Fast Multipole Method on the Cell Broadband Engine: the Near Field Part

Pierre FORTIN, Jean-Luc LAMOTTE
LIP6 - Université Pierre et Marie Curie

Séminaire Performance et Généricité du LRDE

02 décembre 2009 - EPITA

Outline

Outline

The Cell Broadband Engine

Roadrunner

- June 2008: Petaflop barrier broken by the IBM Roadrunner computer
- 12240 Cells + 6120 Dual-core Opterons (2008)
- Cells $\Rightarrow$ over 96% of the 1.3 Pflop/s theoretical peak performance

The Cell Broadband Engine

- 1 general-purpose PowerPC core (PPE)
- 8 Synergistic Processing Elements (SPEs)
  - specialized for high performance computing,
  - independant fast local store (LS)
  - explicit direct memory access (DMA): LS $\leftrightarrow$ Cell main memory
- 3 levels of parallelism:
  - MPI multi-process parallelism
  - multi-thread parallelism among the 8 SPEs
  - SIMD (Single Instruction on Multiple Data) parallelism $\rightarrow$ SPE vector units

Specific architecture

- Suitable for all applications and algorithms? (same question for GPUs...)

N-body problem

In *The Landscape of Parallel Computing Research: A View from Berkeley* (Asanovic et al., 2006):

13 *dwarfs* (kernels) $\longrightarrow$ including the N-body problem

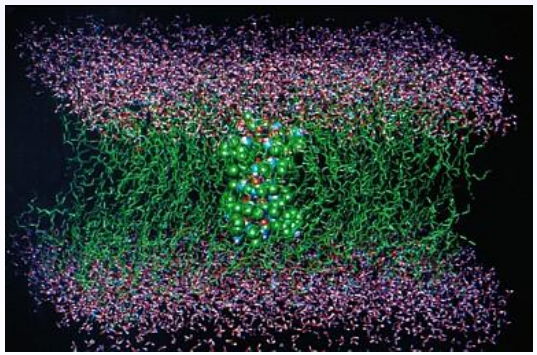
- ★ Pairwise interactions among N bodies (molecular dynamics, astrophysics. . .)

- ★ Direct computation between the $N(N-1)$ pairs $\Rightarrow$ quadratic complexity

$$\sum_{i=1}^{N} \sum_{j \neq i} \frac{q_i q_j}{|z_i - z_j|}$$

- ★ Mutual interaction principle

$$\mathcal{F}_{A \to B} = -\mathcal{F}_{B \to A} \quad \Rightarrow \quad \sum_{i=1}^{N} \sum_{j<i} \frac{q_i q_j}{|z_i - z_j|}$$



Current N-body simulations on the Cell B.E.

Cut-off radius methods

$$\Phi = \Phi_{\text{near}} \qquad \text{since} \qquad \lim_{r \to +\infty} \Phi(r) = \lim_{r \to +\infty} \left(\frac{q}{r} \right) = 0$$

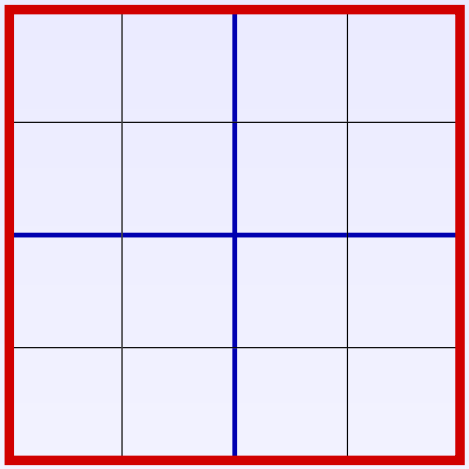
- ★ Computation only with neighboring particles within cut-off radius

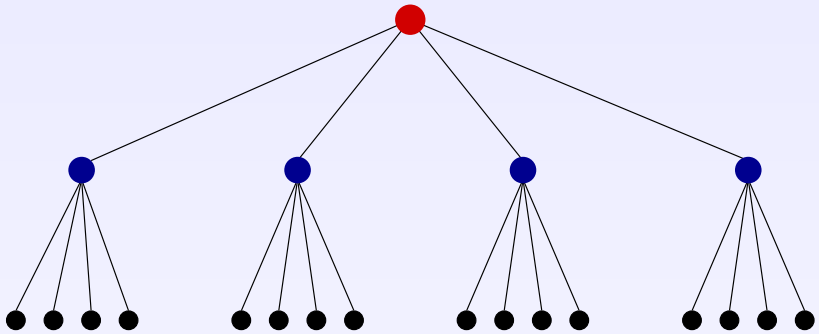
Current performance on 1 Cell B.E.

- ★ Cut-off radius method:
  - De Fabritiis, 2007 — 45 Gflop/s
  - Luttmann *et al.*, 2009 — 60 Gflop/s (for 6 SPEs)
  - Swaminarayan *et al.*, 2008 — 34 Gflop/s (double prec. on PowerXCell8i)
- ★ Full direct computation:
  - Knight *et al.*, 2007 — 83 Gflop/s

Hierarchical methods for N-body problems

- Hierarchical space decomposition with an octree





- Potential decomposition

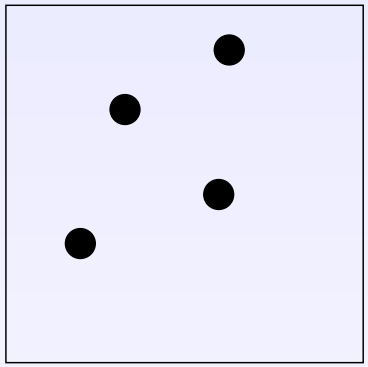
$$\Phi = \Phi_{\text{near}} + \Phi_{\text{far}} \qquad \text{since} \qquad \lim_{r \to +\infty} \Phi(r) = \lim_{r \to +\infty} \left(\frac{q}{r} \right) = 0$$

  - near field $\rightarrow$ direct computation
  - far field $\rightarrow$ approximate computation (with expansions)

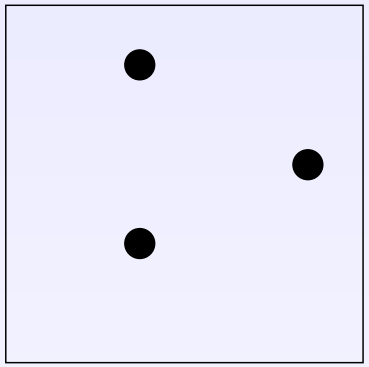
- More precise than cut-off radius methods for long-range interactions

The Fast Multipole Method (FMM) : principle

Y_j^k spherical harmonics used for potential expansions



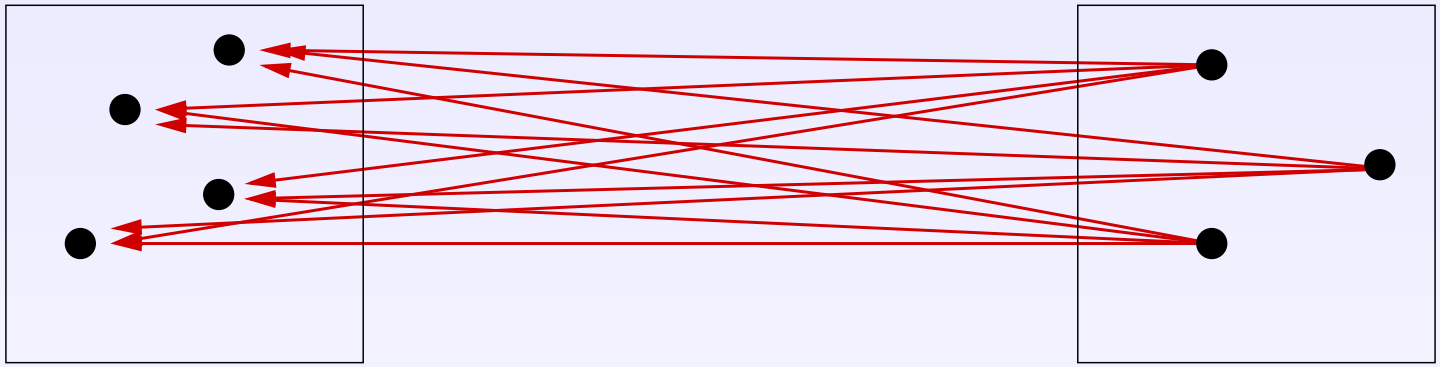
targets



sources

The Fast Multipole Method (FMM) : principle

Y_j^k spherical harmonics used for potential expansions

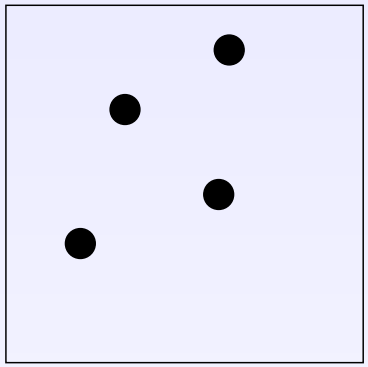


targets sources

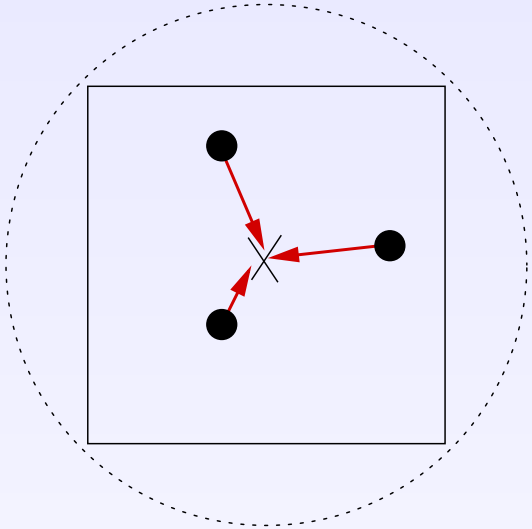
$$\mathcal{O}(N_{\text{targets}} \times N_{\text{sources}})$$

