Design and Implementation of Multi-Threaded and Hybrid Parallel Graph Partitioning Algorithms in Scotch v7

François Pellegrini

Design and Implementation of Multi-Threaded and Hybrid Parallel Graph Partitioning Algorithms in Scotch v7

CSE 2023

F. Pellegrini

Sommaire

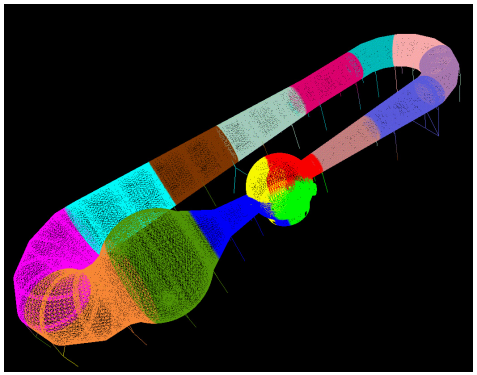
Inria

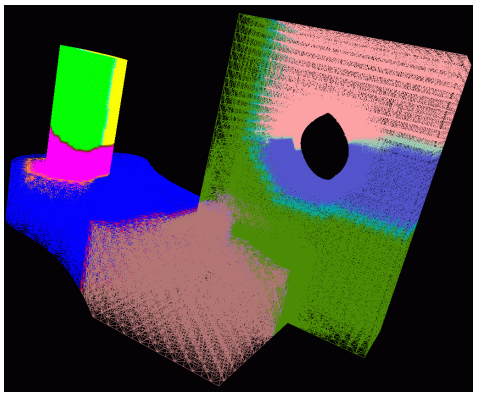
01

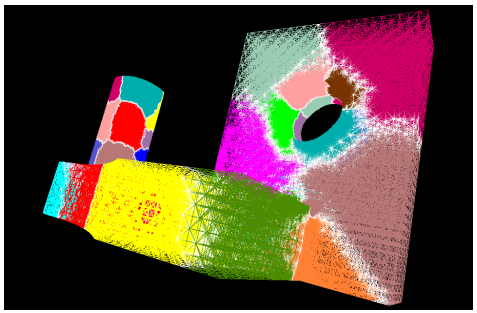
Graph partitioning & Scotch

Inria

- Two main problems are considered:
  - 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 positive-definite matrices
  - Edge separator problem for domain decomposition
  - Vertex separator problem for sparse matrix ordering by nested dissection
  - Also: partitioning with overlap







- Started 01 December 1992 (now 30 y.o.!)
  - Currently in v7.0.3
- Tackles the graph partitioning and mapping problems by way of algorithms that rely only on graph topology
  - Geometry is never taken into account
  - Hierarchical description of target architectures
    - Can also map onto parts of a regular architecture
- Provides a software toolbox:
  - SCOTCH centralized software library and tools
    - POSIX pThreads
  - PT-SCOTCH distributed-memory software library and tools
    - MPI + POSIX pThreads
  - Provides a compatibility library for replacing METIS & PARMETIS in existing software

- Stands for "Parallel and Threaded Scotch"
  - Offspring (and part) of the SCOTCH project
- Devise robust parallel graph partitioning methods
  - Meant to handle graphs of more than a billion vertices distributed across more than a thousand processors
  - Improve over sequential graph partitioning methods if possible
    - Devise new algorithms
    - Provide new features to existing algorithms, such as fixed vertices, multi-weight algorithms, etc.
  - Provide graph repartitioning and remapping methods

