OptiDis: a MPI/OpenMP Dislocation Dynamics Code for Large Scale Simulations

7th MMM, Berkeley 2014

Etcheverry Arnaud [1] Blanchard Pierre [1] Coulaud Olivier [1]
Dupuy Laurent [2]

[1]Inria HiePACS , [2]CEA Saclay

October 10, 2014

Outline

Multi-Scale: time and size simulation

Molecular dynamics

(nm^3/ns)

Observe:
Dislocation mobility and reaction

Dislocation Dynamics

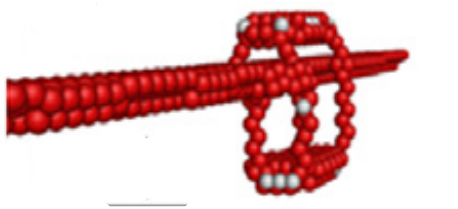
($\mu m^3/\mu s$)

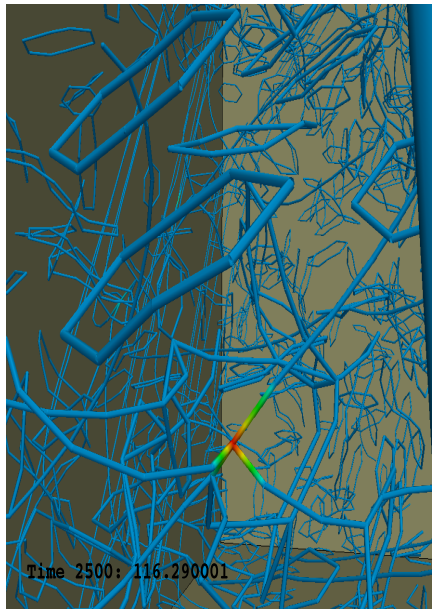
Observe:
Stress-strain behavior

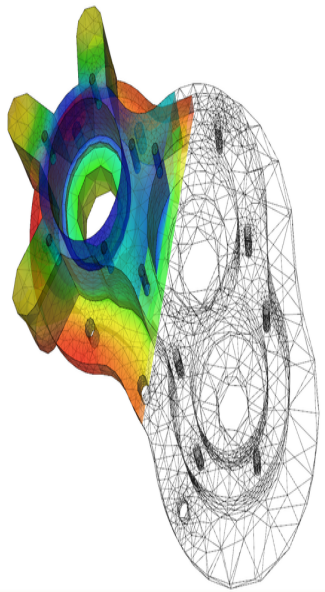
Finite Element Method

(m^3/50 years)

