14. HPC Programming Framework Research Team

14.1. Team members

Naoya Maruyama (Team Leader)
Motohiko Matsuda (Research Scientist)
Soichiro Suzuki (Technical Staff)
Mohamed Wahib (Postdoctoral Researcher)
Michel Müller (Technical Staff)

14.2. 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) Large-Scale Data 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.

14.3. Research Results and Achievements

14.3.1. Large-Scale Data Processing with KMR

Our major achievements with KMR consist of its basic design and the first prototype implementation. The design of KRM is similar to Hadoop, which is a popular MapReduce implementation in Java, but our implementation is completely different. We initially considered reusing much of the Hadoop software components, but because of limited support of the Java programming language on the K computer, we define our own MapReduce as a standard C library. This allows for simpler integration of existing optimized system software components. For example, data shuffling is one of the most challenging processing phase in MapReduce because of its high communication intensity, so exploiting the maximal performance of the underlying interconnect, the Tofu network, is highly important. Our KMR is designed in a way that such optimizations for large-scale supercomputers can be transparently integrated.

14.3.1.1 Prototype Implementation and its Optimizations

The basic prototype implementation runs on both the K computer and standard cluster systems with several K-specific extensions and optimizations, including fast file reading and scalable data shuffling. The storage architecture of the K computer system consists of the global storage, which presents very large capacity (tens of peta bytes), and the local storage, which allows for higher bandwidth and lower latency than the global storage when I/O accesses by user applications exhibit spatial and temporal locality. The two storage systems are organized by the K’s data staging system, however, I/O read performance with the local storage still exhibits scalability problems even with a modest number of nodes. This is often the case with MapReduce, where a large number of Mapper processes simultaneously access input data. In our KMR, this operation is tuned for the K storage architecture by limiting the read concurrency to storage systems and exploiting the inter-node data communications. This optimization effectively introduces additional data staging to the application I/O data flow, where the first staging is performed between the global and local storage systems, and the second staging is between the local storage and compute nodes. A comparative performance study can be found in Fig 1, which shows much more scalable performance than a normal I/O method. Although the additional stage complicates the overall application structure when it is implemented manually, it is completely automated in our KMR library without any user intervention.
Another major optimization is the scalable data shuffling. Since each map operation tends to generate relatively small data to be consumed by the reduce operation, our KMR uses a collective communication algorithm proposed by Bruck et al., which performs communications in log(p) stages, where p is the number of processes. As shown in Fig. 2, the performance with our own implementation exhibits much better performance for small messages, but it quickly increases as the data size grows. Since the selection of communication methods have large performance impact on data shuffling, we plan to develop an intelligent mechanism that automatically choose the most efficient one depending on message sizes.

Fig. 1  Read latency comparison on 192 nodes. The blue line shows the read time when all nodes simultaneously read the data, while the red line shows the performance when only a part of the nodes load the data from the local storage, which are then transferred to the rest of the nodes using MPI over the Tofu network.

Fig. 2  Comparison of data shuffling time on 16384 nodes. The blue line shows the performance when using an all-to-all API of the Fujitsu MPI, while the red line shows the performance using the custom collective communication.
14.3.1.2 Case Study: Metagenome Sequence Analysis

As a case study, we applied KMR to metagenome sequence analysis. Sequence homology search is an important computational method in life science. Sequences similar to known amino acid sequences are searched using large-scale metagenome sequences. We have developed a MapReduce-based implementation using a homology search program called GHOSTX developed at the Tokyo Institute of Technology. We have demonstrated that a large number of compute nodes of the K computer can be used to achieve higher performance, even without parallel programming.

14.3.2 Physis: An Implicitly Parallel Stencil Computation Framework

The main achievement in the Physis framework is optimized code generation to achieve both high productivity and high performance. The Physis DSL translator now has a set of translation passes that apply a variety of generic and stencil-specific optimizations, which achieves comparable performance as hand-tuned stencil code on a GPU. Furthermore, we developed a prototype auto-tuner for Physis, which experimentally finds the best configuration of optimization passes. Detailed evaluation and extended case studies are subject of future work.

Other major achievements include modeling and implementation of scalable fault-tolerance schemes, and evaluation of new accelerator programming models. In particular, we conducted an extensive performance study of OpenACC, which is a new directive-based accelerator programming interface. Our finding suggests that it can greatly simplify programming burden, however, the performance cost compared to tuned CUDA code still needs to be addressed.

14.4. Schedule and Future Plan

Our major milestones in FY2013 are the first release of KMR and further application case studies. The release will be freely available on the K computer with documentation and sample applications. We plan to apply the implementation to a wider variety of applications to demonstrate its effectiveness.

Our current implementation has several limitations, including lack of fault tolerance and load balancing, both of which are important challenges in large-scale machines such as K and subject of our long-term research goals.

14.5. Publication, Presentation and Deliverables

(1) Journal Papers
   - None

(2) Conference Papers
   1. Tetsuya Hoshino, Naoya Maruyama, Satoshi Matsuoka, Ryoji Takaki, "CUDA vs


(3) Invited Talks

(4) Posters and presentations

(5) Patents and Deliverables
-None