Solvers and partitioners in the Bacchus project

11/06/2009

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche BORDEAUX - SUD-OUEST

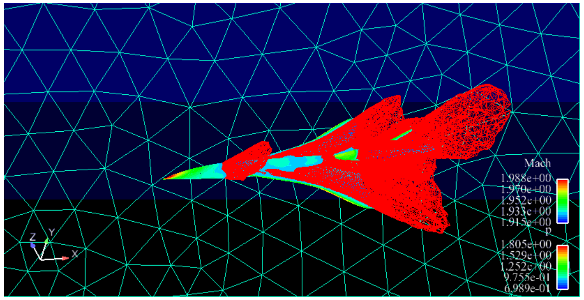
François Pellegrini

Financé par Agence Nationale de la Recherche GIP ANR

INRIA-UIUC joint laboratory

The Bacchus team

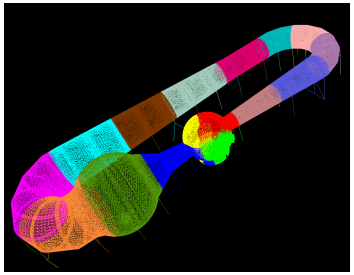
- Purpose
  - Develop and validate numerical methods and tools adapted to problems modeled by PDEs of hyberbolic type
    - Fluid dynamics, aeroacoustics, geophysics MHD, …
- Mixed CS / NA team
  - Head: Rémi Abgrall
  - 7 staff, 10+ interns/PhD/PostDocs

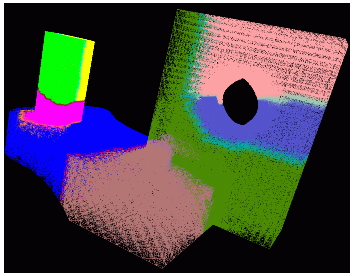


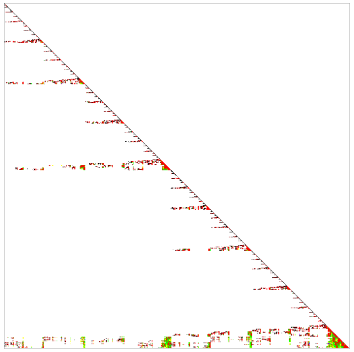
- Tools
  - Simulation platform (FluidBox), Mesher (MMG3D), Solvers (PaStiX, HIPS), Partitioner (Scotch), ...

Features of Scotch (1)

- Toolbox of graph partitioning methods, which can be used in numerous contexts
- Sequential Scotch library
  - Graph partitioning (edge or vertex)
  - Mesh partitioning (elements)
  - Static mapping (edge dilation)
  - Graph reordering
  - Mesh reordering
- Parallel PT-Scotch library
  - Graph partitioning (edge)
  - Static mapping (edge dilation) [prototype]
  - Graph reordering







Features of Scotch (2)

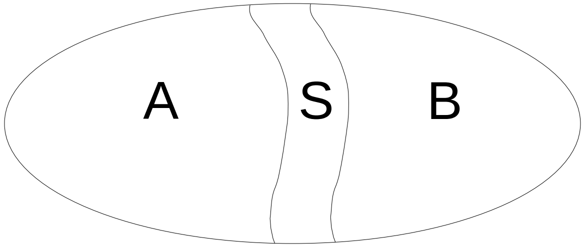
- Usable by means of library function calls or through command-line programs
  - Can be called from C or FORTRAN
  - Reentrant routines usable in a multi-threaded context
- Support of adaptive graphs and meshes
  - Discontinuous data indexing to enable adding vertices
- Software developed in ANSI C
  - MPI for message-passing, optional use of pthreads
- Dynamic parametrization of partitioning methods by means of strategy strings (feature or punishment ? ;-))
- Version 5.1 available under CeCILL-C free software license

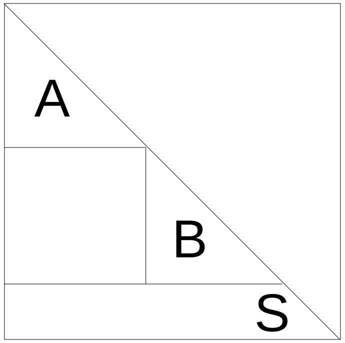
The current Scotch roadmap

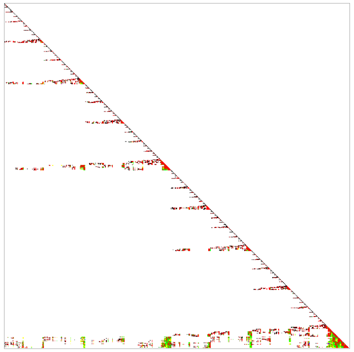
- Devise robust parallel graph partitioning methods
  - Should handle graphs of more than a billion vertices distributed across one thousand processors
- Improve sequential graph partitioning methods if possible
  - Fiduccia-Mattheyses-like local optimization algorithms are both fast and efficient on a very large class of graphs but are intrinsically sequential
- Investigate alternate graph models (meshes/hyper-graphs)

