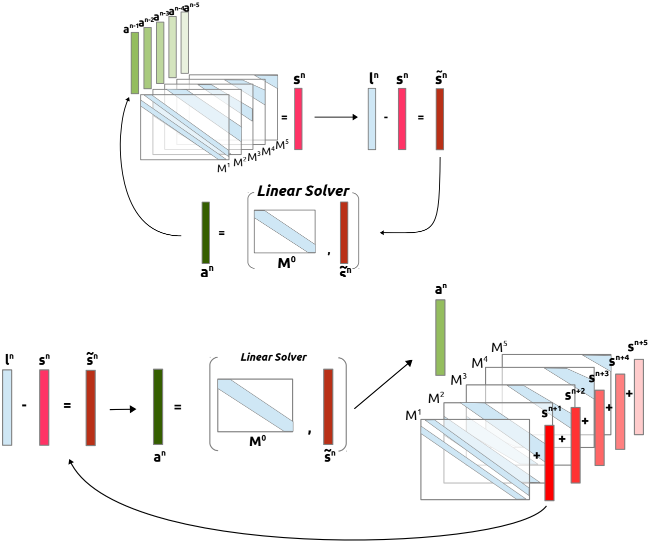
Outline

- Problem Formulation
- BEM Solver (Matrix Approach)
- **Fast-Multipole Method Approach**
  - FMM Algorithm & Parallelization
  - **FMM BEM Solver (Experimental Implementation)**
- Conclusion & Perspectives

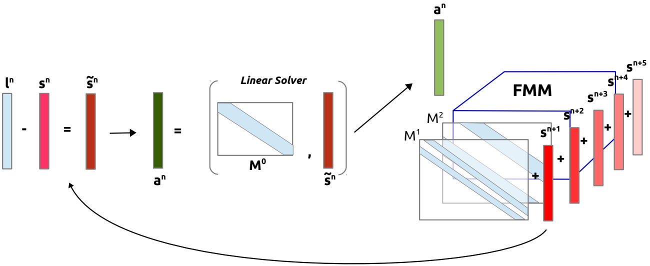
Propagation of the Current State to the Future



Propagation of the Current State to the Future



With FMM



- Far interactions in time (between far elements in space) are computed by the FMM
- The spatial decomposition is given by the octree

Overview

- The octree is over a mesh (integration points)
- Interactions matrices between leaves
- Approximation/FMM
  - development in the time-domain
    - multipole: what a cell emits to the outside
    - local: what a cell receives from the outside
  - operators in FD or TD
  - accurate up-to a chosen frequency
    - the results in the TD of the matrix approach $\neq$ FMM

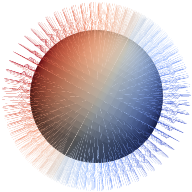


Figure : Complete unit sphere

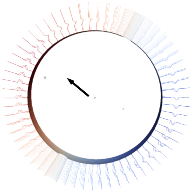
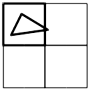
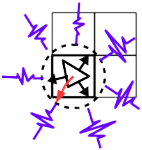


Figure : Truncated unit sphere

Operators (Overview)

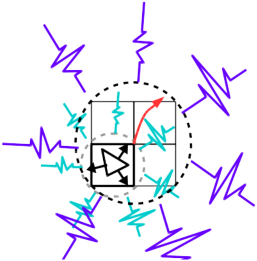
- P2M
  - compute what is emitted by a leaf to the outside

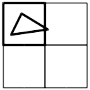




Operators (Overview)

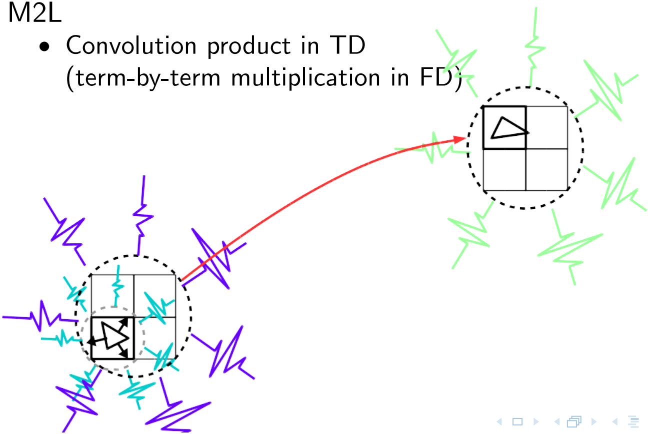
- P2M
  - compute what is emitted by a leaf to the outside
- M2M/L2L
  - Extrapolation + time shift





