EADS INNOVATION WORKS

Mixed BEM/FEM simulation using MUMPS at EADS Innovation Works

April 15th, 2010

SYLVAND Guillaume (CTO IW SE MA) guillaume.sylvand@eads.net

Innovation Works
(former Corporate Research Center
(former CCR))

- EADS : Airbus, Eurocopter, Space (Astrium, Ariane), MBDA, Military aircraft (Typhoon) : 118.000 employees.
- Distributed :
  - Germany : Ottobrun
  - France : Suresnes + Toulouse
  - UK, Singapore, Bangalore (Alleon!), Moscow
- Exclusively research, for business units

Domain

- Volumic applications in EM
- Other Applications : aero-acoustics, time domain,..

Maxwell Equations

- Application : furtivity
- Plane wave illuminating an object
  - What is the EM field sent back ?

- At IW : since 80’s working on *Boundary Elements Methods* (BEM) based on *Integral Equations*

The Boundary Element Methods

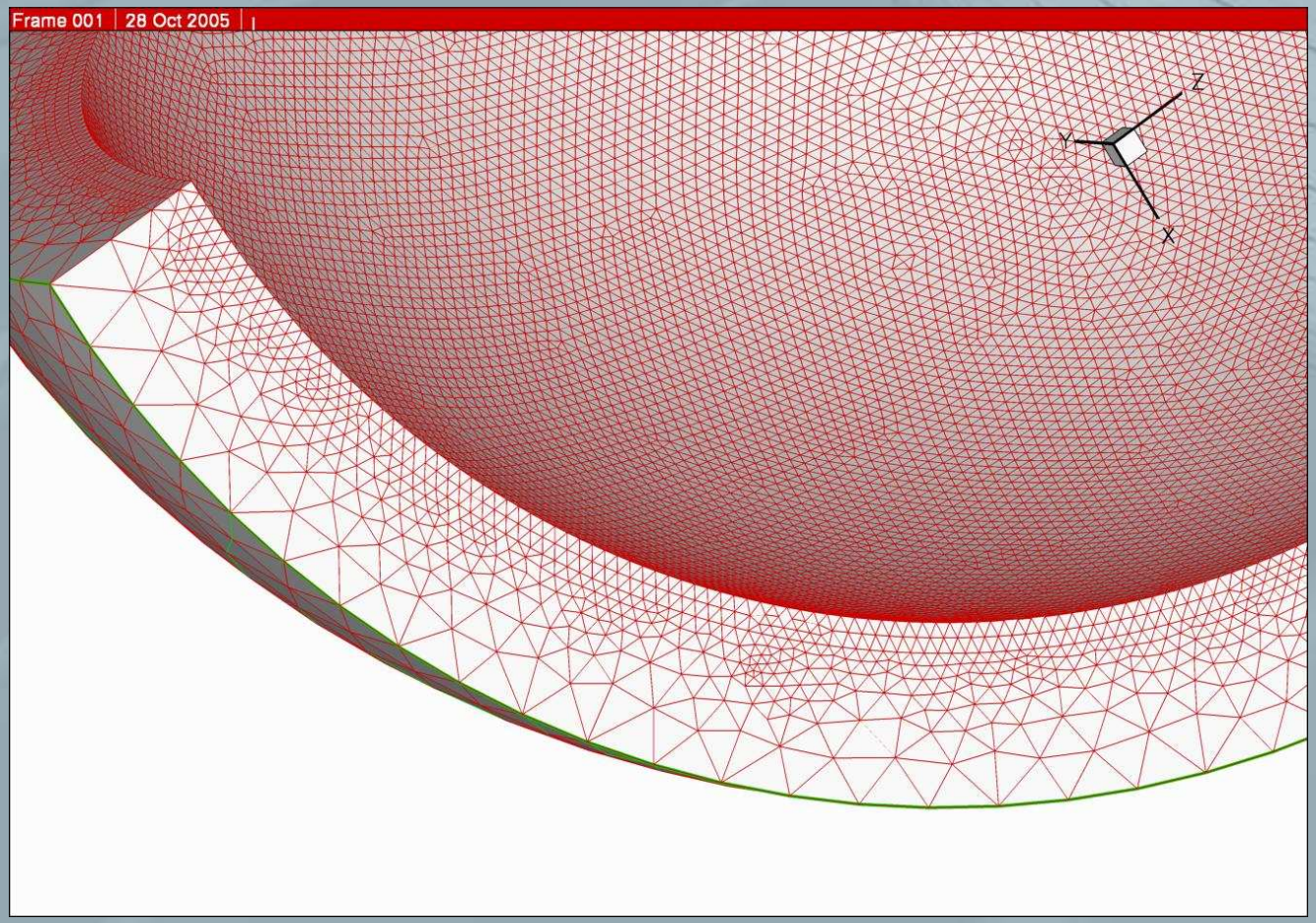
- BEM : developed since the 80's at IW in electromagnetism and acoustics
- Advantages:
  - Very *accurate* method
  - Only surfacic mesh (easier to produce)
  - Less unknowns
  - No troncature condition
- Drawback:
  - *dense* matrix
  - Complex implementation
- In wave propagation : the mesh size is given by the wavelength
- Very large number of unknowns for *high frequency* and/or *large objects*

