




Shared memory parallel algorithms in *Scotch* 6

François Pellegrini

EQUIPE PROJET
BACCHUS
Bordeaux
Sud-Ouest

29/05/2012

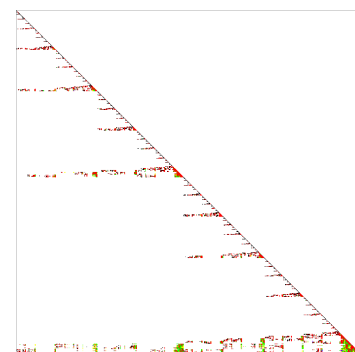
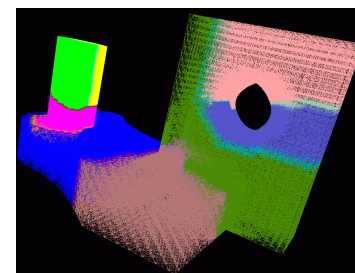
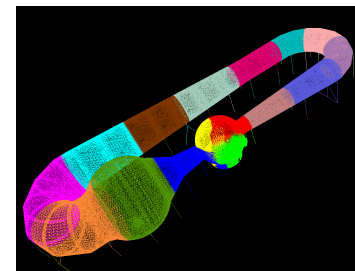
Outline of the talk

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

Context

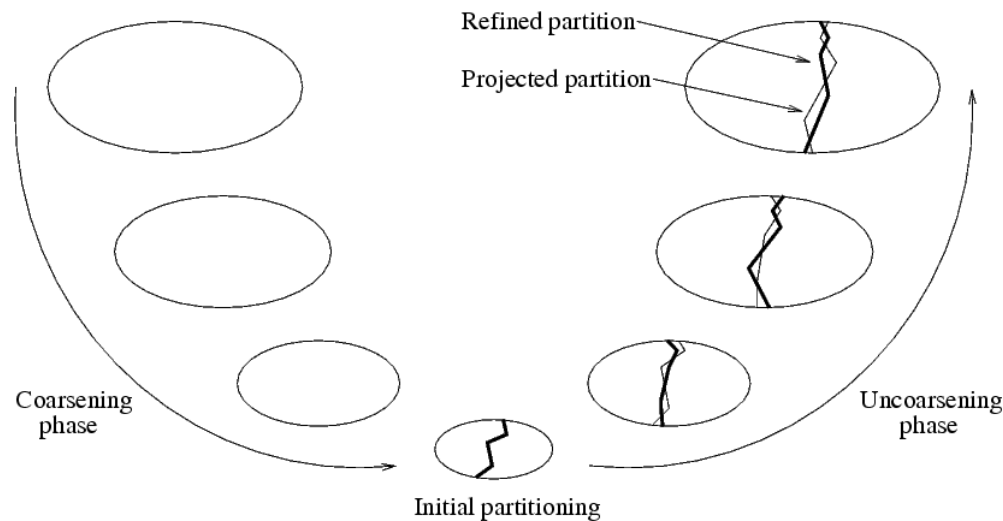
The **Scotch** project

- Toolbox of graph partitioning, static mapping and clustering methods
- Sequential **Scotch** library
 - Graph and mesh partitioning
 - Static mapping (edge dilation)
 - Graph and mesh reordering
 - Graph repartitioning and remapping [v6.0]
- Parallel **PT-Scotch** library
 - Graph partitioning (edge)
 - Static mapping (edge dilation) [v6.1]
 - Graph reordering
 - Graph repartitioning and remapping [v6.1]



Multilevel framework

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



Static mapping

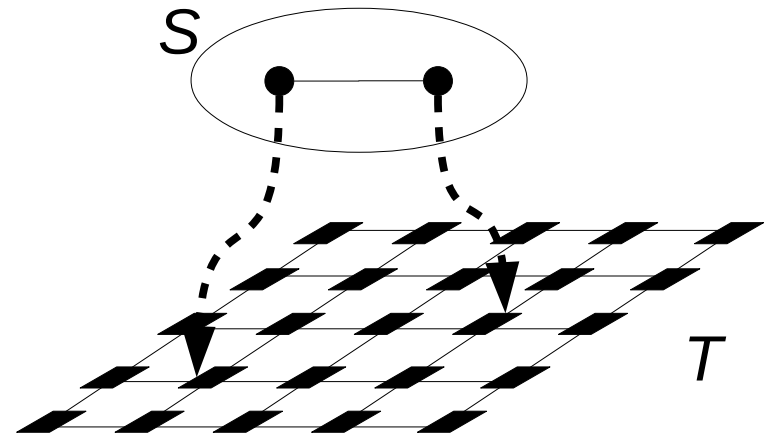
- Compute a mapping of $V(S)$ and $E(S)$ of source graph S to $V(T)$ and $E(T)$ of target architecture graph T , respectively

