HPC Programming Framework Research Team

1. Team members

Naoya Maruyama (Team Leader)
Motohiko Matsuda (Research Scientist)
Shinichiro Takizawa (Research Scientist)
Mohamed Wahib (Postdoctoral Researcher)
Koji Ueno (Student Trainee)
Satoshi Matsuoka (Senior Visiting Scientist)
Tomoko Nakashima (Assistant)
Aya Motohashi (Assistant)

2. Research Activities

We develop high performance, highly productive software stacks that aim to simplify development of highly optimized, fault-tolerant computational science applications on current and future supercomputers, notably the K computer. Our current focus of work includes large-scale data processing, heterogeneous computing, and fault tolerance. A major ongoing project in our group will deliver a MapReduce runtime that is highly optimized for the intra- and inter-node architectures of the K computer as well as its peta-scale hierarchical storage systems. Another major project focuses on performance and productivity in large-scale heterogeneous systems. We also study high performance graph analytics on the K computer. Below is a brief summary of each project.

1) Simplified Parallel Processing with KMR

MapReduce is a simple programming model for manipulating key-value pairs of data, originally presented by Dean and Ghemawat of Google. User-defined map and reduce functions are automatically executed in parallel by the runtime, which in turn enables transparent out-of-core data processing using multiple machines. Our KMR library, which has been developed over the last several years, is similar to the original MapReduce design by Dean and Ghemawat, but its implementation is significantly extended for the node and storage architectures of large-scale supercomputers such as the K computer. Highlights of new development and features are summarized below.

2) Automated Program Transformation for Better Access Locality

One of the important architectural trends is the deeper memory hierarchy consisting of different types of on-chip and off-chip memories with different capacity and performance profiles. To exploit the increase of the processor compute performance, it is becoming more important to effectively exploit near-processor faster memories by taking advantage of any available data access locality. General-purpose optimizing compilers are equipped with advanced program transformations to

address the challenge, however, they are often limited to a small set of rather simple program codes. We addressed the problem in the context of stencil computations on GPUs. More specifically, we have developed an automated framework that optimizes a given user CUDA programs by fusion and fission of CUDA kernels.

3) High Performance Graph Analytics Study with Graph500

Graph analytics is a new class of applications that are increasingly more important in various domains of sciences and societies. However, solving large-scale graph analytics problems is highly challenging due to its strong demand on computational and storage capability and capacity. The K computer, being the forth-fastest machine with Top500, is potentially able to serve such demand, however, the more unstructured characteristics of both computations and data accesses in graph analytics than those in conventional structured simulation codes require detailed attentions to be paid to different aspects of performance analyses and optimizations. As a first case study, we have been evaluating and optimizing the Graph500 benchmark for the K computer.

3. Research Results and Achievements

3.1. KMR (K MapReduce)

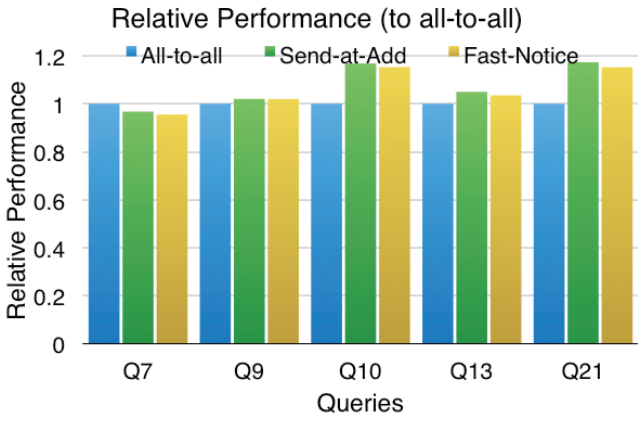
We are working on achieving highly reliable, high performance and high productivity data processing framework for large scale supercomputing systems, especially for the K computer. To this end, we achieved the following five research/development achievements. All these achievements are implemented in our MapReduce framework named KMR (K MapReduce).

