Static and dynamic processing of discrete data structures

C2S@Exa

François Pellegrini

Contents

1

Context

Purpose

- Means and ends: discrete data structures:
  - Graphs
    - "Mesh-like", not "social network-like"
  - Meshes
- Create locality ($\rightarrow$ pole 3)
  - Compute efficient partitions / mappings of graphs / meshes
  - Improve cache locality by adequate local numbering
- Enable PDE solver writers to focus on their "core business" ($\rightarrow$ pole 2)
  - Hide all the MPI "plumbing work"
  - Contribute to efficient parallelization
    - At the node level

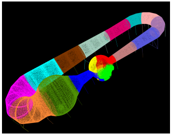
People and support

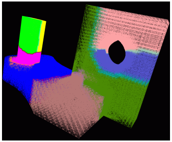
- F. Pellegrini
  - Formerly Bacchus team, now TADaaM since 01/01/2015 ("*Topology-Aware System-Scale Data Management for High-Perfor mance Computing Applications*")
- S. Fourestier
  - PhD defended on 20/06/2013
  - Left project on 11/2014
- C. Lachat
  - PhD defended on 13/12/2013
  - ADT "El Gaucho" from 01/10/2012 to 30/09/2014
  - PIA ELCI ("Bull") ASAP

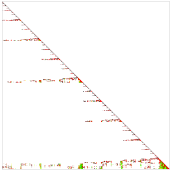
2

Works

The Scotch project







- Toolbox of graph partitioning methods, which can be used in numerous contexts
- Sequential Scotch library (v6.0)
  - Graph and mesh partitioning
  - Static mapping (edge dilation)
  - Graph and mesh reordering
  - Clustering
  - Graph repartitioning and remapping
- Parallel PT-Scotch library (v6.0)
  - Graph partitioning (edge)
  - Static mapping (edge dilation)
  - Graph reordering
  - *Graph repartitioning, remapping* (v6.1)

Three challenges

- Scalability
  - How will the algorithms behave for large numbers of processing elements?
- Heterogeneity
  - How will the architecture of the target machine impact performance?
- Asynchronicity
  - Will our algorithms still be able to rely on fast collective communication?

Design constraints

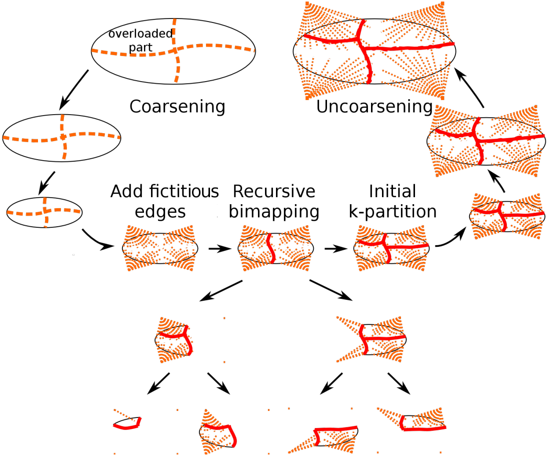
- Parallel algorithms have to be carefully designed
  - Algorithms for distributed memory machines
  - Preserve independence between the number of parts k and the number of processing elements P on which algorithms are to be executed
  - Algorithms must be “quasi-linear” in $|V|$ and/or $|E|$
    - Constants should be kept small
- Data structures must be scalable
  - In $|V|$ and/or $|E|$: graph data must not be duplicated
  - In P and k: arrays in $k|V|$, k^2, kP, $P|V|$ or P^2 are forbidden

Architectural considerations matter

- High-end machines comprise very large numbers of processing units, and will possess NUMA / heterogeneous architectures
- Impacts on our research:
  - Target architecture has to be taken into account
  - Do static mapping and not only graph partitioning
    - Reduces number of neighbors and improves communication locality, at the expense of slight increase in message sizes

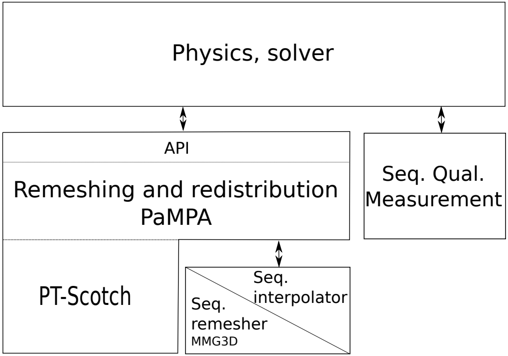
Graph partitioning with fixed vertices

- Used to model repartitioning / remapping
- Algorithms designed and implemented in Scotch 6.0
  - Parallel version scheduled for Scotch 6.1



