Habilitation à Diriger des Recherches

École doctorale de Mathématiques et d'Informatique
Université de Bordeaux 1

Contributions au partitionnement de graphes parallèle multi-niveaux

(*Contributions to parallel multilevel graph partitioning*)

François Pellegrini

3 décembre 2009

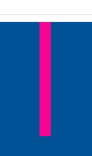
Summary of the talk

- An introduction to combinatorial scientific computing and graph partitioning
- The multi-level framework
- Parallelization of the coarsening phase
- Parallelization of the refinement phase
- Conclusion and future directions for research

An introduction to combinatorial scientific computing and graph partitioning

Context

- Combinatorial scientific computing
  - Community "*concerned with the formulation, application and analysis of discrete methods in scientific applications*" [Hendrickson & Pothen, 2007]
- Takes its mindset and toolset from two main streams of informatics :
  - Graph theory
    - Discrete algorithms
  - Parallel computing
    - Main problems and applications in the field of scientific computing



Sparse matrix ordering (1)

- Solve the linear system :

$$A.x = b$$

where :

- A is **symmetric** (S) : $A = A^T$ (real) or $A = A^*$ (complex)
- A is **definite-positive** (DP) : $\forall\, x \neq 0,\ x^T.A.x > 0$
  - Brings numerical stability properties
- A is **sparse** : the number of non-zero terms in A is small compared to the size of the matrix
  - Depends on problem type (not size)
  - Usually in O(1) per row or column
  - Reduced storage and computations

Sparse matrix ordering (2)

- When A is SDP, the linear system can be solved by means of Cholesky factorization :

$$A = L.L^T$$

where :

  - L is **lower triangular**

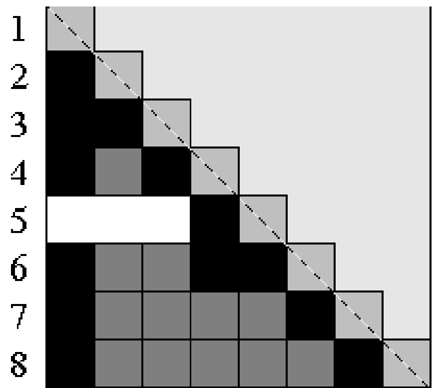
- The factored system $L.L^T.x = b$ can then be solved by triangular solving :
  - $L.y = b$
  - $L^T.x = y$

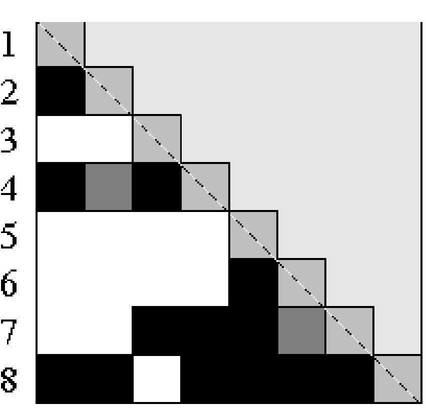
Fill-in (1)

- When factorizing A into $L.L^T$, L incurs **fill-in**
  - In addition to preexisting terms of A, potential non-zero terms are created in L during the factorization process
  - Variant of Gaussian elimination : for each column in index order, terms are added by merging patterns of left columns having non-zero entries of smallest index facing the current diagonal entry
- Storage for these additional terms has to be allocated even if they will hold numerical zeros
  - Value cannot be known in advance
  - **Symbolic factorization**

Fill-in (2)

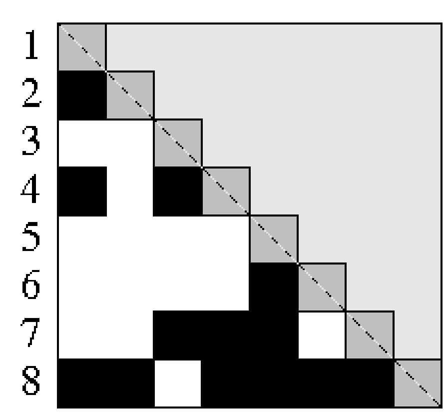
- Fill-in only depends on the order in which unknowns are processed
  - Amounts to solving the permuted system $PAP^T.x = b$
  - Numerical stability of Cholesky factorization is not impacted by the order of the unknowns
  - Yet, different orders can produce very different fill-ins

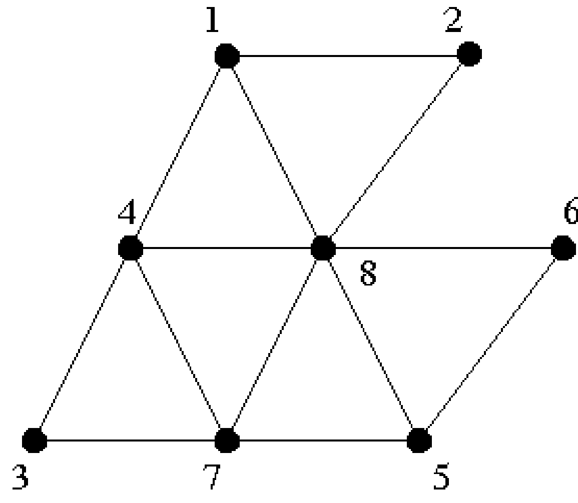




Adjacency graph

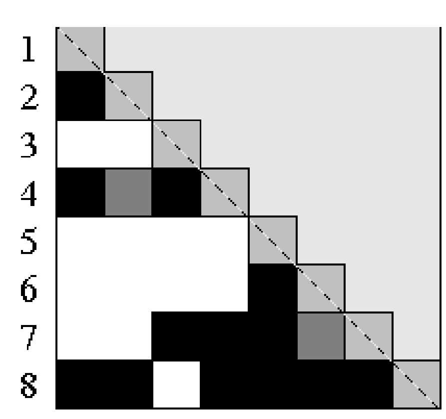
- Symmetric (and sparse) matrices can be represented on the form of adjacency graphs
  - Vertices represent unknowns
  - Edges represent extra-diagonal non zeros
    - Unoriented edges (not arcs)
    - No loop edges

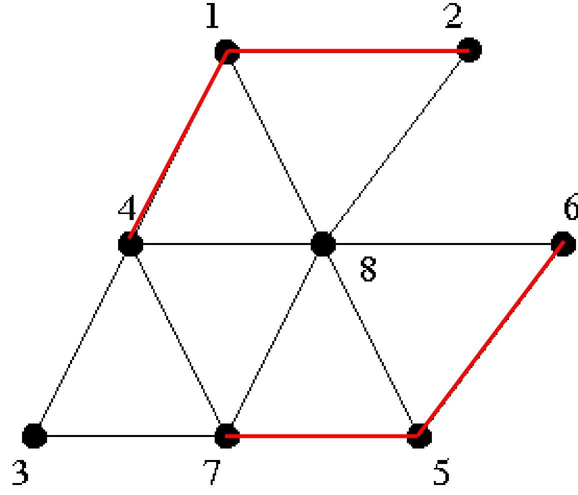




Fill-reducing orderings (1)

- By nature of Gaussian elimination, a zero $a_{i,j}$ term of the matrix will incur fill-in during factorization if there exists in the adjacency graph a path linking vertices v_i and v_j, such that all intermediate vertices have indices smaller than min(i,j)



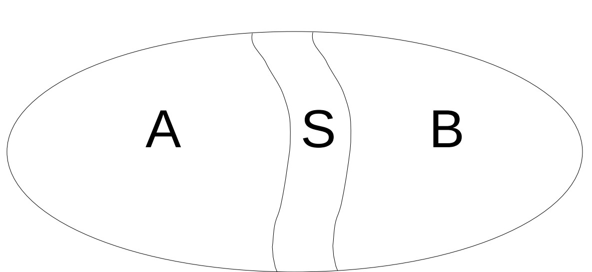


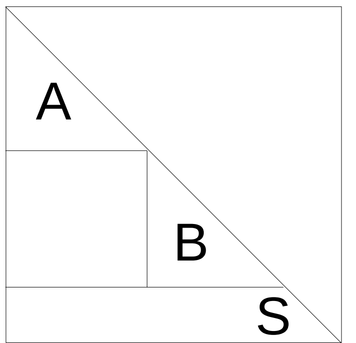
Fill-reducing orderings (2)

