HPC Programming Framework Research Team

1. Team Members

Naoya Maruyama (Team Leader)

Motohiko Matsuda (Research Scientist)

Soichiro Suzuki (Technical Staff)

Mohamed Wahib (Postdoctoral Researcher)

Shinichiro Takizawa (Research Scientist)

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. 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 is currently under active development, is similar to the original MapReduce design by Dean and Ghemawat, but its implementation is significantly extended for the node and storage architectures of the K computer. In particular, we exploit the two-level parallel storage systems so that costly data movement can be minimized. Data shuffling in MapReduce is also a subject of optimizations using the 6-D torus interconnect networks.

2) Physis: An Implicitly Parallel Stencil Computation Framework

Physis is a framework for stencil computations that is designed for a variety of parallel computing systems with a particular focus on programmable GPUs. The primary goals are high productivity and high performance. Physis DSL is a small set of custom programming constructs, and allows for very concise and portable implementations of common stencil computations. A single Physis program runs on x86 CPUs, NVIDIA GPUs, and even clusters of them with no

platform-specific code. This software consists of a DSL translator and runtime layer for each supported platform. The translator automatically generates platform-specific source code from Physis code, which is then compiled by a platform-native compiler to generate final executable code. The runtime component is a thin software layer that performs application-independent common tasks, such as management of GPU devices and network connections.

3. Research Results and Achievements

3.1. Simplified Parallel Programming with KMR

1) Kmrshell: A simple building block for MapReduce-style workflow

MapReduce is already a very simple model, but KMR's new functionality "kmrshell", further simplifies parallel programming which can start thousands of Unix commands without coding in C or Fortran. Starting thousands of programs on distinct data sets is a typical scenario of the jobs running on the K computer, but it requires tedious programming if directly implemented with MPI. "Kmrshell" abstracts invocation of multiple Unix commands as mappers and reducers, and efficiently combines their results through pipelining using KMR's high-performance data shuffling. Unmodified sequential or MPI programs can be executed more easily with kmrshell.

2) Automatic affinity-aware large-scale file I/O

KMR implements a new affinity-aware file loading method, which exploits locality information of the files to the I/O nodes [5-(2)-1]. The FEFS filesystem on K is based on file striping, in which a content of a file is segmented to many small chunks and they are scattered across many I/O nodes. It exposes locality of file contents between the I/O nodes and the computing nodes, enabling more efficient accesses to file segments from a near-by computing node, which also minimizes network contention during file loading. The location information of scattering the segments of a file can be obtained by using the system interface to the FEFS filesystem. KMR divides the computing nodes into the groups by the locality to the I/O nodes, and assigns the task of file loading by their affinity to the segments of a file. The following figure shows the improvement achieved by the affinity-aware file loading method.

