Current challenges for parallel graph (re)partitioning and (re)mapping

Cédric Chevalier, Sébastien Fourestier, Jun-Ho Her, François Pellegrini

EQUIPE PROJET
BACCHUS
Bordeaux
Sud-Ouest

25/07/2012

Outline of the talk

- Graph partitioning and nested dissection
- The 

project and roadmap
- The multilevel framework and its parallelization
- Three challenges for the next roadmap
- Where we are now...

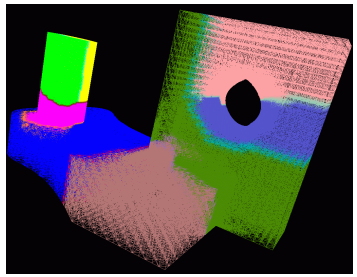
Graph partitioning

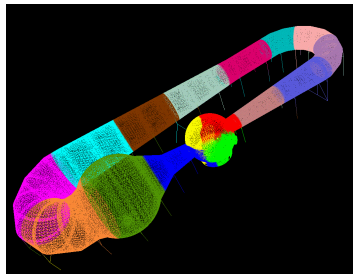
Graph partitioning (1)

- Graph partitioning is an ubiquitous technique which has proven useful in a wide number of application fields
  - Used to model domain-dependent optimization problems
  - “Good solutions” take the form of partitions which minimize vertex or edge cuts, while balancing the weight of graph parts
- NP-hard problem in the general case
- Many algorithms have been proposed in the literature :
  - Graph algorithms, evolutionary algorithms, spectral methods, linear optimization methods, …

Graph partitioning (2)

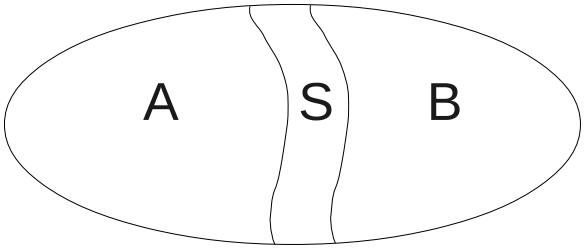
- Two main problems for our team :
  - Sparse matrix ordering for direct methods
  - Domain decomposition for iterative methods
- These problems can be modeled as graph partitioning problems on the adjacency graph of symmetric positive-definite matrices
  - Edge separator problem for domain decomposition
  - Vertex separator problem for sparse matrix ordering by nested dissection

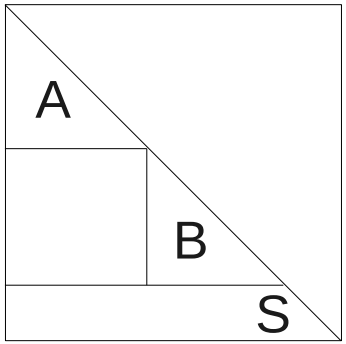


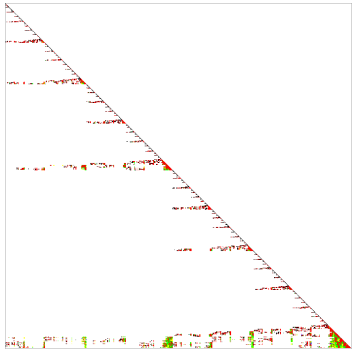


Nested dissection

- Top-down strategy for removing potential fill-inducing paths
- Principle [George, 1973]
  - Find a vertex separator of the graph
  - Order separator vertices with available indices of highest rank
  - Recursively apply the algorithm on the separated subgraphs







The Scotch project and roadmap

The Scotch project

- Devise robust parallel graph partitioning methods
  - Initial roadmap : should handle graphs of more than a billion vertices distributed across one thousand processors

DONE !

- Improve sequential graph partitioning methods if possible
  - E.g. provide new features such as fixed vertices
- Provide graph repartitioning methods
- Investigate alternate graph models (meshes/hypergraphs)
- Provide a software toolbox for scientific applications
  - Scotch sequential software tools
  - PT-Scotch parallel software tools

Parallel partitioning : weak scalability results

- Since version 5.1.10, Scotch is fully 64-bit
  - Can handle graphs above 2 billion vertices
  - But less than 2 billion edges by processing elements, 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
- Quality preserved on these numbers of vertices and of processors, but what will happen for more ?

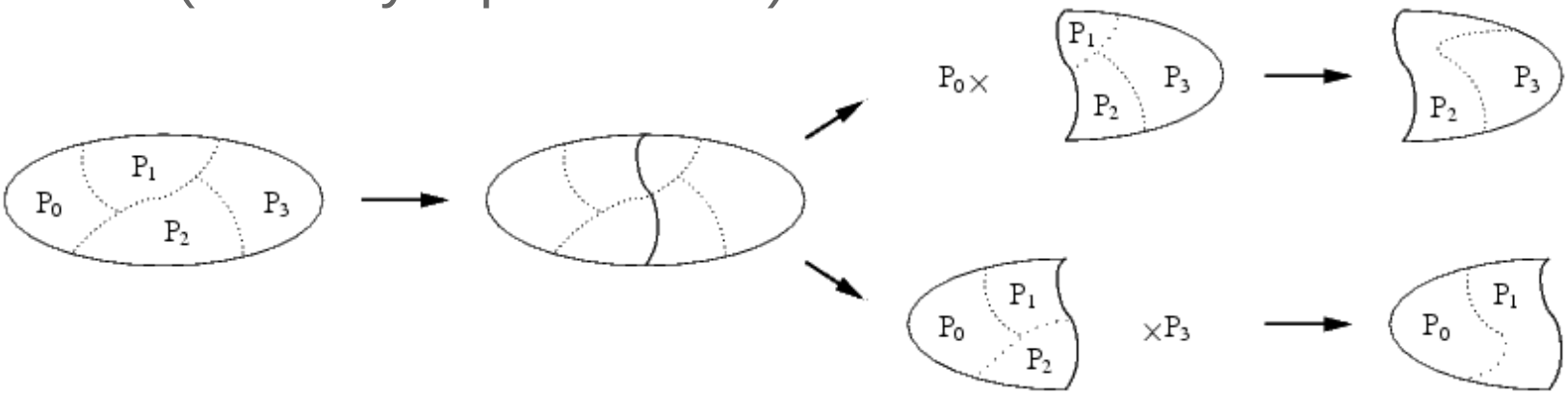
The multilevel framework and its parallelization

