Shared memory parallel algorithms in Scotch 6

François Pellegrini

EQUIPE PROJET
BACCHUS
Bordeaux
Sud-Ouest

29/05/2012

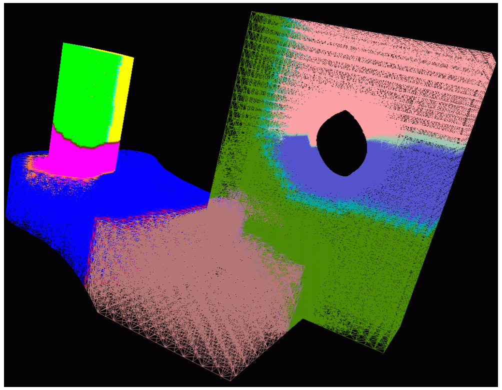
Outline of the talk

- Context
- Why shared-memory parallelism in Scotch ?
- How to implement it
- Next steps

Context

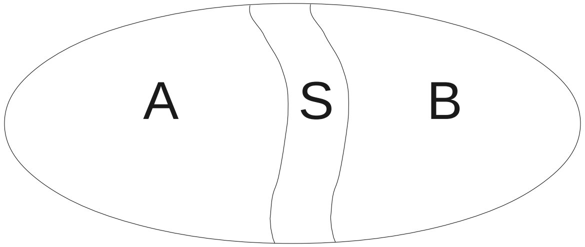
Context

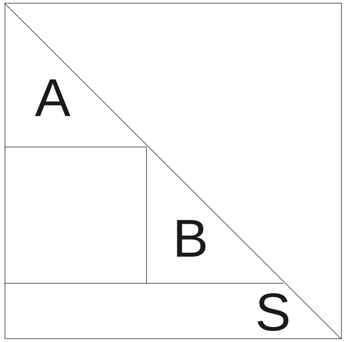
- Scotch is used in MUMPS to compute sparse matrix orderings
- Orderings computed by a combination of methods into strategies
  - Nested dissection on the whole graph
  - Approximate minimum fill on the leaves of the separation tree