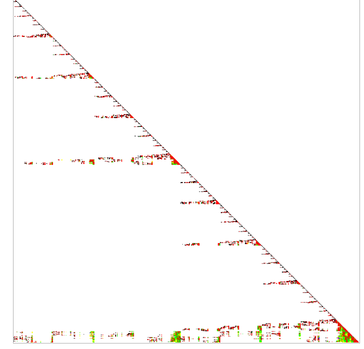
- Fill-reducing orderings are orderings which prevent as much fill-inducing paths as possible
- Two main classes of heuristics :
  - **Minimum degree** methods [Tinney & Walker, 1967]
    - Order first vertices with smaller degrees, so that fewer fill-inducing paths will be likely to pass through them
    - Bottom-up strategy
  - **Nested dissection** methods [George, 1973]
    - Raise impassable barriers of high index vertices to break as many paths as possible
    - Top-down strategy

Nested dissection (1)

- 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







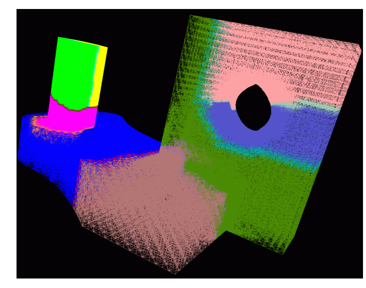
Nested dissection (2)

- The problem of finding fill-reducing orderings has been transformed into a graph partitioning problem
- Balanced bisections are not the only way to go :
  - Unbalanced bisections amount to one-way dissections
    - Useful for graphs with large aspect ratio [George, 1980]
  - Bi-level multisections [Ashcraft & Liu, 1998]
- Yet, recursive bisection has useful properties suitable for parallel linear system solving :
  - Balanced bipartitions provide broad and balanced elimination trees

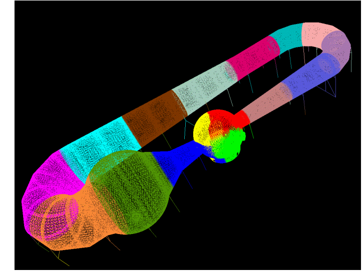
Graph partitioning (1)

- Graph partitioning 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-complete 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

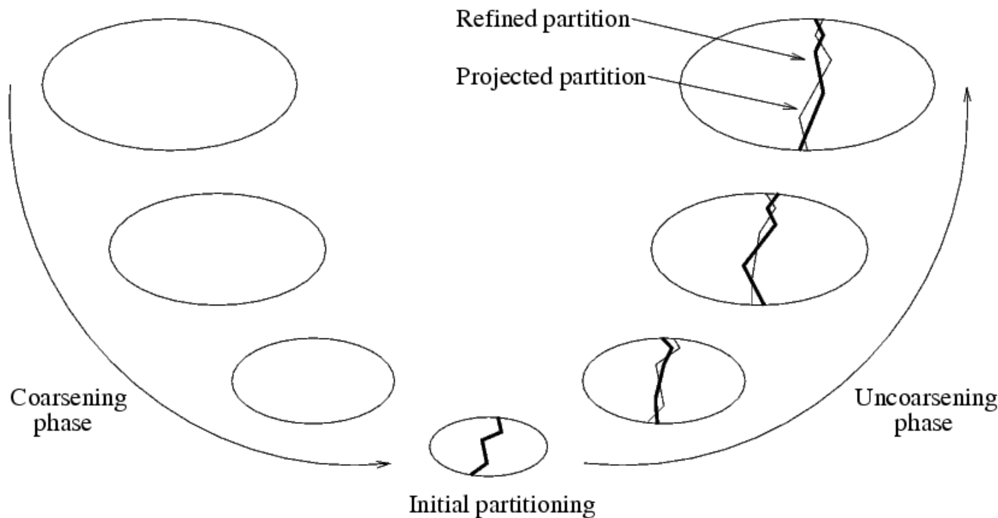
The multi-level framework

Graph partitioning algorithms

- Two main classes of partitioning algorithms :
  - **Global methods** (e.g. genetic algorithms, simulated annealing, greedy graph algorithms)
    - Consider all of the graph data
    - Are most often very slow when quality is desired
  - **Local optimization heuristics** (e.g. Fiduccia-Mattheyses)
    - Optimize an existing partition
    - Applied after some global method
    - Fast but have limited scope

Multi-level 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 (1)

- Coarsening amounts to **quotienting** finer graphs according to some clustering partition, to obtain a coarser graph of similar topological structure

- Several variants exist :
  - **Matching** of vertex pairs [Hendrickson & Leland, 1994]
    - Matchings do not need to be maximal
  - Weighted aggregation of groups of vertices [Chevalier & Safro, 2009]
    - Aims at reducing the impact of coarsening artifacts

Refinement

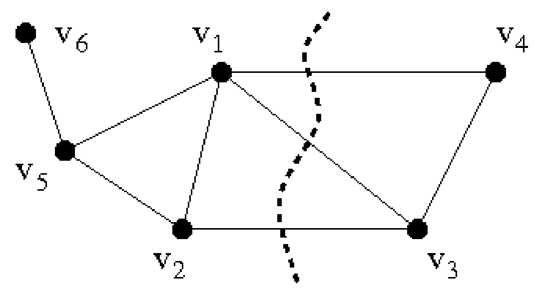
- The partition computed on the coarser graph is prolongated to the finer graphs
  - All of the vertices in every cluster are assigned to the same part as the one of the associated coarse vertex
  - The prolongated solution has the granularity of the coarser graph, because of coarsening artifacts
- The prolongated partition must be refined
  - Only local refinement is needed since global shape is assumed to be good
  - Use of local optimization algorithms

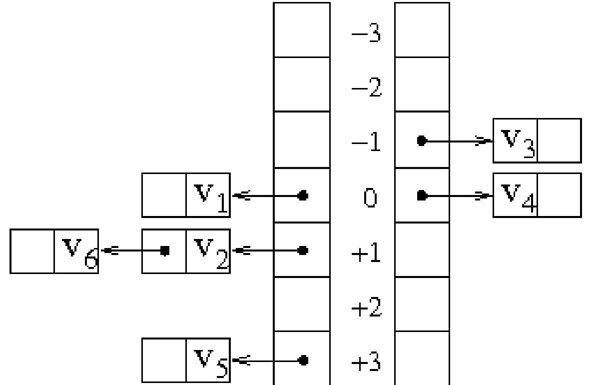
Local optimization algorithms (1)

- Try to improve a current partition by moving vertices between parts across the frontier
- The most widely used algorithms are greedy iterative graph algorithms
  - Both fast and efficient
- Several variants exist :
  - Kernighan-Lin (KL) [1970] : swaps of pairs of vertices between their two parts
  - Fiduccia-Mattheyses (FM) [1982] : individual moves from one part to another
  - Helpful sets [Diekmann *et. al.*, 1995] : moves of clusters

Local optimization algorithms (2)

- Frontier vertices are moved according to their “gain value”, *i.e.* the improvement of the cut that results in moving them





- Gains are updated after every move
  - Creates strong sequentiality constraints
- Detrimental moves can be accepted, provided that further beneficial moves are performed afterwards, which will result in an overall gain
  - Hill-climbing from local minima of the cut cost function

The need to go parallel

- Problem size keeps increasing
  - Graphs of more than ten million vertices cannot be handled on sequential computers
  - Need for scalable parallel graph partitioning tools
- Some parallel graph partitioning tools already exist
  - ParMeTiS [Schloegel, Karypis & Kumar, 1997]
  - ParJOSTLE [Walshaw *et al.*, 1997]
- Existing parallel tools evidence performance problems :
  - Quality of partitions most often decreases when the number of processors increase
  - State-of-the art local optimization algorithms are intrinsically sequential and do not parallelize well

The Scotch roadmap

- Devise robust parallel graph partitioning methods
  - Should handle graphs of more than a billion vertices distributed across one thousand processors
- Improve sequential graph partitioning methods if possible
  - Multi-level FM-like algorithms are both fast and efficient on a very large class of graphs but FM algorithms are intrinsically sequential
- Investigate alternate graph models (meshes/hypergraphs)
- Provide a software toolbox for scientific applications
  - Scotch sequential software tools
  - PT-Scotch parallel software tools

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 !
    - Theory is not likely to help much...
- 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

Parallelization of the coarsening phase

Questions about parallel coarsening

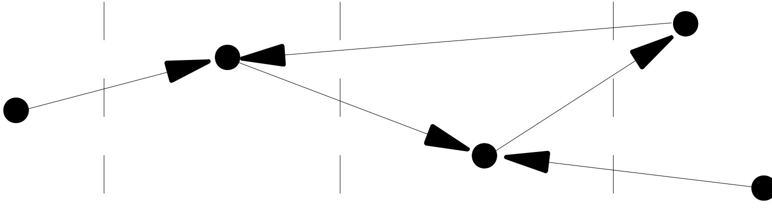
- Is there an efficient and scalable way to compute matchings in parallel ?
  - Matchings do not need to be minimal
  - Matching process should avoid any bias
    - Should not depend in any way on data distribution
- What do we do when coarse graphs are “too small” ?
  - Processing time is dominated by communication start-up time
    - More processes mean more operating system and synchronization hazards

Parallel matching (1)

- All existing parallel coarsening algorithms base on parallel matching to cluster pairs of adjacent vertices
  - Coarse graphs are built according to this clustering
- Doing the matching in parallel is not easy because :
  - The quality of the matching is critical for cut quality
    - Biases in the matching algorithm lead to significant loss of quality
  - Synchronization is required between processes which bear adjacent vertices, to propagate part ownership
  - Graphs may be distributed in a way that requires much communication
    - Else, the task would be too easy...

