A comment on "Process placement in multicore clusters: Algorithmic issues and practical techniques"

Zoltán Ádám Mann

IEEE Transactions on Parallel and Distributed Systems, vol. 27, num. 8, pp. 2475-2476, 2016

Abstract—In "Process placement in multicore clusters: Algorithmic issues and practical techniques," Jeannot, Mercier, and Tessier presented an algorithm called TREEMATCH for determining the best placement of a set of communicating processes on a hierarchically structured computing architecture, described by a tree. In order to speed up the algorithm, it was suggested to decompose levels of the tree with high arity into several levels of smaller arity. The authors conjectured what the optimal strategy for decomposition is. In this contribution, we prove that their conjecture was right.

Index Terms—Parallel programming; high-performance computing; multicore processing

1 INTRODUCTION

In their recent paper [1], Jeannot, Mercier, and Tessier addressed the problem of mapping a set of MPI processes onto a distributed architecture so as to minimize overall execution time. In the context of communication-intensive applications deployed on NUMA (Non-Uniform Memory Access) clusters, this is an important and challenging problem, because process placement has significant impact on application performance. In particular, processes that communicate intensively with each other should be placed as near to each other as possible in order to reduce data access latencies.

To this end, it was suggested that the communication patterns of the application should be matched to the underlying architecture. The authors devised an algorithm called TREEMATCH, consisting of the following three steps:

1) Gathering the communication patterns of the application, i.e., the messages exchanged among the processes. This is done using instrumentation and profiling runs.
2) Gathering information about the available hardware resources in a tree-like structure, including switches, nodes, processors, cores, memories, and caches. This is achieved by a dedicated tool called HWLOC [2].
3) Computing a matching between the MPI processes and the available computing units[1]. This involves traversing the tree of available resources from the leaves upwards, and iteratively grouping the processes.

Finally, the matching of processes to computing units computed by TREEMATCH is used in the actual process deployment, either in the form of resource binding or using the MPI technique called rank reordering.

The whole approach revolves around the hierarchical structure of hardware resources, and the resulting tree-based algorithm. The authors argue that operating directly on this tree provides substantial benefits over previous approaches using a topology matrix [3], [4], because the flattened view of the topology matrix results in information loss compared to the natural tree representation.

The TREEMATCH algorithm was first introduced in an earlier work of Jeannot and Mercier [5]. The main contribution of the recent paper in IEEE Transactions on Parallel and Distributed Systems was a significant reduction in the running time of the TREEMATCH algorithm [1]. This was necessary because finding the optimal grouping of processes on the given level of the tree requires an exponential number of steps. More specifically, let k denote the arity of the next level of the tree (i.e., the number of child nodes that the nodes of that level have, which is assumed to be identical for all nodes of the given level) and let p denote the number of process groups. Then, finding the optimal grouping for the next level requires time proportional to $\binom{p}{k}$, which is exponential in k [1].

In order to accelerate the algorithm, the authors suggested to decompose a level of the tree with high arity into several levels of smaller arity. Specifically, if d is a divisor of k, then a node of the tree with arity k can be decomposed into d nodes of arity k/d. This way, the number of steps of the algorithm changes from $\binom{p}{k}$ to $d \cdot \binom{p}{k/d}$, which may be an improvement, depending on

• *The author is with Budapest University of Technology and Economics*

1. A computing unit may be a processor or a processor core

the parameters. It was shown in the paper [1] that for p large enough and k not prime, a reduction of algorithm runtime can indeed be achieved this way. Moreover, the simulation results showed that the performance of the accelerated TREEMATCH algorithm compares very favorably to competing approaches based on graph partitioning and graph embedding [1].

The main open question is how to choose d so that $d \cdot \binom{p}{k/d}$ is minimal because this will result in the best performance. After checking all cases of $k \leq 128$ and $p < 500,000$, it was conjectured that the optimum is achieved when d is the highest non-trivial divisor of k [1], but no proof was given. In the following, we prove the correctness of this conjecture.

2 RESULT

Theorem 1. *Let $1 < k < p$ be integers, k not a prime, $p > 5$. For any divisor d of k, let $f(d) := d \cdot \binom{p}{k/d}$. Then among all non-trivial divisors of k, $f(d)$ is minimal for the greatest non-trivial divisor of k.*

Proof. Since d divides k, $x := k/d$ is an integer with $1 \leq x \leq k$. In terms of the variable x, we are looking for the minimum of

$$g(x) = \frac{k}{x} \cdot \binom{p}{x} = \frac{k \cdot p!}{x \cdot x! \cdot (p-x)!}.$$

The quotient of two consecutive elements of the $g(x)$ sequence is

$$\begin{aligned} \frac{g(x+1)}{g(x)} &= \frac{x \cdot x! \cdot (p-x)!}{(x+1) \cdot (x+1)! \cdot (p-x-1)!} \\ &= \frac{x \cdot (p-x)}{(x+1)^2}. \end{aligned} \tag{1}$$

Therefore,

$$\begin{aligned} \frac{g(x+1)}{g(x)} > 1 \;&\Leftrightarrow\; x \cdot (p-x) > (x+1)^2 \\ &\Leftrightarrow\; 0 > 2 \cdot x^2 + (2-p) \cdot x + 1. \end{aligned} \tag{2}$$

The expression $2 \cdot x^2 + (2-p) \cdot x + 1$ is a convex quadratic function in x, its discriminant is $\Delta = (2-p)^2 - 8$, which is positive since $p > 5$. Hence, this quadratic polynomial has two real roots x_1 and x_2; the condition of Equation (2) is fulfilled exactly in the interval (x_1, x_2). What we need from this is only that the condition is fulfilled in exactly one interval: that is, if it is fulfilled in two values of x, then it is fulfilled in all other values of x between these two as well.

