# Thread-to-Core Optimization Problem

Muhammad Alfian Amrizal*, Mulya Agung†, Ryusuke Egawa‡, and Hiroyuki Takizawa‡
*Research Institute of Electrical Communication, Tohoku University, Japan
†Graduate School of information Sciences, Tohoku University, Japan
‡Cyberscience Center, Tohoku University, Japan
Email: {alfian@ci.cc., lipei@sc.cc., agung@sc.cc., egawa@, takizawa@}tohoku.ac.jp

Many interesting performance tuning problems exist in the field of HPC. However, these problems mostly require advanced programming skills and deep understanding on the programming models, system architectures, compilers, and so forth. Thus, introducing them to undergraduate level students who generally do not possess such skills is challenging. Therefore, it is important to introduce these tuning problems to the students in a more general way, i.e., a programming-language-free fashion, in order to keep them interested and feel motivated in solving such problems. In this assignment, we introduce students to one kind of the problems called *thread-to-core optimization problem*.

This assignment is part of a class called "Experiment D", which is a compulsory subject for the fourth-year undergraduate students in the Department of Electrical, Information, and Physics Engineering of Tohoku University. The class is also opened for graduate students who are interested in the HPC subject. Three sessions are allocated for introductory experiments using OpenMP and one session for this assignment. Since HPC or parallel and distributed computing (PDC) has not been integrated to the undergraduate level curriculum, the participants do not have any kind of experience on parallel programming prior to this class. Thus, an assignment that puts more emphasize on algorithmic thinking is introduced to increase students' interest toward HPC/PDC topics in general.

Given a multi-threaded application and a multi-core non-uniform memory access (NUMA) system (Figure 1), students are challenged to determine in which core of the system should a thread be placed. They need to develop a thread-to-core placement algorithm that minimizes the total execution time of the application. In a NUMA system, processor cores are grouped and a group of processor cores is called a *numa node* (or simply a "node") [1]. A numa node is associated with a local memory sub-system that consists of caches, memory controllers, and DRAM, as shown in Figure 1. Thus, the performance of memory access depends on the location of the threads that access the data of the node [2]. Although placing highly-communicating threads to the same node might improve performance due to the improved data access locality, placing too many of them in the same node could also decrease performance due to the increased pressure on the local memory sub-system (cache contention, congestion in memory controllers, etc.) [3]. The total execution time is expected to be minimal when a balance between data access locality and pressure on local memory sub-system is achieved.

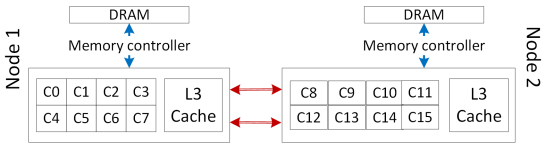


Fig. 1. A modern NUMA system, with two nodes and eight cores per node.

| | | | | |
|---|---|---|---|---|
| 3 | 10 | 1000 | 1000 | 0 |
| 2 | 10 | 100 | 0 | 1000 |
| 1 | 100 | 0 | 100 | 1000 |
| 0 | 0 | 100 | 10 | 10 |
| Thread | 0 | 1 | 2 | 3 |

Fig. 2. A communication matrix that shows the total size of communication with four threads. Each cell ($x$,$y$) of the matrix contains the data size of communication events between a pair of threads $x$ and $y$.

Therefore, students need to explore and develop some kind of load balancing techniques to solve this problem.

Students are presented with two types of communication matrices extracted from an application. The matrices represent: 1) the total number of communication events and 2) the total size of communication between every pair of threads. Figure 2 shows an example of one of the communication matrices for an application that consists of four threads. Using this information, students need to decide a set of pairs (thread_id:core_id) representing which thread in which core. In addition, students are given a multi-core simulator that simulates a real NUMA system. The simulator takes the set (thread_id:core_id) as inputs and gives some outputs such as execution time, L3 cache misses and DRAM queuing delay. Students can refer to these numbers to see the effectiveness of their thread-to-core placement algorithm. Note that, students basically only need these two matrices to solve the thread-to-core optimization problem and hence this assignment is appropriate even for students with little to no parallel programming skill.

This assignment covers some important concepts such as: 1) memory organization, 2) load balancing, 3) simulator based performance tuning, and 4) algorithmic optimizations.

We noted at least two strengths of this assignment: 1) challenging, yet doable, 2) fun because the students can compete with each other to achieve minimal execution time. As for the weakness of this assignment: 1) it is oriented specifically toward thread-to-core optimization problem, which is mainly done in the research-level of HPC (not generally known/used by standard HPC users), 2) it requires a high-

end multi-core server to speed up the simulator. Finally, more assignment's variation can be made by not only considering the spatial aspect (data access locality) but also the temporal aspect (considering threads that communicate almost at the same time) to reduce pressure on the memory sub-system.

The material of the assignment, including the assignment handout, the simulator, and some examples of the results are available at https://www.ci.cc.tohoku.ac.jp/project.html.

## REFERENCES


[1] F. Gaud, B. Lepers, J. Funston, M. Dashti, A. Fedorova, V. Quéma, R. Lachaize, and M. Roth, "Challenges of memory management on modern numa systems," *Communications of the ACM*, vol. 58, no. 12, pp. 59–66, 2015.

[2] M. Diener, E. H. Cruz, M. A. Alves, P. O. Navaux, and I. Koren, "Affinity-based thread and data mapping in shared memory systems," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 64, 2017.

[3] M. Agung, M. A. Amrizal, K. Komatsu, R. Egawa, and H. Takizawa, "A memory congestion-aware mpi process placement for modern numa systems," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 152–161.