Inria

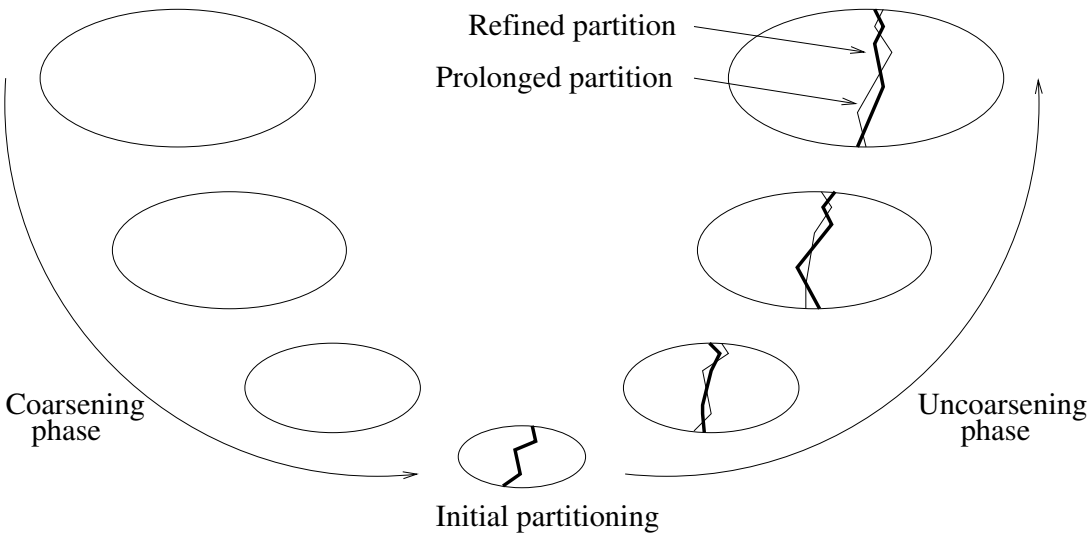
02

Sketch of SCOTCH partitioning algorithms

Inria

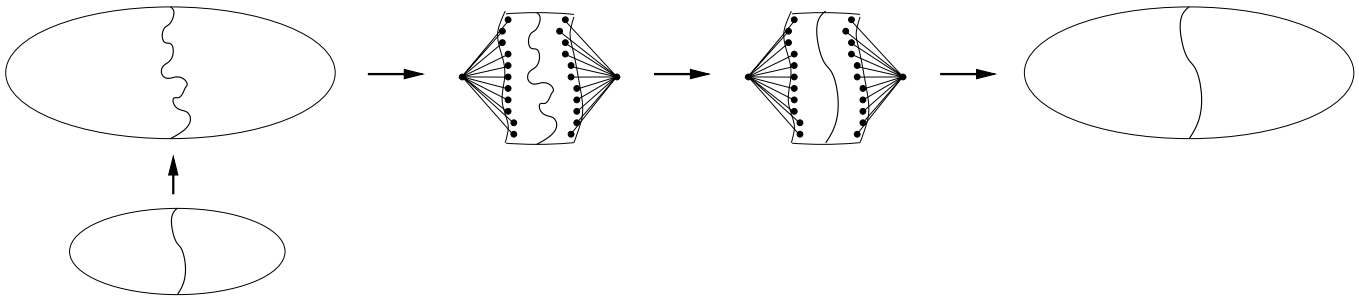
Multilevel framework

- Multilevel frameworks everywhere!
  - > For computing k-way edge-cut partitions
  - > For computing edge-cut bisections
    - – Used to compute by recursive bisection the initial partition on the coarsest graph of a k-way multilevel partitioning
  - > For computing recursive vertex-cut nested dissections
- Finest levels represent the bulk of the work

