Mesh Partitioning and Load Balancing

Sabina Rips

Contents:

- Introduction / Motivation
- Goals of Load Balancing
- Structures
- Tools

Flow Chart of a Parallel (Dynamic) Application

Partitioning of the initial mesh

↓

Computation

Iteration step
Data exchange

+ Meshrefinement and/or - movement

↓

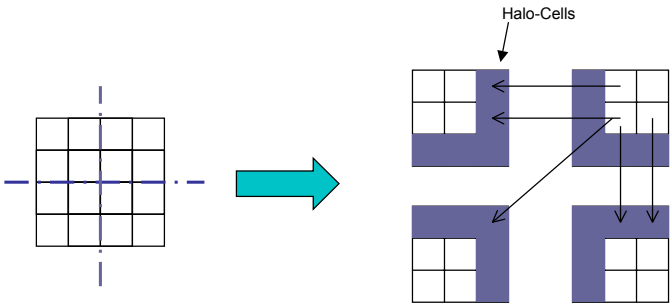
Imbalance

↘ Efficiency loss

↓

Loadredistribution

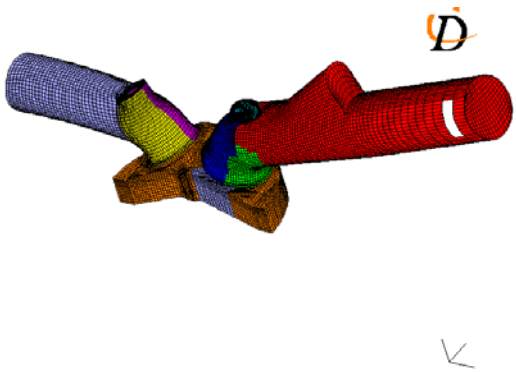
Structure of a partitioned mesh



Global mesh

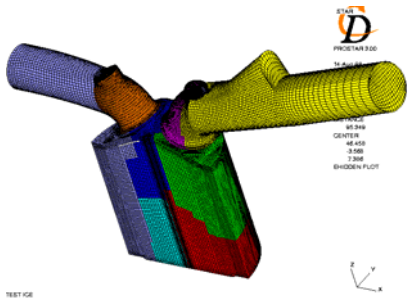
Partitioned mesh

Application Example from HPS-ICE Project



Example: Model DBAG, 360° angle of crankshaft, 148.342 mesh cells

Application Example from HPS-ICE Project



At 540° angle of crankshaft: 273.738 mesh cells

Load Imbalance

Between 360° and 540°: Increase of meshsize by a factor of 2

Without Loadbalancing:

2 Processors are sharing the additional load

→ Significant loss of efficiency

Goals of Loadbalancing

1. Load Balance

2. Minimal communication costs
   - minimize the number of boundary cells, created by the mesh partitioner
   - minimize the number of processors, one has to communicate with

3. Minimal migration costs

Steps of a Loadbalancing Procedure

2 steps of load distribution:

1. Decision:
   - **how many** and
   - **which** cells should be moved
   - **where**

   ⇧ Load balancing algorithms (→ Tools)

2. Migration:

   move the mesh cells to the selected processors

   ⇧ has to be done by the application program

How Do Load Balancer Operate?

How can I handle the problem, using [known] methods/algorithms ?

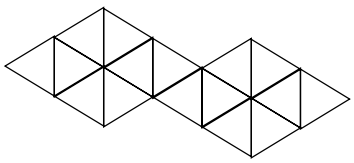
⇧ Mapping of the problem to known structures:

Graphs

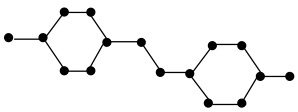
Nodes of the graph ↔ computation costs
Edges of the graph ↔ communication costs

Mapping Mesh ⇨ Graph

Mesh



Graph

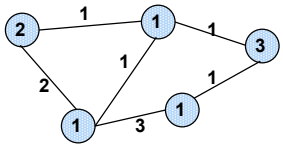


Graph

Further information represented in a graph:

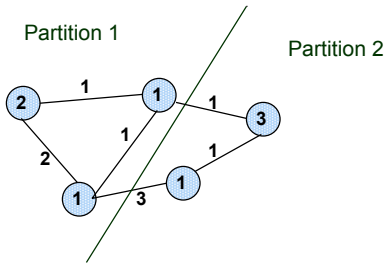
- Computation costs ↔ Weighting of nodes
- Communication costs ↔ Weighting of edges

Example:



Goals of Graph Algorithms

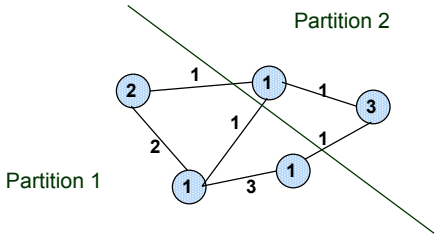
1. Equal number of nodes within each partition
   (*weighted: equal sum of node weights within each partition)*



load balance: 100 %
(weight of cut edges: 4)

Goals of Graph Algorithms

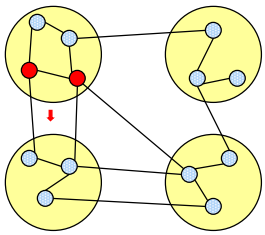
2. Minimal number of cut edges
(*weighted: minimal sum of cut edges weight)*



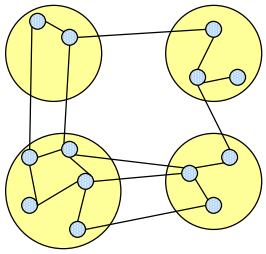
load balance: 100 %
weight of cut edges: 3