- Communication cost function accounts for distance

$$f_C(\tau_{S,T}, \rho_{S,T}) \stackrel{\text{def}}{=} \sum_{e_S \in E(S)} w(e_S) |\rho_{S,T}(e_S)|$$

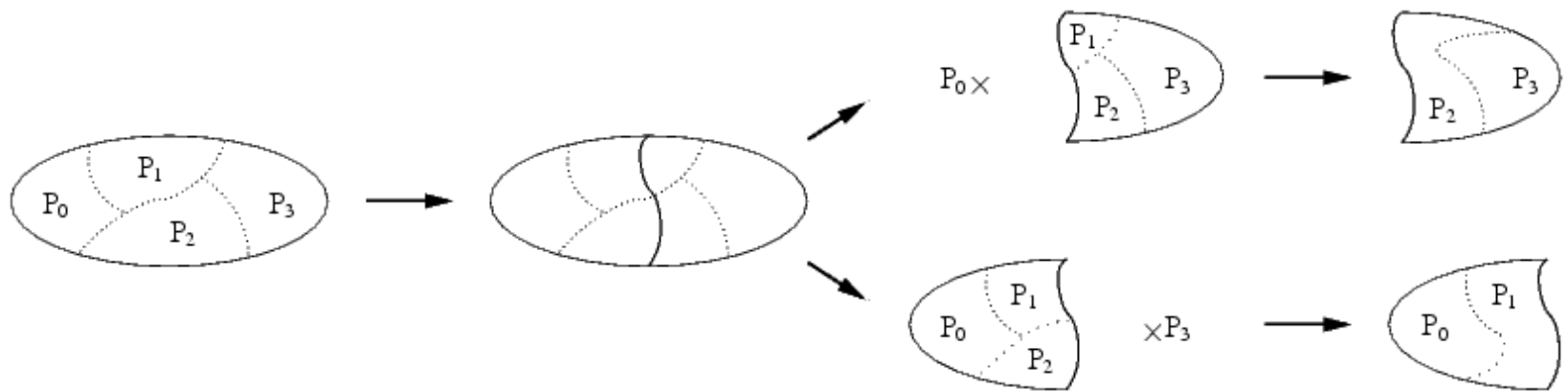
- Static mapping features are present in the sequential **Scotch** library since its inception

- Now in **PT-Scotch** v6
 - PhD of Sébastien Fourestier



Parallelism in Scotch v5 (1)

