The Compilers and MPI Library for SX-9

YOKOYA Yuji, KUDOH Yoshihiro, HAYASAKA Takeshi
Jesper Larsson TRAEFF, Hubert RITZDORF, HAYASHI Yasuharu

Abstract

The SX-9 provides FORTRAN90/SX and C++/SX, respectively, a Fortran compiler and a C/C++ compiler; both of which feature excellent optimization, vectorization and parallelization functions. HPF/SX V2, (a compiler for HPF (High Performance Fortran), which is the *de facto* standard language for distributed parallel processing), and MPI/SX and MPI2/SX, (fully compliant with the distributed parallel processing interfaces MPI-1.3 and MPI-2.1 specifications) are also provided. This paper is intended to introduce the functions and features of the speed-up technology adopted in these programming interfaces for the SX-9.

Keywords

SX Series, supercomputer, compiler, MPI, HPF, Fortran, SX-9, C, C++ language, shared parallel, distributed parallel

1. Introduction

The Supercomputer SX-9 provides a powerful hardware mechanism that can respond to the needs of large-scale, ultra-high-speed S&T computations, particularly in the field of numerical simulations. The compiler and runtime library capabilities are very important for making full use of the powerful hardware and obtaining maximum performances.

In order to enable the hardware to manifest its high level performance, the SX-9 provides FORTRAN90/SX and C++/SX compilers that have further advanced the excellent optimization, auto vectorization and auto parallelization functions already proven in the previous SX Series. Also provided are the HPF/SX V2, MPI/SX and MPI2/SX distributed parallel program interfaces that are tuned especially for the SX-9.

2. FORTRAN90/SX and C++/SX

The FORTRAN90/SX is a compiler in full-compliance with Fortran standard ISO/IEC 1539 and JIS X3001:1998. It also supports functions provided by the latest Fortran standard (usually called Fortran 2003); these include interoperability with C, exceptions and IEEE arithmetic.

C++/SX is a compiler product containing the combined functionalities of C and C++ compilers. The C compiler is compliant with the international C language standard ISO/IEC 9899:1999 and also permits the use of the complex type that has been newly added to the standard. The C++ compiler is compliant to international C++ language standard ISO/IEC 14882:1998 and provides exception handling, runtime type identification and standard template libraries.

Both FORTRAN90/SX and C++/SX support OpenMP Version 2.5, which is the standard API for shared memory type parallel processing.

In the following sections, we introduce the functions and features of these two compilers that have been enhanced for the SX-9.

2.1 Instruction Sorting across Branch Instructions

The SX-9 has a total of 6 vector pipelines including two for multiplication, two for addition, one for logic operations and one for division/square-root operations. It is capable of simultaneous execution of two multiplications, two vector additions, one vector logic operation and one vector division (or square-root operation). It is important to let these vector pipelines operate in parallel whenever possible in order to obtain the maximum performance from the SX-9.

FORTRAN90/SX and C++/SX simulate the operations of the SX-9 hardware and reorder the instruction sequences into optimum order for maximum utilization of the hardware resources. In doing this, they make it possible to sort instructions in a wide range of instruction sequences across branch

```
X(:)= 2.0*ABS(A(:))+3.0*ABS(B(:))+ 4.0*C(:)
```

Image of Vector Instruction Sequence

```
vand   vt1,0x7fffffff,A
vfmul  vt2,2.0,vt1
vand   vt3,0x7fffffff,B
vfmul  vt4,3.0,vt3
vfadd  vt5,vt2,vt4
vfmul  vt6,4.0,C
vfadd  X,vt5,vt6
```

※ vfmul : Vector multiplication
vfadd : Vector addition
vand : Vector AND (corresponds to ABS)

Fig. 1 Array assignment statements and vector instruction sequence of fortran.

