Distributed mesh and graph computations within PaMPA and PT-Scotch libraries

C. Lachat & F. Pellegrini & C. Dobrzynski
INRIA Bordeaux – Sud-Ouest

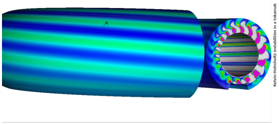
Contents

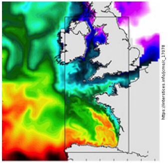
1

Introduction

Introduction

- First part
  - Numerical simulations need more and more resources
  - Clusters become more and more bigger
  - A part of a cluster is used by the program
    $\Rightarrow$ Take into account target architecture for static mapping
- Second part
  - Some numerical simulations based on unstructured meshes
  - Remeshing is often required
  - Sequential remeshers are limited by the size of the mesh
    $\Rightarrow$ Build parallel remesher with sequential remesher





2

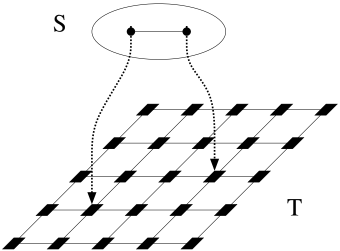
Recent advances on static mapping in Scotch

Context

- Very large scale computers are highly non-uniform
  - Hierarchical architectures
    - Clusters of multiprocessor blades
    - Multi- or even many-core processors
  - Mix of the distributed- and shared-memory paradigms
  - Communication latency and bandwidth depends on the respective locations of intercommunicating processes
- Impact on application software
  - Data locality is essential to achieve performance
  - Target architectures have to be taken into account
  - Tighter interaction between software and system components
    - Batch scheduler should tell applications what processing elements are assigned for execution
    - Process and/or data placement tools have to take scheduler information into account

Static mapping in SCOTCH

- Since its inception in 1992, SCOTCH was designed to compute process-processor mappings that take into account target topology

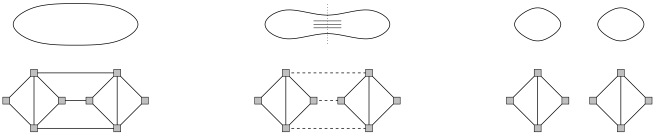


- Communication cost function accounts for distance

$$f_C(\tau_{S,T}, \rho_{S,T}) = \sum_{e_S \in E(S)} w(e_S)|\rho_{S,T}(e_S)|$$

Dual Recursive Bipartitioning (DRB)

- SCOTCH computes its (initial) mappings by means of the Dual Recursive Bipartitioning (DRB) algorithm
  - Recursive process using a “divide & conquer” approach
  - Associates a part of the source graph to each part the target graph
- Until each target subgraph is reduced to a single vertex, do:
  - Bipartition target graph
  - Use target graph bipartition imbalance to bipartition associated source graph



- During each bipartitioning of source graphs, a partial cost function uses distance information regarding both internal and external edges so as to privilege locality

Target graph descriptions

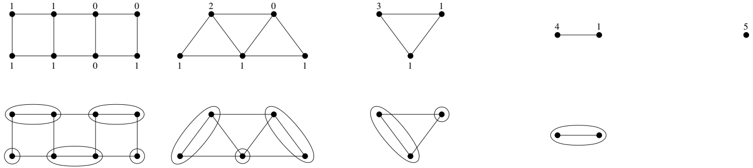
- In order to evaluate the partial cost function while (bi)partitioning the source graph, a target architecture description must provide three abstractions:
  - *Domain* structure: represents a set of processors in the target architecture
  - *Domain bipartitioning function*: bipartitions a given domain into two disjoint subdomains
  - *Domain distance function*: provides (an estimate of) the distance between two domains in the target architecture
- SCOTCH implements two families of target architecture descriptions:
  - Decomposition defined
    - Can represent very irregular target architectures
  - Algorithmically defined
    - We will focus on this class during this talk

Algorithmically-defined architectures

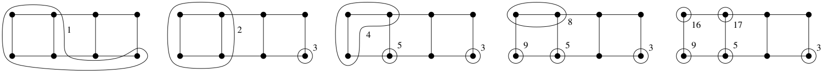
- Classical architectures are represented on the form of hard-coded instances of a generic class
  - E.g.: `mesh2D`, `hcub`, etc.
  - Provide all the necessary information thanks to hard-coded routines
- Distances are provided as shortest path length
  - E.g.: for `mesh2D`, Manhattan distance between centers of rectangular domains
- In former SCOTCH implementations, algorithmically-defined architectures can only describe complete computer systems
  - Yet, a part of a torus is not a torus!
  - Disconnected parts are not managed, either
- Need for a more generic representation

