

Graph repartitioning with *Scotch* and other on-going work

23/11/2010

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE




centre de recherche
BORDEAUX - SUD-OUEST

Sébastien Fourestier



INRIA-UIUC joint laboratory

Summary of the talk

- The  project
- Parallel partitioning: weak scalability results
- Parallel static mapping: overview
- Dynamic repartitioning: early results



The project

- Devise robust parallel graph partitioning/mapping methods
 - Initial roadmap: should handle graphs of more than a billion vertices distributed across one thousand processors
- Provide graph repartitioning methods
- Improve sequential graph partitioning methods if possible
- Investigate alternate graph models (meshes/hypergraphs)
- Provide a software toolbox for scientific applications
 -  sequential software tools
 -  parallel software tools

DONE !



Parallel partitioning: weak scalability results

- Since version 5.1.10, Scotch is now fully 64-bit
 - Can handle graphs above 2 billion vertices
 - But less than 2 billion edges by processing element, because of MPI interface limitations
 - Not really a problem for us on many-core machines
- Several weak scalability experiments performed
 - Up to 8192 processors on the Hera machine at LLNL, with graphs above 2 billion edges
 - Up to 30k cores on BG/L at LLNL



Architectural considerations matter

- Upcoming machines will 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
 - Static mapping and not only graph partitioning
 - Reduces number of neighbors and improves communication locality, at the expense of slight increases in message sizes
 - Dynamic repartitioning capabilities are mandatory

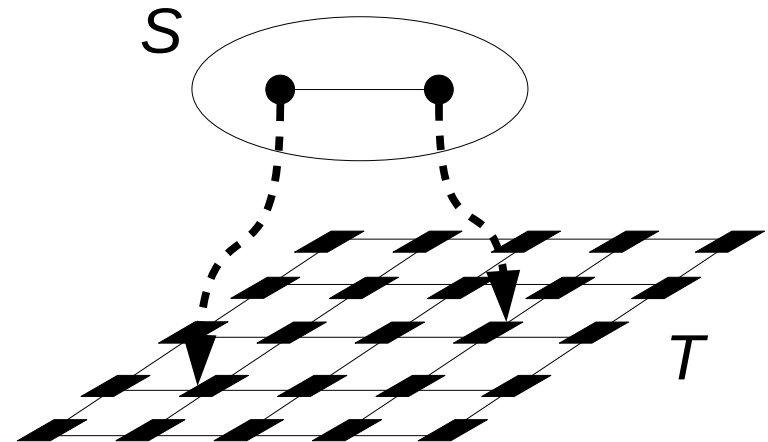


Static mapping

- Compute a mapping of $V(S)$ and $E(S)$ of source graph S to $V(T)$ and $E(T)$ of target architecture graph T , respectively
- Communication cost function accounts for distance

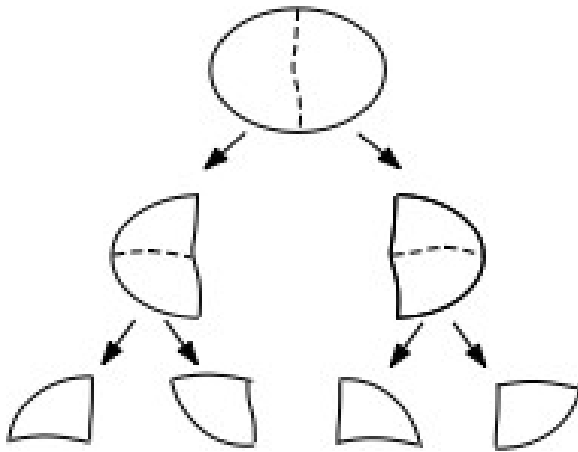
$$f_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{e_S \in E(S)} w(e_S) |\rho_{S,T}(e_S)|$$

- Static mapping features are already present in the sequential **Scotch** library
 - We have to go parallel

