Fine-Grained Synchronization using Global Task Dependencies in DASH

Joseph Schuchart, José Gracia

October 2018, WSSP
Motivation: Strong synchronisation in MPI applications

MPI computation

idle

c0

MPI0

c1

c2

MPI1

c3

c4

c5

MPI2

c6

c7

c8

time
Motivation: Strong synchronisation in MPI applications

MPI+OpenMP
Oakforest PACS (KNL), 16 nodes, 64 Threads each
Outline

Background
- DASH – a C++ PGAS Model
- OpenMP Task Data-Dependencies

Global Data Dependencies with DASH

Preliminary Evaluation

Conclusion and Future Work
Background
DASH – a C++ PGAS Model

- Partitioned Global Address Space (PGAS)
- C++11/14 distributed data structures and C11 runtime (using MPI-3 RMA)
- Follows STL-principles
- Data-centric computation ("owner computes")
# DASH – a C++ PGAS Model

<table>
<thead>
<tr>
<th>Container</th>
<th>Description</th>
<th>Data distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Array&lt;T&gt;</code></td>
<td>1D Array</td>
<td>static, configurable</td>
</tr>
<tr>
<td><code>NArray&lt;T, N&gt;</code></td>
<td>N-dim. Array</td>
<td>static, configurable</td>
</tr>
<tr>
<td><code>Shared&lt;T&gt;</code></td>
<td>Shared scalar</td>
<td>fixed, configurable</td>
</tr>
<tr>
<td><code>Directory*&lt;T&gt;</code></td>
<td>Variable-size,</td>
<td>manual</td>
</tr>
<tr>
<td></td>
<td>locally indexed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Array</td>
<td></td>
</tr>
<tr>
<td><code>CoArray*&lt;T&gt;</code></td>
<td>Similar to CAF</td>
<td>uniform</td>
</tr>
</tbody>
</table>

(*) Under construction
DASH – a C++ PGAS Model

- Partitioned Global Address Space (PGAS)
- C++11/14 distributed data structures and C11 runtime (using MPI-3 RMA)
- Follows STL-principles
- Data-centric computation ("owner computes")

Synchronization:
- PGAS: decoupled synchronization and data transfer
- Team-wide (global) synchronization
- Distributed lock implementation
- No fine-grained synchronization! (yet)
How to Achieve Fine-grained Synchronization?

Task-based execution model for increased concurrency

Existing PGAS+Task approaches:

- Direct task synchronization, remote task invocation; HPX+UPC++ [3, 4]
- (Explicit) synchronization variables; Chapel [1]
- Explicit tags; XMP [5]
- Affinity instead of branches; PaRSEC [2]

```cpp
upcxx::event e;
upcxx::async(rank, &e)(
    Function, args...);
e.wait();

var buffReady$: sync bool;
buffReady.readFE();

#pragma xmp tasklet \get_ready(A, proc, tag)
... 
#pragma xmp tasklet \get(tag)
```
How to Achieve Fine-grained Synchronization?

Task-based execution model for increased concurrency

Existing PGAS+Task approaches:
- Direct task synchronization, remote task invocation; HPX+UPC++ [3, 4]
- (Explicit) synchronization variables; Chapel [1]
- Explicit tags; XMP [5]
- Affinity instead of branches; PaRSEC [2]

DASH requires:
- Scalable distributed task creation
- Implicit, *data-centric* global synchronization
- Recurring dependency patterns

```c
upcxx::event e;
upcxx::async(rank, &e)(
    Function, args...);
e.wait();

var buffReady$: sync bool;
buffReady.readFE();

#pragma xmp tasklet
get_ready(A, proc, tag)
...
#pragma xmp tasklet
get(tag)
```
A step back: OpenMP Tasks