The Fast Multipole Method (FMM) : principle

Y_j^k spherical harmonics used for potential expansions



targets



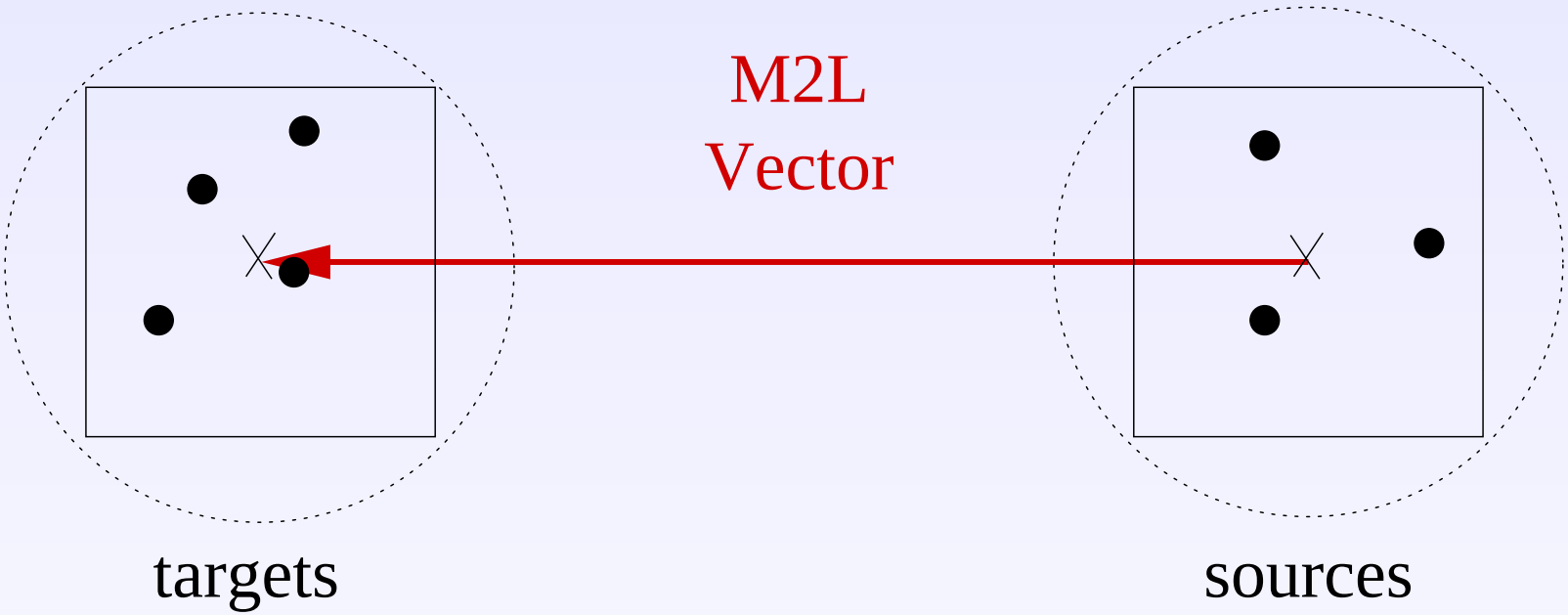
sources

M_j^k multipole exp.:

$$\Phi = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} M_j^k \frac{Y_j^k(\theta, \phi)}{r^{j+1}}$$

The Fast Multipole Method (FMM) : principle

Y_j^k spherical harmonics used for potential expansions



« well-separateness » criterion

L_j^k local exp.:

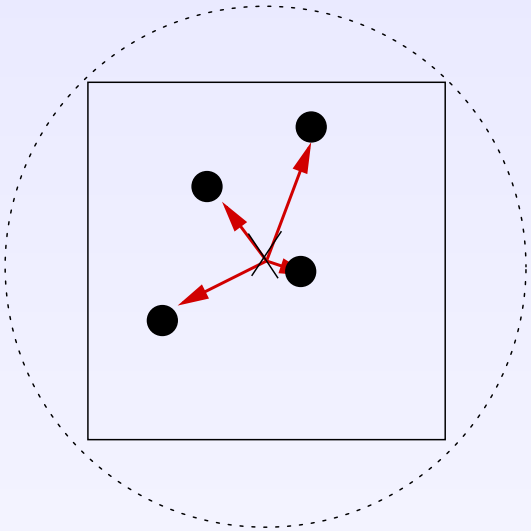
$$\Phi = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} L_j^k Y_j^k(\theta, \phi) r^j$$

M_j^k multipole exp.:

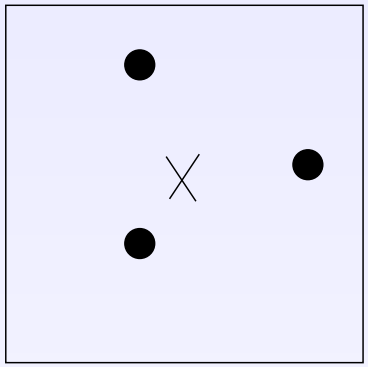
$$\Phi = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} M_j^k \frac{Y_j^k(\theta, \phi)}{r^{j+1}}$$

The Fast Multipole Method (FMM) : principle

Y_j^k spherical harmonics used for potential expansions



targets



sources

« well-separateness » criterion

L_j^k local exp.:

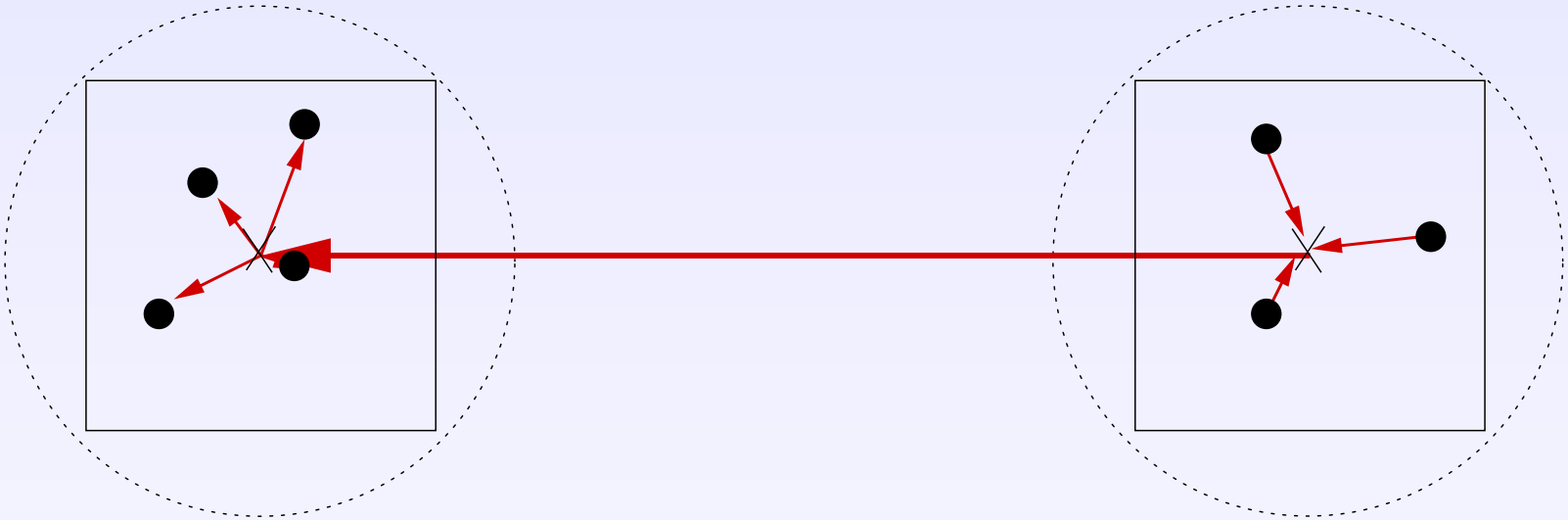
$$\Phi = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} L_j^k Y_j^k(\theta, \phi) r^j$$

M_j^k multipole exp.:

$$\Phi = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} M_j^k \frac{Y_j^k(\theta, \phi)}{r^{j+1}}$$

The Fast Multipole Method (FMM) : principle

Y_j^k spherical harmonics used for potential expansions



$$\mathcal{O}(N_{\text{targets}} + N_{\text{sources}})$$

L_j^k local exp.:

$$\Phi = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k Y_j^k(\theta, \phi) r^j$$