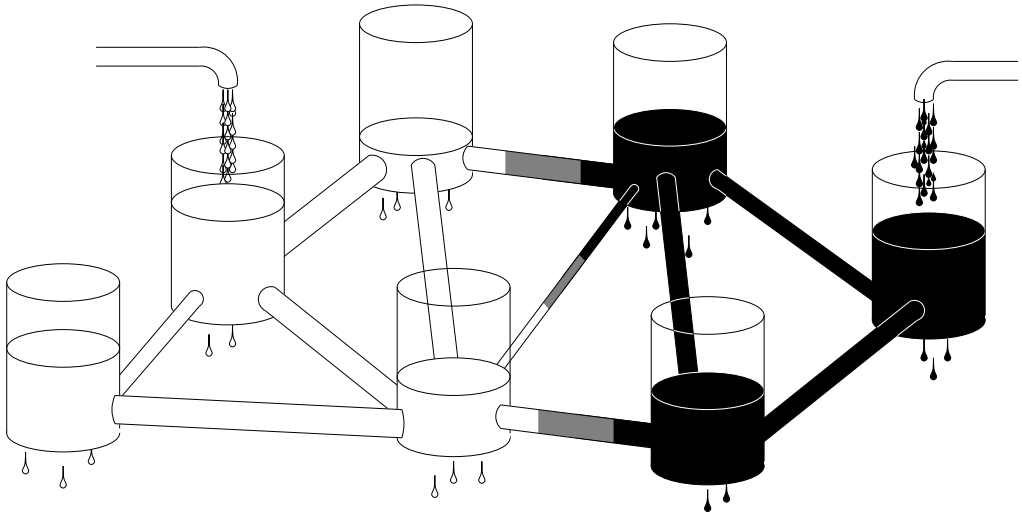


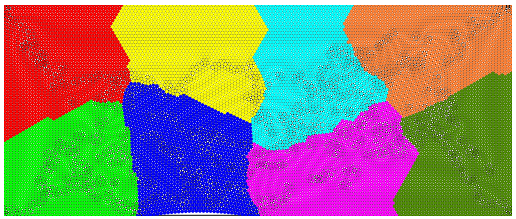
Band graph

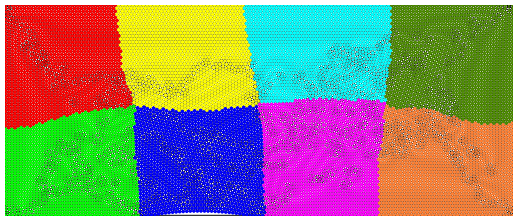
- During the uncoarsening phase, there is no need to consider modifications of the frontier that go beyond a short distance from the prolonged frontier
  - > Else, this modification would have been carried out at the coarser level
  - > Hence, a width of 3 around the frontier suffices
- Reduces the complexity and run-time of many local optimization algorithms
  - > Smaller graphs mean better data locality for computationally-intensive algorithms



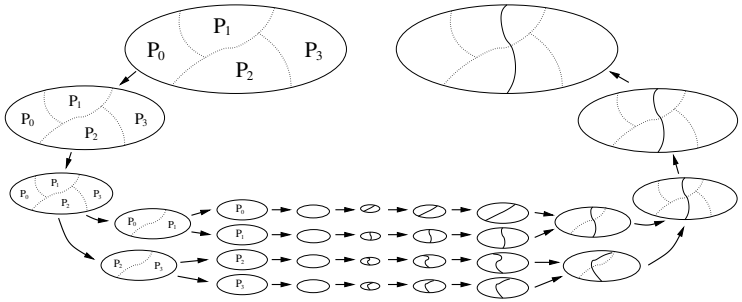
- Band graphs provide a perfect setting for diffusion-based partitioning algorithms
  - > Source and sink vertices already exist and are topologically relevant