Building a bipartitioning hierarchy through matching

- Recursive matching and coarsening allows one to build a locality-preserving bipartitioning tree of a (disconnected) part of any architecture



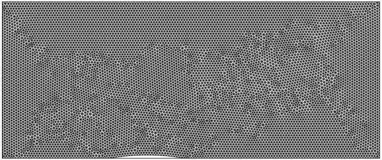
- By traversing the coarsening tree from its root, one can build a locality-preserving bipartitioning tree

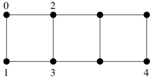


- Tree is unbalanced but processors are distributed that way!

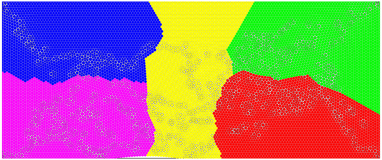
How it works in practice

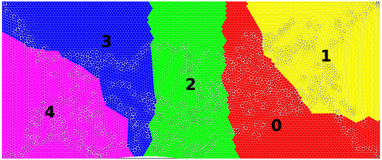
- Mapping onto 5 processors
  - On a complete graph
  - On a part of a 4x2 2D mesh architecture






```
sub 5 0 4 1 5 7
mesh2 4 2
```





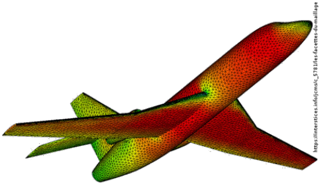
	k5	m4x2(5)
Edge cut	504	561
Edge dilation on m4x2	804	713

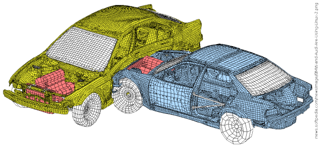
3

Recent advances on large meshes in PaMPA

Context

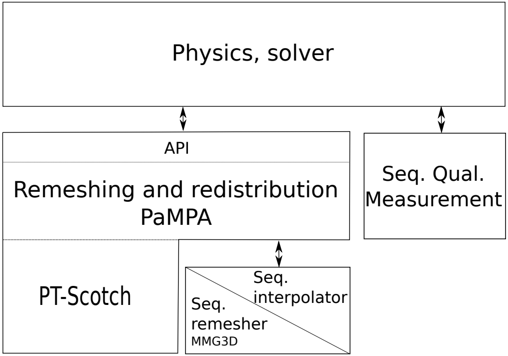
- Space discretization:
  - mesh
- Finite number of points on which values of the problem are computed, e.g.:
  - temperature
  - pressure
  - speed,. . .
- Solution precision depends on mesh quality:
  - need for remeshing
- Problems become bigger and more and more complex
  $\rightarrow$ Use parallel remesher





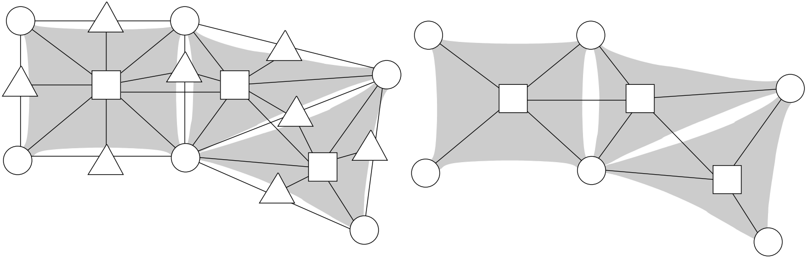
What is PaMPA

- PaMPA: “Parallel Mesh Partitioning and Adaptation”
- Middleware library managing the parallel repartitioning and remeshing of unstructured meshes modeled as interconnected valuated entities
- The user can focus on his/her “core business”:
  - Solver
  - Sequential remesher
    - Coupling with MMG3D provided for tetrahedra



Examples

- The same mesh can lead to different enriched graphs
  - Depending on the requirements of the numerical schemes