ε error

$\Downarrow$

P max degree

$\Downarrow$

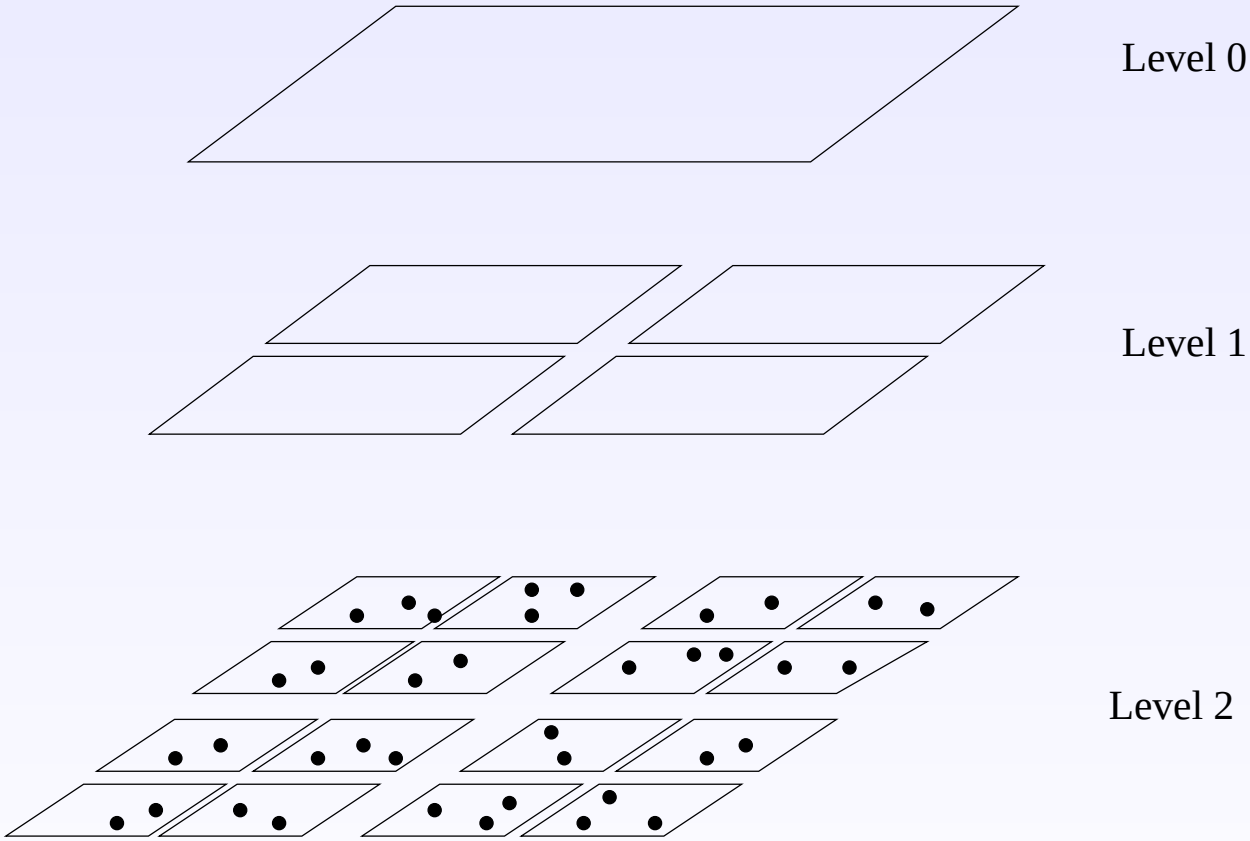
$\mathcal{O}(P^2)$ terms

M_j^k multipole exp.:

$$\Phi = \sum_{j=0}^{P} \sum_{k=-j}^{j} M_j^k \frac{Y_j^k(\theta, \phi)}{r^{j+1}}$$

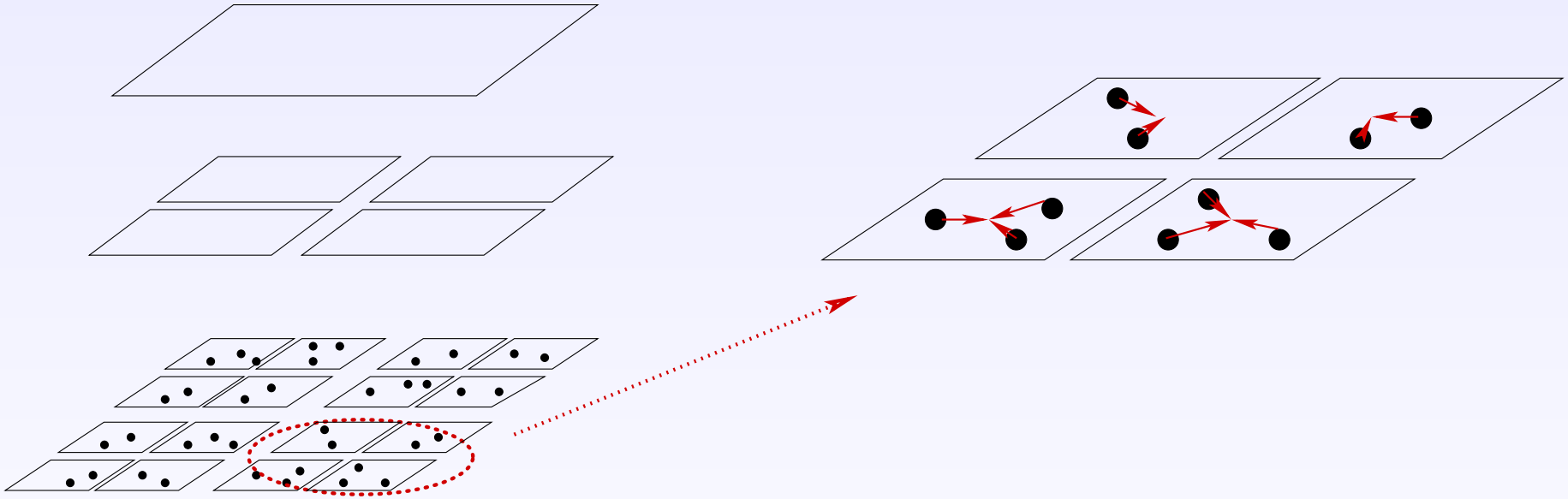
FMM principle : upward pass

Particles stored at the leaf level.