Nested dissection

- 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

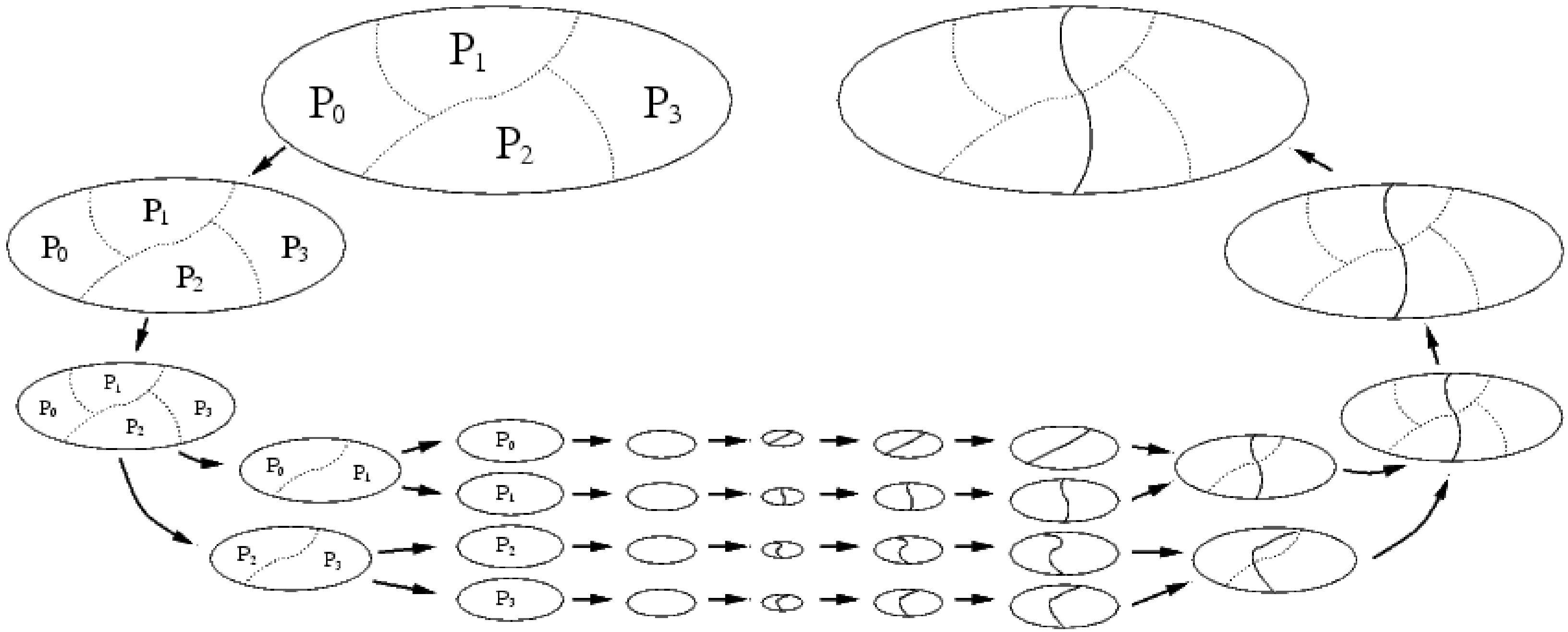






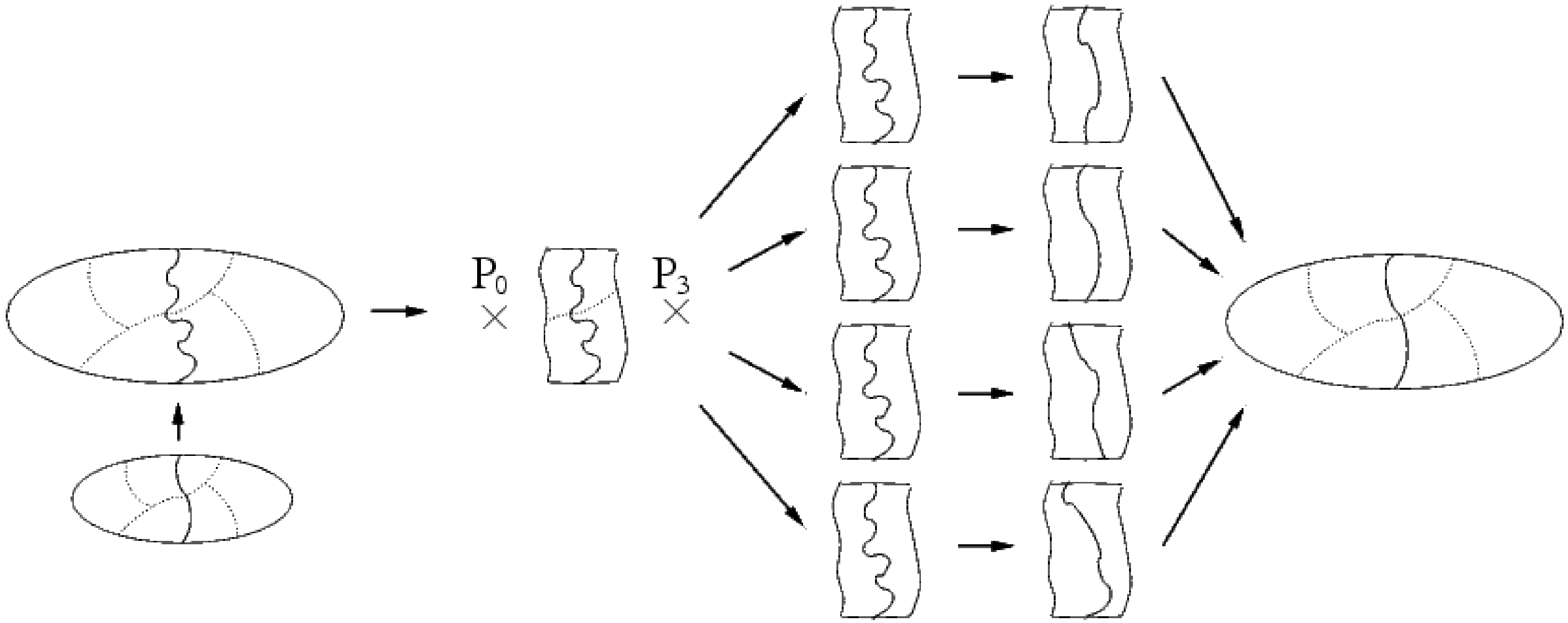
Parallel multi-level framework

- Performs folding and duplication when not enough vertices per processor
  - Allows for multi-sequential exploration of problem space



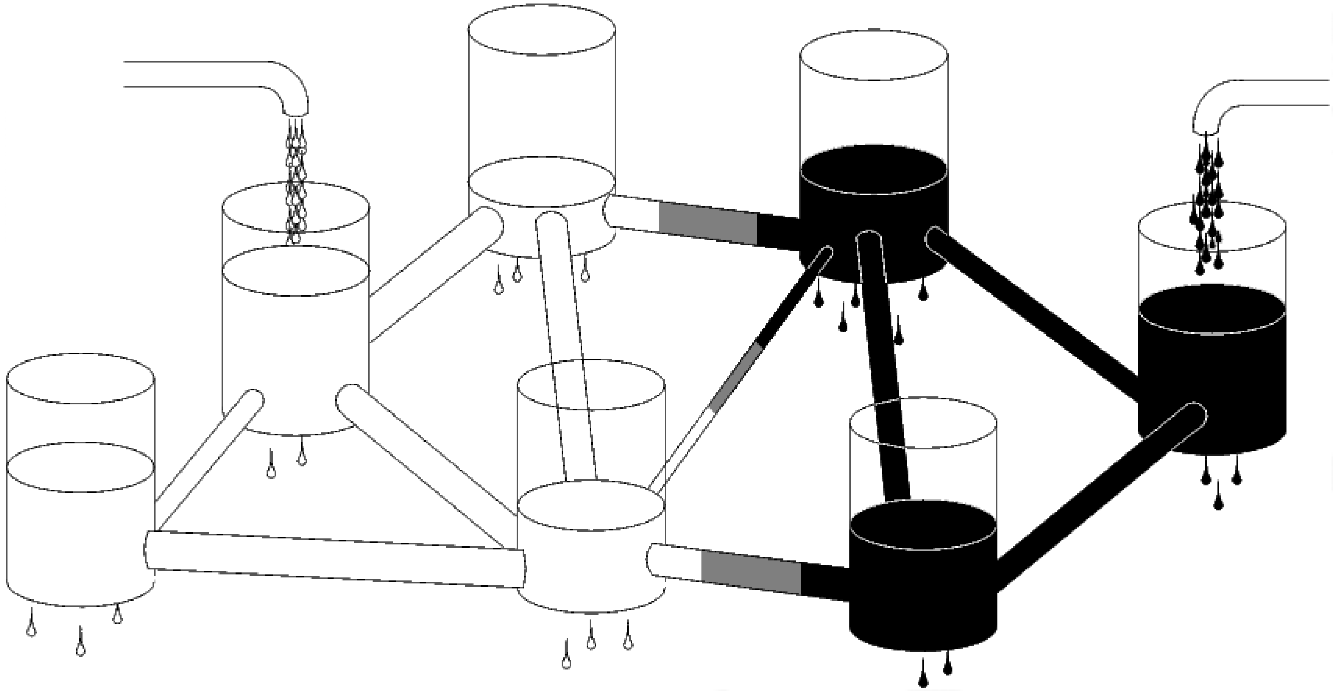
Parallelization of the refinement phase (2)

