

# Efficient and scalable parallel graph partitioning

08/09/2008

INSTITUT NATIONAL  
DE RECHERCHE  
EN INFORMATIQUE  
ET EN AUTOMATIQUE



centre de recherche  
**BORDEAUX - SUD-OUEST**

Cédric Chevalier  
Jun-Ho Her  
François Pellegrini

# Summary of the talk

- Graph partitioning
- Sequential graph partitioning
- Parallel sparse matrix ordering
- Parallel graph partitioning
- On-going work



## The issues at stake

# Graph partitioning (1)

- Ubiquitous technique used in many application fields
  - Used to model domain-dependent optimization problems
  - Minimize vertex or edge cuts while balancing evenly graph parts
- NP-complete problem in the general case
- Many algorithms proposed to date :
  - Graph algorithms
  - Evolutionary algorithms
  - Spectral methods
  - Linear optimization methods



# Graph partitioning (2)

- Two main problems for our ScAIApplix team :
  - Domain decomposition for iterative methods
  - Sparse matrix ordering for direct methods
- These problems can be modeled as graph partitioning problems on the adjacency graph of symmetric matrices
  - Edge separator problem for domain decomposition
  - Vertex separator problem for sparse matrix ordering by nested dissection
  - Minimize separator size while maintaining balance between separated parts
    - Balance constraints may be strict or loose



# Graph partitioning (3)

- Two main classes of algorithms :
  - Global methods (e.g. genetic algorithms, simulated annealing)
    - Consider all of the graph data
    - Efficient but very slow
  - Local optimization heuristics (e.g. Fiduccia-Mattheyses)
    - Optimize an existing partition
    - Applied after global methods
    - Fast but not as efficient as global methods





# Constraints

- Problem size keeps increasing
  - Graphs of more than ten million vertices cannot be handled on sequential computers
  - Need for parallel graph partitioning tools
- Existing parallel tools evidence performance problems
  - Quality of partitions most often decrease in parallel when the number of processors increase
  - State-of-the art local optimization algorithms are intrinsically sequential and do not parallelize well



# The 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/hyper-graphs)
- Provide a software toolbox for scientific applications
  -  software package for sequential tools (existing)
  -  for parallel tools (to be created)

# Sequential graph partitioning

# 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 the type of graphs we are targeting [Simon & Teng, 1993]



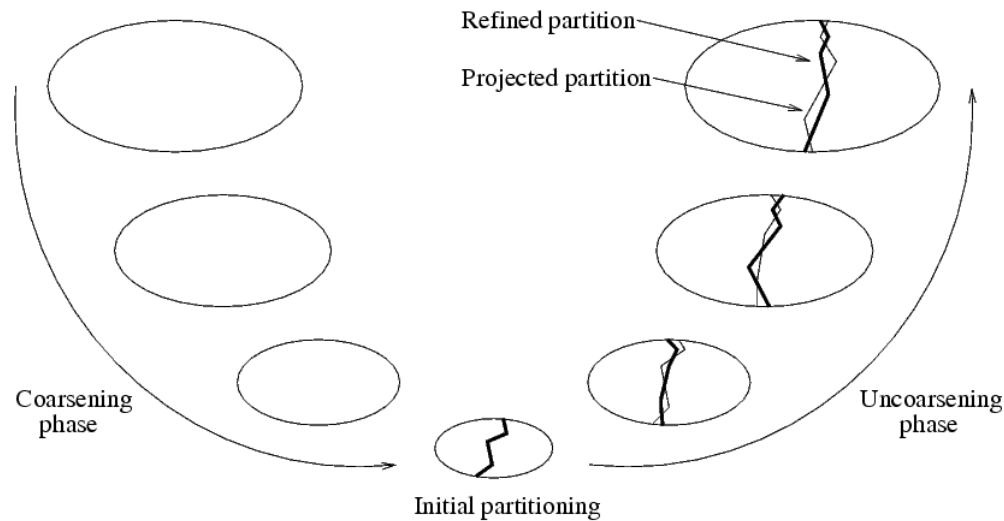
# Multi-level framework (1)

- Principle [Hendrickson & Leland, 1994]
  - Successive contractions of the initial graph by edge merging to obtain smaller graphs of same topological structure
  - Computation of an initial partition on the smallest graph
    - Global optimization methods
  - Projection of the partition to the finer graphs
    - Refinement of the projected partition by means of local optimization algorithms (such as Fiduccia-Mattheyses)



# Multi-level framework (1)

- Principle [Hendrickson & Leland, 1994]
  - Successive coarsenings by quotienting (matching)
  - Initial partitioning of the smallest graph
  - Propagation of the result with local refinement

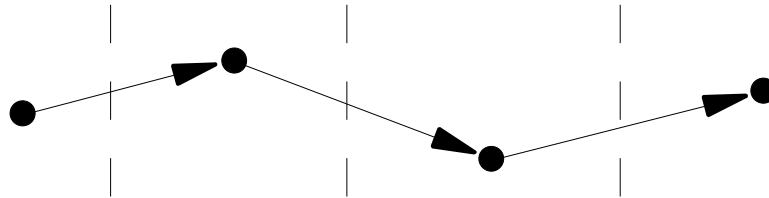


# Parallel matching (1)

- Parallel coarsening requires a parallel matching step, in which pairs of adjacent vertices are mated
  - Coarse graphs are built based on this matching
- 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
  - It requires much synchronization between processors which bear adjacent vertices
  - Graphs may be distributed in a way that yields much communication
    - Else nobody would need our stuff... ;-)

## Parallel matching (2)

- Synchronization between non-local neighbors is critical
  - Dependency chains or loops between mating requests can sequentialize the whole algorithm



- Several algorithms have been proposed to break dependency chains in the mating process



# Parallel matching by graph coloring

- Principle [Karypis and Kumar, 1999]
  - Coloring of the finer graph using Luby's algorithm
    - Based on random numbers
  - A sweep is made of as many rounds as there are colors in the graph coloring
  - Only vertices of the right color can ask for mating during their round
- Some colors have very few vertices
  - Partial sequentialization of the algorithm