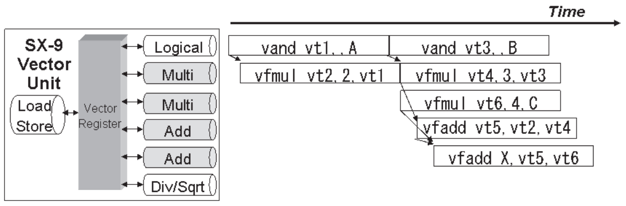


Fig. 2 Image of operation of vector instruction sequence of the SX-9.

instructions and to find more simultaneously-executable instructions.

As an example, **Fig. 1** shows array assignment statements of Fortran and an image of the vector instruction sequence that can be generated from them.

The example has a vector instruction sequence containing two vector AND, three vector multiplication and two vector addition operations for obtaining absolute value ABS from the array assignment statements.

The compilers simulate the operation of the SX-9's vector unit, assign the vector registers and reorder instructions, so that as many vector pipelines as possible can operate simultaneously. Additionally, an instruction sequence is generated with which a total of five vector operations including one AND operation, two multiplications and two additions can be executed simultaneously as shown in **Fig. 2** .

As seen here, the compilers of the SX-9 apply instruction reordering to increase the opportunities of parallel operations of vector pipelines over a wide range, in order to let the SX-9 manifest its high level of operation performance optimally.

2.2 Vector Data Buffering Function

High-speed access to arrays in memory is required for the execution of meteorological and flow analysis programs. Generally, the method for handling these programs is to install a memory cache between the memories and registers. However, as the hardware has the tendency of saving all data in the caches, the consequent insufficiency of cache capacity may mean that for extremely large arrays or when other data is large that really necessary arrays may sometimes be unable to enter the caches. The projected cache benefits may thus become unavailable.

To deal with such a situation, the SX-9 has a hardware mechanism called ADB (Assignable Data Buffer) added between the memory and vector register. ADB can buffer data selectively using the vector data buffering function of the compiler. For instance, when ADB is used to buffer only the frequently accessed arrays, high-speed access to the arrays will be possible for a long period of time.

When the destination of an array or pointer is specified using the on_adb compiler directive option, the loading and storage of the vectors are performed via ADB. At this time, data is buffered in ADB at the first vector loading or storage. When the same data is used later on, the data buffered in ADB is loaded. As data loading from ADB is possible at a higher speed than from memory, this function can increase the speed of second and later vector loading operations.

Fig. 3 shows an example of a specification of an on_adb compiler directive option in a Fortran program. The process-

```
      subroutine sub
      common a(4096),b(4096),c(4096)
!cdir on_adb(a)
!cdir on_adb(c)
      do i=1,4096
            = a(i) + ...          1)
         c(i) = ...               2)
            = a(i) * ...          3)
      enddo
      ...
!cdir on_adb(a)
!cdir on_adb(c)
      do i=1,4096
         b(i) = a(i) + c(i)   4)/5)
      enddo
      end subroutine
```

Fig. 3 Specification in Fortran program.

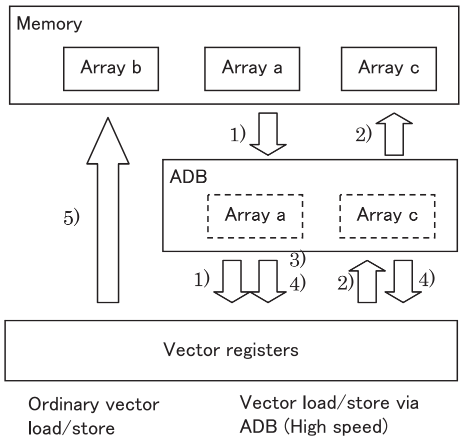


Fig. 4 Operation of ADB.

```
      ...
!cdir on_adb(a)
!cdir on_adb(b)
      do i=1,10000
         a(i) = ... Stored as continuous vectors
         b(i) = ... Stored as continuous vectors
      enddo
      ...
!cdir on_adb(a)
!cdir on_adb(b)
      do i=1,10000
         x(i) = a(idx(i)) + b(idx(i))
         y(i) = a(idx(i)) - b(idx(i))
      enddo
      ...
```

Fig. 5 Improvement of list vector loading performance.

```
     ...
    for (j = 0; j < 10000; j++) {
#pragma cdir on_adb(a)
        for (i = 0; i < 256; i++) {
            a[i] += b[j][i];
        }
    }
    ...
    i = 0;
    while (x[i] >= 0) {         // Unvectorizable
        x[i] = a[i] + ...
        i++;
    }
    ...
```

Fig. 6 Improvement of the scalar loading performance of the array elements.

ing is identical with both programs. The operation of the vector data buffering function is also similar to the above.

