Remarques sur les grands maillages, comment les construire et/ou comment les éviter

Inria, EPI Gamma3, A. Loseille, L. Maréchal and P.L. G.

AE CS@EXA

- Constructing large meshes
- Anisotropy
- Adaptive loop, error estimates and goal oriented
- Multi-threading and GPU

Constructing large size meshes (1/2)

- basic (presumably reliable) algorithms and scaling issues:
  - temporary need of an arbitrarily large "array"
  - cache misses
  - number representation (for floating values but even for integers)
  - robustness (even for integers)
  - funny complexity suddenly revealed :
    - ex 1: $c = 10^{-8} O(n^2) + O(n)$ may have two different behaviors!
    - ex 2: $c = k\, O(n)$ may be $O(n^2)$!
  - new patterns suddenly appear, never met before or being marginal
- remedies are twofold:
  - in terms of algorithms :
    - re-design
    - massive use of (Peano)-Hilbert renumbering
  - in terms of computer science: number representation, multi-threading and GPU uses
- new algorithms in `GHS3D` and `HEXOTIC`, `LP-Lib` and `GM-Lib` libraries
- tet and hex meshes of about 2 billions of elements successfully generated (on a serial machine)

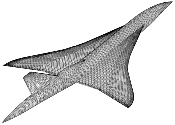
Constructing large size meshes (2/2)

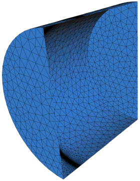
- pre-partitioning of the domain + local meshes (what about interfaces?)
- refinement of a coarse mesh (does a coarse mesh exist?)

on the other side

- partitioning of a large mesh into parts to be distributed (what about interfaces?)

Anisotropy (1/3)

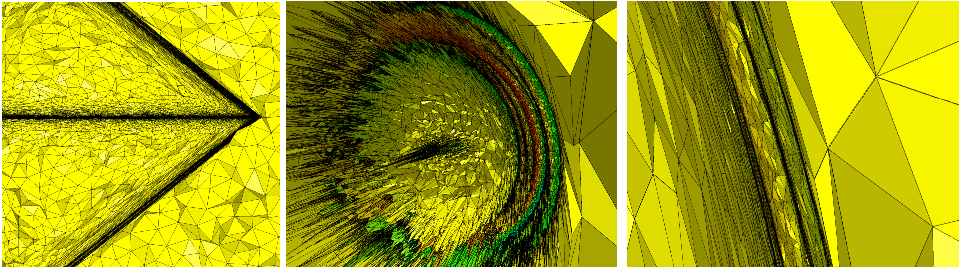




Aircraft = 36m, Mesh 1mm to 30cm

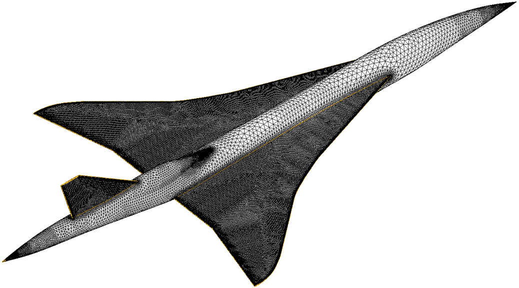
Domain 2km

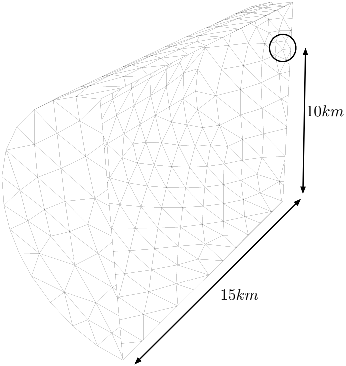
$\approx$ 22.3 millions of tets *i.e.* 0.1 billion of DoF