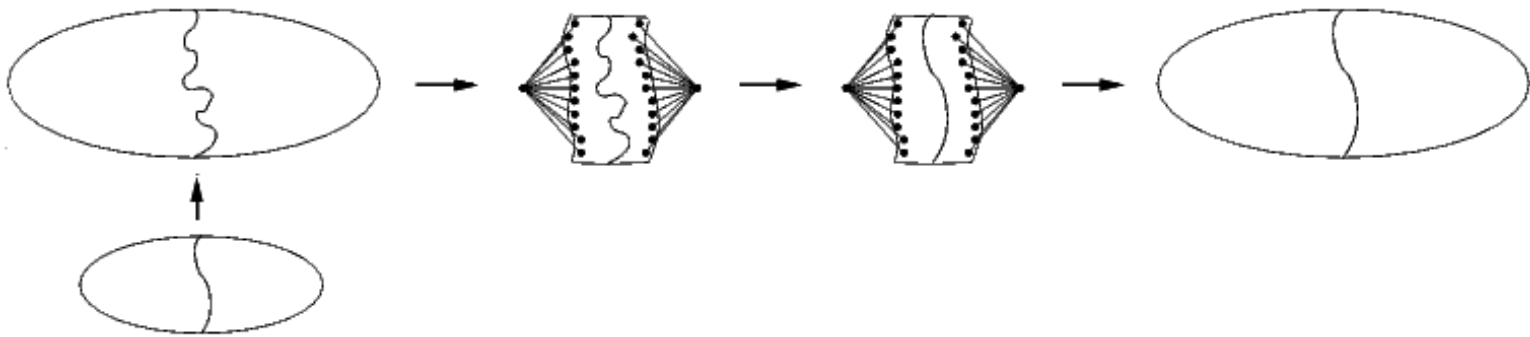
# Parallel probabilistic matching

- Principle [Chevalier, 2007]
  - Do not discriminate between local and non-local neighbors when selecting a neighbor for mating
  - Vertices request for matings with distant neighbors with a probability which depends on the local and external degrees of their neighbors
- Reduces topological biases
- Improves mating probability when data are irregularly distributed



# Band graph (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
  - 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 in local optima resulting from coarsening artifacts



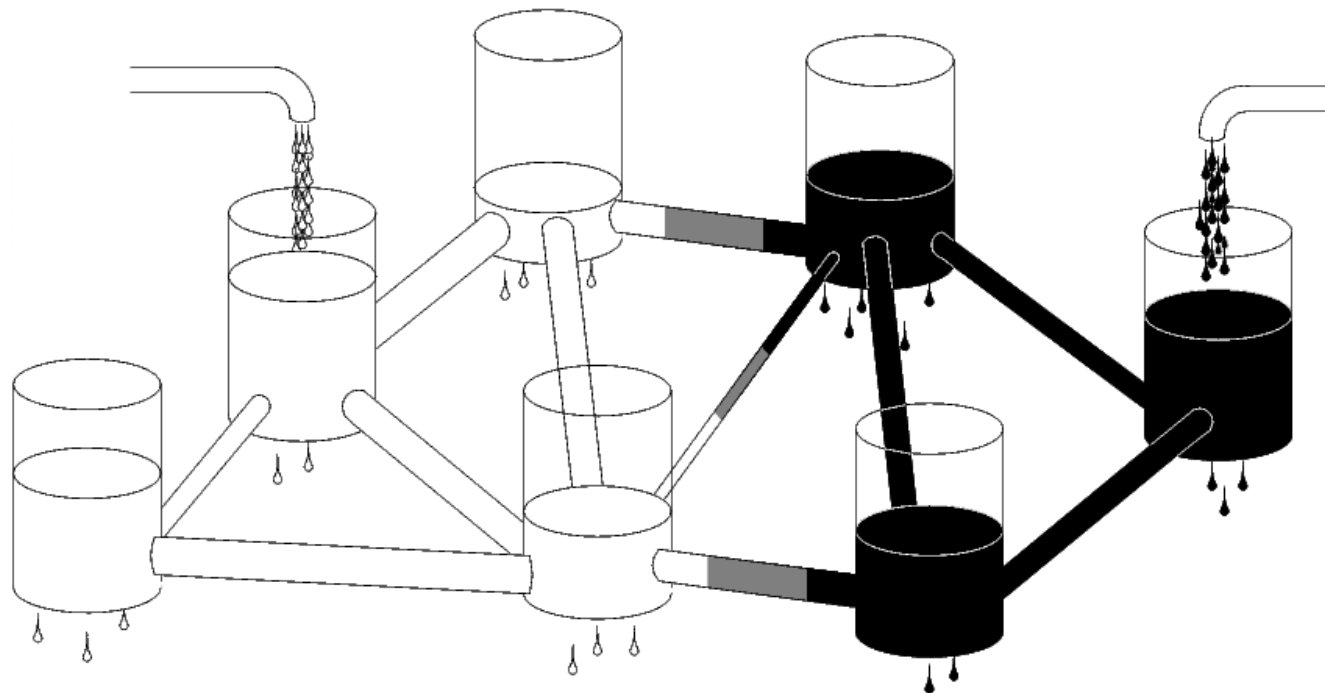
# Jug of the Danaides (1)