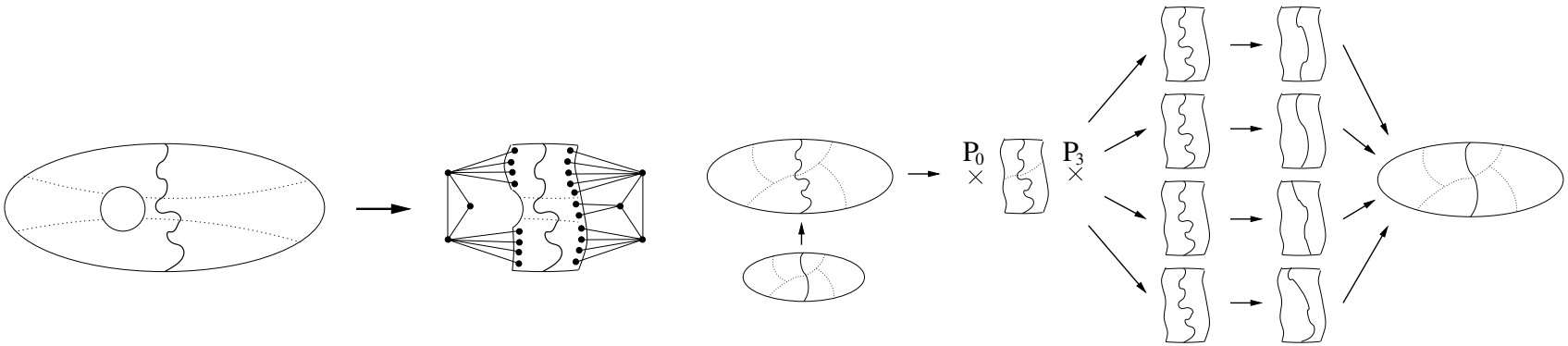




- Distributed multilevel framework
  - > With folding and possibly duplication



- Distributed band graphs



Areas for concurrency

- The multilevel framework(s)
  - Vertex matching
    - Can we preserve a deterministic behavior?
  - Construction of the coarsened graph
    - Each thread does not know is advance how many edges it will create
- Diffusion-based algorithms
  - Very good candidates but very expensive as well
    - About 40 times more than Fiduccia-Mattheyses
- The recursive bipartitioning framework
  - Requires to split in two a pool of threads
  - Requires a thread-safe MPI implementation for PT-Scotch

Inria

03

Challenges of multi-threading

Inria

Challenges of multi-threading

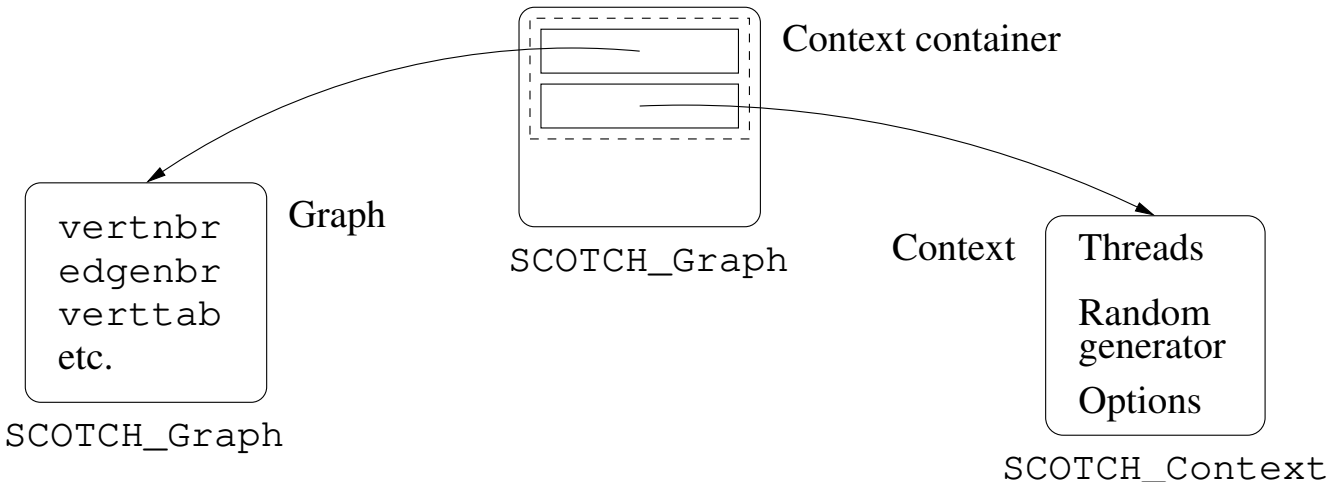
- Use all "available" threads
  - > Depending on the user's will
  - > Re-use existing user's threads if needed
- Run partitioning tasks concurrently
  - > Software must be fully reentrant
  - > Fine control on reproducibility
    - Control on (pseudo-)randomness
    - Control on thread concurrency (deterministic multi-threaded algorithms)