1m precision leads to $\Longrightarrow$ 200 billions of DoF

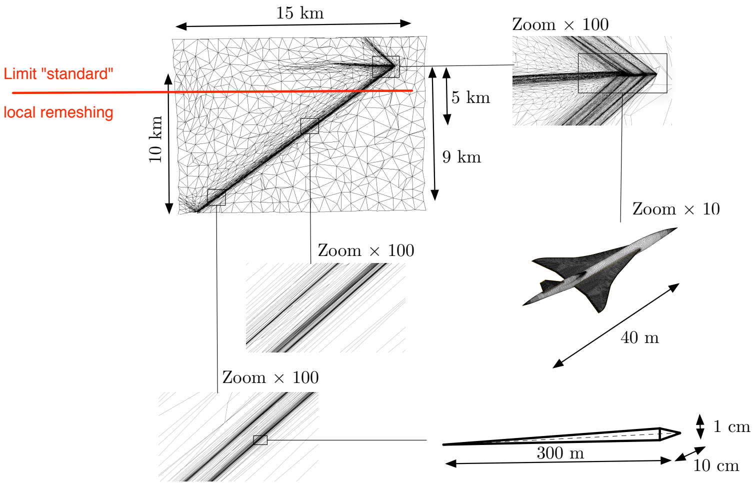
A full scale supersonic simulation





- initial mesh: frontal mesh generation, # vert. 415 535, # tets 2 397 666
- volume $[5.4e^{-11}, 4.7e^{10}]$
- $h_{min}/h_{max} = 1.e^{-9}$

Anisotropy (3/3)



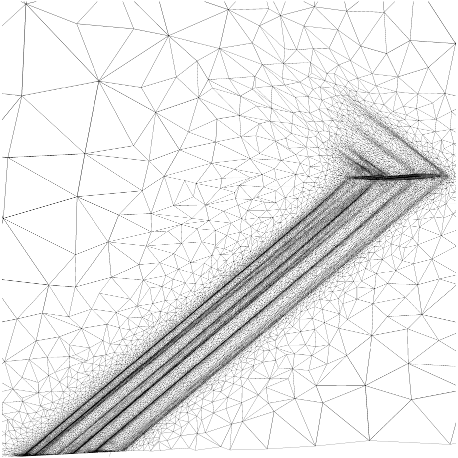
- Error estimate: $\mathbf{L}^2$ estimates $\Longrightarrow$ no h_{min} and small scales
- Solver : Implicit time-stepping
- Adaptation: anisotropy and quality $\Longrightarrow$ accuracy and stability

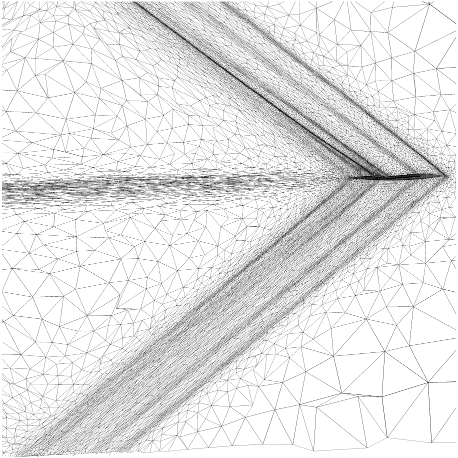
Goal-oriented, adaptive loop (1/3)

- Choice of a functional j and, an area of interest γ and an adjoint state
  - Example of functional :

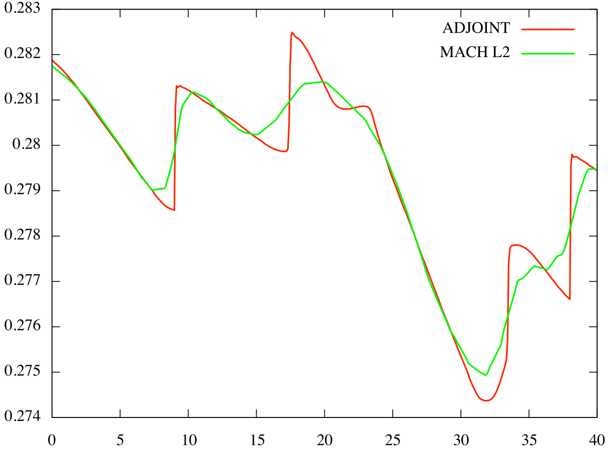
$$j(W) = \int_{\gamma} \left(\frac{p - p_{\infty}}{p_{\infty}} \right)^2 \mathrm{d}\gamma$$