- 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



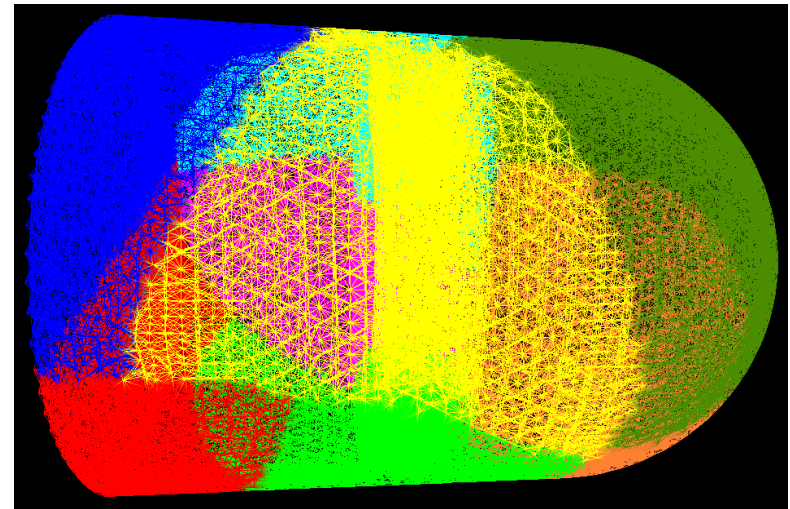
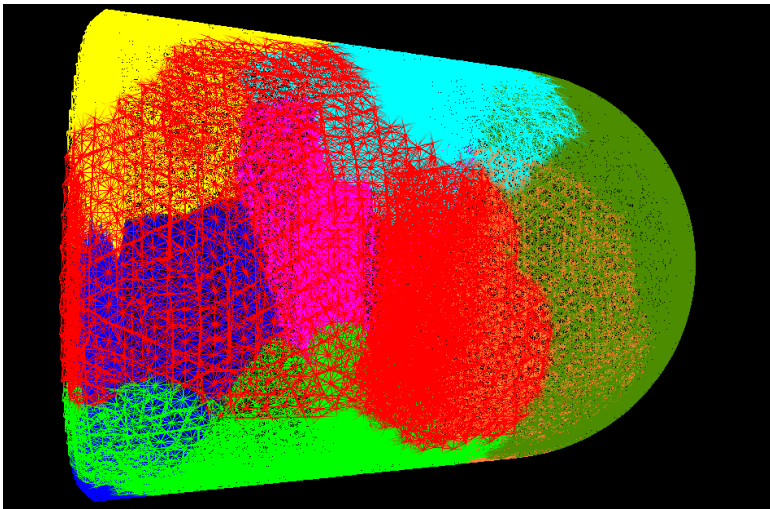
# Jug of the Danaïdes (2)

- Sketch of the algorithm



# Results (1)

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



## Results (2)

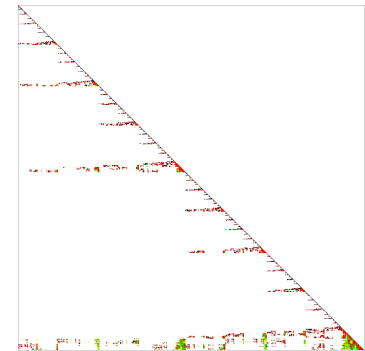
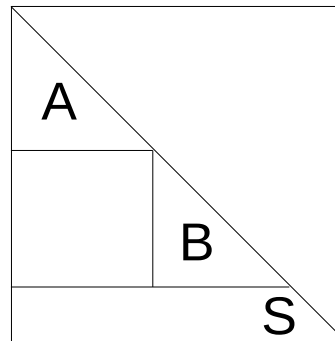
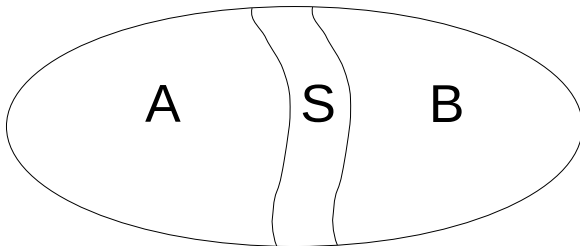
- Average, on a set of test graphs, of recursive bipartitioning results with respect to cut size ( $\Delta\text{Cut}$ ), load imbalance ratio ( $\Delta\text{MaCut}$ ) and maximum diameter of parts ( $\Delta\text{MDi}$ ), compared to multi-level banded Fiduccia-Mattheyses

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

# Parallel sparse matrix ordering

# Nested dissection

- 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



# Parallelization of multi-level nested dissection

- Three levels of concurrency :
  - 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



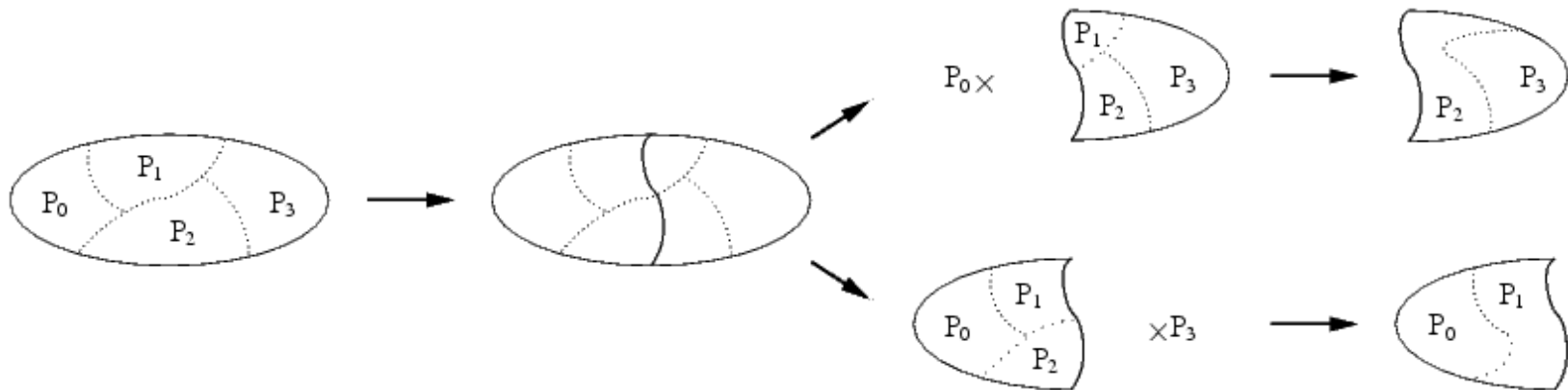
# Parallelization of nested dissection (1)

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



# Parallelization of nested dissection (2)

- The two sub-trees are separated logically but also physically, which reduces network congestion
- The computation of the two induced subgraphs and their folding can be performed in parallel thanks to the creation of a temporary thread per processor (if MPI is thread-safe)