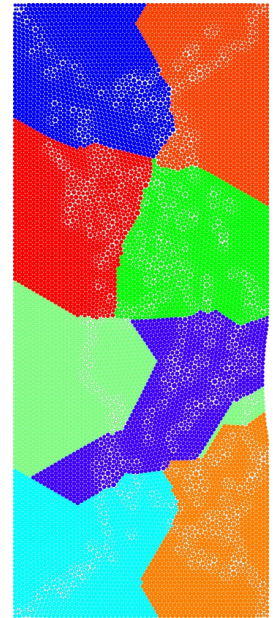
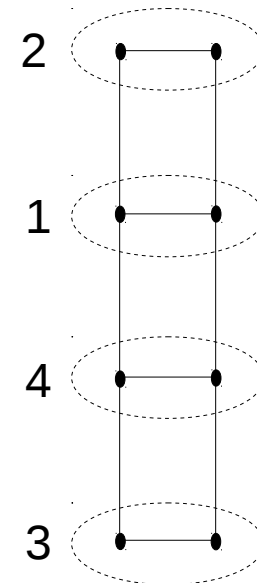
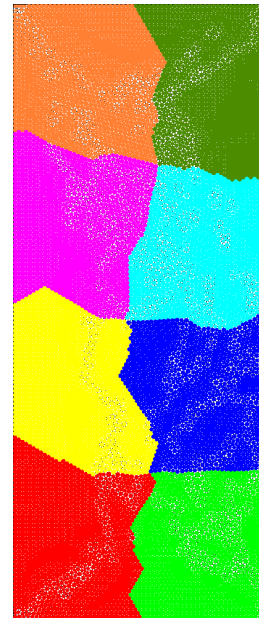


Parallel static mapping

- Recursive bi-mapping cannot be transposed in parallel
 - All subgraphs at some level are supposed to be processed simultaneously for parallel efficiency
 - Yet, ignoring decisions in neighboring subgraphs can lead to “twists”



- Sequential processing only!



New roadmap

- To be able to map graphs of about a trillion vertices spread across a million processing elements
 - Use direct k-way static mapping algorithms
 - Focus on scalability problems related to the large number of processors
 - Asynchronous algorithms to reduce latency induced by collective communication
- Parallel dynamic repartitioning capabilities is mandatory



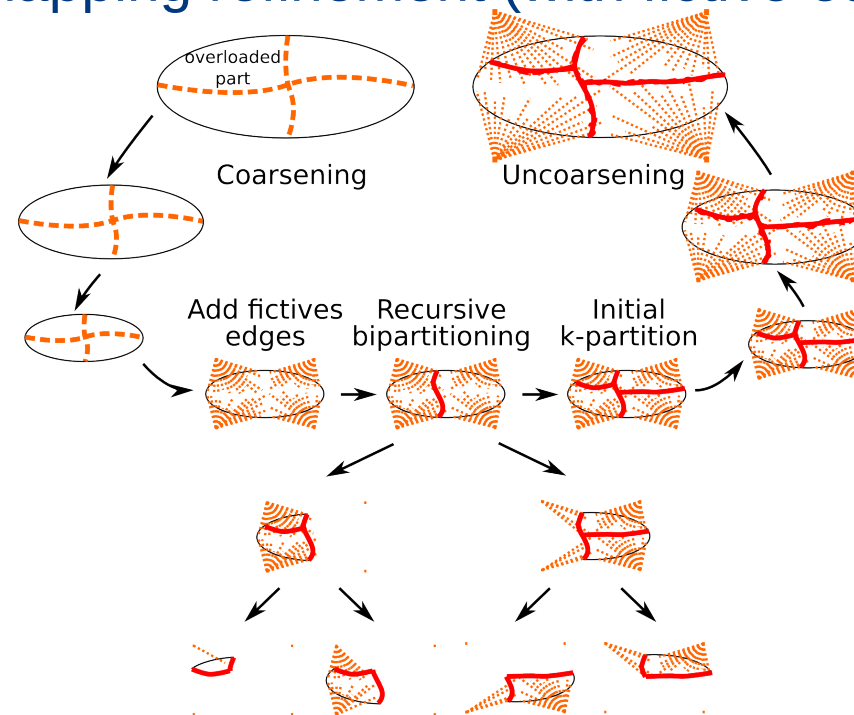
Dynamic repartitioning

- Recompute a balanced partition while minimizing the number of migrated vertices
- Modeling of migration costs by fictive edges that induce an increase of the cut size when a vertex changes of part
- Use of fractional migration costs in graphs with integer weights by multiplication of the original edge loads
- Adaptation of the existing algorithms to account for these extra edges
 - Recursive bipartitioning, k-way partitioning, multilevel framework, FM heuristic, diffusion-based method



Dynamic repartitioning

- Multilevel framework adapted for repartitioning
 - Coarsening mates only vertices belonging to the same part
 - Initial mapping by sequential recursive bi-mapping (with fictive edges)
 - K-way mapping refinement (with fictive edges)