- Parallel algorithms can also be used
  - Genetic algorithms
  - Diffusion algorithms



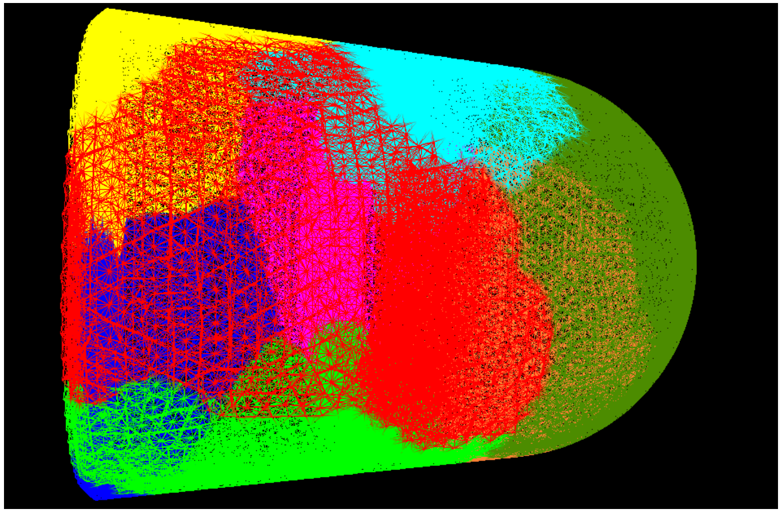
Jug of the Danaides (1)

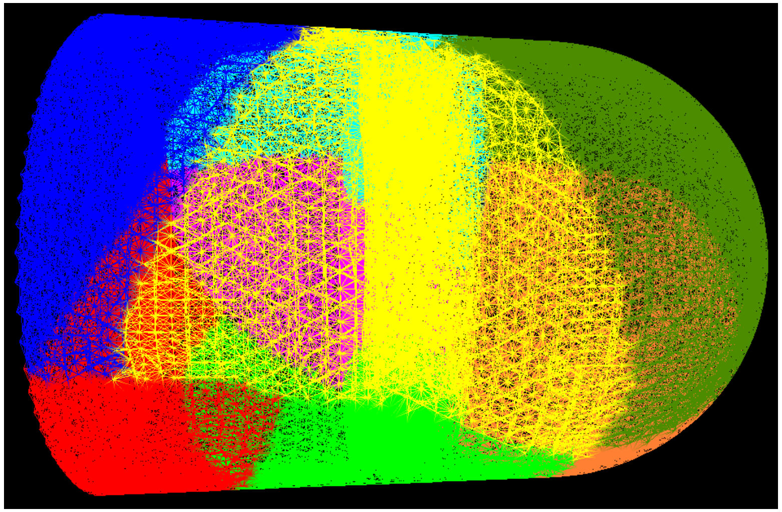
- Sketch of the algorithm



Jug of the Danaides (1)

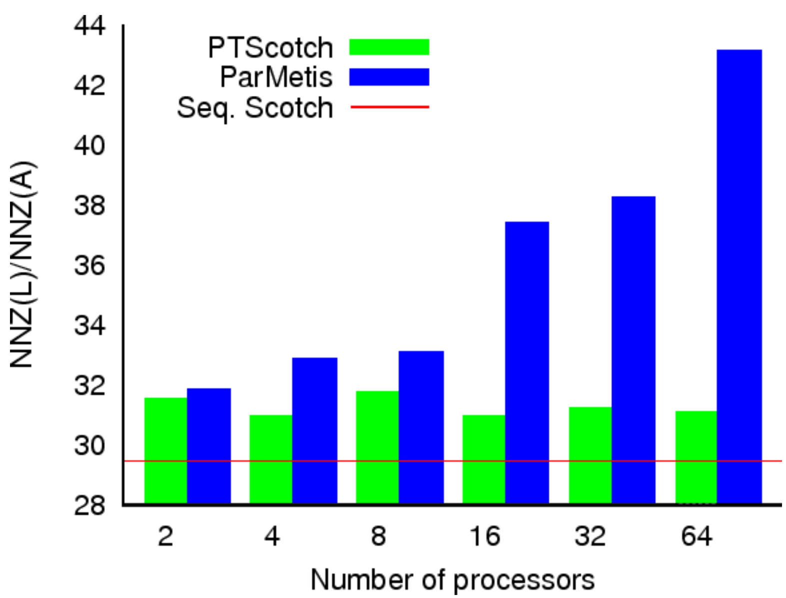
- Using Jug of the Danaides as the optimization algorithm in the multi-level process :
  - Smoothes interfaces
  - Is slower than sequential FM (20 times for 500 iterations, but only 3 times for 40 iterations)

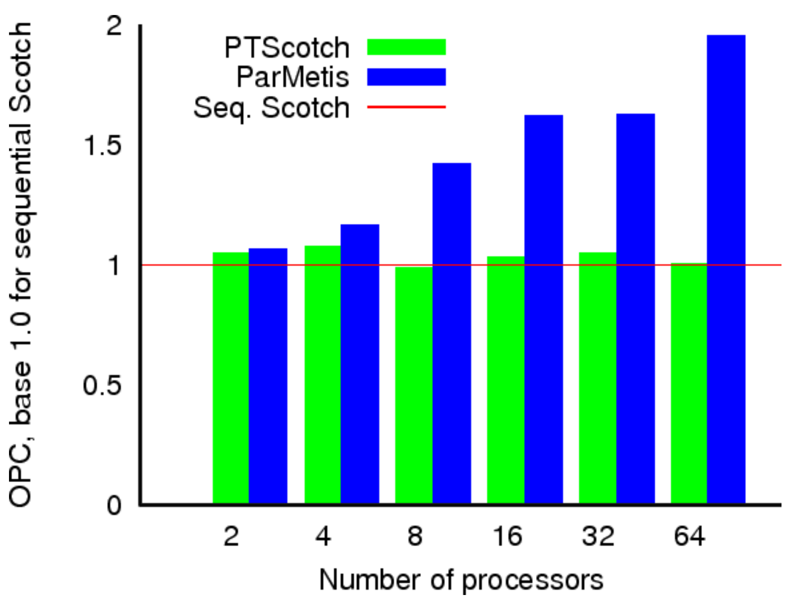




Results for parallel ordering (1)