1) Evaluation of asynchronous communication in MapReduce

KMR is designed to exploit the capacity and capability of supercomputers. To match the super-computer environment, KMR runs on memory, communicates by MPI, and utilizes CPU cores by multithreading. In such a design, the natural choice of shuffling communication is a collective communication MPI_Alltoallv. However, collective communications are synchronous and thus cannot exploit overlapping of communication and computation, which have been demonstrated effective in other MapReduce systems in the literature. Although overlapping would appear to be effective over collectives at first glance, handling MPI requests needed in overlapping takes cost in proportion to the number of processes. The cost hinders the performance especially in large scale, while the collectives are very efficient in large scale. This research evaluates the effectiveness of overlapping versus collective communications.

For this purpose, two new modes of shuffling communication are introduced in addition to the normal all-to-all mode. In the send-at-add mode, messages are sent when the buffer is full during mapping and reducing. In the fast-notice mode, a lightweight event notification is performed

alongside the send-at-add mode messages. We chose TPC-H as a benchmark with a shuffle-heavy workload, and benchmarked five out of the 22 queries defined in TPC-H. This benchmark was performed with 1024 nodes on the K Computer.

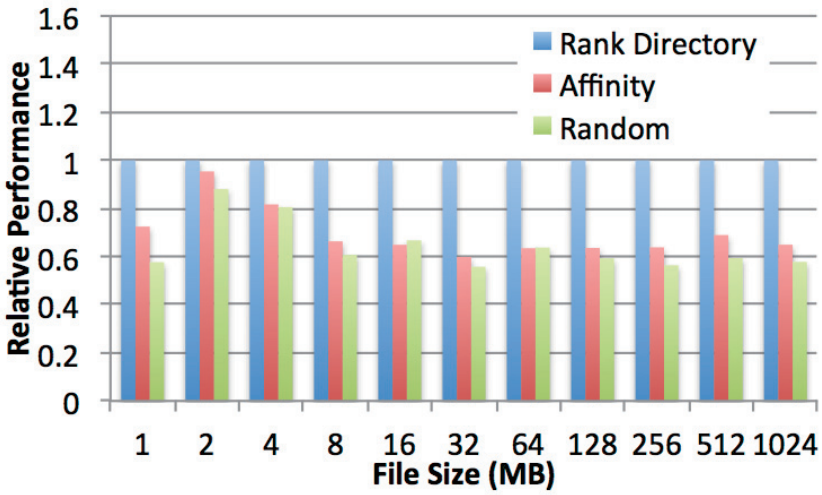


The results show only a slight improvement by overlapping, at most 15% reduction of the execution time. Although the fast-notice mode achieves precise polling timing, it slightly degrades the performance. Analyzing the breakdown of completions of the send-at-add mode messages reveals that the mapping/reducing operations are quickly finished before the messages are exchanged. That is, there is a little chance of overlapping for TPC-H queries. We conclude that the benefit of overlapping is marginal in the current MPI environment.

2) Locality-aware task assignment for improving IO performance

Files in a supercomputing system are stored in shared parallel filesystems, and when these files are used as inputs for a data processing job that uses thousands of compute nodes, these nodes individually but sometimes simultaneously access their inputs. This behavior can raise heavy IO contention on the shared parallel filesystem. We propose a task assignment method that considers locations of tasks' files and distance between the files and nodes to minimize the distance and reduce the IO contention for MapReduce's map task execution phase where such an IO pattern occurs. To implement this method for the K computer, we identify the six dimensional coordinates of nodes and location of file servers that stores each file, and then map nodes and tasks so that IO distance will be minimized.

We performed a simple file read task execution benchmark using K's shared parallel filesystem and rank directory, which is a process local directory. Though our proposal achieves only 70% of performance against rank directory where users explicitly need to specify task and node mappings, it improves 7% of performance against random task assignment [4-1].

