Center for Computational Sciences, University of Tsukuba
www.ccs.tsukuba.ac.jp

XcalableMP : Directive-Based Language eXtention for Scalable and Performance-Aware Parallel Programming

What is XcalableMP?

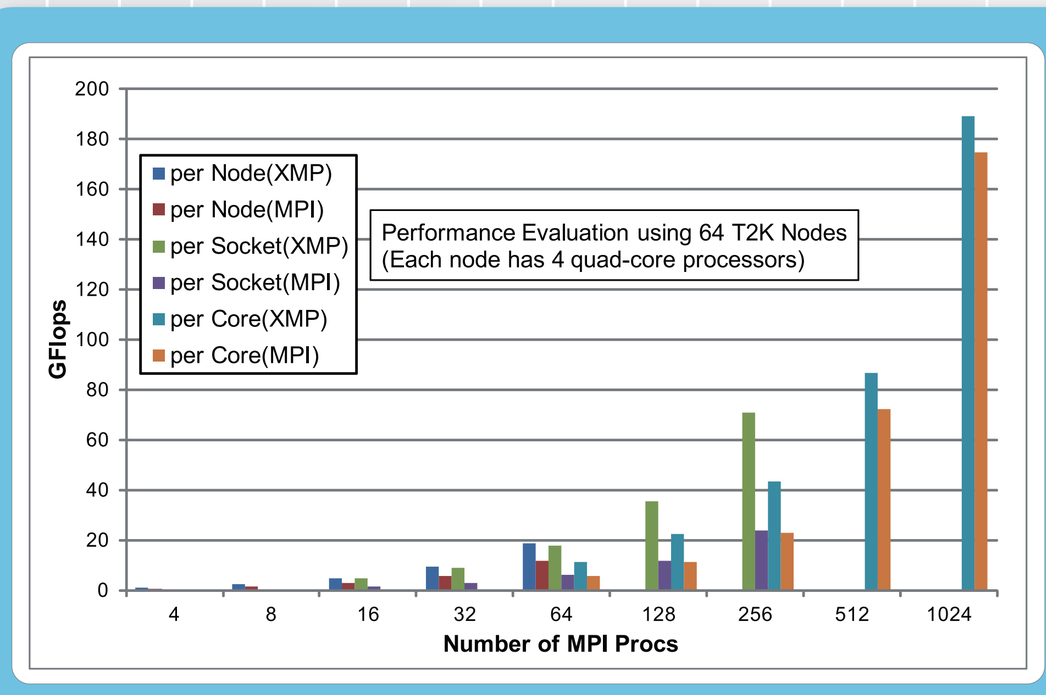
Although MPI is the de facto standard for parallel programming on distributed memory systems, writing MPI programs is often a time-consuming and complicated process. XcalableMP is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and allows them to tune the performance by having minimal and simple notations. The specification has been being designed by the XcalableMP Specification Working Group which consists of members from academia, research labs and industries in Japan.

- XcalableMP supports typical parallelization based on the data parallel paradigm and work mapping under the “global-view” programming model, and enables parallelizing the original sequential code with minimal modification by using simple directives, like OpenMP. Many ideas on “global-view” programming are inherited from HPF (High Performance Fortran).
- The important design principle of XcalableMP is “performance-awareness”. All actions of communication and synchronization are taken by directives, instead of using automatic parallelizing compilers. The user should be aware of what XcalableMP directives are doing in the execution model on the distributed memory architecture.
- XcalableMP also includes CAF-like PGAS (Partitioned Global Address Space) features such as “local-view” programming.
- XcalableMP APIs are defined in C and Fortran 95.