Dynamic repartitioning – Test graphs

- Graphs considered during experimentation:

Graph	Vertex number	Edge number	Average degree
altr4	26089	163038	12.5
oilpan	73752	1761718	47.8
bmw32	227362	5030634	48.7
audikw1	943695	38354076	81.3
conesphere1m	1055039	8023236	15.2
coupole8000	1768161	41656975	47.1

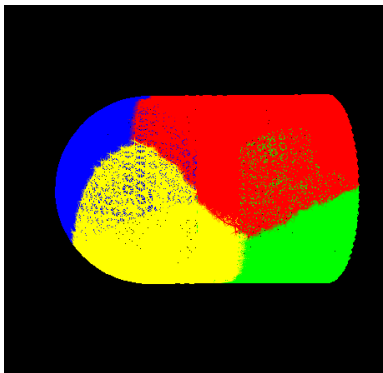
- Specificities:
 - oilpan, bmw32 and coupole8000: same characteristics, with increasing graph size
 - audikw1: lots of edges, highest degree
 - conesphere1m: 1 million vertices



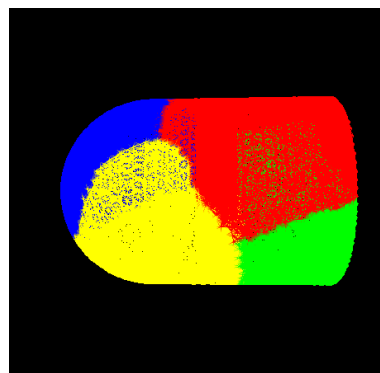
Dynamic repartitioning – Experimentation protocol

- Protocol:
 - Original partition:
 - 4 parts
 - Vertex loads are equal to 1
 - First vertex load changes from 1 to total vertex load / 10
 - Scotch runs on 1 processor, ParMetis on 2 processors
 - Migration cost from 0.01 to 50