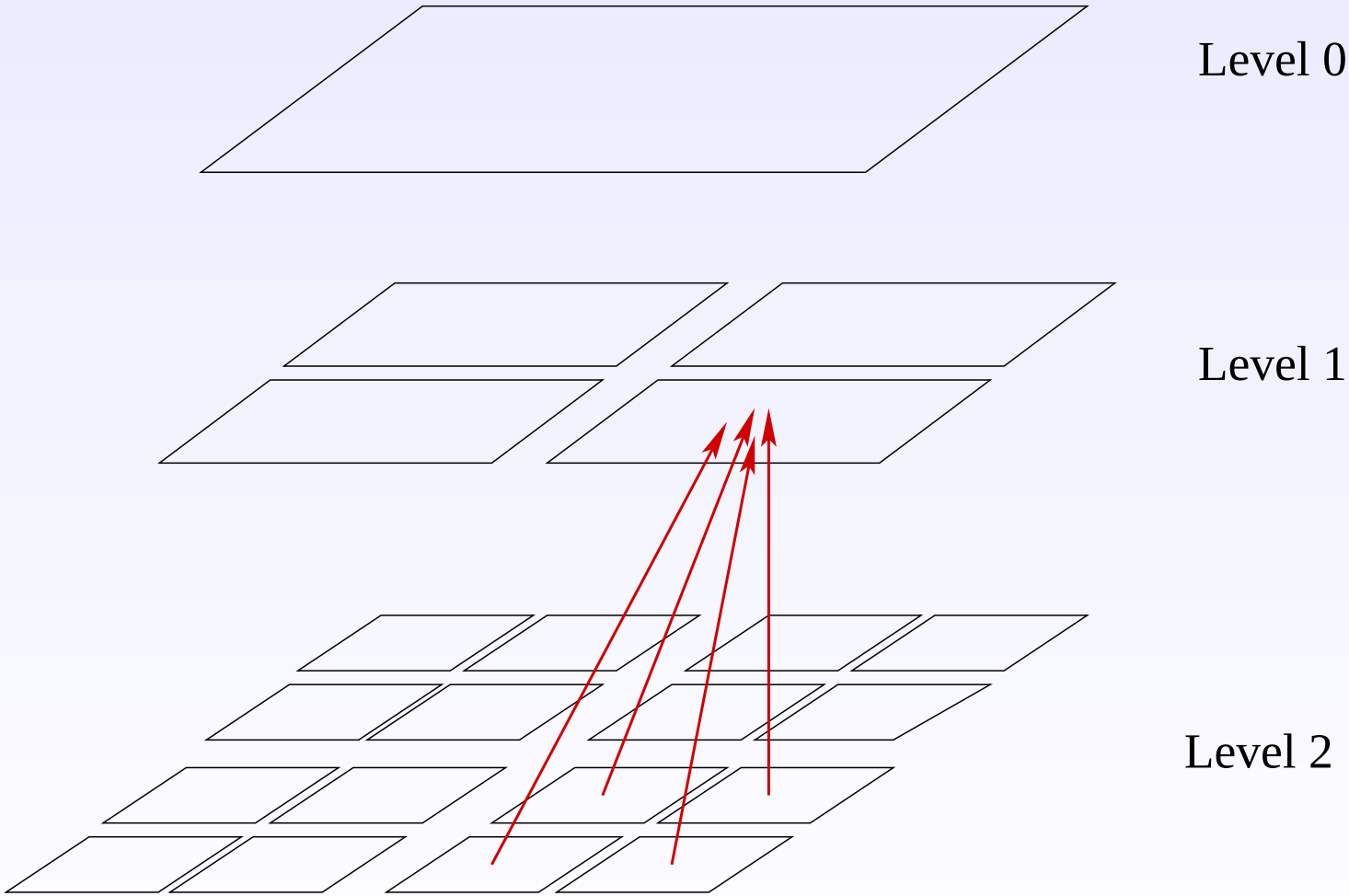
FMM principle : upward pass

Particles $\Rightarrow$ multiple exp. : ***P2M***



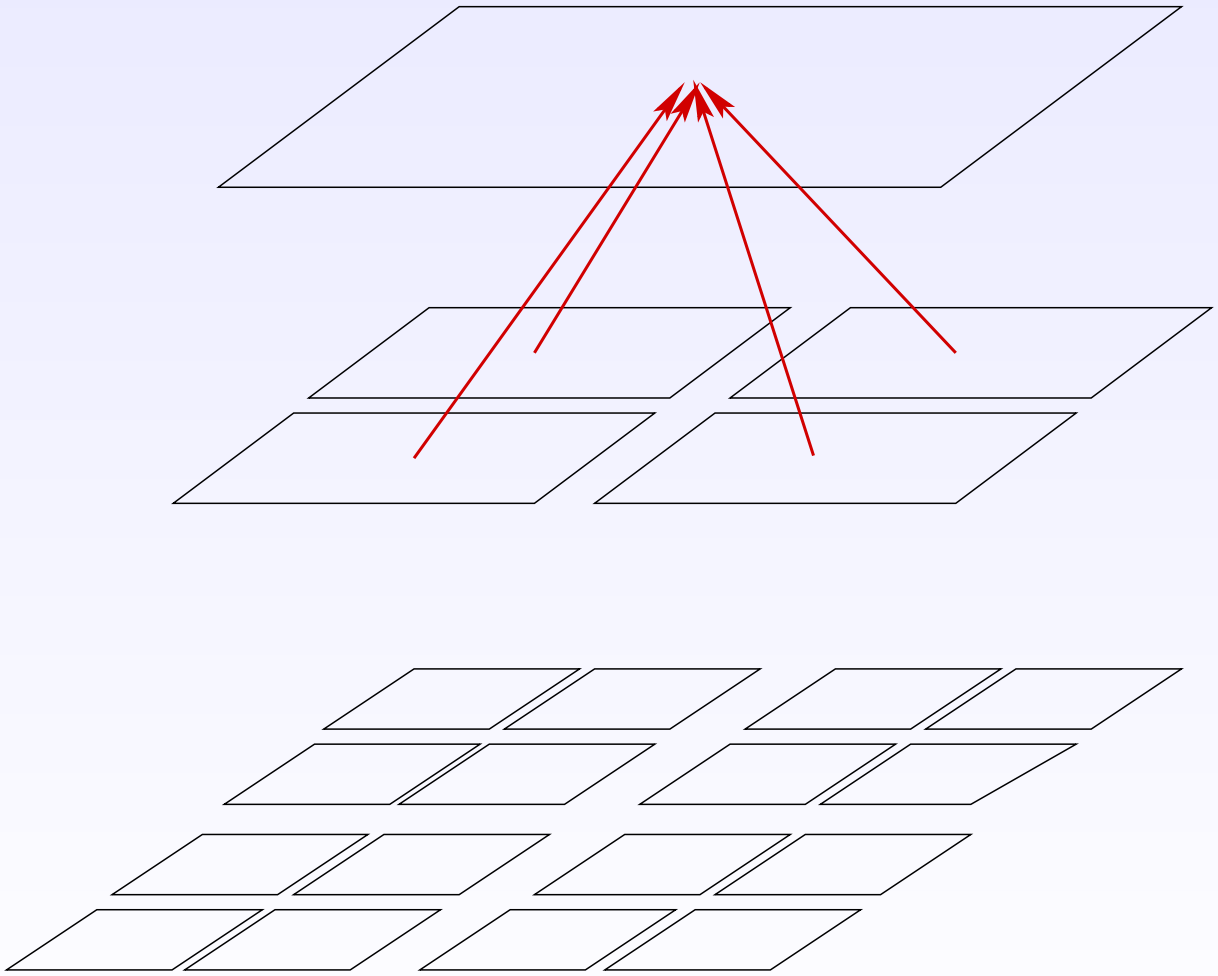
FMM principle : upward pass

Multipole exp. (child) $\Rightarrow$ multipole exp. (father) : ***M2M***



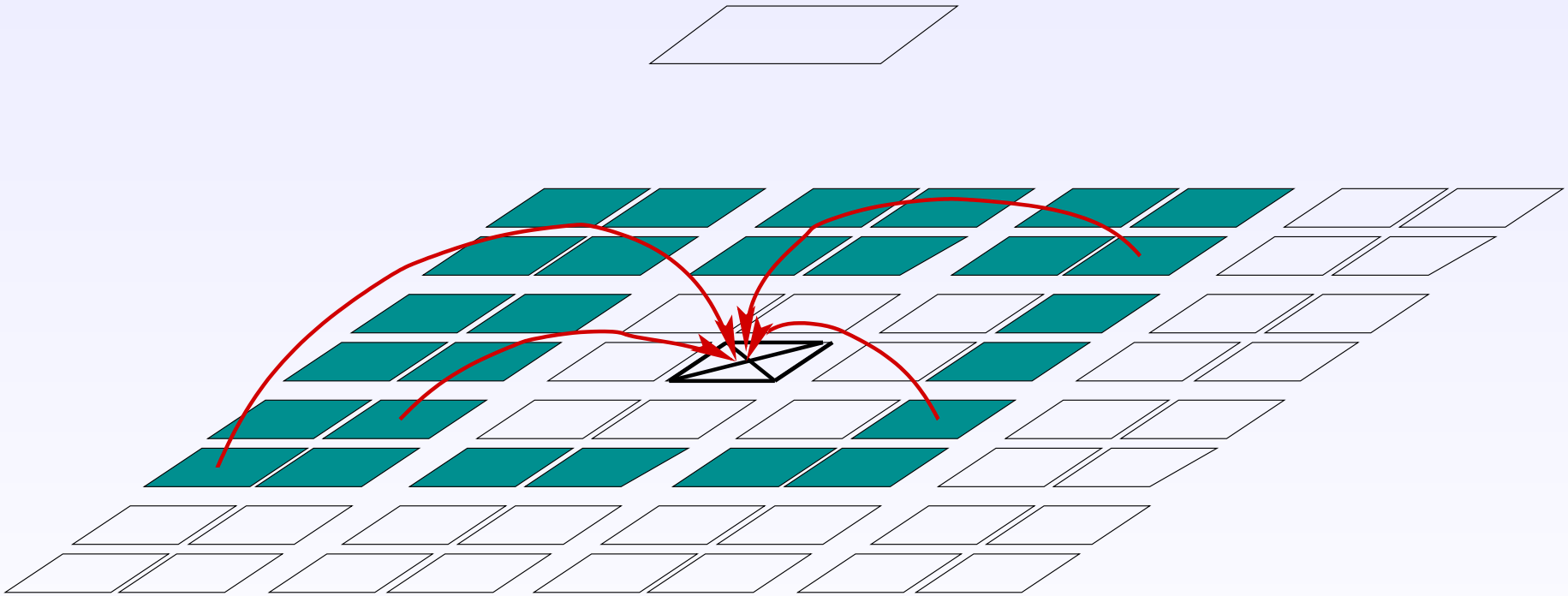
FMM principle : upward pass

Multipole exp. (child) $\Rightarrow$ multipole exp. (father) : ***M2M***



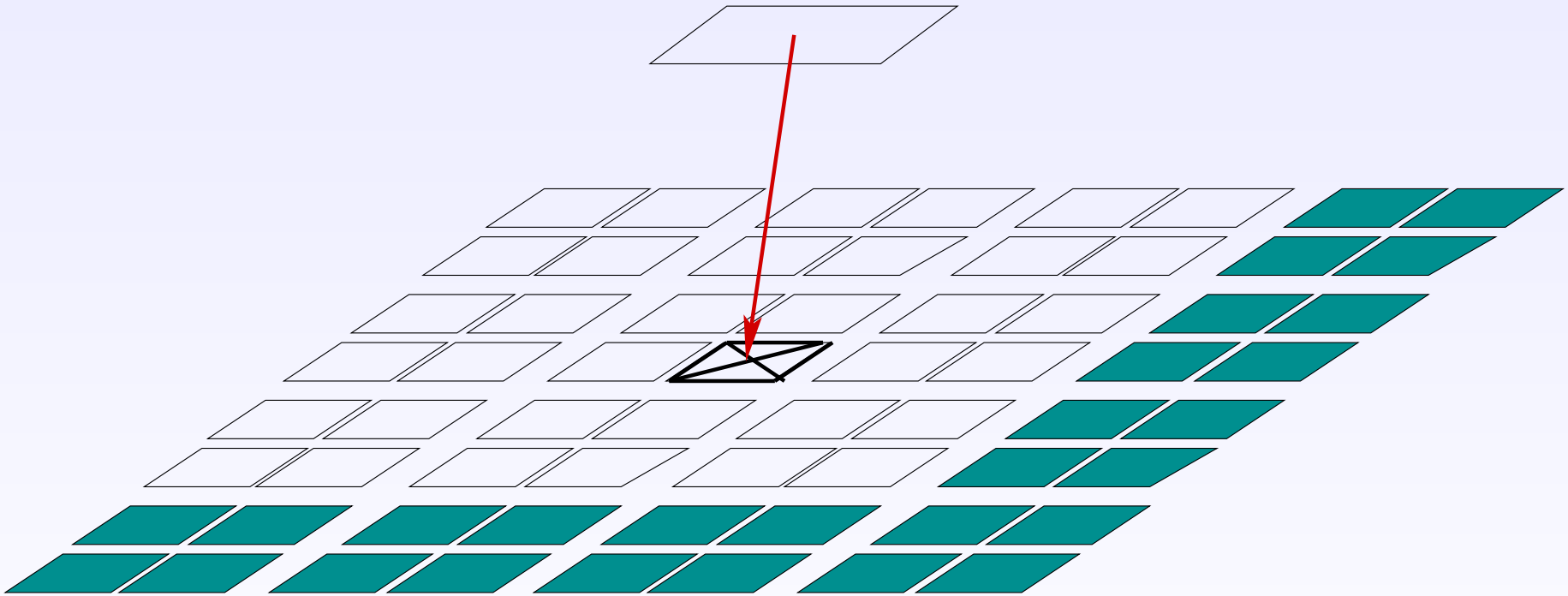
FMM principle : downward pass

Interaction list : « well-separateness », 189 members in $3D$
Multipole exp. $\Rightarrow$ local exp. : ***M2L***



