Parallel repartitioning and remapping in Scotch

Sébastien Fourestier
François Pellegrini

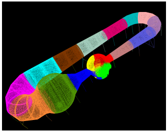
November 21, 2012 – Joint laboratory workshop

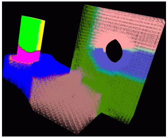
Table of contents

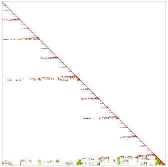
1
Parallel repartitioning

The Scotch project

- Toolbox of graph partitioning methods, which can be used in numerous contexts
- Sequential Scotch library (v6.0)
  - Graph and mesh partitioning
  - Static mapping (edge dilation)
  - Graph and mesh reordering
  - Clustering
  - Graph repartitioning, *remapping*
- Parallel PT-Scotch library (v6.1)
  - Graph partitioning (edge)
  - Static mapping (edge dilation)
  - Graph reordering
  - *Graph repartitioning, remapping*

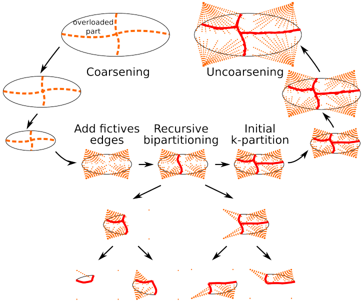




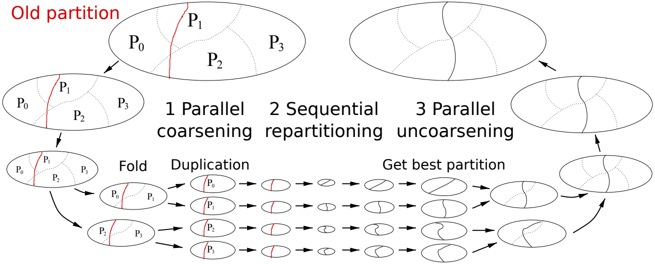


Sequential repartitioning: big picture

- Repartitioning problem
  - Improve *cut* and *balance*
  - Minimizing *migration*
- Multilevel framework for sequential repartitioning
  - Coarsening mates only vertices belonging to the same part
  - Initial repartitioning by recursive bipartitioning
  - K-way refinement



Parallel repartitioning: big picture

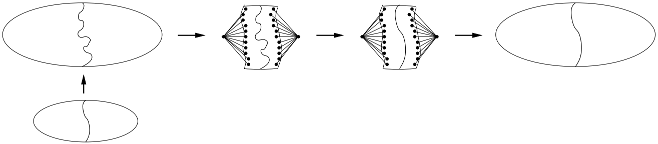


- Parallel multilevel framework for repartitioning
  - Parallel coarsening with fold and duplication
  - Initial repartitioning by multi-sequential *k*-way partitioning
  - Parallel *k*-way refinement

Uncoarsening: parallel k-way refinement

1. K-way Fiduccia-Mattheyses heuristic
   - Computes good partitions while preserving a specified load balance
   - Performs only local optimizations
   - Inherently iterative $\rightarrow$ does not parallelize well
2. Global diffusion-based heuristic
   - Global, scalable and easily parallelizable
   - More expensive, a band graph must be extracted
   - The load balance tolerance cannot be chosen ($\approx$ 5%)

Multi-centralized band graphs



- For first uncoarsening levels ($|V_b| < 10\ 000$), we centralize the band graph to use both Fiduccia-Mattheyses and diffusion-based heuristic
- After we use only the diffusion-based heuristic

Experimental setup

- Initial partitioning
  - 128 parts
  - Vertex loads are equal to 1
  - Balance constraint of 0.05
- We increase by 1 the weights of the vertices that are in the first 32 parts $\rightarrow$ imbalance of ≈ 0.16.
- Various strategies
- Migration cost from 0.1 to 50
  $\rightarrow$ 140 runs for each graph