Features of version 1.0

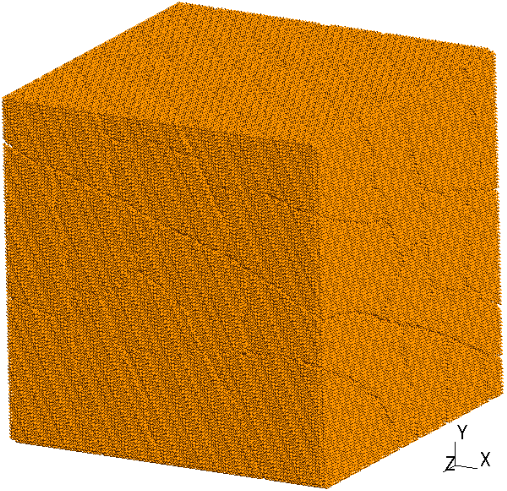
- Overlap greater than 1
- Iterators to loop over entities and sub-entities
- Point-to-point or collective communications
- Parallel I/O
- Parallel partitioning

Work in progress

- Multigrid:
  - described as:
    - several distributed meshes
    - links between distributed meshes
  - Distributed meshes are:
    - provided by the user
    - produced by recursively coarsening meshes by merging elements and removing nodes
  - Partitioning on the coarsest mesh
  - Propagation of the partition to finer levels
  - Load-balancing each level
  - Computing overlap for all the levels according to the coarsest mesh
  - Used in Aerosol software
- Real-time visualisation with ParaView
- Parallel remeshing based on sequential remesher
  - coupled with MMG3D
  - coupling in progress with TetGen