If $x = 1$, we get $\frac{g(x+1)}{g(x)} = \frac{1 \cdot (p-1)}{(1+1)^2} = \frac{p-1}{4} > 1$.

If $x = \frac{p-1}{2} - 1 = \frac{p-3}{2}$, we get $\frac{g(x+1)}{g(x)} = \frac{(p-3) \cdot (p+3)}{(p-1)^2} = \frac{p^2-9}{p^2-2p+1} = 1 + \frac{2p-10}{(p-1)^2} > 1$.

Together with the above observation, we obtain that $\frac{g(x+1)}{g(x)}$ is greater than 1 for all $1 \leq x \leq \left\lfloor \frac{p-1}{2} \right\rfloor - 1$. (It should be noted that, although $g(x)$ is only defined for integer values of x, the expression for $\frac{g(x+1)}{g(x)}$ obtained in Equation (1) is sensible for all positive real numbers. Since $\frac{g(x+1)}{g(x)}$ is greater than 1 in $x = 1$ and in $x = \frac{p-1}{2} - 1$, and $\left\lfloor \frac{p-1}{2} \right\rfloor - 1$ is between these two, the condition is fulfilled here as well.) As a consequence, $g(x)$ is strictly monotonously increasing in the interval $\left[1, \left\lfloor \frac{p-1}{2} \right\rfloor\right]$.

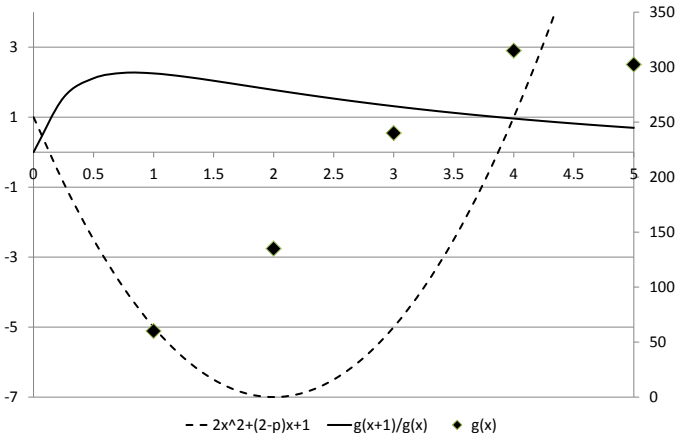

Fig. 1. Example with $p = 10$ and $k = 6$. The values of $2x^2 + (2-p)x + 1$ and $g(x+1)/g(x)$ are shown on the left vertical axis; the values of $g(x)$ are shown on the right vertical axis.

We are interested in the cases when d and x are non-trivial divisors of k, which means that they must be between 2 and $\lfloor k/2 \rfloor$. Since $k < p$, all possible values of x are in the interval $\left[2, \left\lfloor \frac{p-1}{2} \right\rfloor\right]$. Since $g(x)$ is strictly monotonously increasing in this interval, the minimal value of $g(x)$ is taken at the smallest possible value of x, which corresponds to the greatest non-trivial divisor d of k.

An example is shown in Figure 1. Here, $p = 10$, and so $\left\lfloor \frac{p-1}{2} \right\rfloor = 4$. In accordance with the above argumentation, $\frac{g(x+1)}{g(x)}$ is greater than 1 in the $[1, 3]$ interval, and hence $g(x)$ is strictly monotonously increasing in $[1, 4]$. □

ACKNOWLEDGMENTS

This work was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947).

REFERENCES

[1] E. Jeannot, G. Mercier, and F. Tessier, "Process placement in multicore clusters: Algorithmic issues and practical techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 993–1002, 2014.

[2] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: a generic framework for managing hardware affinities in HPC applications," in *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, 2010, pp. 180–186.

[3] H. Chen, W. Chen, J. Huang, and B. R. H. Kuhn, "MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters," in *Proceedings of the 20th Annual International Conference on Supercomputing*, 2006, pp. 353–360.

[4] G. Mercier and J. Clet-Ortega, "Towards an efficient process placement policy for MPI applications in multicore environments," in *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2009, pp. 104–115.

[5] E. Jeannot and G. Mercier, "Near-optimal placement of MPI processes on hierarchical NUMA architectures," in *Euro-Par 2010*, 2010, pp. 199–210.