Design and implementation of efficient tools for parallel partitioning and distribution of very large numerical problems

Cédric Chevalier

SCALAPPLIX project
LaBRI and INRIA Futurs
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, FRANCE

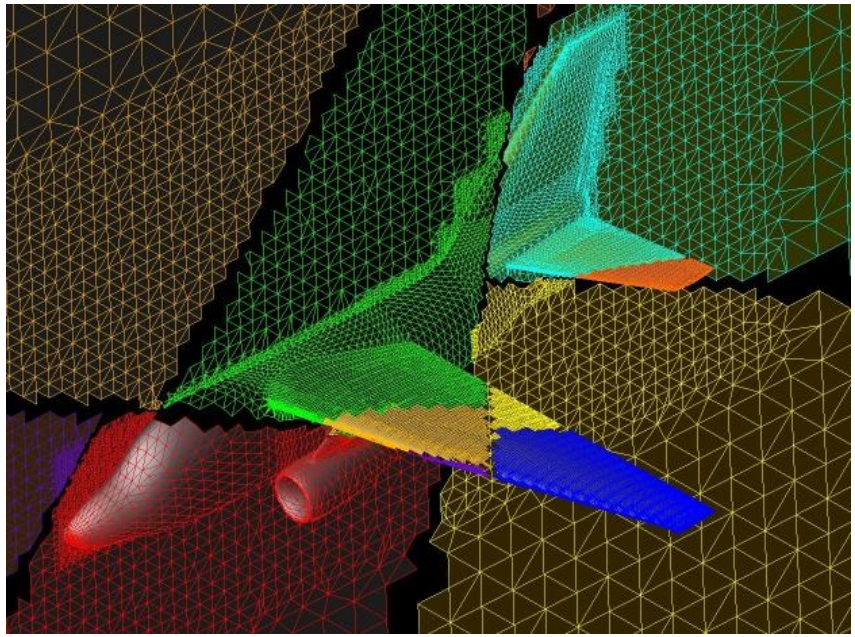
PhD Defense, 28 Sep 2007

Outline

INRIA FUTURS

Applications and graph partitioning

- Numerical simulations are essential in various domains including :
  - mechanics
  - fluid dynamics
  - computational chemistry, . . .
- Problems are more and more elaborate: numerical simulation cannot be run on a single workstation
  → Need of parallelized computations
  → Need to know how to distribute data across the processors: domain decomposition

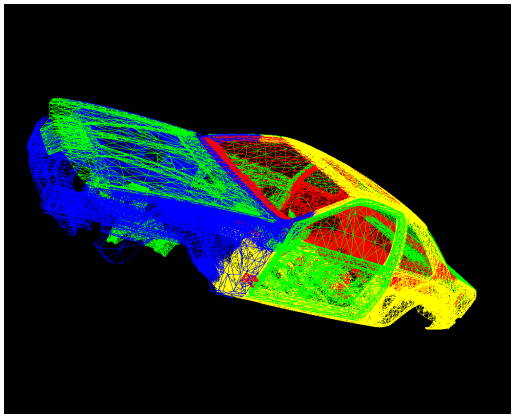


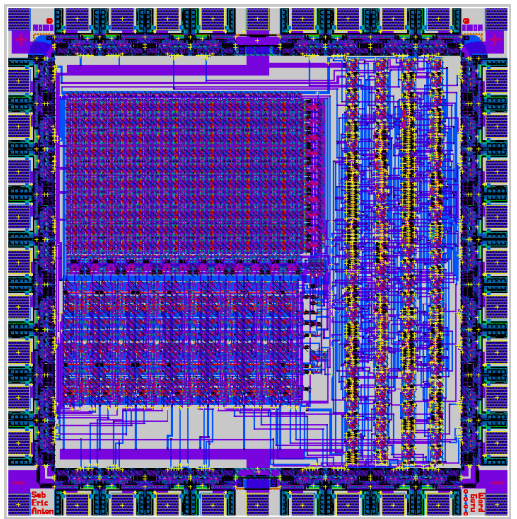
INRIA FUTURS

Applications and graph partitioning

Work load distribution can be modeled as a graph partitioning problem and is mandatory in :

- domain decomposition
- sparse linear algebra problems
- VLSI circuit design
- bio-informatics
- computer imaging
- scheduling and load balancing
- . . .





INRIA FUTURS

Graph Partitioning

Process which consists in dividing vertices of a graph into a given number of sets, while enforcing two typical constraints:

1. Boundary constraint: the size of the interface between parts should be as small as possible
2. Balance constraint: all sets should be evenly weighted

Additional constraints can be considered, such as:

- Connectivity of parts, especially in VLSI design
- Compactness of parts (aspect ratio), in domain decomposition
- ...

Difficulty

Problem is NP-Complete [Garey, Johnson and Stockmeyer, 1976]
$\Rightarrow$ We have only heuristics

INRIA FUTURS

Our tool SCOTCH ?

SCOTCH is a graph partitioning, sparse matrix ordering and static mapping tool developed in team SCALAPPLIX
SCOTCH 5.0 software package provides :

1. A sequential library (SCOTCH) which contains :
   - A static mapper
   - A graph and mesh ($\Longrightarrow$ hypergraph) partitioner
   - A sparse matrix ordering tool
2. A parallel library (PT-SCOTCH) which currently only does sparse matrix ordering : based on MPI and optionally *pthread*
   $\rightarrow$ object of my thesis

INRIA FUTURS

Our tool SCOTCH ?

SCOTCH is a graph partitioning, sparse matrix ordering and static mapping tool developed in team SCALAPPLIX
SCOTCH 5.0 software package provides :

1. A sequential library (SCOTCH) which contains :
   - A static mapper
   - A graph and mesh ($\Longrightarrow$ hypergraph) partitioner
   - A sparse matrix ordering tool
2. A parallel library (PT-SCOTCH) which currently only does sparse matrix ordering : based on MPI and optionally *pthread*
   $\rightarrow$ object of my thesis

INRIA FUTURS

Goals of PT-SCOTCH

- Develop fully parallel graph partitioning and reordering tools, by transposing all features of SCOTCH in parallel
- Provide the same quality as the best sequential tools
- Be scalable in time and memory
  $\Longrightarrow$ Efficiency on large architectures
  $\Longrightarrow$ Ability to handle graphs of about a billion vertices distributed on a thousand of processors

INRIA FUTURS

My Work

Goal

Design and implement an efficient parallel sparse matrix ordering tool

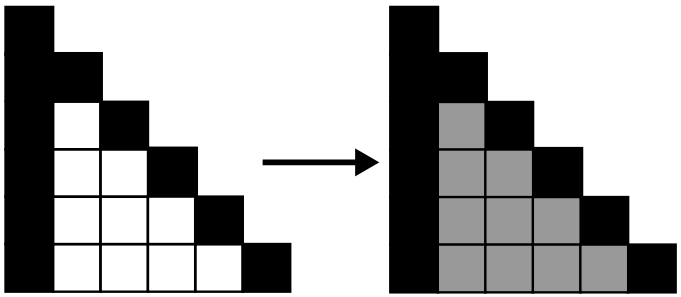
We decided to begin with parallel sparse matrix ordering:

- Uses graph bisection, which is simpler than k-way multisection
- No dependencies between bisections as in the static mapping problem
- The only other available tool, PARMETIS [Karypis and Kumar], provides very poor quality results

INRIA FUTURS

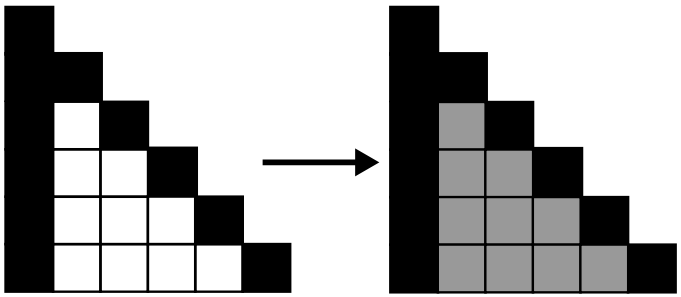
Sparse matrix ordering

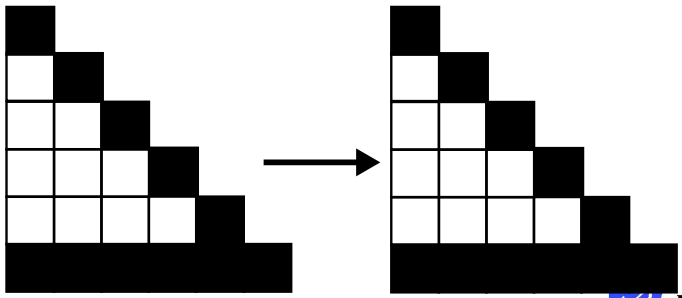
- When solving sparse linear systems with direct methods, non-zero terms are created during the factorization process ($A \longrightarrow LL^t$, $A \longrightarrow LDL^t$ or $A \longrightarrow LU$)
- Fill-in depends on the order of the unknowns
  $\Longrightarrow$ Need to provide fill-reducing orderings
- We do graph ordering in Scotch by means of Nested Dissection using a Multi-Level technique
- Metric of ordering quality: OPC, that is, the Operation Count of Cholesky factorization (overall number of additions, subtractions, multiplications and divisions)
  $\Longrightarrow$ Indirect measurement of separator quality



Sparse matrix ordering

- When solving sparse linear systems with direct methods, non-zero terms are created during the factorization process ($A \longrightarrow LL^t$, $A \longrightarrow LDL^t$ or $A \longrightarrow LU$)
- Fill-in depends on the order of the unknowns
  $\Longrightarrow$ Need to provide fill-reducing orderings
- We do graph ordering in SCOTCH by means of Nested Dissection using a Multi-Level technique
- Metric of ordering quality: OPC, that is, the OPeration Count of Cholesky factorization (overall number of additions, subtractions, multiplications and divisions)
  $\Rightarrow$ Indirect measurement of separator quality

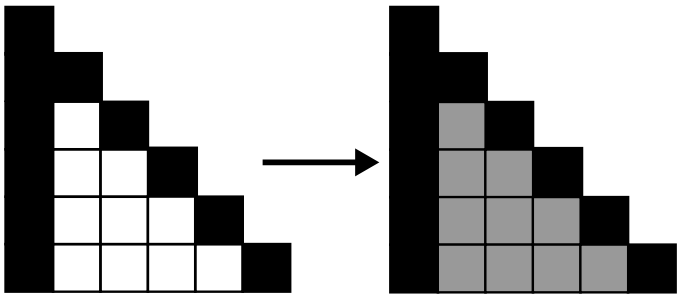


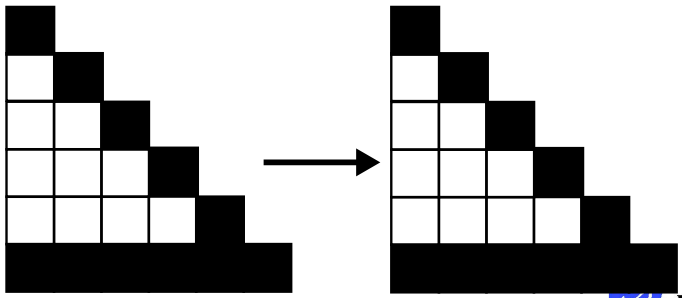


INRIA FUTURS

Sparse matrix ordering

- When solving sparse linear systems with direct methods, non-zero terms are created during the factorization process ($A \longrightarrow LL^t$, $A \longrightarrow LDL^t$ or $A \longrightarrow LU$)
- Fill-in depends on the order of the unknowns
  $\Longrightarrow$ Need to provide fill-reducing orderings
- We do graph ordering in SCOTCH by means of Nested Dissection using a Multi-Level technique
- Metric of ordering quality: OPC, that is, the OPeration Count of Cholesky factorization (overall number of additions, subtractions, multiplications and divisions)
  $\Longrightarrow$ Indirect measurement of separator quality





INRIA FUTURS

Sparse matrix ordering

- When solving sparse linear systems with direct methods, non-zero terms are created during the factorization process ($A \longrightarrow LL^t$, $A \longrightarrow LDL^t$ or $A \longrightarrow LU$)
- Fill-in depends on the order of the unknowns
  $\Longrightarrow$ Need to provide fill-reducing orderings
- We do graph ordering in SCOTCH by means of Nested Dissection using a Multi-Level technique
- Metric of ordering quality: OPC, that is, the OPeration Count of Cholesky factorization (overall number of additions, subtractions, multiplications and divisions)
  $\Rightarrow$ Indirect measurement of separator quality

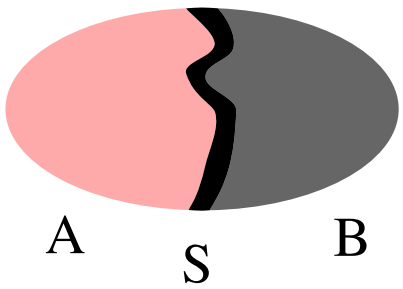
INRIA FUTURS

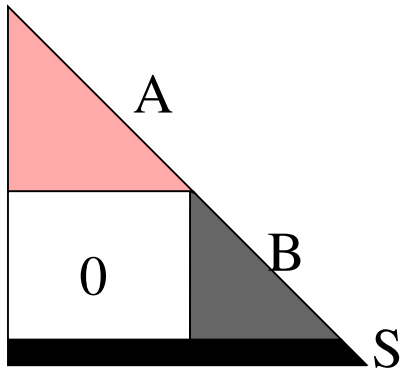
Outline

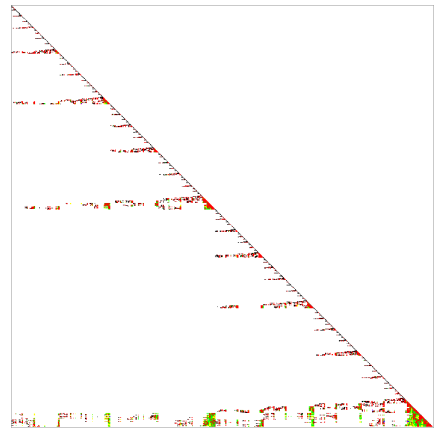
INRIA FUTURS

Sparse matrix ordering by Nested Dissection

- Principle (George 1973)
  - Find a vertex separator in the adjacency graph
  - Number separator vertices with the highest available indices
  - Apply recursively to both separated subgraphs







- Interests
  - Induces high quality block decompositions
  - Increases the concurrency of computations
    ↪ Very suitable for parallel factorization (PaStiX solver)

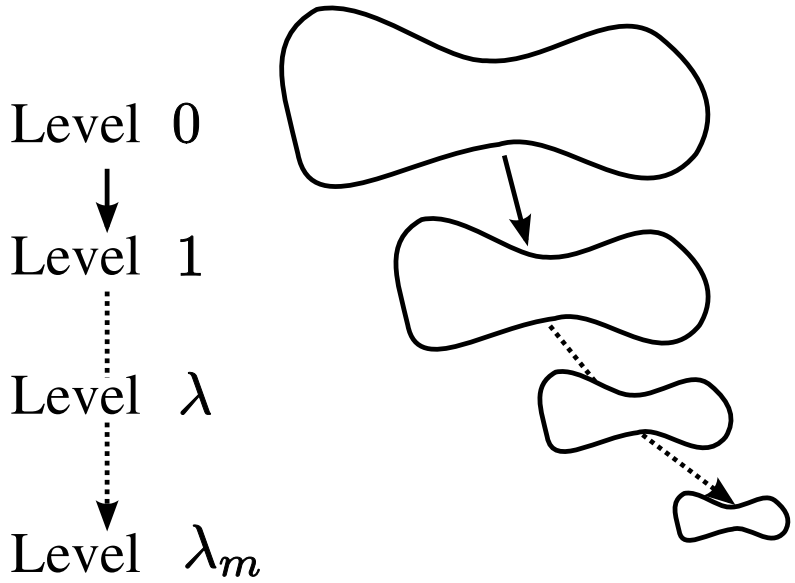
The Multi-Level framework

A classical approach to improve partition quality [Bui and Jones, 1993]