Design choices

- Handle threads manually
  - > No "foreign" and rigid top-down thread management like OpenMP
  - > Use raw POSIX pThreads
    - – Allows to capture existing threads
    - – Allows to assign sub-pools of threads to specific tasks
- Handle random generator(s) manually
  - > Lighweight Mersenne twisters
  - > Allows to assign a generator to each task and thread
- Provide deterministic variants of non-deterministic threaded algorithms
  - > Dynamic selection

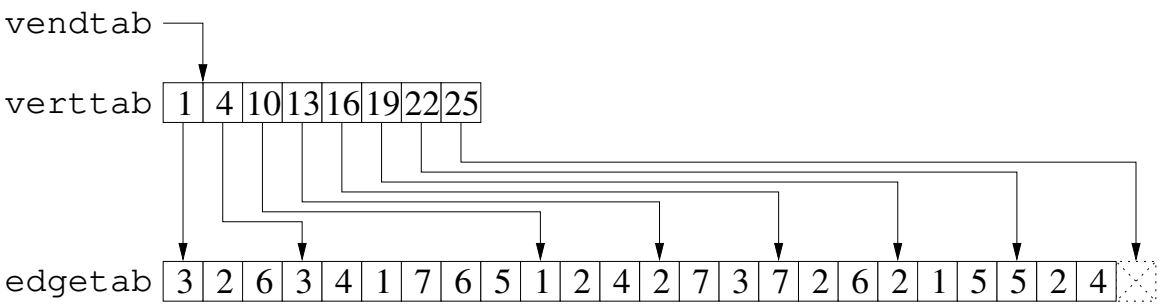
Context objects

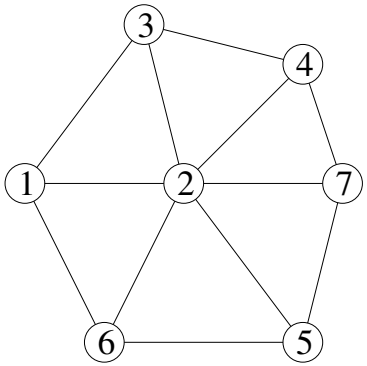
- Embed all run-time information for running a SCOTCH task
  - > Attached pool of threads
  - > Pseudo-random generator and seed
  - > Configuration options
- Encapsulated in opaque Container objects
  - > New opaque objects that pretend to be SCOTCH traditional objects such as Graph, Dgraph and Mesh
  - > No visible change to the SCOTCH interface!

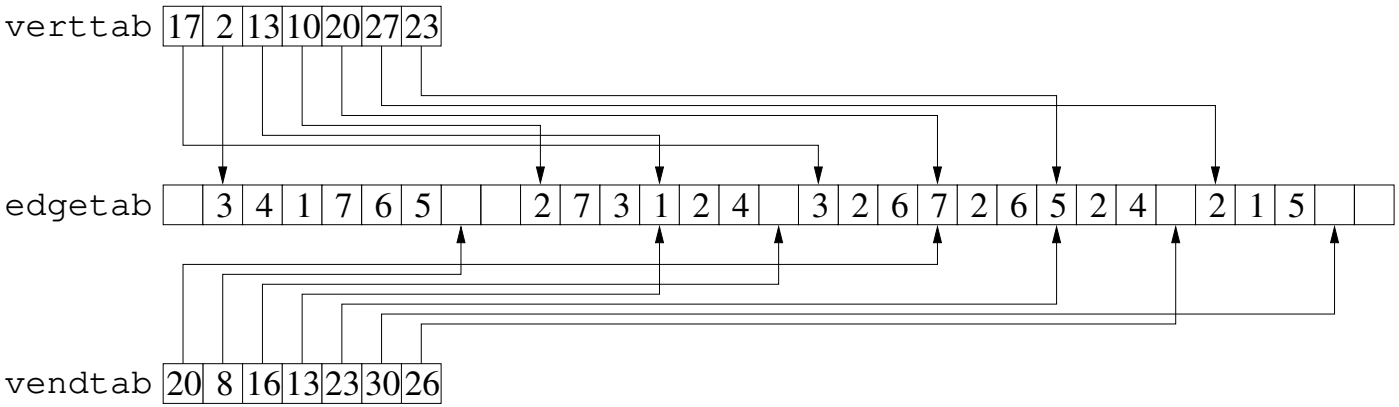


Non-compact Graph

- Feature existing since the inception of SCOTCH
  - > Allows users to make their graph evolve with time
- Facilitates the implementation of many threaded algorithms
  - > When the exact number of vertices to be created by each thread is bounded but not known in advance