From k-partitioning to recursive bipartitioning

- *K*-way graph partitioning can be approximated by a sequence of recursive bipartitions
  - Bipartitioning is easier to implement than *k*-way partitioning
    - No need to choose the destination part of vertices
- It is only an approximation, but a rather good one [Simon & Teng, 1993]

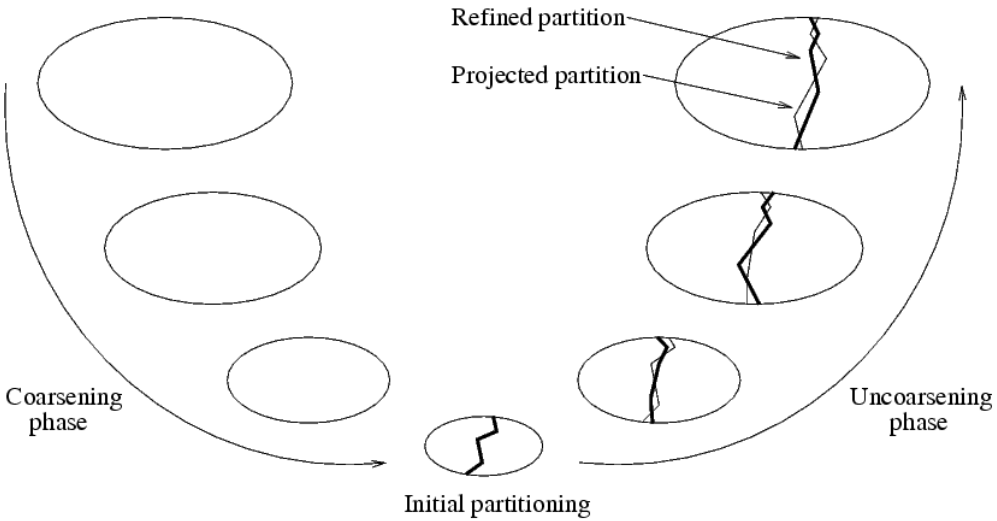
Recursive bipartitioning in parallel

- After a separator has been computed, the two separated subgraphs are folded and redistributed each on a half of the available processors
  - All subgraphs at a same level are processed concurrently on separate subsets of processors
  - Ability to fold a graph on any number of processors (not only a power of 2)



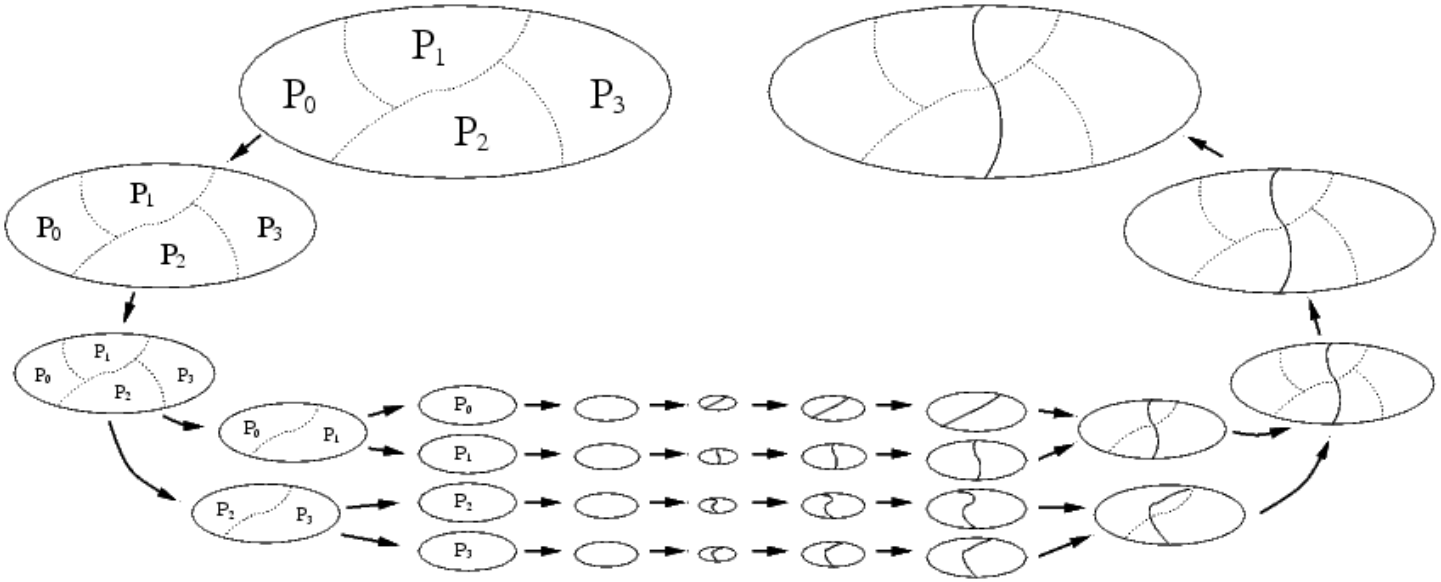
Multilevel framework

- Principle [Hendrickson & Leland, 1994]
  - Create a family of topologically equivalent coarser graphs by clustering groups of vertices
  - Compute an initial partition of the smallest graph
  - Propagate back the result, with local refinement



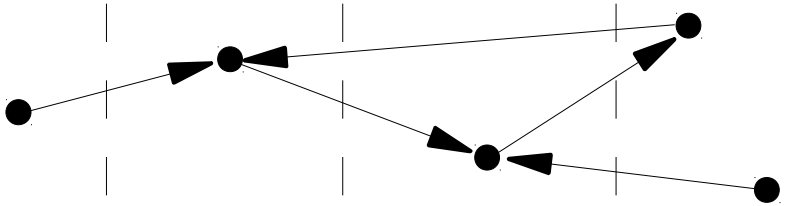
Coarsening in parallel

- The coarsened graph can either be:
  - Kept on the same number of processors: decreases memory and processing cost
  - Folded and duplicated on two subsets of processors: increases quality but also cost



Parallel matching

- Parallel coarsening is based on parallel matching
  - These matchings do not need to be maximal
- Synchronization between non-local neighbors is critical
  - Dependency chains or loops between mating requests can stall the whole algorithm because of sequential constraints



- Some distributed tie-breaking is required
- Too many requests decrease matching probability