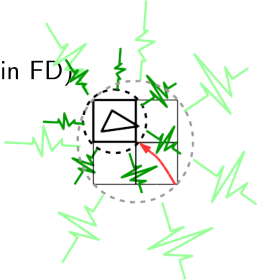
Operators (Overview)

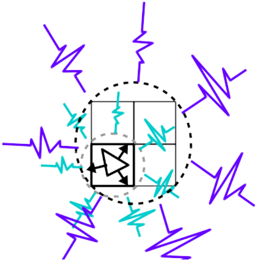
- P2M
  - compute what is emitted by a leaf to the outside
- M2M/L2L
  - Extrapolation + time shift
- M2L
  - Convolution product in TD
    (term-by-term multiplication in FD)



Operators (Overview)

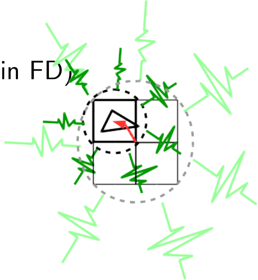
- P2M
  - compute what is emitted by a leaf to the outside
- M2M/L2L
  - Extrapolation + time shift
- M2L
  - Convolution product in TD
    (term-by-term multiplication in FD)

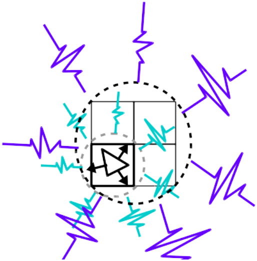




Operators (Overview)

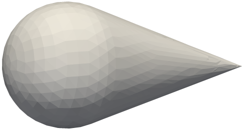
- P2M
  - compute what is emitted by a leaf to the outside
- M2M/L2L
  - Extrapolation + time shift
- M2L
  - Convolution product in TD
    (term-by-term multiplication in FD)
- L2P
  - Integration





Cone-Sphere Test Cases

Case	C-927	C-4269	C-10012
Number of unknowns	927	4269	10012
FMM tree height	3	4	5
Number of leaves	16	64	234
Number of M^k matrices (K^{max})	117	244	370
Number of M^k matrices (leaves)	60	64	49
Number of time steps (T)	2033	4345	6647



Sequential Executions

TD vs. FD operators:

Stages	FMM TD	FMM TD + FD-M2L	FMM FD	Matrix approach
M^k Construction	76 s	76 s	76 s	242 s
Solve	58 122 s	53 241 s	97 861s	7.8 s(*)
Total	58 198 s	53 317 s	97 937 s	249.8 s

Execution time TD-FMM Vs. matrix approach to solve the Case C-927 in double precision.

(*) Our optimized BEM solver.

Parallel Executions (FMM Vs. Matrix Approach)

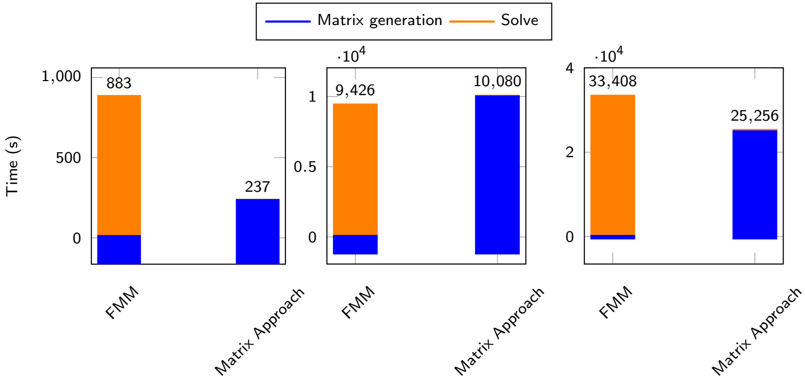


Figure : C-927 (×3.8) Figure : C-4269 (×1) Figure : C-10012 (×1.4)

The captions of the different cases show the overhead of the FMM TD-BEM against the matrix approach.

