11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
July 20–25, 2014, Barcelona, Spain

# MATCHING COMMUNICATION PATTERN WITH UNDERLYING HARDWARE ARCHITECTURE

**Emmanuel Jeannot[1], Guillaume Mercier[2] and François Tessier[3]**

[1] Inria, LaBRI, 200, Avenue de la Vielle Tour 33405 Talence Cedex, France
`http://www.labri.fr/perso/ejeannot/index.html`
[2] IPB, LaBRI, 200, Avenue de la Vielle Tour 33405 Talence Cedex, France
[3] University of Bordeaux, LaBRI, 200, Avenue de la Vielle Tour 33405 Talence Cedex, France
`firstname.lastname@inria.fr`

**Key words:** *Parallel Programming, High Performance Computing, Multicore Processing, Processe Affinity, Topology-Aware Mapping*

Due to the advent of modern hardware architectures of high-performance computers, the way the parallel applications are laid out is of paramount importance for performance. This abstract presents several techniques and algorithms that efficiently address this issue: the mapping of the application's virtual topology (for instance its communication pattern) onto the underlying physical topology. Using such strategy improves the application overall execution time significantly. We have developed two approaches. One regards the mapping of application processes during their deployment and the other consists in load-balancing the execution at runtime to match the dynamic application needs to the topology. All this work is based on the TreeMatch library [2].

TreeMatch is a library to perform process placement based on the topology of the machine and the communication pattern of the application. It provides a permutation of the processes to the processors/cores in order to minimize the communication costs of the application. It has several important features: first, the number of processors can be greater than the number of applications processes ; it assumes that the topology is a tree and does not require valuation of the topology (e.g. communication speeds) ; it implements different placement algorithms that are switched according to the input size ; it is able to prevent the use of some resources in case of non-contiguous resource allocation. TreeMatch is available at: `http://treematch.gforge.inria.fr`.

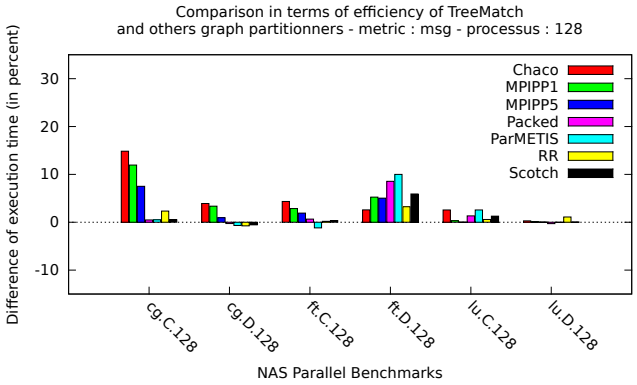


Figure 1: Comaprison of TreeMatch against other mapping strategies for different NAS benchmarks. TreeMatch consistently provides the (near) best solution

The static mapping strategy consists in recording the application communication pattern then applying TreeMatch on this pattern taking into account the underlying topology. The computed process permutation is then given to the process launcher (e.g. mpiexec) to map the processes onto the resources, minimizing the communication costs. Results show that our approach is better than other approaches such as graph partitioning (see Fig 1).

We also developed two load balancers for Charm++ [1]. Charm++ is a message passing-based programming environment based on the C++ language. However, while MPI considers processes in its programming model (with a granularity that is most of the time coarse), Charm++ model is based on a finer granularity by splitting computation in smaller tasks. A useful mechanism offered by this design allows to perform regular dynamic load-balancing.

The two load-balancers we wrote take into account both the computing power and the hierarchical topology depending on the fact that the application is compute-bound or communication-bound.

The first load balancer uses TreeMatch to reorder the tasks according to their communications : the more a task communicates with an other the more it will be close to it. Then it moves tasks to level the CPU load on each core. This step tries to keep the communication affinities as much as possible. Finally, the Hungarian algorithm is called to reorder some groups of tasks in order to minimize their migrations.This load balancer is designed for compute-bound applications as it favors the leveling of CPU loads.

The second load balancer is a hierarchical one because the tasks reordering is carried out on two levels of the topology. Indeed, it first migrates groups of chares assigned on each core in order to reduce the communication cost between these groups. Then, for each node, it replaces the tasks on cores according to their CPU load and affinities. This last step is executed in parallel. This algorithm focuses on communication-bound applications because it first reduces the congestion on the upper links in the topology tree.

These two load balancers gave us improvements for some applications, up to 10% of the execution time.

## REFERENCES

[1] L. Kale and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) 93*. ACM Press, September 1993, pp. 91–108.

[2] E. Jeannot and G. Mercier, "Near-optimal placement of mpi processes on hierarchical numa architectures," *Euro-Par 2010-Parallel Processing*, pp. 199–210, 2010.