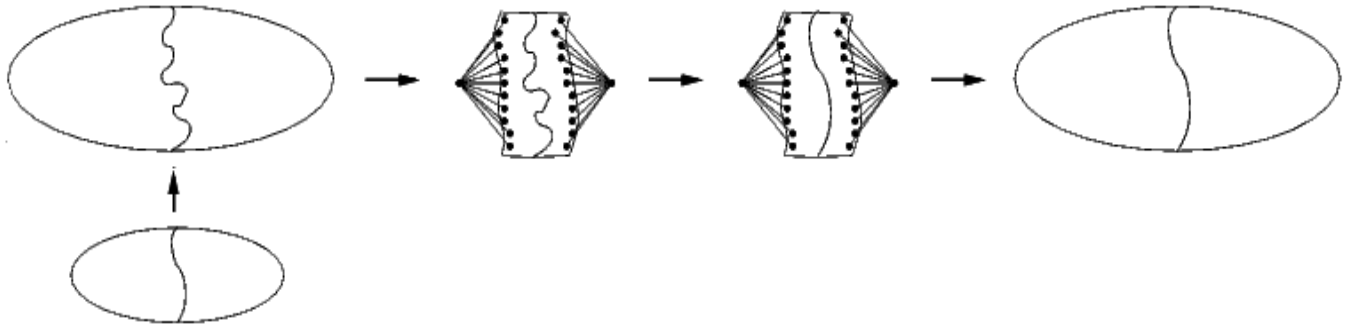
Parallel probabilistic matching

- Principle [Chevalier, 2007]
  - Do not discriminate between local and non-local neighbors when selecting a neighbor for mating
  - Vertices request for matings with their neighbors (whether local or remote) with a prescribed probability
- Reduces topological biases and converges quickly
  - 5 collective passes are enough to match 80 % of the vertices on average

Pass	Matching		Coarsening	
	Avg.	M.a.d.	Avg.	M.a.d.
C1	53.3	12.3	50.4	0.7
C2	68.7	13.6	51.6	2.2
C3	76.2	12.2	52.5	3.3
C4	81.0	10.6	53.2	4.0
C5	84.5	9.1	53.7	4.5
LF	100.0	0.0	59.4	6.8

Band graphs

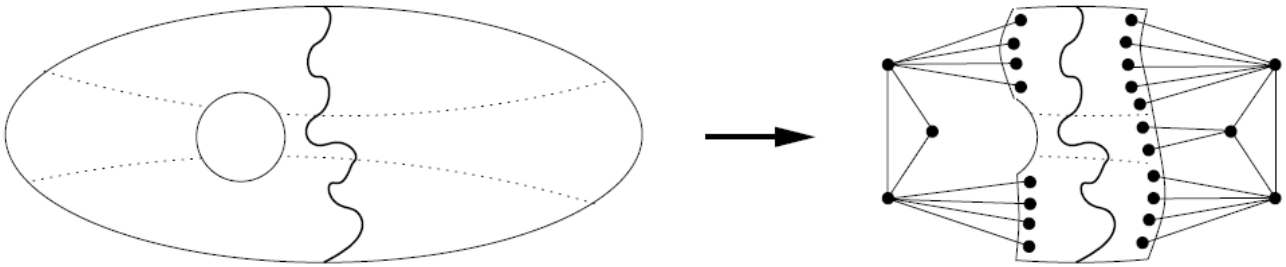
- Principle [Chevalier & Pellegrini, 2006]
  - Only local improvements along the projected cut are necessary, so work only on a small band around the cut



- Reduce problem space dramatically
  - Allow one to use expensive algorithms, such as genetic algorithms

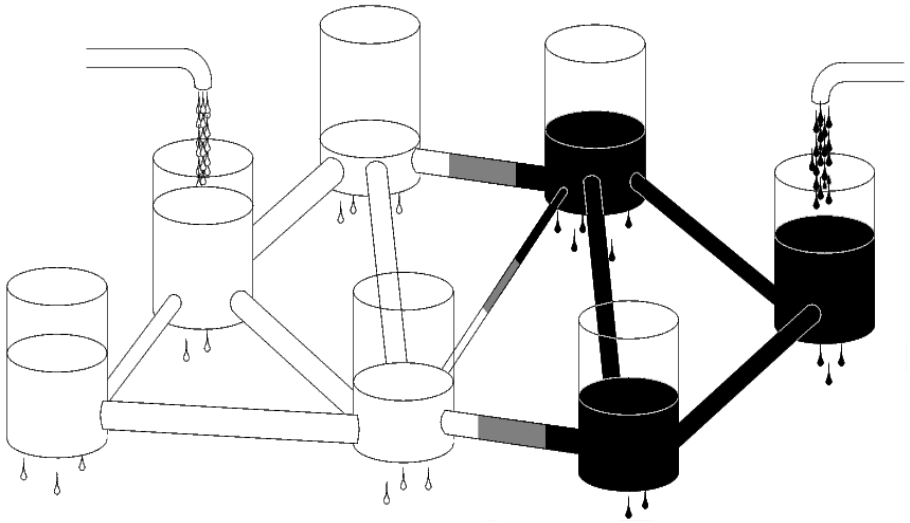
Band graphs in parallel

- Anchor vertices may have very high degrees compared to sequential band graphs
  - Two anchor vertices per process
  - Remote anchor vertices for each part form a clique
    - Will soon be a hypercube to accommodate for large numbers of processes



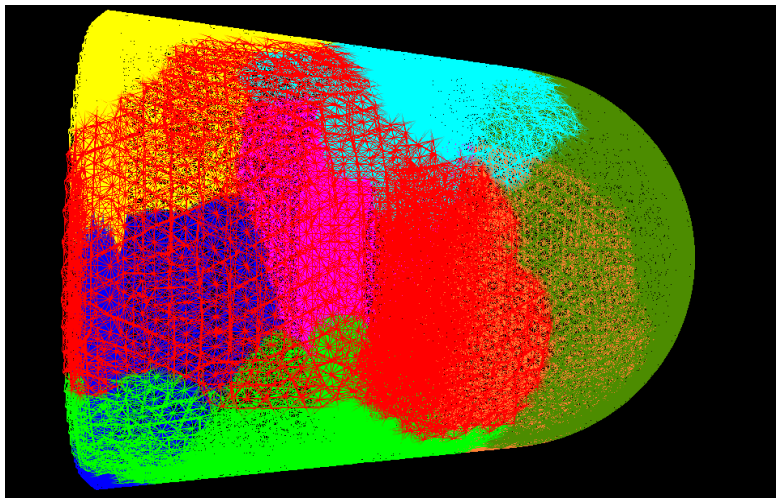
Jug of the Danaides (1)