Observe:
Macroscopic behavior of complex structures

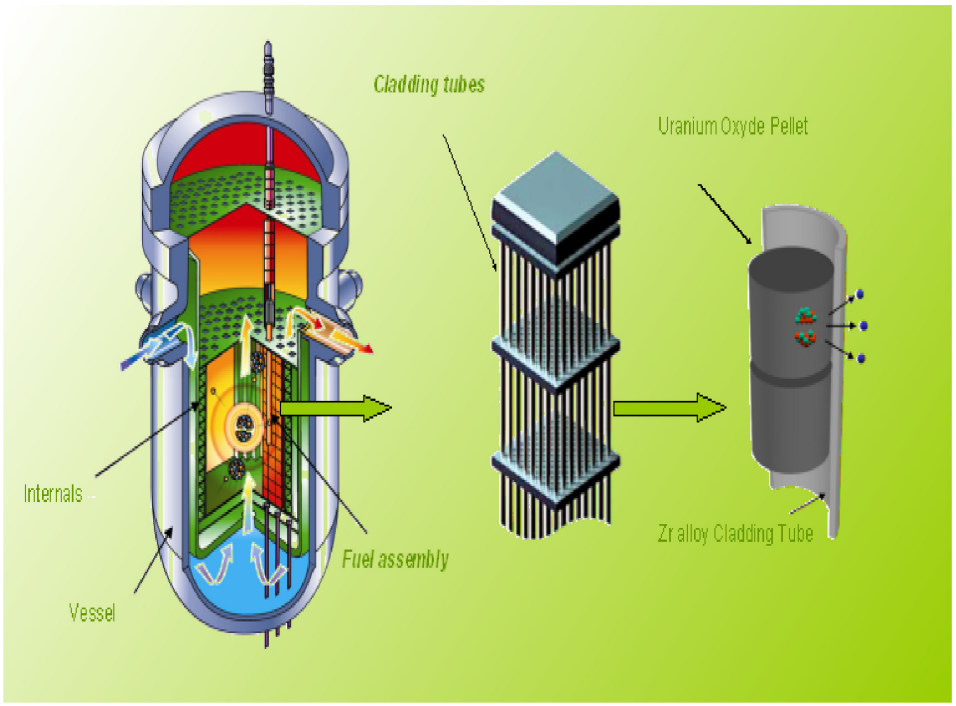


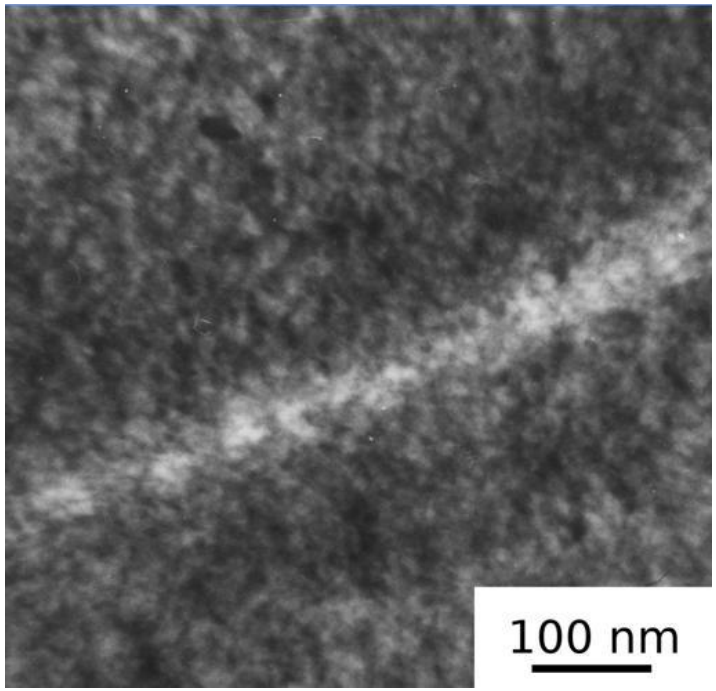




Simulation context

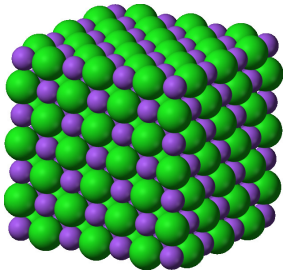
Study plastic deformation in crystal





Study dislocation channeling mechanism in neutron irradiated zirconium alloys

Dislocations caracterization

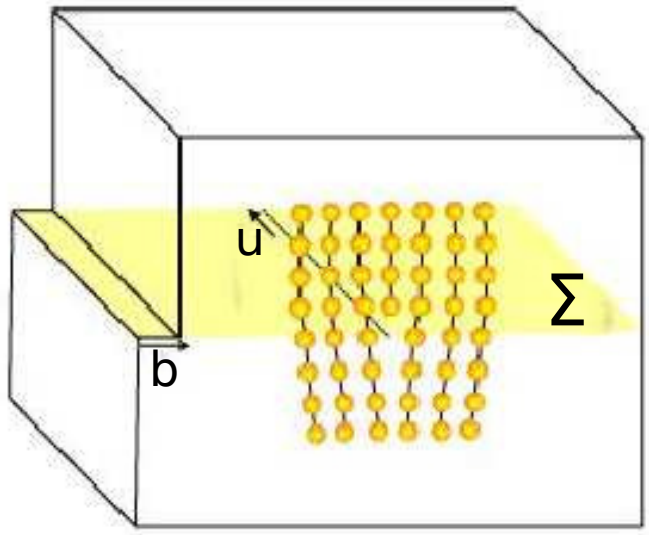


Definition

A Dislocation is a linear crystallographic defect, or irregularity, within a crystal structure

Dislocation motion is controlled by:

- **burgers vector** $\overrightarrow{b}$
- **constraining planes** Σ
- **mobility laws**



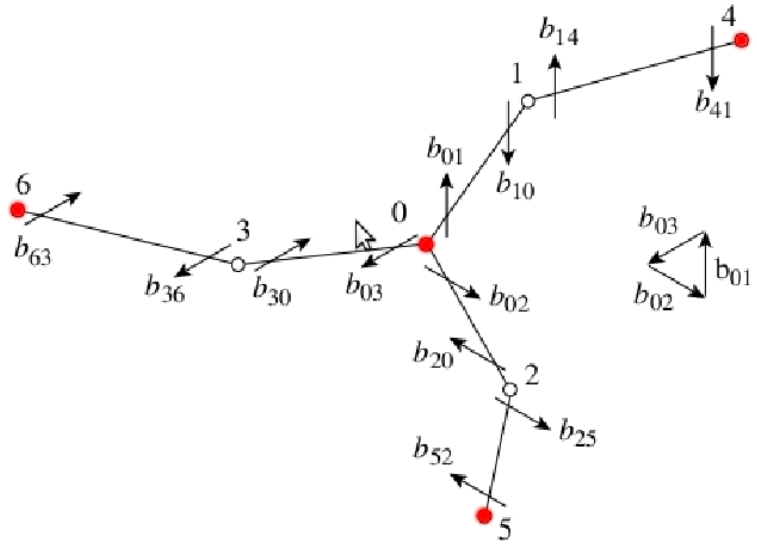
Dislocations discretization: Nodal simulation

Mesh: **1D mesh in 3D space**

- Physical nodes
- Discretization nodes

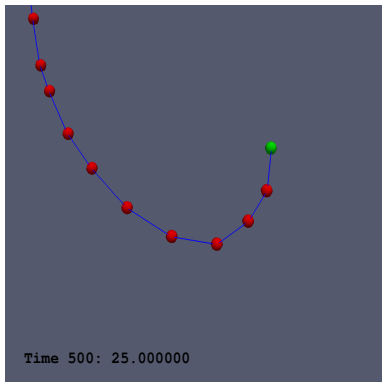
Inter-connected straight segments:

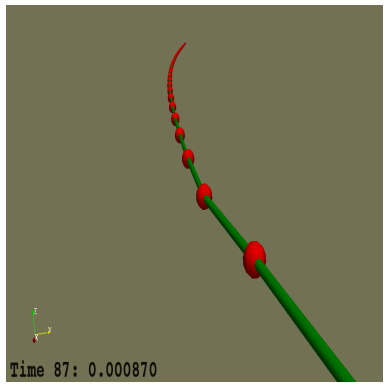
- Network $\rightarrow$ 1D Finite Element
- Unknown linear approximation