Test graphs

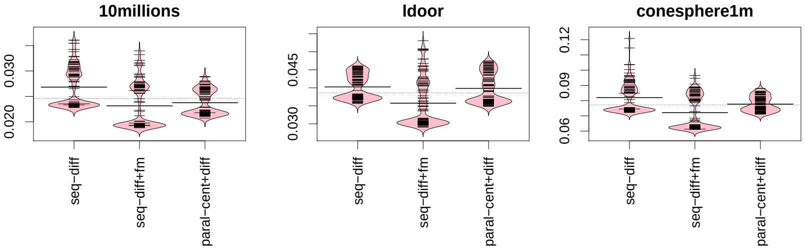
Graph	Description	Size ($\times 10^3$)		Average degree
		$\lvert V \rvert$	$\lvert E \rvert$	
`10millions`	3D electromagnetics	10 423	78 649	15.09
`conesphere1m`	3D electromagnetics	1 055	8 023	15.21
`ldoor`	structural problem	952	22 785	47.86

- Size between 1 and 10 millions of vertices
- Average degree ranging from 15 to 47
- `10millions`: the biggest number of vertices
- `ldoor`: the highest average degree

Repartitioning strategies

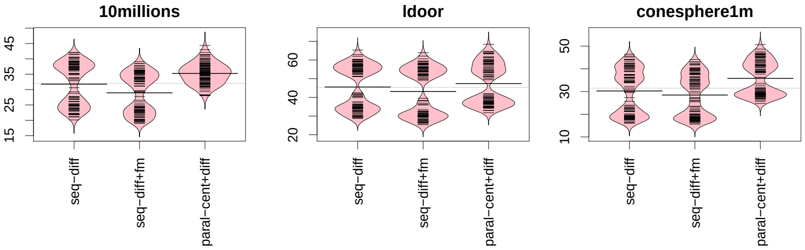
- `seq-diff`
  - Sequential strategy
  - Initial partition: *Recursive bipartitioning*
  - Refinement: *Diffusion*
- `seq-diff+fm` (Scotch default sequential strategy)
  - Sequential strategy
  - Initial partition: *Recursive bipartitioning*
  - Refinement: *Diffusion* + *Fiduccia-Mattheyses*
- `paral-cent+diff`
  - Parallel strategy on 32 cores
  - Initial partition: *Sequential recursive bipartitioning*
  - Multi-centralized refinement: *Diffusion* + *Fid.-Matt.*
  - Parallel refinement: *Diffusion*

Cut



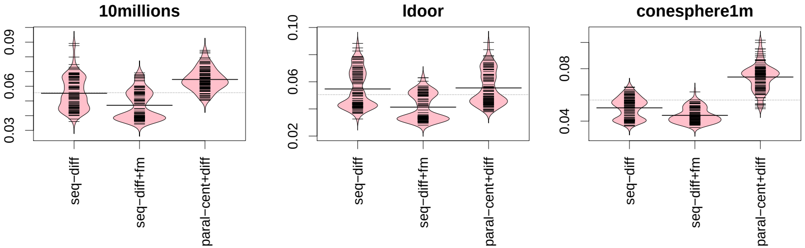
- paral-cent+diff is 7.8 % worse than seq-diff+fm
- paral-cent+diff is 5 % better than seq-diff

Migration



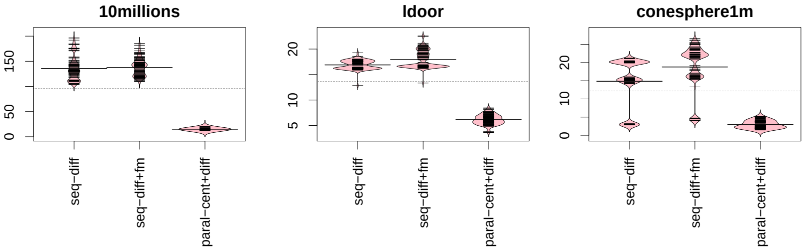
- `paral-cent+diff` strategy migrates a little more
- Multi-centralisation yields partitions with more migration but best cut

Imbalance



- For `conesphere1m`, `paral-cent+diff` is 1.47 times more imbalanced than `seq-diff`
- For other graphs, `paral-cent+diff` is close to `seq-diff`

Time (s)



- In mean, `paral-cent+diff` is 6.96 times cheaper than `seq-diff`
- It is 7.24 times cheaper than `seq-diff+fm`

Summary of experimental results

- `paral-cent+diff` brings a cut 5 % better than `seq-diff`
- It migrates a little more
- It brings a worse imbalance
  - We are currently checking which differences between the sequential and the parallel implementation impact imbalance
- On average, it is 7 times less expensive on 32 cores.