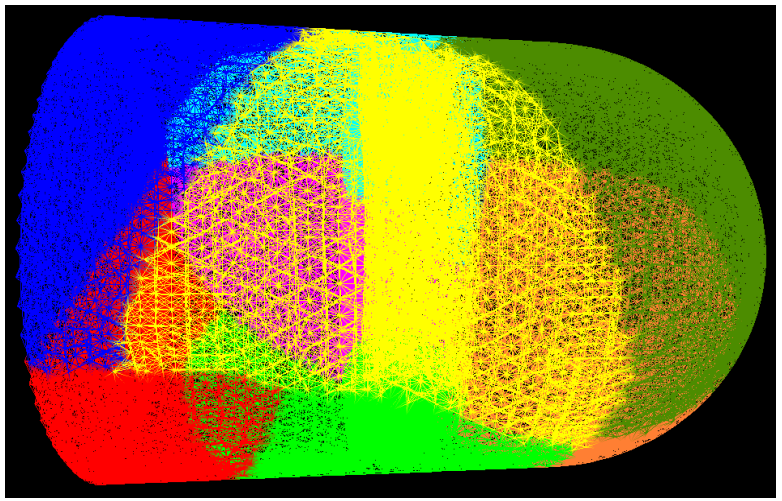
- Principle [Pellegrini, 2007]
  - Analogous to “bubble growing” algorithms but natively integrates the load balancing constraint
  - The graph is modeled as a set of leaking barrels and pipes
  - Two antagonistic liquids flow from two source vertices
  - Liquids vanish when they meet



Jug of the Danaides (2)

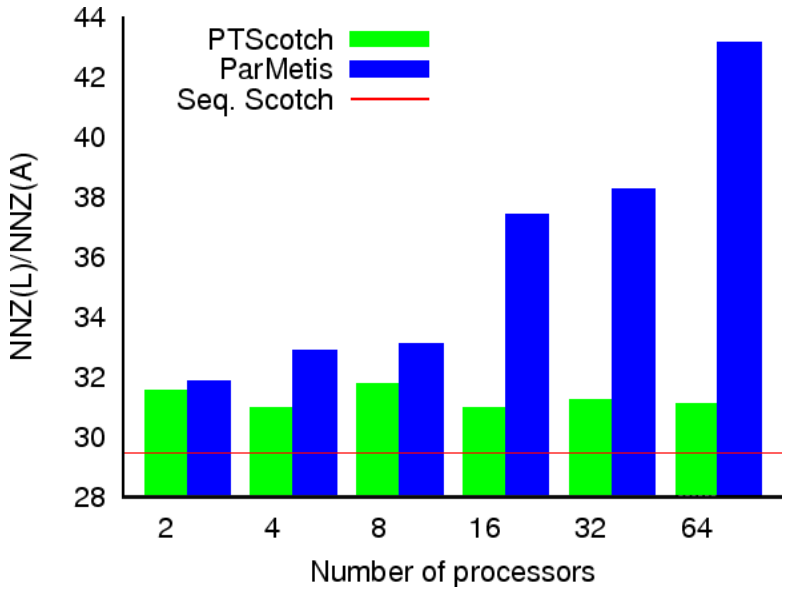
- Using JotD as the refinement algorithm in the multi-level process :
  - Yields smooth interfaces
  - Is slower than sequential FM (20 times for 500 iterations, but only 3 times for 40 iterations)
- Band graph anchor vertices used as source vertices

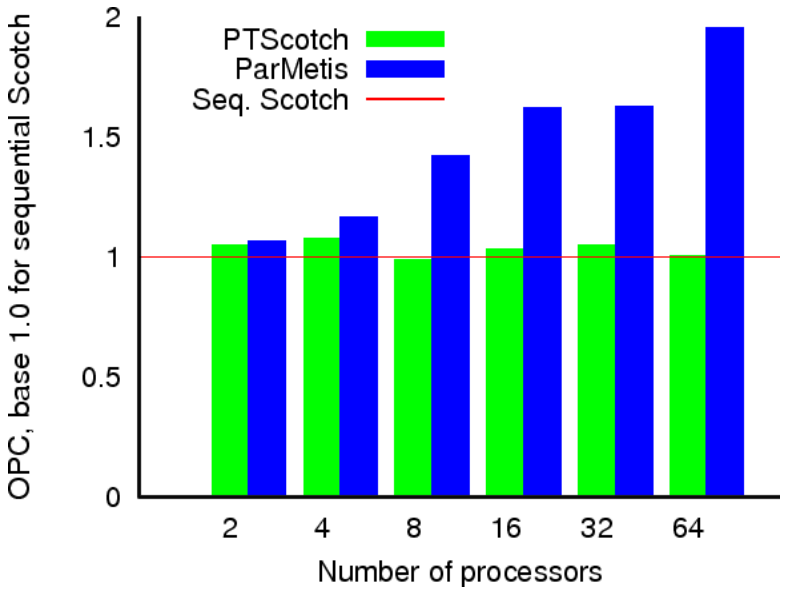




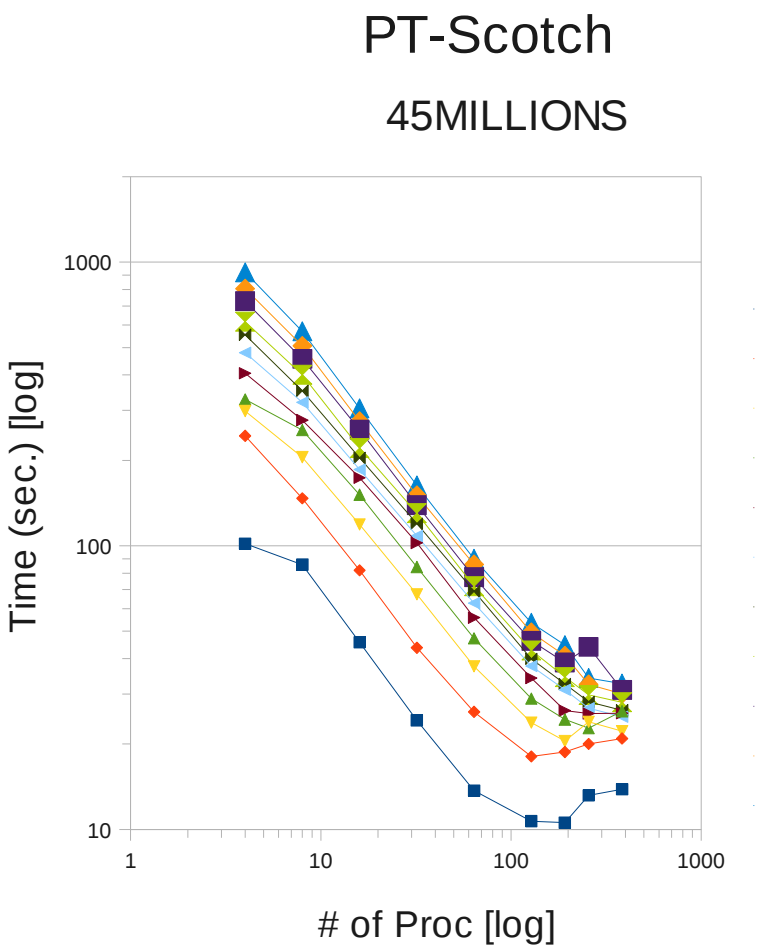
Runtime and sparse matrix ordering quality