Summary:

- Preliminary results
- Best configuration: TD + FD M2L
- Not competitive against the direct approach (maybe on larger test cases)
- Any improvement of the matrix creation will make the FMM less competitive

Additional contributions:

- Incomplete/4D FMM
- Sphere discretization/length APS signal

Conclusion & Perspectives

Conclusion

Dense BEM/Matrix Approach

- Based on a new computational order
- Remove the bottleneck of the SpMV
- Implemented efficiently on modern architectures
- Complete BEM solver

Conclusion

FMM

- Generic and state-of-the-art library
- Several parallelization strategies
- Robust OpenMP/MPI implementation (10 billions particles)
- Modern task-based approach
- ScalFMM

Conclusion

FMM BEM Solver (Preliminary)

- Parallelized using ScalFMM
- Best configuration TD operators + FD M2L
- Our implementation is not faster than the direct approach

Perspectives

TD-BEM

- Improve the construction of the interaction matrices
- M^0 linear solver: small matrix, lots of nodes
- Compare existing solvers (TD vs. FD)

Perspectives

FMM parallelization

- Task-based with implicit MPI communications
- Group-Tree update

Perspectives

FMM BEM

- Study the cost of the solve compare to the direct approach (complexity for some cases)
- Lots of remaining optimizations to test

Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2014).
Task-based fmm for multicore architectures.
SIAM Journal on Scientific Computing, 36(1):C66–C93.

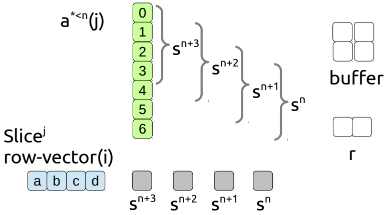
Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., and Takahashi, T. (2015).
Task-based fmm for heterogeneous architectures.
Concurrency and Computation: Practice and Experience, pages n/a–n/a.
cpe.3723.

Baskaran, M. M. and Bordawekar, R. (2008).
Optimizing sparse matrix-vector multiplication on gpus using compile-time and run-time strategies.
IBM Reserach Report, RC24704 (W0812-047).

Bell, N. and Garland, M. (2009).
Implementing sparse matrix-vector multiplication on

Vectorized Implementation (Overview)

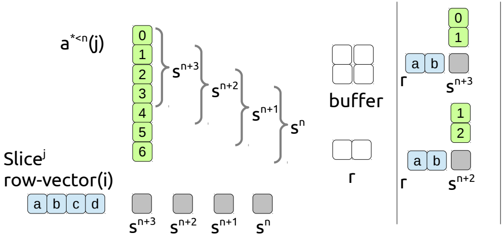
The algorithm is tied to the vectorization type size l_{SIMD}
Example with $n_g = 4$, $v = 4$, $l_{SIMD} = 2$:



- Each value from the vectors is read only once (and maybe copied into the buffer)
- The values in the buffer are shifted to avoid reloading

Vectorized Implementation (Overview)

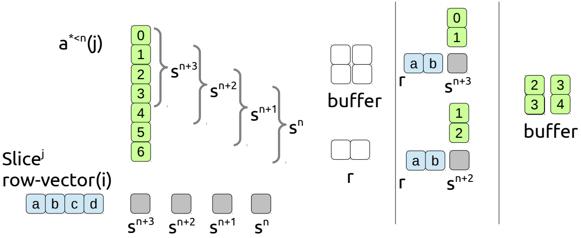
The algorithm is tied to the vectorization type size l_{SIMD}
Example with $n_g = 4$, $v = 4$, $l_{SIMD} = 2$:



- Each value from the vectors is read only once (and maybe copied into the buffer)
- The values in the buffer are shifted to avoid reloading

Vectorized Implementation (Overview)

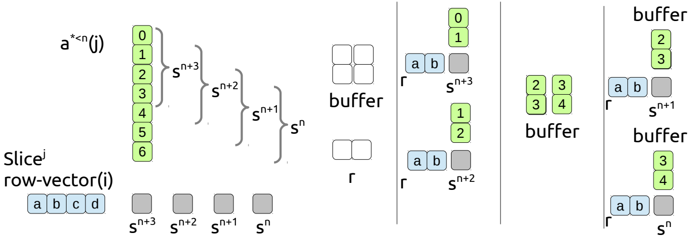
The algorithm is tied to the vectorization type size l_{SIMD}
Example with $n_g = 4$, $v = 4$, $l_{SIMD} = 2$:



- Each value from the vectors is read only once (and maybe copied into the buffer)
- The values in the buffer are shifted to avoid reloading

Vectorized Implementation (Overview)