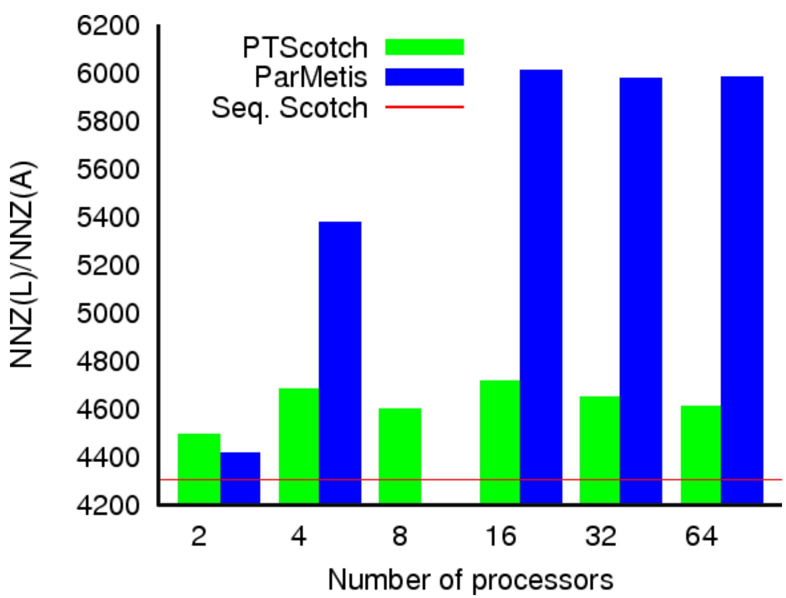
Test case	Number of processes					
	2	4	8	16	32	64
audikw1						
O_{PTS}	**5.73E+12**	**5.65E+12**	**5.54E+12**	**5.45E+12**	**5.45E+12**	**5.45E+12**
O_{PM}	5.82E+12	6.37E+12	7.78E+12	8.88E+12	8.91E+12	1.07E+13
t_{PTS}	73.11	53.19	45.19	33.83	24.74	18.16
t_{PM}	32.69	23.09	17.15	9.80	5.65	3.82

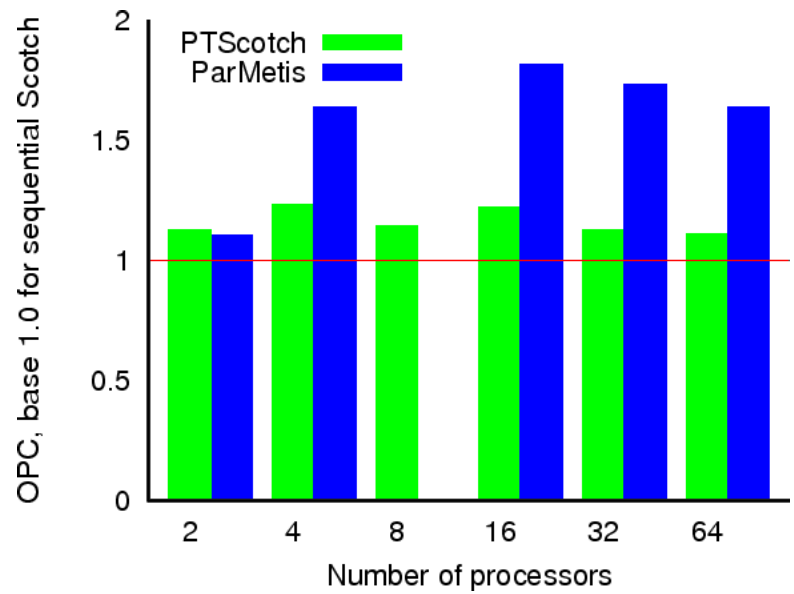




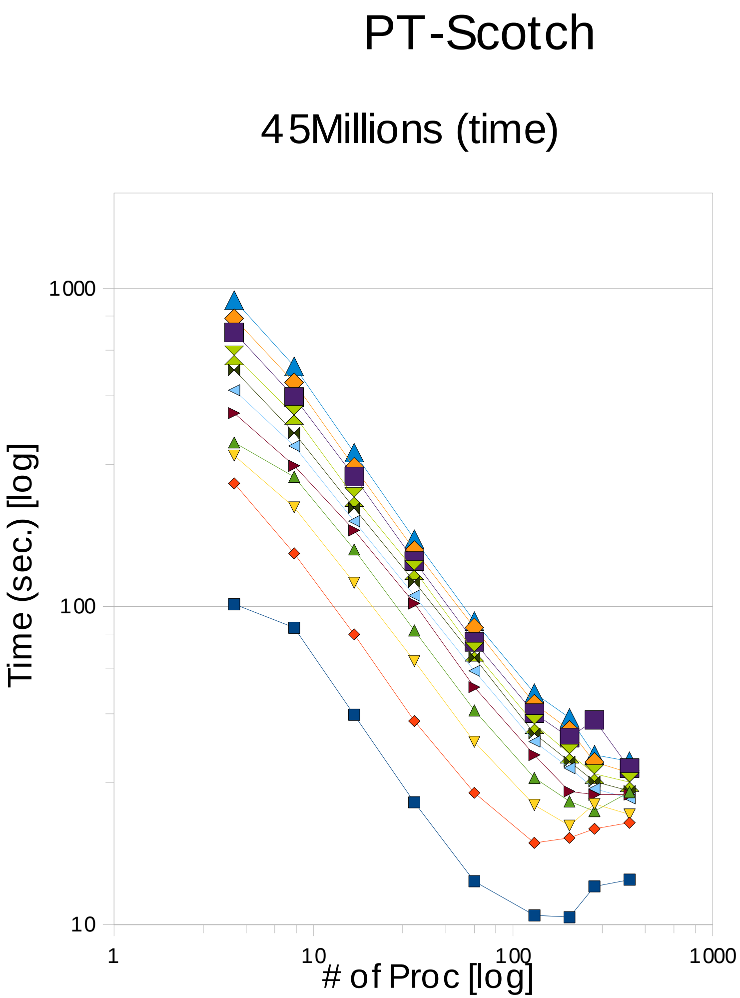
Results for parallel ordering (2)