Fig. 4 shows the operation of the vector data buffering function when the program shown in Fig. 3 is executed.

1) According to the "on_adb(a)" specification immediately before the first loop, the data of array a is buffered in ADB during vector loading of array a.
2) According to the "on_adb(c)" specification, the data of array c is buffered in ADB during vector storing of array c.
3) When array a is referenced for the second time, the data of array a, buffered in ADB is used in vector loading.
4) In referencing arrays a and c in the second loop, the data of arrays a and c that is buffered in ADB is used in the vector loading.
5) Since array b is not specified as using the on_adb directive option, the vector storing of the array is performed directly to the memory without using ADB. In this operation, the data of array b is not buffered in ADB.

Effective use of the vector data buffering function also makes it possible to load list vectors (indirectly referenced vectors) and to improve the scalar loading performance of the array elements.

(1) Improvement of List Vector Loading Performance

When an array used as a list vector is specified using the on_adb compiler directive option and is buffered in ADB, the subsequent data loading will be possible at a higher speed.

Fig. 5 shows an example with a Fortran program. In this example, when arrays a and b are stored as continuous vectors, their data is buffered in ADB. This procedure can improve the memory access performance when referencing list vectors a(idx(i)) and b(idx(i)) in the second loop.

Performance improvements using the same technique are also possible for the C/C++ programs.

(2) Improvement of the Scalar Load Performance of Array Elements

When performing scalar loading of array elements if the array element has already been buffered in ADB, the data is loaded from ADB. For example, when several elements of an array that has been loaded or stored in a vectorized loop are used immediately after the loop, the array elements can be loaded at a higher speed if the array has previously been specified using the on_adb compiler and buffered in ADB.

Fig. 6 shows an example with a C program. In this exam-

ple, the data of array a, is buffered in ADB during vector storing of the array. Consequently, when the elements of the array are referenced in the second loop that cannot be vectorized, the data buffered in ADB is used in scalar loading so that the memory access performance for a[i] in the second loop can be improved.
Performance improvements using the same technique are also possible with the Fortran programs.
In this way, selecting the data to be buffered in ADB optimally and intending to use the buffered data repeatedly whenever possible enables effective use of the vector data buffering function and improves the memory access performance. In the future, we intend to enhance the compilers so that they can select the data to be buffered in ADB and thereby facilitate an increase in the execution speeds of programs.

3. MPI/SX and MPI2/SX

MPI/SX and MPI2/SX fully implement the Message Passing Interface specifications MPI-1.3 and MPI-2.1, as defined by the MPI Forum. MPI is used for message passing type distributed parallel processing, and interfaces are defined for Fortran, C and C++ programming languages.

Since data communication in message passing parallel programs is a major factor determining the efficiency of parallel computation, an efficient MPI library is important and must provide high-speed data transfer. With the SX-9 we have enhanced the IXS system for inter-node communication, and enable configurations with up to 16 RCUs which significantly improve the maximum hardware communication throughput. In the following sections, we briefly discuss the speed increase achieved by MPI/SX and MPI2/SX with the enhanced IXS.

3.1 Improvement of Throughput Performance

For point-to-point communication in which data is transferred between two processes, the enhanced IXS is utilized by dynamically dividing data into smaller blocks (if it is larger than a certain minimum block size). The transfer of these blocks is distributed optimally among multiple RCUs and executed simultaneously. This improves the throughput performance which approaches the peak performance of the SX-9 hardware (**Fig. 7**).