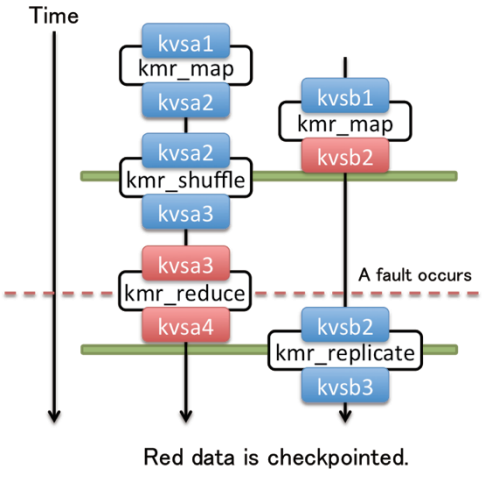


We also conducted an evaluation using a genome sequence-mapping program where 1,173 mapping tasks are simultaneously executed on 1,152 nodes. As a result, the average time of executing a task of our proposal is 870.75 seconds and that of the random task assignment is 882.05 seconds, 2% of performance improvement.

3) Checkpoint/Restart

To enable job completion in face of system faults and to enable long term job execution exceeding the maximum elapse time, we implemented a checkpoint/restart mechanism to KMR (KMRCR) in FY2013. Users can easily and transparently use KMRCR only by setting an environmental variable.

```
kvsa1 = kmr_create_kvs()
kvsa2 = kmr_create_kvs()
kvsa3 = kmr_create_kvs()
kvsa4 = kmr_create_kvs()
kvsb1 = kmr_create_kvs()
kvsb2 = kmr_create_kvs()
kvsb3 = kmr_create_kvs()
kmr_map_once(kvsa1)
kmr_map_once(kvsb1)
kmr_map(kvsa1, kvsa2)
kmr_map(kvsb1, kvsb2)
kmr_shuffle(kvsa2, kvsa3)
kmr_reduce(kvsa3,ksva4)
kmr_replicate(kvsb2,kvsb3)
```



However, as a drawback of ease and transparent use and achieving high reliability, KMRCR takes checkpoints of all KMR data and as a result it causes heavy IO load on disk. To reduce IO load, we implement two options to KMRCR. One is "*no fsync*" mode where fsync to a file is not performed until execution is done. This will reduce reliability, but if a user can assume that MTBF of a system

is larger than expected job completion time and there is only software fault, like killing by signal, It may help increasing performance. The other is "*selective*" mode where only user-specified KMR data are checkpointed. This will lose transparency, but also increase performance.

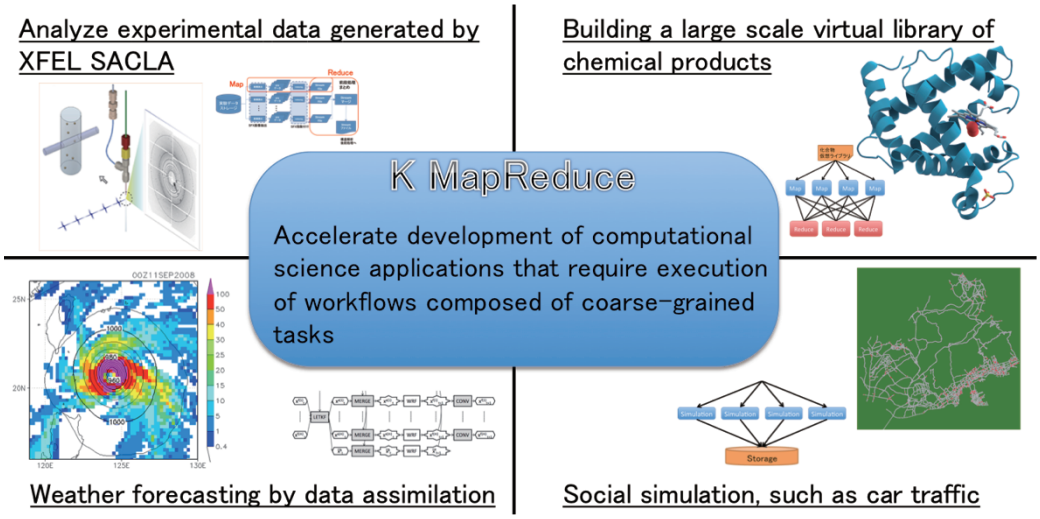
4) KMRRUN: high productive utility for MapReduce job execution

KMR provides C/Fortran API to fully use KMR functions and *kmrshell* utility for programming-less MapReduce execution. *KMRRUN* is a new utility that solves problems that kmrshell has and adds new functions that kmrshell doesn't have: map-only task execution, support for executing MPI program as task and checkpoint/restart.

