

PT-SCOTCH: A tool for efficient parallel graph ordering

Cédric Chevalier and François Pellegrini

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

PMAA'06, Rennes

Outline

- 1 Introduction
 - Graph Partitioning
 - Use of Graph Partitioning
 - The Multi-Level Framework
- 2 Algorithms for efficient parallel reordering
 - Nested Dissection Parallelism
 - Parallelization of the Multi-level algorithm
 - Parallelization of refinement
- 3 Results
 - Time Results
 - Quality Results
- 4 Conclusion

Graph Partitioning

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

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

Use of graph partitioning

- Many applications : load balancing, matrix ordering, database storage, VLSI design, bio-informatics . . .
- Many sequential graph partitioning tools already exist
⇒ SCOTCH, developed within the SCALAPPLIX team at INRIA Futurs
- But size of problems increases steadily
⇒ Need for a parallel graph partitioner as large graphs cannot fit in the memory of a sequential computer
 - The PT-SCOTCH (*Parallel Threaded SCOTCH*) project
 - Currently focusing on matrix ordering (recursive graph bisection problem)

Use of graph partitioning

- Many applications : load balancing, matrix ordering, database storage, VLSI design, bio-informatics . . .
- Many sequential graph partitioning tools already exist
⇒ SCOTCH, developed within the SCALAPPLIX team at INRIA Futurs
- But size of problems increases steadily
⇒ Need for a parallel graph partitioner as large graphs cannot fit in the memory of a sequential computer
 - The PT-SCOTCH (*Parallel Threaded SCOTCH*) project
 - Currently focusing on matrix ordering (recursive graph bisection problem)

Use of graph partitioning

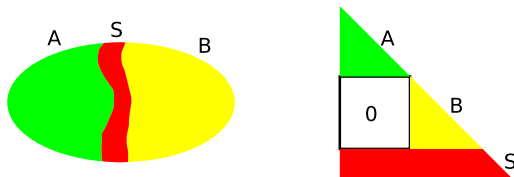
- Many applications : load balancing, matrix ordering, database storage, VLSI design, bio-informatics . . .
- Many sequential graph partitioning tools already exist
⇒ SCOTCH, developed within the SCALAPPLIX team at INRIA Futurs
- But size of problems increases steadily
⇒ Need for a parallel graph partitioner as large graphs cannot fit in the memory of a sequential computer
 - The PT-SCOTCH (*Parallel Threaded SCOTCH*) project
 - Currently focusing on matrix ordering (recursive graph bisection problem)

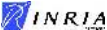
Matrix Ordering

- When solving sparse linear systems with direct methods, non-zero terms are created during the factorization process ($A \rightarrow LL^t$, $A \rightarrow LDL^t$ or $A \rightarrow LU$)
- Fill-in depends on the order of the unknowns
⇒ 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, OPeration Count of Cholesky factorization (overall number of additions, multiplications and divisions)

Matrix Ordering with Nested Dissection

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



- Interest
 - Induces high quality block decompositions
 - Increases the concurrency of computations
 ↪ very suitable for parallel factorization (PASTIX solver) 

The Multi-Level Framework

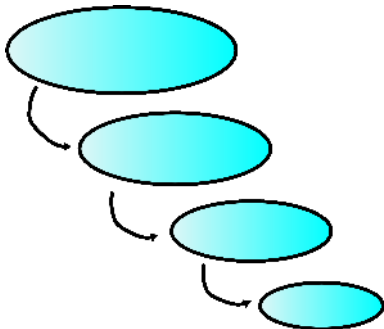
A classical approach to improve partition quality (Barnard and Simon, 1994)

Three steps

- 1 **Coarsening phase**
- 2 Initial separation
- 3 Uncoarsening phase

Decrease the number of vertices by merging pairs of neighbour vertices

⇒ At every step, obtain a smaller graph with the same topology



The Multi-Level Framework

A classical approach to improve partition quality (Barnard and Simon, 1994)

Three steps

- 1 Coarsening phase
- 2 **Initial separation**
- 3 Uncoarsening phase

Apply a local 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

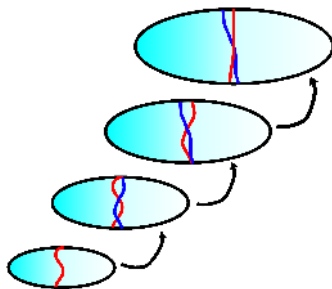
The Multi-Level Framework

A classical approach to improve partition quality (Barnard and Simon, 1994)