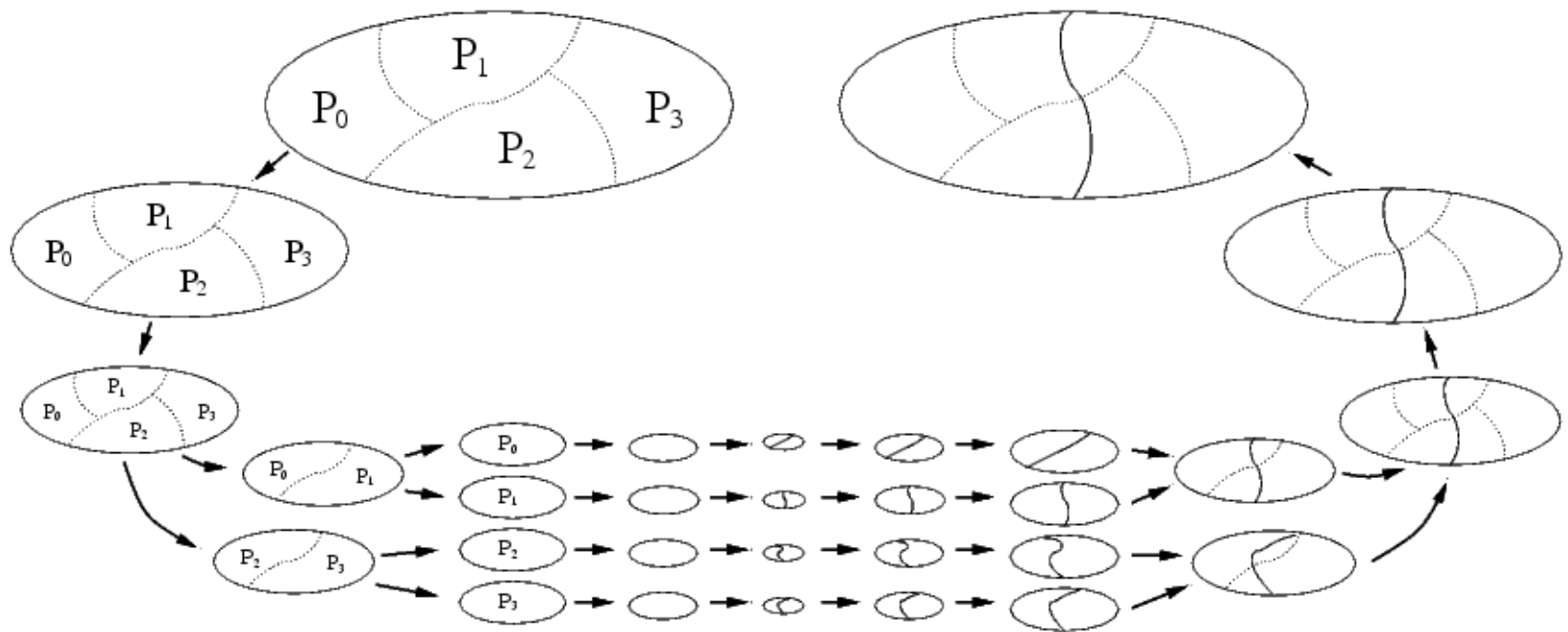
- Distributed-memory parallelism in **PT-Scotch** only
 - Only very limited use of shared-memory threads for recursive bipartitioning



- Yet we wanted to move to direct k-way partitioning
 - Recursive bipartitioning useful only for sparse matrix ordering by nested dissection

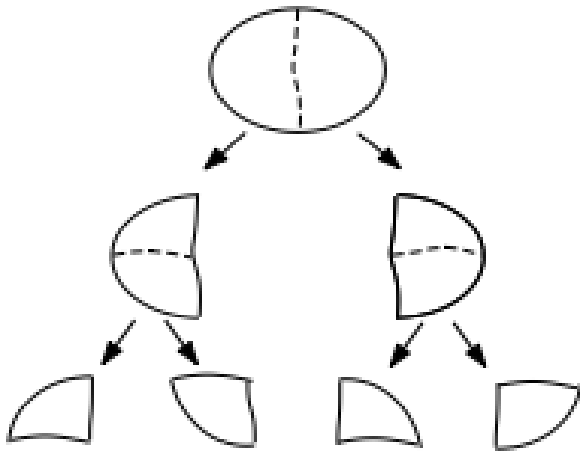
Parallelism in Scotch v5 (2)

- The bulk of the work is performed during the coarsening and the uncoarsening phases
 - Not strictly uncoarsening but local optimization

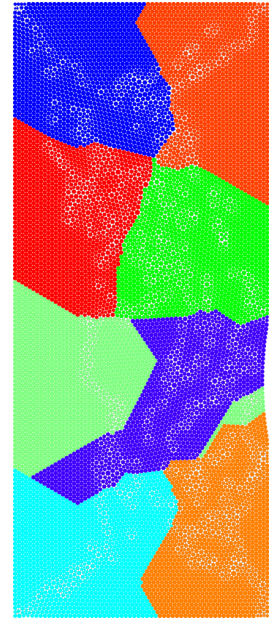
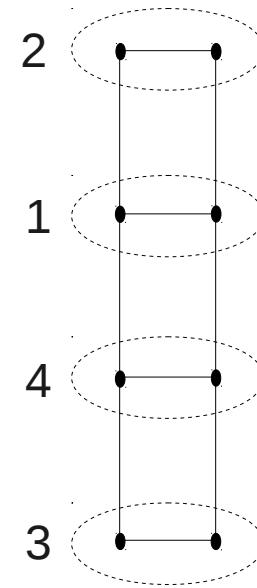
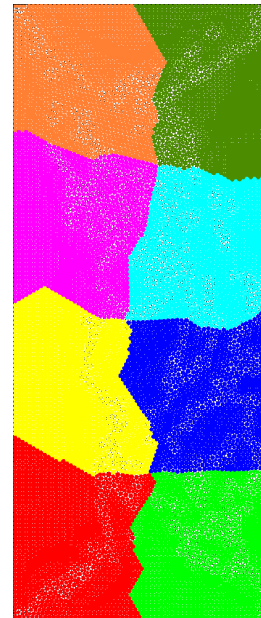