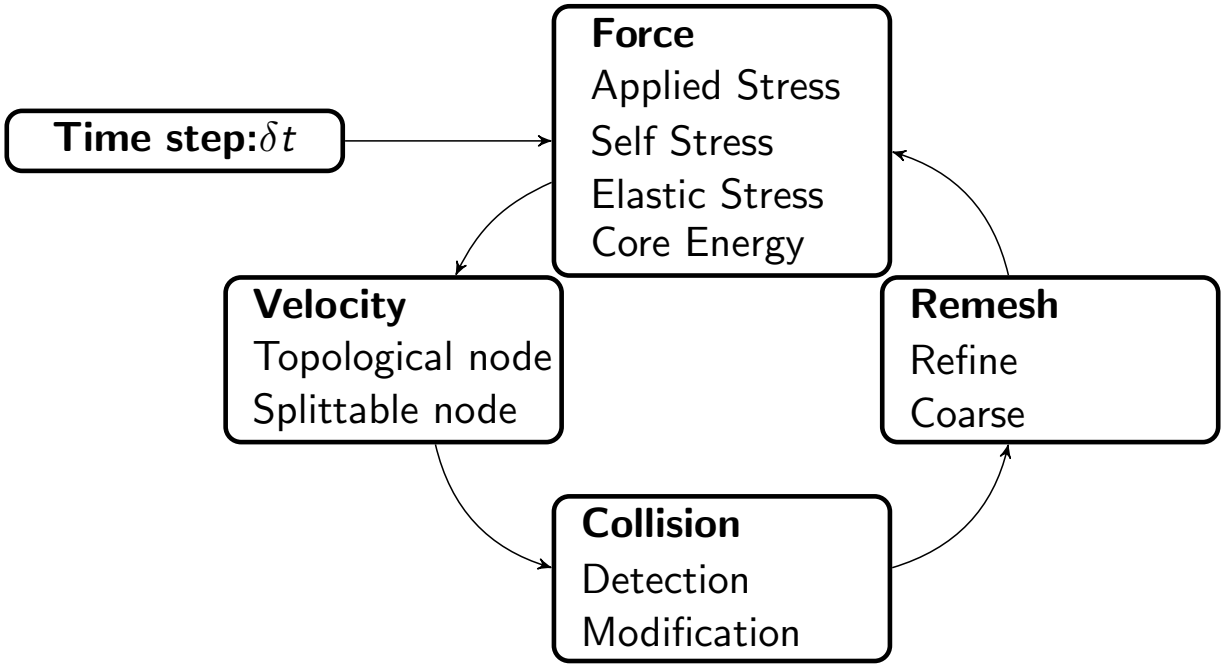
Classical simulation compromise

- fine discretization $\Leftrightarrow$ better curvature approximation
- minimize degrees of freedom $\Leftrightarrow$ fast computation





Time Loop in DD simulation



Features:

- Interaction between segments $\simeq$ n-body problem
- Highly dynamic system

Force computation:

Stress acting on segments:

$$f_i^{nodale} = \oint (F^{PK}(x) N_i(x)) dL$$

Peach Koehler Force: $F^{PK}(x) = (\sigma(x).\vec{b}) \wedge \xi$
Stress can be decomposed as: $\sigma = \sigma_{applied} + \sigma_{self} + \sigma_{elast}$

Elastic stress is a long-range stress field: All-Pairs N-Body problem

Different Methods:

- Cut Off (Box method)
- Fast Multipole ($= \Theta(N)$) work done by **Pierre Blanchard**
  1. Chebyshev formulation
  2. Lagrange formulation accelerated by FFT

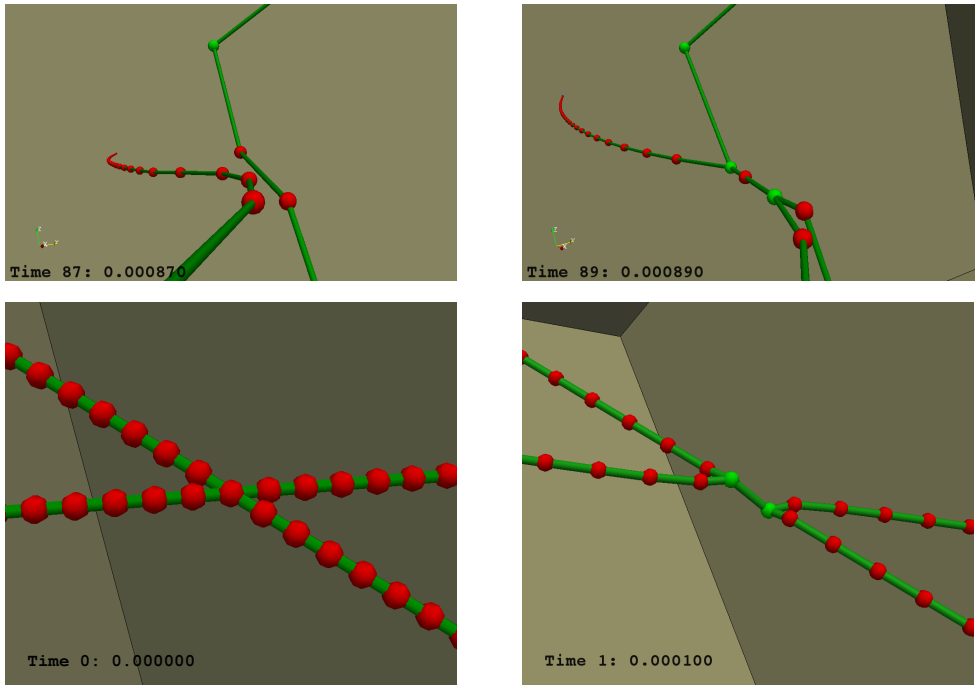
FMM library ScalFMM (*http://scalfmm-public.gforge.inria.fr*)

Topology modification

Three algorithmic steps

- *Split Multi-Arms*
- *Collision detection*
- *Mesh refinement*

Physical accuracy with MD simulation
(*http://optidis.gforge.inria.fr/*)



Highly Dynamic simulation

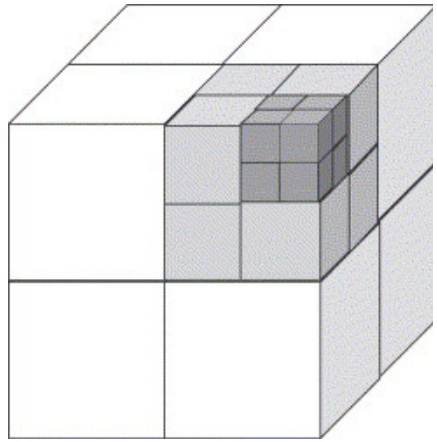
Each of these steps modify the data structures

Outline

Data structures

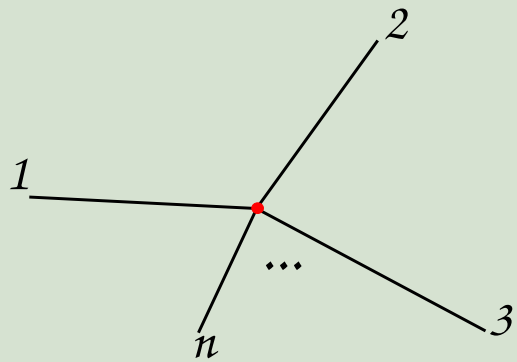
Two different data structures needed:

- Mesh data structure
- Spatial decomposition structure



Data inter-dependencies

- One node is referenced by [1,N] segments
- On segment is referenced by:
  1. Its two extremities
  2. The container for spatial locality



Challenges

Dynamic Side

- Control allocation/deallocation
- Ease insert/delete data
- Keep locality(cache efficiency) while simulation iterate

Memory Cost

- Limit data copies (redundancy)
- Limit data inter-dependency

Computation efficiency

- Fast data access
- Easy to parallelize

Classical mesh representation

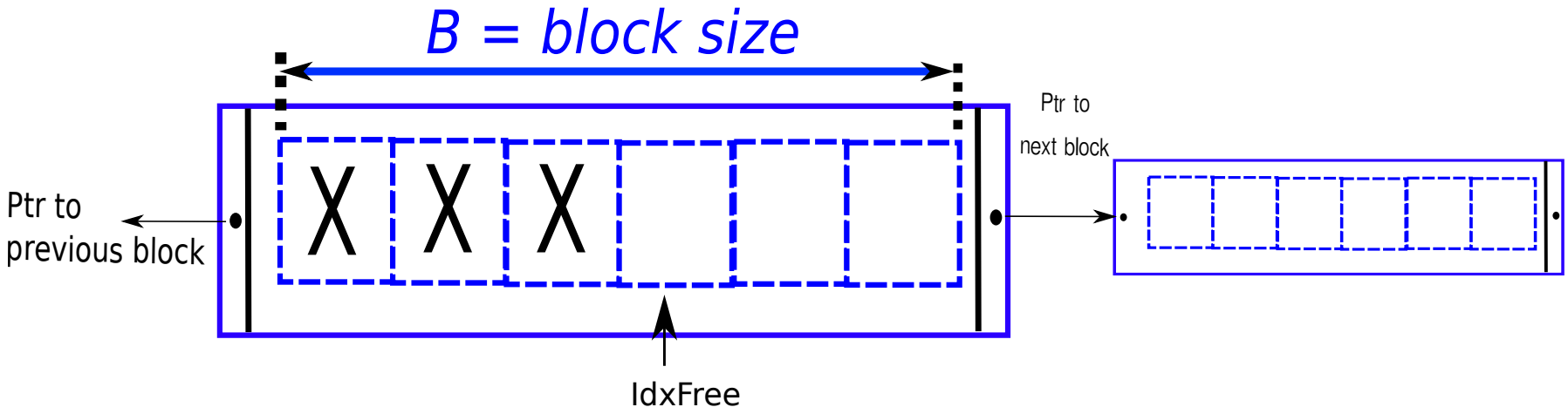
Linked list:

- Highly dynamic (insert/delete)
- Cache inefficient
- Inefficient to process in parallel
  - no easy/efficient with multi-threads
  - no easy pack/unpack communications

Extensible array:

- Cache efficient
- Easy to parallelize (*pragma loop*)
- Not easy to handle as dynamic structure
  $\Longrightarrow$Hole list and reallocation

Adaptive cache oblivious list

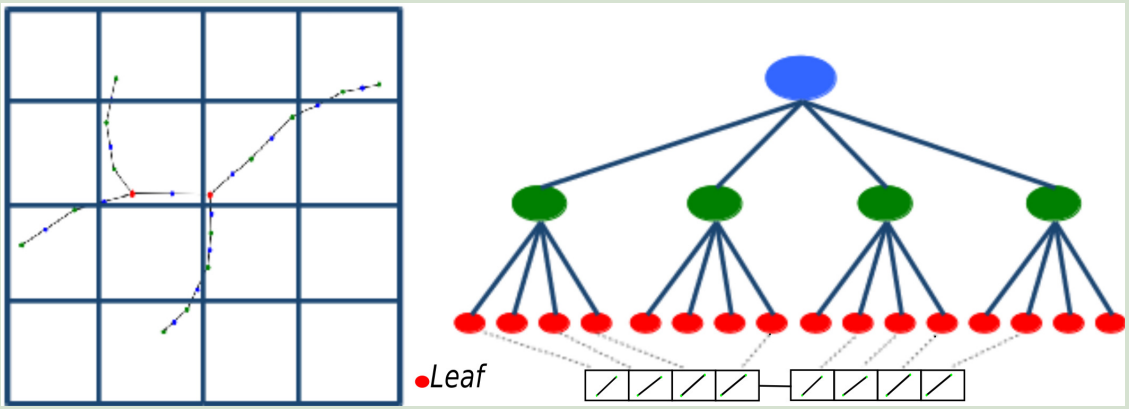


Mesh container: **Adaptive cache oblivious list of array (LOA)**

- Allocate/Dealocate by pool
- Expand/Shrink easily by block
- Keep data contiguous by block
- Cache efficiency with neighbor data access
- Parallel efficiency (lock-free) over blocks

Fit one data structure for each phase

Data structure in use



- List of Array **LOA** for mesh
- Spatial container:**Octree**

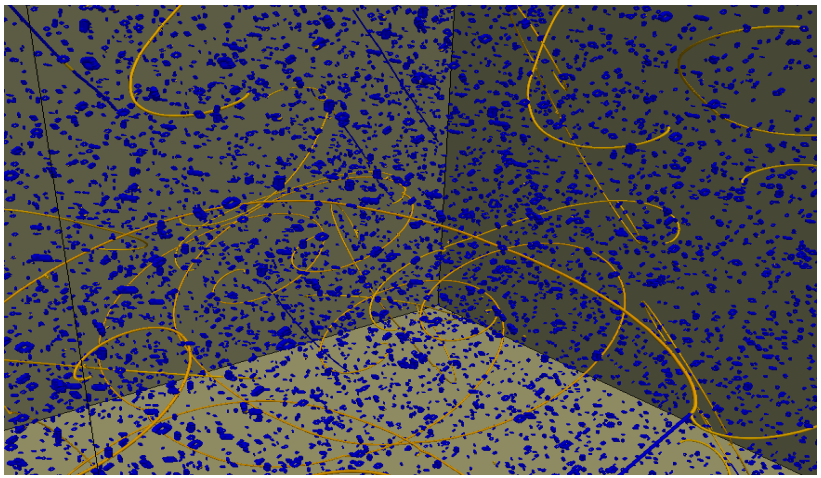
Maintain simple relation

Maintain coherency (while Moving/Deleting/Inserting) between spatial locality and memory locality