Three steps

- 1 Coarsening phase
- 2 Initial separation
- 3 **Uncoarsening phase**

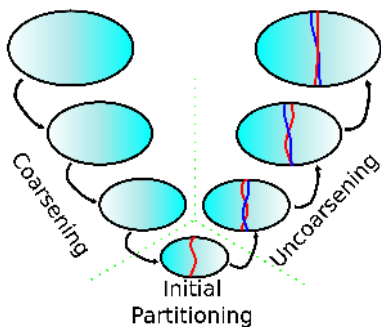
Project the computed partition from the coarsest graph to finer graphs
 Locally optimise using heuristics such as Kernighan-Lin or Fiduccia-Mattheyses (F.M.)



The Multi-Level Framework

Three steps

- 1 Coarsening phase
- 2 Initial separation
- 3 Uncoarsening phase



Limits of this model in parallel

Multi-level algorithms are difficult to parallelize

- Problems with the coarsening step :
 - Parallel formulations of classical sequential coarsening algorithms require **many** distant communications to match vertices located on different processors
 - Coarsening quality decreases when local vertex matching is privileged
- The uncoarsening step is even harder :
 - Best refinement algorithms (like F.M.) are sequential by nature and do not parallelize well

Limits of this model in parallel (2)

Available parallel ordering tools such as PARMETIS also use multi-level schemes

To reduce the amount of communication required to optimize partition boundaries when vertices are located on distant processors, PARMETIS disables the hill-climbing capabilities of their parallel local optimisation algorithms

↔ Gradient-like method

↔ Very poor results when number of processors increases

Our efforts aim at allowing hill-climbing even in parallel in order to obtain the same quality as the best sequential methods

Outline

- 1 Introduction
 - Graph Partitioning
 - Use of Graph Partitioning
 - The Multi-Level Framework
- 2 Algorithms for efficient parallel reordering
 - Nested Dissection Parallelism
 - Parallelization of the Multi-level algorithm
 - Parallelization of refinement
- 3 Results
 - Time Results
 - Quality Results
- 4 Conclusion

Parallelization of multi-level graph bipartitioning

Our parallel graph bipartitioning algorithm exploits three levels of concurrency:

- 1 In the Nested Dissection process itself
- 2 In the Multi-Level coarsening algorithm
- 3 In the refinement process when uncoarsening

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

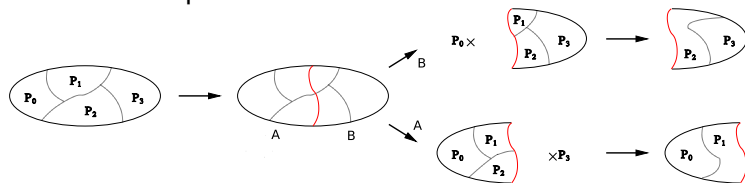
Can fold on any number of processors (not only powers of two)

⇒ Better data locality

⇒ The two subtrees are separated not only logically but also physically, which helps reducing network congestion

Nested Dissection Parallelism (2)

Outline of the process :



The sub-orderings of A and B are computed in parallel

Parallelization of the Multi-level algorithm

During the coarsening phase, every coarser subgraph is folded and duplicated on the remaining subset of processors until it is reduced to one processor

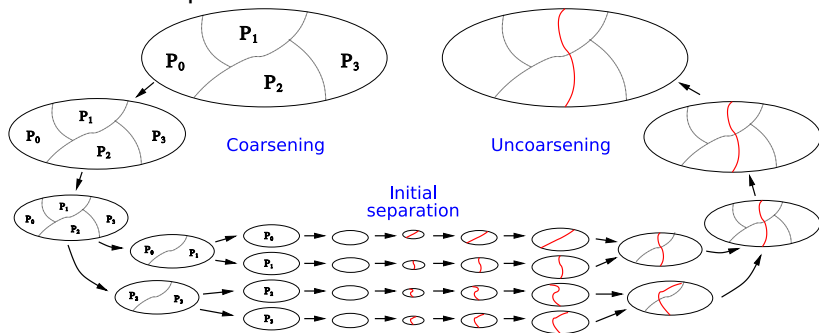
Folding with duplication allows us to improve data locality and speed-up the asynchronous distributed matching of graph vertices, by reducing further the number of distant neighbors

When uncoarsening, only the best of the two partitions is kept and forwarded to the finer level

⇒ Improves partition quality through multiple tries

Parallelization of the Multi-level algorithm (2)

Outline of the process:



Currently, we use folding with duplication at every stage

Parallelization of refinement

During the multi-level uncoarsening phase, experiments show that only vertices located on a small band around the separator may actually be moved during the local refinement process

By explicitly pre-computing, before each refinement step, a distributed band graph to which optimization algorithms will be applied only, we have been able to reduce dramatically problem size for refinement heuristics

(Chevalier and Pellegrini, EuroPar 2006)

Parallelization of refinement (2)

Surprisingly, better partition quality was achieved ($\approx -15\%$ in OPC on average)

We think it is because local optimization algorithms cannot be trapped in purely local optima but must comply with the “global picture” computed on the coarser graphs

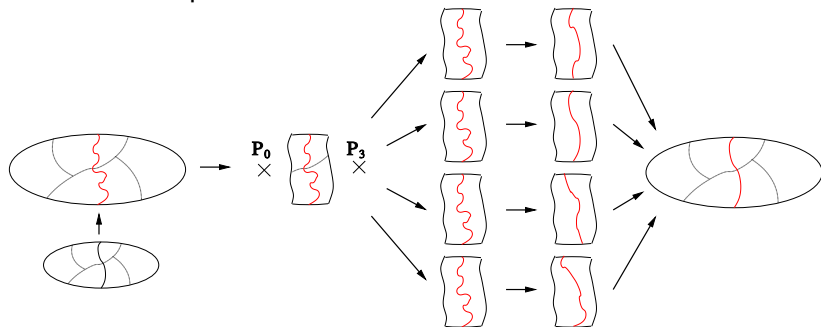
→ Multi-sequential refinement is possible when centralized band graphs fit into the memory of a single node

→ For 3D graphs up to a billion vertices, it is theoretically possible to apply multi-sequential F.M. on the band graphs

→ Costly but highly scalable algorithms such as Genetic Algorithms can also be used at the highest levels of uncoarsening

Parallelization of refinement(3)

Outline of the process:



Currently, we use multi-sequential F.M. refinement on band graphs without Genetic Algorithms

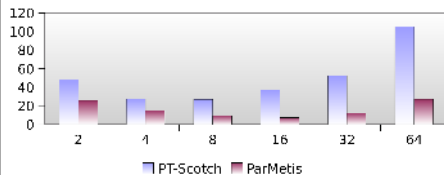
Outline

- 1 Introduction
 - Graph Partitioning
 - Use of Graph Partitioning
 - The Multi-Level Framework
- 2 Algorithms for efficient parallel reordering
 - Nested Dissection Parallelism
 - Parallelization of the Multi-level algorithm
 - Parallelization of refinement
- 3 Results
 - Time Results
 - Quality Results
- 4 Conclusion

Time Results (1)

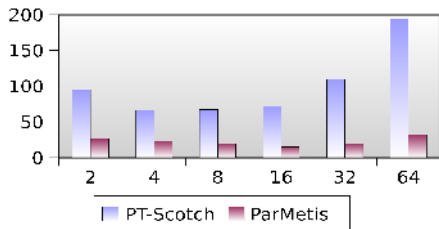
Tests have been run on a eight dual-core opteron computer

Running times for Conesphere1m



Conesphere1m : CEA graph with 10^6 vertices and 8×10^6 edges

Running times for Audikw1



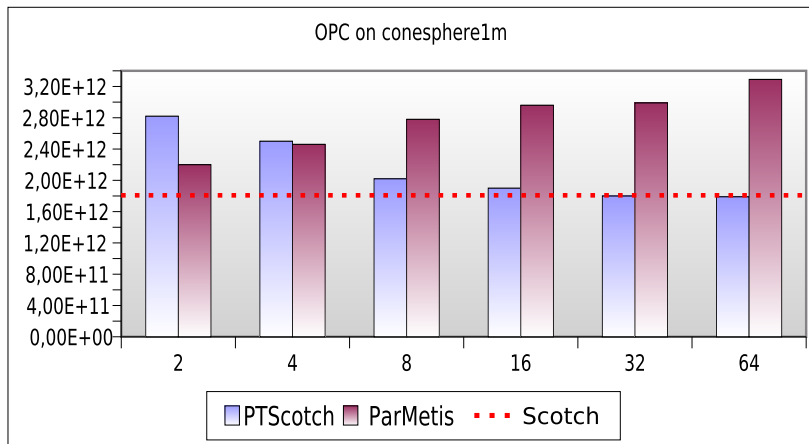
Audikw1 : 0.94×10^6 vertices and 38×10^6 edges

Time Results (2)