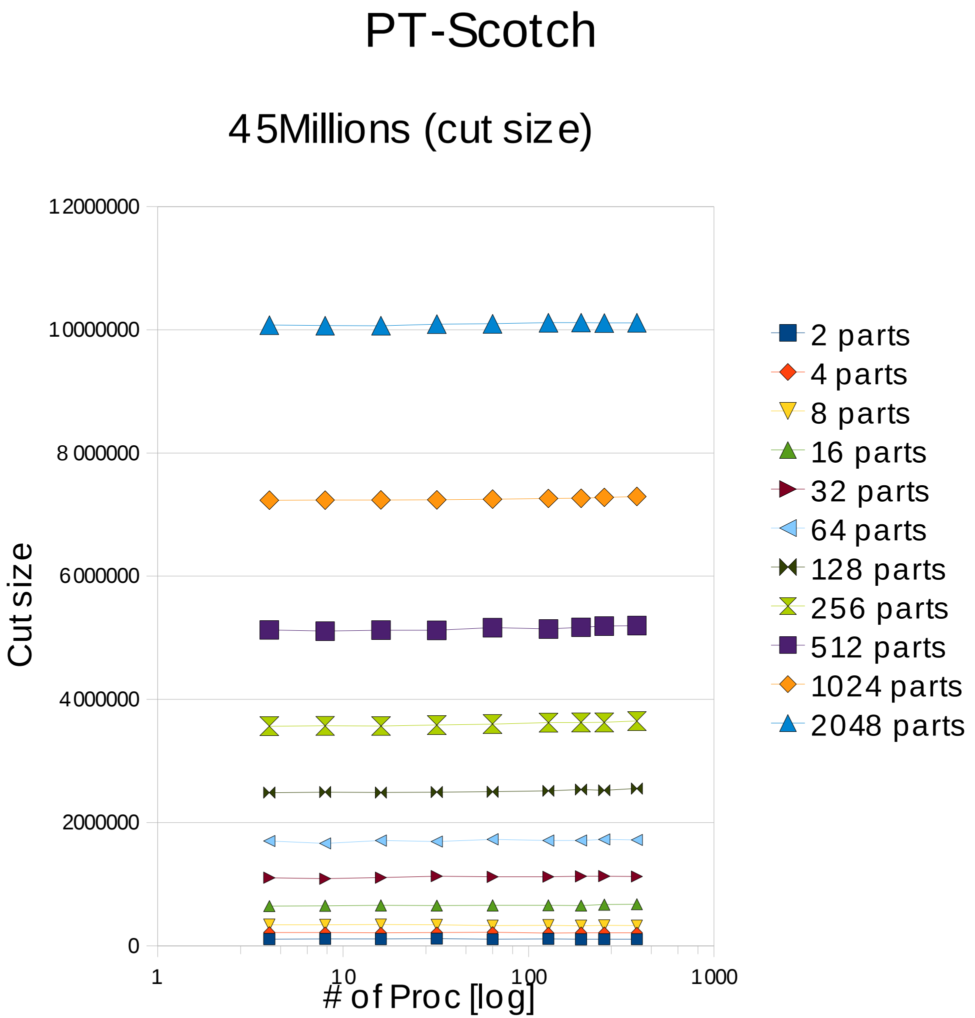
Test case	Number of processes					
	2	4	8	16	32	64
cage15						
O_{PTS}	4.58E+16	**5.01E+16**	**4.64E+16**	**4.94E+16**	**4.58E+16**	**4.50E+16**
O_{PM}	**4.47E+16**	6.64E+16	†	7.36E+16	7.03E+16	6.64E+16
t_{PTS}	540.46	427.38	371.70	340.78	351.38	380.69
t_{PM}	195.93	117.77	†	40.30	22.56	17.83



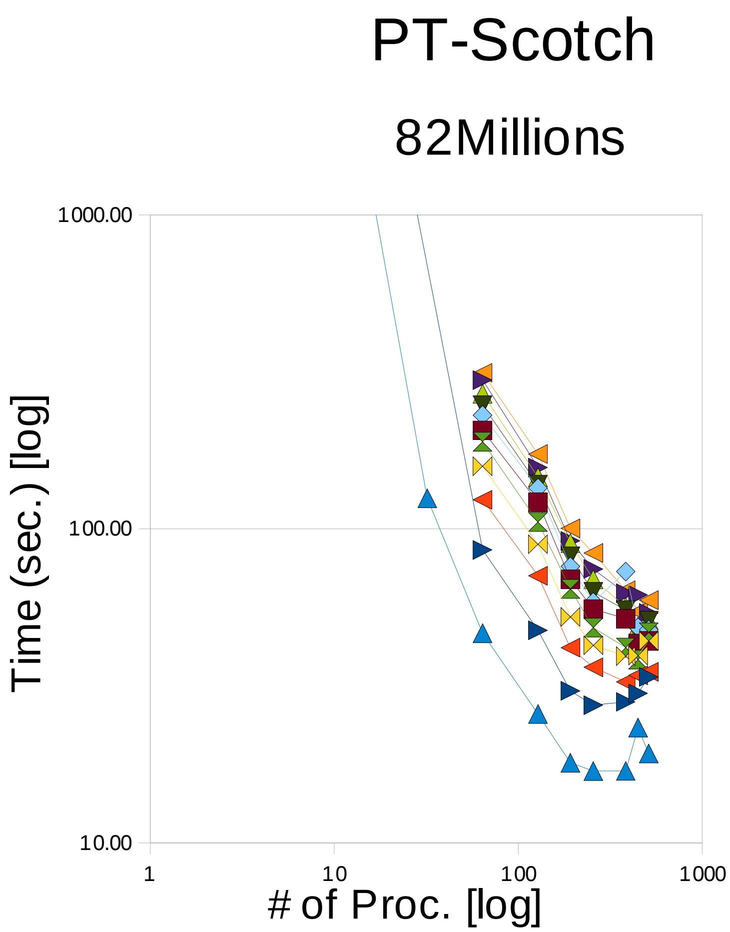


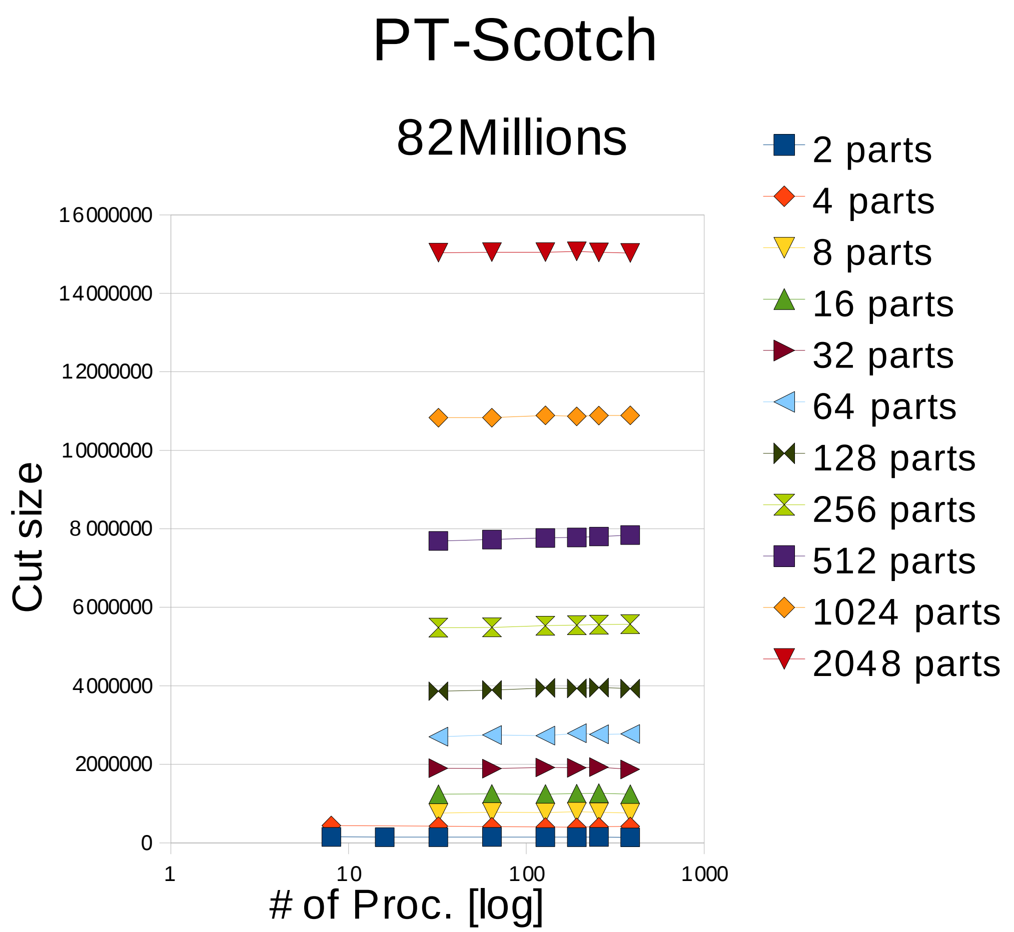
Results for parallel partitioning (1)