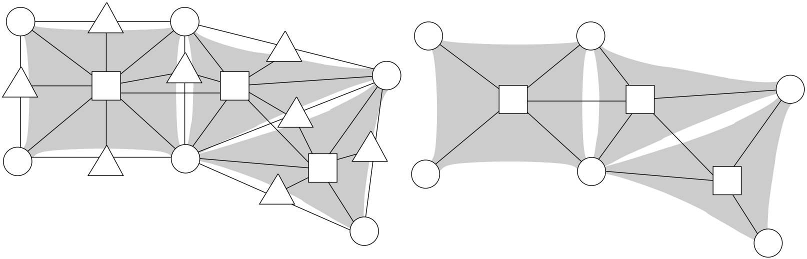
PaMPA

- PaMPA: “Parallel Mesh Partitioning and Adaptation”
- 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 tetrahedral remeshing



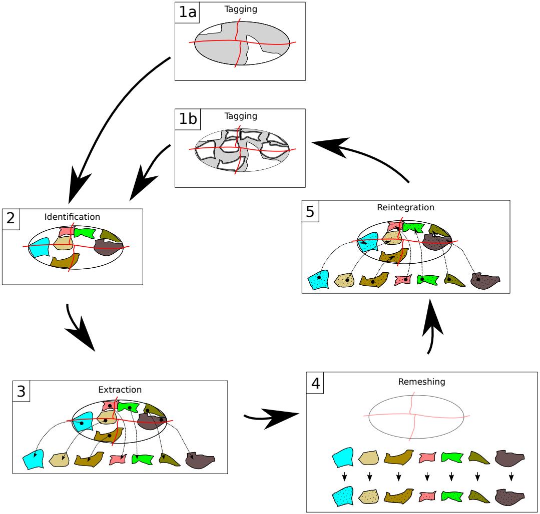
Data structures for representing distributed meshes

- Based on the notion of “enriched graph”
  - Labeled undirected loopless graph
  - Sub-labeling (for separating e.g. boundary and inner faces)
  - With adequate local and global vertex numbering



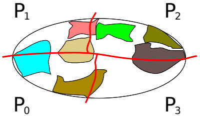
Framework for parallel remeshing

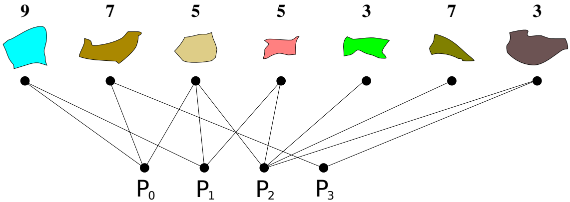
- Iterative process until all tagged elements are remeshed



Extraction of zones to be remeshed

- Two criteria must be considered:
  - Load-balance according to the remesher workload
  - Minimize communication
- Requires partitioning with fixed vertices





3

Results to date

Features of PaMPA 1.0

- Description of distributed unstructured meshes
  - With values attached to enriched graph vertices
- Data exchange with overlap of any width
  - Point-to-point or collective communication
- Iterators to loop over entities and sub-entities
- Parallel partitioning and redistribution
  - Renumbering for cache miss reduction
- Parallel mesh I/O
- Parallel remeshing based on a sequential remesher

Example: Solve system (Jacobi) (1/2)