5) Python API

KMR C/Fortran API enables full access to KMR functions but requires complex programming. On the other hand, using KMRRUN enables ease execution of MapReduce program but lacks flexibility. To achieve both full access to KMR functions and ease programming/executing MapReduce program, we are now designing and implementing Python API for KMR. This is an on-going project and will be release by middle of FY2015.

These above achievements, except the last one, are implemented in KMR and are released as open source software licensed under LGPL-2.1 [5-1]. We released six versions of KMR in FY2014. KMR is already widely used to develop various areas of computational science applications that require execution of workflows composed of coarse-grained tasks, such as large scale data processing and parallel task execution. To further propagate KMR, we hold lectures for KMR twice in this year. There ware 19 attendees in the first lecture and 9 attendees in the second from academia and industry.



3.2 Scalable Kernel Fusion for Memory-Bound Applications

We observed a pattern in a class of stencil-based scientific applications: GPU implementations of some applications relying on finite difference methods can include tens of kernels that are memory-bound. Kernel fusion can improve performance by reducing data traffic to off-chip memory via data reuse. We introduced a problem definition and proposed a scalable method for searching the space of possible kernel fusions to identify optimal kernel fusions for large problems. The proposed method was manually tested on real-world applications and showed promising results [2-3].

As a follow-up, we developed an end-to-end framework for automatically transforming stencil-based CUDA programs to exploit inter-kernel data locality. The CUDA-to-CUDA transformation collectively replaces the user-written kernels by auto-generated kernels optimized for data reuse. The transformation is based on two basic operations, kernel fusion and fission, and relies on a series of automated steps. The framework is modeled to provide the flexibility required for accommodating different applications, allowing the programmer to monitor and amend the intermediate results of different phases of the transformation. We demonstrated the practicality and effectiveness of automatic transformations in exploiting exposed data localities using a variety of real-world applications with large codebases that contain dozens of kernels and data arrays. Experimental results show that the proposed end-to-end automated approach, with minimum intervention from the user, improved performance of six applications with speedups ranging between 1.12x to 1.76x [2-2].

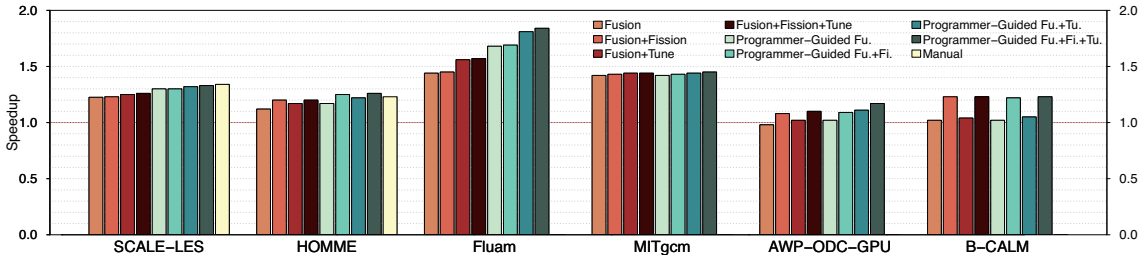


Fig. 1: Nvidia K40 speedup compared to baseline CUDA version for six real-world applications

3.3 High Performance Graph Analytics Study with Graph500

We have extended our implementation of the benchmark several times. Our first extension allowed us to achieve the fastest performance on the Graph500 benchmark list published on June 2014. Our algorithm required the number of nodes to be a power of two, so we used 65536 compute nodes out of more than 80,000 nodes of K. The scale of the system requires highly efficient inter-node parallelization, thus we developed a new algorithm that optimizes inter-node communication using bitmap representations. Furthermore, our algorithm also adaptively uses a sparse vector representation depending on traversal phases. These communication optimizations allowed us to greatly improve the scalability and resulted in 17,977 GTEPS with 65336 nodes for the scale 40 problem of the benchmark. In other words, our highly efficient implementation can traverse an extremely large graph of 1 trillion nodes and 16 trillion edges just under a second. This resulted in the fastest score published on June 2014, followed by the Sequoia system at Lawrence Livermore National Laboratory, which achieved 16,599 GTEPS.

