Latest improvements to Scotch and ongoing collaborations

Sébastien Fourestier
Harshitha Menon

November 22, 2011 – INRIA-UIUC joint laboratory

Table of contents

Scotch 6.0

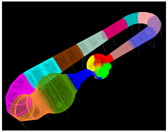
Dynamic load balancing in Charm++

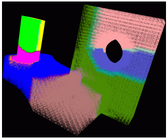
Prospects

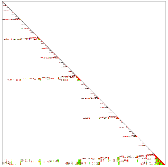
1
Scotch 6.0

The Scotch project

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







New functionalities of Scotch 6.0

- Partitioning and static mapping with fixed vertices
  - Allows some vertices to be fixed on predefined parts
    - Example: place special tasks on I/O nodes
  - Enables multi-phase mapping
    1. Maps the task graph of the first phase
    2. Maps the task graph of the second phase along with the mapped vertices of the first phase
- Sequential repartitioning and remapping with or without fixed vertices
  - Vertex migration costs
    - Is independent of vertex computation weights
    - Can be set individually for each vertex

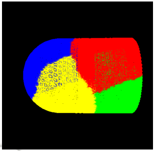
Improvements to Scotch internals

- Added *k*-way refinement algorithms
  - Improves the Scotch execution time
- Improved the recursive bipartitioning algorithm
  - Resultes in better quality
- New exactifier strategies
  - Obtains better load balance by compromising communication cost

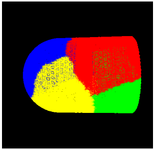
Experimental setup

- Original partition
  - 16 parts
  - Vertex loads are equal to 1
- First vertex load changes from 1 to $\frac{V}{2 \times 16}$ (V: the number of vertices)
- Scotch runs on 1 processor, ParMeTiS on 2 processors
- Migration cost from 0.1 to 50

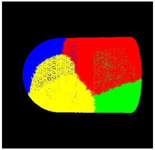
orig. partition



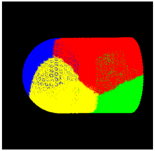
10



1



0.1



Test graphs

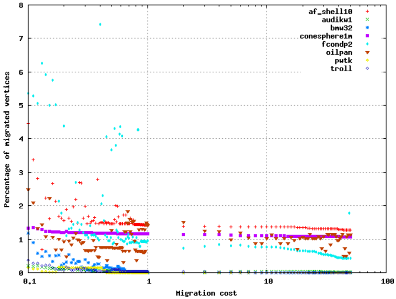
Graph	Vertex number	Edge number	Average degree
oilpan	73 752	1 761 718	47.8
fcondp2	201 822	5 546 247	55.0
troll	213 453	5 885 829	55.1
pwtk	217 918	5 708 253	52.4
bmw32	227 362	5 030 634	48.7
audikw1	943 695	38 354 076	81.3
conesphere1m	1 055 039	8 023 236	15.2
af_shell10	1 508 065	25 582 130	34.0

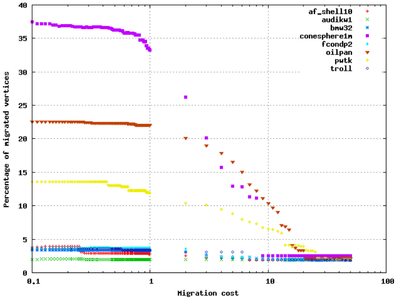
- Specificities:
  - fcondp2, troll, pwtk: close characteristics, same size
  - oilpan, bmw32: same characteristics, increasing size
  - audikw1: the highest degree
  - conesphere1m, af_shell10: > 1 million of vertices