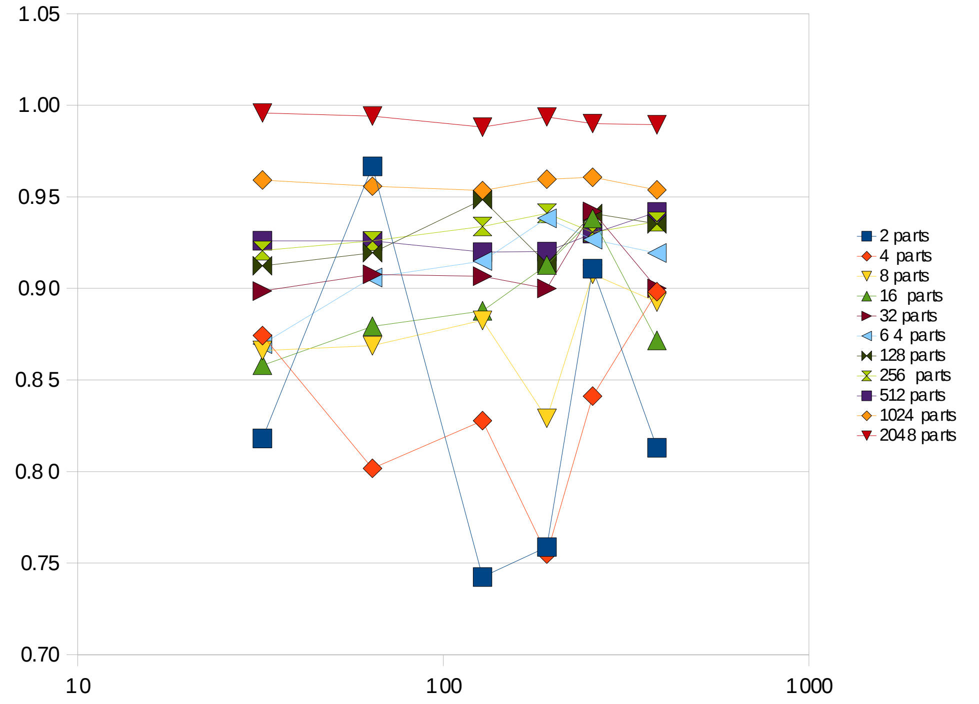
Results for parallel partitioning (2)





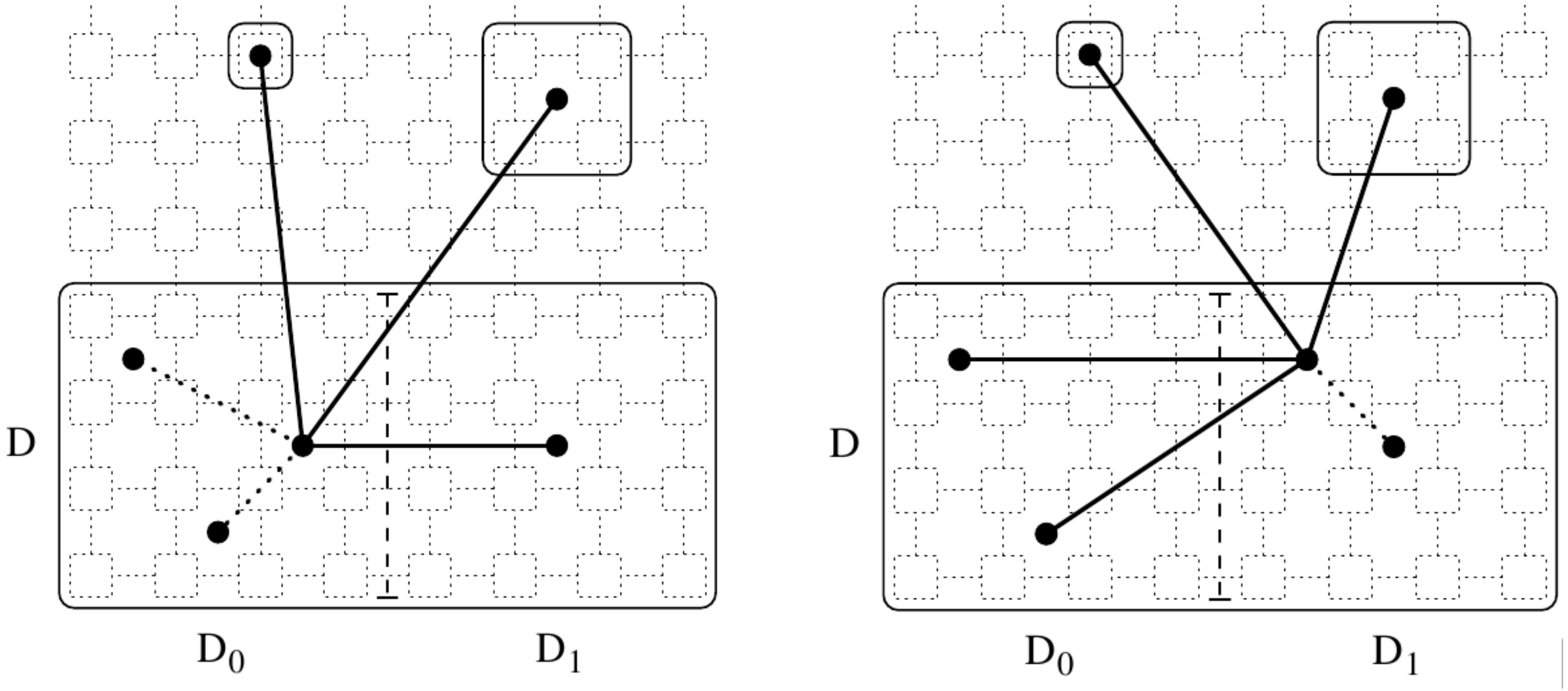
Results for parallel partitioning (3)

- Cut size ratio most often in favor of PT-Scotch vs. ParMeTiS up to 2048 parts
  - Gets worse when number of parts increases as direct k-way is better than recursive bisection
  - Partition quality of ParMeTiS is irregular for small numbers of parts



Static mapping vs. plain partitioning

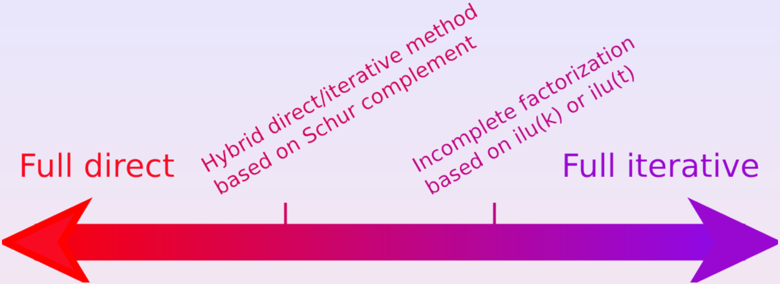
- Brings gains up to 20 % on solving time on “regular” multi-core architectures, and even more for really heterogeneous clusters



In the future ? Go dynamic !