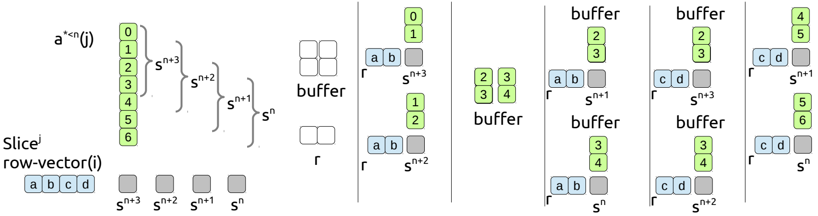
The algorithm is tied to the vectorization type size l_{SIMD}
Example with $n_g = 4,\ v = 4,\ l_{SIMD} = 2$:



- Each value from the vectors is read only once (and maybe copied into the buffer)
- The values in the buffer are shifted to avoid reloading

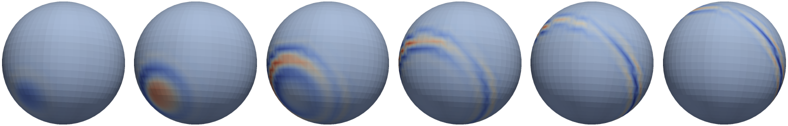
Vectorized Implementation (Overview)

The algorithm is tied to the vectorization type size l_{SIMD}
Example with $n_g = 4$, $v = 4$, $l_{SIMD} = 2$:



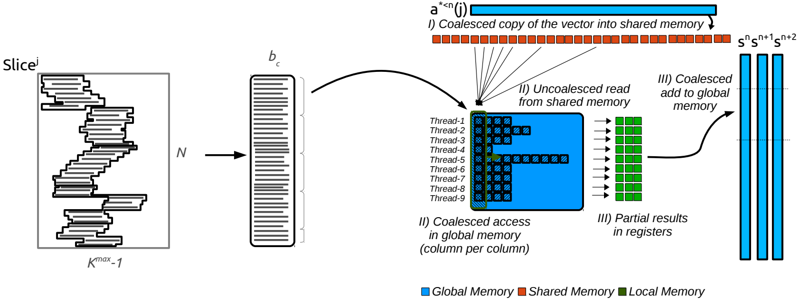
- Each value from the vectors is read only once (and maybe copied into the buffer)
- The values in the buffer are shifted to avoid reloading

In Time or Frequency Domain



Propagation of the wave for several time steps on a target discretized sphere. The different spheres represent the values that will be applied on the included mesh elements.

Contiguous-Blocking Computational Kernel



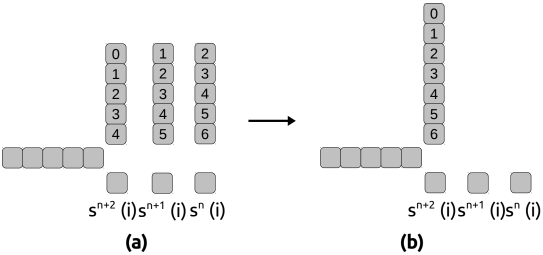
(a) **(b)** **(c)**

(a) the original slice is transformed in a block during the pre-computation stage ($n_g = 3$, $b_c = 11$)

(b) the blocks are moved to the device memory for the summation stage

(c) a thread-block ($nb - threads = 9$) is in charge of the blocks from a slice interval and computes several summation vectors

Multi-vectors/vector Product



Computing one slice-row with 3 vectors ($n_g = 3$)

(a) using 3 scalar products

(b) using the multi-vectors/vector product

In Shared Memory

- OpenMP parallelization
- Summation divided/balanced between the threads
- No communication during the summation
- Multi-threaded M^0 linear solver if possible
- NUMA effects are not handled

On Heterogeneous Nodes

- OpenMP parallelization
- One thread/core per GPU
- An interval of the slices is moved on each GPU
- Intervals are balanced between each iteration with a greedy algorithm
- The memory limit of the GPU may reduce their performance

Application Example

FD-BEM + FMM (antenna at 1GHz)

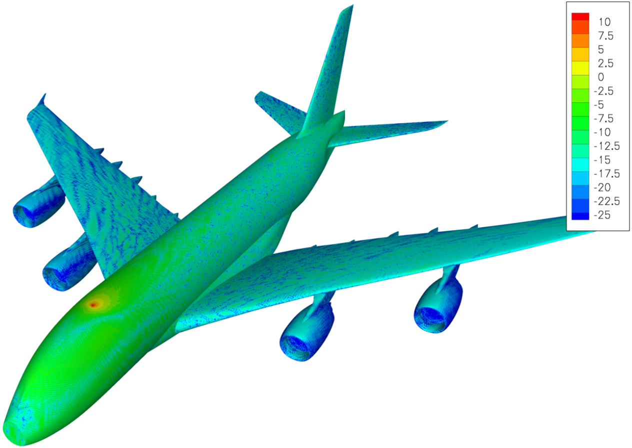
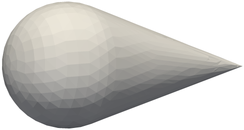


Image from Airbus Group.