OpenMP supports asynchronous tasks since v3.0

Synchronization: Task data dependencies since v4.0
- Describe data flow to form task graph
- Implicit synchronization among *siblings tasks*
- Strict backward matching
Blocked Cholesky Factorization

```c
for (int k = 0; k < nt; k++) {
    #pragma omp task out(A[k][k])
    potrf(A[k][k], ts, ts);

    for (int i = k + 1; i < nt; i++)
        #pragma omp task in(A[k][k]) out(A[k][i])
        trsm(A[k][k], A[k][i], ts, ts);

    for (int i = k + 1; i < nt; i++) {
        for (int j = k + 1; j < i; j++)
            #pragma omp task in(A[k][i], A[k][j]) out(A[j][i])
            gemm(A[k][i], A[k][j], A[j][i], ts, ts);

    #pragma omp task in(A[k][i]) out(A[i][i])
    syrk(A[k][i], A[i][i], ts, ts);
    }
}
```
Blocked Cholesky Factorization

for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local())
        potrf(block_kk);

#pragma omp taskwait
    dash::barrier();

    for (int i = k+1; i < num_blocks; ++i)
        if (block_ki.is_local())
            #pragma omp task
                trsm(block_kk, block_ki);

#pragma omp taskwait
    dash::barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        for (int j = k+1; j < i; ++j)
            if (block_ji.is_local())
                #pragma omp task
                    gemm(block_ki, block_kj, block_ki);

            if (block_ii.is_local())
                #pragma omp task
                    syrk(block_ki, block_ii);
    }

#pragma omp taskwait
    dash::barrier();
}
Limitations of DASH + OpenMP

- Synchronization through collectives dependencies across nodes have to be enforced by barriers
- Synchronization may lead to imbalances
- Complex/impossible to further taskify
Limitations of DASH + OpenMP

- Synchronization through collectives dependencies across nodes have to be enforced by barriers
- Synchronization may lead to imbalances
- Complex/impossible to further taskify
Global Data Dependencies with DASH
Global Data Dependencies

Multiple Iterations

[Diagram of network with nodes and edges representing data dependencies]
Global Data Dependencies

Remote Dependencies

Local Dependencies
Global Data Dependencies

Remote Dependencies

No ordering!

Local Dependencies

Ordered

Serial task creation

Serial task creation
Global Task Ordering

Issue:
Remote dependency resolution depends on order of task instantiation across nodes

How to restore global task ordering?
Global Task Ordering

Proposed Solution: **Task Phases**

- *Logical clock* for tasks and dependencies
- Virtual barrier in task-based execution
- *Happens before*: input dependencies refer to earlier phase
- Additional information provided by the user
Global Task Ordering

Proposed Solution: **Task Phases**

- *Logical clock* for tasks and dependencies
- Virtual barrier in task-based execution
- *Happens before:* input dependencies refer to earlier phase
- Additional information provided by the user
- Overlapping iterations

```cpp
++i;
ph=0
print
ph=1
++i;
ph=1
```

```
... ...
... ...
... ...
ph=0 ph=1 ph=2
... ...
... ...
... ...
ph=0 ph=1 ph=2
```
DASH Prototype Implementation

- Nested tasking runtime using pthreads
- Preliminary C++ API

- Dependencies:
  - Local in/out dependencies (similar to OpenMP)
  - Remote in/out dependencies
  - Flexible copyin: 1-sided or 2-sided

- Active message queue based on MPI
- Re-scheduling task-yield
- Global task cancellation
- Task priorities
Blocked Cholesky Factorization (DASH)

```cpp
for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local()) {
        potrf(block_kk);
    }
    dash::barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        if (block_ki.is_local()) {
            trsm(block_kk, block_ki);
        }
    }
    dash::barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        for (int j = k+1; j < i; ++j) {
            if (block_ji.is_local()) {
                gemm(block_ki, block_kj, block_ki);
            }
        }
        if (block_ii.is_local()) {
            syrk(block_ki, block_ii);
        }
    }
    dash::barrier();
}
```