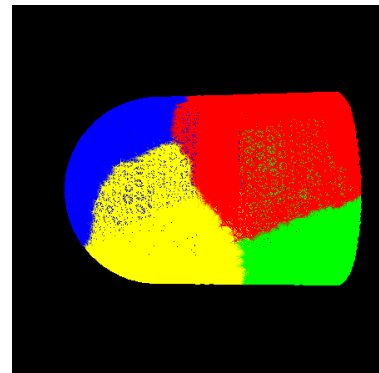
0.1



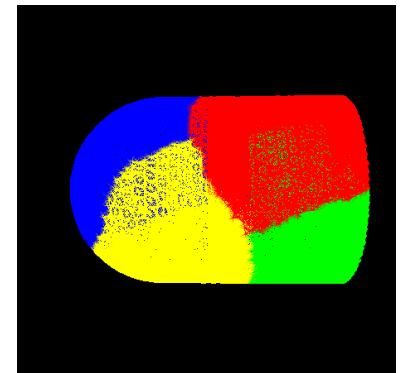
1



10

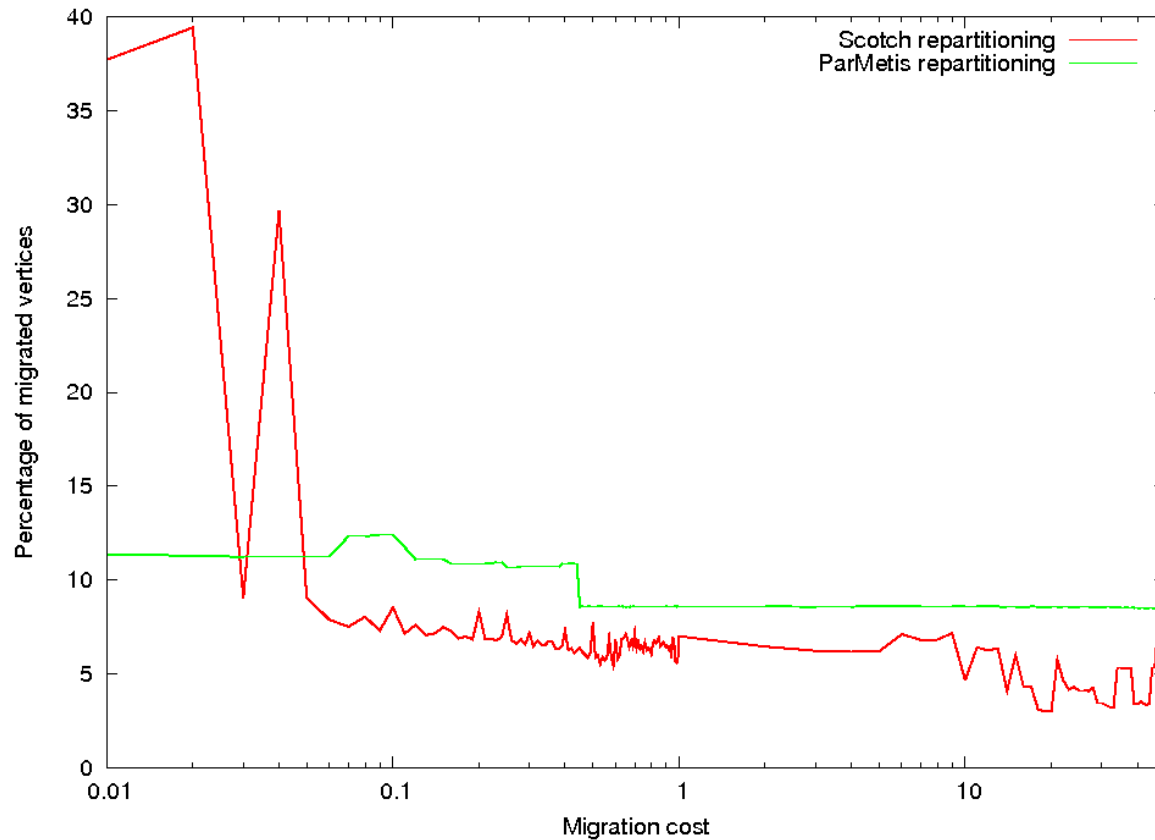


original partition



Dynamic repartitioning – Results with conesphere1m

ParMetis vs Scotch: Percentage of migrated vertices

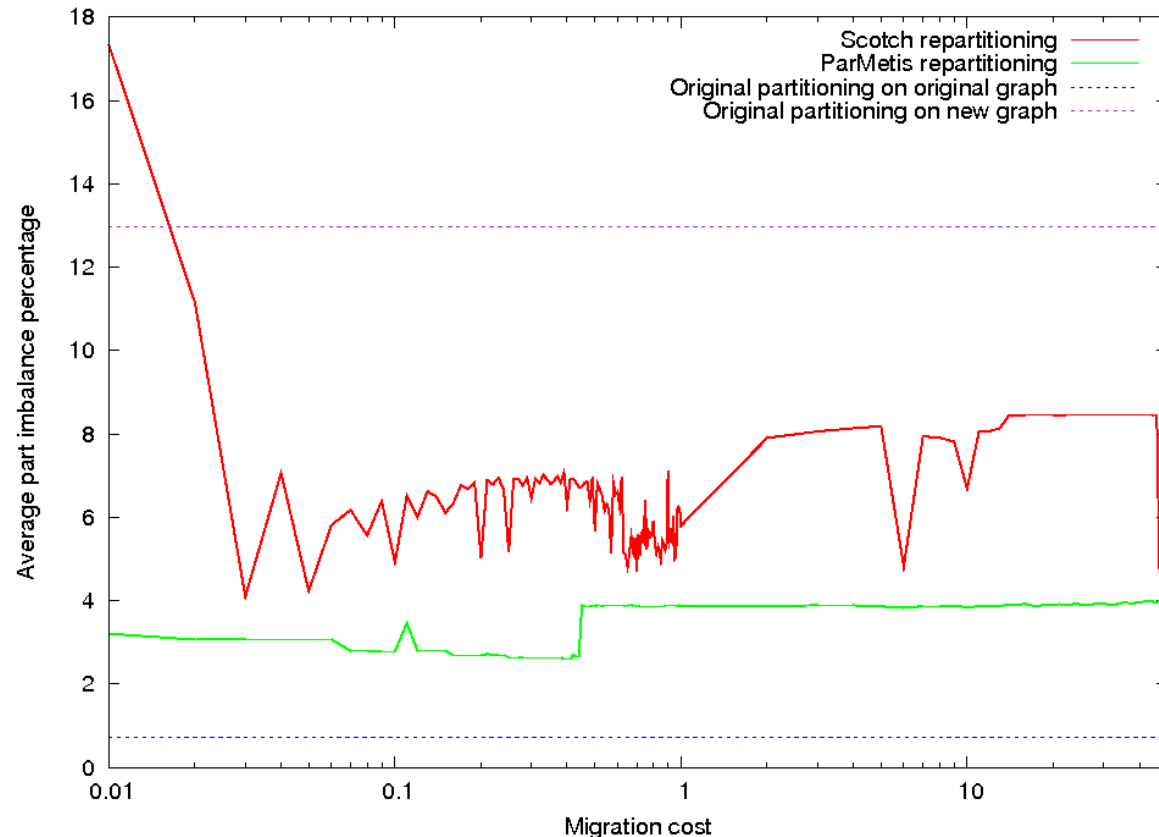