04

Experiments

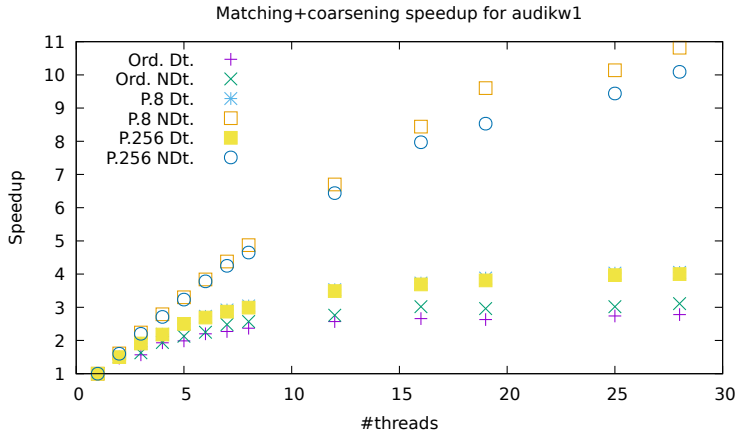
Inria

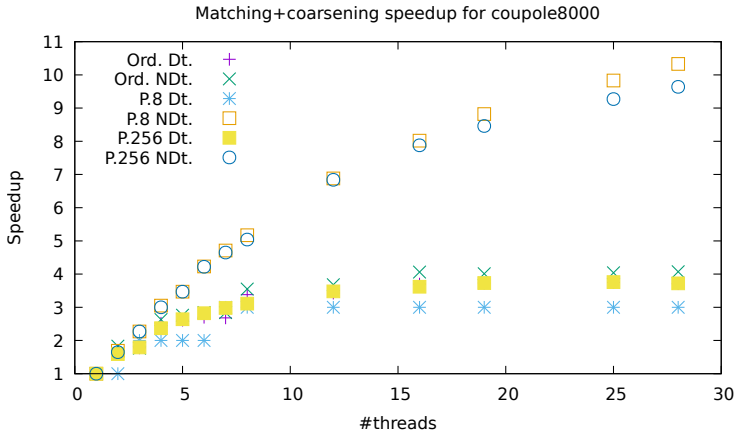
- Multiprocessor: dual-processeur Intel Xeon Broadwell E5-2650L v4 @1.7 GHz, 2 $\times$ 14 cores, 64 GiB main memory
- Test graphs:

Name	$\|V\|$	$\|E\|$	$\Sigma\delta/\|V\|$
audikw1	943695	38354076	26.66
coupole8000	1768161	41656975	10.94