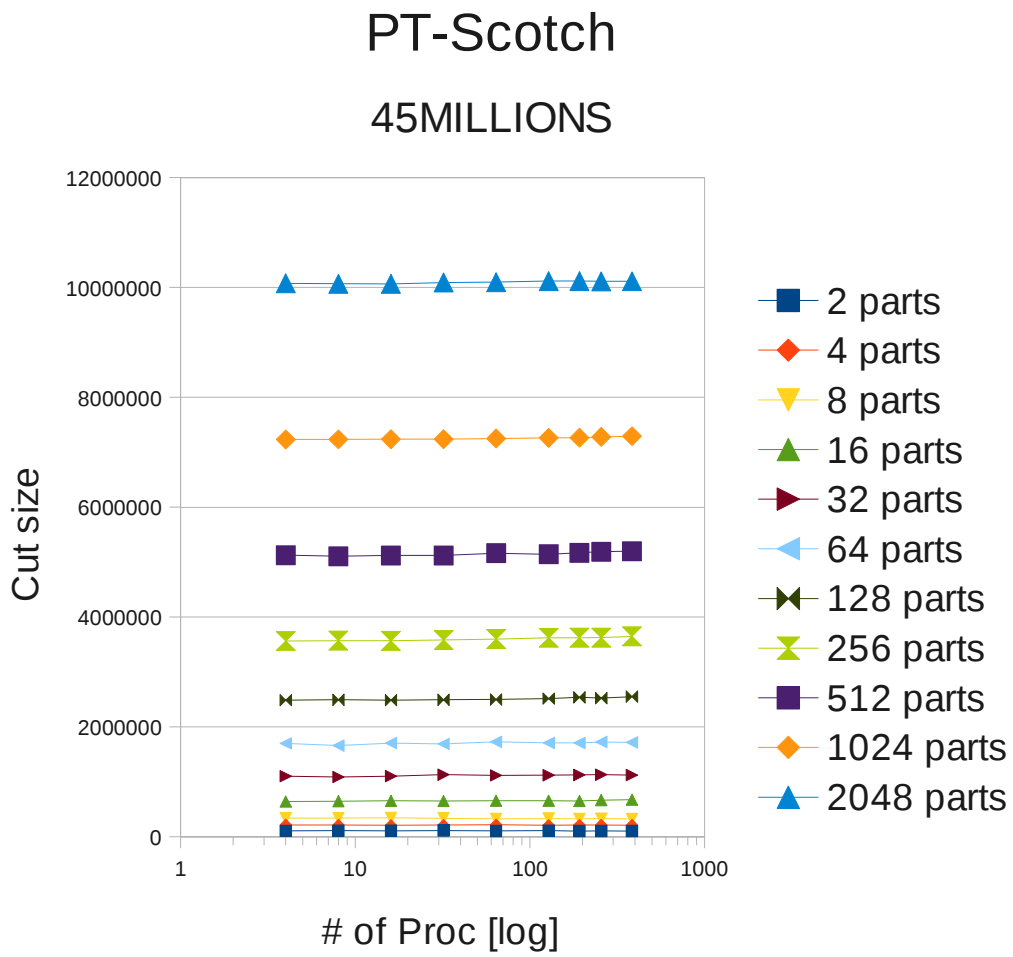
Test case	Number of processes					
	2	4	8	16	32	64
audikw1						
O_{PTS}	**5.73E+12**	**5.65E+12**	**5.54E+12**	**5.45E+12**	**5.45E+12**	**5.45E+12**
O_{PM}	5.82E+12	6.37E+12	7.78E+12	8.88E+12	8.91E+12	1.07E+13
t_{PTS}	64.14	43.72	31.25	20.66	13.86	9.83
t_{PM}	32.69	23.09	17.15	9.80	5.65	3.82





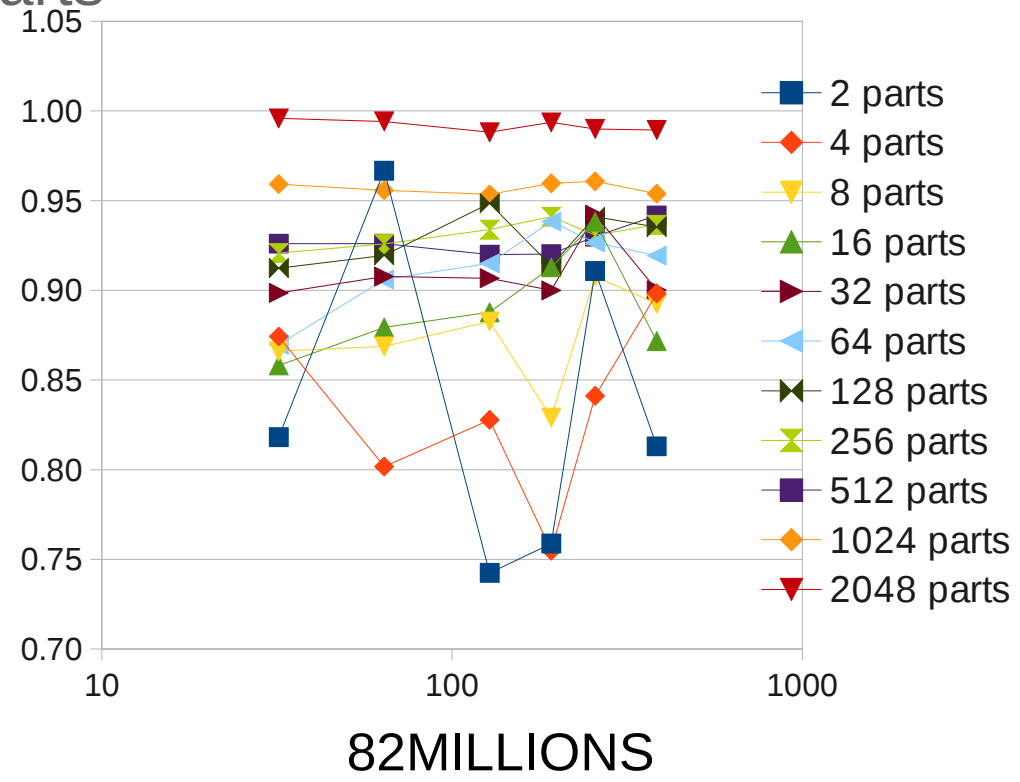
Runtime and partition quality (1)