Three steps

1. Coarsening phase
2. Initial partitioning
3. Uncoarsening phase

Decrease the number of vertices by merging pairs of neighbor vertices
$\Longrightarrow$ At every step, obtain a smaller graph with the same topology



The Multi-Level framework

A classical approach to improve partition quality [Bui and Jones, 1993]

Three steps

1. Coarsening phase
2. Initial partitioning
3. Uncoarsening phase

Apply a global heuristic to compute a partition of the smallest graph

Typically, the size of coarsest graphs is about 100 vertices. Therefore, good initial partitions can be computed at low cost

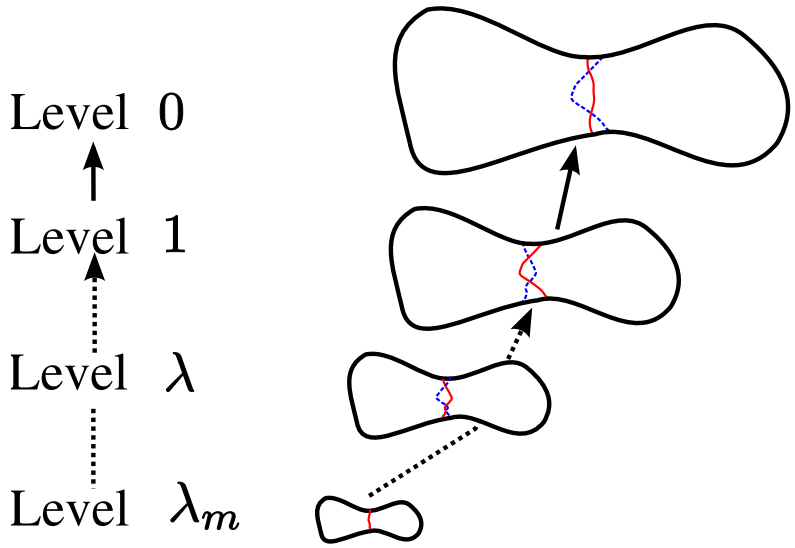
The Multi-Level framework

A classical approach to improve partition quality [Bui and Jones, 1993]

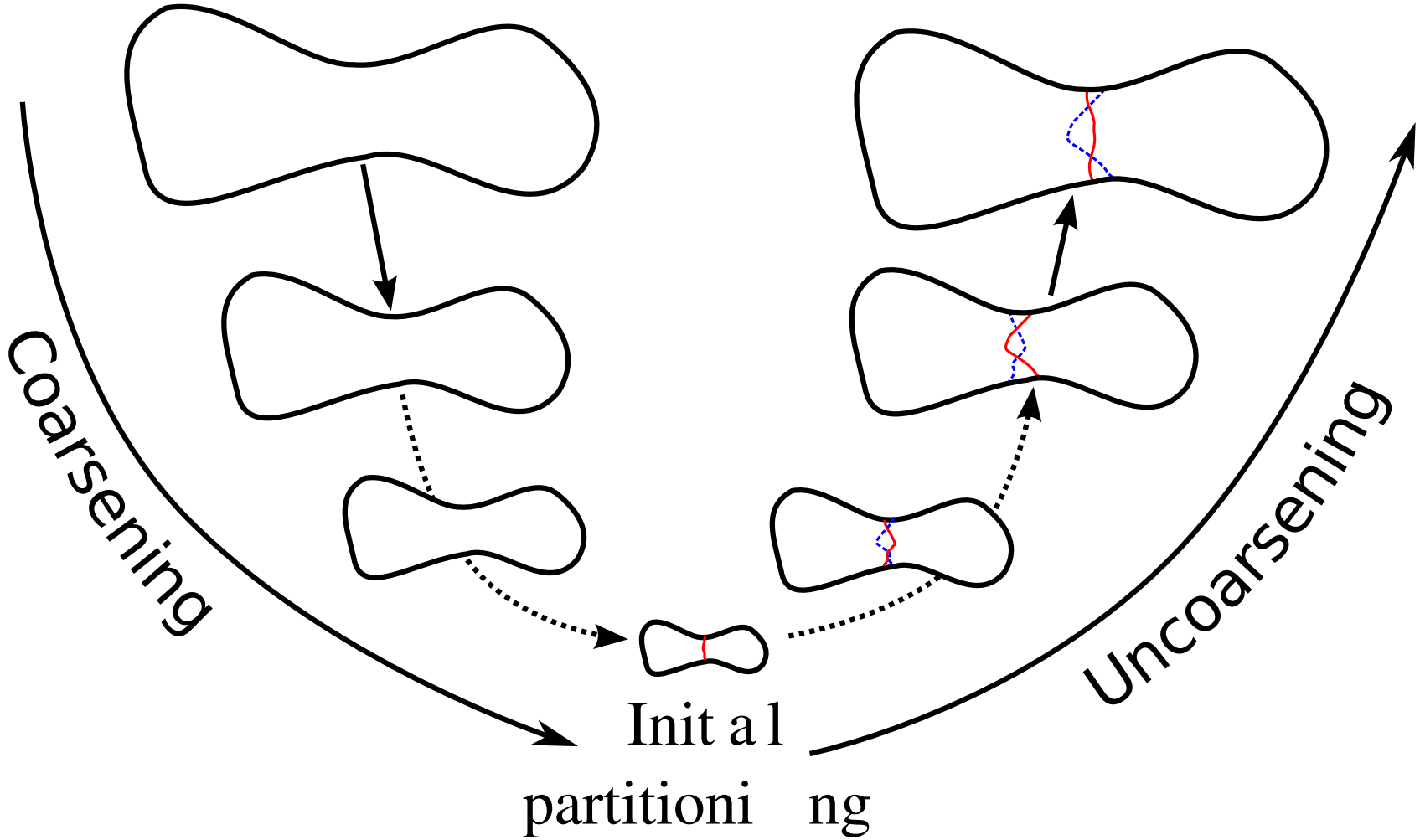
Three steps

1. Coarsening phase
2. Initial partitioning
3. Uncoarsening phase

Project the computed partition from the coarsest graph to finer graphs
Locally refine projected partitions with heuristics such as Kernighan-Lin or Fiduccia-Mattheyses (F.M.)

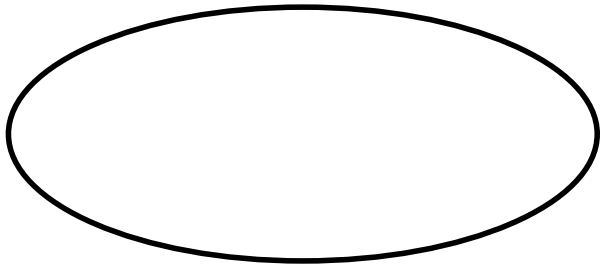


The Multi-Level framework

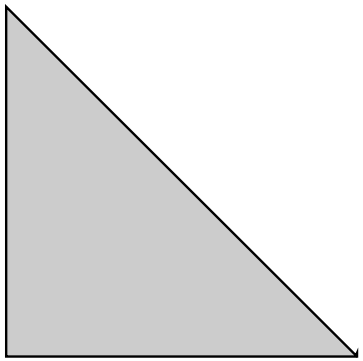


Sequential SCOTCH

First dissection level:
Original graph



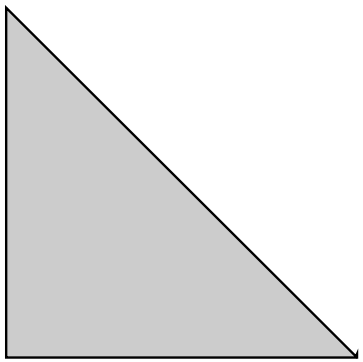
Resulting matrix pattern:

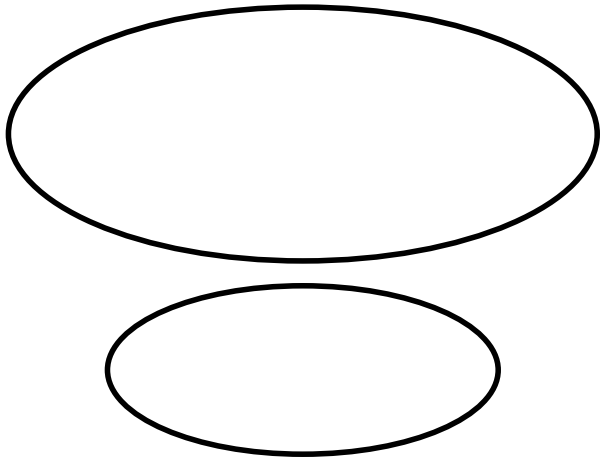


Sequential SCOTCH

First dissection level:
Coarsening phase

Resulting matrix pattern:

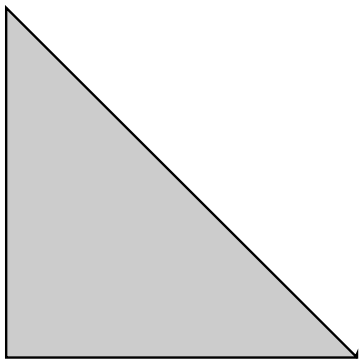


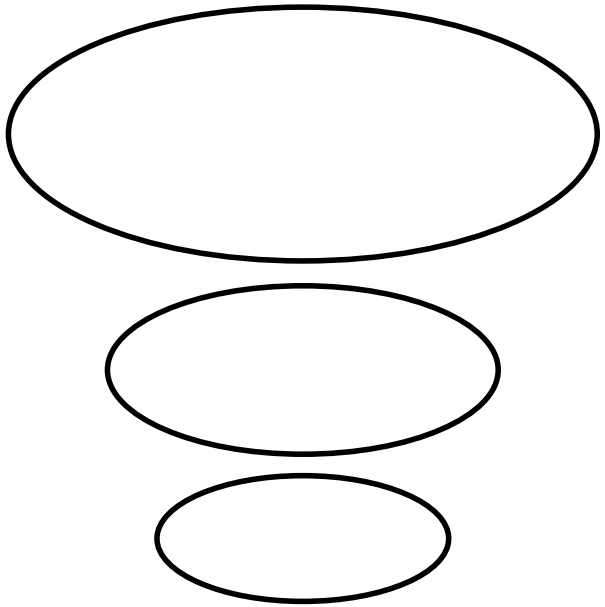


Sequential SCOTCH

First dissection level:
Coarsening phase

Resulting matrix pattern:

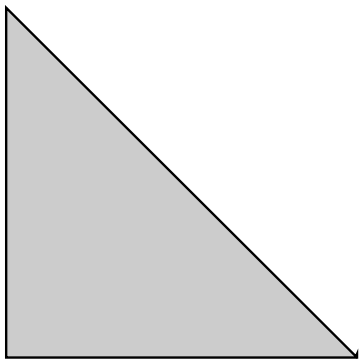


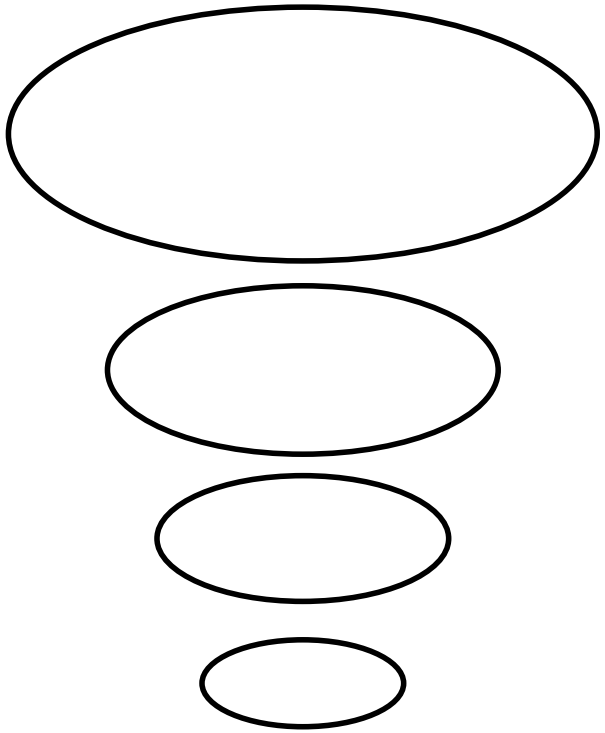


Sequential SCOTCH

First dissection level:
Coarsening phase

Resulting matrix pattern:

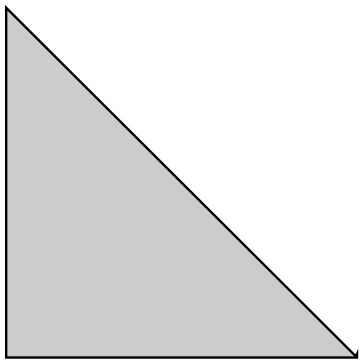


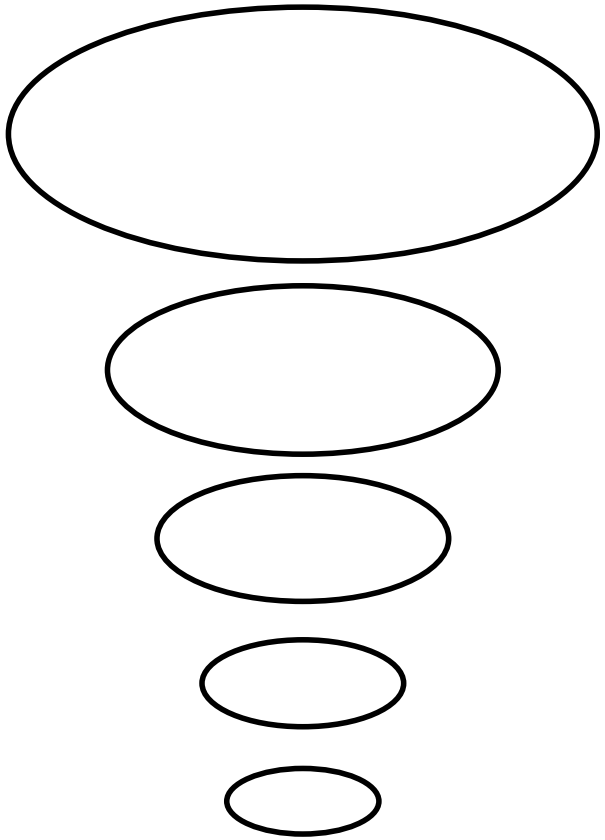


Sequential SCOTCH

First dissection level:
Coarsening phase

Resulting matrix pattern:

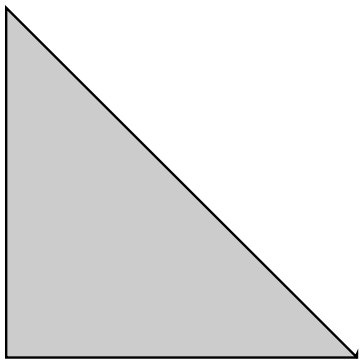


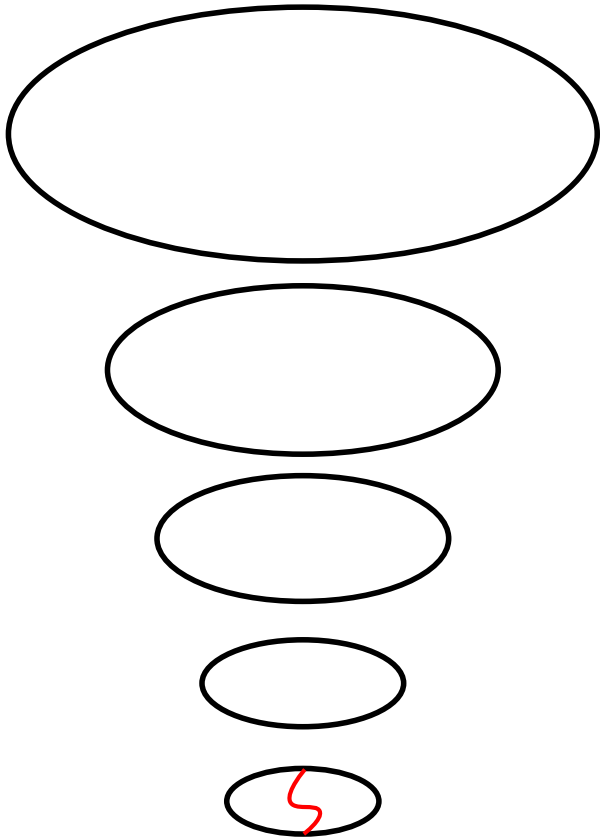


Sequential SCOTCH

First dissection level:
Initial partitioning

Resulting matrix pattern:

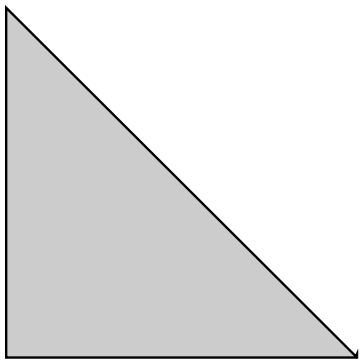


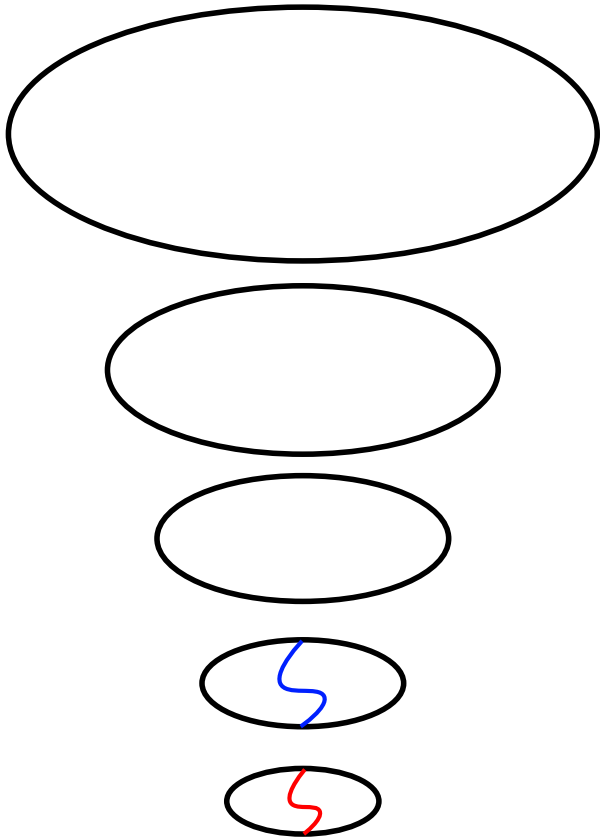


Sequential SCOTCH

First dissection level:
Uncoarsening phase

Resulting matrix pattern:

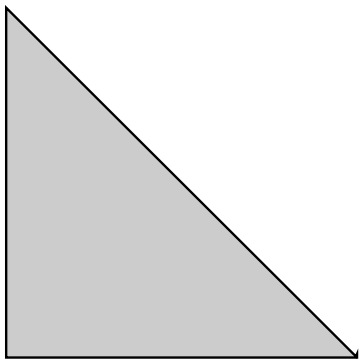


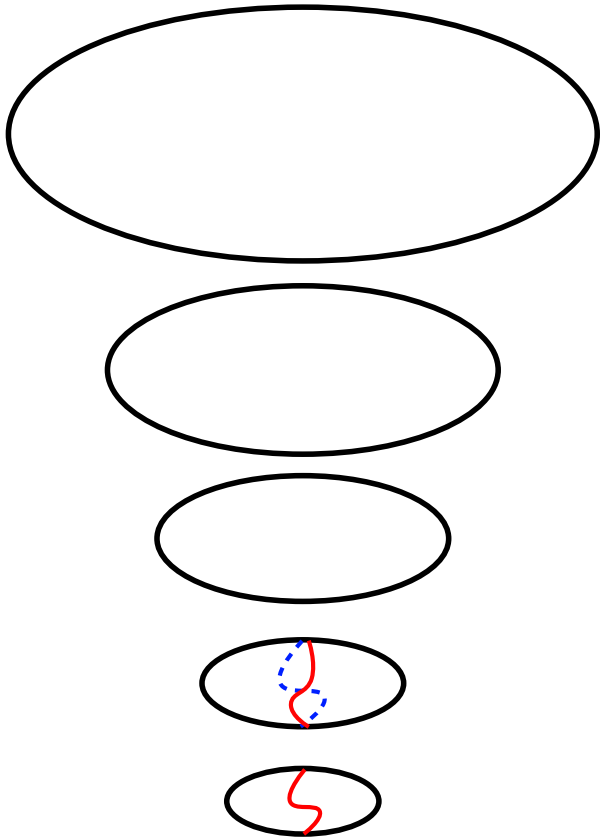


Sequential SCOTCH

First dissection level:
Uncoarsening phase

Resulting matrix pattern:

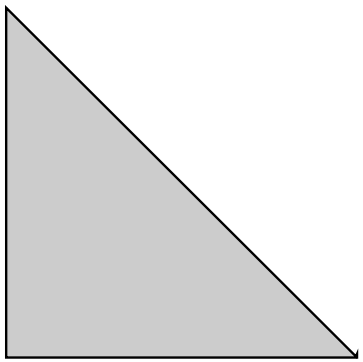


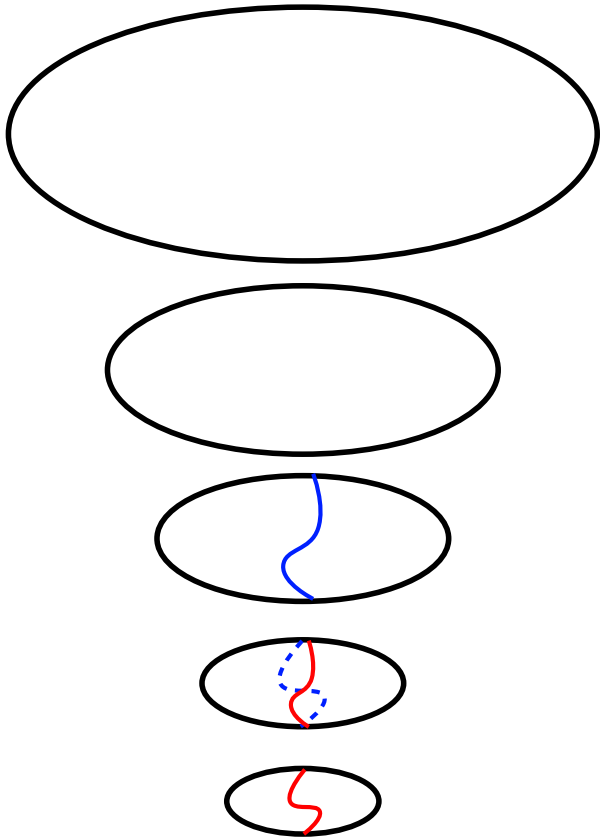


Sequential SCOTCH

First dissection level:
Uncoarsening phase

Resulting matrix pattern:

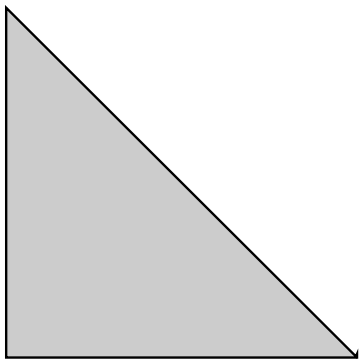


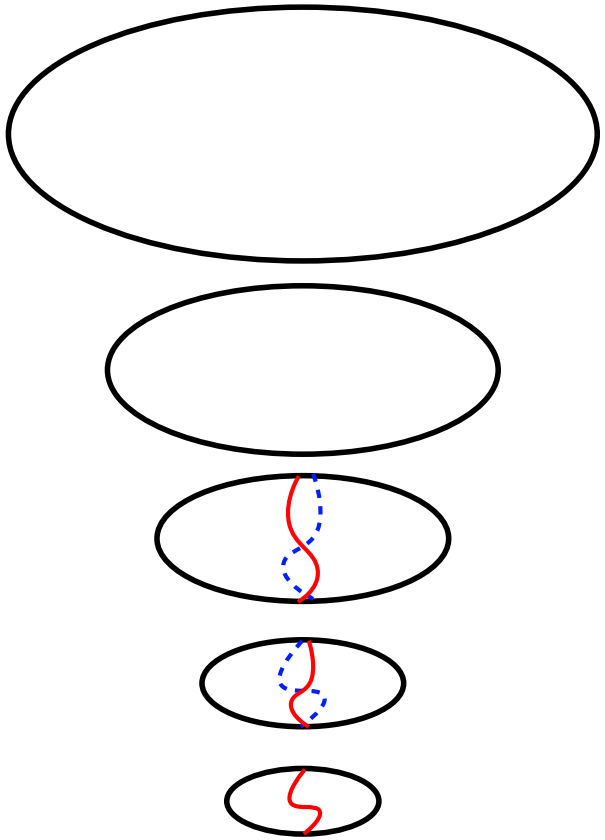


Sequential SCOTCH

First dissection level:
Uncoarsening phase

Resulting matrix pattern:

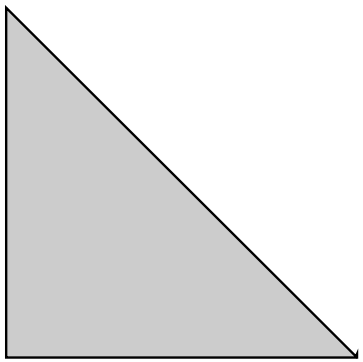


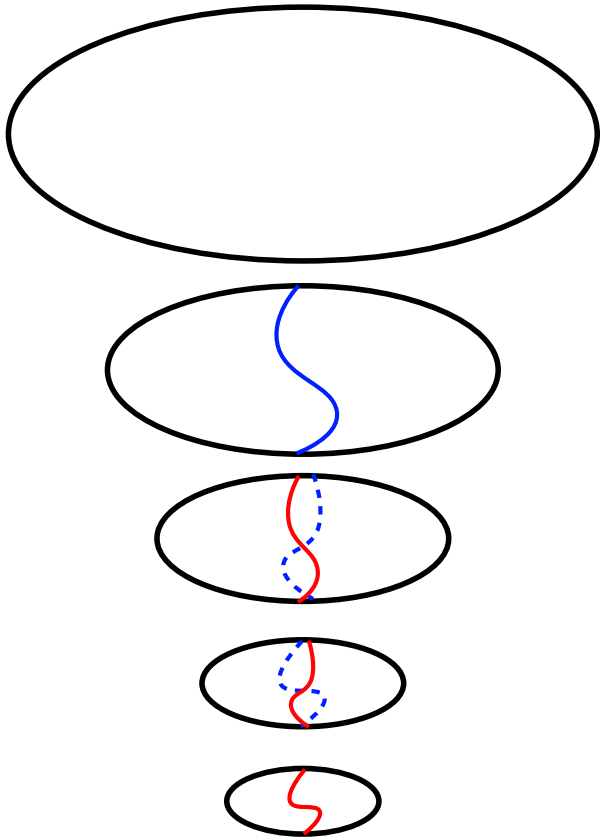


Sequential SCOTCH

First dissection level:
Uncoarsening phase

Resulting matrix pattern:

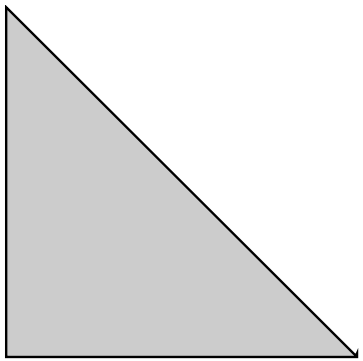


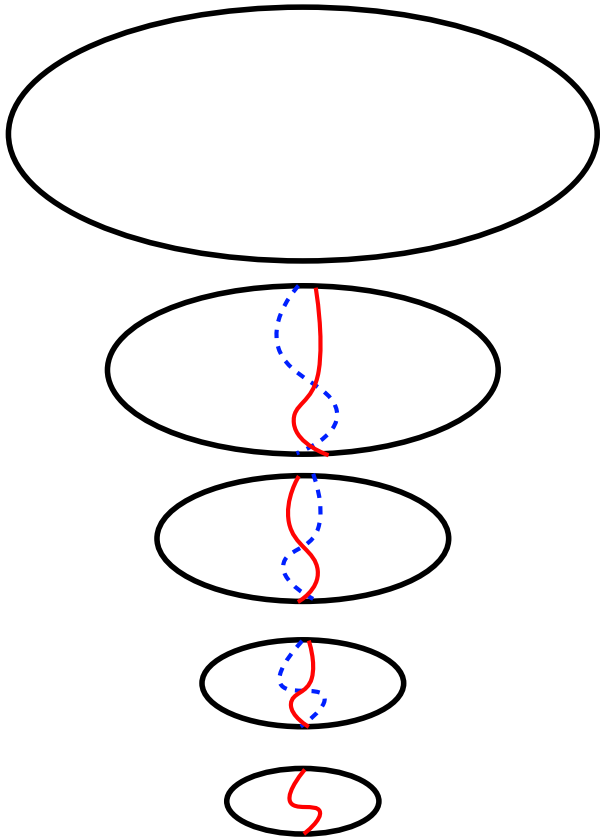


Sequential SCOTCH

Second dissection level:
Uncoarsening phase

Resulting matrix pattern:

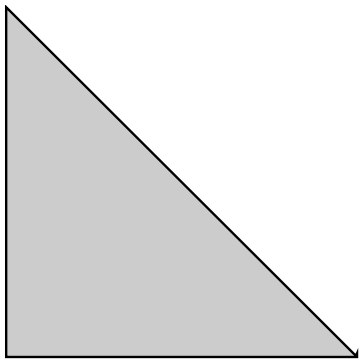


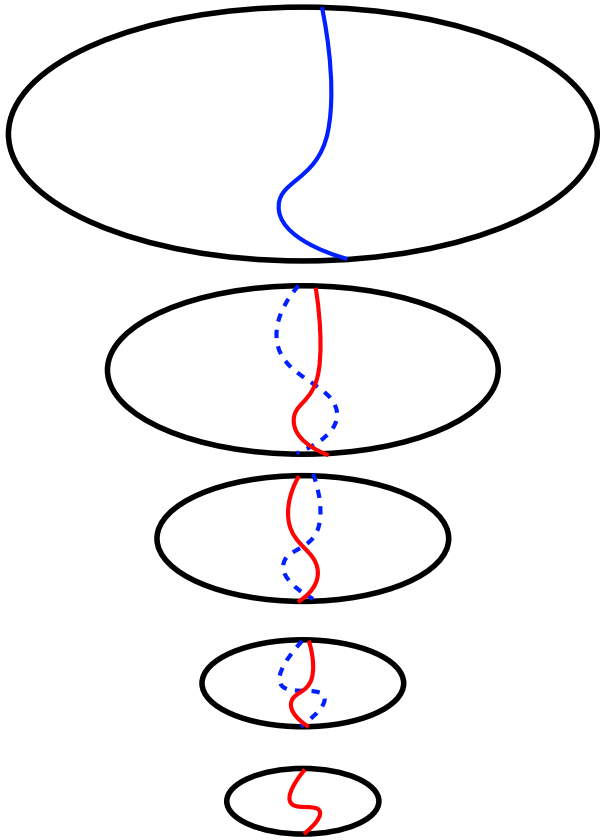


Sequential SCOTCH

Second dissection level:
Uncoarsening phase

Resulting matrix pattern:

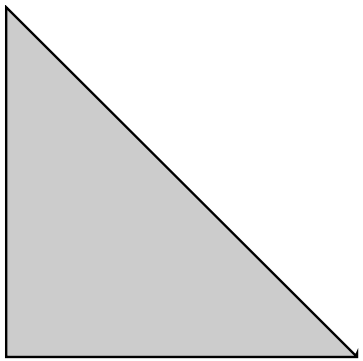


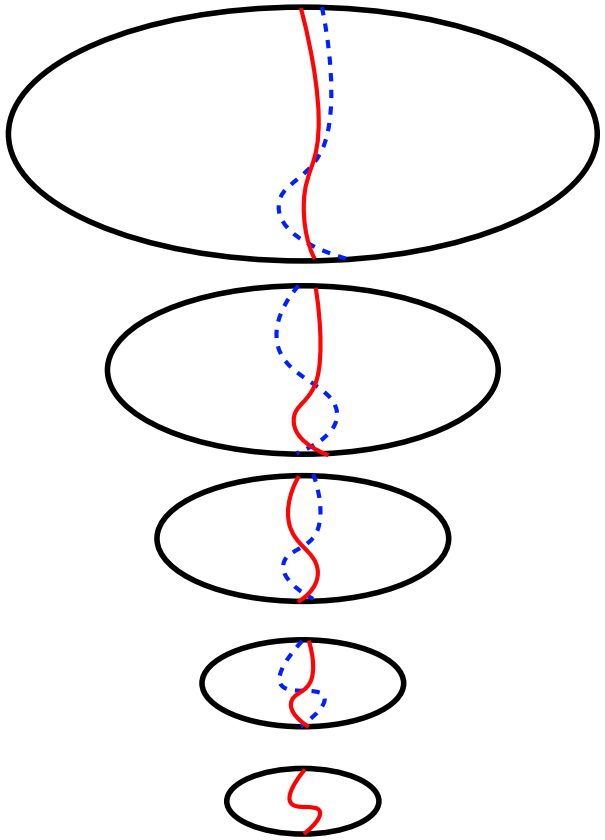


Sequential SCOTCH

Second dissection level:
Uncoarsening phase

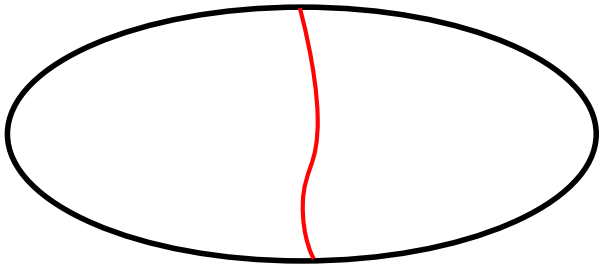
Resulting matrix pattern:



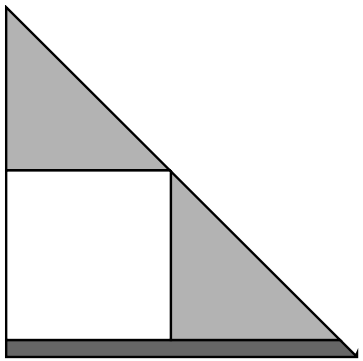


Sequential SCOTCH

Second dissection level:
Bisected graph

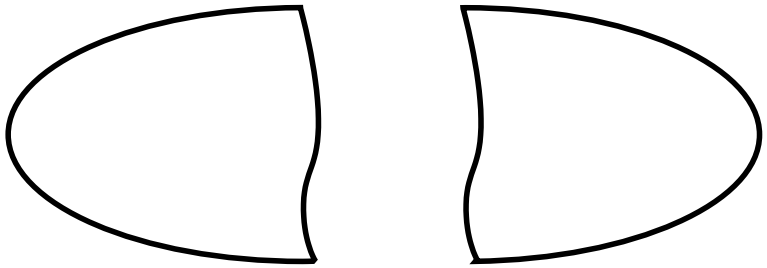


Resulting matrix pattern:

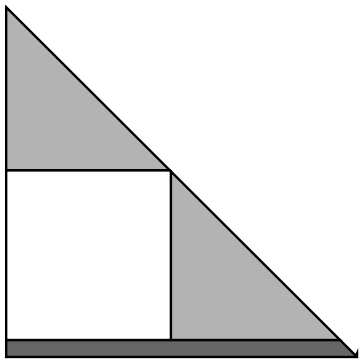


Sequential SCOTCH

Second dissection level:
Two subgraphs



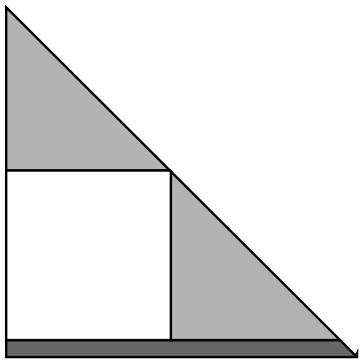
Resulting matrix pattern:

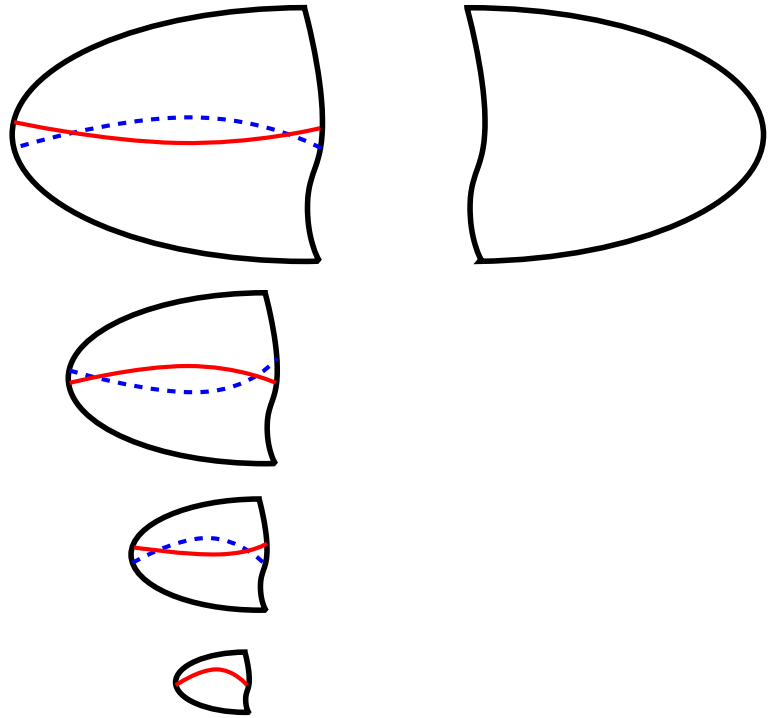


Sequential SCOTCH

Second dissection level:
Processing first subgraph

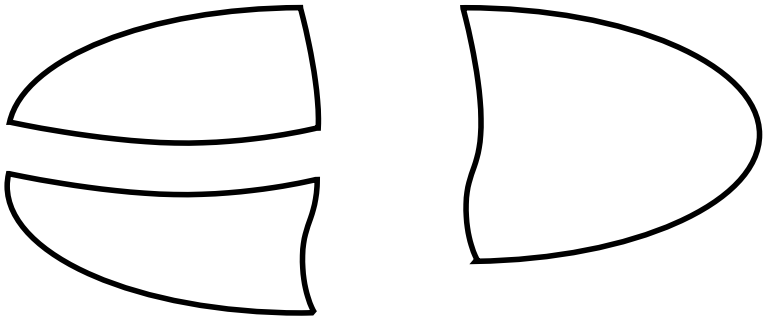
Resulting matrix pattern:



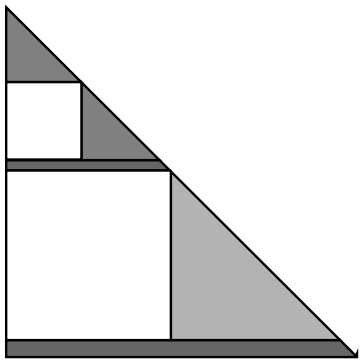


Sequential SCOTCH

Second dissection level:
Processing first subgraph



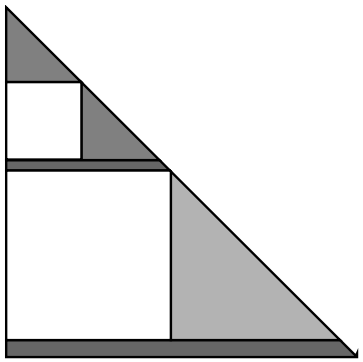
Resulting matrix pattern:

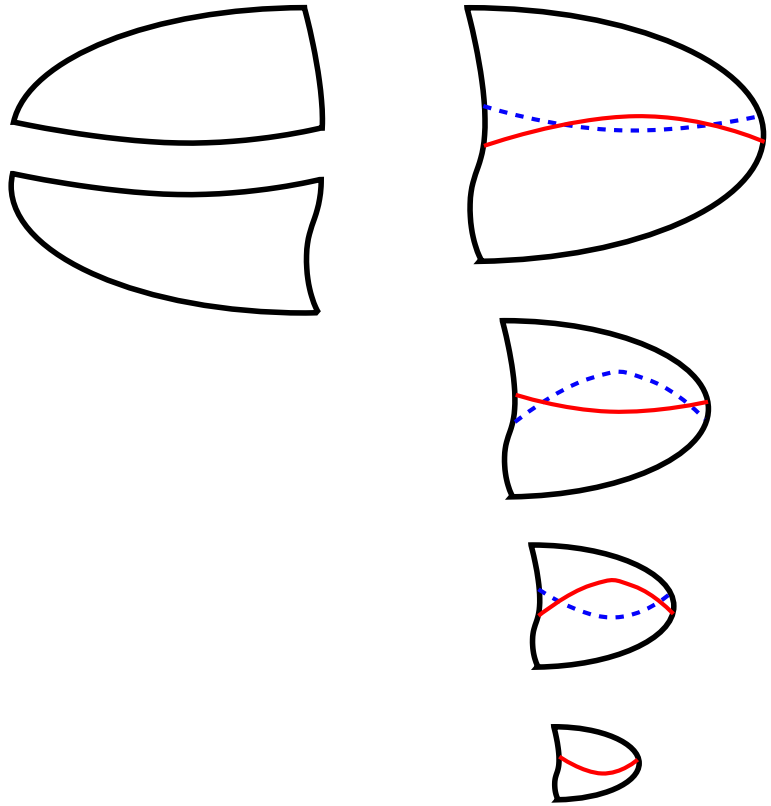


Sequential SCOTCH

Second dissection level:
Processing second subgraph

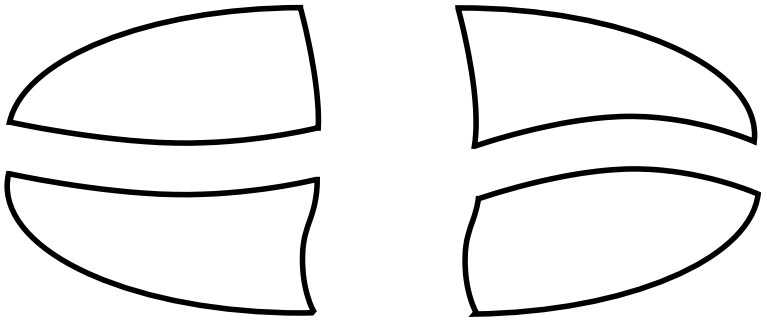
Resulting matrix pattern:



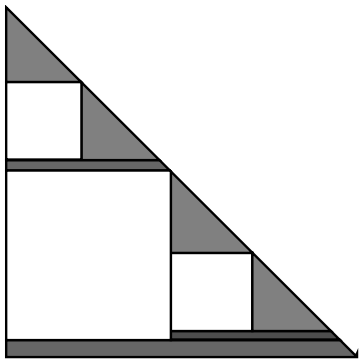


Sequential SCOTCH

Second dissection level:
Processing second subgraph



Resulting matrix pattern:



Parallelism

3 levels of parallelism:

- nested dissection process, all subgraphs of the same level can be computed in parallel
- multi-level, computed on a distributed graph
- refinement on distributed graph

Need a distributed data structure that provides:

- Scalability
- Distribution of vertices
- Distribution of adjacency lists

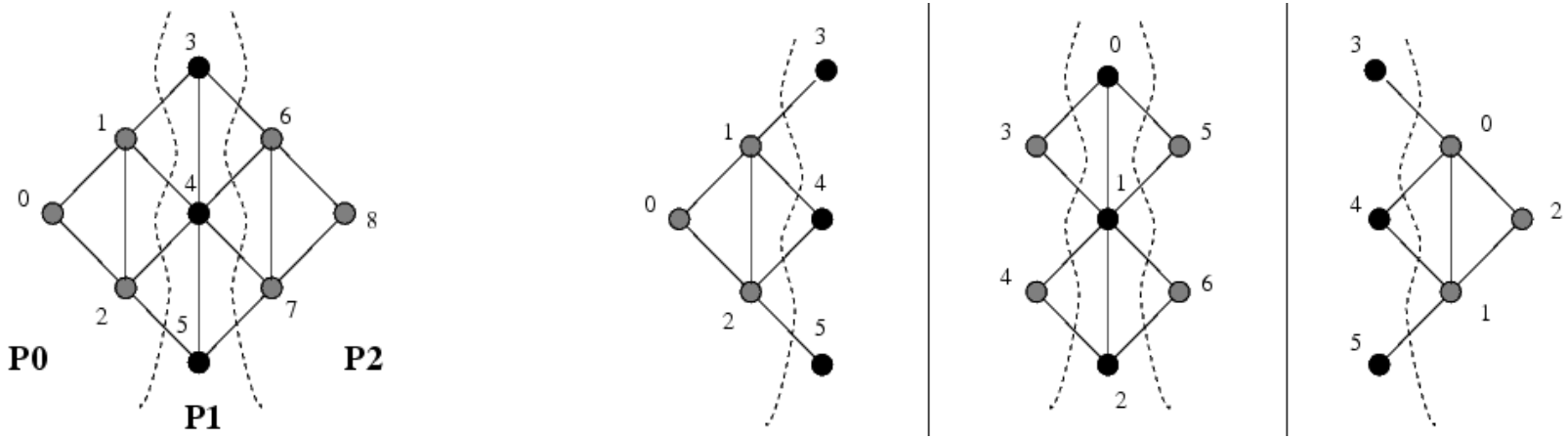
Distributed graph in PT-SCOTCH

Distributed graph

Each process has:

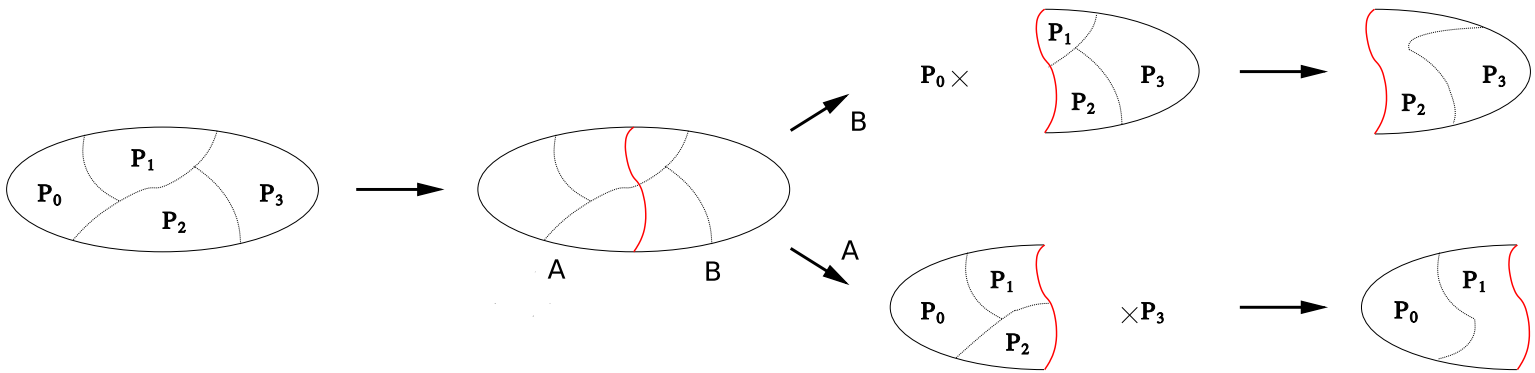
- its own subgraph (local numbering)
- halo: neighbor vertices (“ghosts”), for overlapping communications

No adjacency data for ghost vertices
$\Rightarrow$ at least scalable in number of edges



Parallelization of Nested Dissection

- Straightforward, coarse-grain parallelism
- All subgraphs at the same dissection level are computed concurrently on separate groups of processors
- After a separator is computed, the two separated subgraphs are folded, that is, redistributed, on two subsets of the available processors
  Can fold on any number of processors (not only powers of two)
  $\Longrightarrow$ Better data locality
  $\Longrightarrow$ The two subtrees are separated not only logically but also physically, which helps reducing network congestion



Outline

INRIA FUTURS

Coarsening phase

Principle:

1. Matching phase: make pairing between neighbor vertices
2. Construction phase: merge mated vertices into a new one and update the corresponding edges

We have to parallelize these two steps
Difficulties:

- Matching choices have to be fair to preserve graph topology when coarsen
  $\Rightarrow$ Randomness in mating is important, even when using Heavy Edge Matching
- Coarsened graph has to be well distributed among processes

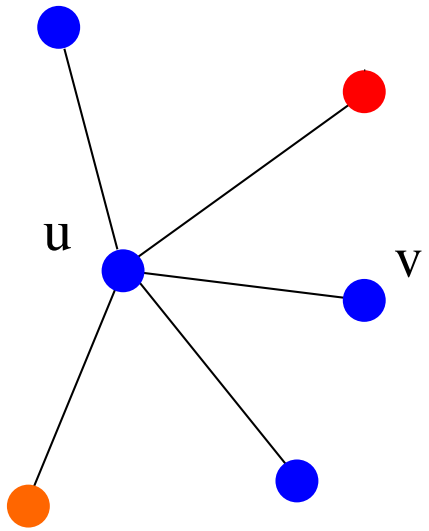
INRIA FUTURS

Matching algorithm

Matching is performed using a parallel multi-phase synchronous algorithm:

Matching algorithm

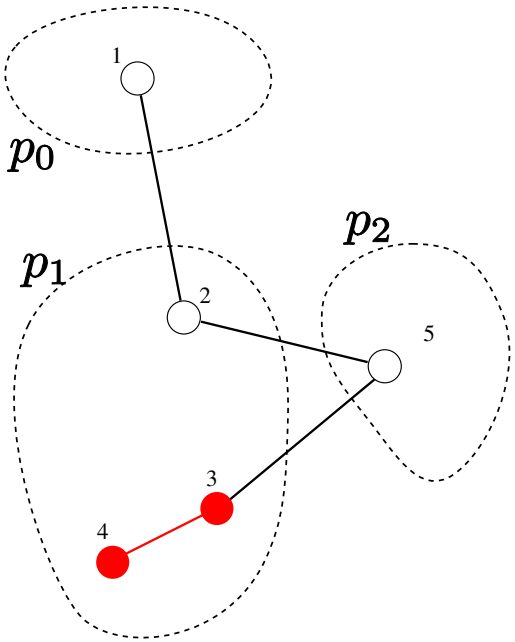
- Step 1: For each unmatched vertex u
  Randomly select an unmatched neighbor vertex v to match with
  If v is local, mark (u, v) as mated
  Else, mark (u, v) as selected
- Step 2: synchronise the selected vertices between processes
- Step 3: While coarsening ratio is not small enough and some vertices can be matched, go to step 1



Synchronisation algorithm is fundamental for keeping a good coarsening quality :

- Acceptance rate of algorithm has to be high
- Accepting “bias” has to be low

Synchronisation algorithm P



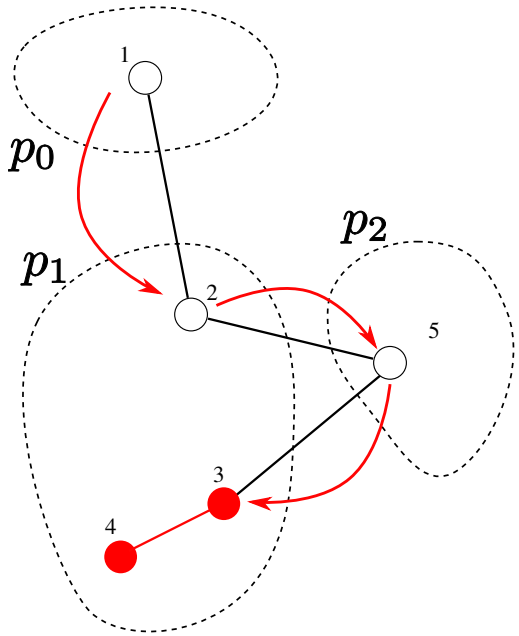
Algorithm

1. Communicate matching wishes to all processes
2. Analyze wishes for each process selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with the requesting vertex

Synchronisation algorithm P



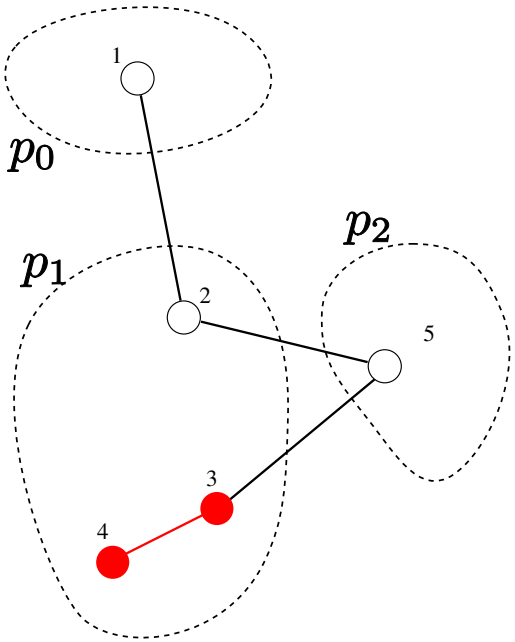
Algorithm

1. Communicate matching wishes to all processes
2. Analyze wishes for each process selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with the requesting vertex

Synchronisation algorithm P



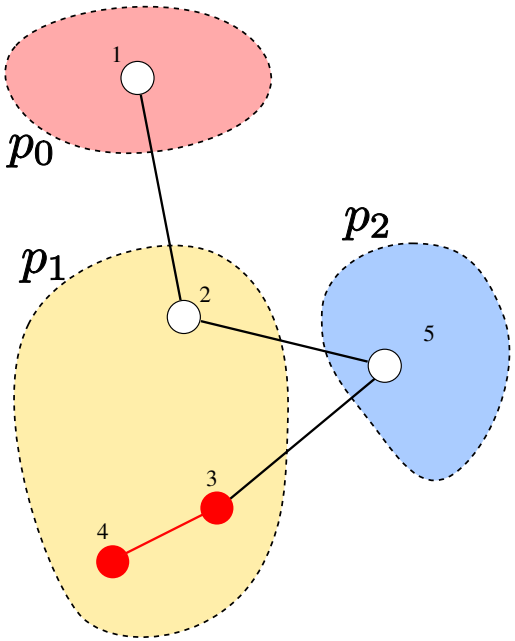
Algorithm

1. Communicate matching wishes to all processes
2. Analyze wishes for each process selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with the requesting vertex

Synchronisation algorithm L



Need a coloring of neighborhood processes graph: we use Luby coloring

Algorithm

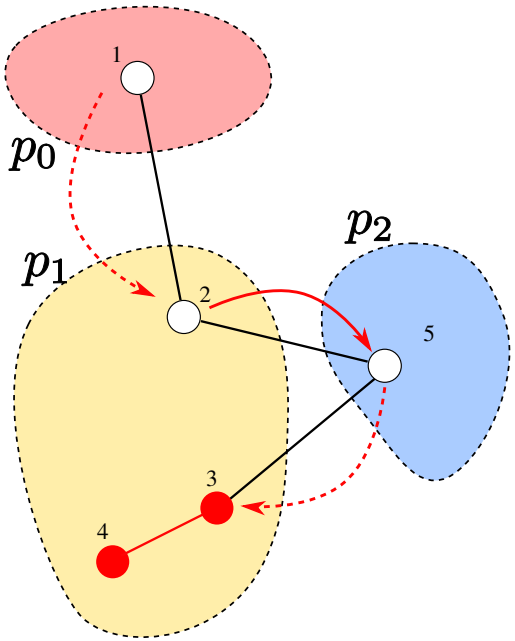
For each color c of processes

- If local process is colored by c
  - Send matching wishes to all neighbors
- If local process is not colored by c
  - Analyze wishes for each neighbor process colored by c selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with a vertex owned by the same process as the requesting vertex

Synchronisation algorithm L



Need a coloring of neighborhood processes graph: we use Luby coloring

Algorithm

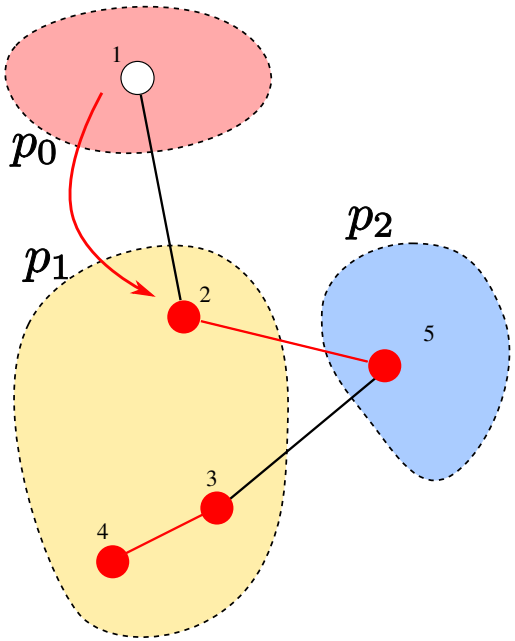
For each color c of processes

- If local process is colored by c
  - Send matching wishes to all neighbors
- If local process is not colored by c
  - Analyze wishes for each neighbor process colored by c selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with a vertex owned by the same process as the requesting vertex

Synchronisation algorithm L



Need a coloring of neighborhood processes graph: we use Luby coloring

Algorithm

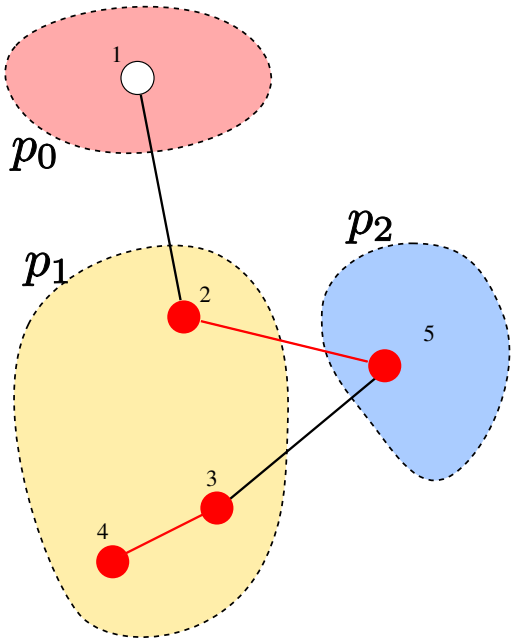
For each color c of processes

- If local process is colored by c
  - Send matching wishes to all neighbors
- If local process is not colored by c
  - Analyze wishes for each neighbor process colored by c selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with a vertex owned by the same process as the requesting vertex

Synchronisation algorithm L



Need a coloring of neighborhood processes graph: we use Luby coloring

Algorithm

For each color c of processes

- If local process is colored by c
  - Send matching wishes to all neighbors
- If local process is not colored by c
  - Analyze wishes for each neighbor process colored by c selected randomly

Authorized pairing

- The wanted vertex is free
- The wanted vertex wishes to be mated with a vertex owned by the same process as the requesting vertex

Comparison between synchronisation algorithms

Algorithm	8 processes ρ	t	OPC	
Originally numbered `audikw1` graph				
P	0.55	81.22	7.06e+12	
L	0.51	55.48	5.77e+12	

Conclusions:

- Contraction ratio higher with L
- With algorithm L, quality of matching is insensitive to the initial graph distribution

But:

- Very low acceptance rate with algorithm P when graph is badly distributed
- One iteration of algorithm P is potentially less expensive than one of L
- Algorithm L is serialized when process graph is complete

Comparison between synchronisation algorithms

Algorithm	8 processes			16 processes		
	ρ	t	OPC	ρ	t	OPC
Originally numbered `audikw1` graph						
P	0.55	81.22	7.06e+12	0.56	67.72	6.75e+12
L	0.51	55.48	5.77e+12	0.51	52.29	5.58e+12

Conclusions:

- Contraction ratio higher with L
- With algorithm L, quality of matching is insensitive to the initial graph distribution

But:

- Very low acceptance rate with algorithm P when graph is badly distributed
- One iteration of algorithm P is potentially less expensive than one of L
- Algorithm L is serialized when process graph is complete

Comparison between synchronisation algorithms

Algorithm	8 processes ρ	8 processes t	8 processes OPC	16 processes ρ	16 processes t	16 processes OPC
Originally numbered `audikw1` graph						
P	0.55	81.22	7.06e+12	0.56	67.72	6.75e+12
L	0.51	55.48	5.77e+12	0.51	52.29	5.58e+12
Randomly numbered `audikw1` graph						
P	0.77	97.67	6.11e+12	0.90	74.29	6.71e+12
L	0.53	86.26	5.39e+12	0.53	71.42	5.56e+12

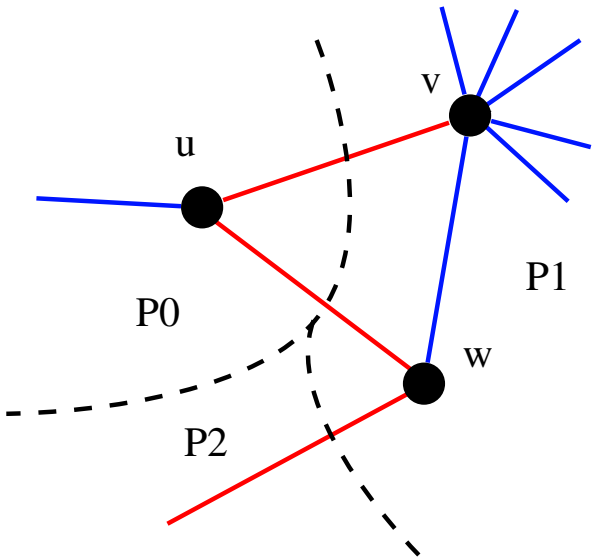
Conclusions:

- Contraction ratio higher with L
- With algorithm L, quality of matching is insensitive to the initial graph distribution

But:

- Very low acceptance rate with algorithm P when graph is badly distributed
- One iteration of algorithm P is potentially less expensive than one of L
- Algorithm L is serialized when process graph is complete

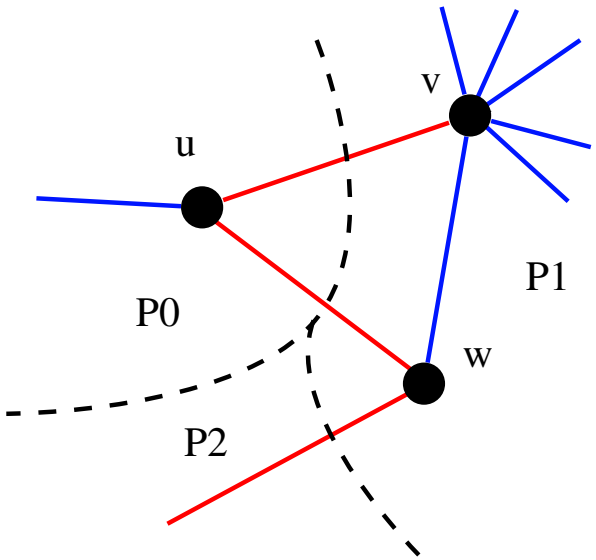
Using probabilities to improve synchronisation



Idea

Favor remote requests which have the highest probability to be accepted

Using probabilities to improve synchronisation



$$P(u) = \frac{1}{3}$$
$$P(v) = \frac{6}{7}$$
$$P(w) = \frac{1}{3}$$

u and w have more chance to be remotely matched

Idea

Favor remote requests which have the highest probability to be accepted

In practice, we treat the vertices with a probability proportional to their local degree
$\Rightarrow$ highly connected boundary vertices are more often free for remote matching

Matching algorithm with selection

New matching algorithm

- Step 1: For each unmatched vertex u
  Ignore u with probability $1 - P(u)$ and go to next vertex
  Randomly select an unmatched neighbor vertex v to match with
  If v is local, mark (u, v) as mated
  Else, mark (u, v) as selected
- Step 2: synchronise the selected vertices between processes
- Step 3: While coarsening ratio is not small enough and some vertices can be matched, go to step 1

Impact of selection

Algorithm	8 processes ρ	8 processes t	8 processes OPC	16 processes ρ	16 processes t	16 processes OPC
Originally numbered `audikw1` graph						
P	0.55	81.22	7.06e+12	0.56	67.72	6.75e+12
SP	0.52	52.89	5.78e+12	0.52	48.34	7.33e+12

- Significantly speeds-up algorithm P
- Improves matching quality by increasing remote matching rate

Impact of selection

Algorithm	8 processes ρ	8 processes t	8 processes OPC	16 processes ρ	16 processes t	16 processes OPC
Originally numbered `audikw1` graph						
P	0.55	81.22	7.06e+12	0.56	67.72	6.75e+12
SP	0.52	52.89	5.78e+12	0.52	48.34	7.33e+12
Randomly numbered `audikw1` graph						
P	0.77	97.67	6.11e+12	0.90	74.29	6.71e+12
SP	0.65	82.18	5.67e+12	0.83	66.79	6.66e+12

- Significantly speeds-up algorithm P
- Improves matching quality by increasing remote matching rate

Coupling SP and L

Algorithm	8 processes ρ	8 processes t	8 processes OPC	16 processes ρ	16 processes t	16 processes OPC
Originally numbered `audikw1` graph						
L	0.51	55.48	5.77e+12	0.51	52.29	5.58e+12
SP+L	0.51	55.32	5.75e+12	0.51	50.29	5.52e+12

- Improves matching quality
- Improves matching speed

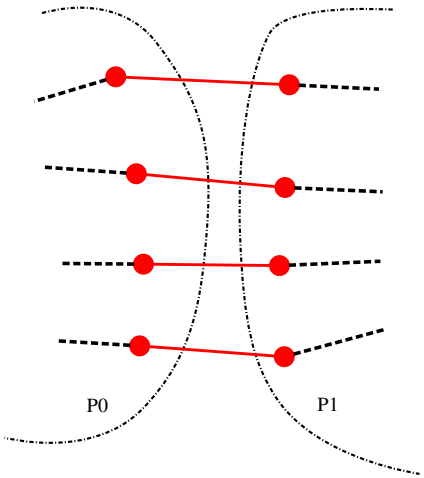
Coupling SP and L

Algorithm	8 processes ρ	8 processes t	8 processes OPC	16 processes ρ	16 processes t	16 processes OPC
Originally numbered `audikw1` graph						
L	0.51	55.48	5.77e+12	0.51	52.29	5.58e+12
SP+L	0.51	55.32	5.75e+12	0.51	50.29	5.52e+12
Randomly numbered `audikw1` graph						
L	0.53	86.26	5.39e+12	0.53	71.42	5.56e+12
SP+L	0.53	78.99	5.49e+12	0.53	71.93	5.51e+12

- Improves matching quality
- Improves matching speed

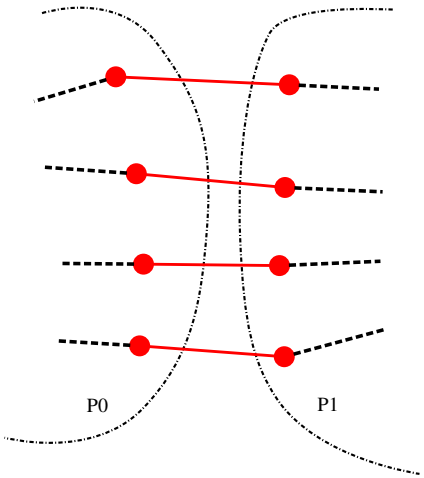
Coarsening summary

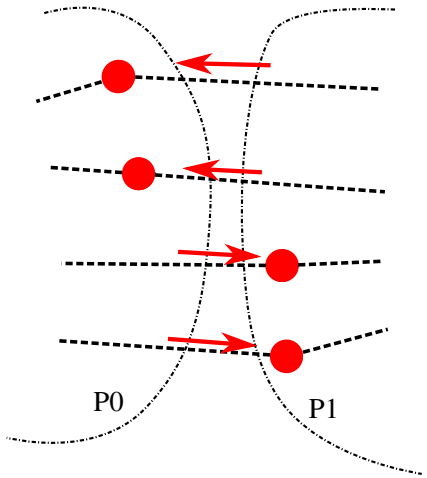
- Randomized tie-breaking at first when many neighbors are available (SP), then deterministic process (L) to complete the matching
- Insensitive, in term of quality, to the initial graph distribution
- Designed to balance coarsened vertices evenly across processors: at the end of the matching phase, all exchanges between each pair of processes are balanced
- Local pairs are numbered before remote ones: computation/communication overlap during coarser graph construction



Coarsening summary

- Randomized tie-breaking at first when many neighbors are available (SP), then deterministic process (L) to complete the matching
- Insensitive, in term of quality, to the initial graph distribution
- Designed to balance coarsened vertices evenly across processors: at the end of the matching phase, all exchanges between each pair of processes are balanced
- Local pairs are numbered before remote ones: computation/communication overlap during coarser graph construction





Coarsening summary

- Randomized tie-breaking at first when many neighbors are available (SP), then deterministic process (L) to complete the matching
- Insensitive, in term of quality, to the initial graph distribution
- Designed to balance coarsened vertices evenly across processors: at the end of the matching phase, all exchanges between each pair of processes are balanced
- Local pairs are numbered before remote ones: computation/communication overlap during coarser graph construction

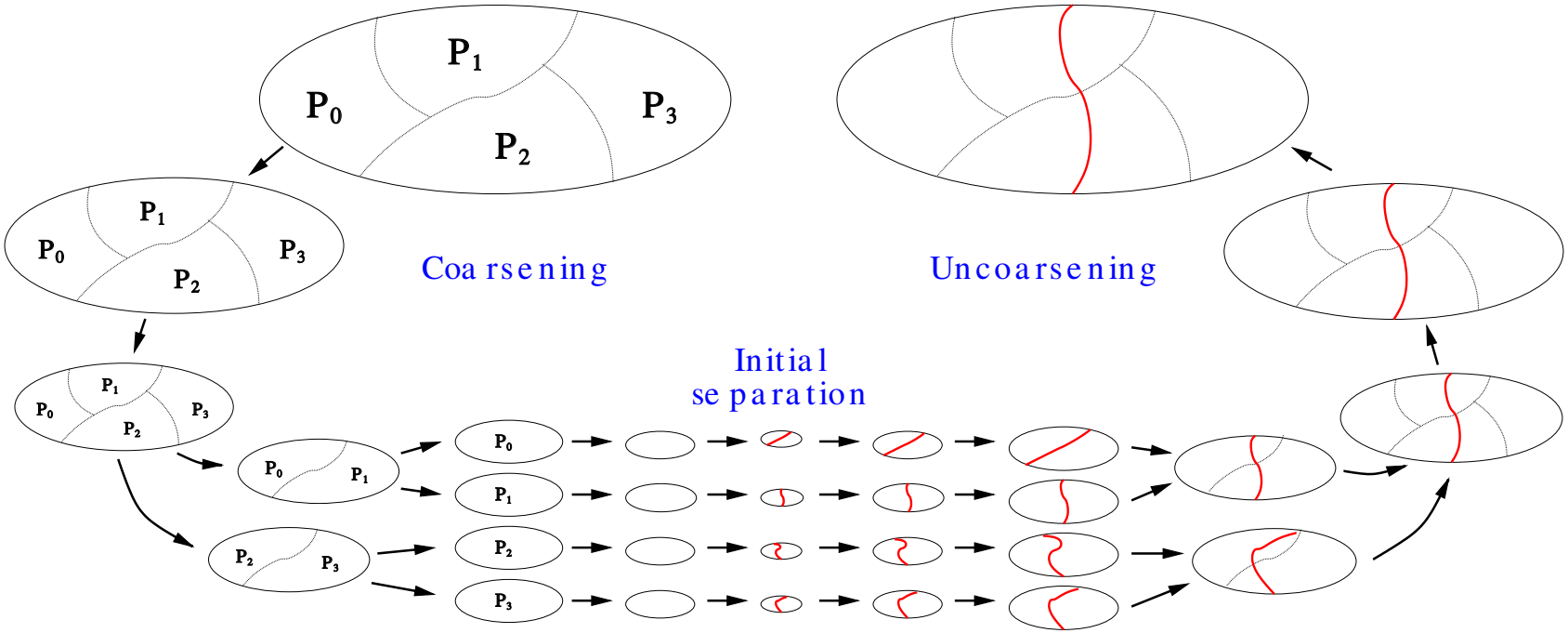
From local vertices From remote

Outline

INRIA FUTURS

The fold-dup feature

Coarsened graphs may be folded and duplicated onto two halves of the processors to reduce communication and increase quality



The fold-dup feature

Experiments show that folding-with-duplication is more efficient when performed on the coarsest graphs:

- Due to memory overcost, it cannot be applied to all levels
  - Every level requires $O(|V|)$ space when always duplicating
  - Without duplication, level k requires only 0 $\left(\frac{|V|}{2^k}\right)$ space
- Improves performances in time by increasing data locality
- Improves quality by allowing multiple computations of initial partitions

Timing results on a 16-CPU/node SMP cluster for `brgm` ($|V| = 3.7e+06$, $|E| = 1.51e+08$):

Strategy	**Number of processors**					
	2	4	8	16	32	64
No F-D	260.31	156.65	102.37	71.19	45.33	†
F-D	276.91	167.26	97.69	61.65	42.85	41.00

Outline

INRIA FUTURS

The uncoarsening phase

Goals

1. Project partition found to coarser graph on the finer graph (*global* aspect)
2. Improve quality of projection by taking account of the finest definition of topology (local optimization)
   $\longrightarrow$ separator displacement

- Projection step is easy to parallelize: only one change in parallel, we must select partition to project back when using folding and duplication
- Refinement algorithms are much harder to parallelize

INRIA FUTURS

Problems with refinement algorithms

- Local optimization algorithms, because of their local nature:
  - Are often trapped in local optima of their cost functions
  - Are intrinsically sequential and therefore do not parallelize well
- Global optimization algorithms are too expensive for large graphs

Need to define a framework within which:

- Sequential local optimization algorithms are still usable on large distributed graphs
- Global algorithms can be run inexpensively

Problems with refinement algorithms

- Local optimization algorithms, because of their local nature:
  - Are often trapped in local optima of their cost functions
  - Are intrinsically sequential and therefore do not parallelize well
- Global optimization algorithms are too expensive for large graphs

Need to define a framework within which:

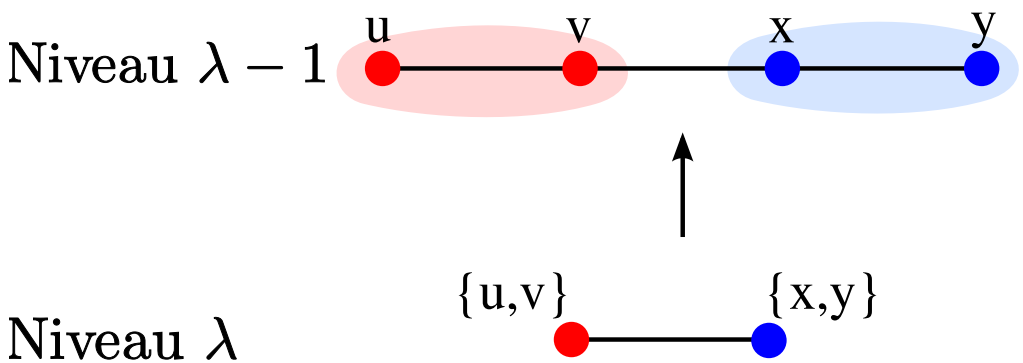
- Sequential local optimization algorithms are still usable on large distributed graphs
- Global algorithms can be run inexpensively

Introducing band graphs

Separator does not move very far during refinement

Explanation

Provided that the coarser graph is topologically close to its finer graph, the projection of the partition computed on the coarser graph is close to the one of the finer, and only further local improvements have to be carried out



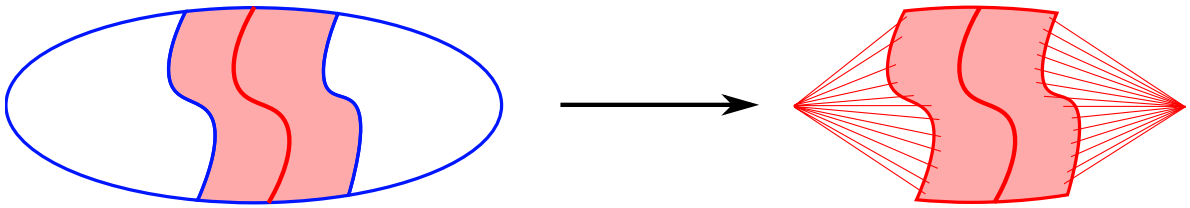
Introducing band graphs

Experimentally evidenced by looking at the distribution of finer separator vertices after local F.M. refinement