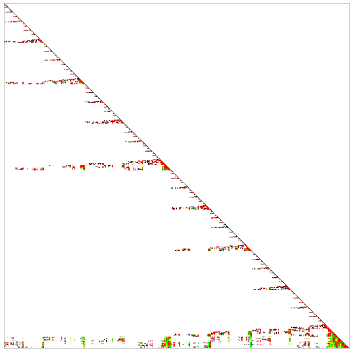


Nested dissection

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

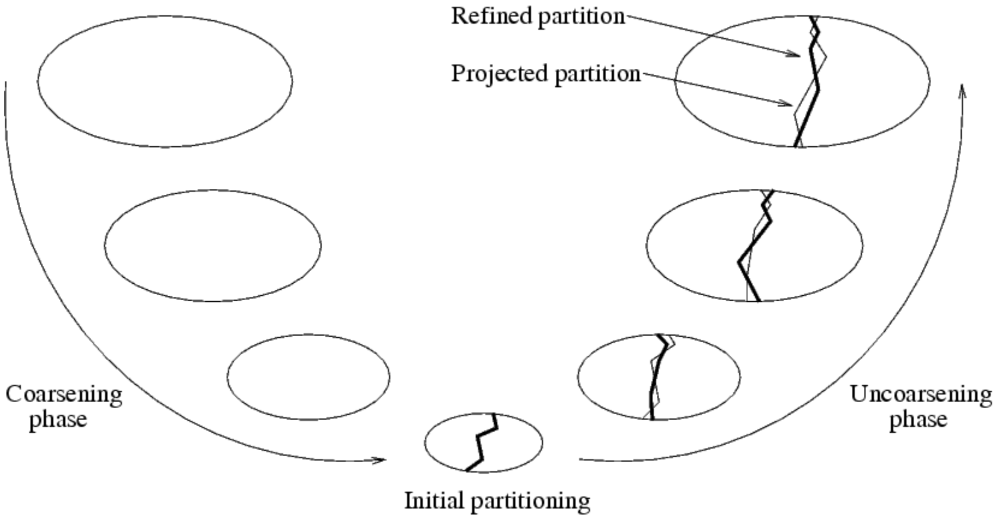






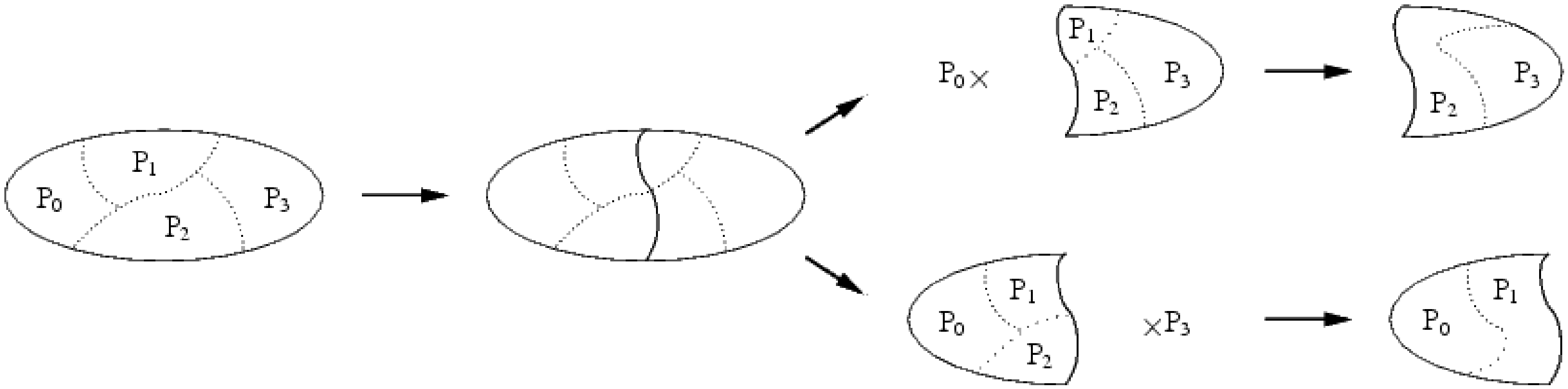
Multi-level framework

- Each bipartitioning is computed using a multilevel framework
  - Successive coarsenings by quotienting (matching)
  - Initial partitioning of the smallest graph
  - Prolongation of the result with local refinement



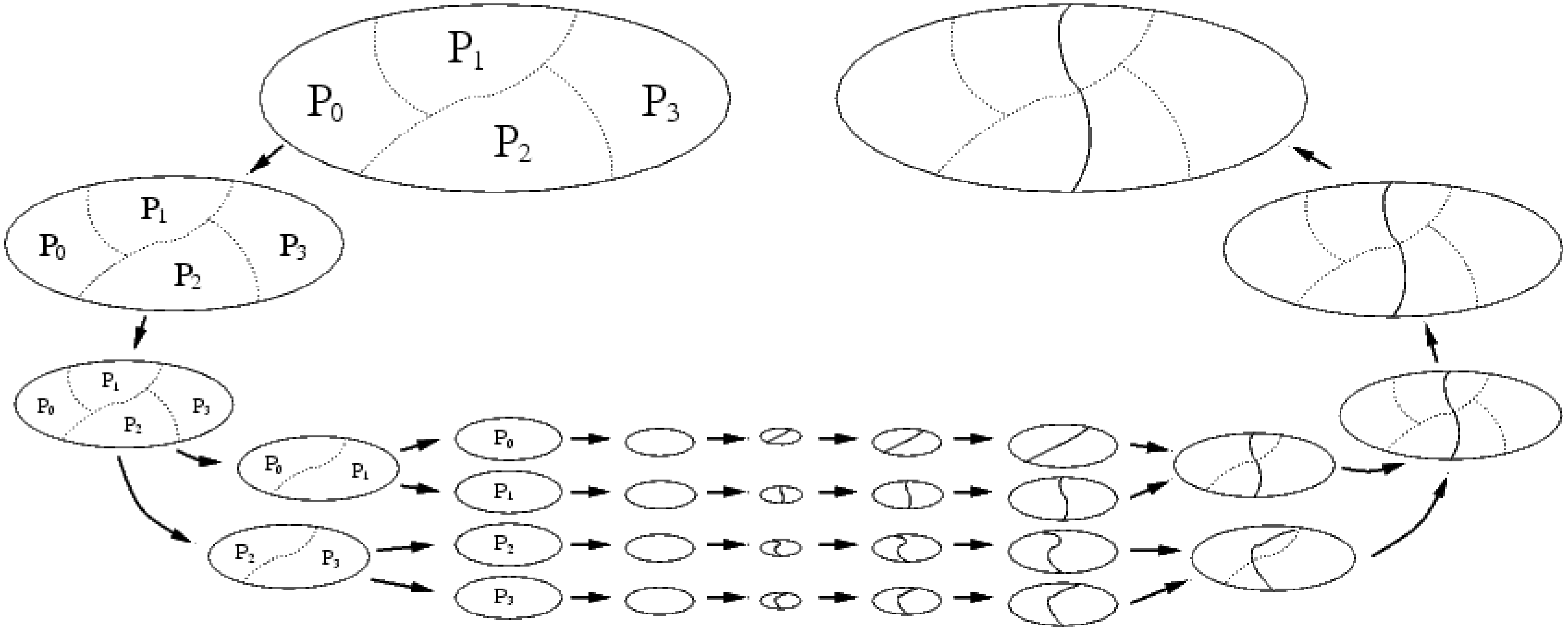
Parallelism in Scotch v5 (1)

