Compiler Technology That Demonstrates Ability of the K computer

● Koutarou Taki ● Manabu Matsuyama ● Hitoshi Murai
● Kazuo Minami

We developed SPARC64 VIIIfx, a new CPU for constructing a huge computing system on a scale of 10 PFLOPS. To make the best use of the features of this CPU, we developed a language package called "Parallelnavi Technical Computing Language." This paper presents compilers for Fortran/C/C++ included in the language package. In these compilers, we enhanced the optimization function for sequential processing (sequential optimization) and the function of the compilers to automatically generate thread parallel processing codes (automatic parallelization) to bring out the best of SPARC64 VIIIfx. Moreover, we have provided a hybrid parallel execution model that combines thread parallel execution and process parallel execution to realize high execution performance in a large-scale system. This model supports the latest industry-standard language specifications, and so it has allowed us to compile a wider range of programs.

1. Introduction

The demand for large-scale, ultra-high-speed computing in the field of scientific computation is ever increasing. In a national project as a joint development with the Institute of Physical and Chemical Research (RIKEN), a target was set to build a system with a LINPACK performance of 10 PFLOPS (a system scale of ten quadrillion [ten thousand trillion] operations per second), which is almost 100 times the performance of conventional systems. To build such a system, we developed a CPU with proprietary extensions to the SPARC architecture.

To maximize the performance of a CPU, it is important to make the most of its features. To that end, we adopted a compiler technology closely linked with CPU architecture enhancement. The CPU performance has been brought out by implementing a new compiler technology in addition to taking advantage of the existing optimization technology.

The K computer,[note 1)] which has been promoted by the Ministry of Education, Culture, Sports, Science and Technology and under development by the RIKEN and Fujitsu, is a massive system with more than 80 000 multi-core CPUs connected together. To achieve high execution efficiency in such a large system, we have provided support for a hybrid execution model combining thread parallel and process parallel execution.

This paper presents the technologies adopted for the compilers intended for the K computer and our approach to improving its performance. The following sections give descriptions with the focus on 1) language specification (support for new standards and industry standards), 2) hardware feature, 3) optimization for sequential

note 1) "K computer" is the English name that RIKEN has been using for the supercomputer of this project since July 2010. "K" comes from the Japanese word "Kei," which means ten peta or 10 to the 16th power.

processing that makes use of SPARC64 VIIIfx (sequential optimization), 4) automatic parallelization technology and hybrid parallel execution model and 5) debugging function.

2. Features concerning language specification

For the K computer, we have developed three compilers: Fortran, C and C++ compilers. In the development of these compilers, we paid attention to the following points in relation to language specification.

1) Adoption of new standards

Programming language standards are not invariable. They have been revised once every few years so as to improve the description efficiency based on past experiences of programming engineering. It is important to support new standards so as to reduce the entire research period including the period of application program development.

2) Support for industry standards

In today's program development, well-developed open-source software (OSS) is generally used, thereby reducing the development person-hours to focus on the core of the problem to be solved. Accordingly, support for industry standard specifications used in OSS is essential.

3) Hardware optimization

It is important to have libraries and optimization mechanisms that allow the performance of the target platform to be maximized. We need to offer appropriate support for the new functions introduced in SPARC64 VIIIfx.

To satisfy these three requirements, we established the following language specification.

- Fortran compiler

Adopted major Fortran 2003 standards. Provided conformity to OpenMP 3.0.

- C compiler

Provided conformity to the C99 standard. Provided support for some of the C language extension specifications implemented in GNU C compiler version 4.1.2. Prepared built-in functions capable of directly handling the single instruction multiple data (SIMD) instructions added in High Performance Computing - Arithmetic Computational Extensions (HPC-ACE). Provided conformity to OpenMP 3.0.

- C++ compiler

Provided conformity to C++03 standard. Prepared GNU extension specification and built-in functions capable of directly handling SIMD instructions equivalent to those of the C compiler mentioned above. Provided conformity to OpenMP 3.0.

For the three compilers above, we prepared BLAS, LAPACK and ScaLAPACK, industry-standard math libraries optimized for the K computer and our math libraries SSL II, C-SSL II and SSL II/MPI.

As a means of inter-node parallelization, we have provided XPFortran, Fujitsu's proprietary parallel programming language, in addition to the Message Passing Interface (MPI) library. XPFortran has grammar with Fortran extended with directives and allows hybrid parallel (automatic thread parallel and MPI process parallel) execution by combining with sequential optimization and automatic parallelization, which will be described later.

3. Hardware features

This section gives a description of HPC-ACE, a set of extensions to SPARC64 VIIIfx, and Virtual Single Processor by Integrated Multicore Parallel Architecture (VISIMPACT), which has already been used for the Fujitsu SPARC architecture.