- Scotch is more sensitive to the migration cost parameter



Dynamic repartitioning – Results with conesphere1m

ParMetis vs Scotch: Average part imbalance percentage

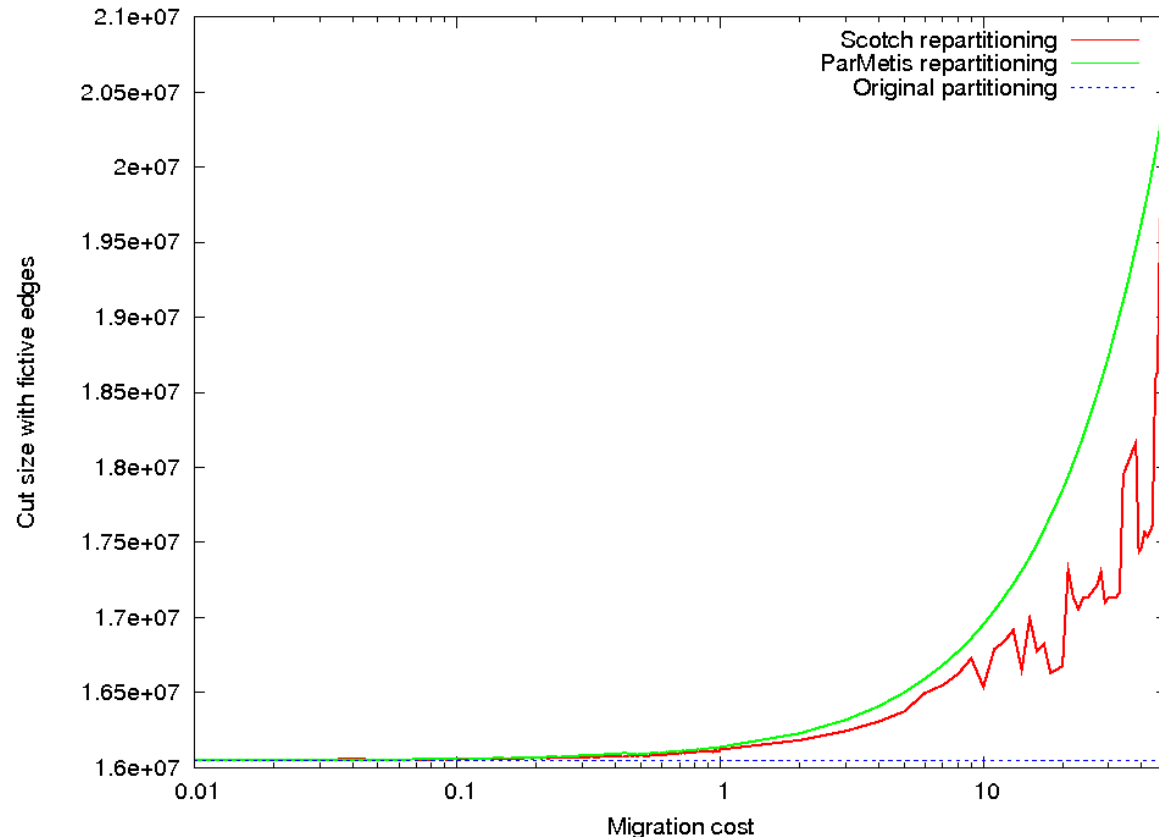


- Both give consistent results
- ParMetis keeps a better balance (with our configuration)