- Orderings can be computed :
  - Sequentially : Scotch library
  - In parallel : PT-Scotch library
- Distributed-memory parallelism
  - Only very limited use of shared-memory threads



Parallelism in Scotch v5 (2)

- The bulk of the work is performed during the coarsening and the uncoarsening phases



Why shared-memory parallelism ?

Reasons for shared-memory parallelism

- Sparse matrix ordering sometimes represents a significant overhead of sparse linear system solving in MUMPS (and other tools)
  - Both for sequential and parallel versions
- For the sequential version :
  - It is too bad not to take advantage of multi-core processors on “sequential” computers and workstations, while MUMPS does
- For the parallel version :
  - It is too bad to resort only to distributed-memory parallelism when parallel architectures possess shared-memory nodes

How to implement it

Basic blocks

- Use of two (hopefully common) technologies :
    - POSIX Pthreads
    - Atomic built-ins
        - __sync_lock_test_and_set () and its friends...
- OpenMP is cool, but sometimes our algorithms require fine synchronization and complex primitives
    - E.g. MPI-like reduction operations, scan, etc.
    - We may lose some cycles when launching threads wrt. OpenMP, though

Target algorithms

- We started with the sequential coarsening algorithms :
  - Matching
  - Graph coarsening
- Implementation already available in Scotch 6.0.0
  - 37% overall improvement in run time on 8 threads
  - The uncoarsening phase is not parallelized yet...

How not to change the interface... (1)

- The Scotch API routines handle opaque SCOTCH_Graph and SCOTCH_Dgraph objects only
    - No additional « options » structure passed, that could hold threading information
- Such an optional argument would have been irrelevant for most publicized API routines
    - Graph coarsening, graph induction, graph coloring, etc...
- Yet, we want these algorithms to be run in parallel

How not to change the interface... (2)

- Handling of multi-threading cannot be performed in the strategy string
  - Because it also concerns the aforementioned routines
  - We must provide a homogeneous mechanism
- Handling of multi-threading should not be attached to the graph structures
  - Because several algorithms can be applied in parallel to the same graph structure
- We don't want to change the interface !

How not to change the interface... (3)

- We plan to create a SCOTCH_Context opaque data structure, that will :
  - Refer internally to the SCOTCH_Graph and SCOTCH_Dgraph data structures
  - Hold optional data such as the number of threads
- Scotch API routines will :
  - Still accept SCOTCH_Graph's and SCOTCH_Dgraph's when default behavior is expected
    - Use of all threads available to the calling thread
  - Accept SCOTCH_Context's when specific behavior is expected

Next steps

Next steps (1)

- Integrate the « hwloc » library
  - Library designed within the RUNTIME Inria Project-Team
  - Will allow us to handle thread locality issues in a platform-independent way
  - First third-party library in Scotch ever
    - Its use would be parametrized, though
    - We already do this for the Linux threads that we added in Scotch 6.0
- Provide threading on Windows
  - Compatibility library provided by Samuel Thibault
- All of the above in Scotch v6.1

Next steps (2)

- Parallelize « sequential » uncoarsening and some local optimization methods
  - Fiduccia-Mattheyses algorithms cannot be parallelized
  - Some threads may be used to run several algorithms at the same time
    - This has to be expressed within the strategy string !
- Add shared-memory parallelism to existing distributed-memory parallel coarsening and uncoarsening methods
  - Turn them into hybrid algorithms
- Partial implementation of the above planned for Scotch v6.2

Thank you for your attention !

Any questions ?

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