XcalableMP and XcalableACC for Productivity and Performance in HPC Challenge Award Competition

Masahiro Nakao,†‡ Hitoshi Murai,† Hidetoshi Iwashita†
Takenori Shimosaka,† Akihiro Tabuchi,∗ Taisuke Boku‡*
Mitsuhisa Sato†‡∗

† RIKEN Advanced Institute for Computational Science, Japan
‡ Center for Computational Sciences, University of Tsukuba
∗ Graduate School of Systems and Information Engineering, University of Tsukuba
Outline

1. XcalableMP (XMP) for cluster systems (14min.)
2. XcalableACC (XACC) for accelerator cluster systems (6min.)

Extension of XMP using OpenACC

Sorry !!, work-in-progress

The submission report is available at http://xcalablemp.org
What is XcalableMP (XMP)?

Directive-based language extensions of Fortran and C

- By XMP specification working group of PC cluster consortium (SC Booth#2924)
- Version 1.2.1 specification available ([http://xcalablemp.org](http://xcalablemp.org))

Support two memory models

- Global-view (HPF-like data/work mapping directives)
- Local-view (coarray)

Implementation of Compiler

- Omni XMP Compiler version 0.9 ([http://omni-compiler.org](http://omni-compiler.org))
- Platforms: Fujitsu the K computer and FX10, Cray XT/XE, IBM BlueGene, NEC SX, Hitachi SR, Linux clusters, etc.
int a[MAX];

#pragma xmp nodes p(4)
#pragma xmp template t(0:MAX-1)
#pragma xmp distribute t(block) on p
#pragma xmp align a[i] with t(i)

main(){
    int i, j, res = 0;

    #pragma xmp loop on t(i) reduction(+:res)
    for(i = 0; i < MAX; i++){
        a[i] = func(i);
        res += array[i];
    }
}

Data distribution

add to the serial code: incremental parallelization

Work mapping and data synchronization
Code example (Local-view)

```c
double a[100]:[*], b[100]:[*];
int me = xmp_node_num();

if(me == 2)
    a[:][1] = b[:];

if(me == 1)
    a[0:50] = b[0:50]:[2];
```

Coarray synax in XMP/C

```
array_name[start:length]:[node_number];
```

XMP/Fortran is upward compatible with Fortran 2008
Results and Machine

Summary

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Nodes</th>
<th>Performance (/peak)</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPL Ver. 1</td>
<td>16,384</td>
<td>971 TFlops (46.3%)</td>
<td>313</td>
</tr>
<tr>
<td>HPL Ver. 2</td>
<td>4,096</td>
<td>423 TFlops (80.7%)</td>
<td>426</td>
</tr>
<tr>
<td>FFT</td>
<td>82,944</td>
<td>212 TFlops (2.0%)</td>
<td>205</td>
</tr>
<tr>
<td>STREAM</td>
<td>82,994</td>
<td>3,583 TB/s (67.5%)</td>
<td>69</td>
</tr>
<tr>
<td>RandomAccess</td>
<td>16,384</td>
<td>254 GUPs</td>
<td>253</td>
</tr>
</tbody>
</table>

Four HPCC Benchmarks

The K computer: 82,944 nodes

- SPARC64 VIIIfx Chip, 128 GFlops
- DDR3 SDRAM 16GB, 64GB/s
- Tofu Interconnect
  - 6D mesh/torus network
  - 5GB/s x 4links x 2

http://www.aics.riken.jp/jp/outreach/photogallery.html
HPL version 1

- Source lines of Code (SLOC) is 313, written in XMP/C
- Block-Cyclic Distribution

```c
double A[N][N];
#pragma xmp nodes p(P,Q)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```

Programmer can use BLAS for distributed array.

- Panel Broadcast by using `gmove` directive

```c
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
    :
#pragma xmp gmove
A_L[k:len][0:NB] = A[k:len][j:NB];
```
HPL version 2

- SLOC is **426**, written in XMP/C
- "Lookahead algorithm" by using `gmove` directive with `async` clause
  
  Overlap communication and calculation

```c
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
#pragma xmp gmove async(1)
A_L[k:len][0:NB] = A[k:len][j:NB];
for(m=j+NB;m<N;m+=NB){
  for(n=j+NB;n<N;n+=NB){
    cblas_dgemm(&A[m][n], ..);
    if(xmp_test_async(1)){
      // receive A[k:len][j:NB];
    :}
  :}
```

Confirm whether data with async clause comes or not.
Performance of HPL

XMP-HPL Version 2 has a good scalability. Sorry, the measurement in 16,384 nodes is late for this BoF.
RandomAccess

- SLOC is **253**, written in XMP/C
- **Local-view programming with XMP/C coarray syntax**
- The XMP RandomAccess is iterated over sets of CHUNK updates on each node