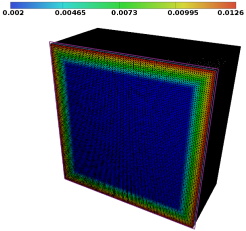
```
UaPrec = 0.          ! Suppose A = L + D + U, system to solve : A x = b
CALL PAMPAF_dmeshItInit(dm, ENTITY_NODE, ENTITY_NODE, it_ngb, ierr)
DO irelax = 1, Nrelax
  res       = 0.
  CALL PAMPAF_dmeshItInitStart(dm, ENTITY_NODE, PAMPAF_VERT_BOUNDARY, it_vrt,  ierr)
  DO WHILE (PAMPAF_itHasMore(it_vrt))
    is = PAMPAF_itCurEnttVertNum(it_vrt)
    CALL PAMPAF_dmeshMatLineData(dm, ENTITY_NODE, is, I1, I1Fin, ierr)
    CALL PAMPAF_itStart( it_ngb,  is,  ierr)
    res0     = RHS(is)                        ! res0  = b

    iv = i1
    DO WHILE (PAMPAF_itHasMore(it_ngb))
      js = PAMPAF_itCurEnttVertNum(it_ngb)
      PAMPAF_itNext(it_ngb)
      res0  = res0 - MatCSR%Vals(iv) * UaPrec(js) !res0  = b - (L + U) x^n
      iv = iv + 1
    END DO
    Ua(is)  = res0 / MatCSR%Diag(is)          !x^n+1 = ( b - (L + U) x^n )/D
    PAMPAF_itNext(it_vrt)
  END DO

  CALL PAMPAF_dmeshHaloValueAsync(dm, ENTITY_NODE, PAMPA_TAG_SOL, req, ierr)
```

Example: Solve system (Jacobi) (2/2)

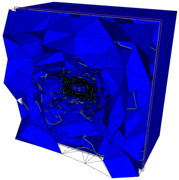
```
  CALL PAMPAF_dmeshItInitStart(dm, ENTITY_NODE, PAMPAF_VERT_INTERNAL, it_vrt, ierr)
  DO WHILE (PAMPAF_itHasMore(it_vrt))
    is = PAMPAF_itCurEnttVertNum(it_vrt)
    CALL PAMPAF_dmeshMatLineData(dm, ENTITY_NODE, is, I1, I1Fin, ierr)
    CALL PAMPAF_itStart( it_ngb, is, ierr)
    res0     = RHS(is)                             ! res0  = b

    iv = i1
    DO WHILE (PAMPAF_itHasMore(it_ngb))
      js = PAMPAF_itCurEnttVertNum(it_ngb)
      PAMPAF_itNext(it_ngb)
      res0   = res0 - MatCSR%Vals(iv) * UaPrec(js) !res0  = b - (L + U) x^n
      iv = iv + 1
    END DO
    Ua(is)   = res0 / MatCSR%Diag(is)              !x^n+1 = ( b - (L + U) x^n )/D
    PAMPAF_itNext(it_vrt)
  END DO

  CALL PAMPAF_dmeshHaloWait(req, ierr)

  UaPrec     = Ua
END DO ! end loop on irelax
```

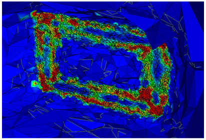
Parallel remeshing test cases

Isotropic mesh

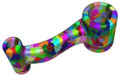


Anisotropic mesh





Parallel remeshing on an isotropic mesh



	PaMPA-MMG3D	
	on 240 processors	on 480 processors
Initial number of elements	27 044 943	
Used memory (kb)	651 185 792	542 832 960
Elapsed time	00h34m59s	00h29m03s
Elapsed time × number procs	139h56m	232h24m
Final number of elements	609 671 387	612 426 645
Smallest edge length	0.2911	0.1852
Largest edge length	8.3451	7.3611
Worst element quality	335.7041	190.4122
Element quality between 1 and 2	98.92%	98.97%
Edge length between 0.71 and 1.41	97.20%	97.39%

Industrialization of PaMPA

- Several Inria teams and groups already use PaMPA
  - Bacchus & Cagire: AeroSol solver framework for fluid dynamics
  - Castor: Plato prototype solver
  - Num3sis: Prototype interfacing work
- Industrial interest in parallel remeshing
  - CD Adapco, Dassault, Airbus, etc.
- Project in progress with involvement of DTI
  - Release as GPL'd free software
  - Creation of a community
  - Creation of a start-up?

4

Perspectives

Perspectives

- PaMPA is now ready for solver software development
  - Testbed for the tuning of TADaaM works on mapping
- Development of (PT-)Scotch is going on
  - PhD on multi-constraint partitioning started on 08/12/2014 (CEA/DAM funding, as “Projet Phare”)
  - Works with HiePACS on specific partitioning and ordering algorithms
- PIA ELCI (“Bull”)
  - Scalability studies of PT-Scotch and PaMPA
  - Man-powered by C. Lachat
- H2020 “AltExa” project submitted
  - With Inria teams HiePACS, ROMA and KIT, TU. Vienna, U. Utrecht, ETH Zürich