3.1 HPC-ACE

HPC-ACE is an architecture based on the conventional SPARC architecture extended for supercomputers. HPC-ACE includes the following four extensions.

1) Register expansion

With reference to the conventional SPARC

architecture, the number of integer registers has been expanded to 188 from 156 and the number of floating-point registers to 256 from 32. This provides potential for improved instruction-level parallelism and reduces the number of data backups and restorations to and from the memory resulting from insufficient registers, which leads to faster applications.

2) Addition of SIMD instructions

SIMD provides a mechanism for performing the same operation on multiple pieces of data simultaneously with one instruction. HPC-ACE allows two operations to be executed at the same time in one instruction.

3) Addition of new instructions

New instructions often used in scientific computation have been provided including reciprocal approximation, trigonometric function auxiliary, masked substitute and maximum/minimum value instructions. Reciprocal approximation instruction does not interrupt pipeline operation and high computational throughput can be maintained. With a simple division program, use of reciprocal approximation instruction reduces the runtime to one-third or less. The introduction of masked substitute instruction allows the conditional branches (IF statements) in loop processing to be reduced, which accelerates the optimization of loop processing.

4) Sector cache

Sector cache is a function in which cache is divided into two fields to separate reusable data from temporary data so that the reusable data is cached as much as possible. This improves the cache hit ratio, and allows applications to speed up.

3.2 VISIMPACT

VISIMPACT is an architecture that allows the user to handle multiple cores as one high-speed CPU. This mechanism can be used to speed up thread parallel processing and hybrid parallel processing in multi-core CPUs.

Two important technologies have been realized with VISIMPACT.

1) Shared L2 cache

The structure of sharing the L2 cache between all cores of the CPU mitigates the impact of false sharing[note 2)] and suppresses the degradation of application performance.

2) Inter-core hardware barrier

It may be necessary to synchronize between threads to allow for inter-core thread parallel processing. Inter-core hardware barrier processing is approximately ten times faster than software barrier processing; even when the scale of the problem is small (parallelism is fine-grained), high-efficiency inter-core thread parallel processing can be achieved.

4. Sequential optimization technology

Of the enhancements to compilers for hardware features mentioned in the previous section, this section describes the technology for sequential optimization.

4.1 Register expansion and SIMD

The registers expanded by HPC-ACE can be shared with registers defined by SPARC-V9 for use. This means that the number of registers to be allocated by compilers has simply increased.

While non-SIMD instructions can access all arbitrary registers, SIMD instructions perform simultaneous operation on a pair of basic and extended registers. There is a restriction that an extended register number is fixed to be the corresponding basic register number plus 256. For example, a SIMD operation instruction performs operation simultaneously on f[0] and f[256]. Because of this restriction, analyzing and

note 2) In a system composed of multiple cores with respective caches, when non-shared data specific to cores are on the same cache line, false sharing is a phenomenon in which writing to the data concerned causes the cache line to be invalidated and inter-cache data transfer occurs.

optimizing dependencies become complicated when SIMD and non-SIMD instructions are mixed together. Still, it has a significant effect of allocating registers with minimum waste. For example, if a non-SIMD instruction allocates f[0] and f[2], the number of registers that can be used by a SIMD instruction is decreased by two but, if f[0] and f[256] are allocated, the decrease in the number of registers that a SIMD instruction can use is only one.

When there is any restriction to the registers to be allocated, data are generally transferred to registers that satisfy the restriction but data transfer is minimized to improve the execution performance. However, if register allocation is performed simultaneously with optimization through minimizing the number of transfer instructions, the complexity of processing increases, which means a lot of time and memory is required for compilation.

We have taken an approach of generating transfer instructions before register allocation to allocate the same registers before and after transfer as much as possible. Data transfer instructions with the same registers before and after transfer can be deleted after register allocation, and so generation and execution of more transfer instructions than required is avoided. In this way, the optimum register allocation has been achieved within a practicable range of time and memory amount.

4.2 Extraction of instruction-level parallelism

Instruction-level parallelism is an indicator of how much of processing can be executed in parallel in an instruction sequence. Higher parallelism means the computing units are idle for a shorter time and are kept busy, which leads to faster execution of applications. A compiler makes use of optimization such as instruction scheduler and software pipelining to rearrange the order of instructions and generate an instruction sequence so as to achieve higher parallelism. With improved instruction-level parallelism, however, the need for temporarily holding data in registers also increases, resulting in a larger number of registers used. For this reason, the maximum instruction-level parallelism could not always be extracted with the conventional architecture.

With HPC-ACE, the larger number of registers allows enhanced instruction scheduler and software pipelining to be positively applied. Combining such optimization with reciprocal approximation and masked substitute instructions has led to the achievement of even higher instruction-level parallelism than before.