Parallel matching (2)

- 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 matching by graph coloring (1)

- Principle [Karypis and Kumar, 1999]
  - Compute a vertex coloring of the finer graph
    - No two neighbor vertices have the same color
  - A matching sweep is made of as many rounds as there are colors in the graph coloring
    - Only vertices of the current color can ask for mating during their round

- Removes chains, as well as many collisions

Parallel matching by graph coloring (2)

- Vertex colorings are computed using a distributed version of Luby's algorithm
  - Every vertex draws a random number
  - Vertices which are local maxima are painted with the first color, after which they are removed from the graph
  - The algorithm iterates until all vertices are colored
- Some colors have very few vertices
  - Partial sequentialization of the algorithm
  - Too many rounds for coloring, and for mating

998699	908307	848574	804471	777182	757239	743009	733290
720049	700112	661773	590857	478982	338154	200174	99186
41975	15071	4750	1377	372	108	20	6

Graph: 10millions, |V| =10,424 k, |E| = 78,649 k, δ=15.09, type: 3D electromagnetics mesh, CEA/CESTA

Parallel probabilistic matching (1)

- Principle [Her & Pellegrini, 2009] [Chevalier, 2007]
  - The algorithm consists in a fixed number of passes
  - During each pass, yet unmatched vertices draw a random bit value.
    - If it is zero, the vertex is inactive during the pass
    - If it is one, the vertex sends a mating request to any one of its presumably unmatched neighbors
    - Sought for vertices reply positively or negatively
- Reduces topological biases
- Improves mating probability when data are irregularly distributed

Parallel probabilistic matching (2)

- Unlike all of its predecessors, this algorithm makes no assumption on the distribution of the graph vertices
  - Cannot induce any bias due to distribution artifacts
  - Converges quicky
    - 5 collective passes are enough to match 80 % of the vertices

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

Parallelization of the coarsening phase (1)

- Once the parallel matching algorithm terminates, the coarsened graph is built
  - Using the same number of processes as the one used by the finer graph
- At this stage, the coarsened graph can either be :
  - Kept on the same number of processes
    - Decreases memory and processing cost
  - Folded on half of the processes [Karypis et *al.*, 1997]
    - To reduce communication cost and improve data locality (reduces bias of biased algorithms)
  - Folded and duplicated on two subsets of processors [Chevalier & Pellegrini, 2008]

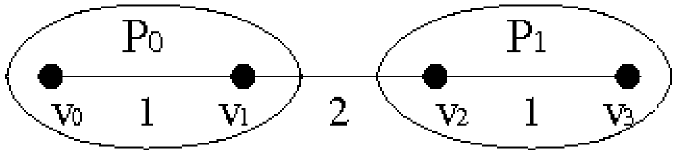
Parallelization of the coarsening phase (2)

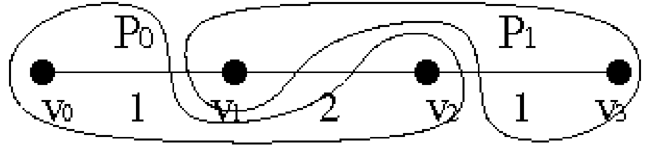
- It is preferable to use folding and duplication only in the last stages of the coarsening process
  - All of the processes will compute distinct initial partitions

Parallelization of the refinement phase

Questions about parallel refinement

- Can we find a way to use global algorithms instead of local optimization algorithms ?
  - Local optimization algorithms do not parallelize well...

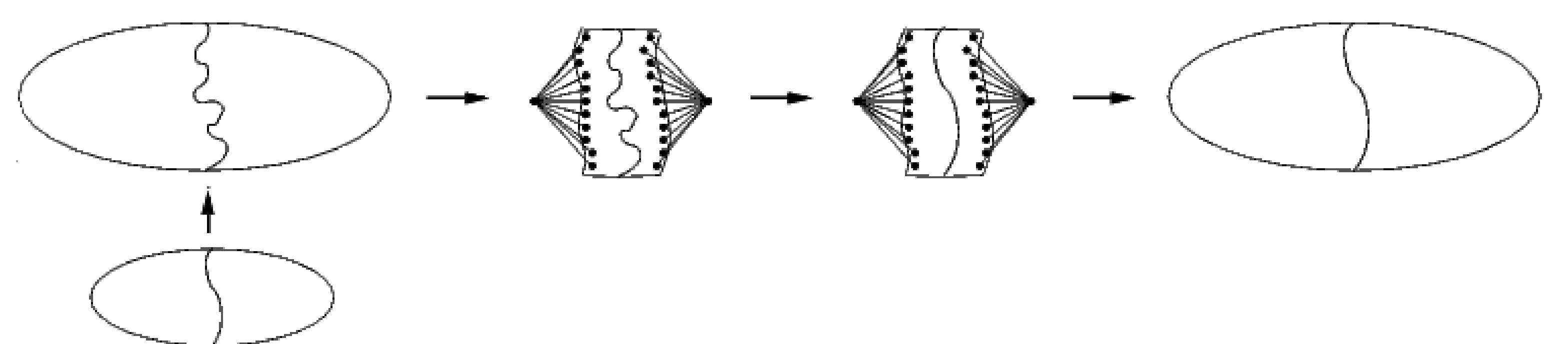




- Can we preserve as much as possible the quality of existing local optimization algorithms ?
  - Especially needed for sparse matrix ordering
    - Concerns “small” graphs, with less than one hundred million vertices

Band graphs (1)

- Principle [Chevalier & Pellegrini, 2006]
  - Since only local improvements are necessary on the finer graph, it is not necessary to provide the refinement algorithm with all of the graph data, as only a small band around the projected separator is necessary

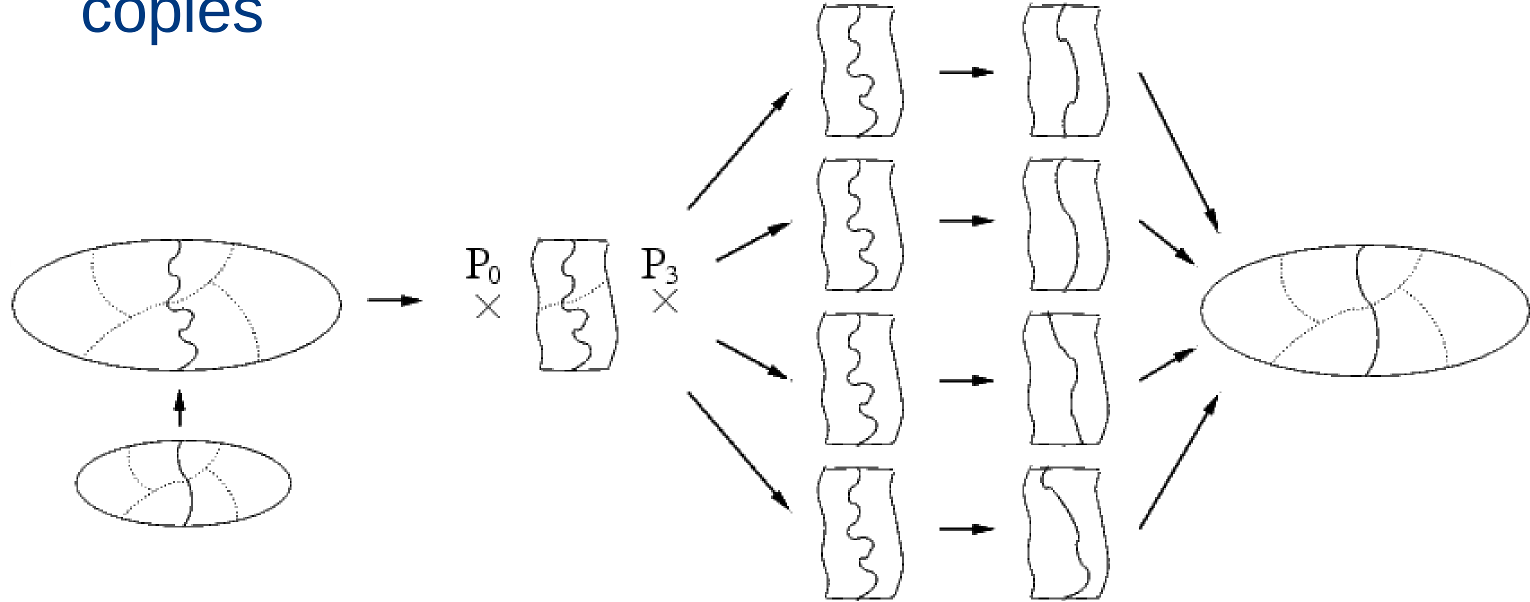


Band graph (2)