Distance from coarser separator				
0	1	2	3	≥ 4
80%	17%	1.5%	$< 0.5\%$	$< 0.1\%$

Idea [Chevalier and Pellegrini, 2006]

Only consider for refinement vertices that are likely to move
$\Longrightarrow$ Extract a priori a band subgraph around the coarser separator and apply optimization heuristics only to it



$\Longrightarrow$ Dramatic reduction in problem size:
for a 10^9 vertices 3D graph, separator is about 10^6 vertices

Using band graphs

Extracting band graphs allows us:

- To go on using classical sequential refinement algorithms, on smaller graphs
  $\Longrightarrow$ Guarantees no loss of quality over existing sequential software
- To use parallel algorithms which are much too expensive to be run on large graphs
- To use new approaches which exploit the fact that the band constrains the refinement process

Banded Fiduccia-Mattheyses

- Very little overcost in time: $\sim$ 3% of increase in time compared to plain F.M.
- While the default bipartitioning strategy of SCOTCH 4.0 uses two runs of multi-level bisection and keeps the best, we can achieve equivalent quality with only one run of multi-level banded F.M.
  $\Longrightarrow$ Overall gain in execution time
  $\Longrightarrow$ No loss of quality but, on the contrary, an improvement

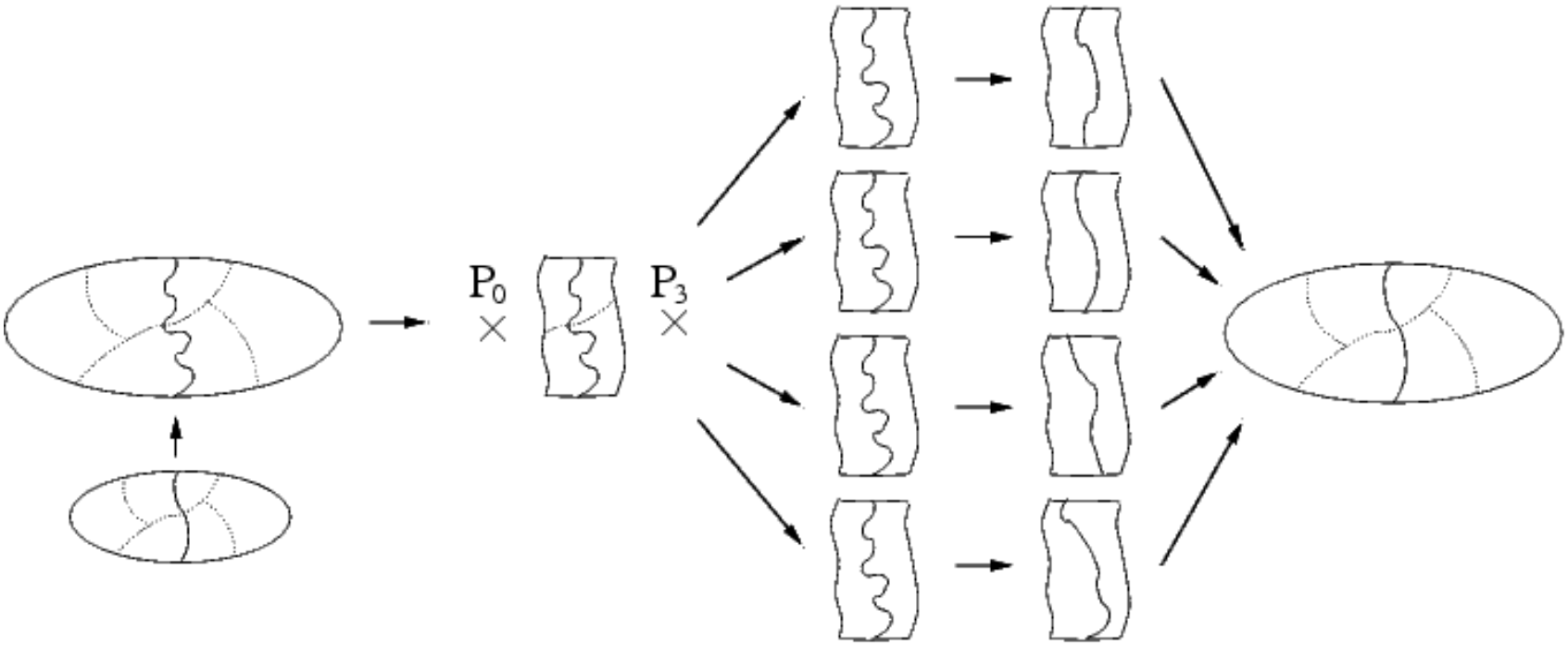
Explanation

Better accounting for the global topology of graphs by limiting the ability to be trapped in purely local optima whom may be coarsening artefacts

INRIA FUTURS

Multi-centralized banded Fiduccia-Mattheyses

1. Extract band graph in parallel
2. Each process owns his centralized band graph
3. Each process runs F.M. on his centralized band graph
4. The best partition is broadcasted to every process and kept to go to next level



Multi-centralized banded Fiduccia-Mattheyses (2)

Consequences:

- PT-Scotch exploits processors in a multi-sequential way for refinement
- Quality of refinement increases when number of processes increases
- Definitely not scalable but useful before scalable optimization algorithms are available

Genetic Algorithms (G.A.) [Holland, 1970s]

- Meta-heuristics which can be applied to solve various optimization problems
- Easy to parallelize
- Known to be very expensive in time, as problem size increases
- An ideal candidate to operate on band graphs

Genetic Algorithms (G.A.) [Holland, 1970s]

- Meta-heuristics which can be applied to solve various optimization problems
- Easy to parallelize
- Known to be very expensive in time, as problem size increases
- An ideal candidate to operate on band graphs

INRIA FUTURS

G.A. for graph partitioning

To use G.A.s, we must be able to compare the quality of two partitions
Two criteria:

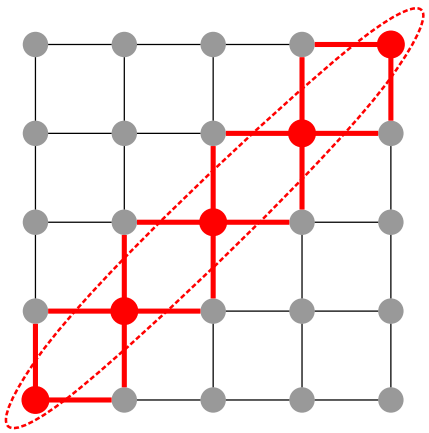
- Separator size (number of separator vertices)
- Separator shape (number of edges adjacent to separator vertices)

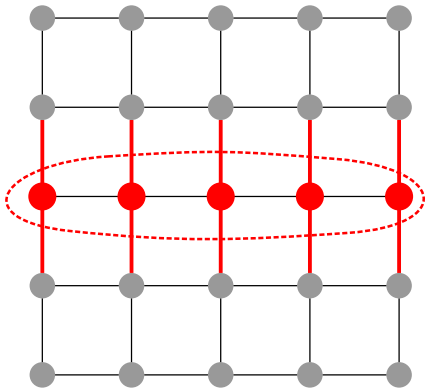
INRIA FUTURS

G.A. for graph partitioning

To use G.A.s, we must be able to compare the quality of two partitions
Two criteria:

- Separator size (number of separator vertices)
- Separator shape (number of edges adjacent to separator vertices)

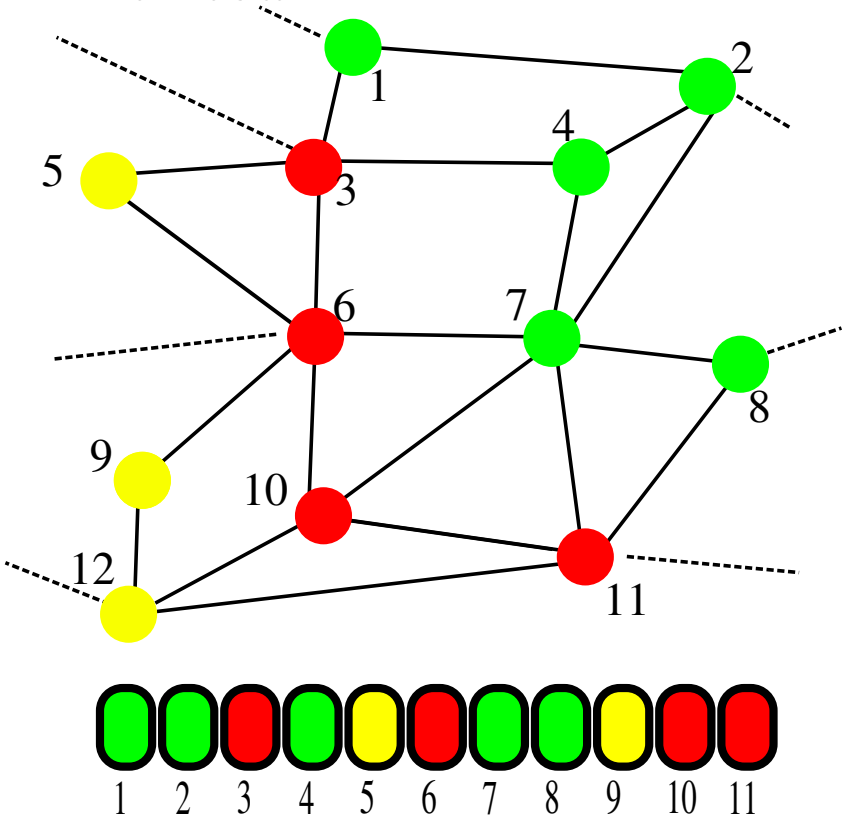




Our G.A. implementation

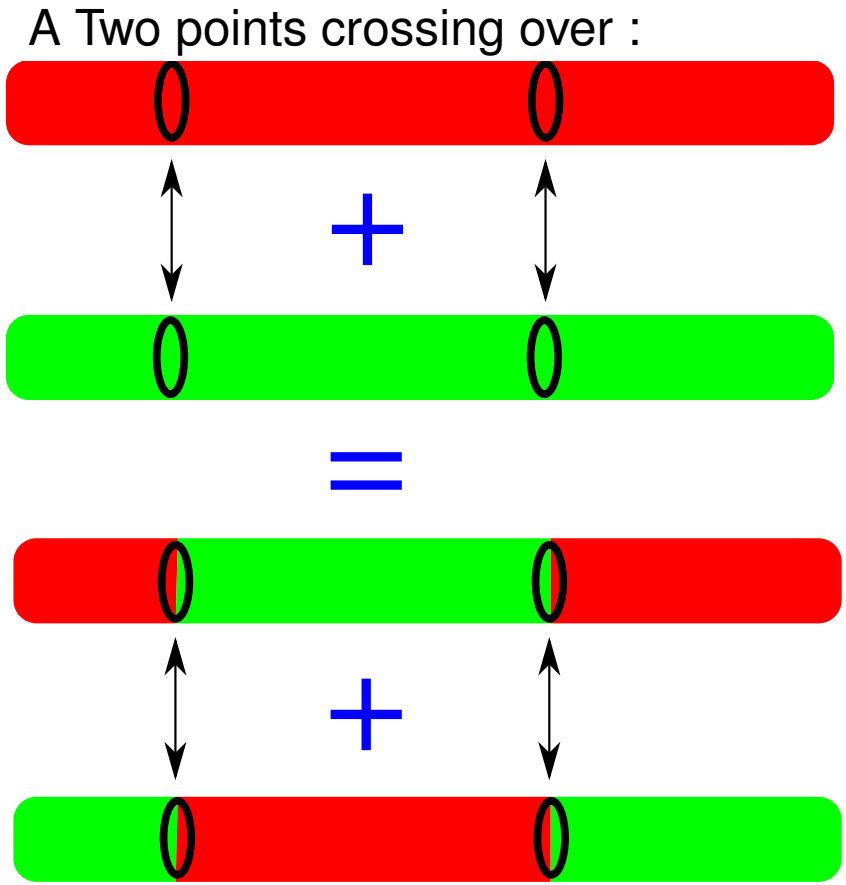
- Mono-chromosome individuals : an uni-dimensional array which associates a number between 0 and 2 to any subgraph vertex index
- Reproduction operator is a classical two-points cross-over operator
- Selection operator is a mix between roulette and elitism

An individual :



Our G.A. implementation

- Mono-chromosome individuals : an uni-dimensional array which associates a number between 0 and 2 to any subgraph vertex index
- Reproduction operator is a classical two-points cross-over operator
- Selection operator is a mix between roulette and elitism



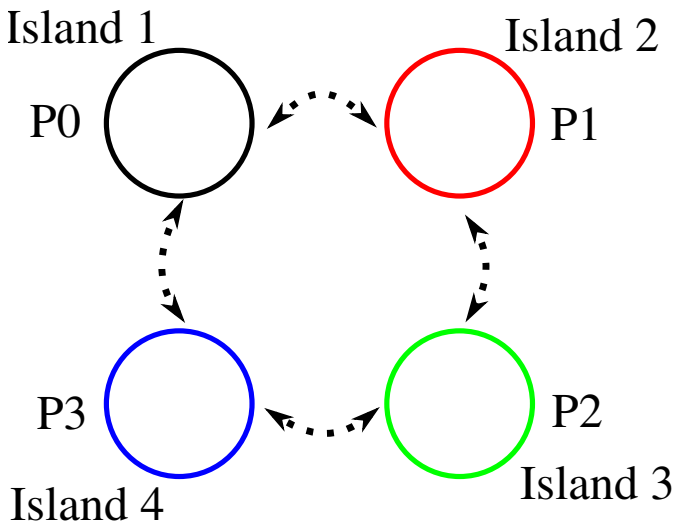
Our G.A. implementation

- Mono-chromosome individuals : an uni-dimensional array which associates a number between 0 and 2 to any subgraph vertex index
- Reproduction operator is a classical two-points cross-over operator
- Selection operator is a mix between roulette and elitism

- Elitism : Best individuals are kept in the next generation
- Roulette : At each turn, two individuals are selected to be parents of the next generation. The probability to be chosen is proportional to the quality of the partition

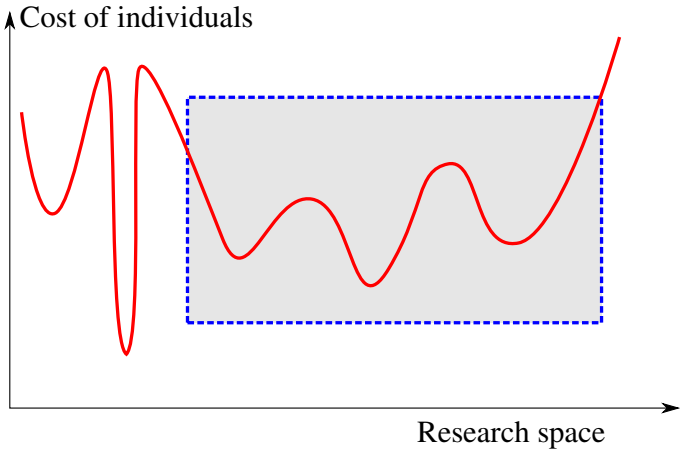
Genetic Algorithms in parallel

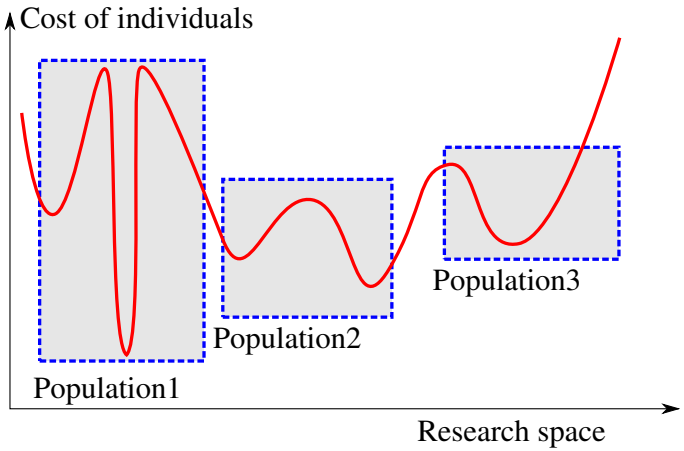
- Multi-deme model (Goldberg, 1997) :
  - Several isolated populations (islands or demes) :
    one population per processor in our case
  - Occasionally, a few *champions* are allowed to migrate between demes :
    Very few inter-processor communications, thus model is easy to parallelize
- Increases the opportunity to converge to the global minimum of the partition cost function