Cone-Sphere Test Cases

Case	C-927	C-4269	C-10012	C-22468
Number of unknowns	927	4269	10012	C-22468
FMM tree height	3	4	5	6
Number of leaves in the FMM tree	16	64	234	936
Number of NNZ interaction matrices (K^{max})	117	244	370	551
Number of NNZ matrices between FMM leaves	60	64	49	37
Number of time steps (T)	2033	4345	6647	9957
Size of the simulation box	3.3	7.3	11	16
F_{max}	348	337	335	334
Incomplete FMM coefficient $l = h - 1$	16	18	13	10
Incomplete FMM coefficient $l = 2$	16	36	52	80



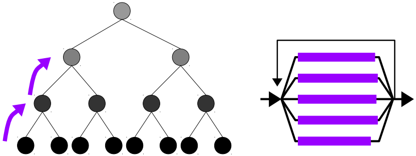
Multi-vectors/vector Product (GPU)

For the Contiguous-Blocking scheme:

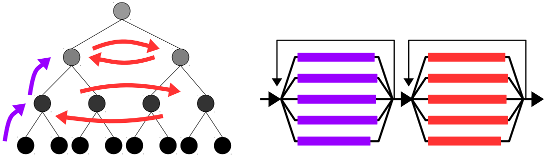
	Width (b_c)							
	16	32	64	128	16	32	64	128
	GPU				CPU			
Single	243	338	431	496 (11%)	4.3	5.5	7.8	6.8 (17%)
Double	143	199	248	286 (20%)	3.9	5.6	4.2	4.3 (21%)

GFlop/s for 420 slices (6400 rows and b_c columns)
(%) percentage of the peak performance

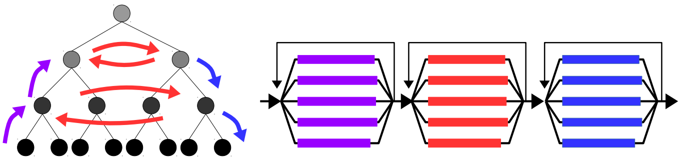
Fork-join (OpenMP)



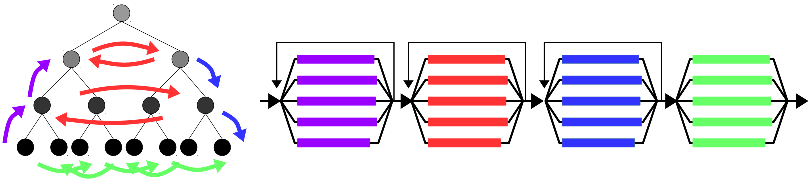
Fork-join (OpenMP)



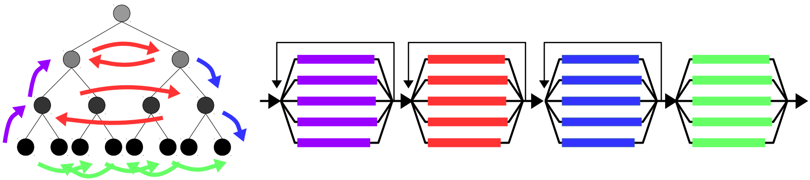
Fork-join (OpenMP)



Fork-join (OpenMP)

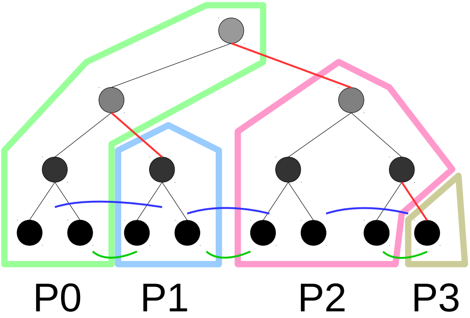


Fork-join (OpenMP)