- Next steps
  - Parallel static mapping (almost done)
  - Dynamic repartitioning on heterogeneous architectures [PhD of Sébastien Fourestier]
  - Parallel hyper graph partitioning ?
    - Only if gains can be expected over existing works
- Move upwards to application mesh models
- Parallel adaptive remeshing [work with C. Dobrzynski]
  - Take into account the numerical stability of the problem being studied
  - Take advantage of the work done in PT-Scotch on distributed adaptive graphs

Spectrum of algebraic linear solvers



The “spectrum” of linear algebra solvers

Direct:

- Robust/accurate for general problems
- BLAS-3 based implementation
- Memory/CPU prohibitive for large 3D problems
- Limited parallel scalability

Iterative:

- Problem dependent efficiency/controlled accuracy
- Only mat-vec required, fine grain computation
- Less memory usage, possible trade-off with CPU
- Attractive ”built-in” parallel features

MURGE : a common API to the sparse linear solvers of BACCHUS

http://murge.gforge.inria.fr

Features

- Through one interface, one can access to many solver strategies.
- One can enter a graph/matrix in a centralized or distributed way.
- Simple formats : coordinate, CSR or CSC.
- Very easy to implement an assembly phase using MURGE.
- MURGE proposes Fortran and C prototypes.

General structure of the code

```
MURGE_Initialize(idnbr, ierror)
MURGE_SetDefaultOptions(id, MURGE_ITERATIVE)  /* Choose general strategy */
MURGE_SetOptionInt(id, MURGE_DOF, 3)  /* Set degrees of freedom */
..
MURGE_Graph_XX(id..) /* Enter the graph : several possibilities */
DO
 MURGE_SetOptionReal(id, MURGE_DROPTOL1, 0.001) /* Threshold for ILUT */
 MURGE_SetOptionReal(id, MURGE_PREC, 1e-7)  /* Precision of solution */
 ...
 /** Enter new coefficient for the matrix **/
  MURGE_AssemblyXX(id..) /* Enter the matrix coefficients */
 DO
   MURGE_SetRHS(id, rhs) /* Set the RHS */
   MURGE_GetSol(id, x)   /* Get the solution */
 END
  MURGE_MatrixReset(id) /* Reset matrix coefficients */
END
MURGE_Clean(id)  /* Clean-up for system "id" */
MURGE_Finalize() /* Clean-up all remaining structure */
```

PaStiX Features

- LLt, LDLt, LU factorization with supernodal implementation
- Static pivoting + Refinement: CG/GMRES
- 1D/2D block distribution + Full BLAS3
- Simple/Double precision + Float/Complex operations

- MPI/Threads implementation (SMP/Cluster/Multicore/NUMA)
- **Dynamic scheduling inside SMP nodes (static mapping)**
- Support external ordering library (PT-Scotch/METIS)

- Multiple RHS (direct factorization)
- **Incomplete factorization with ILU(k) preconditionner**
- Out-of Core implementation (in SMP mode only)

Dynamic Scheduling for NUMA and multicore architectures

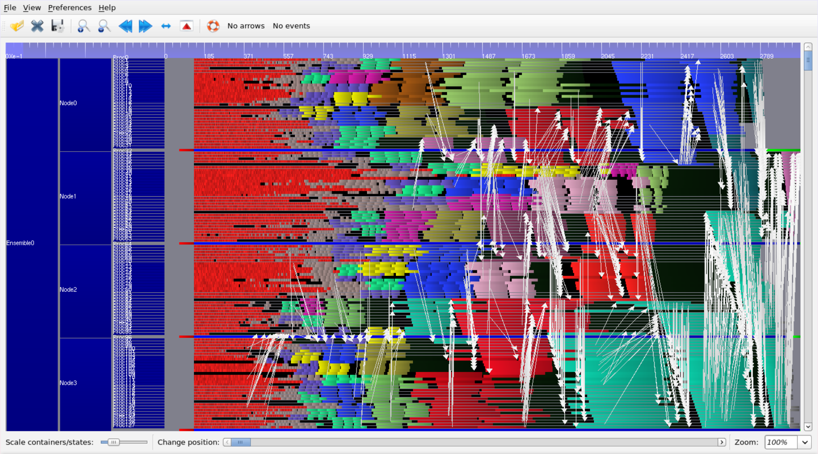
Needs

- Adapt to NUMA architectures
- Improve memory affinity (take care of memory hierarchy)
- Reduce idle-times due to I/O (communications and disk access in future works)
- Use dedicated threads for communications and disk access

Proposed solution

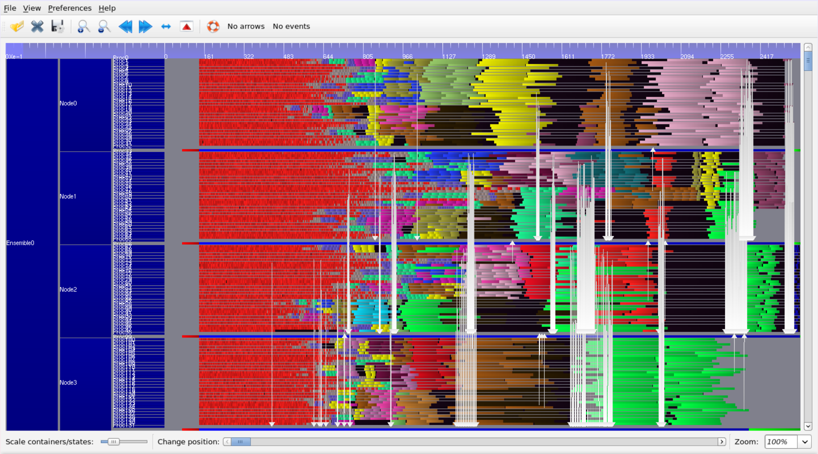
- Based on a classical work stealing algorithm
- Stealing is limited to preserve memory affinity
- Use dedicated threads for I/O and communication in order to give them an higher priority
- Suitable to GP-GPU programming model

Static Scheduling Gantt Diagram



- Each color gives the number of candidate processors for the task (level in the tree)
- *10Million* test case on IDRIS IBM Power6 with 4 MPI process of 32 threads

Dynamic Scheduling Gantt Diagram



- Reduces time by 10-15% on SMP cluster
- Better results are expected on NUMA clusters

Direct Solver Highlights

Main users

- Electomagnetism and structural mechanics at CEA-DAM-CESTA
- MHD Plasma instabilities for ITER at CEA-Cadarache
- Fluid mechanics at IMB Bordeaux

Highlights

The direct solver PaStiX has been successfully used by CEA/CESTA to solve a huge symmetric complex sparse linear system arising from a 3D electromagnetism code on the TERA-10 CEA supercomputer.