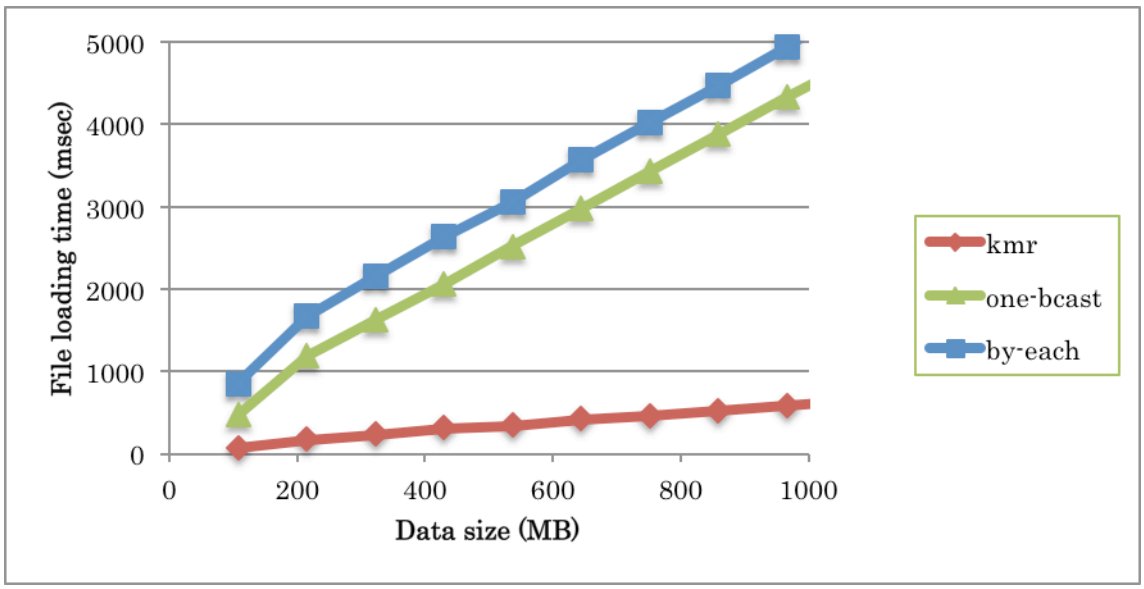


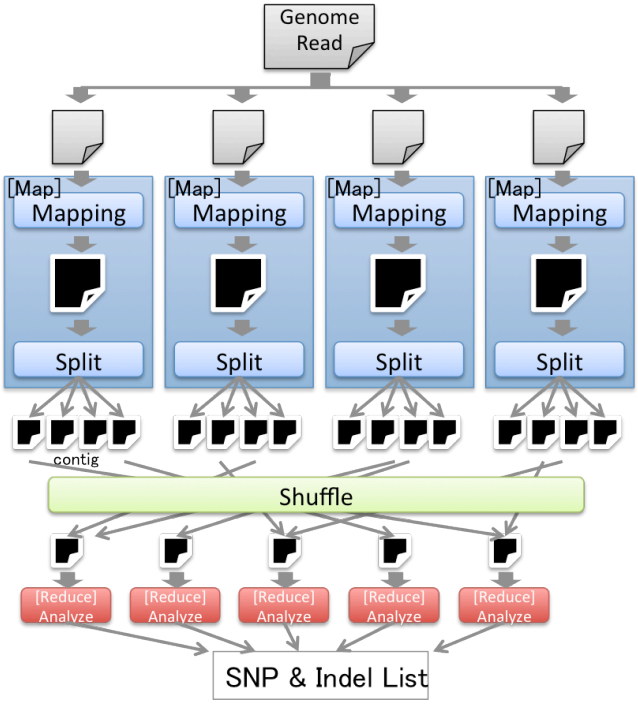
Figure. File loading performance. The line with "kmr" is by affinity-aware file loading. The "one-bcast" is the older method in which one node loads a file and broadcasts it. The "by-each" is a naïve method in which each node loads a file independently.

3) Fault-tolerant MapReduce

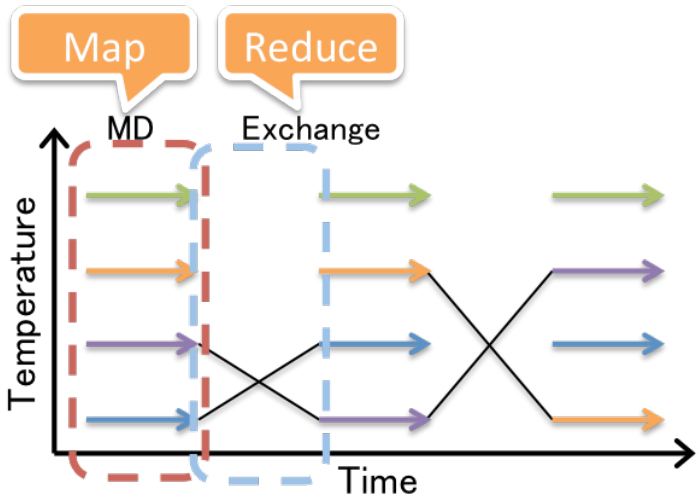
To enable job completion in face of system faults and to enable long term job execution exceeding the maximum elapse time, we implemented a checkpoint/restart feature to KMR. We designed this feature to automatically and transparently perform checkpoint/restart from KMR users. This feature will be included in the next release, which will be available early in FY2014.

