§26. Evaluation of the XcalableMP Parallel Language on the Plasma Simulator System

Nakao, M. (RIKEN AICS)

i) Objective MPI (Message Passing Interface) has been widely used to develop parallel applications on distributed memory system such as the Plasma Simulator system, Fujitsu PRIMEHPC FX100. However, the programming cost of MPI is high. To tackle this problem, the XcalableMP (XMP) parallel language has been proposed[1, 2]. XMP is a directive-based language extensions of C and Fortran, which enables programmers to develop parallel applications with high performance easily. I have developed an omni compiler[3], which is an XMP compiler as a reference implementation. The objectives of this study are to port the omni compiler[3] to FX100 and to evaluate its performance.

ii) Porting I create a setting file for FX100 in the omni compiler so that a user can build the omni compiler easily. When building the omni compiler, a user only indicates the machine's name. In the setting file, Fujitsu compiler's optimal options are set to build a runtime of the omni compiler to achieve high performance.

The following explains how to install the omni compiler onto FX100.

1. Get the latest omni compiler from the official site[3]
2. Expand the omni compiler

```
$ tar xfj omni-compiler.tar.bz2
```

3. Indicate the machine's name

```
$ ./configure --target=FX100-linux-
  gnu
```

4. Build and install the omni compiler

```
$ make; make install
```

iii) Performance Evaluation To evaluate a performance of the omni compiler on FX100, I implemented the Himeno benchmark[4]. The Himeno Benchmark evaluates a performance of incompressible fluid analysis code using the Jacobi iteration method. The reason which I select this benchmark is a good example of a stencil application benchmark which is widely used in a computational science field.

Fig. 1 shows a part of code written in XMP Fortran. The **nodes**, **template**, **distribute**, and **align** directives distribute the array p onto each process. The **shadow** directive sets the width of the overlapped region. The **loop** directive parallelizes the following loop statement. Moreover, the OpenMP **parallel** directive parallelizes the parallelized loop statement. The **reflect** directive synchronizes data of the overlapped region onto the neighboring process.

```
real p(imax,jmax,kmax)
!$xmp nodes n(8,8)
!$xmp template t(imax,jmax,kmax)
!$xmp distribute t(*,block,block) onto n
!$xmp align (*,j,k) with t(*,j,k) :: p
!$xmp shadow p(0,2:1,2:1)
...
!$xmp loop (J,K) on t(*,J,K)
!$omp parallel do private(S0, ...)
DO K = 2, kmax-1
  DO J = 2, jmax-1
    DO I = 2, imax-1
      S0 = a(I,J,K,1)*p(I+1,J,K)+..
             :
!$xmp reflect (p)
```

Fig. 1: Part of code in XcalableMP

This parallelization is so straightforward that a programmer only adds XMP directives into the sequential Himeno Benchmark. The source lines of codes (SLOC) of XMP Himeno Benchmark is 137 where nineteen XMP directives are used. Hence, the SLOC of the sequential Himeno Benchmark is 118. Besides, the SLOC of the MPI Himeno Benchmark is 380. These results shows that XMP has a good productivity.

To evaluate performance of the Himeno benchmark, I used 32 threads and one process per compute node of FX100 and strong-scaling. Fig. 2 shows the result, which indicates the omni compiler has a good performance.

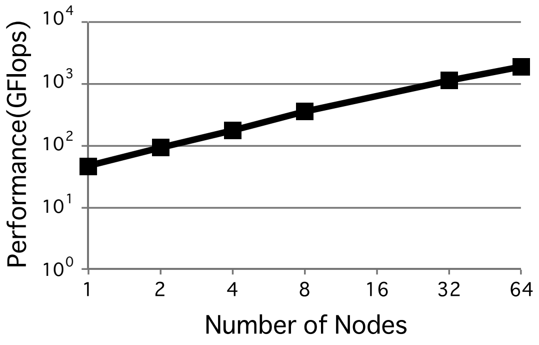

Fig. 2: Performance result

1) http://xcalablemp.org

2) Masahiro Nakao, et al. :Productivity and Performance of the HPC Challenge Benchmarks with the XcalableMP PGAS language, 7th International Conference on PGAS Programming Models, 2013.

3) http://omni-compiler.org

4) http://accc.riken.jp/supercom/himenobmt/