Dynamic repartitioning – Results with conesphere1m

ParMetis vs Scotch: Cut size with fictive edges

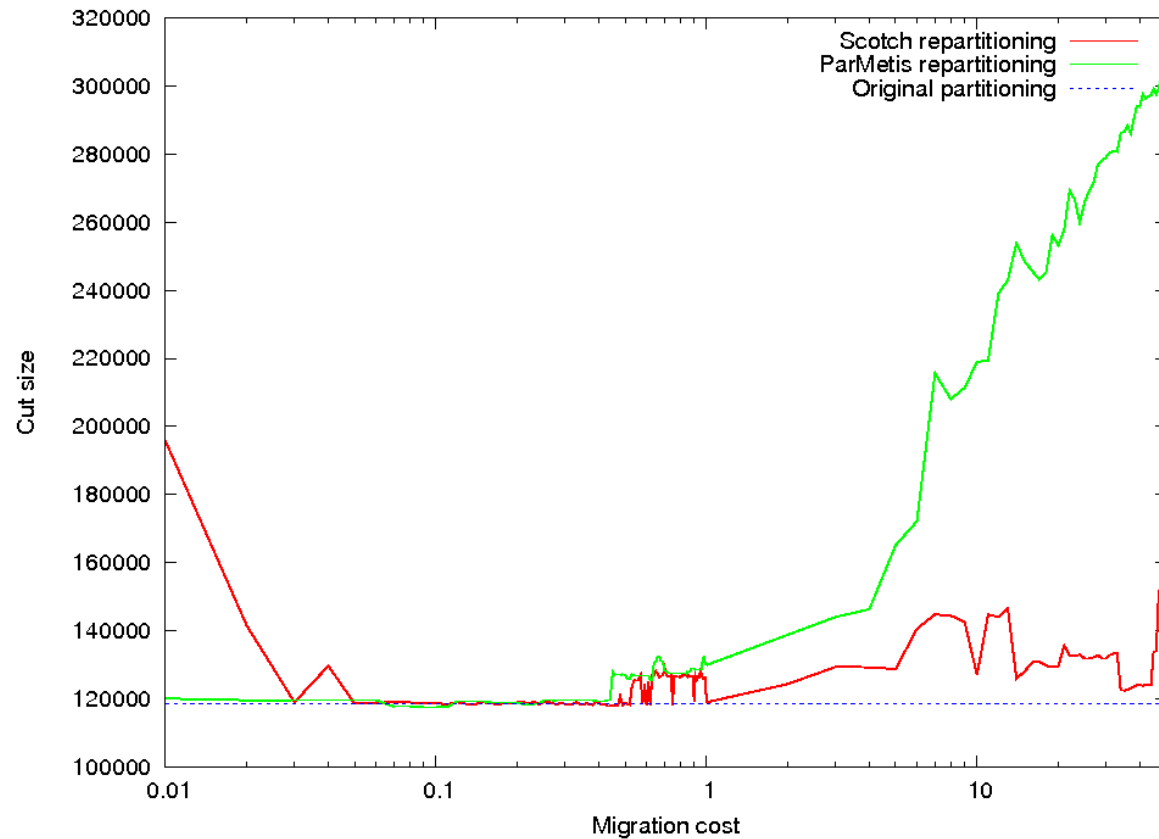


- Better cut with Scotch repartitioning
- This is consistent: lower constraint on part imbalance