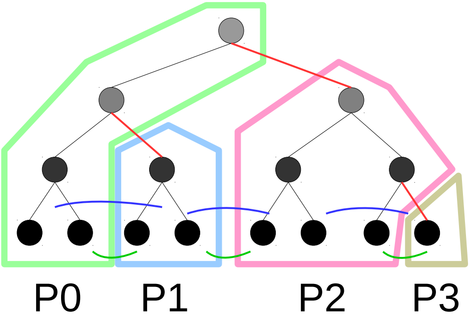
- Level by level
- Critical balancing
- Possible bottleneck (top of the tree)
- Difficult to mix near/far fields

Fork-join+Message-passing (Hybrid OpenMP/MPI)



- Distribute the tree between nodes
- Progress level by level
- Communication between all stages

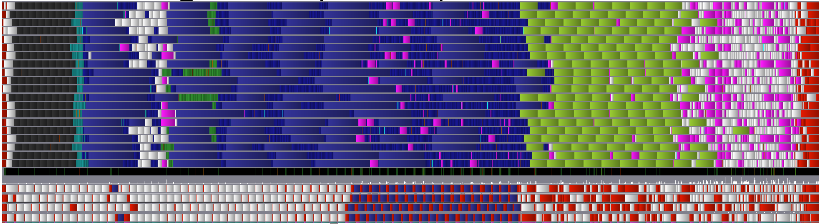
Fork-join+Message-passing (Hybrid OpenMP/MPI)



- Distribute the tree between nodes
- Progress level by level
- Communication between all stages

Poor parallelism expression

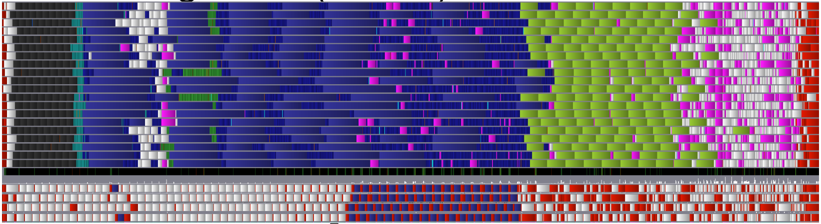
Trace - Heterogeneous (4GPUs)



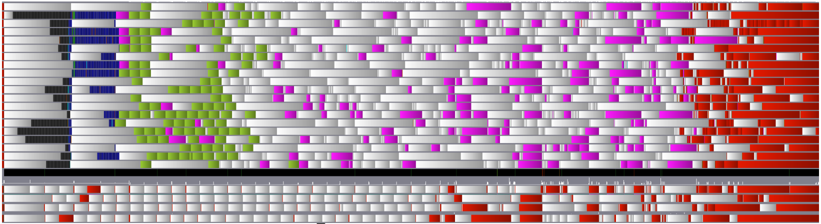
$h = 7/n_g = 1500/Acc = 10^{-7}/17s$

24 threads, $N = 30$ millions, uniform distribution, Uniform/Lagrange

Trace - Heterogeneous (4GPUs)



$h = 7 / n_g = 1500 / Acc = 10^{-7} / 17s$



$h = 6 / n_g = 300 / Acc = 10^{-7} / 39s$ *(not on the same scale)*

24 threads, $N = 30$ millions, uniform distribution, Uniform/Lagrange

Flop/Cost Estimation

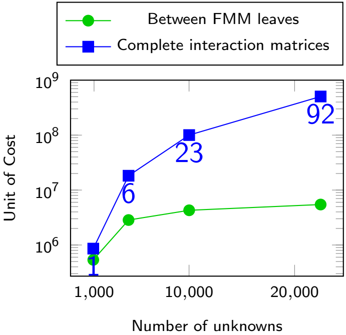


Figure : Matrix generation cost estimation

The numbers above the slower plot represent the slow-down factors against the faster method.

Flop/Cost Estimation

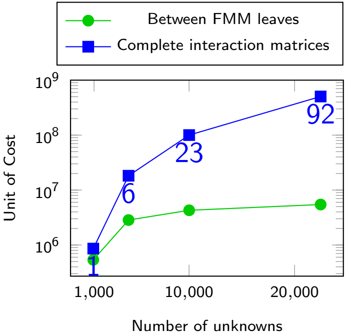


Figure : Matrix generation cost estimation

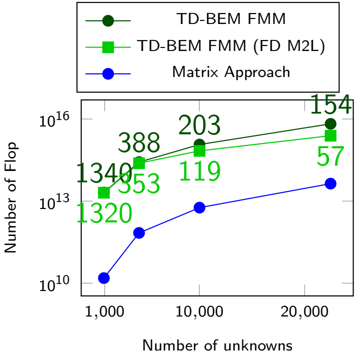
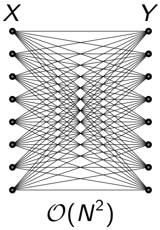


Figure : Summation stage Flop estimation

The numbers above the slower plot represent the slow-down factors against the faster method.