Hybrid Parallelsim

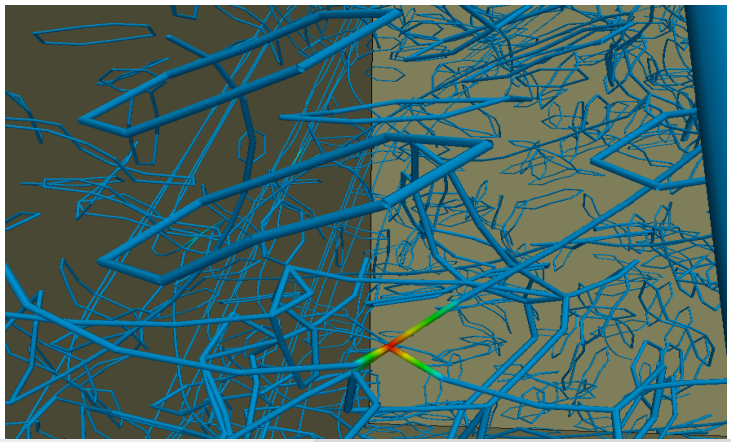
Aim to big simulation with $10^6 dof$

- Shared memory parallelism
- Message Passing Interface



Main computational **hot points**

- Force Computation
- Collision detection



Hot points: Spatial decomposition techniques

Long range elastic field: Fast multipole method

Computation based on spatial decomposition:

$$FMM = Far_{field} + Near_{field}$$

ScalFMM library

- De-Correlate parallelism and computation kernel
- Efficient hybrid parallelism

Collision detection $\Leftrightarrow$ FMM $Near_{field}$ interaction

Collision detection

- proximity collision with direct neighbors (27 boxes)
- plug collision detection kernel in ScalFMM

Task parallelism: OpenMP

Specific focus with data inter-dependency on:

- Race conditions
- Deadlock

Mechanism to maintain network and data structure coherency

Thread access data concurrently

- Spread tasks over blocks in the **LOA**
- Lock-free mechanism in topological steps
- Coloring algorithm in octree
- Scheduling thread algorithm for fine granularity

Distributed memory parallelism

Segments move from one box to another

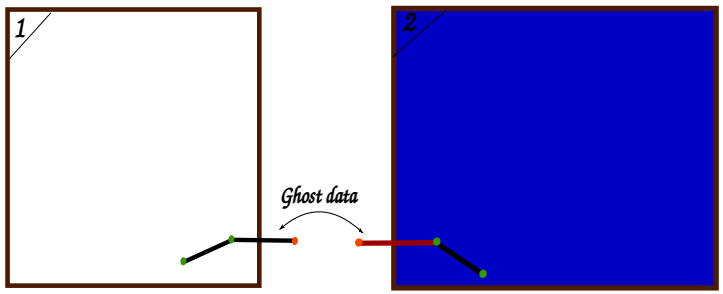
Main distribution idea

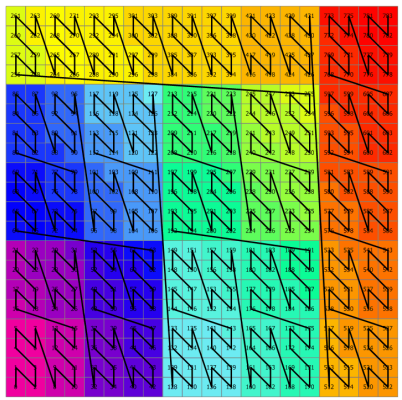
Space domain decomposition follow Morton space filling curve

Shared segments problem:

- Introduces the notion of ghost segments

Segment attributed in each domain by barycenter





Distributed parallelism

Main focus on segment migration

Migration

Migration ends each time step

- Send segments that leave the local domain
- Receive segments that enter the local domain
- Update the Octree for next time step

Data structure consequences

Need to handle properly movement (insertion/deletion) in our **LOA**

Work Load imbalance

- Inter nodes: Morton intervals load balance
- Intra node: thread level load balancing algorithm