- Interests
  - Band graphs need only be of width 3 around the projected separator
    - Maximum distance at which fine vertices can be when their coarse vertices are neighbors
  - Since band graphs are several orders of magnitude smaller than full graphs, expensive algorithms can be applied to them more easily
  - Band graphs constrain refinement algorithms and prevent them from falling into local optima resulting from coarsening artifacts
- Distributed band graphs are easy to create

Multi-centralization (1)

- Principle [Chevalier & Pellegrini, 2008]
  - Since band graphs are supposed to be small, they can be multi-centralized such that sequential local optimization algorithms can still be applied to their copies



Multi-centralization (2)

Not scalable, but :

- Rather inexpensive for mesh graphs
- Yields results which are equivalent to, or even better than, the sequential version
  - Better exploration of problem space
- Is fine, to date, for sparse matrix ordering

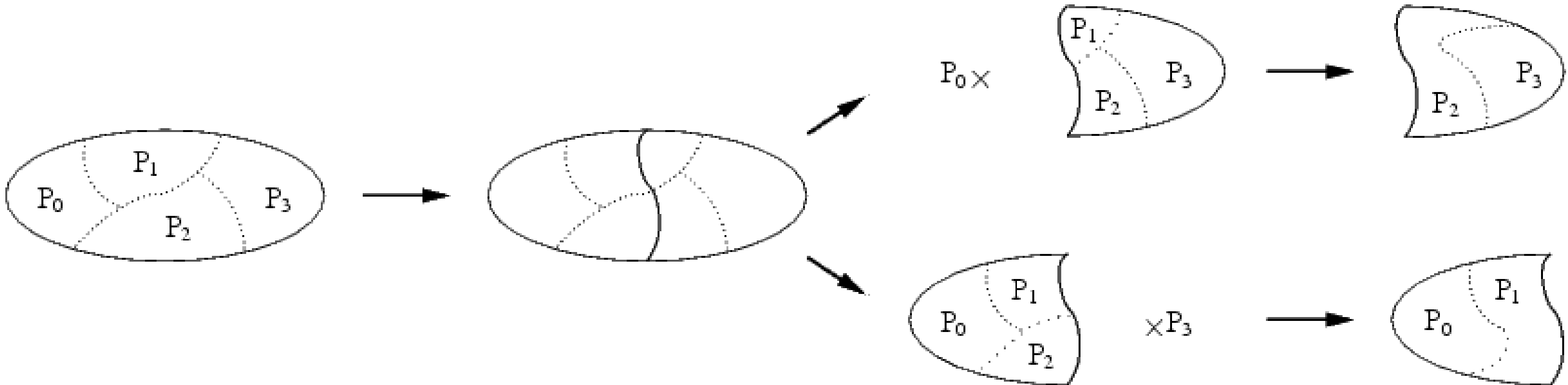
- Parallel algorithms can also be used
  - Genetic algorithms [Chevalier & Pellegrini, 2006]
  - Diffusion-based algorithms

Parallelization of nested dissection (1)

- Three levels of concurrency : [Chevalier & Pellegrini, 2006]
  - In the nested dissection process itself
    - Straightforward, coarse grain parallelism
    - Redistribution of subgraph data across processors
  - In the coarsening phase of the multi-level algorithm
    - Synchronous probabilistic matching algorithm
    - Folding and duplication in the coarser stages
  - In the refinement process during the uncoarsening phase
    - Multi-sequential optimization

Parallelization of nested dissection (2)

- After a separator has been computed, the two separated subgraphs are folded and redistributed each on a half of the available processors
  - The two sub-trees are separated logically but also physically, which reduces network congestion
  - Temporary folding thread (if MPI is thread-safe)



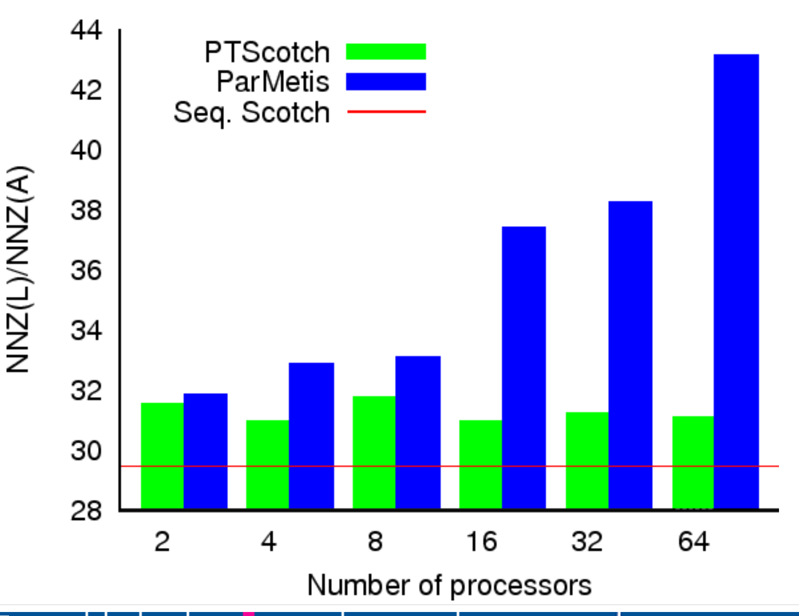
Results for parallel sparse matrix ordering (1)

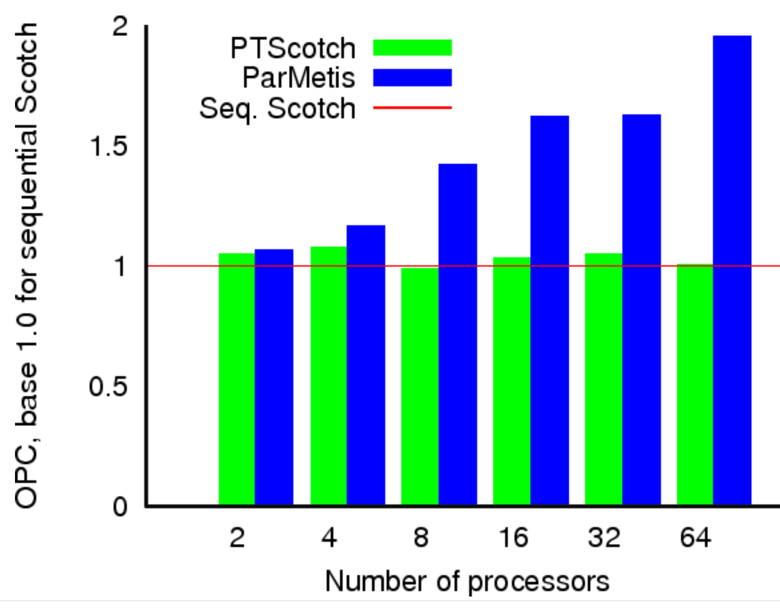
- Metrics are :
  - NNZ, the number of non zeros in L
  - OPC, the operation count of Cholesky factorization
- Indirect metrics
  - Many parameters impact solver performance
  - Can't even be computed a priori
    - We use separator size as the metric for bipartitions

Graph	Size ($\times 10^3$)		Average degree	O_{SS}	Description
	|V|	|E|			
audikw1	944	38354	81.28	5.48E+12	3D mechanics mesh, Parasol
cage15	5154	47022	18.24	4.06E+16	DNA electrophoresis, UU

Results for parallel sparse matrix ordering (2)