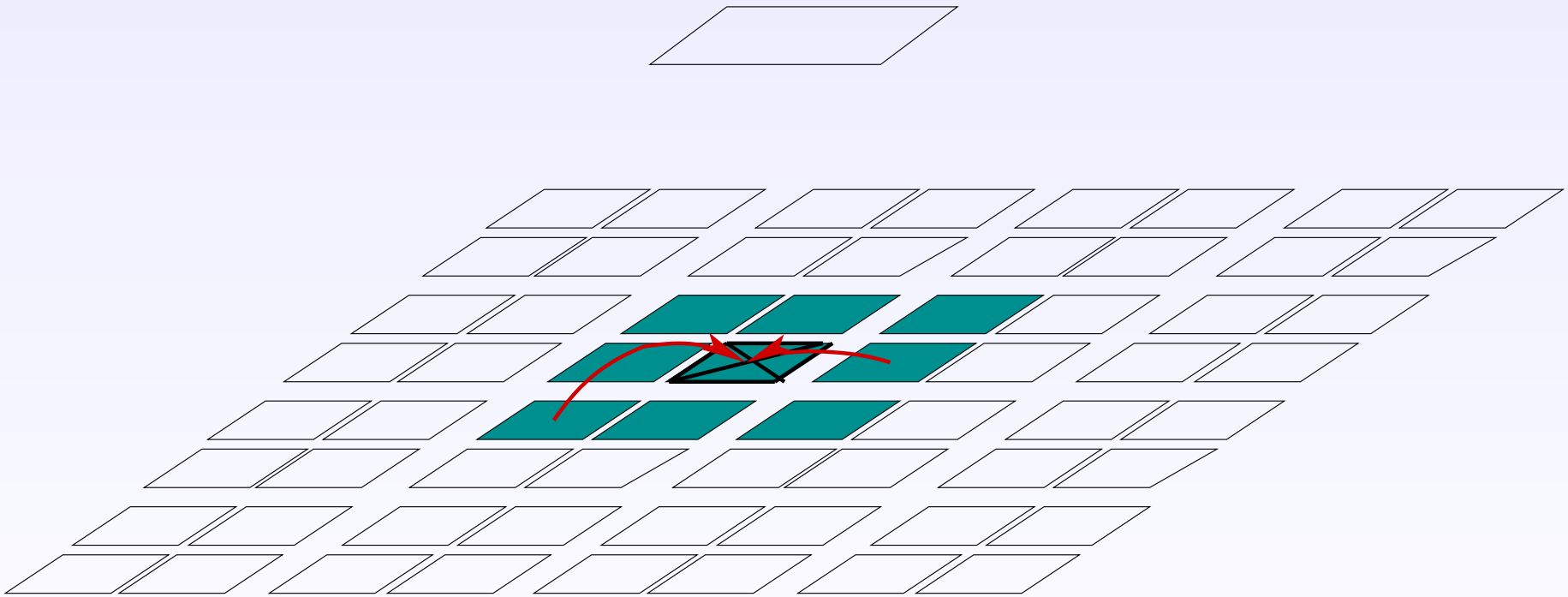
FMM principle : downward pass

Local exp. (father) $\Rightarrow$ local exp. (child) : ***L2L***



FMM principle : downward pass

At the leaf level → direct computation : ***P2P***
Direct computation list: nearest neighbors



Fast Multipole Method (FMM)

- ★ $\mathcal{O}(N)$ operation count (with optimal octree height)
- ★ Far field:
  - multipole and local expansions
  - upward pass & downward pass of the octree
- ★ Near field:
  - direct computation between 26 nearest neighbors $=$ *pair* computation
    $\longrightarrow$ 13 neighbors thanks to the mutual interaction principle
  - direct computation for all particles within each leaf $=$ *own* computation
- ★ Hybrid MPI-thread FMB (*Fast Multipole with BLAS*) parallel code:
  - efficient far-field computation with BLAS routines in the FMB code
    (*Coulaud, Fortin, Roman, Journal of Computational Physics, 2008*)

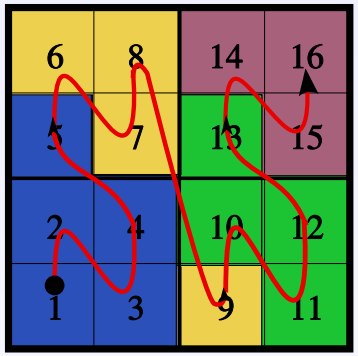
    $\Rightarrow$ direct porting on Cell!
    When optimized level 3 BLAS CGEMM/ZGEMM routines are available...

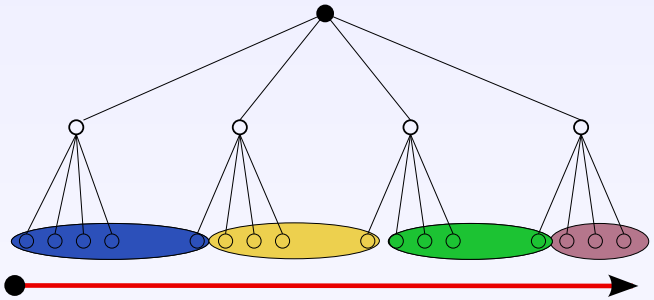
    $\Rightarrow$ **We focus here on the near-field computation.**

FMB multi-thread parallelization (*Coulaud, Fortin, Roman, ISPDC 2007*)

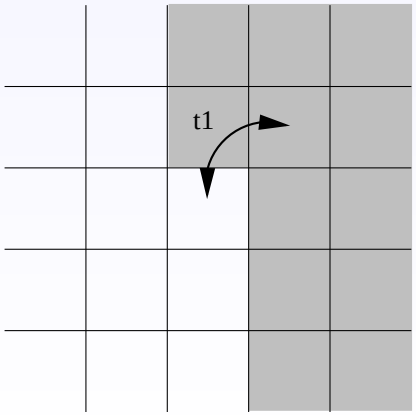
- ★ Basis for our Cell B.E. implementation
- ★ POSIX Threads in shared memory
- ★ Static octree decomposition among the threads $\Rightarrow$
  - load balancing
  - data locality

- ★ Morton decomposition:
  octree $+$
  Morton ordering $+$
  cost function
  $\Rightarrow$ 1 interval per thread





Decomposition between 4 threads



- ★ Mutual interactions: write/write conflicts

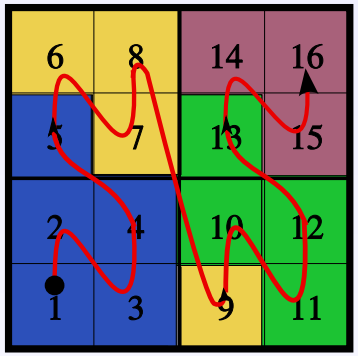
  $\Rightarrow$ mutual exclusion at each leaf
  (1 *"lock" bit* per leaf and 1 mutex per interval)

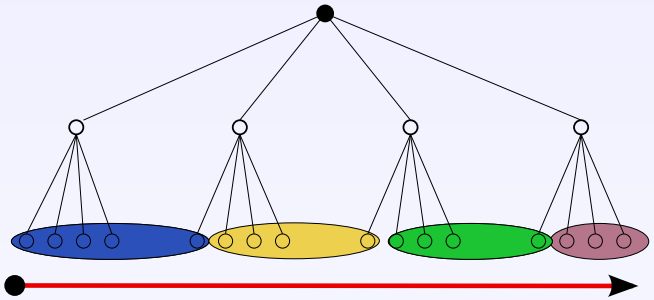
  $+$ postponed conflict resolution
  (FIFO structures)

FMB multi-thread parallelization (*Coulaud, Fortin, Roman, ISPDC 2007*)

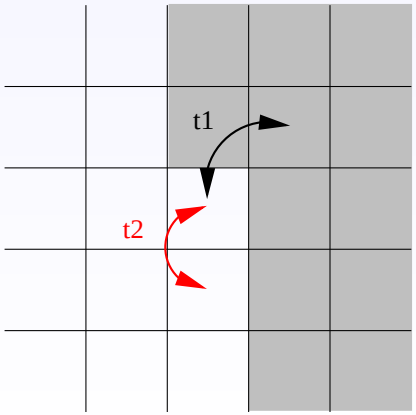
- ★ Basis for our Cell B.E. implementation
- ★ POSIX Threads in shared memory
- ★ Static octree decomposition among the threads $\Rightarrow$
  - load balancing
  - data locality

- ★ Morton decomposition:
  octree $+$
  Morton ordering $+$
  cost function
  $\Rightarrow$ 1 interval per thread





Decomposition between 4 threads



- ★ Mutual interactions: write/write conflicts

  $\Rightarrow$ mutual exclusion at each leaf
  (1 *"lock" bit* per leaf and 1 mutex per interval)

  $+$ postponed conflict resolution
  (FIFO structures)

Outline

Design of efficient computation kernels

Objectives

- ★ Force computation only (no potential computed) in single precision
- ★ Exploiting at most the mutual interaction principle

Starting point

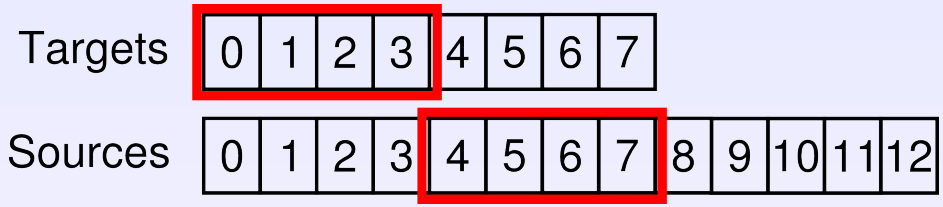
- ★ Low numbers of particles per leaf → each pair or own computation computed by only 1 SPE