The Fast multipole method

- FMM : launched in 97 at CRC
- It is a way to compute matrix-vector product
- FMM can be used to solve Boundary Element Method problems with an *iterative* solver
- Very powerful : easily solve problems with *Millions of unknowns*.
- FMM is based on a mathematical formulae that allows to speed-up the treatment of the interactions between “distant” parts of the object.

Volumic/surfacic coupling

- BEM = only homogeneous isotropic media
- For inhomogenities or anisotropic cases : volumic scheme



Volumic/surfacic coupling

- 2x2 Block system
  - Surfacic = 1 full matrix (can be treated with FMM)
  - Volumic = 3 sparse matrices

$$\begin{bmatrix} A_{cc} & A_{cv} \\ A_{vc} & A_{vv} \end{bmatrix} \times \begin{bmatrix} X_c \\ X_v \end{bmatrix} = \begin{bmatrix} b_c \\ b_v \end{bmatrix}$$

- Several Mdof in the volume, kdof on the surface

Solver choice for BEM

- Full complex matrix
  - Direct : $O(n^3)$
  - Iterative : $O(n^2)\ N_{iter}$
  - Iterative + FMM : $O(n \log n)\ N_{iter}$

- How to handle the FEM part ?
  - 3 solvers

Solvers for BEM/FEM
Schur complement + direct solver :

- Extract Xv from eq. 2 :

$$X_v = A_{vv}^{-1}.(b_v - A_{vc}.X_c)$$

- Insert in eq. 1 :

$$(A_{cc} - A_{cv}.A_{vv}^{-1}.A_{vc})X_c = b_c - A_{vv}{}^{-1}.b_v$$

- We compute explicitly this matrix then we use a direct solver on it
- MUMPS is used for *large* number of sparse RHS (the RHS are the column of *Avc*

Solvers for BEM/FEM (2)
Schur complement + iterative solver :

- Same idea leads to :

$$(A_{cc} - A_{cv}.A_{vv}^{-1}.A_{vc})X_c = b_c - A_{vv}{}^{-1}.b_v$$

- We do not compute explicitly this matrix : we use a pseudo-matrix
- MUMPS is used for small number of RHS at each iteration
- Can be used with FMM
- Standard use of mumps

Solvers for BEM/FEM (3)
Iterative solver :

- Iterative solver on the initial system with meta-matrix

$$\begin{bmatrix} A_{cc} & A_{cv} \\ A_{vc} & A_{vv} \end{bmatrix} \times \begin{bmatrix} X_c \\ X_v \end{bmatrix} = \begin{bmatrix} b_c \\ b_v \end{bmatrix}$$

- MUMPS can be used as a preconditionner for Avv :

$$\begin{bmatrix} A_{cc} & A_{cv} \\ A_{vc} & A_{vv} \end{bmatrix} \times \begin{bmatrix} M_{cc} & 0 \\ 0 & M_{vv} \end{bmatrix} \times \begin{bmatrix} X_c \\ X_v \end{bmatrix} = \begin{bmatrix} b_c \\ b_v \end{bmatrix}$$

- Can be used with FMM
- Less efficient : very large number of unknowns (surface + volume)