Percentage of migrated vertices

Scotch

ParMETIS

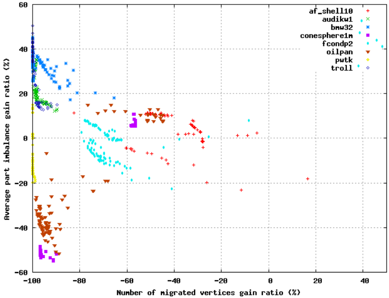




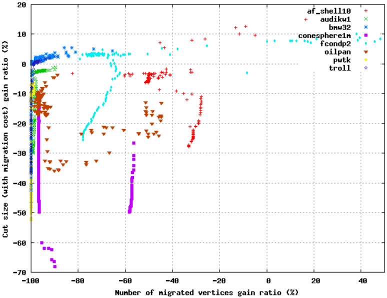
- Scotch migrates up to 4 times less than ParMETIS

Average load imbalance and cut size

Average load imbalance



Cut size



- On average:
  - Scotch and ParMeTiS have close imbalance
  - Scotch brings a 25% better cut size

Execution time

- Execution time on average:
  - Scotch (1 proc): 7.35 s
  - ParMeTiS (2 procs): 1.07 s
- Scotch is 9 times slower than ParMeTiS (standard deviation: 4.47)
- Causes of the overhead:
  - The gain brought by the parallelism on 2 processors
  - To improve quality, we are using both the diffusive method and the Fiduccia-Mattheyses heuristic
  - The overhead induced by the Scotch mapping functionalities (it takes target architecture into account during the gain computation)

Summary of experimental results

- Scotch migrate less than ParMeTiS
- On average, Scotch and ParMeTiS have close imbalance
- Scotch is 9 times slower than ParMeTiS
- Scotch brings a 25% better cut size
- We are tuning Scotch 6.0 mapping strategy before official publication

2
Dynamic load balancing in CHARM++

The CHARM++ project

- A portable object oriented programming language with a message driven execution model
- Capabilities:
  - Promotes natural expression of parallelism
  - Supports modularity
  - Overlaps communication and computation
  - Dynamic load balancing
  - Tolerates component failures

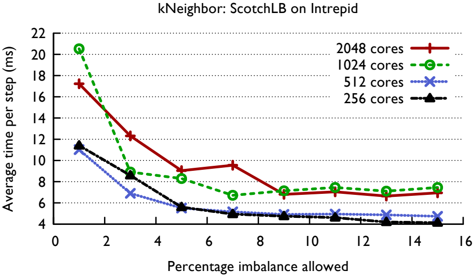
Dynamic load balancing in CHARM++

- Principle of persistence
  - Communication pattern and computational load of objects tend to persist over time
- Measurement-based load balancing
  - Instruments computation time and communication volume at runtime
  - Uses the database to make load balancing decisions
- Various load balancing strategies exist in CHARM++
- Scotch is used in CHARM++ as a load balancing strategy aimed at optimizing communication

kNeighbor benchmark

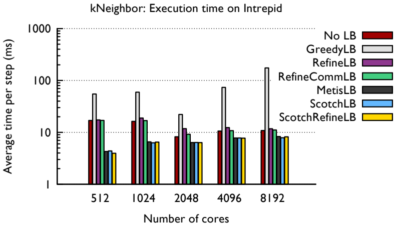
- Communication intensive benchmark
- In each iteration:
    - Each object exchanges a message of size 8 KB with fourteen other objects
- Object computational load is chosen uniformly at random
- The experiments were run on Intrepid (Blue Gene/P)

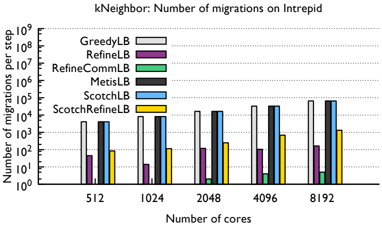
kNeighbor: Imbalance ratio



- Imbalance ratio indicates the percentage of load imbalance permissible during load balancing
- High imbalance ratio assist in optimizing communication cost ; 8-12% imbalance gives the best results