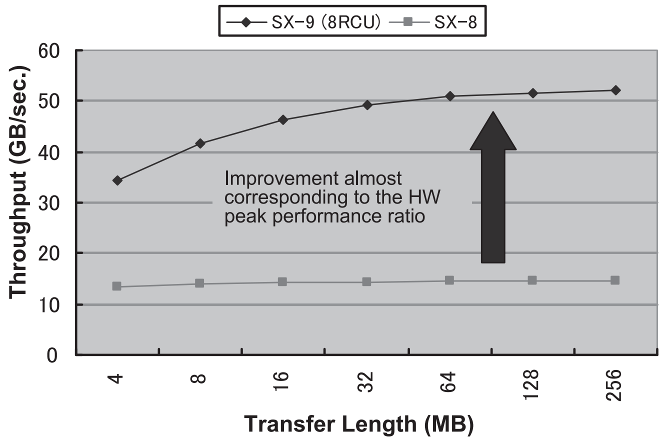


Fig. 7 Comparison of inter-node communication throughput performances.

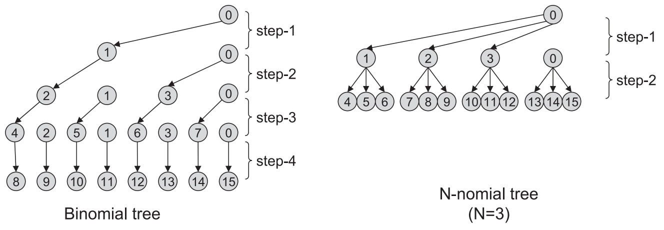


Fig. 8 Binomial and N-nomial communication tree structures and data transfer (Example of broadcast communication).

3.2 Enhancement of the Collective Communication Algorithm

The collective communication operations, in which multiple processes participate in simultaneous data transfers, are implemented using various tree-like communication structures built over the SX-9 nodes. For small data, binomial trees usually give the best performance. In order to utilize efficiently the multiple RCUs of the SX-9 for small data, these structures are generalized to N-nomial trees (N can be up to 16), where data are sent simultaneously by multiple RCUs. This reduces the number of stages required for a data transfer to all nodes(**Fig. 8**). For large data, the techniques for point-to-point communication are used to improve the individual transfers, in combination with for instance pipelined binary trees. By these techniques we have succeeded in increasing the communication speed by full use of the hardware properties from the latency level of small transfer sizes to the throughput level of the large transfer sizes.

4. HPF/SX V2

HPF (High Performance Fortran) is an extension of Fortran and is the *de facto* standard language for distributed parallel processing.

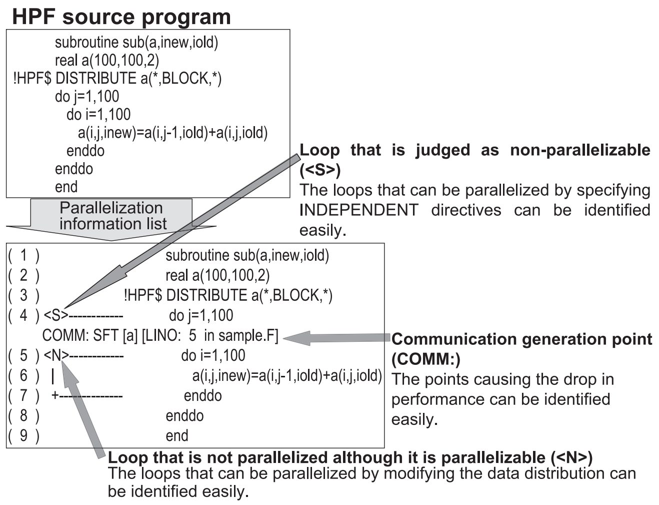
In distributed parallel programming, it is necessary to consider three factors, which are data distribution, computation partitioning and communication. When HPF is used, computation partitioning and generation of communication can be performed automatically by simply inserting comment-line directives for specifying the data distribution method in an existing Fortran program. This allows even beginners to develop distributed parallel programs easily.