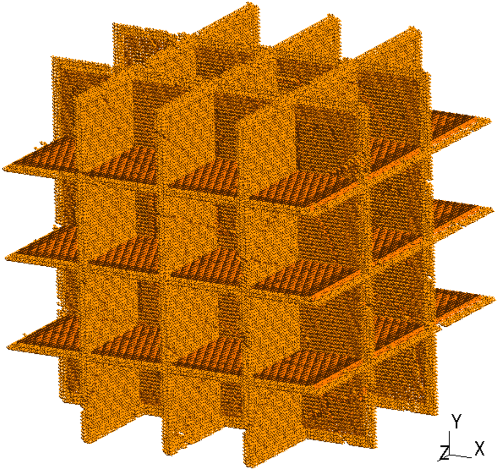
Parallel remeshing: example

- iteration 1



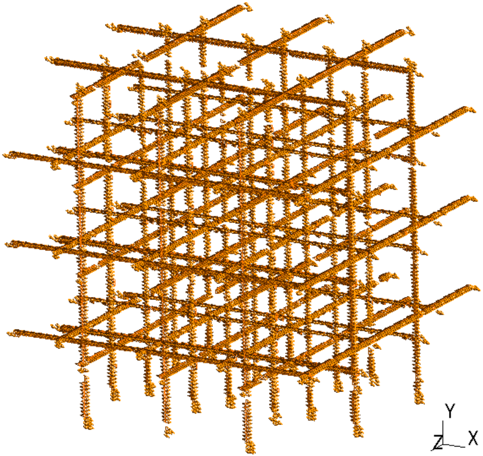
Parallel remeshing: example

- iteration 2



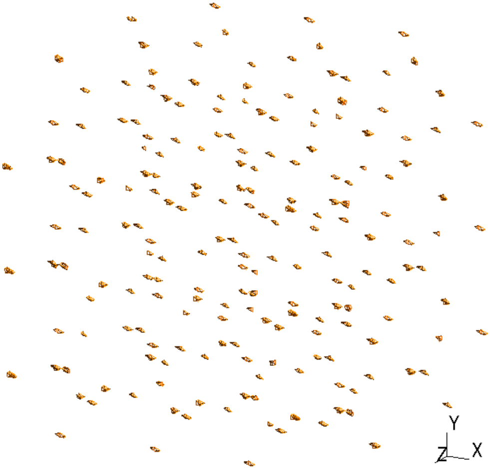
Parallel remeshing: example

- iteration 3



Parallel remeshing: example

- iteration 4

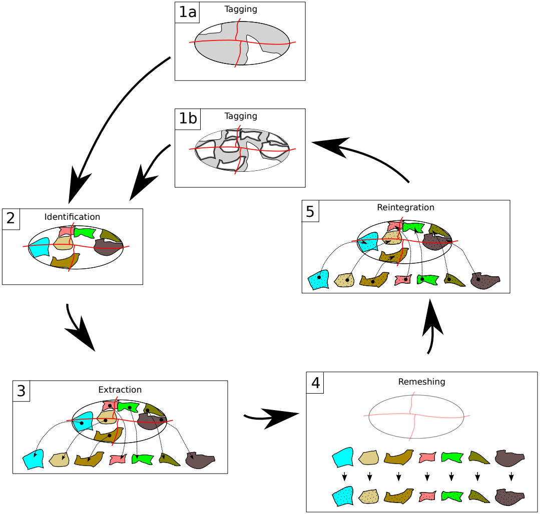


Parallel remeshing: example

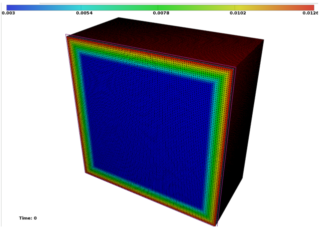
- at the end

Parallel remeshing: global scheme

Iterative process until all tagged elements are remeshed



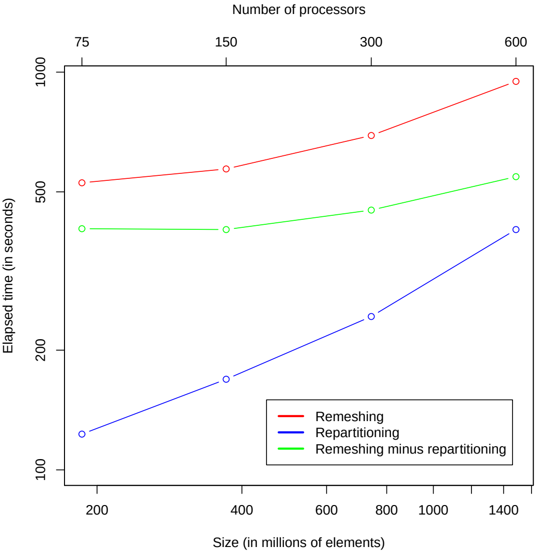
Parallel remeshing: scalability test (1/2)



Cut of the original cube

- PaMPA coupled with MMG3D4
- Domain space (originally a cube with graduated isotropic metric) is duplicated when the number of processor is doubled
- Geometrical partitioning used during zone identification step
- Remeshing time can be studied without repartitioning time, because it will be improved for two reasons:
  - For now, reintegration step only do a local load balancing
  - Repartitioning is from scratch (PT-Scotch v6.1 will take into account the initial partitioning)

Parallel remeshing: scalability test (2/2)



4

Conclusion

Conclusion

- Recent advances on static mapping in Scotch
  - (Disconnected parts of) large architectures can now be represented efficiently in SCOTCH
    - To date, implemented in SCOTCH only, not in PT-SCOTCH
    - Released soon in SCOTCH 6.0.5
  - PT-SCOTCH is planned to perform parallel static mapping starting from branch 6.1
    - Prototype available since the PhD of Sébastien Fourestier, but needs intensive regression testing before release
- Recent advances on distributed mesh computations with PaMPA
  - PaMPA is dedicated to distributed meshes
  - We have devised an efficient scheme for parallel remeshing of very large meshes, which can be coupled with any sequential remesher
  - We can achieve the same quality as sequential remeshers
  - Release of version 1.0 almost finalized
    - Available soon from Inria Gforge
  - Work in progress will be available in version 2.0

This research work was partly funded by *Investissements d'Avenir* (PIA) project ELCI

Conclusion

- Recent advances on static mapping in Scotch
  - (Disconnected parts of) large architectures can now be represented efficiently in SCOTCH
    - To date, implemented in SCOTCH only, not in PT-SCOTCH
    - Released soon in SCOTCH 6.0.5
  - PT-SCOTCH is planned to perform parallel static mapping starting from branch 6.1
    - Prototype available since the PhD of Sébastien Fourestier, but needs intensive regression testing before release
- Recent advances on distributed mesh computations with PaMPA
  - PaMPA is dedicated to distributed meshes
  - We have devised an efficient scheme for parallel remeshing of very large meshes, which can be coupled with any sequential remesher
  - We can achieve the same quality as sequential remeshers
  - Release of version 1.0 almost finalized
    - Available soon from Inria Gforge
  - Work in progress will be available in version 2.0

This research work was partly funded by *Investissements d'Avenir* (PIA) project ELCI

Thank you for your attention!