kNeighbor: Execution time & migrations


kNeighbor: Execution time on Intrepid

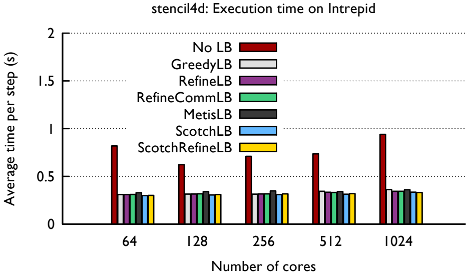

kNeighbor: Number of migrations on Intrepid

- METIS and Scotch have better execution time than the other load balancers
- ScotchRefineLB migrates 50-70% fewer objects than ScotchLB and still gives performance very similar to METIS and Scotch

stencil4d benchmark

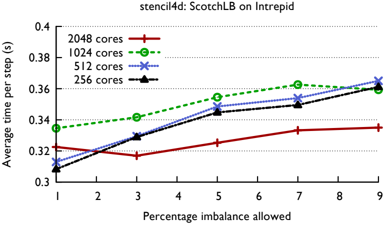
- Representative of the communication pattern in a Lattice QCD code
- Computation intensive benchmark
- In each iteration:
  - Each object exchanges boundary data with its eight neighbors
  - Once the data exchange is done, each object computes a 9-point stencil on its data
- The experiments were run on Intrepid (Blue Gene/P)

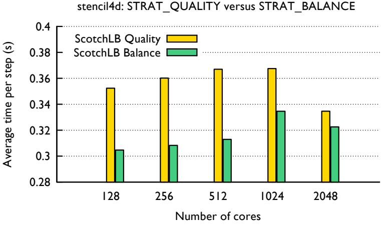
`stencil4d`: Execution time



- All load balancers reduce the execution time by 50-65% compared to `No LB`
- Due to the imbalance ratio parameter, `ScotchLB` gives 7-11% better performance compared to `MetisLB`

stencil4d: Imbalance ratio & strategies


stencil4d: ScotchLB on Intrepid


stencil4d: STRAT_QUALITY versus STRAT_BALANCE

- Best performance is obtained when strict load balance is ensured
- `STRAT_BALANCE` outperforms `STRAT_QUALITY` for `stencil4d` because it prefers balancing loads over optimizing communication

3
Prospects

Ongoing work

- Tuning Scotch 6.0 mapping strategy before official publication
- Parallel version of Scotch 6.0 new functionalities
  - Parallel static mapping
  - Parallel partitioning and static mapping with fixed vertices
  - Parallel repartitioning and remapping
  - Parallel repartitioning and remapping with fixed vertices
- Planed to be in Scotch 6.1
  - Release at the beginning of 2012

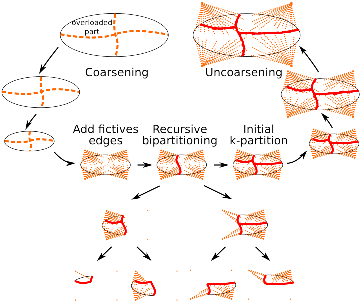
Ongoing collaborations within the joint laboratory

- Load balancing within Charm++
  - Sanjay Kalé, Abhinav Bhatelé and Harshitha Menon
  - A paper has been submitted to IPDPS
- Multi-phase mapping for OpenAtom (Charm++)
  - Anshu Arya and Ramprasad Venkataraman
- Scotch static mapping comparison
  - Torsten Hoelfer
- Clustering (fault resilence) (Charm++)
  - Esteban Meneses-Rojas
- Power-aware load balancing (Charm++)
  - Osman Sarood

Thanks

Dynamic repartitioning

- Multilevel framework adapted for repartitioning
  - Coarsening mates only vertices belonging to the same part
  - Initial mapping by recursive bimapping (with fictive edges)
  - K-way mapping refinement (with fictive edges)



Jug of the Danaides

- Sketch of the algorithm