4) KMR application examples

As MapReduce is originally designed for data parallel processing, it is easy to apply MapReduce to scientific data analysis [5-(2)-2]. We applied KMR to implement genome analysis, especially to cancer cell's mutations detection. As shown in the following figure, if we split the input genome sequences into small parts we can perform mapping and split them in parallel. This process can be mapped to the Map part of MapReduce. The split results should be merged based on their patterns and it can be mapped to the Shuffle part of MapReduce. The final analysis process can be implemented as Reduce part of MapReduce. KMR also starts to be used to analyzing huge amount of images, more than 1 million/day, generated by SACLA.



MapReduce can be applied to represent certain kind of scientific application's workflows, such as embarrassingly parallel tasks and ensemble simulations. We used KMR to implement Replica-Exchange Molecular Dynamics (REMD), a representative ensemble simulation. REMD is an iterative application that runs MDs in parallel and performs exchange after that in iteration. MapReduce can be applied to represent the iteration where Map runs MDs and Reduces does exchange. KMR starts to be used in AICS research tams to represents their application workflow, such as data assimilation for weather forecasting, social simulation and visualization.



By applying MapReduce to the above applications, application researchers or developers can reduce their coding tasks as MapReduce provides mechanisms for task management, and they can concentrate on implementing the application logics. Furthermore, as KMR is designed especially for the K computer, using KMR will benefit performance on the K computer.

3.2 Physis: An Implicitly Parallel Stencil Computation Framework

One of the main design goals of Physis is to generate high quality code optimized for memory-bound applications on GPU accelerators. A classical optimization technique for increasing memory-bound applications is loop fusion, which translates loops using the same data arrays into a single loop so that accesses to the same data can be reused via on-chip memories such as registers and caches. A similar technique can be applied to GPU applications that consist of a call sequence of GPU-offloaded functions (called kernels in CUDA terminology). By creating a fused kernel that include multiple kernels using the same data arrays, accesses to off-chip DRAM memory can be reduced by exploiting on-chip memories such as GPU shared memory.

We envision that such code transformation technique can be automatically implemented with high-level frameworks such as Physis. The Physis DSL, while limited to stencil computations with regular grids, gives the framework with flexibility in generating any combinations of fused kernels within the constraint of data dependencies among stencils. However, a naive greedy approach to fusing kernels will not be necessarily effective due to the following challenges. First, kernel fusion does not always lead to more efficient code even with shared data being cached at on-chip memories. Other performance-critical architectural constraints, such as the capacity of shared memory, latency to accessing the shared memory, and potential increase of register pressure, need also be carefully considered when deciding fusing a given set of kernels. Furthermore, production applications tend to contain a large number of kernels, much as loops in CPU codes, therefore the number of potential combinations of fusions can be intractably large with simple greedy approaches.

To address the two challenges and achieve speedup by effectively reducing the off-chip memory traffic, we formulate kernel fusion as a combinatorial optimization problem and propose the use of an approximation search heuristic to search the exponentially proportional space of possible fusions.

More specifically, we derive the optimization problem by constructing a data dependency graph and order-of-execution graph for the kernels in the program. Our search heuristic uses a light-weight and accurate projection method of performance upper bound to evaluate the quality of candidate solutions. The upper bound on performance is projected for potential fused kernels without requiring any form of code representation. This light-weight method is essential to enable fusions for applications with a large number of kernels and data arrays.

Our main achievements so far are: a) A formulation of the kernel fusion as an optimization problem, b) The use of a scalable search heuristic to search for near-to-optimal solutions in the space of possible kernel fusions, c) Using a highly accurate codeless upper-bound performance projection to guide KF and, d) Experimental evidence of the effectiveness of the kernel fusion optimization when applied to a test suite and two real world weather applications with tens of kernels and data arrays. As will be shown later, the introduced search method identified the optimal kernels to fuse for the weather applications within a reasonable amount of time. The actual kernel fusion transformation with two climate-modeling applications resulted in more than 1.35x and 1.2x speedup on NVIDIA Kepler GPUs.