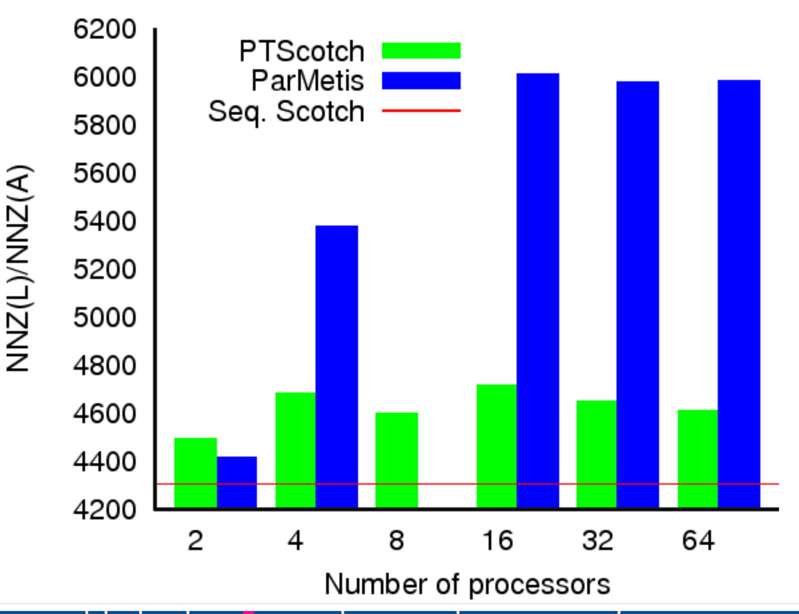
Test case	Number of processes					
	2	4	8	16	32	64
audikw1						
O_{PTS}	**5.73E+12**	**5.65E+12**	**5.54E+12**	**5.45E+12**	**5.45E+12**	**5.45E+12**
O_{PM}	5.82E+12	6.37E+12	7.78E+12	8.88E+12	8.91E+12	1.07E+13
t_{PTS}	64.14	43.72	31.25	20.66	13.86	9.83
t_{PM}	32.69	23.09	17.15	9.80	5.65	3.82

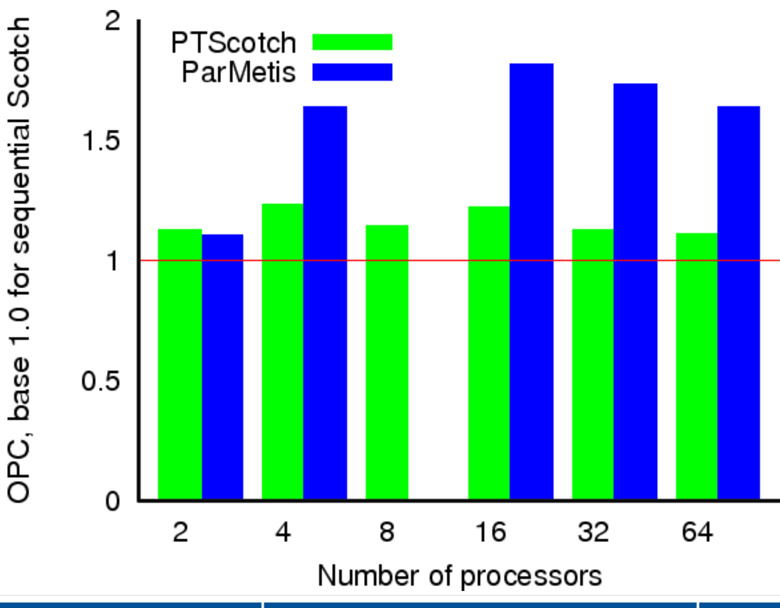




Results for parallel sparse matrix ordering (3)

Test case	Number of processes					
	2	4	8	16	32	64
cage15						
O_{PTS}	4.58E+16	**5.01E+16**	**4.64E+16**	**4.94E+16**	**4.58E+16**	**4.50E+16**
O_{PM}	**4.47E+16**	6.64E+16	†	7.36E+16	7.03E+16	6.64E+16
t_{PTS}	396.72	318.54	225.46	238.96	290.68	235.80
t_{PM}	195.93	117.77	†	40.30	22.56	17.83



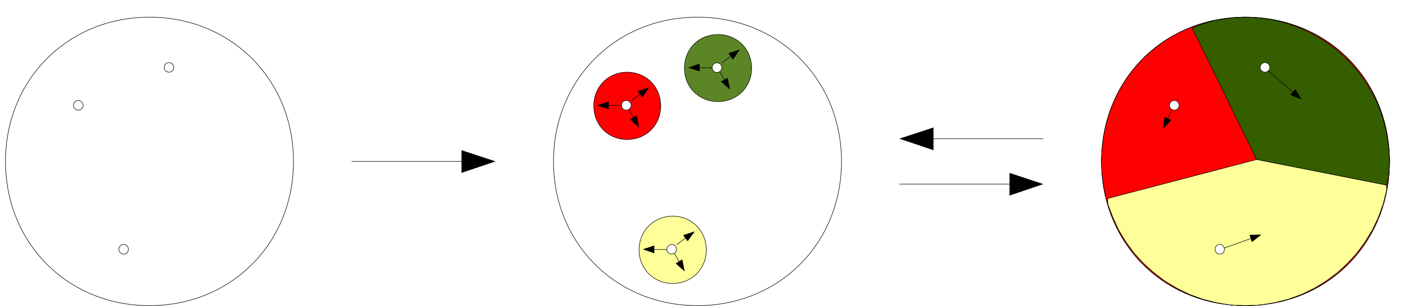


Recursive graph bipartitioning

- K-way graph partitioning can be approximated by a sequence of recursive bipartitionings
  - 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 for mesh graphs [Simon & Teng, 1993]

Diffusion algorithms (1)

- Principle [Walshaw, Cross & Everett, 1995]
  - Optimize shapes of subdomains by analogy with the auto-organization of soap bubbles with respect to the shape of their interfaces
  - Randomly select seeds, grow subdomains, and iterate to re-center seeds until convergence



Taken from [Meyerhenke & Schamberger, 2006]

Diffusion algorithms (2)

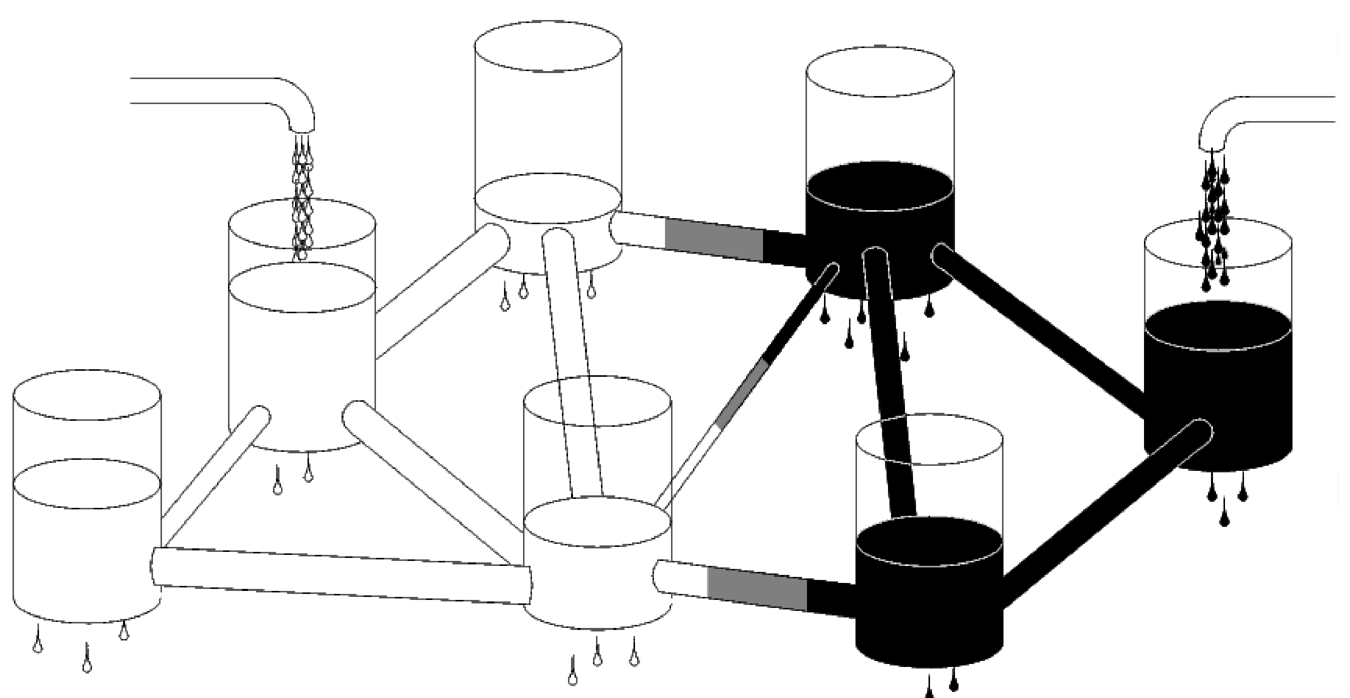
- Interest
  - Improves partition shapes for FEM iterative methods
  - Randomly select seeds, grow subdomains, and iterate to re-center seeds until convergence
- Drawbacks
  - Do not explicitly enforce load balance
  - Global iterative methods, slow for large graphs