Runtime and partition quality (2)

- Cut size ratio is most often in favor of PT-Scotch vs. ParMeTiS up to 2048 parts
  - Partition quality of ParMeTiS is irregular for small numbers of parts
  - Gets worse when number of parts increases as recursive bipartitioning prevents performing global optimization



Three challenges for the next roadmap

New roadmap

- To be able to map graphs of about a trillion vertices spread across a million processing elements
  - Same number of vertices per processing element as in the first roadmap
  - Focus on scalability problems related to the large number of processors
- Parallel dynamic repartitioning capabilities are mandatory

Three challenges

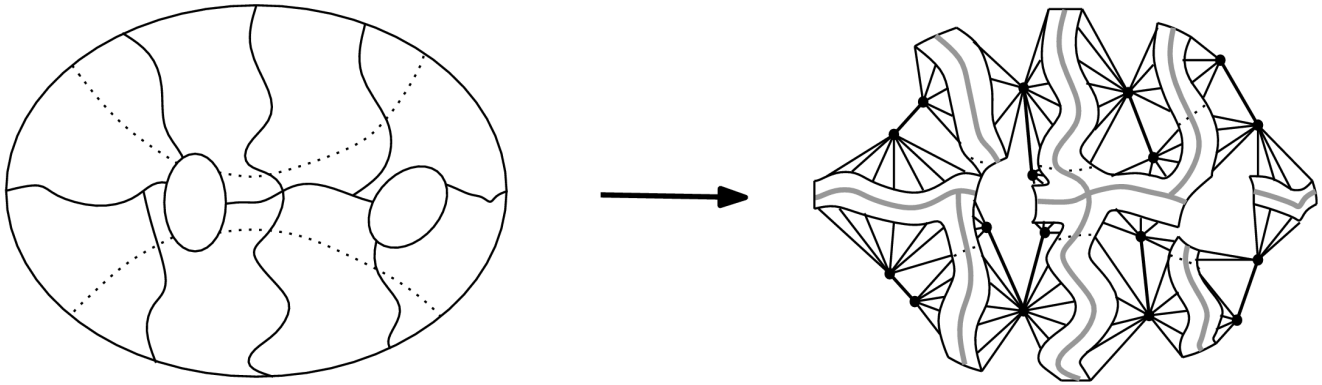
- Scalability
  - How will the algorithms behave for large numbers of processing elements ?
- Heterogeneity
  - How will the architecture of the target machine impact performance ?
- Asynchronicity
  - Will our algorithms still be able to rely on fast collective communication ?

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

Parallel direct k-way graph partitioning

- Extension to k parts of the multilevel framework used for recursive bipartitioning
  - Straightforward for the multi-level framework itself
  - Relies on distributed k-way band graphs



Architectural considerations matter

- Upcoming machines will comprise very large numbers of processing units, and will possess NUMA / heterogeneous architectures
  - More than a million processing elements on the *Blue Waters* machine to be built at UIUC
- 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