4. Schedule and Future Plan

We plan to continue the development of KMR for further 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 also plan to release a new version of KMR with a checkpoint/restart mechanism so that KMR applications can continue execution even in the presence of system failures due to, e.g., hardware faults.

The Physis framework will also be further extended with more advanced code generation techniques such as kernel fusion as presented above. We plan to realize the model-based scalable kernel fusion within the framework so that the performance of stencil applications written in Physis can be further more efficient.

5. Publication, Presentation and Deliverables

(1) Conference Papers

[1] Naoya Maruyama, Takayuki Aoki, "Optimizing Stencil Computations for NVIDIA Kepler GPUs," International Workshop on High-Performance Stencil Computations, Vienna, January 2014.

[2] Shinichiro Takizawa, Motohiko Matsuda and Naoya Maruyama: Supporting Workflow Management of Scientific Applications by MapReduce Programming Model. IPSJ HPCS2014 (2014).

[3] Toshiya Komoda, Shinobu Miwa, Hiroshi Nakamura, Naoya Maruyama, "Integrating Multi-GPU Execution into an OpenACC Compiler," 42nd International Conference on Parallel

Processing (ICPP), pp. 260--269, Lyon, France, October 2013.

[4] Mohamed Attia Wahib, Naoya Maruyama, "Highly Optimized Full GPU-Acceleration of Non-hydrostatic Weather Model SCALE-LES," IEEE Cluster 2013, Indianapolis, IN, USA, September 2013.

[5] Motohiko Matsuda, Naoya Maruyama, Shinichiro Takizawa, "K MapReduce: A Scalable Tool for Data-Processing and Search/Ensemble Applications on Large-Scale Supercomputers," IEEE Cluster 2013, Indianapolis, IN, USA, September 2013.

[6] Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka, Ryoji Takaki, "CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application," Proceedings of the 2013 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013), Delft, the Netherlands, May 2013.

[7] Mohamed Slim Bouguerra, Ana Gainaru, Leonardo Bautista Gomez, Franck Cappello, Satoshi Matsuoka, Naoya Maruyama, "Improving the Computing Efficiency of HPC Systems Using a Combination of Proactive and Preventive Checkpointing," Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13), Boston, USA, May 2013.

(2) Invited Talks

[8] Naoya Maruyama, "Miniapps for Enabling Architecture-Application Co-design for Exascale Supercomputing," 19th Workshop on Sustained Simulation Performance, Invited Talk, March 2014.

[9] Naoya Maruyama, High performance and high productivity with application frameworks, Kyoto University, Invited talk, July 2013.

(3) Posters and Presentations

[10] Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka, "OpenACC Performance and Optimization Studies With Kernel and Application Benchmarks," GPU Technology Conference, Poster, San Jose, CA, USA, March 2014.

[11] Kento Sato, Akira Nukada, Naoya Maruyama, Satoshi Matsuoka, "I/O Acceleration With GPU for I/O-Bound Applications," GPU Technology Conference, Poster, San Jose, CA, USA, March 2014.

[12] Mohamed Wahib, Naoya Maruyama, "Scalable Kernel Fusion for Memory-Bound GPU Applications," GPU Technology Conference, Poster, San Jose, CA, USA, March 2014.

[13] Mohamed Attia Wahib, Naoya Maruyama, "Scalable Kernel Fusion for Memory-Bound GPU Applications," SIAM Conference on Parallel Processing, MS 49: Parallel Methods and Algorithms for Extreme Computing, Portland, Oregon, USA, February 2014.

(4) Patents and Deliverables

[14] Motohiko Matsuda, Shinichiro Takizawa, Naoya Maruyama, "KMR: A MapReduce Library for K," http://mt.aics.riken.jp/kmr/, 2013.