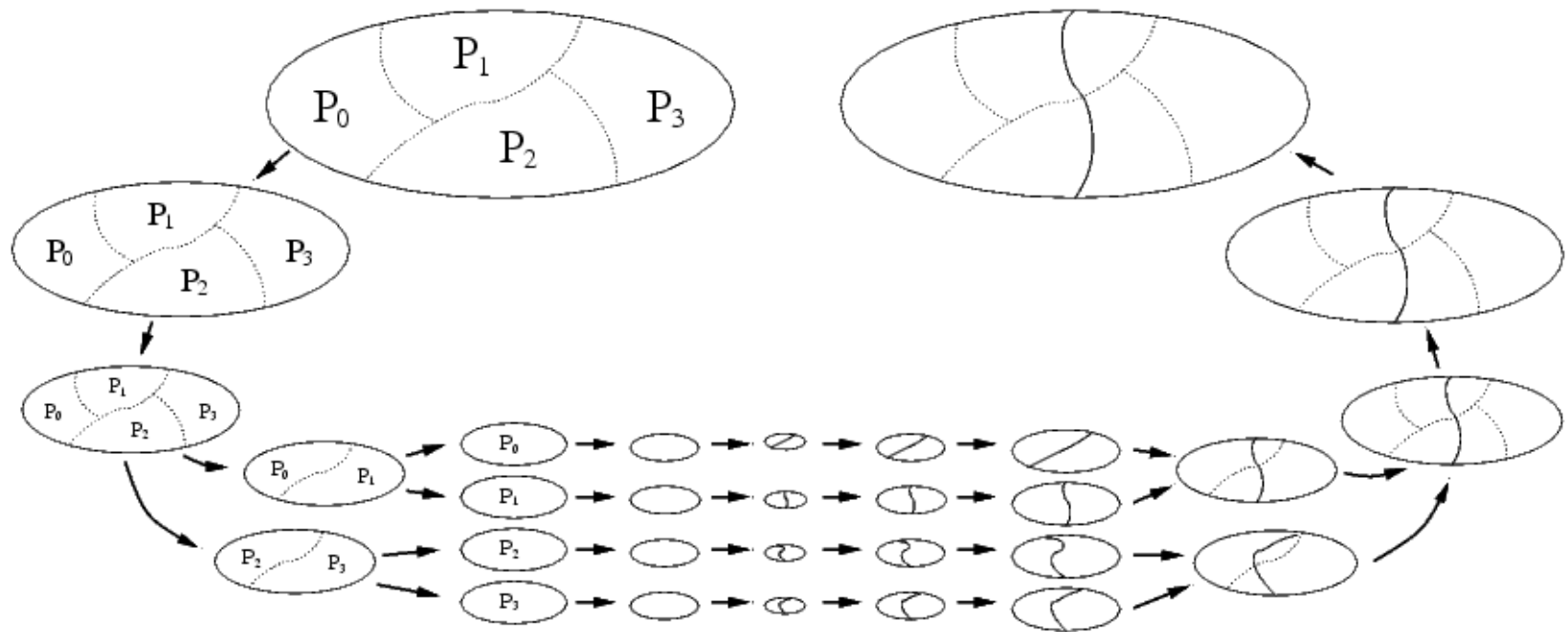
# Parallelization of the coarsening phase (1)

- Matchings are performed in parallel
  - Several algorithms (synchronous or asynchronous) have been studied to reduce dependencies between mating decisions
- 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



# Parallelization of the coarsening phase (2)

- It is preferable to use folding and duplication only in the last stages of the coarsening process



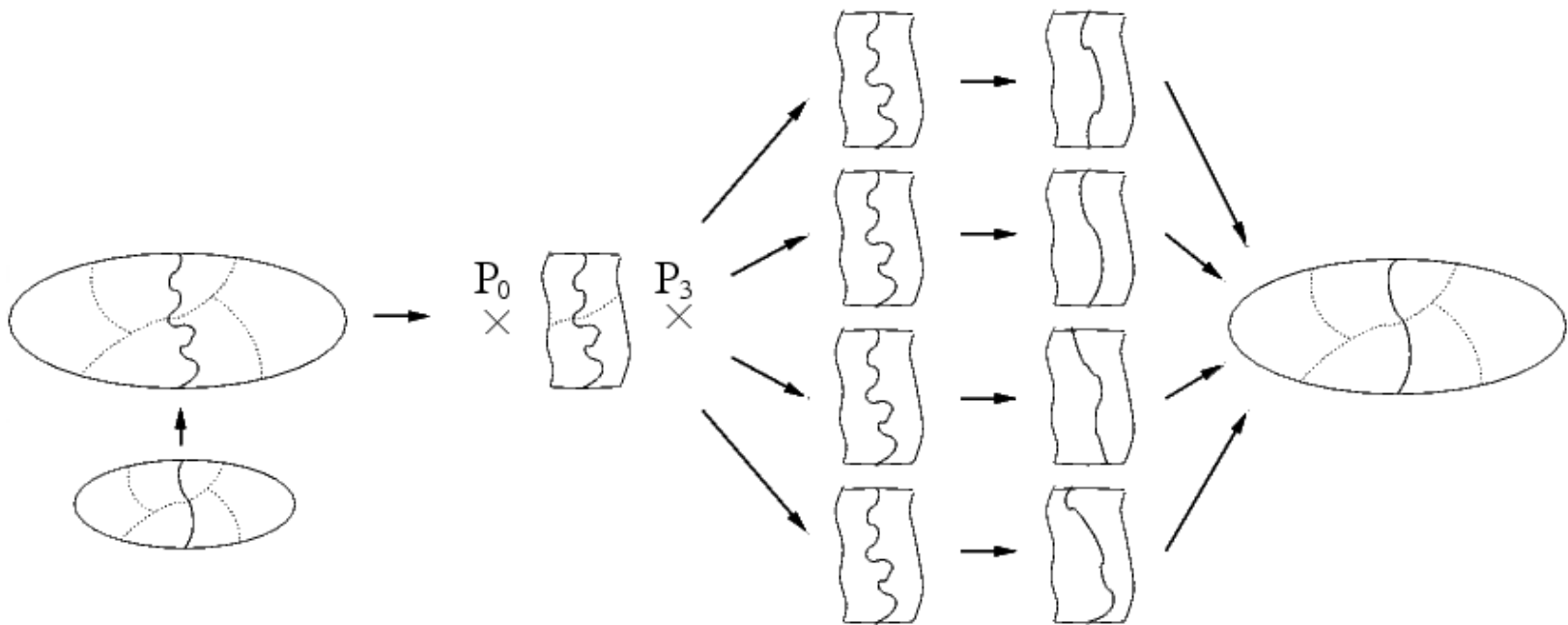
# Parallelization of the refinement phase (1)