- **45 millions unknowns**: required 1.4 Petaflops and was completed in half an hour on 2048 processors.
- **83 millions unknowns**: required 5 Petaflops and was completed in 5 hours on 768 processors.

To our knowledge a system of this size and this kind has never been solved by a direct solver.

Block ILU(k): a supernode amalgamation algorithm for an efficient block Incomplete factorization

Derive a block incomplete LU factorization from the supernodal parallel direct solver

- Based on existing package PaStiX
- Level-3 BLAS incomplete factorization implementation
- Fill-in strategy based on level-fill among block structures identified thanks to the quotient graph
- Amalgation strategy to enlarge block size to improve BLAS-3 efficiency

Highlights

- Handles efficiently high level-of-fill
- Solving time can be 2-4 faster than with scalar ILU(k)
- Scalable parallel implementation

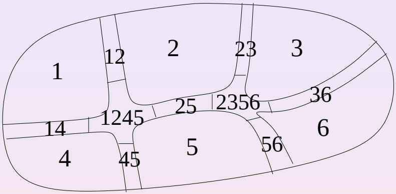
HIPS Features

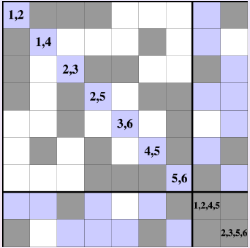
- LLt, LDLt, LU factorizations : supernodal implementation (BLAS-3).
- ILUCT, ICT : scalar column left-looking factorization.
- Full iterative or hybrid direct/iterative methods.
- Krylov method : CG/GMRES
- Simple/Double precision and Float/Complex operations

- Use external ordering and partitioning library : SCOTCH or METIS

- Requires only C + MPI
- Fortran interface
- Can use a domain decomposition given by the user

HIPS: domain interface based fill-in policy

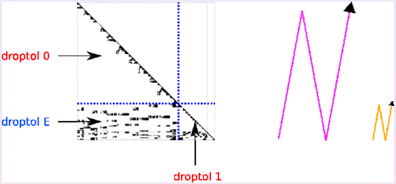


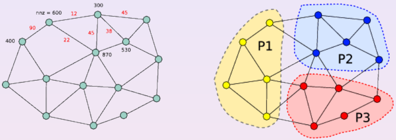


Robust block incomplete factorization of the Schur complement

- Hierachy of separators (wirebasket like - faces , edges, vertices)
- Block incomplete factorization with "geometrical" fill-in policy to express parallelism
  (Global factorization using only local sub-domain matrices)
- MIS ordering to express parallelism within incomplete factorisation steps

HIPS: preconditioners





Main features

- Iterative or “hybrid” direct/iterative method are implemented.
- Mix direct supernodal (BLAS-3) and sparse ILUT factorization in a seamless manner.
- Memory/Load balancing : distribute the domains on the processors (domains $>$ processors).

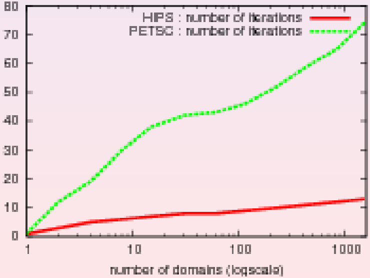
HIPS vs Additive Schwarz (from PETSc)

Experimental conditions

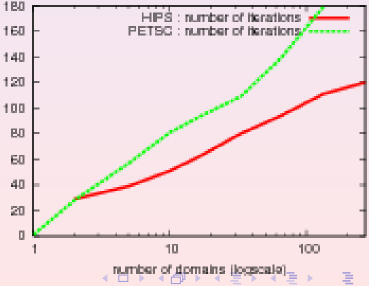
These curves compare HIPS (Hybrid) with Additive Schwarz from PETSc.
Comparison on the same domain decomposition (from SCOTCH)
Parameters were tuned to compare the result with a very similar fill-in
We set MUMPS as local direct solver in PETSc

Iterations

Haltere

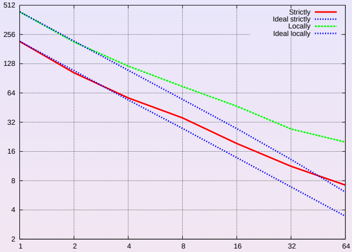


MHD

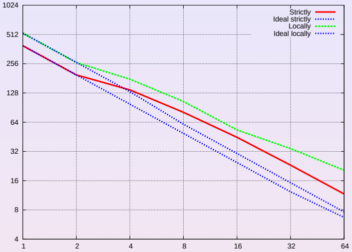


HIPS: Parallel time [strong] scalability

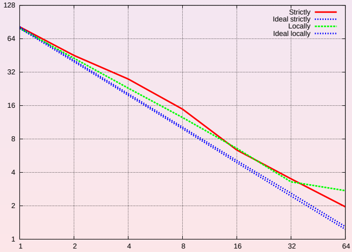
MHD1 (485, 597) : 64 domains



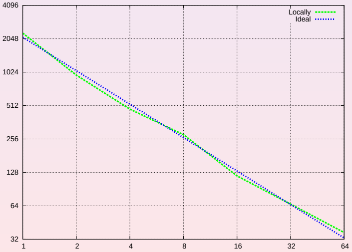
AUDI (943, 695) : 231 domains



HALTERE (1, 288, 825): 1062 domains



AMANDE (6, 994, 683): 2062 domains



Hybrid solver : Amande up to 2048 procs (Jade, CINES)

- Amandes : N=6, 994, 683 , NNZ=58, 477, 383
- Additive Schwarz, ILUT or ILUk failed
- 2053 domains of $\simeq$ 3770 nodes
- $(droptol_0; droptol_E, droptol_1) = (0, 0, 0.001)$ $\Rightarrow$ 7 iterations

Nb proc	Precond. (sec.)	Solve (sec.)	Total (sec.)	Memory Efficiency	
				Precond.	Solve
1	803.12	104.87	907.99	1.00	1.00
2	384.12	58.84	442.95	0.99	1.00
4	205.96	46.87	252.83	0.99	0.99
8	129.35	21.00	150.35	0.97	0.99
16	65.50	18.81	84.31	0.96	0.98
32	35.15	9.42	44.57	0.93	0.97
64	18.51	4.79	23.31	0.87	0.96
128	9.84	2.41	12.25	0.83	0.94
256	5.84	1.41	7.26	0.75	0.90
512	3.80	0.69	4.49	0.62	0.82
1024	3.44	0.38	3.82	0.46	0.69
2048	4.76	0.34	5.10	0.29	0.39

Prospects for hexa-scale computing

Today

- Ready for peta-scale computing
- Scale up to thousands of processors

Tomorrow

- Avoid global synchronizations (collective communications)
- Parallelization of the pre- and post- computing steps
- Better coupling between our libraries and simulation codes (avoid data redistributions)

BACCHUS team

http://murge.gforge.inria.fr

http://scotch.gforge.inria.fr

http://pastix.gforge.inria.fr

http://hips.gforge.inria.fr