Goal-oriented, adaptive loop (2/3)

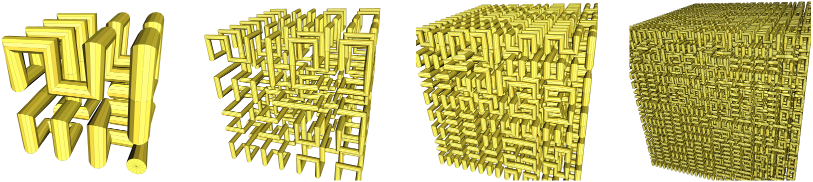




Goal-oriented, adaptive loop (1/3)



Hilbert curves:



Map a 3D domain onto a 1D domain

Application to mesh re-ordering:

1. Re-order vertices in order to be compact

$$\| Idx(\mathbf{v}_1) - Idx(\mathbf{v}_2) \| \text{ small if } \| \mathbf{v}_1 - \mathbf{v}_2 \| \text{ small}$$

2. Sort entities: Tetrahedra, edges, etc.. $Tet = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4]$

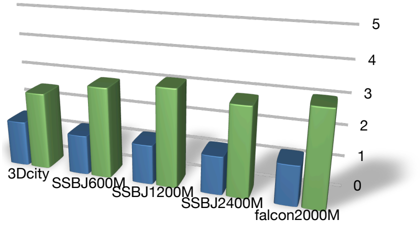
$$\text{Hilbert} \Longrightarrow \min_{Tet} \left(idx(Tet[i]) - idx(Tet[j]) \right)$$

Sort Entities by minimal index : $Min_i Idx(Tet[i])$

Hilbert Impact I. (2/2)

Serial scaling with re-ordering

- Flow solver

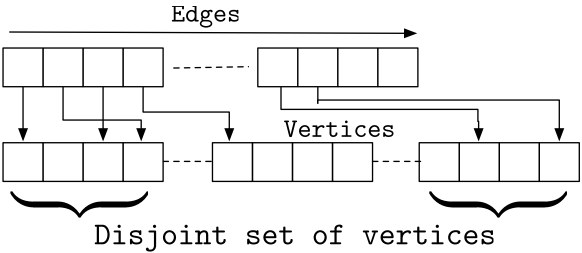


Entities sort	1.3	to	1.5
+ Hilbert sort	2.5	to	3

Hilbert Impact II. Parallelization (1/6)

A solution for parallel computing

Hilbert re-ordering create implicitly independent set of blocks



$\Longrightarrow$ The two blocks of edges can be run in parallel

Idea

Split entities and manage collisions for parallel runs

Hilbert Impact II. Parallelization (2/6)

The pros:

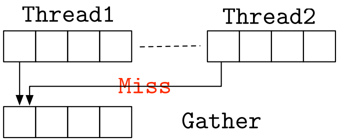
- Small impact on the serial code

```
BeginDependency(Tetrahedra,Vertices);
for (iTet=1; iTet<=NbrTet; ++iTet) {
  for (j=0; j<4; ++j) {
    AddDependency( iTet, Tet[iTet].Ver[j] );
  }
}
EndDependency(Tetrahedra,Vertices);
```

```
Solve(Tetrahedra,iBeg,iEnd) {
  for (iTet=iBeg; iTet<=iEnd; ++iTet) {
    // .... same as serial
  }
}
```

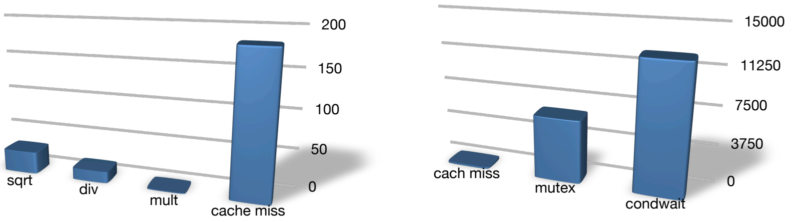
- Load-balancing on-the-fly
- Asynchronous parallelization
- No overhead memory

20 to 30% faster than Scatter/Gather for all test cases on 8 cpus.