Tests

- Many tests on 1 configuration (18.758 dof surface, 2.423.135 dof volumic)
- Various compilations (intel 10 or 11, Gnu) x (Debug, Optimised) x (openMPI, HPMPI) x (blas netlib/MKL 9 or 10)
  - *Conclusion 1 : pbm in intel 10 with « –O ». « -O1 » is OK*

Tests

- Various MUMPS version (4.7.3, 4.8.4, 4.9)
  - Time : Gain of 3% in-core, up to 2.2 x faster in out-of-core with 4.9.
  - Gain of 20 % in memory.
  - Overcost of OOC is reduced to (10% in time)
  - *Conclusion 2 : switch to 4.9*

Speed-up 4.7.3 -> 4.9		
	intel 11	intel 10
global	1,57	1,58
in-core	1,02	1,03
out-of-core	2,23	2,23

Memory gain 4.7.3 -> 4.9		
	intel 11	intel 10
global	1,22	1,23
in-core	1,23	1,23
out-of-core	1,20	1,22

Tests

- Ordering test : 6 orderings
  - Best results with scotch (2% in time 10 % in memory over metis)
  - *Conclusion 3 : switch to scotch*

Speed-up METIS->SCOTCH		
	intel 11	intel 10
global	1,05	1,02
in-core	1,05	1,01
out-of-core	1,05	1,03

Memory gain METIS->SCOTCH		
	intel 11	intel 10
global	1,12	1,10
in-core	1,07	1,07
out-of-core	1,17	1,13

Tests

- Single precision
  - Gain of x1.5 in time and memory
  - Accuracy : depends on the application (we need very high accuracy in these applications)
  - *Conclusion 4 : to be tested…*

Memory gain / Speedup					
intel 11			intel 10		
Local	Global	Time	Local	Global	Time
1,66	1,61	1,57	1,66	1,61	1,51

		Intel 11,Linux, hpmpi					
		Factorisation		Résolution			
Mode	nbRHS	Local (Mb)	Global (Mb)	Local (Mb)	Global (Mb)	Time (s)	Result
Out-of-core.	1	402	11550	182	4552	33704,1	OK
Out-of-core.	4	402	11550	304	8462	12554,7	OK
Out-of-core.	8	402	11550	466	13681	8550,2	OK
Out-of-core.	16	402	11550	790	24113	6826,1	OK
Out-of-core.	32	402	11550	1429	44689	6228,3	OK
Out-of-core.	64	402	11550	2736	85811	6039,6	OK
In-core.	1	891	22454	805	19767	21870,3	OK
In-core.	4	891	22454	928	23660	9000,2	OK
In-core.	8	891	22454	1093	28856	6607,2	OK
In-core.	16	891	22454	1423	39246	6168,5	OK
In-core.	32	891	22454	2073	59720	5637,3	OK
In-core.	64	891	22454	3373	100663	5625,2	OK

Tests

- Number of RHS : test ic/ooc for 1, 2, 4, 8, 16, 32, 64 RHS
  - More is better
  - Memory used by resolution now returned.
  - *Conclusion 5 : double the default value (16 ic, 32 ooc) to gain 10 to 15% in time.*

Tests

- Compiler : intel 10 vs 11
  - 1% gain only on xeon 5160 (core 2)
  - 6% on Nehalem
  - *Conclusion 6 : not bad !*
- CPU tests : Xeon 5160 (tic) vs Xeon 5570 (hpc3)
  - Gain of a factor 2 !
  - But still bug on some other part of the software on Nehalem
  - *Conclusion 7 : powerfull but…*

Speed-up	
intel 10 à intel 11	Tic à HPC3
tic hpc3	intel 11 intel 10
1,01 1,06	2,01 1,90

Tests

- Scalability : from 1 to 128 cores.
  - Time decreases up to 128
  - Efficiency below 50 % at 32 cores.
  - could be improved with threads/MPI (multithreaded blas insufficient)
  - *Conclusion 8 : correct*