Global synchronization to ensure happens-before relation
Blocked Cholesky Factorization (DASH Tasks)

```cpp
for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local()) {
        dash::async([=](){ potrf(block_kk); },
                    dash::out(block_kk));
    } dash::async_barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        if (block_ki.is_local()) {
            dash::async([=](){ trsm(block_kk, block_ki); },
                        dash::in(block_kk), dash::out(block_ki));
        } dash::async_barrier();

        for (int j = k+1; j < i; ++j) {
            if (block_ji.is_local()) {
                dash::async([=](){ gemm(block_ki, block_kj, block_ki); },
                             dash::in(block_ki), dash::in(block_kj), dash::out(block_ji));
            } dash::async_barrier();

            if (block_ii.is_local()) {
                dash::async([=](){ syrk(block_ki, block_ii); },
                             dash::in(block_ki), dash::out(block_ii));
            } dash::async_barrier();
        } dash::async_barrier();
    } dash::complete();
```

`async` introduces a new task using C++ lambda notation.
Blocked Cholesky Factorization (DASH Tasks)

```cpp
for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local()) {
        dash::async(=[](){ potrf(block_kk); },
                     dash::out(block_kk));
    }
    dash::async_barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        if (block_ki.is_local()) {
            dash::async(=[](){ trsm(block_kk, block_ki); },
                         dash::in(block_kk), dash::out(block_ki));
        }
        dash::async_barrier();

        for (int j = k+1; j < i; ++j) {
            if (block_ji.is_local()) {
                dash::async(=[](){ gemm(block_ki, block_kj, block_ki); },
                              dash::in(block_ki), dash::in(block_kj), dash::out(block_ji));
            }
        }
        if (block_ii.is_local()) {
            dash::async(=[](){ syrk(block_ki, block_ii); },
                           dash::in(block_ki), dash::out(block_ii));
        }
    }
    dash::async_barrier();
}
dash::complete();
```

Non-blocking phase barrier to ensure happens-before relation.
Blocked Cholesky Factorization (DASH Tasks)

```cpp
for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local()) {
        dash::async([=](){ potrf(block_kk); },
        dash::out(block_kk));
    }
    dash::async_barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        if (block_ki.is_local()) {
            dash::async([=](){ trsm(block_kk, block_ki); },
            dash::in(block_kk), dash::out(block_ki));
        }
        dash::async_barrier();
    }

    for (int i = k+1; i < num_blocks; ++i) {
        for (int j = k+1; j < i; ++j) {
            if (block_ji.is_local()) {
                dash::async([=](){ gemm(block_ki, block_kj, block_ki); },
                dash::in(block_ki), dash::in(block_kj), dash::out(block_ji));
            }
        }
        if (block_ii.is_local()) {
            dash::async([=](){ syrk(block_ki, block_ii); },
            dash::in(block_ki), dash::out(block_ii));
        }
    }
    dash::async_barrier();
}
dash::complete();
```
Blocked Cholesky Factorization (DASH Tasks)

```c
for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local()) {
        dash::async([=](){ potrf(block_kk); },
                     dash::out(block_kk));
    }
    dash::async_barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        if (block_ki.is_local()) {
            dash::async([=](){ trsm(block_kk, block_ki); },
                         dash::in(block_kk), dash::out(block_ki));
        }
        dash::async_barrier();

        for (int j = k+1; j < i; ++j) {
            if (block_ji.is_local()) {
                dash::async([=](){ gemm(block_ki, block_kj, block_ki); },
                             dash::in(block_ki), dash::in(block_kj), dash::out(block_ji));
            }
        }
        if (block_ii.is_local()) {
            dash::async([=](){ syrk(block_ki, block_ii); },
                         dash::in(block_ki), dash::out(block_ii));
        }
    }
    dash::async_barrier();
}
```

