Dynamic task parallelism in PGAS language XcalableMP

Keisuke Tsugane†1, Masahiro Nakao†2, Jinpil Lee†2, Hitoshi Murai†2, Mitsuhsa Sato†1,2

†1 Graduate School of Systems and Information Engineering, University of Tsukuba
†2 Advanced Institute for Computational Science, RIKEN
Outline

• Background
• Programming model
  – OpenMP
  – XcalableMP
  – StarPU
• Dynamic Task Parallelism in XcalableMP
• Evaluation
  – Implementation of Block Cholesky Factorization
  – Evaluation of performance and productivity
• Conclusion
Outline

• **Background**
• Programming model
  – OpenMP
  – XcalableMP
  – StarPU
• Dynamic Task Parallelism in XcalableMP
• Evaluation
  – Implementation of Block Cholesky Factorization
  – Evaluation of performance and productivity
• Conclusion
Background

- Large scale clusters based on many core processor have been deploying in many sites
  - 01, Tianhe-2, China
  - 08, Stampede, U.S.A
  - 93, COMA, Japan
  - Top500 at ISC 2015

- Firstly, we need to consider the performance on many core processor in node
  - In node, OpenMP is widely used
  - Cost of synchronization is large
    - barrier or
      - task-to-task synchronization
Background

• MPI is used widely for programming distributed memory systems
  – Low productivity is an important problem
    • data distribution explicitly
    • primitive API for inter-node communication

• XcalableMP (XMP) is proposed and developed as a new PGAS parallel programming language
  – extension of existing languages (C and Fortran) by directives
  – only adding directives to a serial code

• Even if users use PGAS, task parallelism is complex
  – XMP + OpenMP
    • task description for distributed data
    • inter-node communication on task-to-task
Objective

• We propose the **tasklet directive** for dynamic task parallelism in XcalableMP
  – To implement the block cholesky factorization
  – To evaluate performance and productivity
    • by comparing with StarPU

- XMP tasklet directive is now under consideration by XMP Specification Working Group for XMP 2.0
- This research is one of the proposal function
Outline

• Background

• **Programming model**
  – OpenMP
  – XcalableMP
  – StarPU

• Dynamic Task Parallelism in XcalableMP

• Performance Evaluation
  – Implementation of Block Cholesky Factorization
  – Evaluation of performance and productivity

• Conclusion
OpenMP

- **task** directive
  - Task parallelism facilitates the parallelization where work is generated dynamically
    - as in recursive structure or while loop
  - OpenMP 4.0 adds *depend* clause on the *task* directive
    - describe to synchronize between task-to-task
    - Users express the *dependency-type* and *list items*
      - *dependency-type* : *in*, *out*, and *inout*

```c
#pragma omp parallel
#pragma omp single
{
  int x, y;
  #pragma omp task depend(out:x)
  taskA();
  #pragma omp task depend(out:y)
  taskB();
  #pragma omp task depend(in:x, y)
  taskC();
}
```
XcalableMP (XMP)

• PGAS parallel programming language on distributed memory systems
  – extension of existing languages (C and Fortran) by directives
  – specification: PC Cluster Consortium Japan
  – reference implementation: Omni XMP Compiler implemented by University of Tsukuba, RIKEN

• Directive-based language by pragma
  – only adding directives to a serial code

• Two programming models are supported:
  – global-view model
  – local-view model
Global-view Programming Model

• Support typical data distribution, parallel execution and communication by adding directives
• Data distribution
  – using template as virtual index space

int A[N];
#pragma xmp nodes P(4)
#pragma xmp template T(0:N-1)
#pragma xmp distribute T(block) on P
#pragma xmp align a[i] with T(i)

Each element of array is assigned to each processor via the distributed template
StarPU

- StarPU is a task programming library which is developed at INRIA
- Users can describe the task parallelism for CPUs and GPUs
- StarPU primary data structure is the \textit{codelet}
  - task function, number of data buffer, data dependency, and more
  - Data using in task is aligned with \textit{starpu\_data\_handle}
- Task-to-task communication is automatically executed
  - Users don’t need to describe the explicit communication

```c
void taskA(void *descr[], void * _args){
  int *A = (int *)STARPU_VARIABLE_GET_PTR(descr[0]);
  /* calculation */
}

static struct starpu_codelet codelet_taskA =
  {
    .cpu_funcs = {taskA},
    .nbuffers = 1,
    .modes = {STARPU_RW},
  };
/* ... */

int A;
starpu_data_handle_t data_handles;
starpu_variable_data_register(&data_handles, &A);
starpu_mpi_insert_task(&codelet_taskA, STARPU_RW, ¥ data_handles);
```
Outline

