XcalableMP Ver. 1.0
Highly Productive Parallel Programming Language

Masahiro NAKAO
Center for Computational Sciences
University of Tsukuba
Background

- MPI is widely used as a parallel programming model on distributed memory systems
  - Time-consuming
  - Complicated process
- Another programming model is needed!!
  - High performance
  - Easy to program (High productivity)

Development of XcalableMP (XMP)
e-Science Project

- XMP Working Group designs XMP specification
  - XMP Working Group consists of members from
    - academia: U. Tsukuba, U. Tokyo, Kyoto U. and Kyusyu U.
    - research labs: RIKEN AICS, NIFS, JAXA, JAMSTEC/ES
    - industries: Fujitsu, NEC, Hitachi
  - Specification Version 1.0 is released !! (Nov. 2011)
- University of Tsukuba develops an Omni XMP compiler as a reference implementation
  - the K Computer, CRAY platforms(HECToR), Linux cluster
- Evaluation of Performance and Productivity
What's XcalableMP

XcalableMP is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations. The specification is being designed by XcalableMP Specification Working Group which consists of members from academia and research labs to industries in Japan.

- **Specification of XcalableMP, version 1.0** (Nov. 14, 2011)

If you have any comments and requests, please contact to Prof. Mitsufusa Sato (msato at cs.tsukuba.ac.jp). Your comments and contributions will be appreciated!!

The features of XcalableMP are summarized as follows:

- XcalableMP supports typical parallelization based on the data parallel paradigm and work mapping under "global view programming model", and enables parallelizing the original sequential code using minimal modification with 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, different from automatic parallelizing compilers. The user should be aware of what happens by XcalableMP directives in the execution model on the distributed memory architecture.
- XcalableMP also includes a CAF-like PGAS (Partitioned Global Address Space) feature as "local-view" programming.
- Extention of existing base languages with directives is useful to reduce rewriting and educational costs. XcalableMP APIs are

http://www.xcalablemp.org/
Agenda

- Overview of XMP
- XMP Programming Model
- Evaluation of Performance and Productivity of XMP
Overview of XMP

- XMP is a directive-based language extension like OpenMP and HPF based on C and Fortran95
  - To reduce code-writing and educational costs
- The basic execution model of XMP is SPMD
  - A thread starts execution in each node independently (as in MPI)
- “Performance-awareness” for explicit communication, sync. and work-mapping
  - All actions occur when thread encounters directives or XMP’s extended syntax
  - XMP compiler generates communication only where a user inserts them to facilitate performance tuning
XMP C version

```c
int array[100];
#pragma xmp nodes p(*)
#pragma xmp template t(0:99)
#pragma xmp distribute t(block) onto p
#pragma xmp align array[i] with t(i)

main(){
#pragma xmp loop on t(i) reduction(+:res)
    for(i = 0; i < 100; i++){
        array[i] = func(i);
        res += array[i];
    }
}
```
real a(100)
!$xmp nodes p(*)
!$xmp template t(100)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
:

!$xmp loop on t(i) reduction(+:res)
  do i=1, 100
    a(i) = func(i)
    res = res + a(i)
  enddo
int array[100];

main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    dx = 100/size;
    llimit = rank * dx;
    if(rank != (size -1)) ulimit = llimit + dx;
    else ulimit = 100;

    temp_res = 0;
    for(i=llimit; i < ulimit; i++){
        array[i] = func(i);
        temp_res += array[i];
    }

    MPI_Allreduce(&temp_res, &res, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    MPI_Finalize();
}
Agenda

- Overview of XMP
- **XMP Programming Model**
- Evaluation of Performance and Productivity of XMP
Programming Model

- Global View Model (like as HPF)
  - programmer describes data distribution, work mapping, communication and sync. by using directives
  - supports typical techniques for data-mapping and work-mapping
  - rich communication and sync. directives, such as “shadow”, “reflect” and “gmove”

- Local View Model (like as Coarray Fortran)
  - enables programmer to transfer data by using one-sided comm. easily

WPSE2012@Kobe, Japan
The directives define a data distribution among nodes:

```c
#pragma xmp nodes p(4)
#pragma xmp template t(0:15)
#pragma xmp distribute t(block) on p
#pragma xmp align a[i] with t(i)
```
Parallel Execution of loop

- Loop directive is inserted before loop statement

```c
#pragma xmp loop on t(i)
for(i=2;i<=10;i++){...}
```

Execute “for” loop in parallel with affinity to array distribution

Each node computes Red elements in parallel
Example of Data Mapping

- **block**
- **cyclic**
- **block-cyclic (block size = 3)**
- **generalized-block**

Legend:
- node1
- node2
- node3
- node4
#pragma xmp distribute t(block) onto p
#pragma xmp align a[i][*] with t(i)