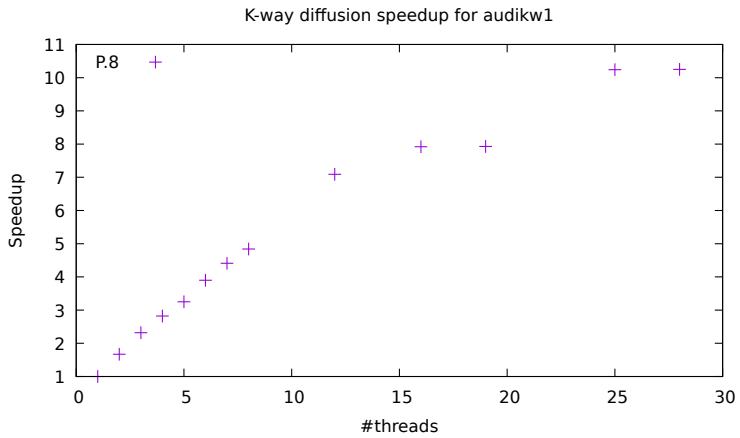
Matching and coarsening

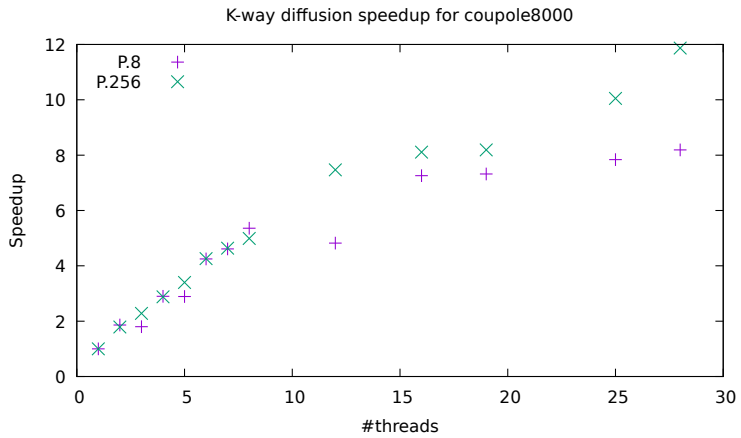
- The matching phase is multi-pass and has a significant impact on run time
- We do not have a multi-threaded deterministic matching algorithm
  - > Deterministic matching uses a sequential algorithm
- The speedup of the matching and coarsening process is bounded by Amdahl's law





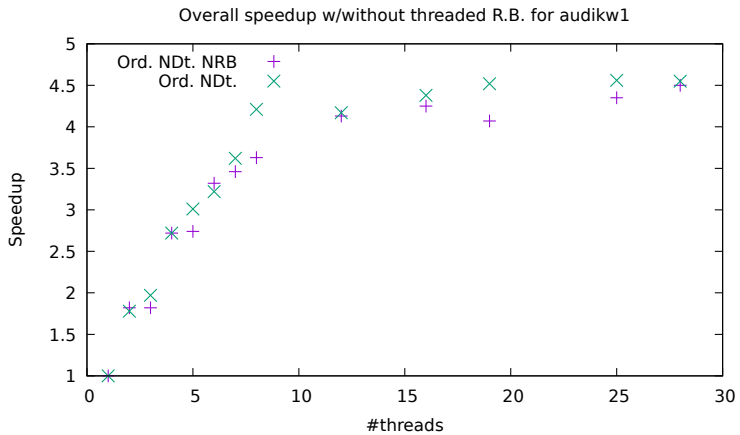
- Only used for graph partitioning
  - > Relies on a k-way band graph
  - > K-way band graphs may not be feasible when there are too many parts