SIMD code

- ★ "Structure of arrays" (SOA) data layout
- ★ Computation by blocks of 4 bodies
  → array padding with zero mass bodies

Optimisation of the *pair* computation kernel

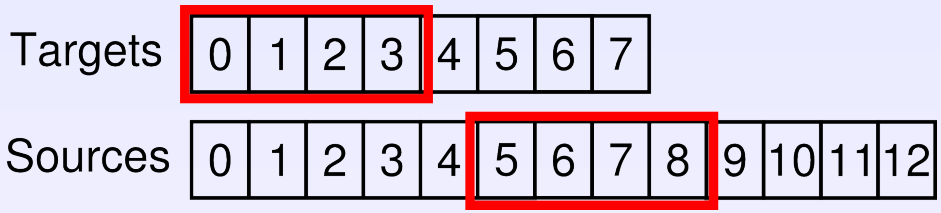
- Data layout:



$\longrightarrow$ 4 interactions / 8 body loads

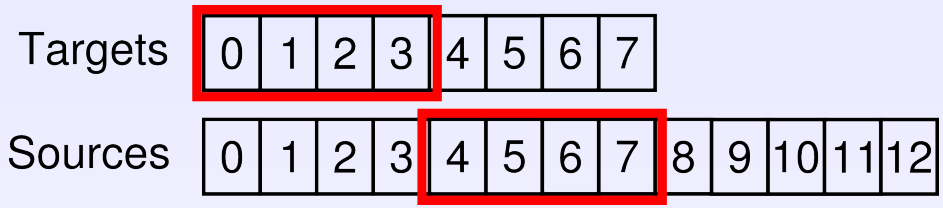
Optimisation of the *pair* computation kernel

- Data layout:



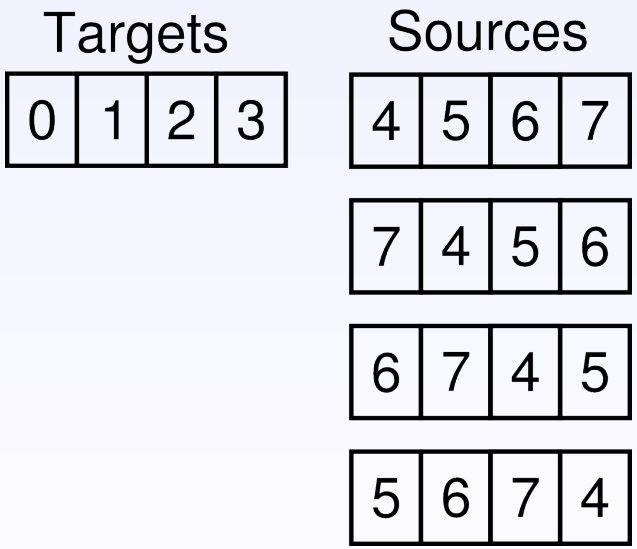
Optimisation of the *pair* computation kernel

- Data layout:



$\longrightarrow$ 16 interactions / 8 body loads

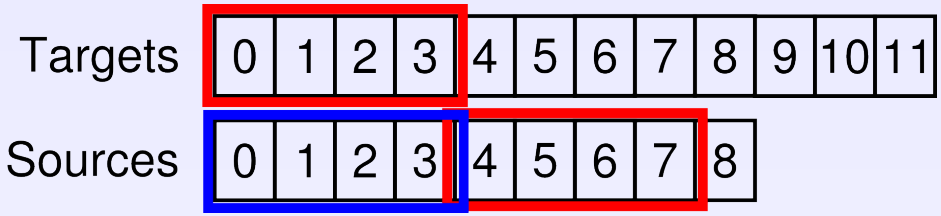
- Quadword rotates (dual-issued with floating point instructions):



$\longrightarrow$ thanks to numerous SPE vector registers

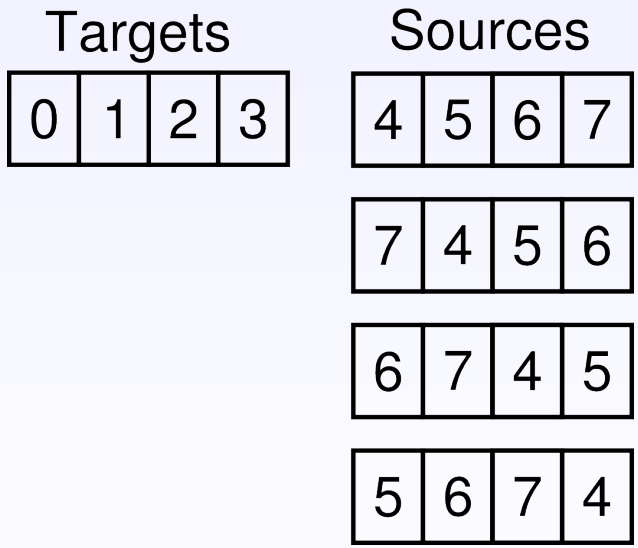
Optimisation of the *pair* computation kernel

- Data layout:



$\rightarrow$ 32 interactions / 12 body loads

- Quadword rotates (dual-issued with floating point instructions):



$\rightarrow$ thanks to numerous SPE vector registers

Design of efficient computation kernels (2)

SIMD code (2)

- ★ Many instructions in the internal loop body reordered at best by the compiler
- ★ Internal loop unrolled manually + interleaving of iteration instructions
- ★ *Own* computation kernel: interactions among the same 4 bodies → no use of mutual interaction principle
- ★ IBM rsqrtf4 vector function : floating-point $\frac{1}{\sqrt{x}}$ estimate + 1 Newton-Raphson iteration → single floating point precision

Flops per interaction

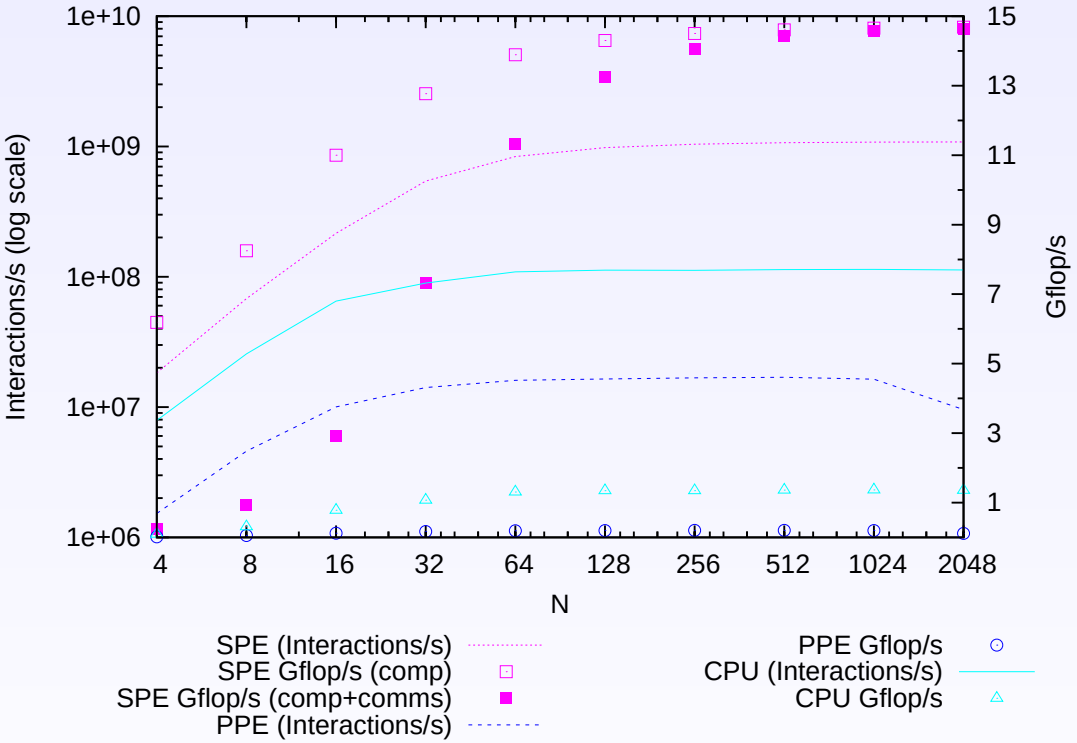
- ★ Pair computation: 27 flops/interaction
  ⇒ but thanks to mutual interaction principle: 13.5 flops/interaction
- ★ Own block computation: 24 flops/interaction
- ★ For reference, on CPU and PPE: 12 flops/interaction (mutual used)

Theoretical peak performance

- ★ 7 fused multiply-add (FMA) / 27 flops
  ⇒ 67.5% of SPE peak performance = 17.28 Gflop/s on 1 SPE