```c
u64Int recv[LOGPROCS][RCHUNK+1]:[*];
...
for (j = 0; j < logNumProcs; j++) {
    recv[j][0:num]:[i_partner] = send[i][0:num];
    #pragma xmp sync_memory
    #pragma xmp post(p(i_partner), 0)
    #pragma xmp wait(p(j_partner))
}
```

A point-to-point synchronization is specified with the XMP’s post and wait directives to realize asynchronous behavior of this algorithm
Performance of RandomAccess

Last year, to implement the post/wait directives, XMP uses MPI_Send/Recv. This year, to implement them, XMP uses RDMA of the K computer.
FFT and STREAM

Code cleanup and performance improvement.
Please refer to the submission report at http://xcalablemp.org

**FFT (SLOC 205, XMP/F)**

- **This year**: 212 TFlops, 82,944 nodes
- **Last year**: 50 TFlops, 38,864 nodes

**STREAM (SLOC 69, XMP/C)**

- **This year**: 3,583 TB/s, 82,944 nodes
- **Last year**: 2,439 TB/s, 82,944 nodes

http://xcalablemp.org
Compare to two versions

- Improvement rate (on the same nodes) **37 - 94% improvement !!**

![Bar chart showing ratios for different benchmarks](chart)

- SLOC

![Bar chart showing SLOC for different benchmarks](chart)

Last year, work-in-progress to clean up code

Good
Outline

1. XcalableMP (XMP) for cluster systems  (14min.)

2. XcalableACC (XACC) for accelerator cluster systems
   Extension of XMP using OpenACC  (6min.)

   Sorry !!, work-in-progress

The submission report is available at http://xcalablemp.org
What is XcalableACC?

Extension of XMP using OpenACC for accelerator clusters

Feature:

- Mix XMP and OpenACC directives **seamlessly**
- Support transferring data among accelerators **directly**
Difference XMP and XACC memory models

- **XMP memory model**
  Global Indexing

- **XACC memory model**
  Global Indexing

Map “global Indexing” to accelerators
XACC code example

```c
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p(x, y)
#pragma xmp template t(0:YSIZE-1, 0:XSIZE-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align [j][i] with t(i,j) :: u, uu
#pragma xmp shadow uu[1:1][1:1]
...
#pragma acc data copy(u) copyin(uu)
{
  for(k=0; k<MAX_ITER; k++){
    #pragma xmp loop (y,x) on t(y,x)
    #pragma acc parallel loop collapse(2)
      for(x=1; x<XSIZE-1; x++)
        for(y=1; y<YSIZE-1; y++)
          uu[x][y] = u[x][y];

    #pragma xmp reflect (uu) acc

    #pragma xmp loop (y,x) on t(y,x)
    #pragma acc parallel loop collapse(2)
      for(x=1; x<XSIZE-1; x++)
        for(y=1; y<YSIZE-1; y++)
          u[x][y] = (uu[x-1][y]+uu[x+1][y]+uu[x][y-1]+uu[x][y+1])/4.0;
  }
  // end k
} // end data
```

Laplace's equation

Data Distribution

Transfer XMP distributed arrays to accelerator

OpenACC directive parallelizes the loop statement parallelized by XMP directive

Exchange halo region of uu[][]

When “acc” clause is specified in XMP communication directive, data on accelerator is transferred.
Results and Machine

Summary

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Nodes</th>
<th>#CPUs</th>
<th>#GPUs</th>
<th>Performance (/peak)</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPL</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>7 TFlops (4.2%)</td>
<td>343</td>
</tr>
<tr>
<td>FFT</td>
<td>32</td>
<td>64</td>
<td>-</td>
<td>257 GFlops (0.1%)</td>
<td>205</td>
</tr>
<tr>
<td>STREAM</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>15 TB/s (20.4%)</td>
<td>84</td>
</tr>
<tr>
<td>HIMENO</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>14 TFlops (1.4%)</td>
<td>253</td>
</tr>
</tbody>
</table>

HA-PACS/TCA: 64 nodes

- Ivy Bridge E5-2680v2, 224GFlops x 2 Sockets
- DDR3 SDRAM 128GB, 59.7GB/s x 2
- Infiniband 4xQDR x 2 rails : 8GB/s
- NVIDIA K20X (4GPUs / Node)
  - 1.31 TFlops/GPU(SP), 3.95 TFlops/GPU(DP)
  - 250GB/s/GPU

http://www.ccs.tsukuba.ac.jp/CCS/eng/research-activities/projects/ha-pacs
The XACC STREAM uses both CPUs and GPUs together, **XMP, OpenACC**, and **OpenMP** directives are used.