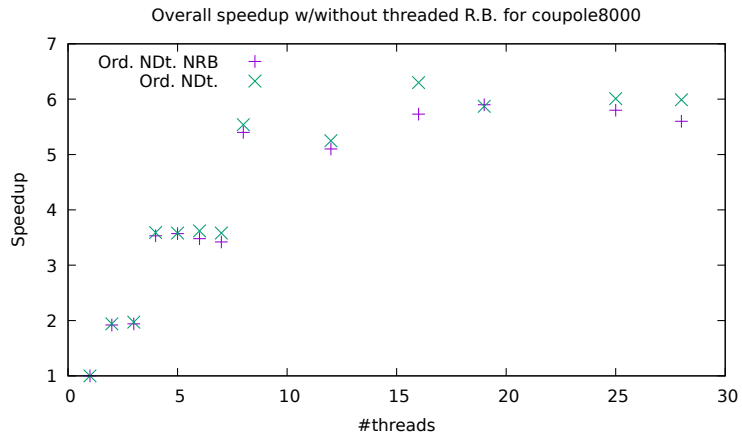




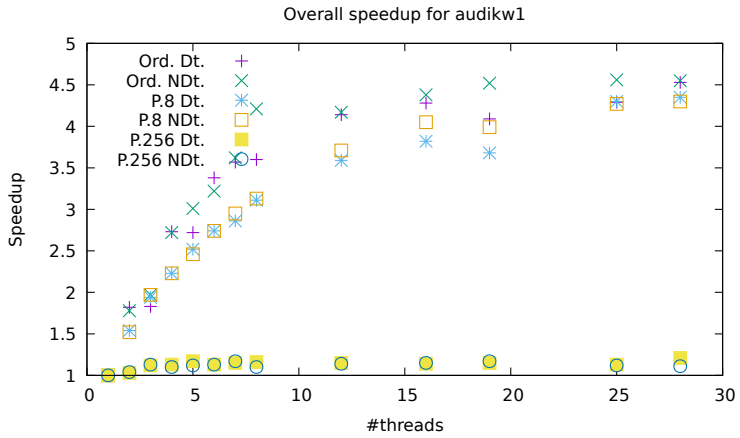
Recursive bipartitioning

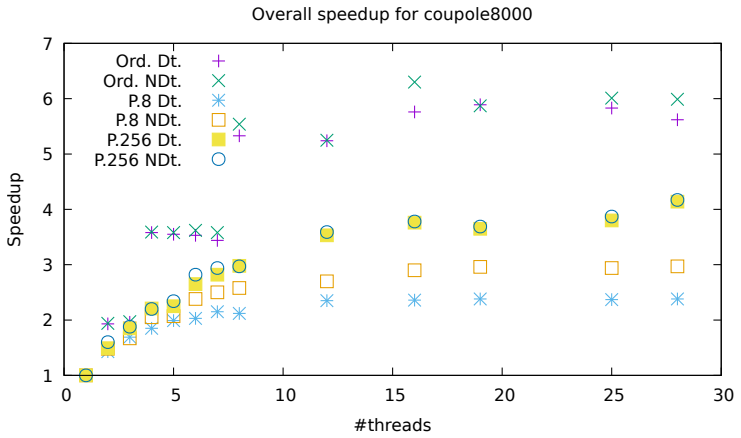
- After a cut is computed, both branches are explored concurrently, with half of the threads each
  - > Creates embarrassingly parallel computations
  - > Yet without any control on load balance
  - > Only valid for partitioning, not mapping
- Not significant in the case of direct k-way graph partitioning
  - > The coarsest graph of the k-way multilevel framework is small
- Competition for threads between recursive bipartitioning and the multilevel framework
  - > No clear winner at this stage





- When k-way band graphs cannot be created, run time is dominated by sequential local optimization algorithms (e.g. Fiduccia-Mattheyses)





05

Conclusion

Inria

Critical review

- Graph partitioning is essentially a memory-bound problem
  - > Quasi-linear-time heuristics are available for many kinds of common graphs (meshes, etc.)
  - > Algorithms that parallelize well are much more expensive (e.g. diffusion-based algorithms, genetic algorithms)
    - – Yet they do not scale enough to fully replace their sequential counterparts in current settings
- Multi-threaded algorithms in Scotch v7.0 are however useful in production contexts
  - > Make sure data placement is not "too expensive" compared to subsequent computations
  - > The new thread model gives much more flexibility to users

On-going work

- Implementation of a multi-threaded genetic algorithm
  - > Re-engineering within the new SCOTCH thread model of a pre-existing and obsoleted genetic algorithm
    - Benefits from reduced problem size with band graphs
    - More suited to static mapping than diffusion-based methods
- Analysis of the efficiency of threads in PT-SCOTCH v7.0
  - > E.g. with respect to communication
- Several other scheduled works:
  - > Vertex separation method that balances the size of the halos between the two parts (v7.1)
  - > Multi-constraint partitioning (v8.0)
  - > Parallel static mapping and dynamic remapping
  - > Etc.

- Aims at gathering organizations that want to secure the future of SCOTCH as an essential toolbox for their activities:
  - Maintenance
  - New developments
- Call for founding members is on-going
- All the relevant information is here:

`https://team.inria.fr/tadaam/call-for-founding-members-for-the-scotch-consortium/`