Hilbert Impact II. Parallelization (3/6)

Problematic: Synchronization costs

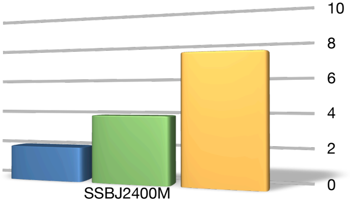


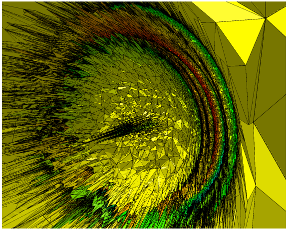
Adaptive strategy according to the amount of work

- Light amount of work: linear loop, without dependencies
  $\Longrightarrow$ Threads are run with interlacing, no overhead time for load-balancing
- Huge amount of work: main solver loops
  $\Longrightarrow$ Threads manage load-balancing and dependencies, but negligible overhead time.

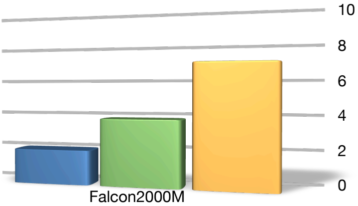
Hilbert Impact II. Parallelization (4/6)

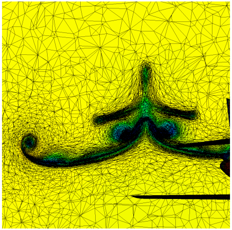
- SSBJ scaling, Core 2 Duo @ 2,5Ghz
  $2,360,877$ Vertices, $13,933,849$ Tetrahedra





- Falcon scaling, Core 2 Duo @ 2,5Ghz
  $2,025,231$ Vertices, $11,860,697$ Tetrahedra





Hilbert Impact II. Parallelization (5/6)

Ongoing work

- Modify memory management,
- Test ordering strategy: static versus dynamic
- Cache line management: Modify and fit package size,
- SGI's monitoring tools.

Partial current results: Acceleration factor w.r.t serial runs

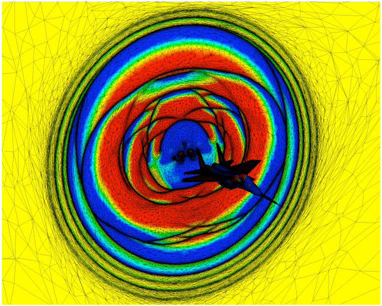
Test cases	2	4	8	16	32
Onera m6Wing	1.7	3	7.7	11	14.6
NASA Spike	1.6	2.7	4.2	6.58	10.3
Onera m6Wing	1.96	3.91	7.69	15	29
NASA Spike	2	3.9	7.8	15.6	28

Hilbert Impact II. Parallelization (6/6)

- Current run at 64 threads
- Current run on larger anisotropic meshes: Quiet-boom F15
  - Vertices $60,000,000$
  - Tetrahedra $400,000,000$

NASA Dryden Flight Research Center Photo Collection
http://www.dfrc.nasa.gov/Gallery/Photo/index.html
NASA Photo: ED06-0184-23 Date: September 27, 2006 Photo By: Carla Thomas

NASA F-15B #836 in flight with Quiet Spike attached.