Results for pair computation on 1 SPE

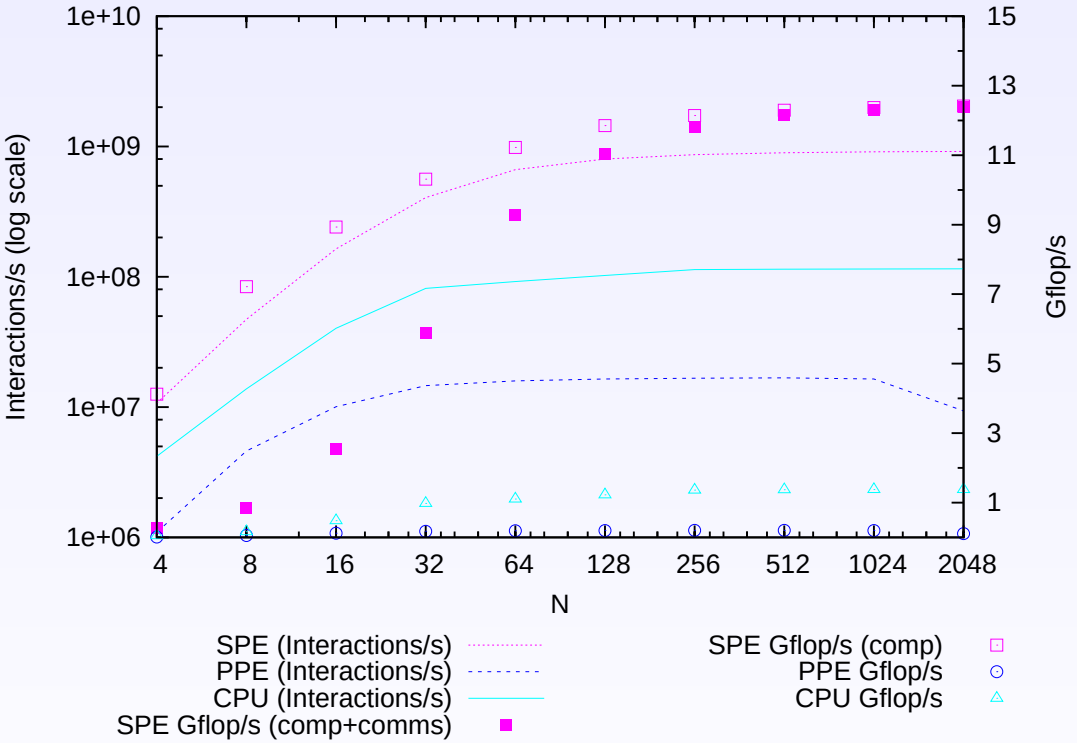
1 SPE / 1 PPE / 1 CPU core (Intel Xeon 5150, 2.66GHz)



- ★ PPE performs poorly
- ★ SPE up to 10x faster than CPU
- ★ SPE: up to 14.6 Gflop/s $\longrightarrow$ very good compared to theoretical 17.28 Gflop/s
- ★ DMA transfers not costful for high enough N values

Results for own computation on 1 SPE

1 SPE / 1 PPE / 1 CPU core (Intel Xeon 5150, 2.66GHz)

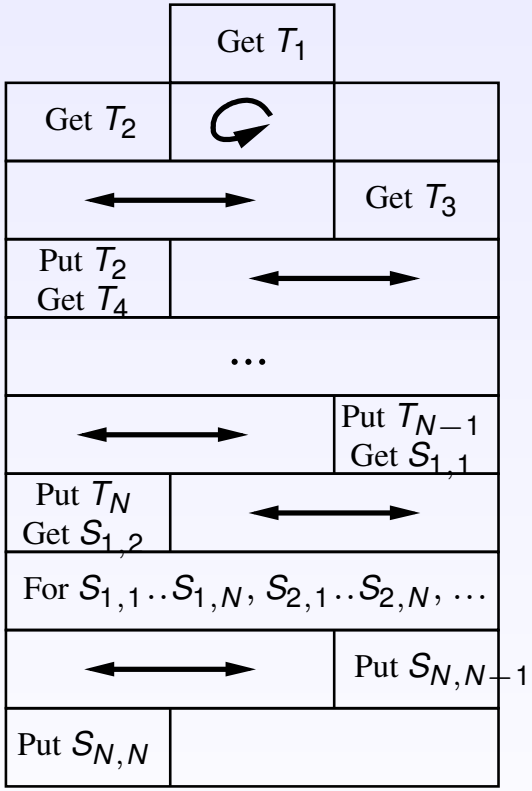


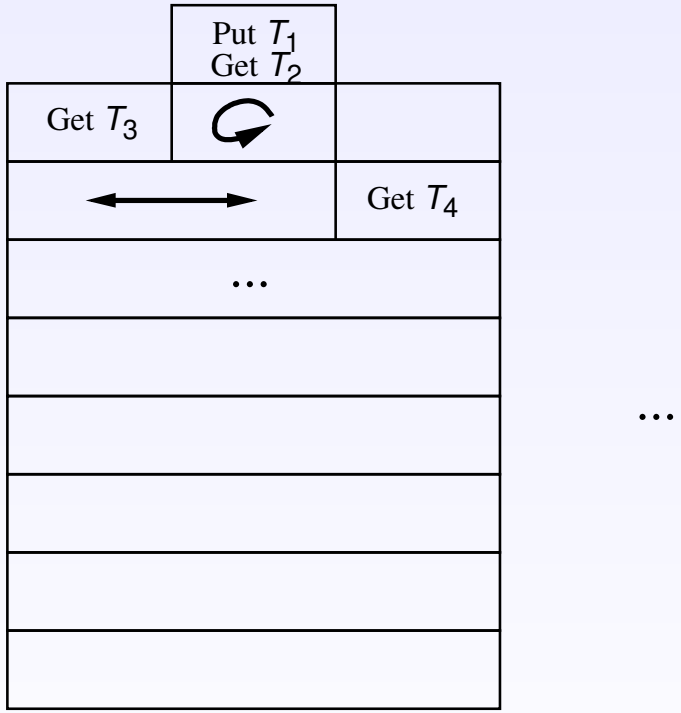
- ★ Same conclusion as for pair computation
- ★ SPE: up to 12.4 Gflop/s

Outline

DMA transfer design

- Bodies data transfered and treated by chunk of 2048 bodies
- Algorithm for computing 1 ***task*** = own computation of target leaf T + all pair computations between T and its nearest neighbors $S_1, ..., S_N$





...

- Only 3 shared I/O buffers
- Almost all DMA transfers overlapped with computation

PPE-SPE synchronization

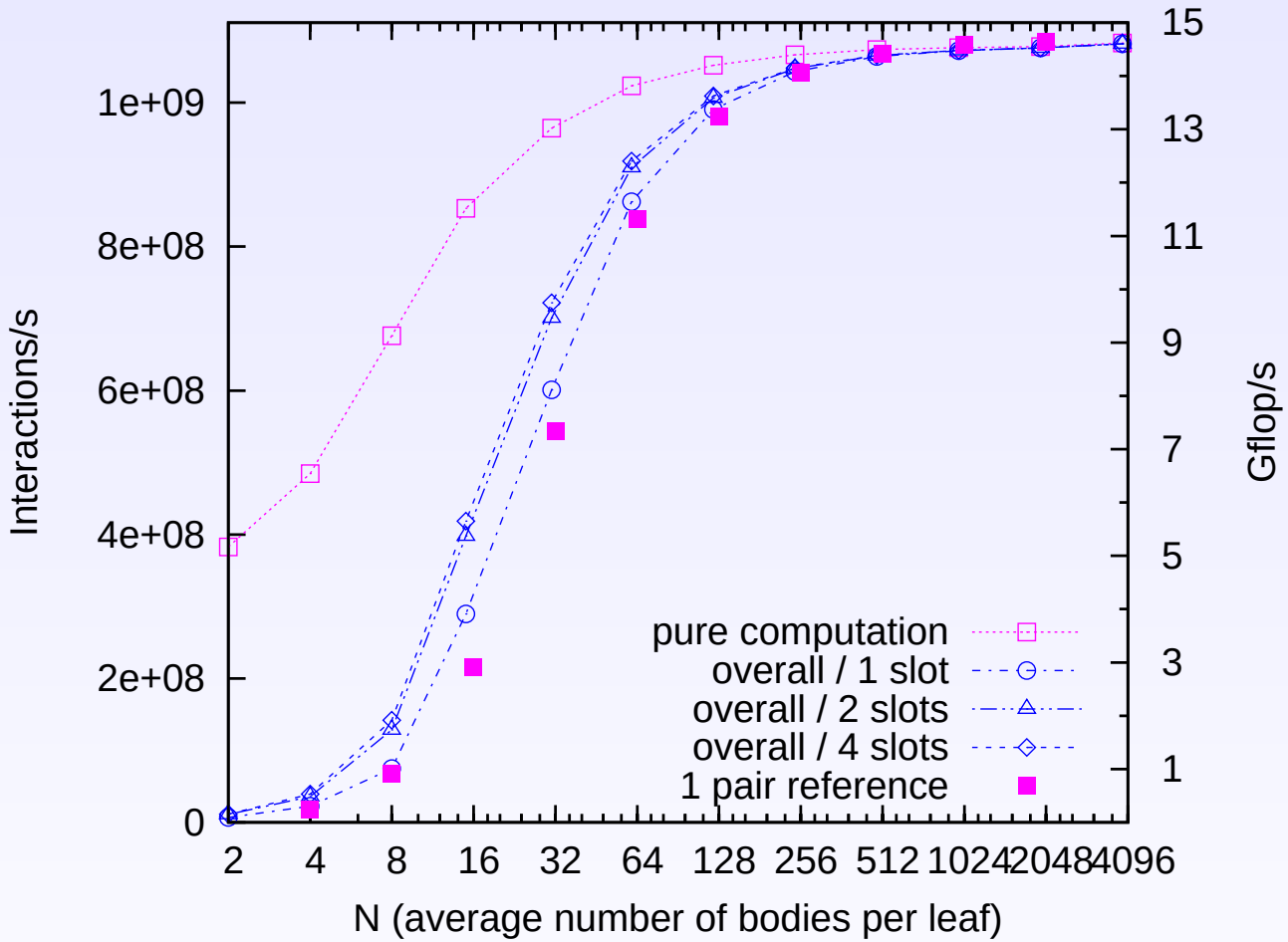
Objective

- ⋆ Maintain the computation kernel performance on the overall FMM near field computation **on 1 SPE**
  → minimize the time where the SPE is idle (between 2 computations)
  → fast notifications between the PPE and the SPE

PPE-SPE task synchronization

- ⋆ Task notification by PPE→SPE mailbox
- ⋆ Using several "*slots*"
  → several tasks assigned to SPE at any time
  → next task already available on the next slot
  → up to 4 possible slots
- ⋆ After task computation: SPE DMA writes in the Cell main memory
  - fastest SPE→PPE notification of task end
  - allows notification overwriting

Overall near field part on 1 SPE



- ★ Task DMA overlapping ⇒ overall performance better than 1 pair reference
- ★ 2 or 4 slots ⇒ performance ↗ (now use 2 slots)
- ★ Overall performance close to pure computation for $N \geq 64$
- ★ Overall performance maintained for $N \geq 2048$ (buffer size)

Outline

Objective and load balancing

Objective

- ⋆ Maintain the computation kernel performance on the overall FMM near field computation **on up to 16 SPEs**
  → minimize the time where SPEs are idle
  → responsive PPE code

Load balancing

- ⋆ No interaction computed on the PPE
- ⋆ Among the homogeneous SPEs:
  use static load balancing of FMB multi-thread parallelization

Locking stategy

Previous FMB multi-thread parallelization

- ★ lock bits set/unset for each pair or own computation
  → fine-grained locks and fined-grained computations
  → too strong synchronisation overhead on the Cell B.E.