- As in the sequential algorithm, a distributed band graph is built by keeping only vertices which are at some small distance from the projected separator
- Since this graph is very small, it can be multi-centralized such that sequential local optimization algorithms can be applied to its copies
  - Not scalable but rather inexpensive and yields results which are equivalent to or even better than the sequential version



# Parallelization of the refinement phase (2)

- Parallel algorithms can also be used
  - Genetic algorithms
  - Diffusion algorithms



# Results (1)

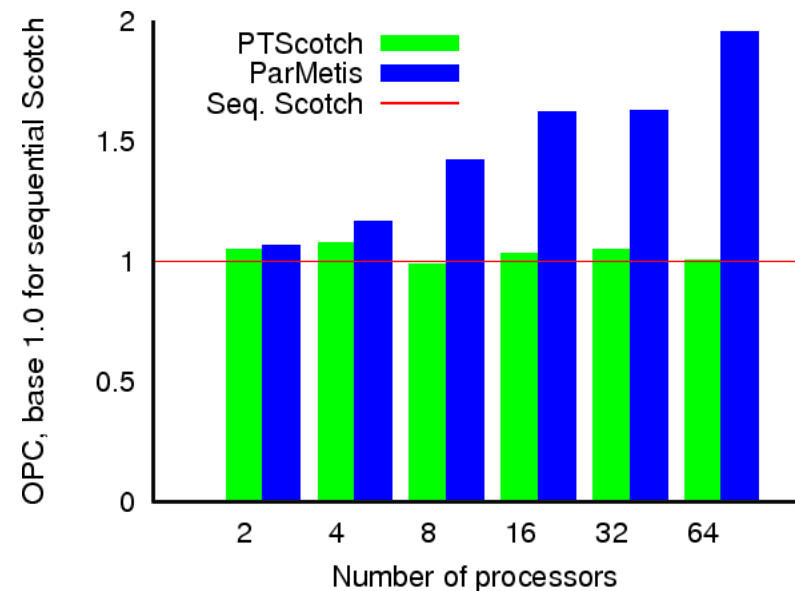
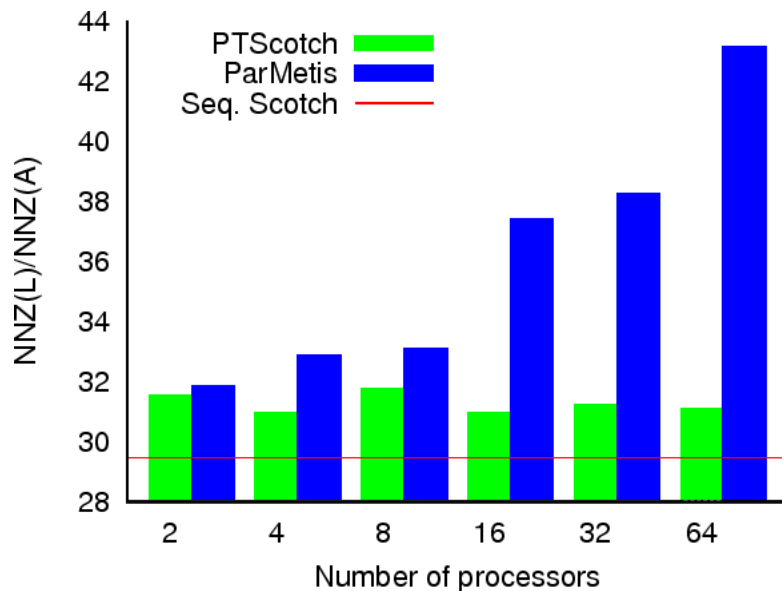
- Metric is OPC, the operation count of Cholesky factorization

Graph	Size ( $\times 10^3$ )		Average degree	$O_{ss}$	Description
	V	E			
audikw1	944	38354	81.28	5.48E+12	3D mechanics mesh, Parasol
brgm	3699	151940	82.14	2.70E+13	3D geophysics mesh, BRGM
cage15	5154	47022	18.24	4.06E+16	DNA electrophoresis, UF
quimonda07	8613	29143	6.76	8.92E+10	Circuit simulation, Quimonda
23millions	23114	175686	7.6	1.29E+14	CEA/CESTA