2
Shared-memory parallel algorithms

Why invest in shared-memory parallelism

- Most users now have multi- or many-core machines
  - From laptops to high-end supercomputers
- Shared-memory parallelism is almost always less expensive than explicit message passing parallelism
  - No need to allocate and fill user-managed communication buffers
- Two applications of shared-memory parallelism
  - Reduce number of MPI processes up to one per node for PT-Scotch
  - Use threads transparently for the (no longer) “sequential” Scotch

Implementation details

- We use Posix Pthreads
  - Already used in other routines of Scotch
  - Allowed us to implement a framework of primitives:
    - Barrier, reduction, scan, join, etc.
- Limitations as of version `6.0`
  - Number of threads set up at compile time
  - Thread allocation performed by increasing core numbers
    - May not always reflect real core and memory affinity
    - Will use `hwloc` in next release to ensure it

Algorithms at stake

- We focused on the most expensive algorithms
- Matching and coarsening
  - Involves all graph vertices
  - Expensive at the highest levels of the multilevel process
- The diffusion method
  - Involves band graph vertices only
  - Expensive because of floating-point computations and number of passes to perform

Diffusion heuristic

- Almost embarrassingly parallel
- Synchronization after each iteration
- Deterministic results whatever the number of threads is

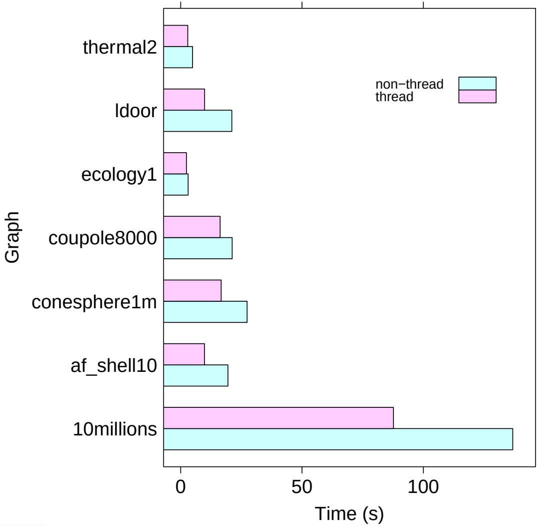
- Experimental setup
  - Partitioning into 128 parts
  - Use the `seq-diff+fm` strategy
  - Use 8 threads

Test graphs

Graph	Description	Size ($\times 10^3$)		Average degree
		$\|V\|$	$\|E\|$	
10millions	3D electromagnetics	10 423	78 649	15.09
af_shell10	structural problem	1 508	25 582	33.93
conesphere1m	3D electromagnetics	1 055	8 023	15.21
coupole8000	3D structural mechanics	1 768	41 656	47.12
ecology1	2D/3D problem	1 000	1 998	4.00
ldoor	structural problem	952	22 785	47.86
thermal2	thermal problem	1 228	3 676	5.99

- Graphs from various domains
- Size between 1 and 10 millions of vertices
- Average degree ranging from 4 to 47

Run time (including non-threaded routines)



- Average gain 37.78 %

3
Remapping

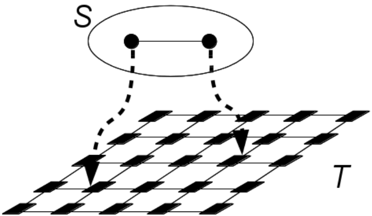
Static mapping

- 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

$$|\rho_{S,T}(e_S)| \text{ : Path load in } T$$

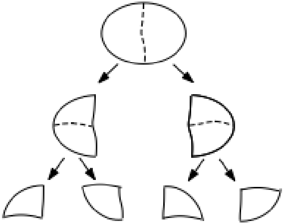
$$\text{Dilation: } \sum_{e_S \in E(S)} w(e_S)|\rho_{S,T}(e_S)|$$