#pragma xmp distribute t(block) onto p
#pragma xmp align a[*][i] with t(i)

#pragma xmp distribute t(block,cyclic) onto p
#pragma xmp align a[i][j] with t(i,j)
Local View Model

- One-sided communication for local data (Put/Get)
- In XMP Fortran, this function is compatible with that of CAF
- In XMP C, C is extended to support the *array section* notation
- Uses GASNet/ARMCI, which are high-performance communication layer

```c
#pragma xmp coarray b: [*]
        :
if (me == 1)
a[0:3] = b[3:3]:[2];    // Get
```

```
0 2
a[]

b[]
```

node 1                     node 2

```
4 6
3 5
```

base length node number
## Other directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduction</td>
<td>Aggregation</td>
</tr>
<tr>
<td>bcast</td>
<td>Broadcast</td>
</tr>
<tr>
<td>barrier</td>
<td>Synchronization</td>
</tr>
<tr>
<td>shadow/reflect</td>
<td>Create shadow region/sync.</td>
</tr>
<tr>
<td>gmove</td>
<td>Transfer for distributed data</td>
</tr>
</tbody>
</table>
shadow/reflect directives

- If neighbor data is required, then only shadow area can be synchronized
  
  ```
  #pragma xmp shadow a[1:1]
  ```

- Shadow directive defines width of shadow area

- \( b[i] = \text{array}[i-1] + \text{array}[i+1]; \)

```plaintext
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[a[]]:
```

Node 1

Node 2

Node 3

Node 4

Reflect directive is to synchronize only shadow region

```plaintext
#pragma xmp reflect (array)
```
Gmove directive

- Communication for distributed array
  - uses array section notation in XMP C
  - Programmer doesn’t need to know where each data is distributed

```c
#pragma xmp gmove
a[2:4] = b[3:4];
```

![Diagram showing Gmove directive and array distribution across nodes]
Agenda

- Overview of XMP
- XMP Programming Model
- Evaluation of Performance and Productivity of XMP
Evaluation

- Examines the performance and productivity of XMP
- Implementations of benchmarks
  - NAS Parallel Benchmarks
    - CG, EP, IS, BT, LU, FT, MG
  - HPC Challenge Benchmarks
    - HPL, FFT, RandomAccess, STREAM
  - Finalist of HPCC Class2 in SC10 and SC09
- Laplace Solver
- Himeno Benchmark, and so on
Environment

- T2K Tsukuba System
  - CPU: AMD Opteron Quad-Core 8356 2.3GHz (4 sockets)
  - Memory: DDR2 667MHz 32GB
  - Network: Infiniband DDR(4rails) 8GB/s
Laplace Solver

- "thread" clause for multicore cluster
- shadow/reflect directive

Performance

\[
\text{for}(y = 1; y < N-1; y++) \\
\text{for}(x = 1; x < N-1; x++) \\
\text{tmp}_a[y][x] = a[y][x];
\]

Productivity

<table>
<thead>
<tr>
<th>Number of Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
</tr>
<tr>
<td>158</td>
</tr>
</tbody>
</table>

WPSE2012@Kobe, Japan
Conjugate Gradient

- Local view programming
- reduction directives for local variable

Performance

```
#pragma xmp coarray w, w1:*]
        :
for( i=n_cols; i>=0; i-- ){
    w[l:count[k][i]] += w1[m:count[k][i]][k];
```

Productivity
High Performance Linpack

- block-cyclic distribution
- BLAS Lib. is used directly from distributed array

**Performance**

```
#pragma xmp distribute \  
t(cyclic(NB), cyclic(NB)) onto p  
#pragma xmp align A[i][j] with t(j, i)  
:  
cblas_dgemm(..., &A[y][x], ...);
```

**Productivity**

<table>
<thead>
<tr>
<th>Number of Lines</th>
<th>XMP</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>8800</td>
<td>201</td>
</tr>
</tbody>
</table>

Graph showing Performance (GFlops) vs. Number of CPUs for XMP and Original(MPI).
International Collaboration

- **Interface of XMP program profile to Scalasca**
- **Scalasca** is a software tool that supports the performance optimization of parallel programs
- **Scalasca** is developed by JÜLICH and German Research School for Simulation Sciences

```
#pragma xmp gmove profile
...
#pragma xmp loop on t(i) profile
```
Summary & Future work

- XcalableMP was proposed as a new programming model to facilitate program parallel applications for distributed memory systems

- Evaluation of Performance and Productivity
  - Performance of XMP is compatible with that of MPI
  - Productivity of XMP is higher than that of MPI

- Future work
  - Performance evaluation on larger environment
  - For accelerators (GPU, etc), Parallel I/O, and Interface of MPI library