Dynamic repartitioning – Results with conesphere1m

ParMetis vs Scotch: Cut size

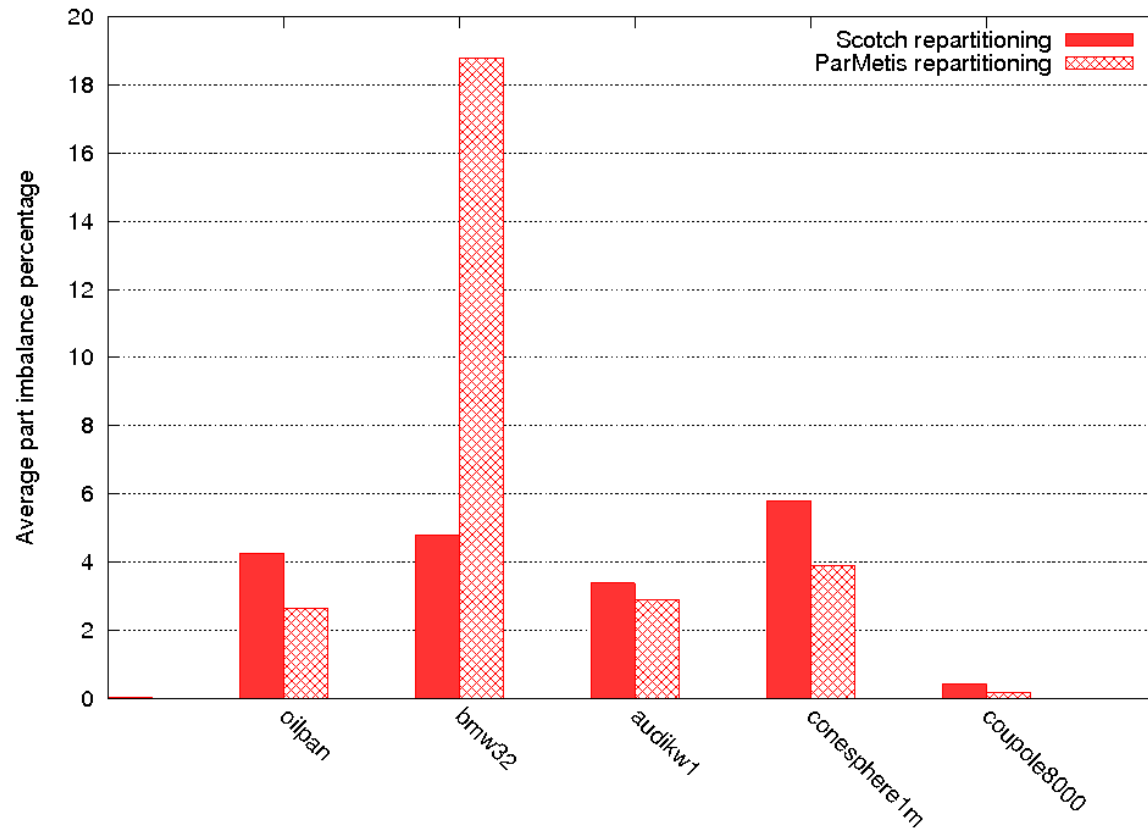


- Scotch keeps a good cut with high migration costs
→ Better precision



Dynamic repartitioning — Results with mig. cost = 1

ParMetis vs Scotch: Average part imbalance

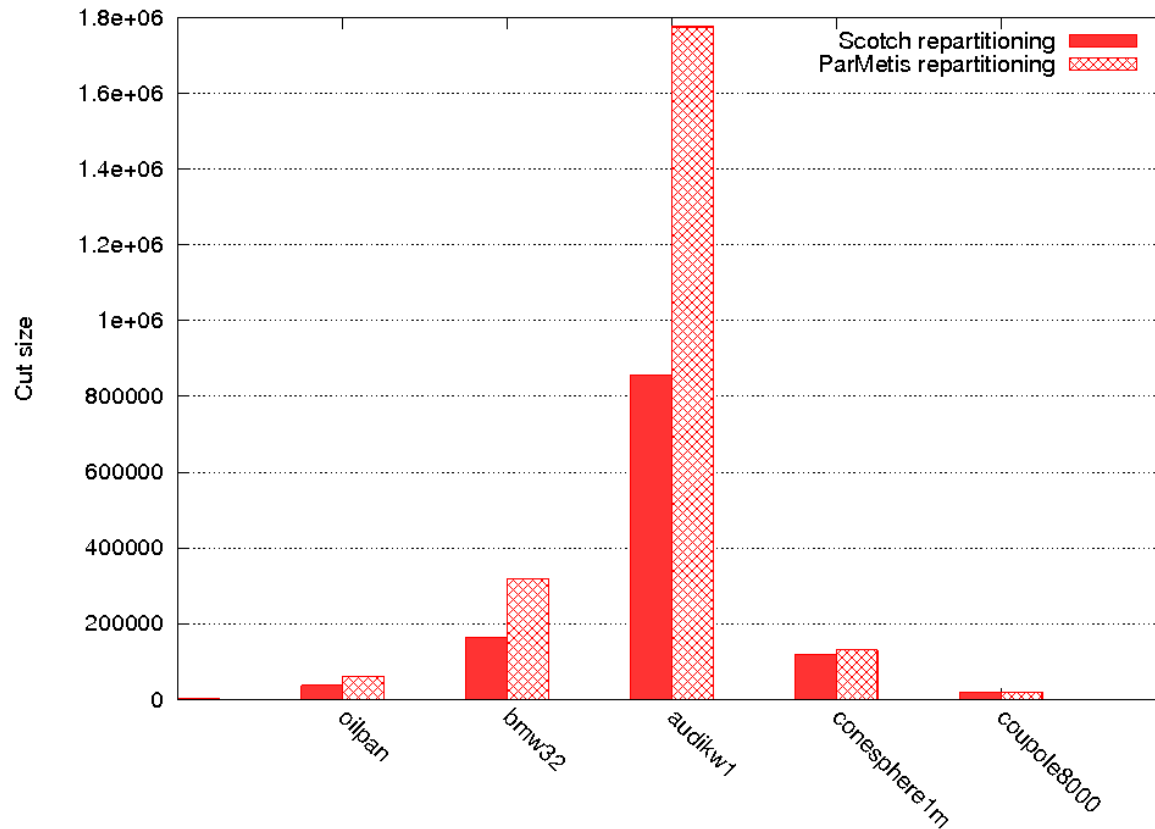


- In average (without bmw32), ParMetis brings a balance 36.6 % better than Scotch (st. deviation: 18.9 %)



Dynamic repartitioning — Results with mig. cost = 1


ParMetis vs Scotch: Cut size



- In average, Scotch brings a cut size 29.8 % better than ParMetis (st. deviation: 22.8 %)



On-going work

- Next steps
 - Efficient parallel methods to refine k-partitions
 - Parallel dynamic repartitioning
 - Dynamic repartitioning with fixed vertices
 - Parallel hypergraph partitioning?
 - Only if gains can be expected over existing works