Goals of Graph Algorithms

2a. minimal processor degree
(node degree: number of neighbours)



Max. processor degree: 3



Max. processor degree: 2

Goals of Graph Algorithms

Task for non weighted graphs:

find a **decomposition** of the graph,

- with preferably the **same number of nodes** within each partition
- with a **minimal number of cut edges** between the partitions

⇧ **NP-complete** problem

(it cannot be solved in reasonable time in case of bigger problems)

⇧ use of **heuristics**,

which give a solution near the optimal one

Tools

- **Jostle**: (Chris Walshaw, University of Greenwich)
  - initial graph partitioning
  - dynamic load distribution
  - serial and parallel
  - http://www.gre.ac.uk/jostle
  - HLRS: http://www.hlrs.de/organization/par/services/tools/loadbalancer/jostle.html
- **Metis:** (George Karypis, University of Minnesota)
  - multilevel partitioning algorithms
  - initial partitioning
  - serial
  - (**parMetis**: parallel version)
  - http://www.cs.umn.edu/~karypis/metis/metis.html
  - HLRS: http://www.hlrs.de/organization/par/services/tools/loadbalancer/metis.html

Jostle

- operates with an already **partitioned and distributed mesh**
- in the initial case: at first partitioning of the mesh using simple methods
- operates in general **parallel**
  - no need of gathering the data on one processor
  - partitioning effort is spread to the processors
- migrates boundary layers to the neighbouring processors

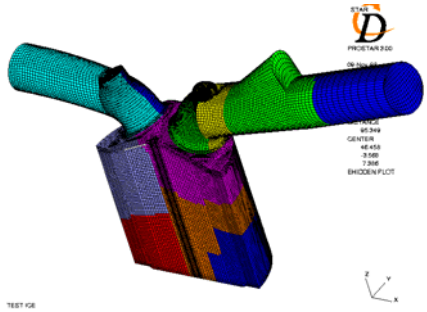
  ⇧ **minimization of cell migration**

Jostle

- available as stand-alone program (only serial version) and as procedure call (C und FORTRAN)
- different graph formats possible
- controlling the behaviour is possible by using parameters (e. g. load imbalance)
- Call (C):

```
void pjostle(int *nnodes, int *offset, int *core, int *halo,
             int *index, int *degree, int *node_wt, int *partition,
             int *local_nedges, int *edges, int *edge_wt,
             int *network, int *processor_wt, int *output_level,
             int *dimension, double *coords);
```

Jostle (serial)



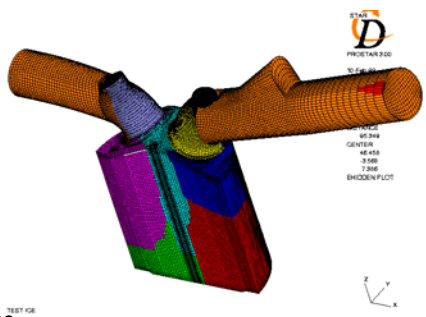
Balance: ≈100 %

Cut edges: 10.509

Run time (SGI R4600): 58,9 sec

8 Partitions, 273.738 Cells

Jostle (parallel)



Balance: ≈ 100 %

Cut edges: 10.638

Run time (SGI R4600):532,2 sec

(in case of running 8 MPI-Processes)

8 Partitions, 273.738 Cells

Metis

- serial graph partitioning
- available as standalone program and as procedure call (C und FORTRAN)
- fixed data format
- consecutive cell numbering required

- Call(C):

```
METIS_PartGraphRecursive(int *n, idxtype *xadj, idxtype *adjncy,
    idxtype *vwgt, idxtype *adjwgt, int *wgtflag, int *numflag,
    int *nparts, int *options, int *edgecut, idxtype *part);
```


ParMetis

- static & dynamic partitioning
- consecutive cell numbering required
- no settings possible

- Call for graph repartitioning (C):

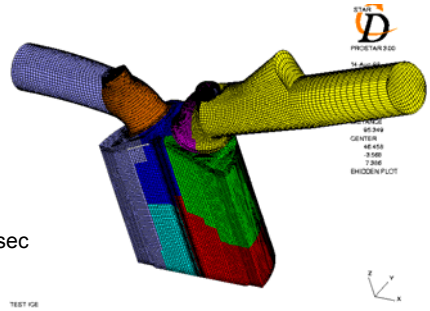
```
ParMETIS_RepartLDiffusion(idxtype *vtxdist, idxtype *xadj,
    idxtype *adjncy, idxtype *vwgt, idxtype *adjwgt, int *wgtflag,
    int *numflag, int *options, int *edgecut, idxtype *part,
    MPI_Comm *comm);
```

Metis

Balance: 100 %

Cut edges: 10.007

Run time (SGI R4600): 34,8 sec



8 Partitions, 273.738 Cells

Tools: Comparison

Tool	Cut edges	Run time (sec)	+	-
Jostle Ser. Par.	 10.509 10.638	 58,9 532,2	• serial & **parallel** • arbitrary cell numbering • various user settings	• worse results
Metis	10.007	34,8	• fast • good results	• serial

Summary

Strategy:

1. At first:

   initial partitioning:

   a. choose appropriate serial partitioner (**Jostle, Metis**)

   b. distribute the cells to the processors

2. In case of an existing load imbalance:

   dynamic load balancing:

   a. cost-benefit-calculation

   b. in case of benefit:

   - when migration tool available:

     ⇧ parallel LB ⇧ **Jostle**

   - else:
     - gather cell info on a particular processor
     - proceed like in case of initial partitioning

Overview