4.3 Effect of HPC-ACE

We have evaluated the performance of the compiler sequential optimization that makes use of register expansion and SIMD instructions, which are features of HPC-ACE. To do so, we used 146 Fortran performance evaluation programs owned by our division (**Figure 1**). The vertical axis of the figure indicates the performance ratio (improvement rate) with reference to the performance with only the basic registers allocated and no SIMD instructions used as performance value 1. The result showed that a performance improvement of 1.26 times on average and 4.03 times at the maximum has been confirmed with only expanded registers

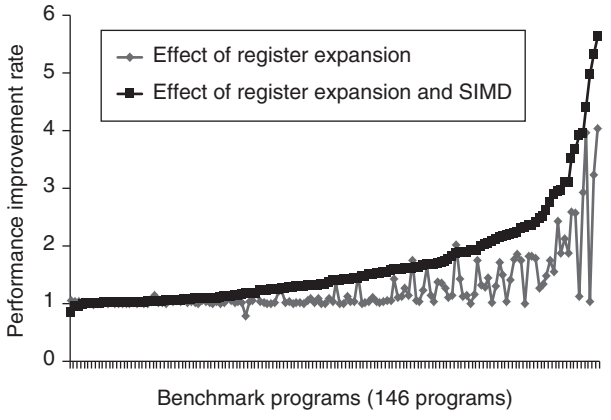

Figure 1
Results of evaluating register expansion and SIMD.

used (and no SIMD instructions used). With both expanded registers and SIMD instructions used, a performance improvement of 1.66 times on average and 5.62 times at the maximum has been confirmed.

5. Automatic parallelization technology and execution model

Of the technologies implemented for enhancing the existing automatic parallelization, this section describes variable privatization, utilization of the hardware barrier function and an execution model that combines thread parallel and process parallel processing (hybrid parallel model).

5.1 Variable privatization

Privatizing intra-loop variables has allowed automatic parallelization of loops that could not be automatically parallelized in the past and expansion of the range of parallelization, which has led to an improved parallelization rate. With this function, automatic parallelization has become possible for all loops with OpenMP directives in the NAS Parallel Benchmarks (**Table 1**). That is, with programs like the NAS Parallel Benchmarks, manual program analysis and addition of OpenMP directives are not required but equivalent parallelization can be expected by means of automatic parallelization.

Table 1
Automatic parallelization performance with NPB 3.3.

	Number of loops with OpenMP	Number of loops to be automatically parallelized
CG	25	25
FT	13	13
MG	22	22
BT	38	38
SP	74	74
LU	64	64

5.2 Utilization of hardware barrier function

The hardware barrier function enhanced in SPARC64 VIIIfx has been used in the thread parallelization runtime library to improve the overhead in parallel execution to 350 ns from 750 ns. This improvement has allowed parallelization of fine-grained loops, which were not parallelized because the parallelization effect was not obtained with the conventional technology, and the execution performance has been improved.

5.3 Hybrid parallel execution model

For a parallel application that runs on a PC cluster or such like, flat MPI (a parallel execution model that uses MPI also for parallel execution between cores) is often used in a multi-core configuration as well. With flat MPI, the number of running parallel processes increases as the number of CPU cores used by the parallel application increases. The usage of buffer memory for communication between parallel processes and the number of communications required increase according to the number of entities at the other end of communication, or parallel processes. This limits the amount of memory that can be used by the application. In addition, the limit to the amount of communication buffer may cause degraded communication performance. For these reasons, the execution efficiency may be reduced.

For the K computer, which aims to achieve high performance by ultra-high parallelization, a hybrid programming model is provided: Process parallel execution is used for inter-node parallelization and thread parallel execution that makes use of VISIMPACT for intra-node parallelization. In this way, the increase in the number of parallel processes is minimized to avoid reducing the amount of usable memory, by which parallel programs on many nodes can be executed (**Figure 2**).

6. Debugging function

In addition to debugging with the conventional tools, Fujitsu's compilers provide a runtime checking function, which checks a program at the time of compilation and running and gives a warning if there is any deviation from the language specification. This function allows application developers to detect errors to some extent by translating and running programs. With the existing compilers, however, enabling this checking function caused the runtime to increase by a few tens to a few hundreds of times.

For the compilers of the K computer, this function has been revised and implemented as a "built-in debugging function" that runs at high speed. This built-in debugging function conducts minimum checking such as procedural parameter and subscript range checks. For detecting any undefined variable, a function has been provided in which *not a number* (NaN) is set as the initial value and any invalid operation is detected as an exception. A simplified version that only outputs line numbers and error types as error messages and a detailed version that additionally outputs variable names and procedure names have been made selectable as required.

The built-in debugging function developed is available with C/C++ compilers in addition to Fortran and can also be combined with thread parallelization (OpenMP and automatic parallelization).