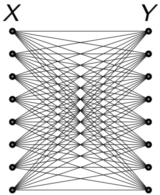
Reducing the Complexity

Direct computation $0(N^2)$

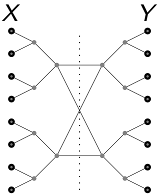


Reducing the Complexity

Direct computation $0(N^2)$ $\rightarrow$ FMM $O(N)$



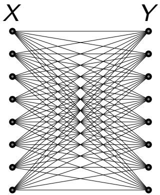
$\mathcal{O}(N^2)$



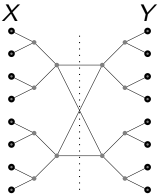
$\mathcal{O}(N)$

Reducing the Complexity

Direct computation $0(N^2)$ $\rightarrow$ FMM $O(N)$



$\mathcal{O}(N^2)$



$\mathcal{O}(N)$

- Spatial decomposition $\rightarrow$ Potential decomposition

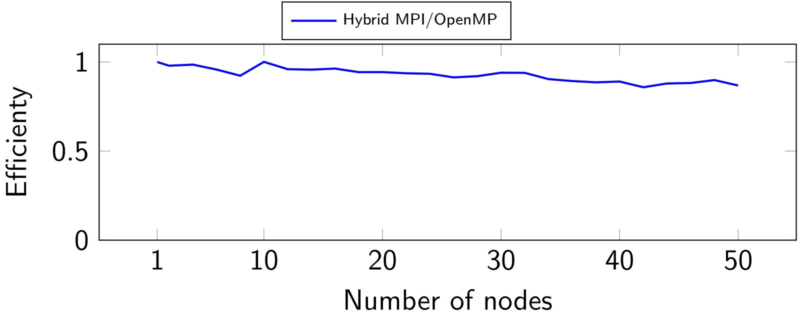
$$f_i = f_i^{near} + f_i^{far}$$

- Near field is computed by direct interactions
- The far field is done using different operators

Airplane Simulation

- Acoustics
- $N = 23\,962$
- 10 823 time iterations
- $K^{max} = 341$ interaction matrices M^k ($\approx 5.5 \times 10^9$ NNZ)
- Computing $s^n \approx 11\ GFlop$
- Total $\approx 130\,651\ GFlop$
- $n_g = 8$
- $70GB$ of data
- CPU node : 2 Dodeca-core Haswell Intel Xeon E5-2680 at $2,50GHz$ and $128GB$ (DDR4) of shared memory
- GPUs per node: 4 NVIDIA Kepler K40M GPU ($745MHz$), 2880 Cores, $12GB$ of dedicated memory

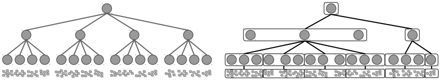
Hybrid MPI/OpenMP



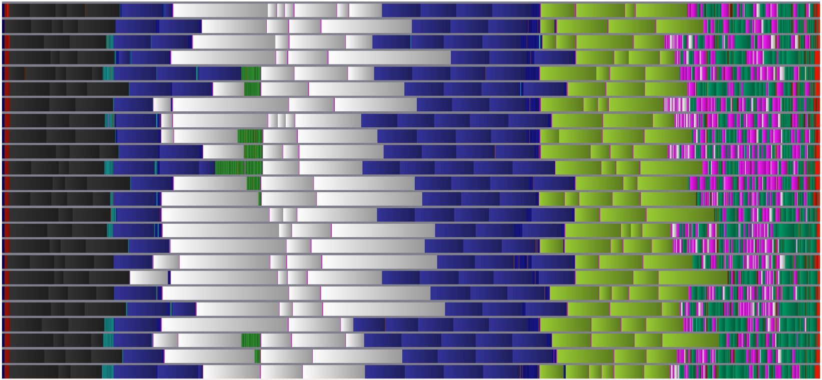
Efficiency: Uniform distribution, Spherical Expansion/Rotation Kernel, $N = 200$ millions, $h = 8$, $Acc = 10^{-3}$, from 1 to 50 nodes (24 threads per node), for $np = 50$ the execution time is $2.24s$