Genetic Algorithms in parallel

- Multi-deme model (Goldberg, 1997) :
  - Several isolated populations (islands or demes) :
    one population per processor in our case
  - Occasionally, a few *champions* are allowed to migrate between demes :
    Very few inter-processor communications, thus model is easy to parallelize
- Increases the opportunity to converge to the global minimum of the partition cost function





Banded Genetic Algorithms

Tests carried out with a threaded version of SCOTCH and a multi-deme algorithm without splitting chromosomes on several processors:

- Achieve good quality
- Are more expensive than banded F.M. but more parallelizable

$\Rightarrow$ We can extend this multi-sequential method in parallel without increasing too much the volume of communications, simply by splitting individual chromosomes across several processors

Outline

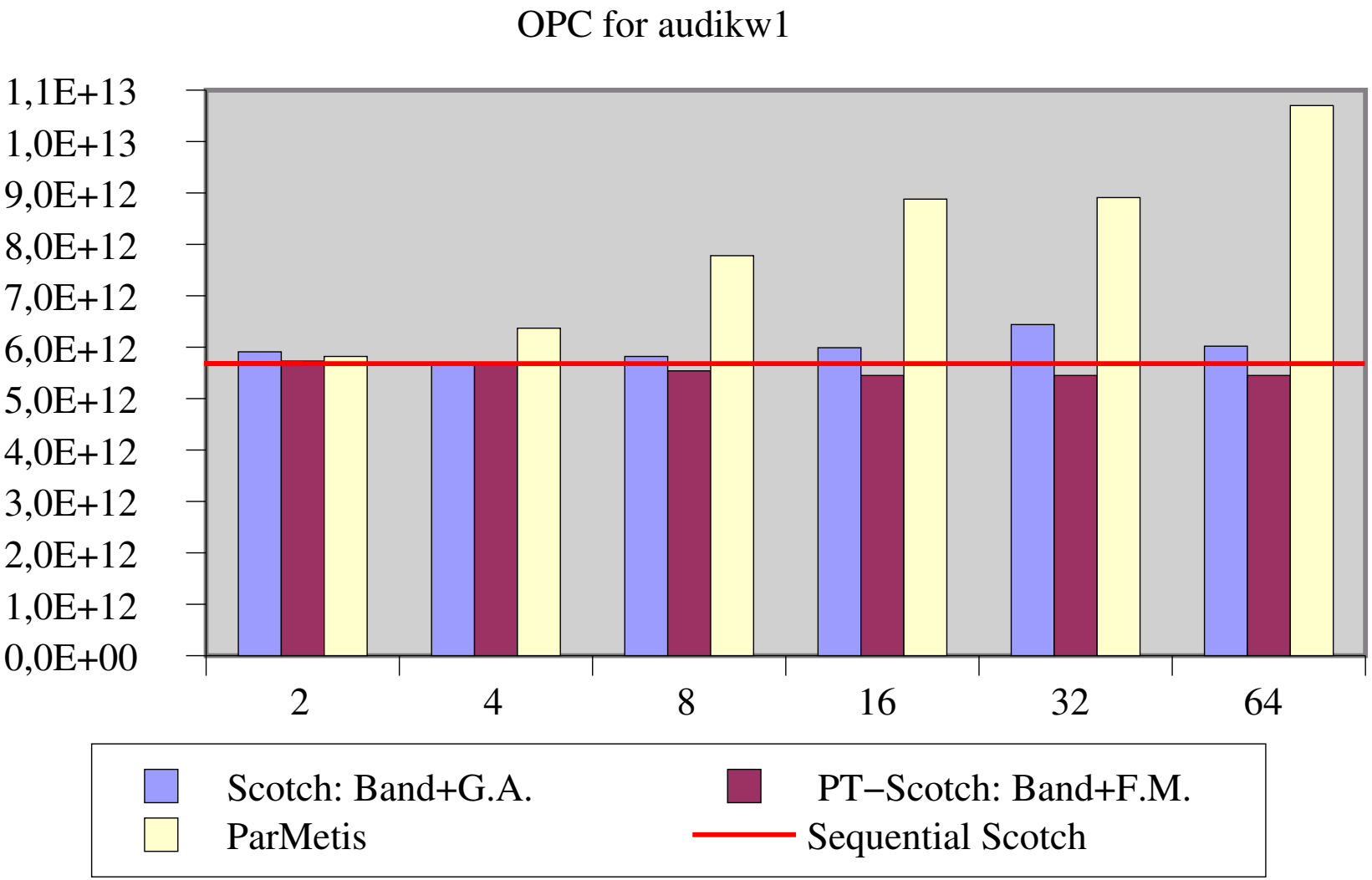
INRIA FUTURS

Experimental results (1)

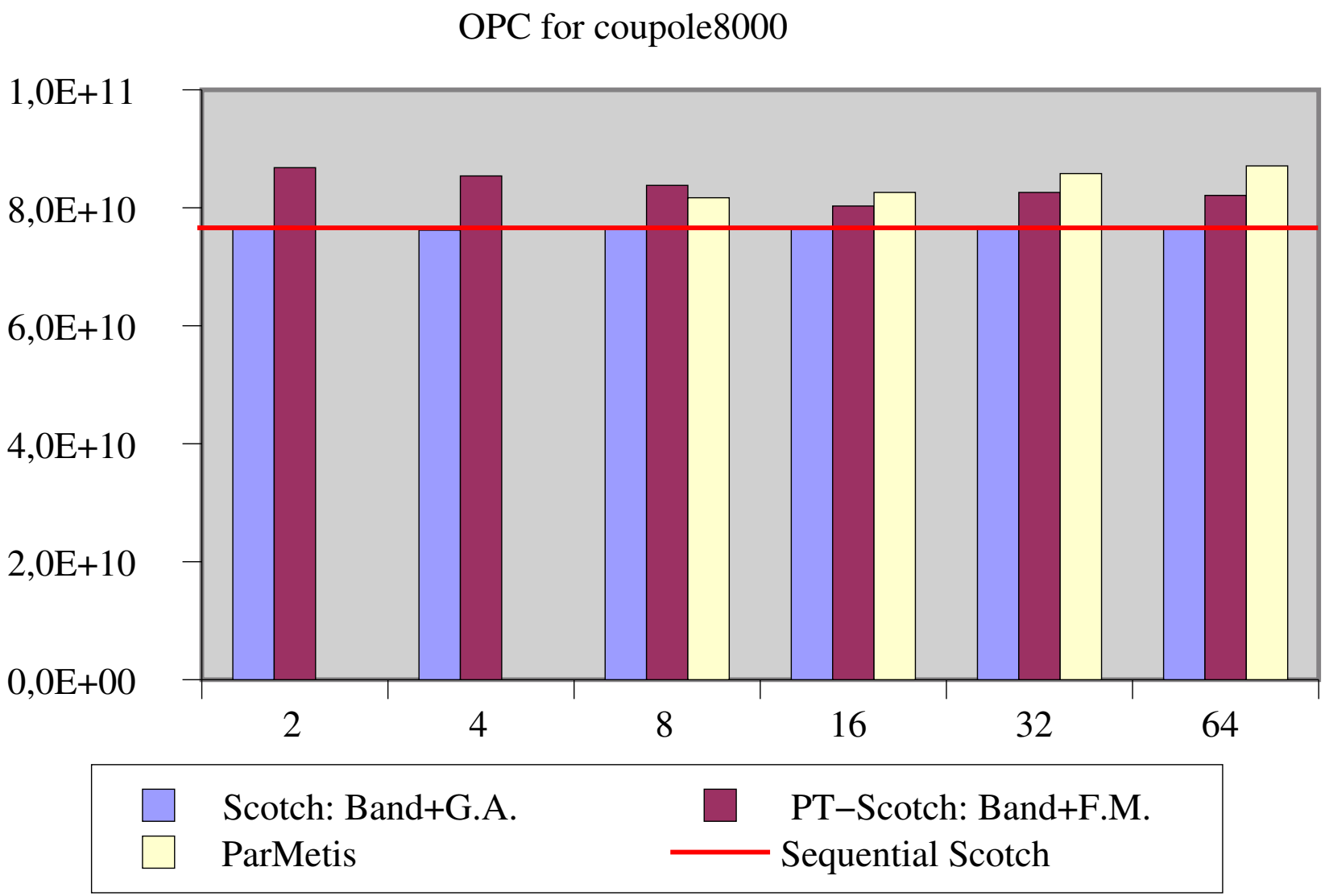
Our largest test graphs to date:

Graph	Size ($\times 10^3$)		$\bar{\delta}$	OPC	Description
	$\|V\|$	$\|E\|$			
audikw1	944	38354	81.28	5.48e+12	3D Mechanics mesh
coupole8000	1768	41657	47.12	7.66e+12	CEA/CESTA
brgm	3699	151940	82.14	2.70e+13	3D geophysics mesh
cage15	5154	47022	18.24	4.06e+16	DNA electrophoresis
qimonda07	8613	29143	6.76	8.92e+10	Circuit simulation
23millions	23114	175686	7.6	1.29e+14	CEA/CESTA

Results on audikw1 ($|V| = 0.94e+6, |E| = 38e+6$)



Results on coupole8000 ($|V| = 1.76e+6, |E| = 41e+6$)



Results with multi-sequential band FM

Test case	Number of processes					
	2	4	8	16	32	64
qimonda07 (seq : 8.92e+10)						
OPC	–	–	5.80e10	6.38e10	6.94e10	7.70e10
t(s)	–	–	34.68	22.23	17.30	16.62
brgm (seq : 2.70+13)						
OPC	2.70e13	2.55e13	2.65e13	2.88e13	2.86e13	2.87e13
t(s)	276.90	167.26	97.69	61.65	42.85	41.00
23millions (seq : 1.29e+14)						
OPC	1.45e14	2.91e14	3.99e14	2.71e14	1.94e14	2.45e14
t(s)	671.60	416.45	295.38	211.68	147.35	103.73

For all of these graphs, PARMETIS crashes
Runs done on a Power5 SMP computer, with 8 dual core CPU per node

INRIA FUTURS

Outline

INRIA FUTURS

Conclusion

- We have implemented the parallel ordering feature of PT-SCOTCH
- We can achieve the same quality than sequential tools and still remain scalable (save for multi-sequential refinement)

To do this we have developed and implemented several new algorithms which can be reused in the context of graph partitioning:

- Pre-constrained banding, which has also improved the sequential tool
- Multi-level parallelization

INRIA FUTURS

Conclusion (2)

For the sparse matrix reordering tool:

- The time taken by our algorithm is about twice the one of PARMETIS, but seems to be scalable
  $\longrightarrow$ Multi-sequential F.M. not scalable, visible on smaller graphs
- Quality is clearly better than the one of PARMETIS
  - Does not decrease on average when number of processors increases
  - Can even increase along with the number of processors

INRIA FUTURS

Conclusion

Work available as part of PT-SCOTCH in SCOTCH 5.0

Distributed on CeCiLL-C licence:
`http://gforge.inria.fr/projects/scotch/`

INRIA FUTURS

That's all folk

Thank you for your attention

Perspectives

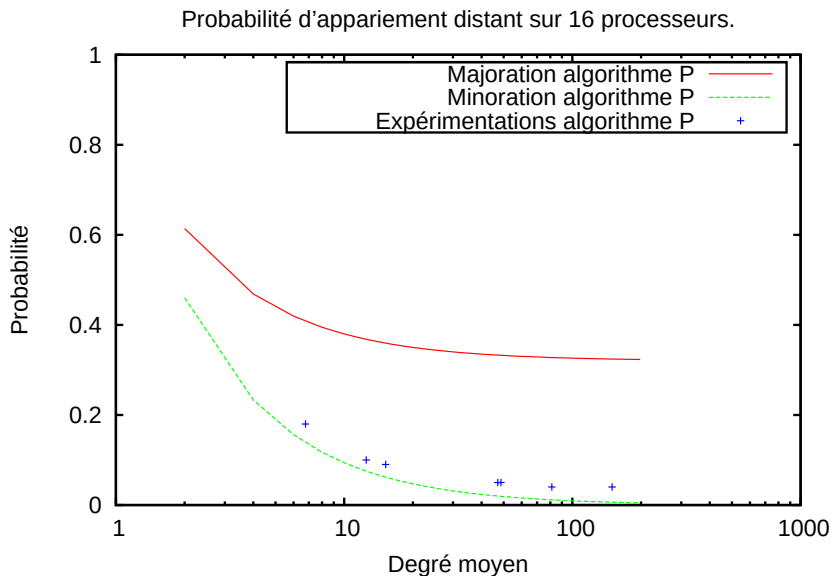
- Parallelize band refinement algorithms, for cases when band graphs will no longer fit in memory:
  - Fully parallel Genetic Algorithm which spreads individual chromosomes across processors
  - Parallel diffusion algorithm
- Basing on these existing parallel routines, design efficient and scalable k-way graph partitioning algorithms

INRIA FUTURS

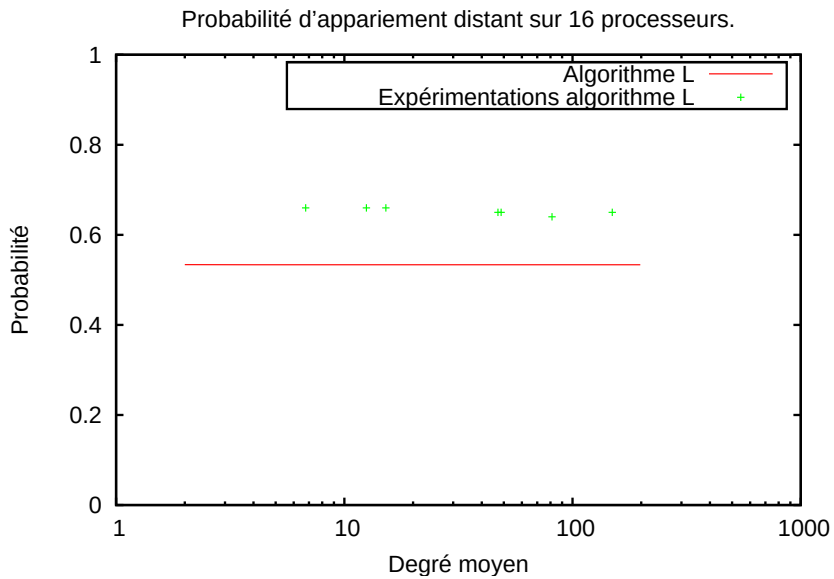
Comparison between synchronisation algorithms (1)

Remote matching rates during the first loop of synchronisation algorithms:

Algorithm P:



Algorithm L:



$\longrightarrow$ Very poor performance of algorithm P when average degree increases

Moreover:

- for algorithm P, performance decreases when number of processes increases
- for algorithm L, remote match rate increases when number of processes increases

Refinement problem in parallel