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
  - Two antagonistic liquids (Scotch and anti-Scotch) flow from two source vertices
  - Liquids vanish when they meet

Jug of the Danaides (2)

- Sketch of the algorithm



Jug of the Danaides (3)

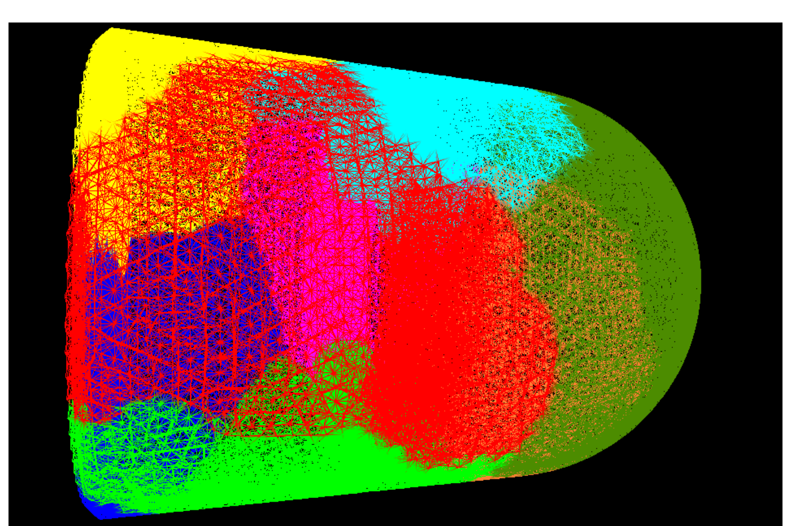
- Outline of the algorithm
  - Iterative algorithm
  - Every barrel leaks (at most) one unit of liquid per unit of vertex weight and of time
    - Similar to return drain edges in the Bubble-FOS/C algorithm of [Meyerhenke & Schamberger, 2006]
  - Injecting $|W_V|/2$ units of each of the liquids ensures convergence (whole system leaks at most $|W_v|$ per turn)
    - Anchor vertices of band graphs taken as sources
  - No need to wait for full convergence
    - We just want to know which liquid dominates in each of the barrels

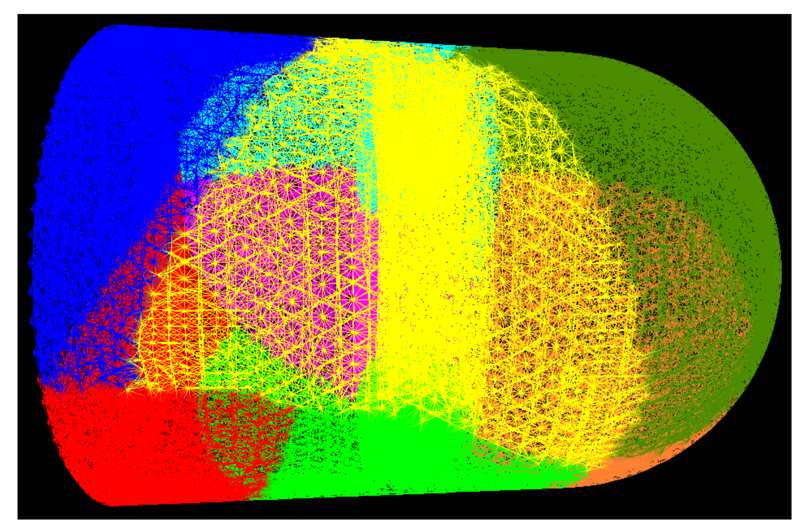
Jug of the Danaides (4)

- While the nature of the algorithm is very similar to diffusion methods, it has some specificities
  - Current diffusion-based methods compute and stabilize flows from each of the seeds, then select for each vertex the flow of maximum value
    - Data of size k|V| has to be maintained
    - Our algorithm elects the winner at each step and requires only data of size |E|
  - The amount of liquid leaked is not a fraction of the amount present on each vertex, but a fixed value
    - Flows cannot span on more than the prescribed amount of weights

Jug of the Danaides (5)

- Using JotD as the optimization algorithm in the multi-level process :
  - Smooths interfaces
  - Is slower than sequential FM (20 times for 500 iterations)





Jug of the Danaides (6)

- Average, on a set of test graphs, of recursive bipartitioning results with respect to cut size (ΔCut), load imbalance ratio (ΔMaCut) and maximum diameter of parts (ΔMDi), compared to multi-level banded Fiduccia-Mattheyses

Method	**RMBD**				**RMBDF**		**RMBaDF**
	500	200	100	40	500	40	40
ΔCut (%)	+19.51	+20.02	+18.15	+21.49	+2.26	+3.10	-3.17
Δ MaCut (%)	+0.58	+1.12	+1.80	+9.76	-0.95	-0.29	-0.21
Δ MDi (%)	+3.86	+1.92	+4.69	+5.43	+2.26	+3.10	-3.24
Δ Time ($\times$)	21.31	9.33	5.33	2.93	21.47	2.99	3.07

Parallel graph partitioning by R.B. (1)

- Sample test graphs

Graph	$\lvert V \rvert$ ($\times 10^3$)	$\lvert E \rvert$ ($\times 10^3$)	Avg.Deg.	Description
10MILLIONS	10424	78649	15.09	3D electromagnetics
23MILLIONS	23114	175686	15.20	3D electromagnetics
45MILLIONS	45241	335749	14.84	3D electromagnetics
82MILLIONS	82294	609508	14.81	3D electromagnetics
AUDIKW1	944	38354	81.28	3D mechanics mesh
BRGM	3699	151940	82.14	3D geophysics mesh
CAGE15	5154	47022	18.24	DNA electrophoresis
COUPOLE8000	1768	41657	47.12	3D structural mechanics
THREAD	30	2220	149.32	Connector problem

Runtime and partition quality (1)

PT-Scotch

45MILLIONS

Time (sec.) [log]

1000
100
10

1 10 100 1000

# of Proc [log]

PT-Scotch

45MILLIONS

Cut size

12000000
10000000
8 000000
6 000000
4 000000
2000000
0

1 10 100 1000

# of Proc [log]

- 2 parts
- 4 parts
- 8 parts
- 16 parts
- 32 parts
- 64 parts
- 128 parts
- 256 parts
- 512 parts
- 1024 parts
- 2048 parts

Runtime and partition quality (2)

PT-Scotch

82MILLIONS

Time (sec.) [log]

1000.00
100.00
10.00

1 10 100 1000

# of Proc. [log]

PT-Scotch

82MILLIONS

Cut size

16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0

1 10 100 1000

# of Proc. [log]

- 2 parts
- 4 parts
- 8 parts
- 16 parts
- 32 parts
- 64 parts
- 128 parts
- 256 parts
- 512 parts
- 1024 parts
- 2048 parts

Runtime and partition quality (3)