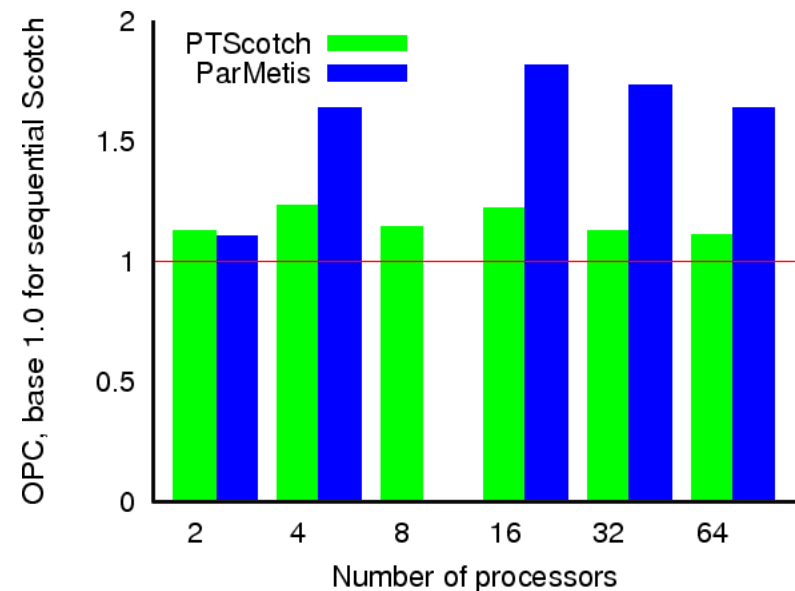
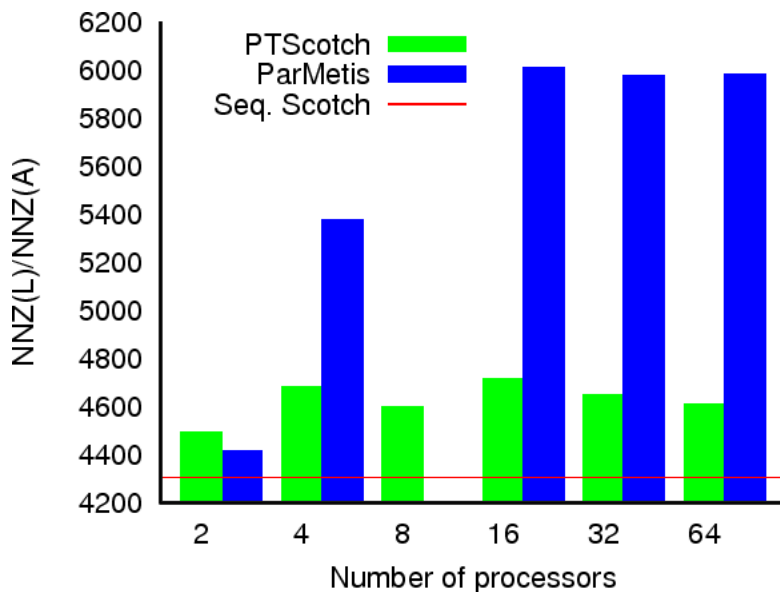
# Results (2)

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
$t_{PTS}$	73.11	53.19	45.19	33.83	24.74	18.16
$t_{PM}$	32.69	23.09	17.15	9.80	5.65	3.82



# Results (3)

Test case	Number of processes					
	2	4	8	16	32	64
cage15						
$O_{PTS}$	4.58E+16	<b>5.01E+16</b>	<b>4.64E+16</b>	<b>4.94E+16</b>	<b>4.58E+16</b>	<b>4.50E+16</b>
$O_{PM}$	<b>4.47E+16</b>	6.64E+16	†	7.36E+16	7.03E+16	6.64E+16
$t_{PTS}$	540.46	427.38	371.70	340.78	351.38	380.69
$t_{PM}$	195.93	117.77	†	40.30	22.56	17.83



# On-going work

- Largest case processed to date :
  - Full 3D, 83+ million unknowns graph
  - Linear system ordered and solved by our PaStiX direct solver on the TERA-10 machine of CEA
  - 100+ million cases to be solved by means of our HIPS hybrid solver
- Interfacing with parallel symbolic factorization
  - MUMPS solver
  - PaStiX solver



# Parallel graph partitioning

# Parallelization of k-way partitioning

- At first, by means of recursive graph bipartitioning
- Three levels of concurrency :
  - In the recursive bipartitioning 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



# Parallelization of the refinement phase

- As in the sequential algorithm, a distributed band graph is built by keeping only vertices which are at some small distance from the projected separator
- Local optimization algorithms are run on the distributed band graph only
  - Parallel diffusion
- Since this graph is very small, it can be multi-centralized such that sequential local optimization algorithms can be applied to its copies
  - Not scalable but rather inexpensive and yields results which are equivalent to or even better than the sequential version



# Results

- Two strategies were tried : both use parallel multi-level and band graph refinement, but have a different band local optimization method
  - The first one uses sequential FM (PTS-BSFM)
  - The second one uses parallel diffusion (PTS-BD)
  - Imbalance ratio of at most 0.5 %
- Bipartition: comparison with ParMeTiS
- K-way partitioning for 10Millions graph
- Tests run on an IBM cluster of 4 SMP nodes with 8 dual-core Power 5 processors each
  - 64 cores at most



# Bipartition results - PTS-BD vs. PTS-BSFM (1)

Graph	2 Proc			4 Proc			8 Proc		
	PTS-BD	PTS-BSFM	%	PTS-BD	PTS-BSFM	%	PTS-BD	PTS-BSFM	%
altr4	1750	2068	-15.38	1678	1921	-12.65	1820	2015	-9.68
bmw32	20009	17075	17.18	15524	18800	-17.43	17673	17013	3.88
conosphere1m	22061	26434	-16.54	22167	26566	-16.56	22141	29464	-24.85
audikw1	107865	114904	-6.13	114354	116409	-1.77	108882	113469	-4.04
coupole8000	3079	3223	-4.47	3079	3172	-2.93	3089	3143	-1.72
10Millions	47420	58394	-18.79	45959	62260	-26.18	47543	91608	-48.1
23Millions	91427	116131	-21.27	92201	126237	-26.96	90610	125139	-27.59
thread	56214	56977	-1.34	55836	56249	-0.73	55890	56845	-1.68
Graph	16 Proc			32 Proc			64 Proc		
	PTS-BD	PTS-BSFM	%	PTS-BD	PTS-BSFM	%	PTS-BD	PTS-BSFM	%
altr4	1804	1886	-4.35	1704	1892	-9.94	1760	1900	-7.37
bmw32	16130	17285	-6.68	14775	16046	-7.92	18643	16871	10.5
conosphere1m	21658	29126	-25.64	22067	29661	-25.6	21952	29449	-25.46
audikw1	116235	111945	3.83	108027	111241	-2.89	108801	111190	-2.15
coupole8000	3091	3130	-1.25	3089	3179	-2.83	3079	3218	-4.32
10Millions	46629	65325	-28.62	46564	80804	-42.37	46114	71366	-35.38
23Millions	91397	129688	-29.53	92078	128794	-28.51			
thread	55917	56081	-0.29	56205	56214	-0.02	55863	56669	-1.42
<b>AVERAGE GAIN :</b>			<b>-12</b>						