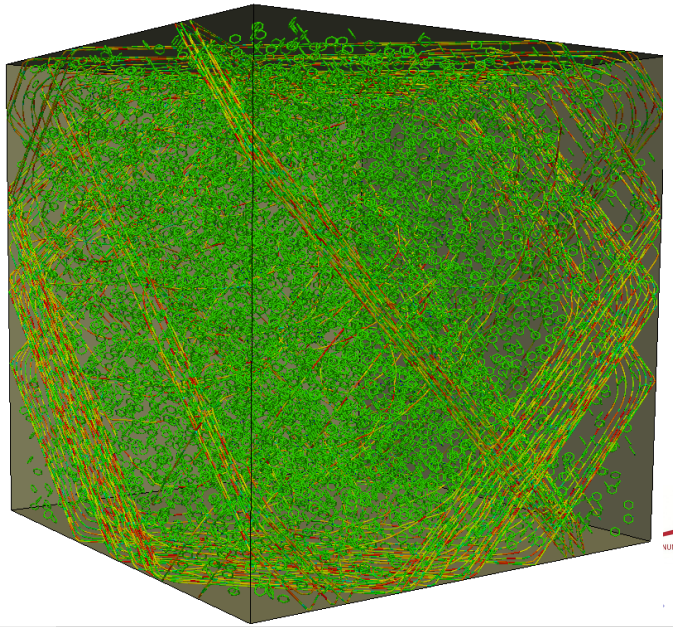
Experimental setup

Hardware/Software

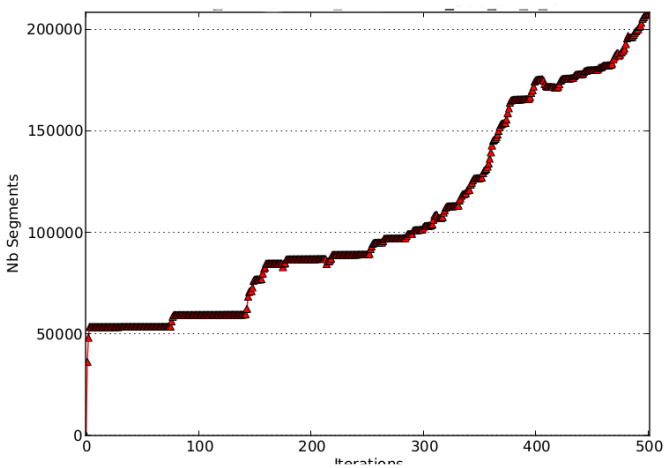
- **8** nodes Intel(R) Xeon(R) E5-2670 at 2.5 GHz with 2 sockets of 10 cores and 64 GB of shared memory
- Gcc 4.9
- Open-MPI 1.8.3
- OpenMP 3.1

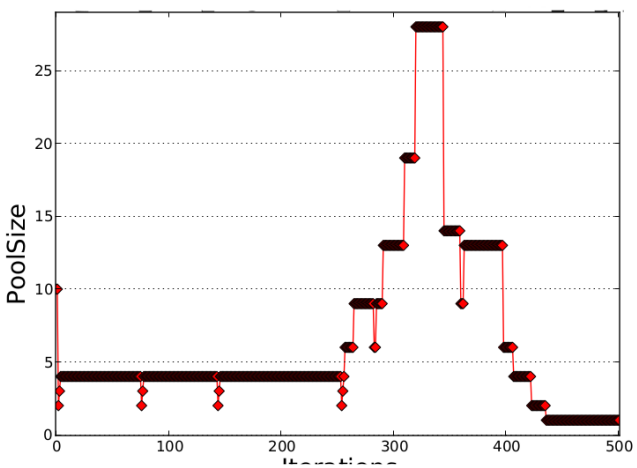
Simulation

- Density: 6×10^{21} defects.m^{-3}
- from 200 000 to 520 000 segments
- full FMM calculation



Simulation behavior





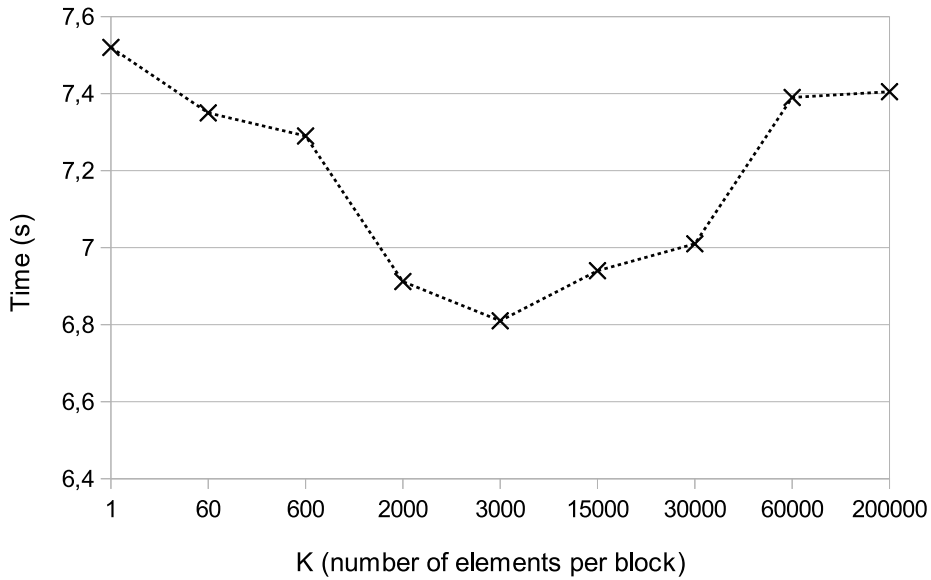
Different phases

- Static at start
- Dynamic after activation of FR sources

Dynamism $\Rightarrow$ Not the same amount of work

Measure average time over many time steps

Influence of block size on force computation



Benchmark from usual list to classical array; to assess **LOA** benefits

- **LOA reduces indirection**
- **LOA keeps spatial/memory locality coherency**

Shared memory parallelism benchmark

- Benchmark use OpenMP only
- Use **LOA** with block size fixed (3 000)
- Average load of 50 000 segments

Threads	Time (in s)	Speedup	Efficiency (in %)
1	22.15	1	100
2	11.17	1.98	99.1
4	5.77	3.83	95.9
8	3.06	7.23	90.4
16	1.73	12.80	80.0
20	1.48	14.97	74.8

Maintain 74.8% **efficiency on** 20 **cores**

Hybrid parallelism benchmark

- Using 1 node with 20 cores
- Benchmark hybrid parallelism MPI/OpenMP maintaining 20 compute units
- Average load of 50 000 segments

MPI $\times$ Thread	Time (in s)
20×1	1.95
10×2	1.85
5×4	2.12
4×5	1.86
2×10	1.81
1×20	**1.79**

Taking advantage of shared memory

Multi nodes hybrid parallelism benchmark

- Benefit from shared memory: use 20 threads per nodes
- Larger test case with about 750 000 segments

Resources	Time (in s)	Speedup	Efficiency (%)
1 node (20 cores)	27.8	1.0	100
2 nodes (40 cores)	15.4	1.80	90.2
3 nodes (60 cores)	12.1	2.29	76.5
4 nodes (80 cores)	9.1	3.05	76.3
5 nodes (100 cores)	7.09	3.96	75.9
6 nodes (120 cores)	6.11	4.53	75.5
7 nodes (140 cores)	5.91	4.71	67.2
8 nodes (160 cores)	4.98	5.58	69.7