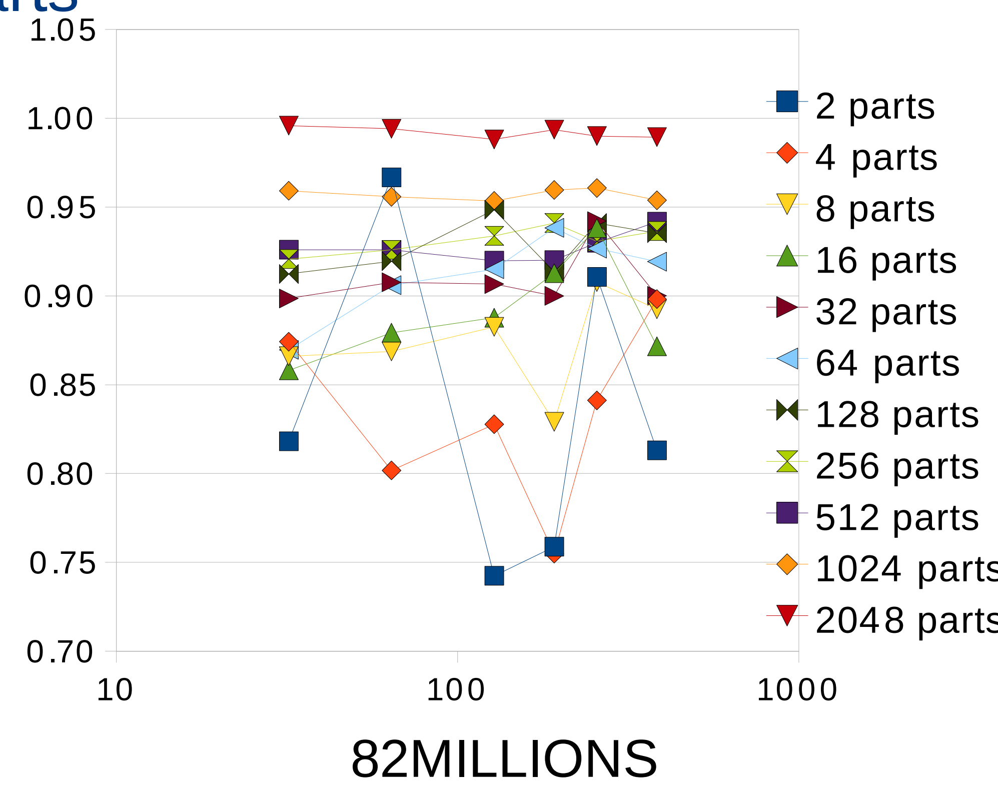
Test case	Number of processes:Number of parts						
	32:2	**32:32**	**32:1024**	**384:2**	**384:256**	**384:1024**	P_{Peak}**:2**
				45MILLIONS			
C_{PTS}	**1.15E+05**	**1.13E+06**	**7.24E+06**	**1.06E+05**	**3.65E+06**	**7.29E+06**	**1.05E+05**
C_{PM}	1.26E+05	1.38E+06	7.57E+06	1.39E+05	3.81E+06	7.62E+06	1.26E+05
t_{PTS}	**24.24**	102.29	150.56	**13.85**	28.08	30.04	**10.26(192)**
t_{PM}	84.55	48.24	36.21	28.72	25.65	23.15	21.51(256)
				82MILLIONS			
C_{PTS}	**1.46E+05**	**1.90E+06**	**1.08E+07**	**1.40E+05**	**5.57E+06**	**1.09E+07**	**1.45E+05**
C_{PM}	1.78E+05	2.12E+06	1.13E+07	1.73E+05	5.95E+06	1.14E+07	1.61E+05
t_{PTS}	**46.48**	189.42	297.76	**23.26**	46.91	61.54	**16.93(192)**
t_{PM}	176.4	85.87	76.42	32.83	30.22	26.9	30.00(256)

Runtime and partition quality (4)

Test case	Number of processes:Number of parts						
	32:2	32:32	32:1024	384:2	384:256	384:1024	P_{Peak}:2
AUDIKW1							
C_{PTS}	**1.08E+05**	2.08E+06	1.00E+07	**1.05E+05**	5.81E+06	9.96E+06	**1.11E+05**
C_{PM}	1.14E+05	2.04E+06	9.76E+06	1.15E+05	5.76E+06	9.76E+06	1.12E+05
t_{PTS}	**3.51**	11.84	17.35	5.87	10.72	10.06	3.01(128)
t_{PM}	3.9	3.59	5.27	4.45	4.62	4.51	2.37(192)
THREAD							
C_{PTS}	**5.60E+04**	6.15E+05	**1.82E+06**	**5.60E+04**	**1.29E+06**	**1.82E+06**	**5.62E+04**
C_{PM}	5.62E+04	6.03E+05	1.84E+06	5.73E+04	1.29E+06	1.84E+06	5.63E+04
t_{PTS}	**0.53**	0.97	1.07	**0.85**	1.27	1.28	**0.47(16)**
t_{PM}	0.77	0.75	1.99	2	0.89	2.07	0.52(8)

Runtime and partition quality (5)

- 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 global optimization (greedy algorithm)



Runtime and partition quality (6)

- In most cases, PT-Scotch produces better partitions
  - About 20% better when bipartitioning graph 82MILLIONS
- For a large number of parts, ParMeTiS provides slightly better for graphs AUDIKW1, THREAD and BRGM
  - These graphs have a high average degree
  - The greedy nature of recursive bipartitioning negatively impacts cut quality on the long term

Conclusion

Where we are now...

- Parallel sparse matrix ordering
  - Bottleneck removed for the near future
    - More work to be done as size of problems increases
  - Graph of 82+ million unknowns ordered and system solved by the PaStiX parallel direct solver on the Tera10 machine at CEA
- Parallel graph partitioning
  - Parallel k-way graph partitioning by recursive bipartitioning

The Scotch software package

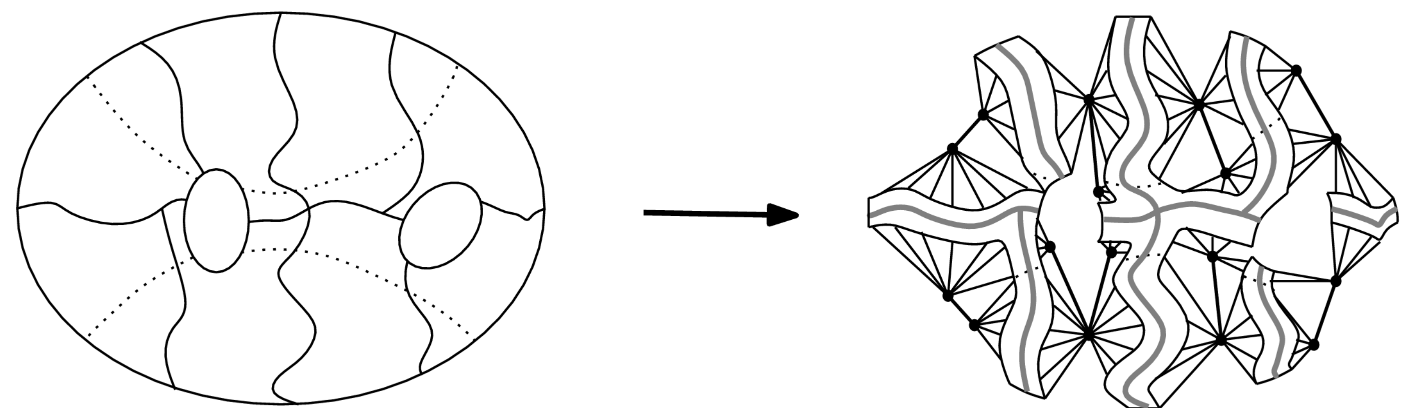
- All of the algorithms are available to the community
  - Scientific reproducibility
  - Freely available from the INRIA Gforge
  - Modular and documented code (≈100k lines of C)
- Upgrades on a regular basis
  - Version 4.0 : February 2004 : 2500+ direct downloads
  - About one major release per year (5.2 almost ready)
- Usage by third-party software
  - Emilio (CEA/CESTA), Code_Aster (EDF), Dolfin/Fenics (Simula), MUMPS (ENSEEITH, LIP & LaBRI), PaStiX (LaBRI), SuperLU (U. C. Berkeley), Zoltan (Sandia), ...

Where we are heading to...

- 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 (joint lab with INRIA)
- Impacts on our research :
  - Topology of target architecture has to be taken into account
    - Static mapping and not only graph partitioning
  - Dynamic repartitioning capabilities are mandatory

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
  - K-way band graphs are already available



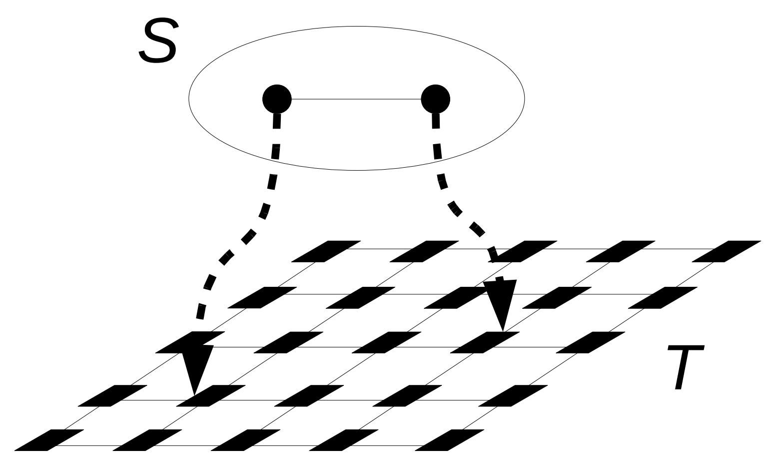
  - Stability problems with our diffusion-based algorithms

Parallel static mapping (1)