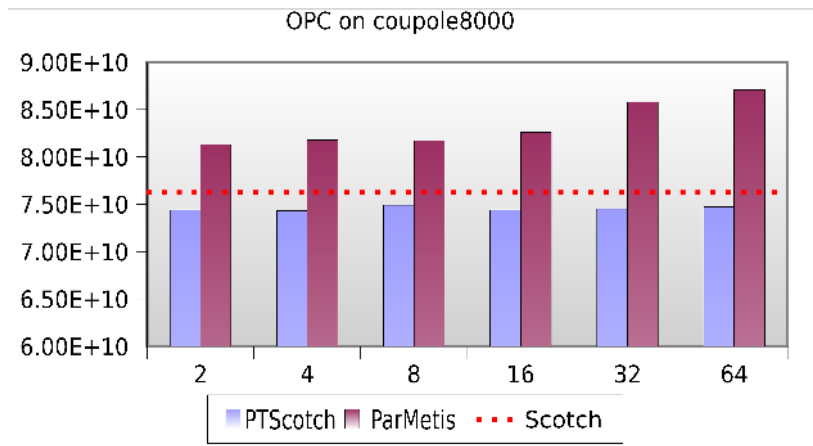
Scalability in time is not good when graphs have too few vertices, because the cost of the coarsening algorithm dramatically increases along with the number of distant neighbors.

...But in the above tests, above 16 processes, we sequentialize duplicate folded computations that were supposed to be run in parallel...

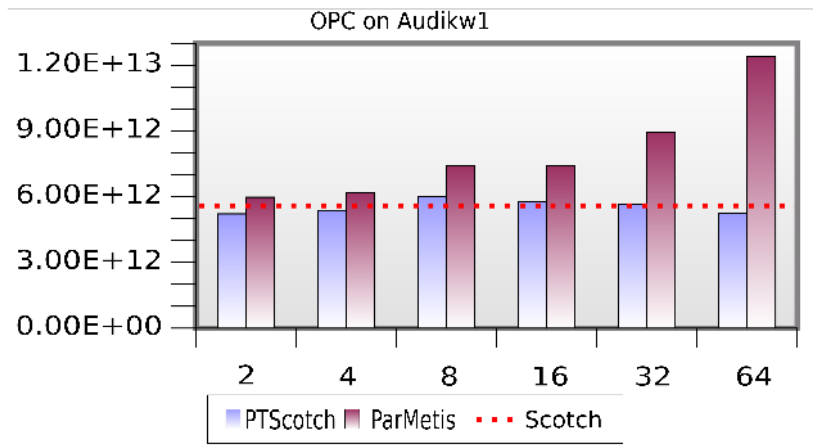
Quality Results (1)



Quality Results (2)



Quality Results (3)



Quality Results (4)

The increase of quality along with the number of processors can be explained by the increase of the searched space due to concurrency in :

- The multi-level algorithm: the best separator for a level is chosen among the two yielded by the coarser level
- The multi-sequential refinement: for any pair of finer and coarser graphs, the finer separator is the best among all band graphs built around the projection of the coarser separator, refined by sequential F.M.

Outline

- 1 Introduction
 - Graph Partitioning
 - Use of Graph Partitioning
 - The Multi-Level Framework
- 2 Algorithms for efficient parallel reordering
 - Nested Dissection Parallelism
 - Parallelization of the Multi-level algorithm
 - Parallelization of refinement
- 3 Results
 - Time Results
 - Quality Results
- 4 Conclusion

Conclusion

- High quality parallel ordering can be achieved by parallelizing three key algorithms :
 - 1 Nested Dissection
 - 2 Multi-Level
 - 3 Separator refinement
- First results are very satisfactory in terms of quality, but the scalability of the matching algorithm must be improved significantly

Conclusion

- High quality parallel ordering can be achieved by parallelizing three key algorithms :
 - 1 Nested Dissection
 - 2 Multi-Level
 - 3 Separator refinement
- First results are very satisfactory in terms of quality, but the scalability of the matching algorithm must be improved significantly

Work in progress

- Optimize coarsening by implementing asynchronous multi-buffered matching
(will be available very soon)
- Parallelize refinement over the band when band graphs no longer fit into memory
→ Build a fully parallel Genetic Algorithm to be used on band graphs when these latter cannot fit in the memory of a single node
- Design efficient and scalable k-way graph partitioning algorithms

First public release of PT-SCOTCH (ordering only) planned for November 2006



<http://www.labri.fr/~pelegrin/scotch>
<http://gforge.inria.fr/projects/scotch>

Any questions ?