As the second optimization, we have extended the implementation so that an arbitrary number of nodes can be used. While the first version was restricted to 65,536 nodes at maximum, this extension potentially allows us to use as many nodes as 82,944 and to boost the performance by up to 1.2x. However, it can also negatively impact the performance of communication since some of the key MPI communication routines are highly optimized for cases when the number of nodes is a power of two, which is a typical usage pattern in most of scientific simulation codes. As a result, the final performance was 19,585 GTEPS, which resulted in the second rank in the Graph500 list published on November 2014. We observed that the majority of the benchmark time is now spent in communication phases, and are investigating possible optimizations for further performance improvements.

4. Schedule and Future Plan

We plan to continue the development of KMR for simplifying usage of large-scale systems. In

particular, our primary focus in the coming years is to optimize I/O intensive applications by further exploiting data locality on the K computer. Toward that end, we will first try to identify common I/O access patterns of the computational science applications running on K, and examine potential improvements of I/O performance by various static and runtime techniques such as runtime code and data migrations. We plan to implement such advanced optimizations with the KMR library so that user applications built using the KMR library can transparently use our optimizations.

We plan to release the kernel-fusion framework in the near future. We also plan to extend the methodology to different application domains since memory access performance is almost a ubiquitously important problem. For the graph analytics study, we plan to continue our performance and optimization studies using the Graph500 benchmark.

5. Publication, Presentation and Deliverables

(1) Journal Papers

(2) Conference Papers

1. Motohiko Matsuda, Shinichiro Takizawa, Naoya Maruyama, Evaluation of Asynchronous MPI Communication in Map-Reduce System on the K Computer, EuroMPI Workshop 2014, Kyoto, Japan
2. Mohamed Wahib, Naoya Maruyama, Automated GPU Kernel Transformations in Large-Scale Production Stencil Applications, HPDC'15, ACM Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing, Portland, US
3. Mohamed Wahib, Naoya Maruyama, Scalable Kernel Fusion for Memory-Bound GPU Applications, SC'14, ACM/IEEE Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, US
4. Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka, "An OpenACC Extension for Data Layout Transformation," Proceedings of the First Workshop on Accelerator Programming using Directives (WACCPD '14), New Orleans, LA, November 2014.
5. Kento Sato, Kathryn Mohor, Adam Moody, Todd Gamblin, Bronis R. de Supinski, Naoya Maruyama, Satoshi Matsuoka, "A User-level Infiniband-based File System and Checkpoint Strategy for Burst Buffers," 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'14), Chicago, IL, May 2014.
6. Kento Sato, Adam Moody, Kathryn Mohor, Todd Gamblin, Bronis R. de Supinski, Naoya Maruyama, Satoshi Matsuoka, "Fault Tolerant Messaging Interface for Fast and Transparent

Recovery," 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS'14), Phoenix, AZ, May 2014.

(3) Invited Talks

1. Naoya Maruyama, "Japanese HPC Update: Exascale Research and Next-Genearation Flagship Supercomputer," 3rd Workshop on Extreme-Scale Programming Tools, Keynote Speech, New Orleans, LA, November 2014.
2. Naoya Maruyama, "High Performance and Highly Productive Stencil Framework for GPU Accelerators," International Workshop on Codesign, Invited Talk, Gungzhou, CN, November 2014.

(4) Posters and presentations

1. Shinichiro Takizawa, Motohiko Matsuda, Naoya Maruyama, Towards Locality-aware Large-scale Data Processing, Annual Meeting on Advanced Computing System and Infrastructure (ACSI) 2015, Tsukuba, Japan

(5) Patents and Deliverables

1. Motohiko Matsuda, Shinichiro Takizawa, Naoya Maruyama, "KMR: A MapReduce Library for K", http://mt.aics.riken.jp/kmr/, 2013.
2. No. 1 on the Graph500 Ranking of Supercomputers, June 2014.
3. No. 2 on the Graph500 Ranking of Supercomputers, November 2014.