Group-tree

- Granularity G
- A group $\rightarrow$ G cells/leaves
- Good locality
- Low iteration complexity
- Dependencies between cells $\neq$ between groups



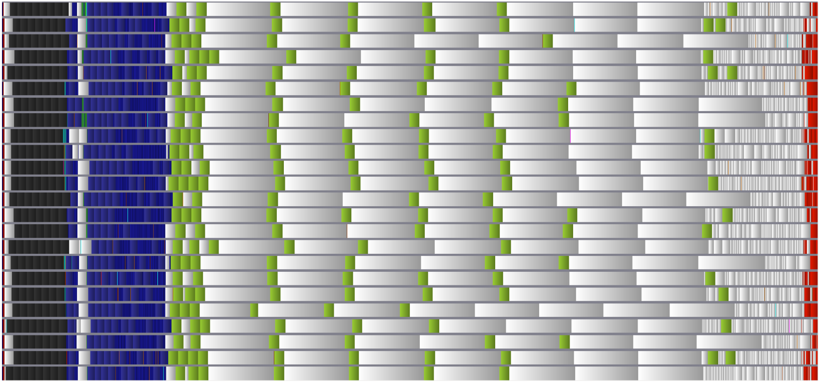
Trace - Shared Memory - 24CPUs



$N = 20$ millions, ellipsoid distribution, Spherical Expansion/Rotation Kernel, $Acc = 10^{-3}$, $h = 11$ and $n_g = 8000$ in $5.2s$.

Legend: P2P (■), P2M (■) , M2M (■) , M2L (■), L2L (■), L2P (■) and Idle (■)

Trace - 24CPUs



$N = 30$ millions, uniform distribution, Spherical Expansion/Rotation Kernel, $Acc = 10^{-3}$, $h = 7$ and $n_g = 1500$ in 15.5s.

Legend: P2P (■), P2M (■) , M2M (■) , M2L (■), L2L (■), L2P (■) and Idle (■)

Test Cases

Interactions between N particles for two distributions:

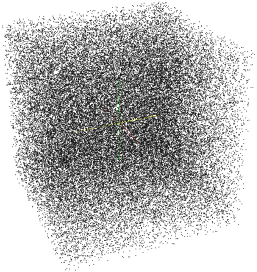


Figure : Uniform

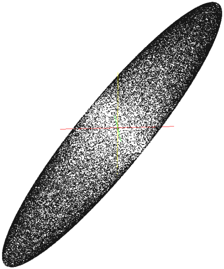


Figure : Ellipsoid

The height of the tree (h) is chosen such that the execution time is minimal in sequential.

Parallel Strategies for FMM BEM

Three strategies (Fork-join OpenMP)

- Threaded FMM: divide each level between threads (ScalFMM classic)

- Threaded kernel: divide the work inside the kernel

- Mix FMM/Kernel: two layers of parallelism, one in the FMM and a second in the kernel

Parallel Executions

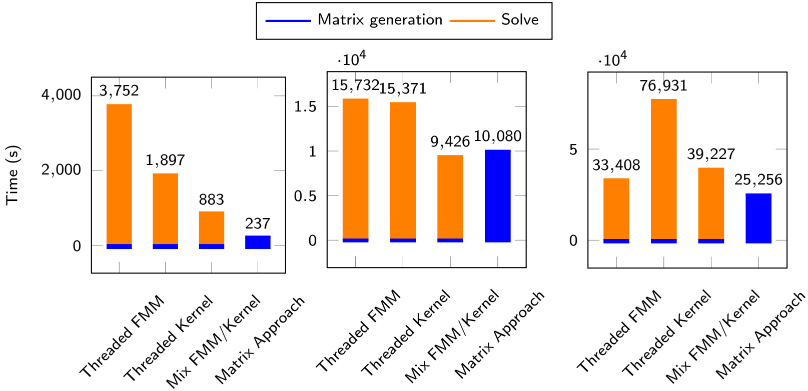


Figure : C-927

Figure : C-4269 ($\times 1$)

Figure : C-10012

The captions of the different cases show the overhead of the FMM TD-BEM against the matrix approach.

Parallel Executions FMM Vs. Matrix Approach