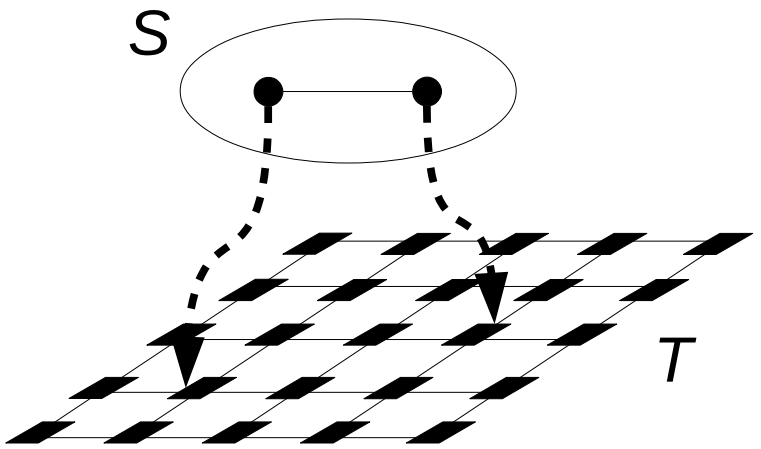
Static mapping

- 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

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

- Communication cost function accounts for distance

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

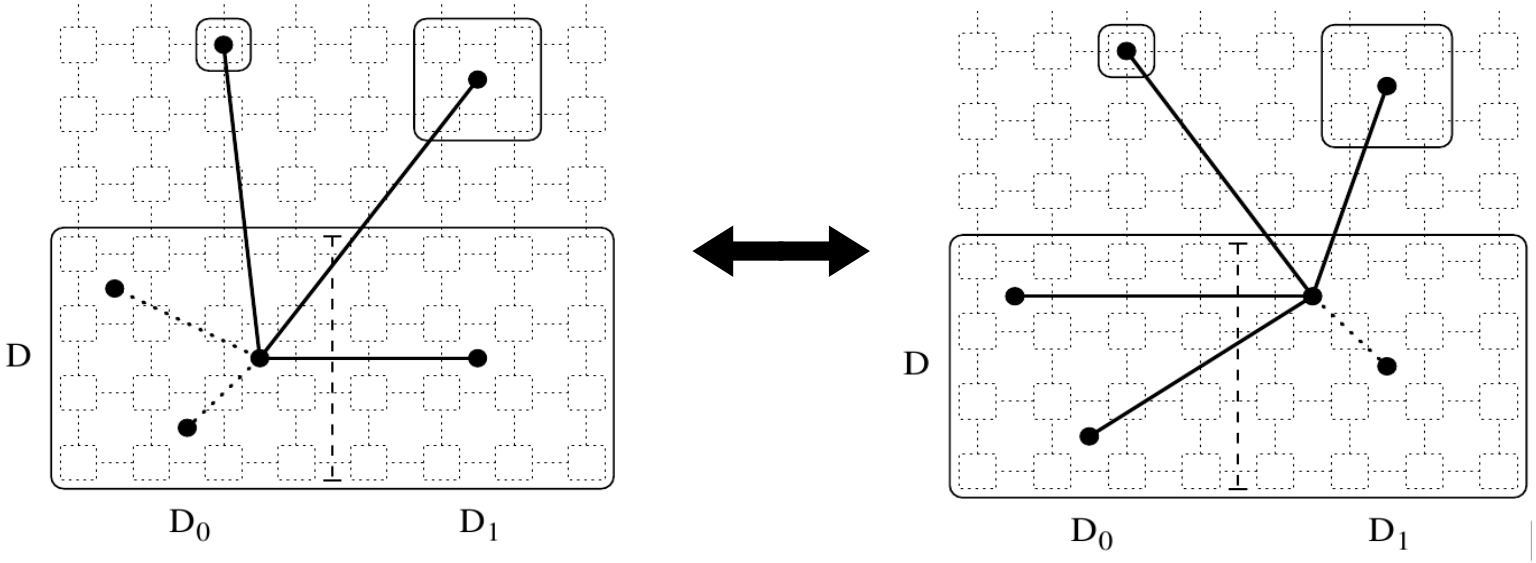


Recursive bi-mapping

- Partial cost function for recursive bipartitioning

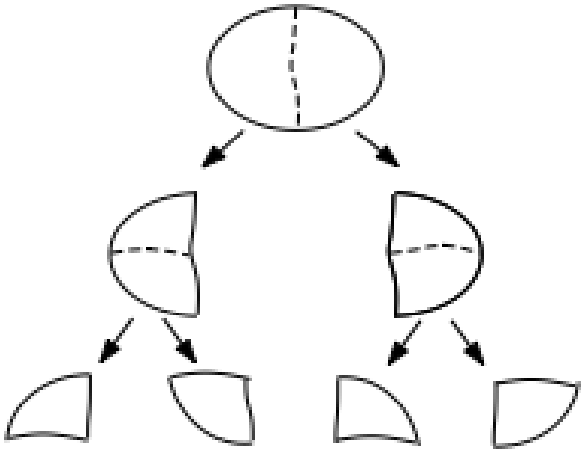
$$f'_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{\substack{v \in V(S') \\ \{v,v'\} \in E(S)}} w(\{v,v'\}) \, |\rho_{S,T}(\{v,v'\})|$$

- Decision depends on available mapping information

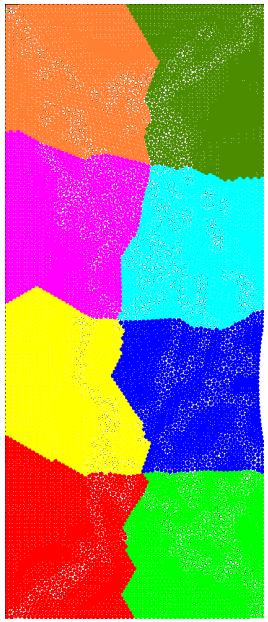


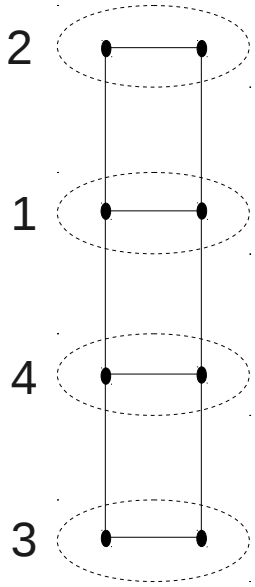
Parallel static mapping (1)

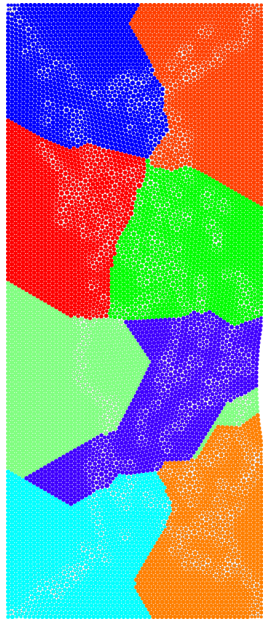
- Recursive bi-mapping cannot be parallelized as is
  - All subgraphs at some level are supposed to be processed simultaneously for parallel efficiency
  - Yet, ignoring decisions in neighboring subgraphs can lead to “twists”



- Sequential processing only!