- 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

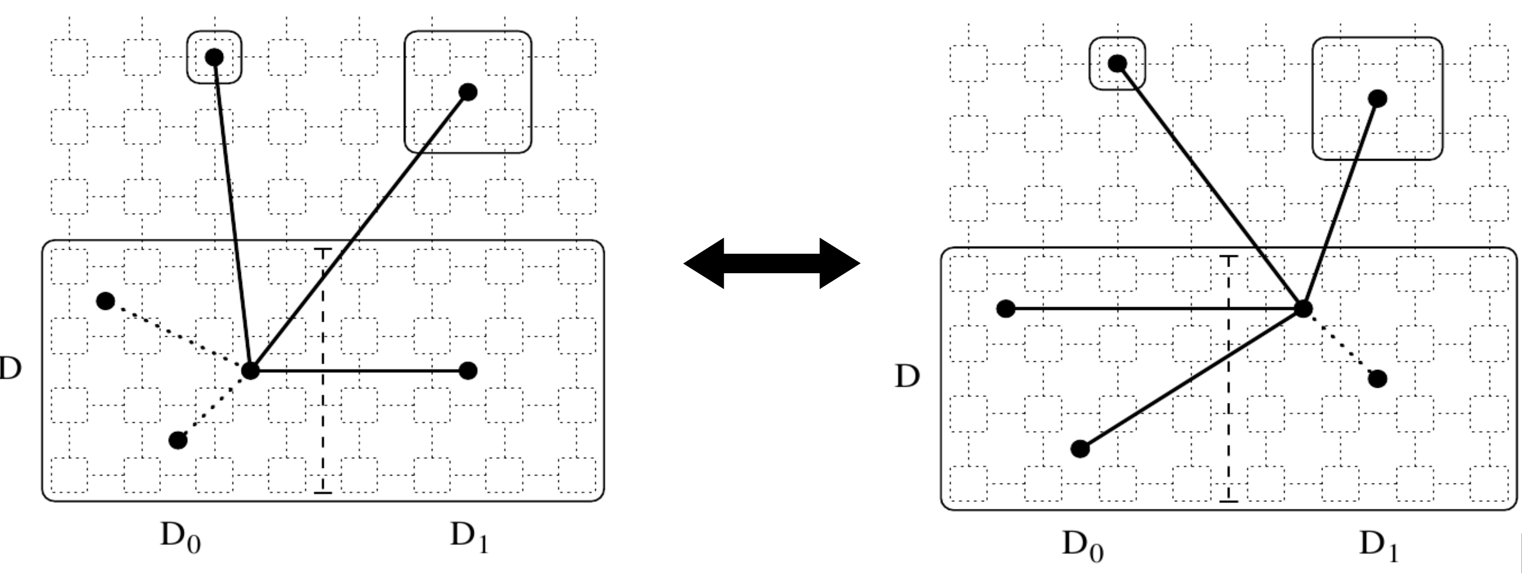


Parallel static mapping (2)

- Partial cost function in the context of recursive bipartitioning

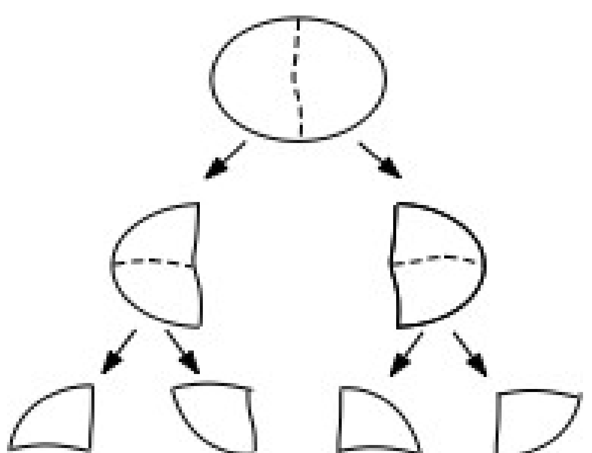
$$f'_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{\substack{v \in V(S') \\ \{v,v'\} \in E(S)}} w(\{v,v'\}) \, |\rho_{S,T}(\{v,v'\})|$$

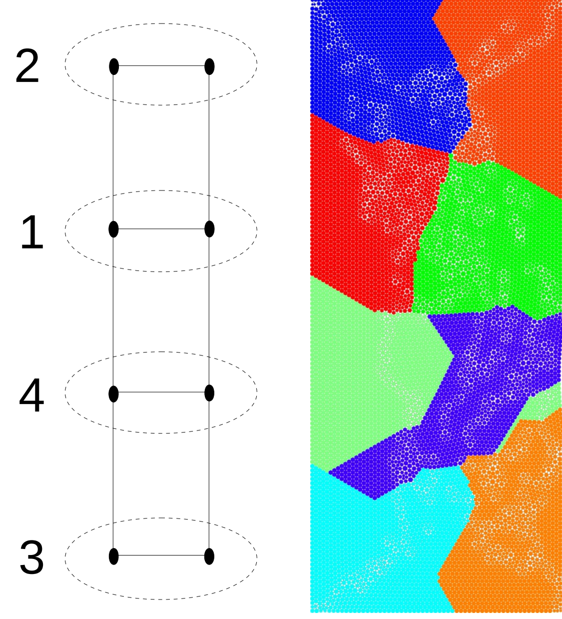
- Decision making depends on available mapping information



Parallel static mapping (3)

- 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”

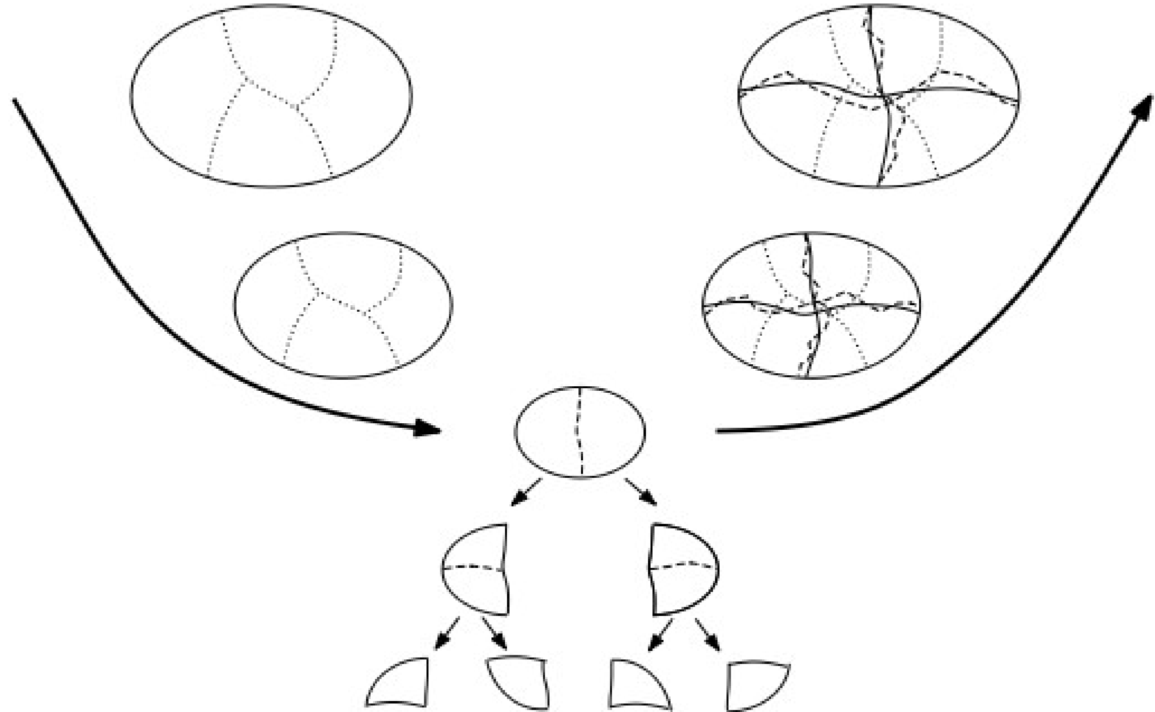




- Only sequential processing works!

Parallel static mapping (4)

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



Dynamic remeshing and repartitioning

- Move upwards from the production of general-purpose tools to more specific application domains
  - Motivation for joining the Bacchus team
- Parallel adaptive remeshing
  - Take into account the numerical stability of the problem being studied
  - Take advantage of the work done in PT-Scotch on distributed graphs
- Dynamically repartition the remeshed graphs

Thanks !

- To all the Scotch-men :
  - Cédric Chevalier
  - Jun-Ho Her
  - Sébastien Fourestier
  - Cédric Lachat