New locking strategy

- ★ set together all lock bits of the whole task
- ★ if some lock bits already set ⇒ FIFOs to postpone the whole task
- ★ computation grain ↗ but possible deadlocks...
  ⇒ move from multi-thread PPE to single thread PPE
  - ▸ deadlocks easily avoided
  - ▸ no mutexes required
  - ▸ avoids costful thread context switches

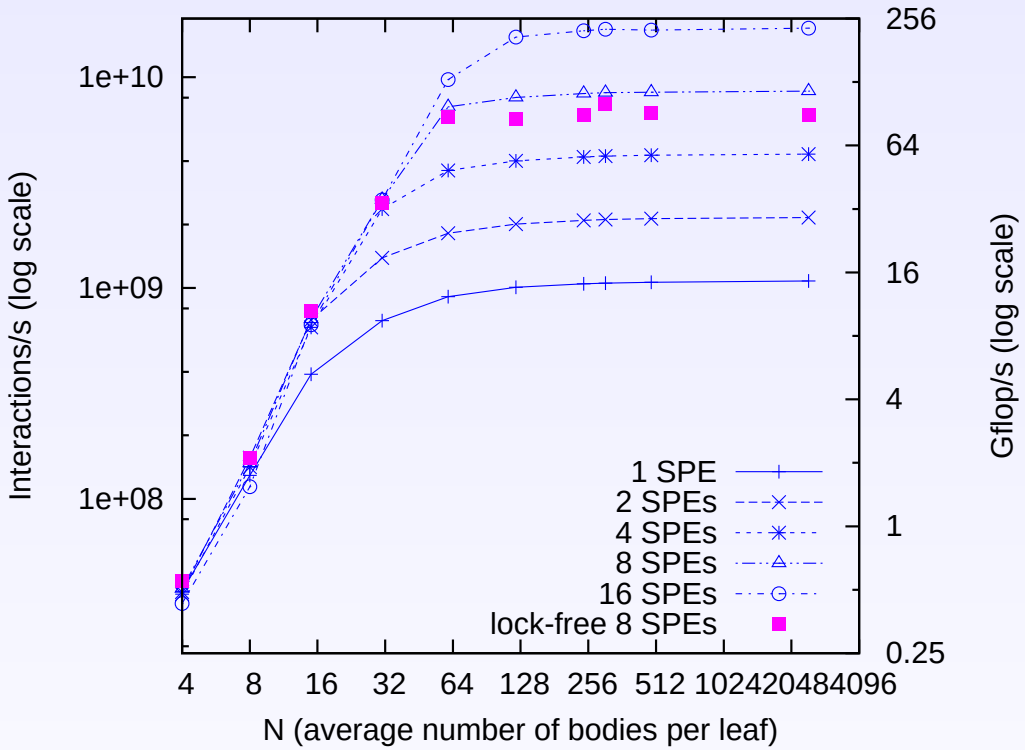
  ⇒ more responsive PPE to all SPEs

For comparison purpose: lock-free version

- ★ pair computations without mutual when the 2 leafs ∈ to 2 different threads
- ★ PPE management ↘ but SPE work ↗ (redundant computations)

Overall near field part on multiple SPEs: uniform distribution

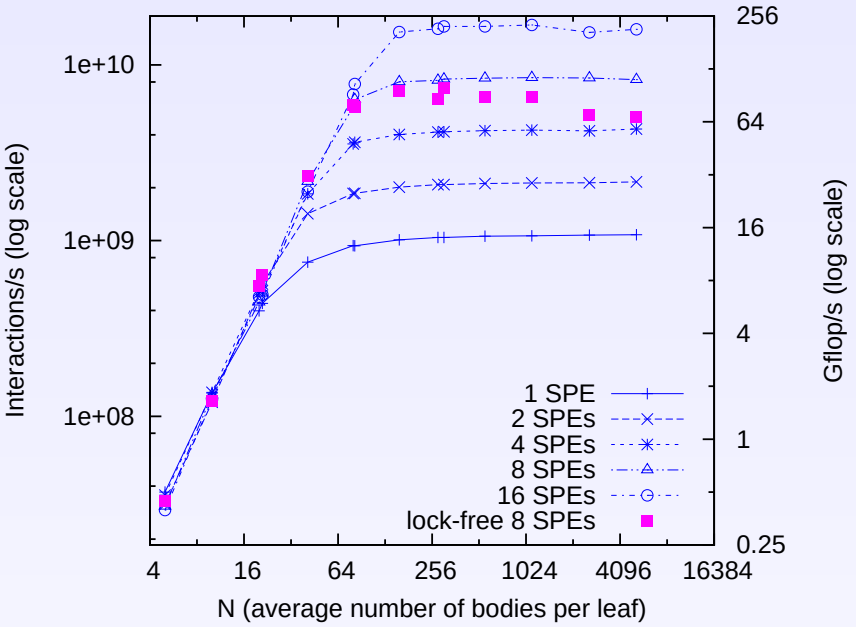
Up to 16 SPEs on 1 IBM QS20 blade (CINES, France)



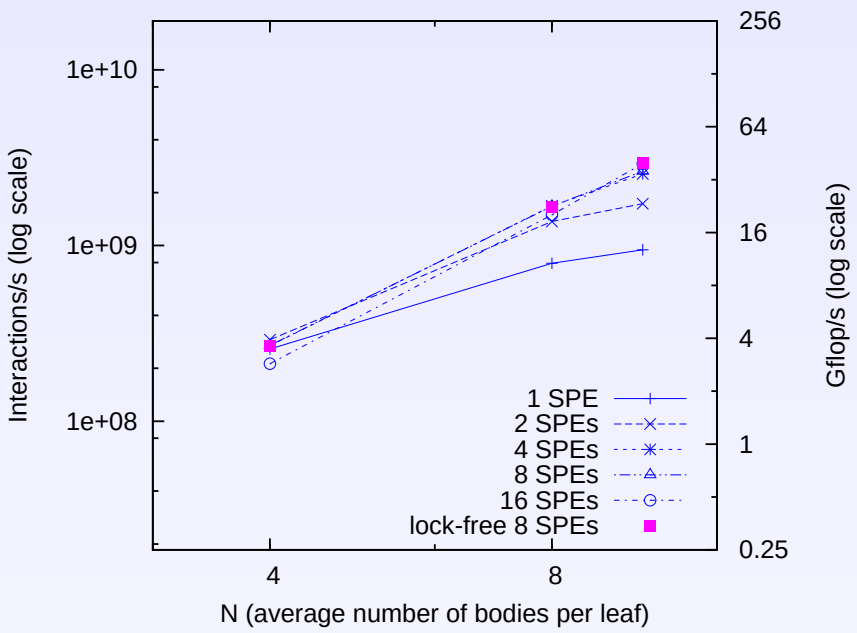
- ★ For $N \geq 128$ very good parallel accelerations up to 16 SPEs
- ★ Responsive enough single thread PPE code for 16 SPEs
- ★ Still very efficient on 1 Cell with $N \approx 64$

- ★ Too low $N \Rightarrow$ no good parallel efficiencies
  - ► too small computation grain
  - ► PPE not responsive enough
- ★ Lock-free version
  - ► hardly faster for low $N \Rightarrow$ PPE hardware not powerful enough...
  - ► slower for high N (because of redundant computations)

Overall near field part on multiple SPEs: cylinder and Plummer model



Cylinder



Plummer model

- ★ Non uniform cylindric distribution
  - same performance and same conclusions
  - validates our load balancing for both uniform and non uniform distributions
- ★ Highly concentratred astrophysical Plummer model
  - too many almost empty leafs with very low computation grain. . .

Outline

Conclusion and future work

Conclusion

- ★ First implementation of the Fast Multipole Method near field part on the Cell
- ★ Very efficient implementation for average number bodies/leaf ≥ 128
  - with up to 16 SPEs
  - for both uniform and non uniform distributions

Performance summary:

	FMM near field		Full direct computation			
	FMB / Cell (13.5 flops/interaction, since mutual)		Literature on 1 Cell	NVIDIA Tesla C1060 (20 flops/interaction, since no mutual)		
	1 Cell	1 QS20	($N = 8192$)	($N = 128$)	($N = 1024$)	($N = 16384$)
Nb of interactions/s	8.5×10^9	17×10^9		1.8×10^9	8.6×10^9	17.9×10^9
Gflop/s	115.8	230.4	≤ 83	35.5	171.1	359.0

Future work

- ★ Find optimized complex BLAS routines for far field Cell implementation
- ★ Looking for bigger Cell-based supercomputer