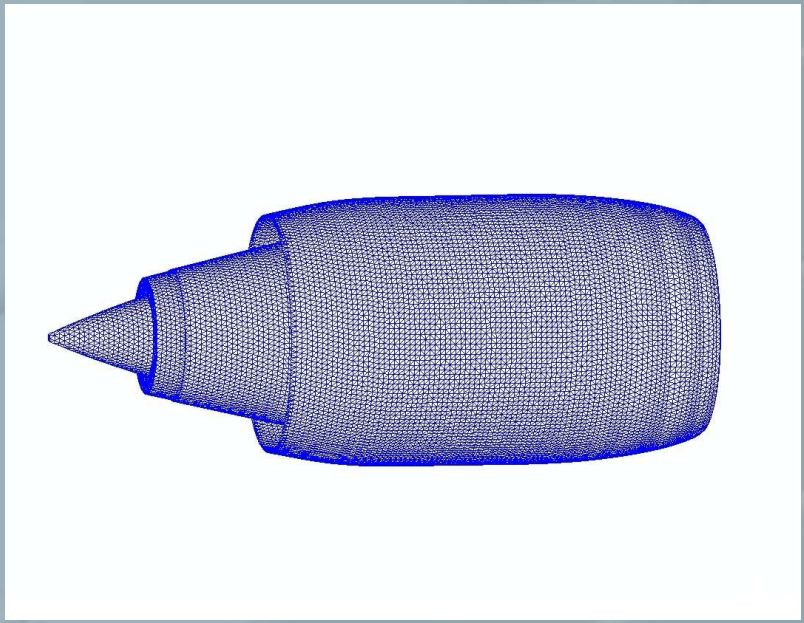
Efficacité parallèle				
	Tic		Nehalem	
processeurs	intel 11	intel 10	intel 11	intel 10
2	0,97	0,97	0,95	0,95
4	0,72	0,72	0,86	0,86
8	0,68	0,68	0,71	0,70
16	0,52	0,52	0,54	0,53
32	0,40	0,41	0,45	0,44
64	0,27	0,27	0,28	0,27
128	0,17	0,17	0,19	0,18

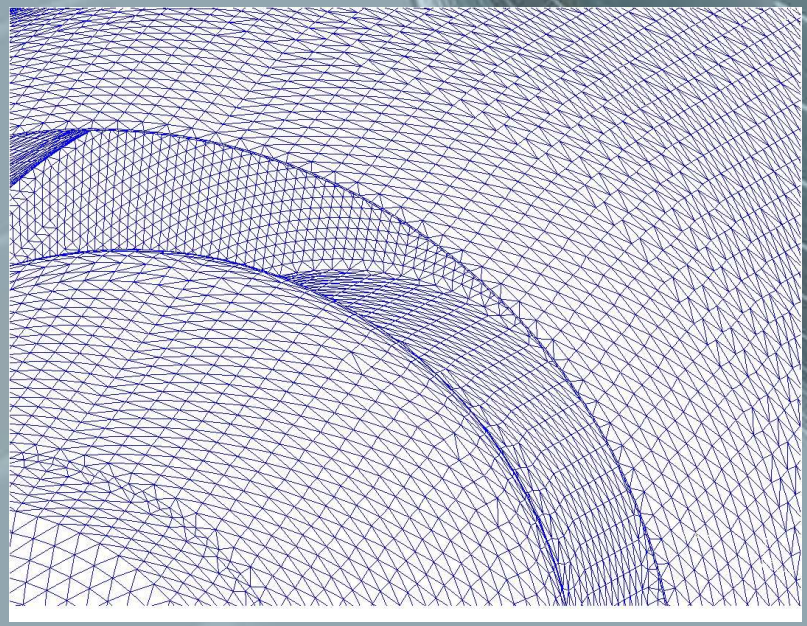
Latest results

- Software : Mumps 4.9 + scotch + « -O1 » + more RHS :
  - Nice speed-up and memory reduction
  - Everything is fine
  - Deployed
- Hardware : Migration toward Nehalem + intel 11

Other applications

- Model for acoustic radiation by engines (Airbus) with realistic flow





- Time domain

Feature « requests » (or ideas ?)

- Mixed parallelism : MPI+openMP
- Mixed computation : CPU+GPU

- Interested of course in anything that makes MUMPS harder, better, faster, stronger !

- Small things :
  - Use **`#define MUMPS_XX 42`** in mumps.h to replace icntl(23)=41 by something more human-readable

Thank you