Code Examples (Laplace Equation Solver)

XMP

```
!XMP$ nodes p(XPROCS, YPROCS)
!XMP$ template t(XSIZE, YSIZE)
!XMP$ distribute t(block, block) onto p
!XMP$ align (i, j) with t(i, j) :: u, uu
!XMP$ shadow uu(1:1, 1:1)
. . .
do k = 1, niter
!XMP$ loop (x, y) on t(x, y)
  do x = 2, YSIZE - 1
    do y = 2, XSIZE - 1
      uu(x, y) = u(x, y)
    end do
  end do
!XMP$ reflect uu
!XMP$ loop (x, y) on t(x, y)
  do x = 2, YSIZE - 1
    do y = 2, XSIZE - 1
      u(x, y) = (uu(x-1, y) + uu(x+1, y) +
                 uu(x, y-1) + uu(x, y+1)) / 4.0
    end do
  end do
end do
```



The parallel version of the Laplace equation solver can be easily written by using shadow pragmas in XMP. The performance is equivalent (or even better in some cases) to the MPI version.

Performance of XcalableMP Codes

HPF

```
!HPF$ processsors p(XPROCS, YPROCS)
!HPF$ template t(XSIZE, YSIZE)
!HPF$ distribute t(block, block) onto p
!HPF$ align (i,j) with t(i,j) :: u, uu
!HPF$ shadow uu(1,1)
. . .
do k = 1, niter
!HPF$ independent
  do y = 2, YSIZE - 1
!HPF$ on home(t(:, y))
    do x = 2, XSIZE - 1
!HPF$ on home(t(x, y))
      uu(x,y) = u(x,y)
    end do
  end do
!HPF$ reflect uu
!HPF$ independent
  do y = 2, YSIZE - 1
!HPF$ on home(t(:, y))
    do x = 2, XSIZE - 1
!HPF$ on home(t(x, y))
      u(x,y) = (uu(x-1,y) + uu(x+1,y) + uu(x,y-1) + uu(x,y+1)) / 4.0
    end do
  end do
end do
```

MPI

```
do k = 1, niter
  do y = 1, y_data_size
    do x = 1, x_data_size
      uu(x,y) = u(x,y)
    end do
  end do
  if (x_rank.ne.0)
    call mpi_send(uu, . . . , x_rank–1, . . .)
  end if
  if (x_rank.ne.(x_comm_size-1))
    call mpi_recv(uu, . . . , x_rank+1, . . .)
  end if
  if (x_rank.ne.(x_comm_size-1))
    call mpi_send(uu, . . . , x_rank+1, . . .)
  end if
  if (x_rank.ne.0)
    call mpi_recv(uu, . . . , x_rank-1, . . .)
  end if
  . . .
  do y = 1, y_data_size
    do x = 1, x_data_size
      u(x,y) = (uu(x-1,y) + uu(x+1,y) + uu(x,y-1) + uu(x,y+1))/4.0
    end do
  end do
end do
```

CAF

```
double precision u(0:x_data_size+1, 0:y_data_size+1)[XPROCS, *]
double precision uu(0:x_data_size+1, 0:y_data_size+1)[XPROCS, *]
do k = 1, niter
  do y = 1, y_data_size
    do x = 1, x_data_size
      uu(x,y) = u(x,y)
    end do
  end do
  sync all
  if (x_rank.ne.1)
    uu(0, 1:y_size) = uu(x_size+1, 1:y_size)[x_rank-1, y_rank]
  end if
  if (x_rank.ne.XPROCS)
    uu(x_size+1,1:y_size) = uu(0, 1:y_size)[x_rank+1, y_rank]
  end if
  if (y_rank.ne.1)
    uu(1:x_size, 0) = uu(1:x_size, y_size+1)[x_rank, y_rank-1]
  end if
  if (y_rank.ne.YPROCS)
    uu(1:x_size, y_size+1) = uu(1:x_size, 0)[x_rank, y_rank+1]
  end if
  sync all
  do y = 1, y_data_size
    do x = 1, x_data_size
      u(x,y) = (uu(x-1,y) + uu(x+1,y) + uu(x,y-1) + uu(x,y+1))/4.0
    end do
  end do
end do
```

This research is carried out as a part of “Seamless and Highly-productive Parallel Programming Environment for High-performance computing” project funded by Ministry of Education, Culture, Sports, Science and Technology, JAPAN.

www.xcalablemp.org