SpeedUp of 111 on hybrid implementation on 8 nodes with 160 cores

Conclusion

Results

- Efficient adaptive data structure for dynamic problems
- Efficient hybrid parallelism implementation
- Simulation stable on multi-core cluster with hybrid parallelism
- Run large scale benchmark

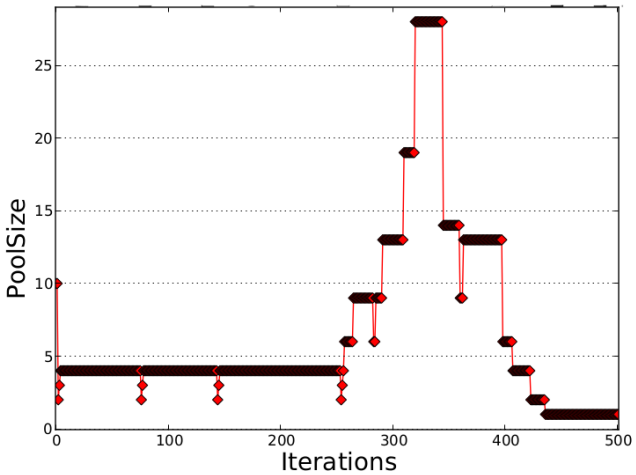
Perspectives and ongoing

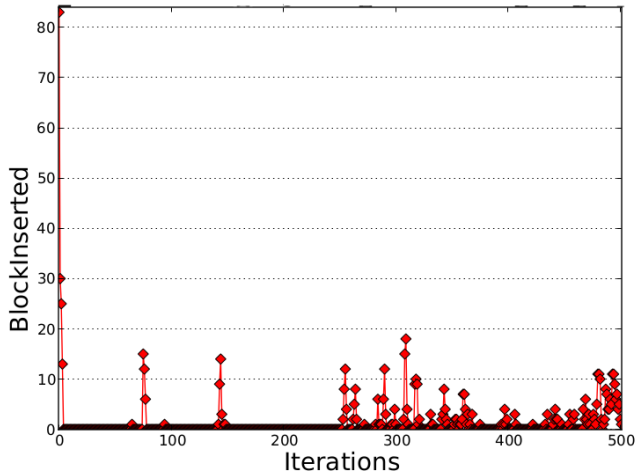
- Implement solutions (runtime systems) to take advantage of heterogeneous nodes (GPU,Mic)
- Run bigger benchmark on larger cluster for further performance investigations

Thank you for your attention,
Any questions?