# Bipartition results - PTS-BD vs. PTS-BSFM (2)

- Cut size of PTS-BD is 12 % better for (first) bipartition than PTS-BSFM
- Speed of PTS-BD is 20 to 60 % better (500 diffusion steps)



# Bipartition results - PTS-BD vs. ParMeTis (1)

Graph	2 Proc			4 Proc			8 Proc		
	PTS-BD	ParMeTiS	%	PTS-BD	ParMeTiS	%	PTS-BD	ParMeTiS	%
altr4	1750	2101	-16.71	1678	1696	-1.06	1820	1729	5.26
bmw32	20009	17571	13.88	15524	18912	-17.91	17673	18294	-3.39
conesphere1m	22061	25311	-12.84	22167	23405	-5.29	22141	21757	1.76
audikw1	107865	106128	1.64	114354	104957	8.95	108882	109673	-0.72
couple8000	3079	3448	-10.7	3079	3409	-9.68	3089	3145	-1.78
10Millions	47420			45959	53869	-14.68	47543	50109	-5.12
23Millions	91427			92201			90610		
thread	56214	57074	-1.51	55836	56106	-0.48	55890	56304	-0.74
Graph	16 Proc			32 Proc			64 Proc		
	PTS-BD	ParMeTiS	%	PTS-BD	ParMeTiS	%	PTS-BD	ParMeTiS	%
altr4	1804	1634	10.4	1704	1719	-0.87	1760	1773	-0.73
bmw32	16130	18042	-10.6	14775	17933	-17.61	18643	17874	4.3
conesphere1m	21658	23933	-9.51	22067	24156	-8.65	21952	22414	-2.06
audikw1	116235	116737	-0.43	108027	110149	-1.93	108801	115506	-5.8
couple8000	3091	6297	<b>-50.91</b>	3089	3174	-2.68	3079	3157	-2.47
10Millions	46629	50509	-7.68	46564	52072	-10.58	46114		
23Millions	91397			92078					
thread	55917	56670	-1.33	56205	56547	-0.6	55863	57186	-2.31
<b>AVERAGE GAIN :</b>			<b>-3.56</b>						

# Bipartition results - PTS-BD vs. ParMeTis (2)

- Cut size of PTS-BD is 3.5 % better for (first) bipartition than ParMeTiS for cases where the latter provides valid results
- Slower than ParMeTiS by a factor 10 to 15



# K-way partition results - for 10 Millions graph

## — Execution time (sec.)

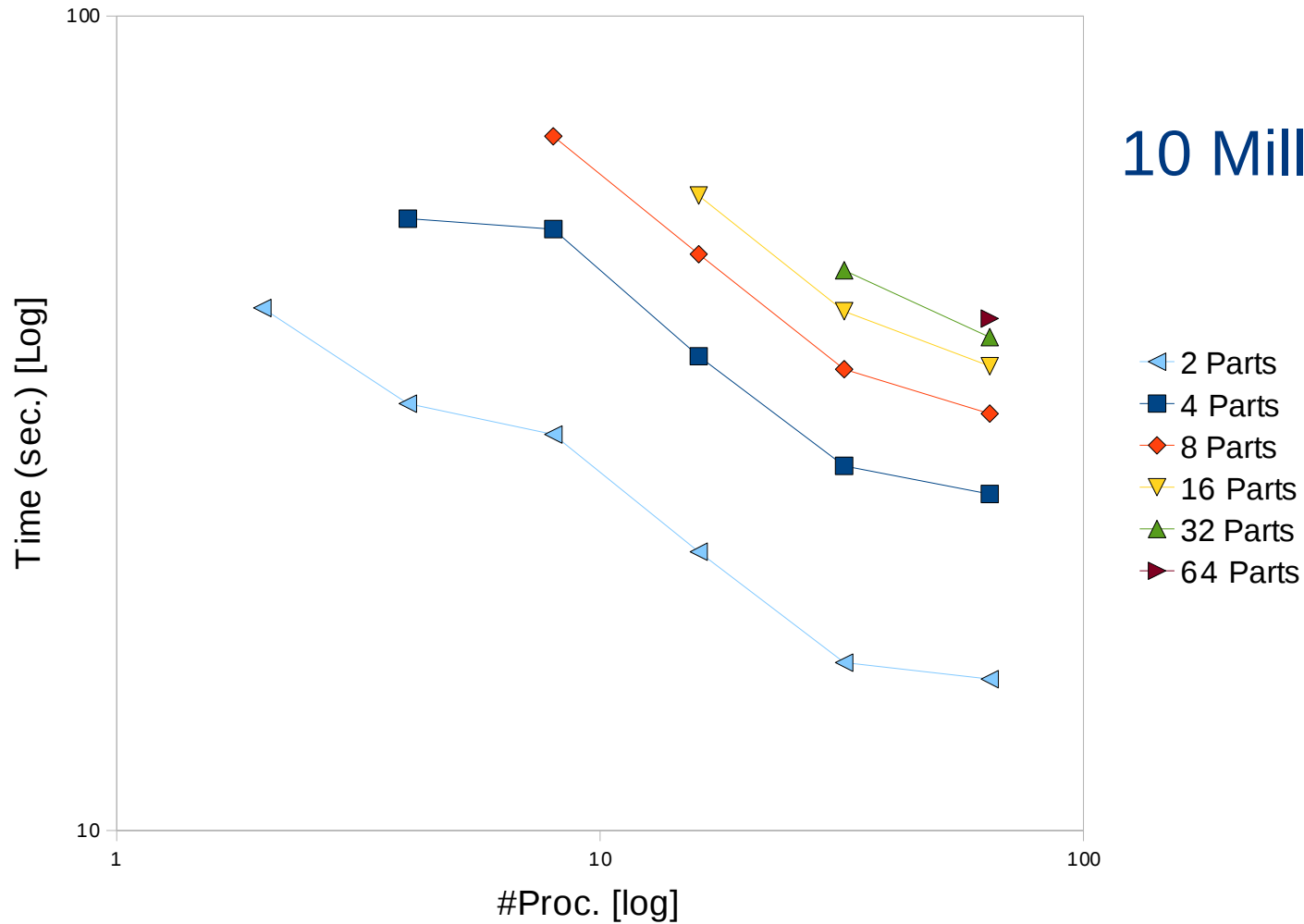
	2 Proc	4 Proc	8 Proc	16 Proc	32 Proc	64 Proc
2 Parts	43.81	33.42	30.64	21.99	16.08	15.34
4 Parts		56.38	54.74	38.22	28.03	25.89
8 Parts			71.17	50.99	36.84	32.5
16 Parts				60.19	43.4	37.16
32 Parts					48.72	40.34
64 Parts						42.51