```c
#pragma xmp nodes p(*)
#pragma acc data copy(a[0:GPU_SIZE], b[0:GPU_SIZE], c[0:GPU_SIZE])
{
    for (k=0; k<NTIMES; k++) {
        #pragma xmp barrier
        times[k] = -xmp_wtime();

        #pragma acc parallel loop async
        for (j=0; j<GPU_SIZE; j++)
            a[j] = b[j] + scalar*c[j];

        #pragma omp parallel for
        for (j=GPU_SIZE; j<MAX_SIZE; j++)
            a[j] = b[j] + scalar*c[j];

        #pragma acc wait
        #pragma xmp barrier
        times[k] += xmp_wtime();
    }
} // acc data
```

**on GPU**

**on CPU**

Wait until GPU task completes
Performance of STREAM

- XACCC (CPU + GPU)
  - SLOC: 84
  - 15 TB/s
  - 64 nodes (256 GPUs)

- XMP (Only CPU)
  - SLOC: 69
  - 5 TB/s
  - 64 nodes

reasonable performance

Number of nodes

GB/s
HIMENO Benchmark

- Stencil application of incompressible fluid analysis code
- Solving the Poisson’s equation
- Sequential and MPI Version HIMENO Benchmark is available at http://accc.riken.jp/2444.htm

```c
float p[MIMAX][MJMAX][MKMAX];
// Define distributed array and halo

#pragma acc data copy(p) ..
{
  ..
#pragma xmp reflect (p) acc
  ..
#pragma xmp loop (k,j,i) on t(k,j,i)
#pragma acc parallel loop ..
for(i=1; i<MIMAX; ++i)
  for(j=1; j<MJMAX; ++j){
#pragma acc loop vector ..
  for(k=1; k<MKMAX; ++k){
    S0 = p[i+1][j][k] * ..;
```
Performance of HIMENO

- XACC (only GPU)
  - SLOC: 213
  - GFlops: 14 TFlops
  - Number of nodes: 64 nodes (256 GPUs)

- MPI version HIMENO (Only CPU)
  - SLOC: 325
  - GFlops: 1.6 TFlops
  - Number of nodes: 64 nodes

Reasonable performance
HPL and FFT

Sorry !! work-in-progress for implementing and tuning.

HPL

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>GFlops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
</tr>
<tr>
<td>8</td>
<td>800</td>
</tr>
<tr>
<td>16</td>
<td>1600</td>
</tr>
<tr>
<td>32</td>
<td>3200</td>
</tr>
</tbody>
</table>

Using cuBLAS

XACC (only GPU)
SLOC:343
7 TFlops (4.2%)
32 nodes (128GPUs)

FFT

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>GFlops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

XMP (only CPU)
SLOC:205
257 GFlops (0.1%)
32 nodes

Will use FFTE-CUDA

Time of transfer data between CPU and host memory dominates the total computation time
## Conclusion

### XMP on the K computer

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># Nodes</th>
<th>Performance (/peak)</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ver. 1</td>
<td>16,384</td>
<td>971 TFlops (46.3%)</td>
<td>313</td>
</tr>
<tr>
<td>Ver. 2</td>
<td>4,096</td>
<td>423 TFlops (80.7%)</td>
<td>426</td>
</tr>
<tr>
<td>FFT</td>
<td>82,944</td>
<td>212 TFlops (2.0%)</td>
<td>205</td>
</tr>
<tr>
<td>STREAM</td>
<td>82,994</td>
<td>3,583 TB/s (67.5%)</td>
<td>69</td>
</tr>
<tr>
<td>RandomAccess</td>
<td>16,384</td>
<td>254 GUPs</td>
<td>253</td>
</tr>
</tbody>
</table>

### XACC on HA-PACS/TCA

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Nodes</th>
<th>#CPUs</th>
<th>#GPUs</th>
<th>Performance (/peak)</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPL</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>7 TFlops (4.2%)</td>
<td>343</td>
</tr>
<tr>
<td>FFT</td>
<td>32</td>
<td>64</td>
<td>-</td>
<td>257 GFlops (0.1%)</td>
<td>205</td>
</tr>
<tr>
<td>STREAM</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>15 TB/s (20.4%)</td>
<td>84</td>
</tr>
<tr>
<td>HIMENO</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>14 TFlops (1.4%)</td>
<td>253</td>
</tr>
</tbody>
</table>

Good productivity and performance!!

We will improve HPL and FFT next year.
For more information

Please visit our booth !!

- RIKEN AICS (Advanced Institute for Computational Science) #2413
- Center for Computational Sciences, University of Tsukuba #3215