XcalableACC: A Directive-Based Language Extension for Accelerator Clusters

Hitoshi Murai and Masahiro Nakao

RIKEN AICS
Introduction (1)

- Accelerated clusters (e.g. GPU clusters) have become very popular HPC platforms.
- MPI+CUDA style programming lowers productivity.
- Two directive-based languages exist:
  - XcalableMP (XMP) as an alternative to MPI
  - OpenACC as an alternative to CUDA
Introduction (2)

• Challenges in accelerated clusters
  – Hierarchical parallelism
    • among nodes -> XMP
    • (within a node -> OpenMP)
    • among ACCs -> ?
    • within ACCs -> OpenACC
  – Direct comm. among ACCs (physical or logical)
    • Tightly Coupled Accelerators (TCA)
    • NVLink
    • NVIDIA GPUDirect
Goals

• Proposing a new programming language for accelerated clusters by combining XcalableMP and OpenACC
  – hierarchical parallelism
  – direct communication among accelerators

• Developing its compiler

  ➔ Realizing high performance and productivity on accelerated clusters
Outline of This Talk

• What are XcalableMP and OpenACC?
• Design of XcalableACC
• Implementation of the Omni XcalableACC compiler
• Evaluation (to be presented by Nakao-san)
What's XcalableMP?

- Directive-based PGAS extension for Fortran and C
  - Proposed by XMP Spec. WG of PC Cluster Consortium.
- Supports two parallelization paradigms:
  - Global-view (with HPF-like data/work mapping directives)
  - Local-view (with coarray)
- Allows mixture with MPI and/or OpenMP.

```plaintext
!$xmp nodes p(2,2)  
!$xmp template t(n,n)  
!$xmp distribute t(block,block) onto p  
  real a(n,n)  
!$xmp align a(i,j) with t(i,j)  
!$xmp shadow a(1,1)  

!$xmp reflect (a)  
!$xmp loop (i,j) on t(i,j)  
  do j = 2, n-1  
  do i = 2, n-1  
  w = a(i-1,j) + a(i+1,j) + ...  
  ...
```
What's OpenACC?

• Directive-based extension to program accelerators for C/C++/Fortran
  – Developed by Cray, CAPS, PGI (NVIDIA)

• Based on the **offload** model
  – A host (CPU) offloads data/work to devices (accelerators, ACCs)

• Portability across OSs, CPUs and ACCs.
Basic Concepts of XACC

- XACC = XMP + OpenACC + XACC Extensions

<table>
<thead>
<tr>
<th>XMP directives</th>
<th>distributed-memory parallelism among nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenACC directives</td>
<td>accelerator(s) within a node</td>
</tr>
<tr>
<td>XACC Extensions</td>
<td>• hierarchical parallelism</td>
</tr>
<tr>
<td></td>
<td>• direct comm. among ACCs</td>
</tr>
</tbody>
</table>

- With XACC, XMP features (including coarray) could be applied to ACCs for productivity.
Execution Model of XACC

Array/Work

Distribution among nodes

Distribution among ACCs.

Comm. among CPUs

Direct Comm. among ACCs

node

Oct. 29, 2015
LENS2015
Syntax of XACC

• Basically, diagonal combination of XMP and OpenACC
  – XMP outer and OpenACC inner (first distribute among nodes, and then onto accelerators)

• Two kinds of directives (sentinels)
  – #pragma xmp: XMP directives and XACC extensions for direct comm.
  – #pragma acc: OpenACC directives and XACC extensions for hierarchical parallelism.
XACC Extensions

• `pragma xmp` (for direct comm. among ACCs)
  – acc clause

• `pragma acc` (for hierarchical parallelism)
  – device directive
  – on_device clause
  – layout clause
  – shadow clause
  – barrier_device directive
void foo(){
    #pragma xmp nodes p(4)
    
    #pragma xmp template t(0:99)
    #pragma xmp distribute t(block) onto p
    
    float a[100][100];
    #pragma xmp align a[i][*] with t(i)
    #pragma xmp shadow a[1:1][0]
    
    #pragma xmp reflect (a)
    
    #pragma xmp loop (i) on t(i)
    for (int i = 0; i < 100; i++){
        for (int j = 0; j < 99; j++){
            a[i][j+1] = 1;
        }
    }
}
void foo()

#pragma xmp nodes p(4)
#pragma acc device d(*)

#pragma xmp template t(0:99)
#pragma xmp distribute t(block) onto p

float a[100][100];
#pragma xmp align a[i][*] with t(i)
#pragma xmp shadow a[1:1][0]
#pragma acc declare copy(a) layout([*][block]) shadow([0][1:1]) on_device(d)

#pragma xmp reflect (a) acc

#pragma xmp loop (i) on t(i)
  for (int i = 0; i < 100; i++){
    #pragma acc parallel loop layout(a[*][j+1]) on_device(d)
    for (int j = 0; j < 99; j++){
      a[i][j+1] = 1;
    }
  }
}
void foo(){

#pragma acc device d(*)

float a[100][100];

#pragma acc declare copy(a) layout([*][block]) shadow([0][1:1]) on_device(d)

#pragma xmp reflect (a) acc

for (int i = 0; i < 100; i++){
    #pragma acc parallel loop layout(a[*][j+1]) on_device(d)
    for (int j = 0; j < 99; j++){
        a[i][j+1] = 1;
    }
}
}
Omni XcalableACC

• Being developed as an extension of the Omni XMP compiler.

• Primary target: HA-PACS/TCA

Note: Omni itself has an experimental function of an OpenACC compiler, and therefore can be used as the backend of Omni XACC.
Impl. of Coarrays

- Two kinds of coarray implementation are available on Omni XACC.

<table>
<thead>
<tr>
<th>Method</th>
<th>MVAPICH2-GDR</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>put / get</td>
<td>MPI_Put / MPI_Get</td>
<td>start DMA</td>
</tr>
<tr>
<td>sync memory</td>
<td>MPI_Win_flush_all</td>
<td>wait for DMA to complete</td>
</tr>
<tr>
<td>post / wait</td>
<td>MPI_Send / MPI_Recv</td>
<td>packet comm. using DMA and a ring buffer</td>
</tr>
</tbody>
</table>

```c
#pragma xmp nodes p(2)
int a[N]:[*];
#pragma acc declare(a)
... if (this_image() == 1) { // origin
    #pragma acc host data use_device(a)
    a[:][2] = a[:];
    xmp_sync_memory(&status);
    #pragma xmp post(p(2), 0)
} else { // target
    #pragma xmp wait(p(1), 0)
    xmp_sync_memory(&status);
}```
Ping-Pong Performance

- HA-PACS/TCA
- MVAPICH2-GDR 2.1a
Hybrid Communication

• The Omni XACC runtime could utilize both IB and TCA together
  – to take advantage of each, for performance; and
  – to overcome the size limitation of the TCA subcluster, for scalability.

• Applied to the patterns:
  – neighborhood (stencil)
  – Bcast
  – Allgather
  – Allreduce

Comm. schedule of Allgather
Evaluation
Summary

• A new programming language XcalableACC for accelerated clusters is proposed.

• XACC = XMP + OpenACC + XACC extensions

• Omni XACC is being developed to support TCA.

• The preliminary evaluations showed high performance and productivity of XACC.
Future Works

• Fix the XACC language.
• Implement Omni XACC.
• Real applications.
• More evaluation, and improvement of the language and the compiler.