- Move upwards to application mesh models
- Parallel adaptive remeshing [work with C. Dobrzynski]
 - Take into account the numerical stability of the problem being studied
 - Take advantage of the work done in  on distributed adaptive graphs



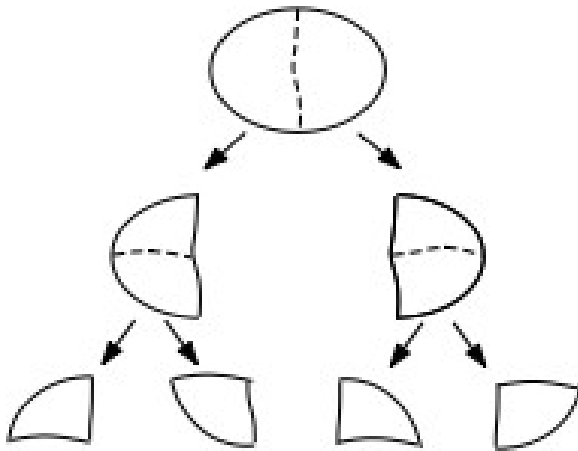
INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



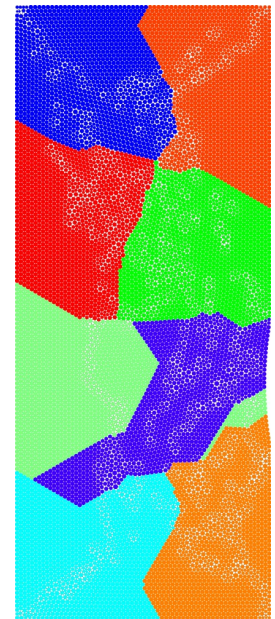
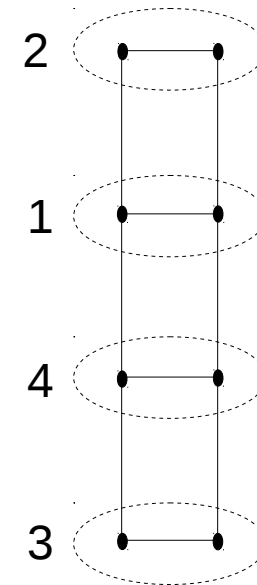
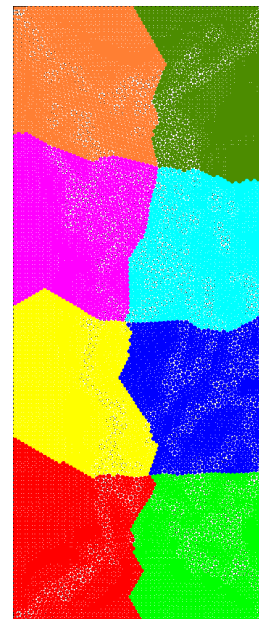
centre de recherche
BORDEAUX - SUD-OUEST

Parallel static mapping

- Recursive bi-mapping cannot be transposed in parallel
 - All subgraphs at some level are supposed to be processed simultaneously for parallel efficiency
 - Yet, ignoring decisions in neighboring subgraphs can lead to “twists”



- Sequential processing only!



Jug of the Danaïdes

- Sketch of the algorithm