Parallel static mapping

- Recursive bi-mapping won't do in parallel
 - All subgraphs at some level are supposed to be processed simultaneously for parallel efficiency
 - Yet, ignoring decisions in neighboring subgraphs can lead to “twists”

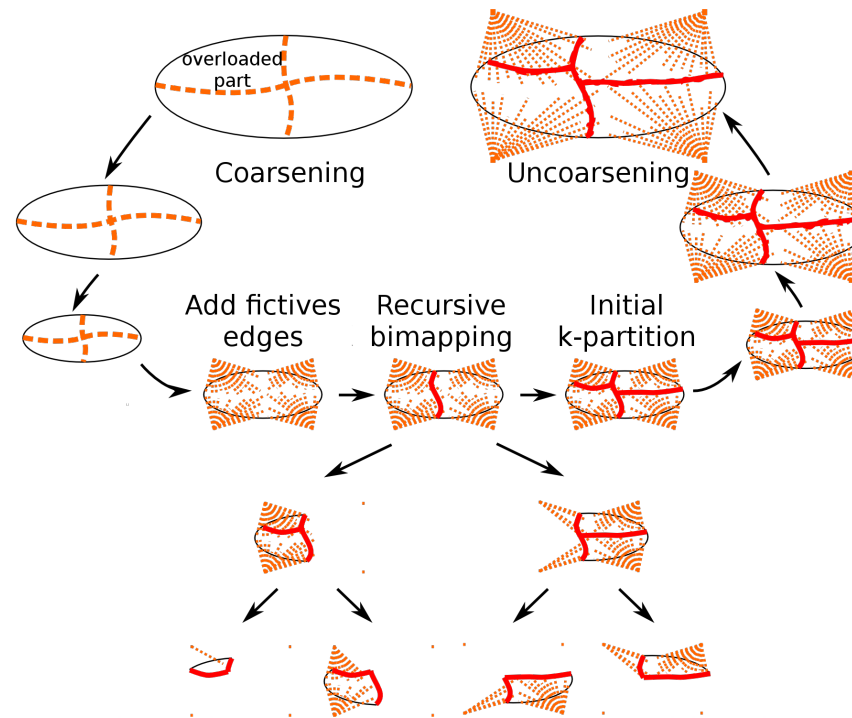


- Sequential processing only!



Parallel dynamic remapping

- Bias cut cost function with fictitious edges [Devine *et al.*]
- Take advantage of the k-way multilevel framework
 - Initial mapping is computed sequentially (no twists !)
 - Take dilation into account during k-way refinement
 - Sequential initial task may become too large some day



Why shared-memory parallelism ?

Reasons for shared-memory parallelism

- For the “sequential” version :
 - It is too bad not to take advantage of multi-core processors on “sequential” computers and workstations
 - Users don't want to meddle with new interfaces and third-party libraries they do not know
- For the parallel version :
 - It is too bad to resort only to distributed-memory parallelism when parallel architectures possess shared-memory nodes
 - Critical for exascale-class machines

How to proceed ?


- Three criteria to consider :
 - Locality, locality, locality !
- We are already bad at that :
 - Our op/byte rate is low
 - Initial graph data may be randomly distributed

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

First experiments in the sequential realm

- We started with the sequential coarsening algorithms :
 - Matching
 - Graph coarsening
 - Diffusion-based local optimization
- Implementation already available in  6.0.0
 - 37% overall improvement in run time on 8 threads

And in the parallel realm ?

- When computing an initial partitioning, data can be distributed arbitrarily
 - We already knew that for the sequential case
 - Some works on GPU algorithms [Fagginger Auer]
 - Matching algorithms are likely to be here just to compete to fill mating request buffers...
 - Coarsened graph building routines may behave better
 - Yet not much shared-memory locality to expect
 - Local optimization algorithms are expected to be scalable, though
- The situation should improve for dynamic repartitioning

(Not so) trivial aspects

Better handling of threading issues

- 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

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

In the context of the JLPC

- Seeking early users for the parallel repartitioning (and remapping !) features of the upcoming v6.1
 - Sanjay's group on Charm++
 - Other volunteers ?

Thank you for your attention !

Any questions ?

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