The impact of the debugging function on execution performance has been evaluated by using the Fortran program for HPC performance evaluation owned by our division (**Figure 3**). As the optimization option, -Kfast has been specified. It has been found that using the new built-in debugging function requires 1) 8.4 times as much time for error check + detailed message output, 2) 5.6 times as much time for error check with NaN + detailed message output and 3) 2.7 times as much time for error check with NaN + simplified message output. As compared with the existing debugging functions, which required

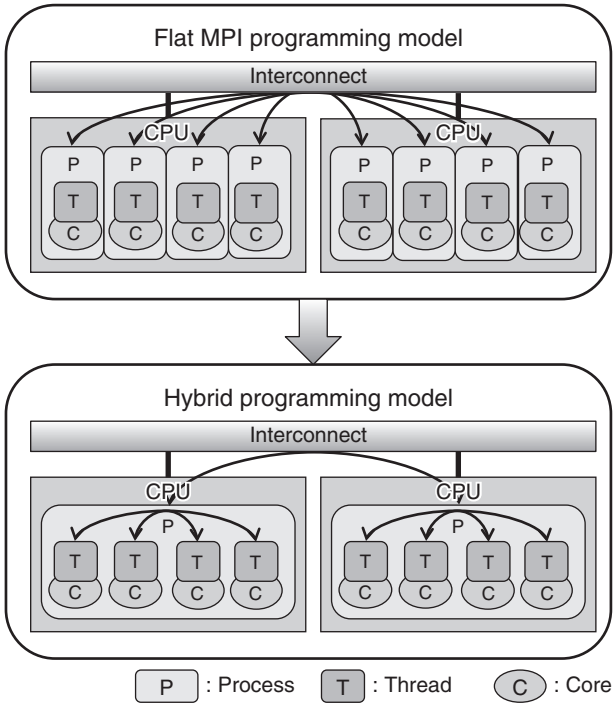

Figure 2
Hybrid parallel model.

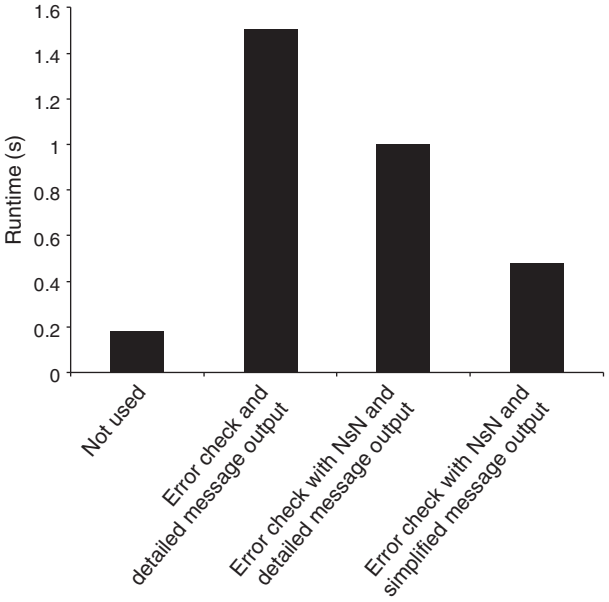

Figure 3
Runtime of new built-in debugging function.

a few tens to a few hundreds of times as much time, the performance has significantly improved and debugging takes only 2.7 times as much at the shortest, which we believe has reached a practical level.

7. Conclusion

To allow thread parallel processing, Fujitsu compilers have long provided automatic parallelization to which the vectorization function of compilers for VPP machines is applied. Thanks to the accumulation of technologies over the years, the change due to the increase of the number of cores to eight with SPARC64 VIIIfx is small, and this has allowed us to focus on enhancing the automatic parallelization itself, and high parallelization efficiency has been successfully achieved.

Fujitsu has announced 16-core SPARC64 IXfx architecture, a successor to SPARC64 VIIIfx, and has already started to offer PRIMEHPC FX10, a system incorporating this CPU. Our division has implemented the compiler technology that has been commercialized with the language package for the K computer also in a product for PRIMEHPC FX10 called Technical Computing Suite.

The demand for faster supercomputers is ever increasing and the extension and expansion of hardware specifications will unavoidably continue in the future. We can expect future high performance supercomputers to have even more cores. Fundamental breakthroughs will eventually become necessary. Based on the successful experience of becoming the world's number one with the K computer, we intend to overcome these challenges and continue to offer systems and products that meet users' needs.

Koutarou Taki
Fujitsu Ltd.
Mr. Taki is currently engaged in compiler optimization development.

Hitoshi Murai
RIKEN
Dr. Murai is engaged in research and development of programming environment.

Manabu Matsuyama
Fujitsu Ltd.
Mr. Matsuyama is currently engaged in compiler C/C++ language front-end development.

Kazuo Minami
RIKEN
Mr. Minami is engaged in research and development of application software for high parallelization and high performance.