• Background
• Programming model
  – OpenMP
  – XcalableMP
  – StarPU
• **Dynamic Task Parallelism in XcalableMP**
• Evaluation
  – Implementation of Block Cholesky Factorization
  – Evaluation of performance and productivity
• Conclusion
Dynamic task parallelism on distributed memory systems

- Task-to-task dependency is needed for dynamic task parallelism
  - intra-node: OpenMP task depend
  - inter-node: MPI or PGAS communication

- For example,
  - Array A[100] is distributed to 4 nodes
  - Each node has 25 elements of array A
  - taskA calculates A[0:25]
  - taskB calculates A[75:25] from A[0:25] which is performed by taskA
  - MPI_Send task is generated after taskA execution in Node 1
  - MPI_Recv task is generated before taskB execution in Node 4
tasklet directive

- **tasklet** directive is a one of proposal for dynamic task parallelism in XMP
  - Execution node is determined by distributed array in global-view
  - Users need to describe the local buffer for receive data
tasklet directive

#pragma xmp tasklet tasklet-format[, tasklet-format, ...] [on {node-ref | template-ref}]
(structured-block)

where tasklet-format is:
  dependence-type (array name[, {node-ref | template-ref}][, local buffer[, tag]])

and dependence-type is:
in, out, or inout

• Dependency-type is same as OpenMP task depend
  – array name is distributed array
  – XMP node or template determine the communication partner
  – When node-ref or template-ref is omitted, the tasklet is executed same as OpenMP task depend

• Execution node is decided by on clause
  – When on clause is omitted, the task is executed on node which has distributed array indicated by out or inout dependency-type
  – on clause has XMP node or template
Code Translation

- *tasklet* directive is translated to MPI+OpenMP code
  - Base compiler is a Omni XMP Compiler which is a source-to-source translator changed from XMP directive to MPI code
  - In this study, we program the **MPI+OpenMP code** which is translated from *tasklet* directive
    - It is the first step to evaluate the performance of dynamic task parallelism on distributed memory systems
Code Translation

**XMP**

```c
#pragma xmp tasklet out(A[0:25], P(4))
taskA();
#pragma xmp tasklet in(A[0:25], P(1), B) out(A[75:25])
taskB();
#pragma xmp taskletwait
```

**MPI+OpenMP**

```c
#pragma omp parallel
#pragma omp single
{
  if ( I have A[0:25] ) {
    #pragma omp task depend(out:A[0:25])
    taskA();
    #pragma omp task depend(inout:A[0:25])
    MPI_Isend(A, 25, P(4), tag, ...);
  }
  if ( I have A[75:25] ) {
    #pragma omp task depend(inout:B)
    {
      MPI_Irecv(B, 25, P(1), tag, ...);
      MPI_Wait(...);
    }
    #pragma omp task depend(in:A[0:25], B) ¥
    #pragma omp task depend(out:A[75:25])
    testB();
  }
}
#pragma omp taskwait
MPI_Waitall(...);
```

- Code image translated from XMP tasklet directive
  - out : MPI_Isend
  - in : MPI_Irecv
Outline

• Background
• Programming model
  – OpenMP
  – XcalableMP
  – StarPU
• Dynamic Task Parallelism in XcalableMP
• Evaluation
  – Implementation of Block Cholesky Factorization
  – Evaluation of performance and productivity
• Conclusion
Block Cholesky Factorization

• Typical behavior of block cholesky factorization is as follows
  – computes the Cholesky factorization (potrf)
  – solves a triangular matrix equation (trsm)
  – performs a symmetric rank-k update (syrk)
  – computes a matrix multiplication (gemm)

• These operations are performed by block unit

• We used the Intel MKL to perform these operations
Block Cholesky Factorization

OpenMP

double A[nt][nt][ts*ts];

#pragma omp parallel
#pragma omp single
{
  for (int k = 0; k < nt; k++) {
    #pragma omp task depend(inout:A[k][k])
    omp_potrf (A[k][k], ts, ts);
  }
  for (int i = k + 1; i < nt; i++) {
    #pragma omp task depend(in:A[k][k])
    depend(inout:A[k][i])
    omp_trsm (A[k][k], A[k][i], ts, ts);
  }
  /* ... */
}
#pragma omp taskwait