## — Cut size

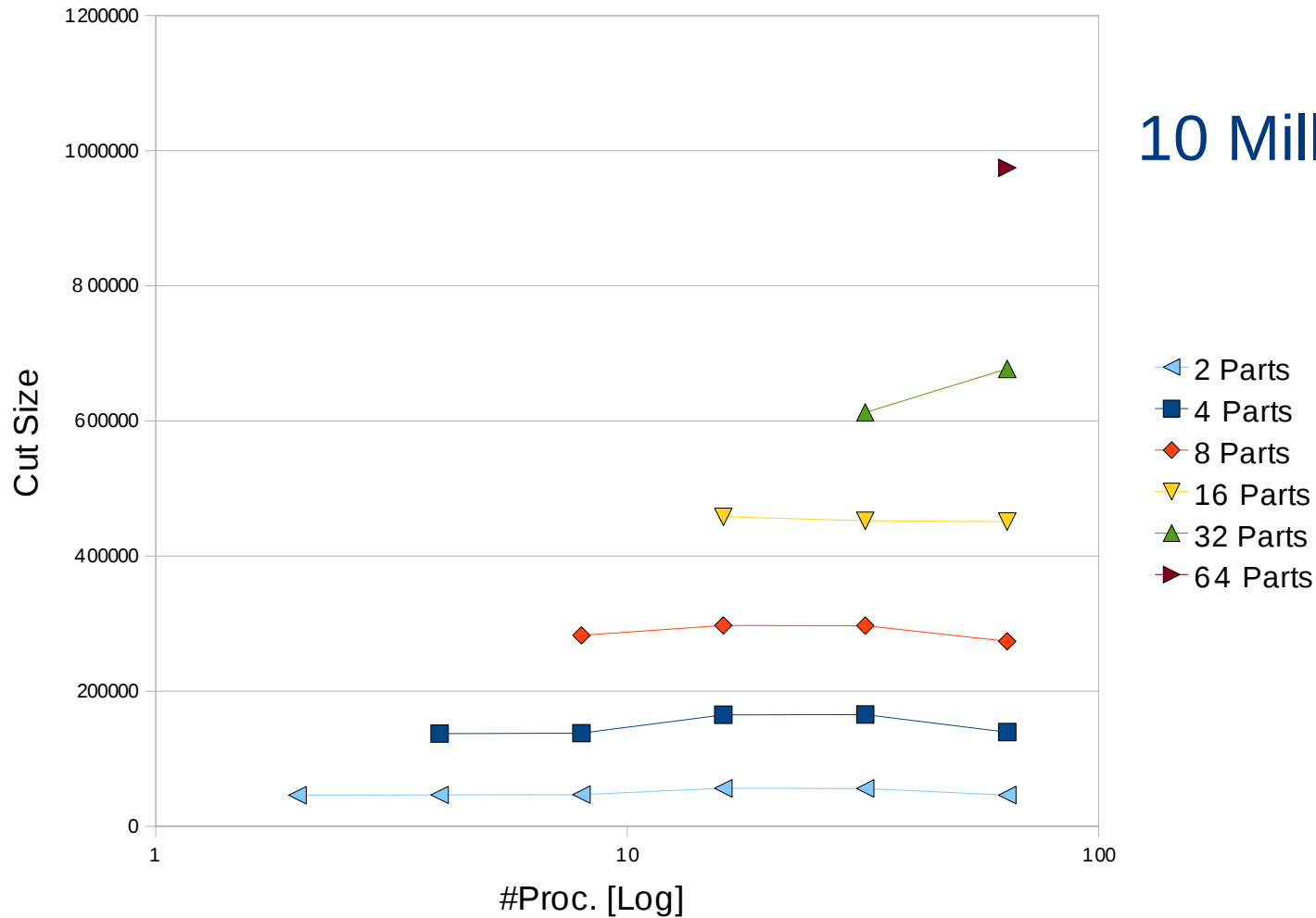
	2 Proc	4 Proc	8 Proc	16 Proc	32 Proc	64 Proc
2 Parts	46015	46433	46976	56187	56025	46199
4 Parts		137203	137916	165079	165436	139423
8 Parts			282687	297302	297113	273845
16 Parts				458106	452253	450898
32 Parts					612579	677112
64 Parts						974725



# K-way partition results - scalability



# K-way partition results - partition quality



## Where we are now

INSTITUT NATIONAL  
DE RECHERCHE  
EN INFORMATIQUE  
ET EN AUTOMATIQUE




centre de recherche  
**BORDEAUX - SUD-OUEST**

# The software package

- All of the algorithms are available to the community
  - Scientific reproducibility
  - External contributions through open-source licensing
  - Freely available from the INRIA Gforge  
<http://www.labri.fr/~pelegrin/scotch/>
- Version 5.1 will be publicly available this week



# On-going work

- Parallel direct k-way partitioning algorithms
  - Parallel graph bipartitioning was a shortcut
- Parallel static mapping algorithms
  -  can do static mapping and not only graph partitioning
  - This has been found very efficient for NUMA architectures [Fourestier 2008, to be published]
- Parallel dynamic repartitioning