- Static mapping features are already present in the sequential Scotch library

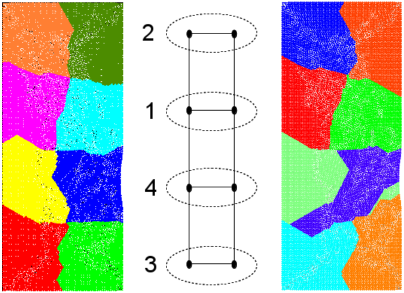


Parallel static mapping and "twists"

- Recursive bi-mapping cannot be transposed 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"

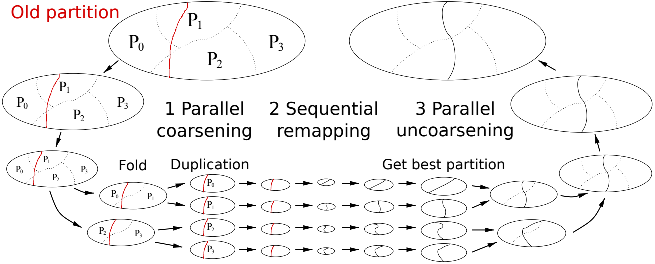


- Sequential processing only!



Sequential and parallel dynamic remapping

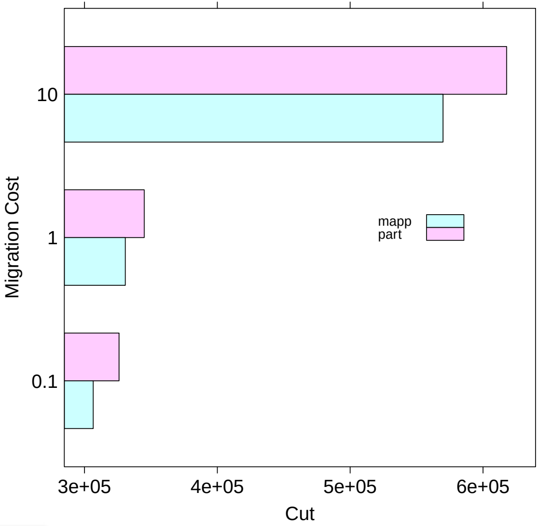
- Take advantage of the k-way multilevel framework
  - Initial mapping is computed sequentially (no twists !)
  - Take dilation into account during *k*-way sequential or parallel refinement
  - Contribution to improve diffusion heuristic to handle dilation



Experimental setup

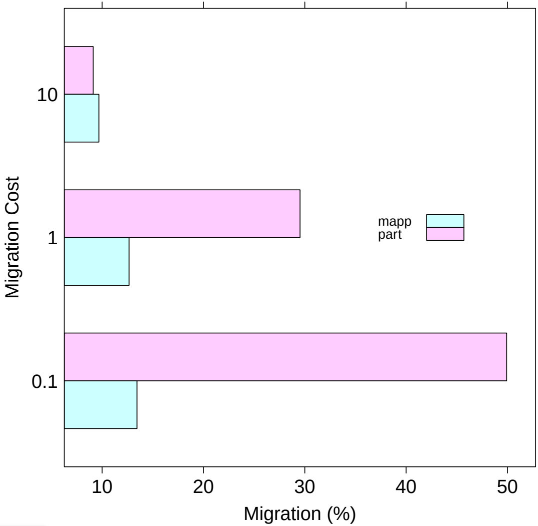
- Initial mapping
  - 3D torus: $2 \times 2 \times 2$ (8 processors)
  - Vertex loads are equal to 1
  - Balance constraint of 0.05
- We increase by 1 the weights of the vertices that are in the first 2 processors $\rightarrow$ imbalance of ≈ 0.16.
- Graph: `10millions`
- Migration cost: 0.1, 1 and 10

Cut



- Improve the cut (comprising dilation) by 6 %

Migration number (%)



- Need more work to be as sensible to migration cost as repartitioning

4
Prospects

Prospects

- On going work
  - Run more experiments to improve
    - Sequential remapping
    - Parallel repartitioning
    - Parallel remapping
  - Integrate shared-memory improvements to PT-Scotch
- On going collaboration
  - Load balancing within CHARM++
- Potential collaborations
  - Load balancing within MPI
  - Evaluation of remapping on real applications

Thanks