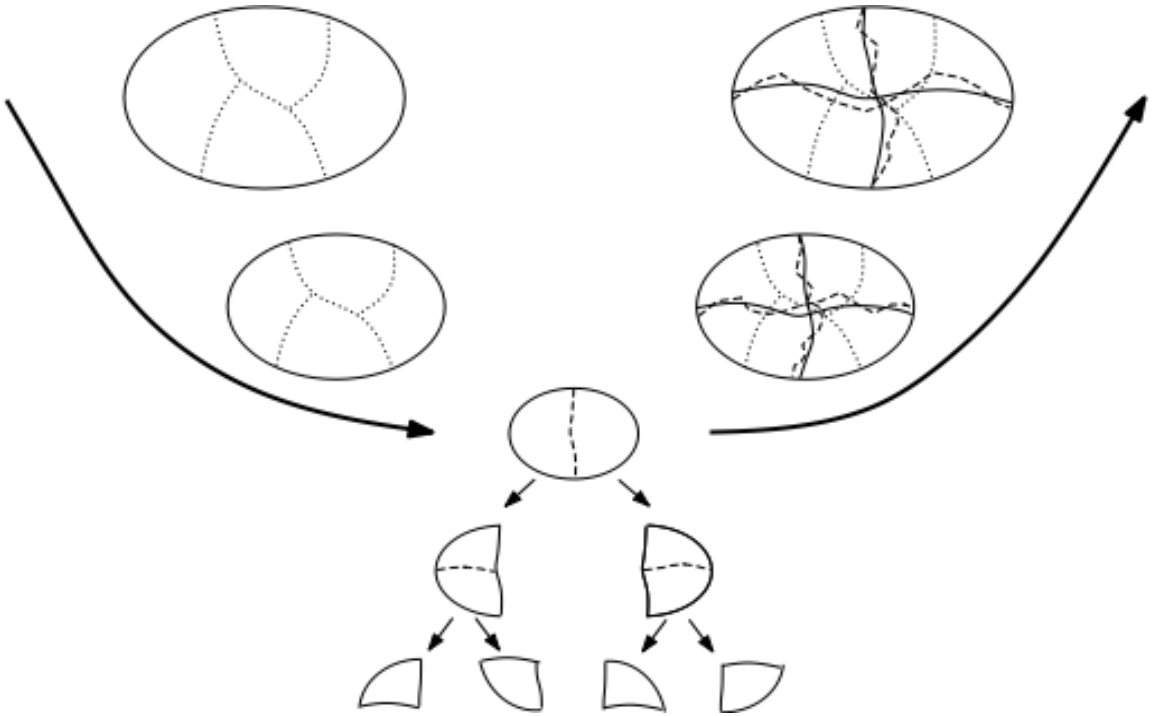






Parallel static mapping (2)

- Parallel multilevel framework for static mapping
  - Parallel coarsening and k-way mapping refinement
  - Initial mapping by sequential recursive bi-mapping



Asynchronous algorithms

- Need for algorithms that can evolve asynchronously at different paces depending on communication latency
  - Genetic algorithms are good candidates at a global level but are still too slow to converge
  - Diffusion-based methods can be envisioned
    - Most probably on the form of influence methods
  - Multilevel optimization algorithms can also be considered

Where we are now...

On-going work (1)

- Dynamic graph repartitioning with fixed vertices, based on a direct k-way partitioning framework [PhD of Sébastien Fourestier]
  - Sequential code extensively tested (v. 6.0)
  - Parallel code being completed (v. 6.0 or 6.1)
- Shared memory parallelism (pthread-based)
  - Coding started for Scotch (v. 6.0 or 6.1)
  - Will have to be extended to PT-Scotch (v. 6.1 or 6.2)

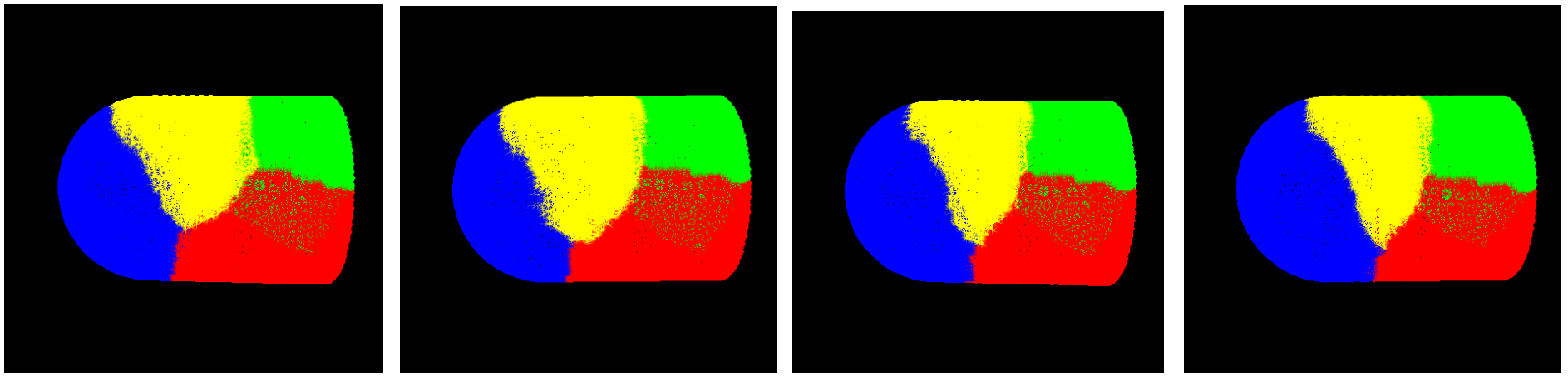
On-going work (2)

- Multi-criterion (re)partitioning ?
  - PhD thesis may start next December
- Asynchronous algorithms for exaflop computing ?
  - PhD thesis next year ?
- Parallel hypergraph partitioning ?
  - Only if gains can be expected over existing works
- Move upwards to application mesh models : PaMPA
  - Joint work with C. Lachat and C. Dobrzynski
  - Based on the expertise accumulated on the handling of distributed graphs

Repartitioning

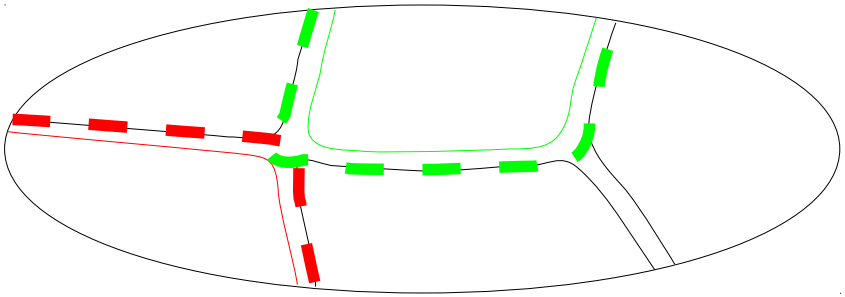
- Ability to compute repartitions that take into account fixed vertices, new vertices and a wide range of migration costs

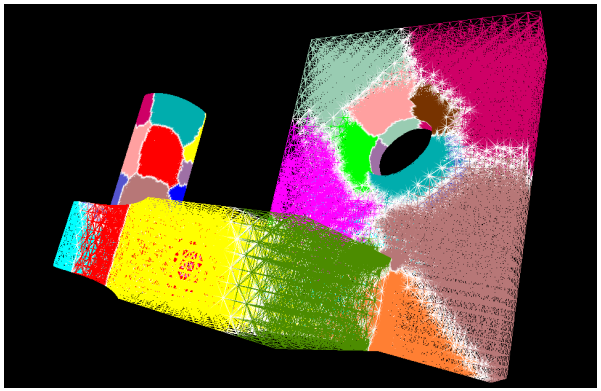
0.1 | 1 | 10 | original partition



K-way vertex partitioning with overlap

- Balance part loads according to inner vertices as well as neighboring separator vertices [Post-doc of Jun-Ho Her]
  - Separator vertices may contribute to several parts
- Mixed results in terms of HID (experiments with HIPS)
  - Must take other criteria into account





Thank you for your attention !

Any questions ?

http://scotch.gforge.inria.fr/