HPF/SX V2 is a compiler supporting HPF2.0, and major parts of the HPF Approved Extensions and HPF/JA extensions. It features enhancement of the productivity/maintainability improvement functions in distributed parallel programming, which are special features of HPF, for the SX-9 Series. In the following sections, we will introduce some of the most important improved functions.

4.1 Tuning Support Functions

(1) Parallelization Information List

In addition to the existing diagnostic messages, the parallelization information list indicating the parallelization status and communication generation status can be generated by specifying a compile-time option. The parallelization information list makes it easy to extract the points where communications are generated, the loops that are judged not to be parallelizable by the compiler and the loops that are not parallelized as well as the parallelized loops, so that the users can easily identify the points to be tuned (**Fig. 9**).



```
        subroutine sub(a,inew,iold)
        real a(100,100,2)
!HPF$ DISTRIBUTE a(*,BLOCK,*)
        do j=1,100
          do i=1,100
            a(i,j,inew)=a(i,j-1,iold)+a(i,j,iold)
          enddo
        enddo
        end
```

```
( 1 )                 subroutine sub(a,inew,iold)
( 2 )                 real a(100,100,2)
( 3 )               !HPF$ DISTRIBUTE a(*,BLOCK,*)
( 4 ) <S>------------    do j=1,100
      COMM: SFT [a] [LINO:  5  in sample.F]
( 5 ) <N>------------      do i=1,100
( 6 )  |                     a(i,j,inew)=a(i,j-1,iold)+a(i,j,iold)
( 7 )  +-------------      enddo
( 8 )                 enddo
( 9 )                 end
```

Loop that is judged as non-parallelizable (<S>)
The loops that can be parallelized by specifying INDEPENDENT directives can be identified easily.

Communication generation point (COMM:)
The points causing the drop in performance can be identified easily.

Loop that is not parallelized although it is parallelizable (<N>)
The loops that can be parallelized by modifying the data distribution can be identified easily.

Fig. 9 Parallelization information list.

(2) Procedure Boundary Communication Detection

When the distribution specified to a dummy or an actual argument is inappropriate, the performance is degraded due to the generation of copies accompanied with communications at the invocation of and return form the procedure. HPF/SX V2 is able to output the information on the argument and procedure names when a communication is generated on a procedure boundary by specifying a runtime option.

4.2 Debugging Support Functions

(1) Array References out of Declared Bounds Detection

An option is available to output the array names and program line numbers containing array references out of declared bounds. This function does not hinder vectorization and parallelization in principle, so it can be used to check array references out of declared bounds in a practical execution time.

(2) Inter-procedure Array Characteristic Inconsistency Detection

In HPF, the characteristics such as the shapes and types of the variables associated with each other across procedures, namely common block variables and arguments, should in principle be identical. An option is available to detect inconsistency between these variables and output the names of the variables and procedures at runtime.

5. Conclusion

This paper introduced the compilers and MPI libraries of the SX-9. In the future, we intend to optimize and enhance the language functions of FORTRAN90/SX, C++/SX and HPF/SX V2. We will also increase the system speed by tuning the performances of MPI/SX and MPI2/SX so that the SX-9 can offer its optimum hardware performance.

The Compilers and MPI Library for SX-9

Authors' Profiles

YOKOYA Yuji
Expert,
1st Computers Software Division,
Computers Software Operations Unit,
NEC Corporation

KUDOH Yoshihiro
Expert,
1st Computers Software Division,
Computers Software Operations Unit,
NEC Corporation

HAYASAKA Takeshi
Assistant Manager,
1st Computers Software Division,
Computers Software Operations Unit,
NEC Corporation

Jesper Larsson TRAEFF
Chief Researcher,
IT Research Division,
NEC Laboratories Europe

Hubert RITZDORF
Research Fellow,
IT Research Division,
NEC Laboratories Europe

HAYASHI Yasuharu
Assistant Manager,
1st Computers Software Division,
Computers Software Operations Unit,
NEC Corporation