XMP

double A[nt][nt][ts*ts], B[ts*ts];
#pragma xmp node P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

for (int k = 0; k < nt; k++) {
  #pragma xmp tasklet inout(A[k][k], T(k+1:nt-1))
  omp_potrf (A[k][k], ts, ts);
}
for (int i = k + 1; i < nt; i++) {
  #pragma xmp tasklet in(A[k][k])
  #pragma xmp tasklet depend(inout:A[k][i])
  omp_trsm (A[k][k], A[k][i], ts, ts);
  }
  /* ... */
}
#pragma xmp taskletwait

We add directives and slightly modify to the OpenMP code
Experimental Settings

- Platform: COMA (CCS, University of Tsukuba)

<table>
<thead>
<tr>
<th></th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon E5-2670v2 x 2</td>
</tr>
<tr>
<td></td>
<td>CPU (10 cores/CPU) x 2 = 20 cores</td>
</tr>
<tr>
<td>Memory</td>
<td>64 GB</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Infiniband FDR Connect-X3</td>
</tr>
<tr>
<td>OS</td>
<td>Red Hat Enterprise Linux Server 6.4</td>
</tr>
<tr>
<td>C Compiler</td>
<td>Intel 15.0.2</td>
</tr>
<tr>
<td>MPI</td>
<td>Intel MPI 5.0.3</td>
</tr>
<tr>
<td>Intel MKL</td>
<td>11.2.2</td>
</tr>
</tbody>
</table>

- This research used computational resources of COMA provided by Interdisciplinary Computational Science Program in Center for Computational Sciences, University of Tsukuba
Experimental Settings

• We evaluate the performance and productivity of cholesky factorization with *tasklet* directive
  – by comparing with StarPU

• Problem size (N x N)
  – N : 4096, 8192

• Preliminary evaluation
  – We explore an appropriate block size
  – 1 node execution, 1 ~ 16 threads per node, 1 process per node
  – Block size (M x M)
    • M : 32 ~ 1024

• Performance evaluation
  – Up to 16 nodes, 1 ~ 16 threads per node, 1 process per node
  – distribution pattern : one dimensional cyclic distribution
Preliminary Evaluation

• We explored an appropriate block size
  – Matrix size (N x N)
    • N : 4096, 8192
  – Block size (M x M)
    • M : 32 ~ 1024
  – 1 node, 1 to 16 threads

appropriate block size is 128x128
Performance Evaluation

- Matrix size $N : 4096$, Block size $M : 128$
  - “XMP(MPI+OpenMP)” indicates the code which is translated from tasklet directive

- The performance of “XMP(MPI+OpenMP)” is lower than “StarPU” as a while.
  - “XMP(MPI+OpenMP)” is 60 ~ 94% against “StarPU”
Performance Evaluation

- Matrix size N : 8192, Block size M : 128
  - “XMP(MPI+OpenMP)” indicates the code which is translated from tasklet directive

- The performance of “XMP(MPI+OpenMP)” is higher than “StarPU” in case of 8 and 16 nodes
  - “XMP(MPI+OpenMP)” is 60 ~ 134% against “StarPU”
Productivity

- XMP implementation
  - adding directives and small modification
  - comparison of the number of line with several implementation
    - smaller with the “XMP” than “MPI+OpenMP” and “StarPU”
    - same as “OpenMP” in spite of distributed memory systems

<table>
<thead>
<tr>
<th></th>
<th>OpenMP</th>
<th>MPI +OpenMP</th>
<th>StarPU</th>
<th>XMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP directive</td>
<td>7</td>
<td>11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>XMP directive</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>10</td>
</tr>
<tr>
<td>other</td>
<td>404</td>
<td>574</td>
<td>489</td>
<td>408</td>
</tr>
</tbody>
</table>
Outline

• Background
• Programming model
  – OpenMP
  – XcalableMP
  – StarPU
• Dynamic Task Parallelism in XcalableMP
• Evaluation
  – Implementation of Block Cholesky Factorization
  – Evaluation of performance and productivity
• Conclusion
Conclusion and Future Work

• We proposed the tasklet directive for dynamic task parallelism in XcalableMP
  – We programmed the MPI+OpenMP code which is translated from tasklet directive
  – In most cases, the performance of block cholesky code in MPI +OpenMP are lower than that of StarPU version
  – tasklet directive obtained the high productivity
    • only adding directives and small modification

• Future work
  – investigation of the causes of performance degradation
  – evaluation of our method using other distribution method
    • block-cyclic, two dimensional distribution
  – evaluation on Intel Xeon Phi