This work was supported by the French ANR grants ANR-10-COSI-0011

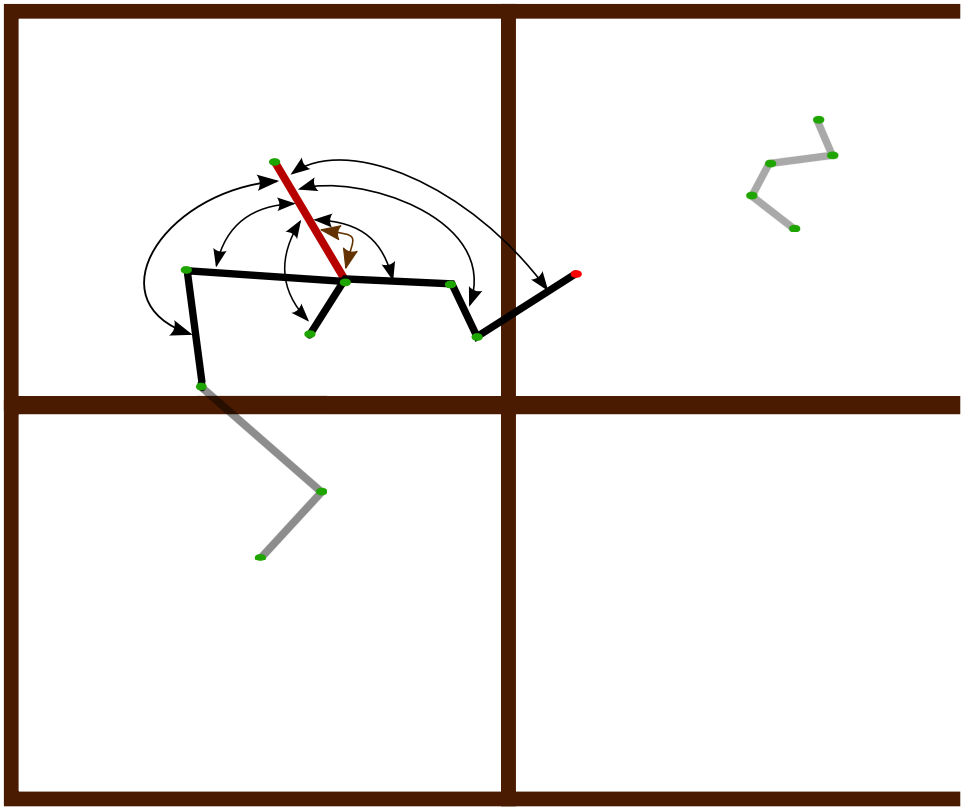
Appendix


PoolSize
Iterations
0
5
10
15
20
25
0
100
200
300
400
500

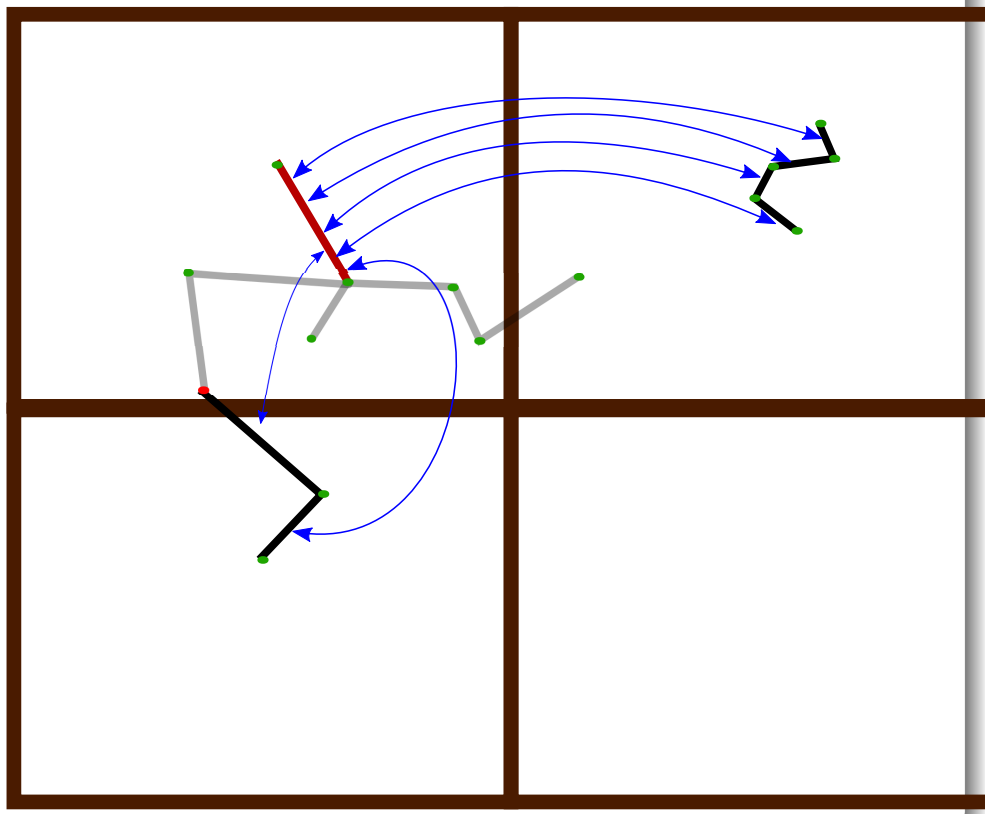

BlockInserted
Iterations
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500

OpenMP

Force interne



Force externe



retour retour Force

Split

N bras

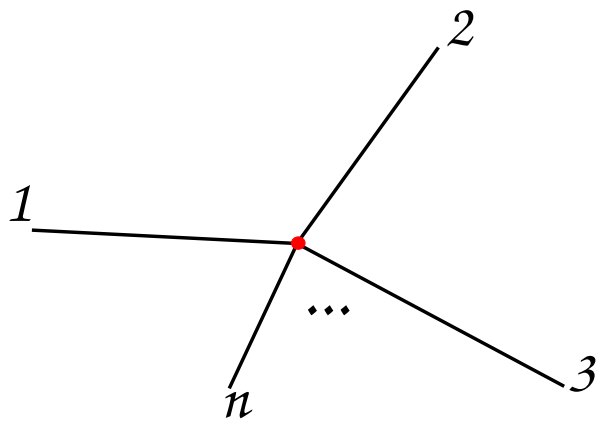


Figure : Nœud à n bras (potentiellement splitable)

retour

Segments voisins

Référencement des segments

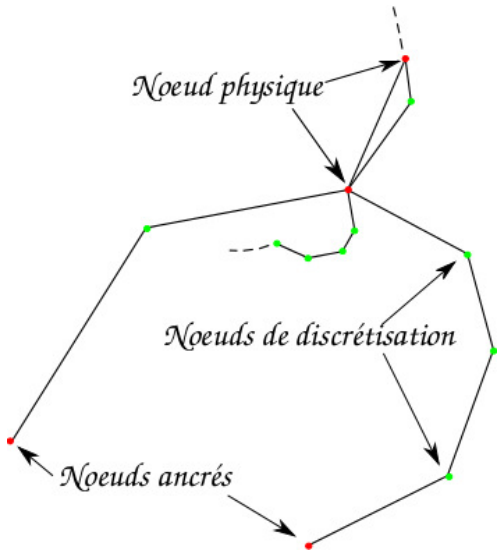
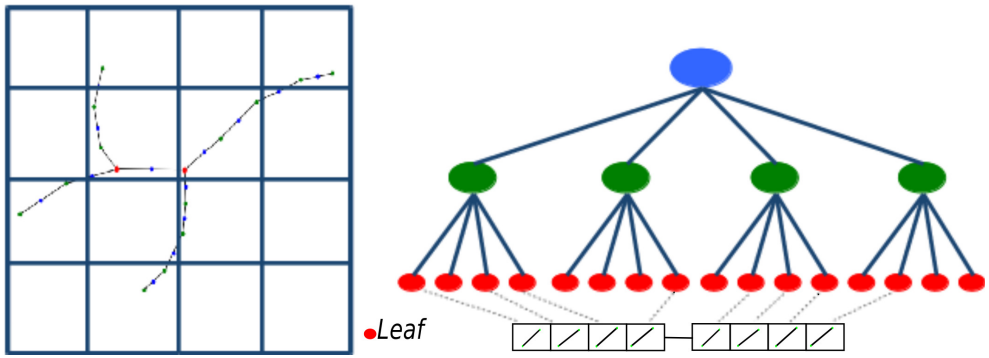


Figure : Diverses situations pour les voisins

retour

Spatial locality

Référencement des segments



retour

La liste

Struct BlockContainer

Attributs:

- BlockNodes nodes
- BlockNodesContainer* Next
- BlockNodesContainer* Previous

La liste

Class BlockNodes

Attributs:

- Node []
- int capacity
- int lasEmptyPosition

La liste

Class BlockNodes

Méthodes privées:

- void copyHalfIn(BlockNodes& block);
  ¡! Copy une partie d'un bloc vers un bloc voisin
- void shiftTo(int position); ¡! Copy le dernier nœud vers la position donnée

La liste

Class BlockNodes

Méthodes publiques:

- int push();
- int getLastIndexFilled() const;

Shared memory parallelism

OpenMP

- loops and sections parallelism
- Task parallelism (*No tasks dependencies and priorities*)

```
function IterateListTask()
    #pragma omp parallel
        #pragma omp single
            while blocks=  IteratorList.hasnextBlock() do
                #pragma omp task (firstprivate(block))
                    forall the data in block do
                        doTreatment();
```