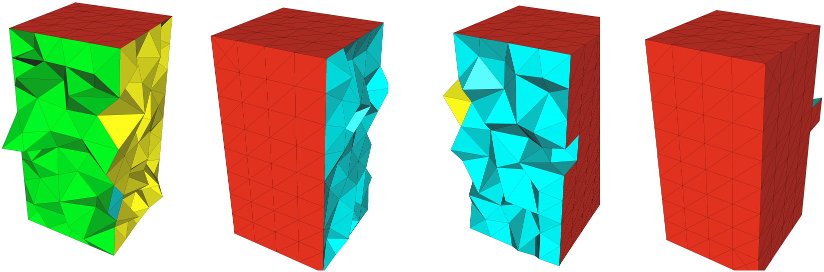
Hilbert Impact III. Mesh partitioning (1/5)

Goal: parallel anisotropic adaptive mesh adaptation

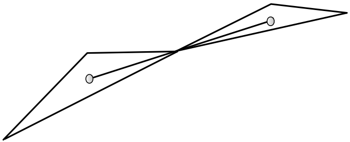
Use Hilbert curves compactness property to create partitions

Algorithm

1. Create the list of gravity centers of tetrahedra
2. Re-order by Hilbert this list
3. Split this list in equal parts according to the number of required parts



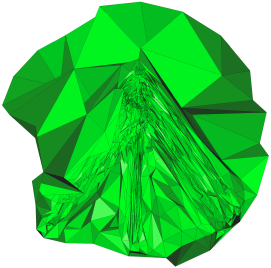
Problematic: Creation of unconected sub-domains on anisotropic unstructured meshes



Correction algorithm

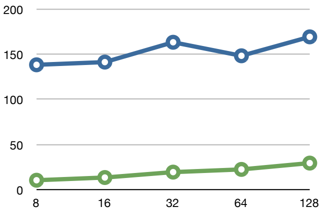
1. Compute sub-domains from initial partitions
2. Merge neighboring sub-domains until equality

Gathering parts



- Find geometrically one point on each interface
- Use topology to recover mappings

Cpu time in sec to create 2^n partitions



- Anisotropic SSBJ test case
- 22.3 millions of tetrahedra
- Cpu time for split/gather in serial

Hilbert Impact III. Mesh partitioning (4/5)

Parallel anisotropic mesh adaptation

1. Split the initial mesh: each part is ordered using Hilbert based strategy
2. Do only point insertion, collapses, swaps
3. Merge new adapted parts, and split the new mesh with random interfaces: each part is ordered using Hilbert based strategy
4. Do mesh optimization: swaps and smoothing
5. return to 1

Hilbert Impact III. Mesh partitioning (5/5)

Scaling on SSBJ test case: $> 600,000$ inserted points

8 cpus speed-up: 7.8 with Hilbert 45.5 with initial (7min)

Features

- Mesh generation of the order of several minutes, up to 1 million inserted points

$\Longrightarrow$ Meshing time (re)-becomes negligible with respect to solver time

- Possibility of predicting each part mesh adaptation time by using metric density
- No change of the executable, another adaptive mesh generator may be used
- Improves the serial mesh adaptation algorithm: divide and conquer strategy
  - Reduce scale factors
  - Reduce randomized algorithms effects
  - Provide optimal ordered meshes for each stage: insertion and optimization

One (our) view of HPC

Initial problem of size N (DoF)

1. Anisotropic mesh adaptation:

   $\Longrightarrow$ reduction of N from $\approx$ Reduction of DoF to $N^{\frac{1}{3}}$ to $N^{\frac{2}{3}}$

   Ex: $1,000,000$ DoF leads to $1,000$ DoF

2. Cache miss reduction and re-ordering

   $\Longrightarrow$ reduction of N from ≈ 3 to 10

3. Multi-threaded parallelization

   $\Longrightarrow$ reduction of N from ≈ 4 to 32

- View as the new serial programming
- Try optimal use of computing resources: seek for near-optimal speed-up
- Approach compatible with distributed parallelization

Softwares

- GHS3D: Hilberted (Up to 2 billion elements)
- HEXOTIC: Hilberted + Multi-threaded (10 million to 1 billion)
- SHRIMP: Adaptive parallel mesh generation (through FEFLO.A) and mesh partitioning
- FEFLO.A: Hilberted + Multi-threaded, Anisotropic mesh generation (10 to 100 million tets)
- Multi-threaded codes use LP-Lib