Wait for completion of all tasks
Blocked Cholesky Factorization (DASH Tasks)

```cpp
for (int k = 0; k < num_blocks; ++k) {
    if (block_kk.is_local()) {
        dash::async([=](){ potrf(block_kk); },
                     dash::out(block_kk));
    }
    dash::async_barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        if (block_ki.is_local()) {
            dash::async([=](){ trsm(block_kk, block_ki); },
                         dash::in(block_kk), dash::out(block_ki));
        }
    }
    dash::async_barrier();

    for (int i = k+1; i < num_blocks; ++i) {
        for (int j = k+1; j < i; ++j) {
            if (block_ji.is_local()) {
                dash::async([=](){ gemm(block_ki, block_kj, block_ki); },
                             dash::in(block_ki), dash::in(block_kj), dash::out(block_ji));
            }
        }
        if (block_ii.is_local()) {
            dash::async([=](){ syrk(block_ki, block_ii); },
                         dash::in(block_ki), dash::out(block_ii));
        }
    }
    dash::async_barrier();
}
```

dash::complete();
Preliminary Evaluation
Platforms

Systems under test:

- Laki (HLRS):
  2 x ’Haswell’ E5-2680v3 12-core, IB, GCC 7.1.0

- Oakforest PACS (U Tsukuba):
  Intel Xeon Phi 7250 (KNL)
  OmniPath, Intel 18.0.1
Evaluation Kernel

- Blocked Cholesky Factorization
  - 4 operations: `potrf`, `trsm`, `syrk`, `dgemm`
  - Fine-grained task dependencies
  - Scheduling-sensitive

- Implementations:
  - DTasks: DASH Tasks w/ manual block prefetching
  - Copyin: DASH Tasks w/ `copyin` dependency
  - DASH+OpenMP: OpenMP tasks between global sync
  - MPI+OpenMP: single communication task
  - MPI+DTasks: block-wise communication tasks (test-yield-loop)
Results: Laki

Cholesky Factorization

Laki, N=65536, BS=512

- Copyin (Get)
- DTasks
- DASH+OpenMP
- MPI+OpenMP
- MPI+DTasks
Results: Oakforest PACS

Cholesky Factorization

Oakforest PACS, N=65536, BS=512

- Copyin (Sendrecv)
- DTasks
- DASH+OpenMP
- MPI+OpenMP
- MPI+DTasks

Performance [GF/s] vs 
#Nodes (x60 threads)
Conclusion and Future Work
Conclusion

Global task data dependencies:

- Concept known from OpenMP
- Distributed task creation and synchronization
- Fine-grained, data-centric synchronization
- (Some) Ordering information required from user
  - Replace blocking `barrier()` with `async_barrier()`
- Promising first results (even on systems that poorly support RDMA)
Future Work

- Work on distributed scheduler
  - Advanced dependency types (concurrent, commutative)
  - Performance optimizations
  - Edge out bugs
- Tool support
- Interoperability with OpenMP (OmpSs?)
- Applications
  - Lulesh (Shock Hydrodynamics, already ported to DASH)
  - Ideas are welcome (preferably C/C++)
Thank you for your attention!

Questions?

joseph.schuchart@hlrs.de
github.com/dash-project/
dash-project.org
Bibliography

B. L. Chamberlain, D. Callahan, and H. P. Zima.
Parallel programmability and the Chapel language.

R. Hoque, T. Herault, G. Bosilca, and J. Dongarra.
Dynamic Task Discovery in PaRSEC: A Data-flow Task-based Runtime.

H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, and D. Fey.
HPX: A Task Based Programming Model in a Global Address Space.

V. Kumar, Y. Zheng, V. Cavé, Z. Budimlić, and V. Sarkar.
Habaneroupc++: A compiler-free pgas library.

Multi-tasking Execution in PGAS Language XcalableMP and Communication Optimization on Many-core Clusters.
Cholesky: MPI+OpenMP

N = 32,768, bs = 512, column-wise distribution, MPI+OpenMP (10s)
Oakforest PACS (KNL), 16 nodes, 64 Threads each, w/ block pre-fetching
Cholesky: DASH+Tasks+Copyin

N = 32,768, bs = 512, column-wise distribution, DASH+Tasks+Copyin (3.0s) Oakforest PACS (KNL), 16 nodes, 64 Threads each, w/ block pre-fetching