Global Task Data Dependencies in PGAS Applications

Joseph Schuchart

October 19, 2017

The DASH Programming Model

- SPMD-style programming (cf. MPI, OpenShmem, . . .)
- Data-centric computation
- Thread-safety guarantees (some limitations apply)
- Synchronization:
  - PGAS: decoupled synchronization and data transfer
  - Team-wide (global) synchronization
  - Distributed lock implementation

The DASH Programming Model

- SPMD-style programming (cf. MPI, OpenShmem, . . .)
- Data-centric computation
- Thread-safety guarantees (some limitations apply)
- 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

How to Achieve Fine-grained Synchronization?

Task-based execution model for increased concurrency

Existing PGAS+Task approaches:

- Support direct task synchronization
- Rely on remote task invocation
- Use (explicit) synchronization variables

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

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

How to Achieve Fine-grained Synchronization?

Task-based execution model for increased concurrency

Existing PGAS+Task approaches:

- Support direct task synchronization
- Rely on remote task invocation
- Use (explicit) synchronization variables

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

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

DASH requires:

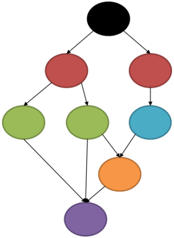
- Distributed task creation and synchronization
- Implicit, data-centric synchronization
- Recurring dependency patterns

A step back: OpenMP Tasks

OpenMP supports asynchronous tasks since v3.0

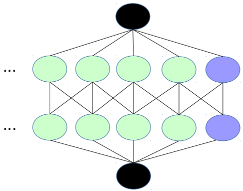
Synchronization: Task data dependencies since v4.0

- Describe data flow to form task graph
- Implicit synchronization among sibling tasks
- Covers RAW, WAR, and WAW dependencies
- Strict backward matching



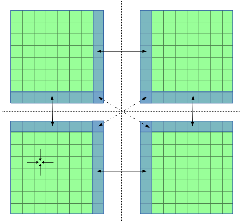
Local Task Dependencies

```
double d[2][N];

for (int t = 0; t < TIMESTEPS; t++) {

  int out = t%2;

#pragma omp task \
        depend(in:  d[out][N-2]) \
        depend(out: d[out][N-1])
  { compute_boundary(d[out]) }

  for (int i = 1; i < N-1; i++) {
#pragma omp task \
          depend(in: d[out][i-1], d[out][i+1]) \
          depend(out: d[out][i])
    { compute_cell(out, i); }
  }
}
```



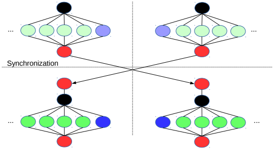
Local Task Dependencies

```
double d[2][N];

for (int t = 0; t < TIMESTEPS; t++) {

  int out = t%2;

#pragma omp task \
        depend(in:  d[out][N-2]) \
        depend(out: d[out][N-1])
  { compute_boundary(d[out]) }

  for (int i = 1; i < N-1; i++) {
#pragma omp task \
          depend(in: d[out][i-1], d[out][i+1]) \
          depend(out: d[out][i])
    { compute_cell(out, i); }
  }

  #pragma omp taskwait
  exchange_boundaries();
}
```



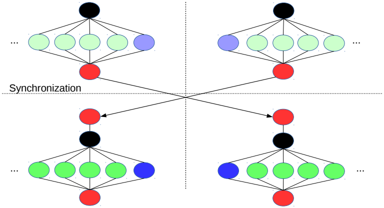
Local Task Dependencies

```
double d[2][N];

for (int t = 0; t < TIMESTEPS; t++) {

  int out = t%2;

#pragma omp task \
        depend(in:  d[out][N-2]) \
        depend(out: d[out][N-1])
  { compute_boundary(d[out]) }

  for (int i = 1; i < N-1; i++) {
#pragma omp task \
          depend(in: d[out][i-1], d[out][i+1]) \
          depend(out: d[out][i])
    { compute_cell(out, i); }
  }

  #pragma omp taskwait
  exchange_boundaries();
}
```

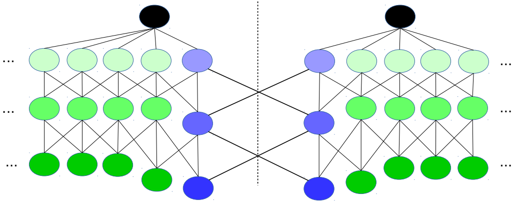


DASH + OpenMP

- Synchronization through collectives
- Synchronization slack (imbalances)
- Complex to further taskify

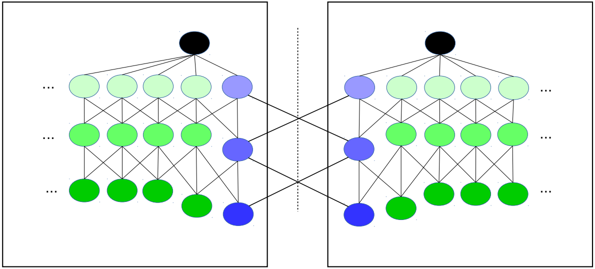


Global Task Dependencies

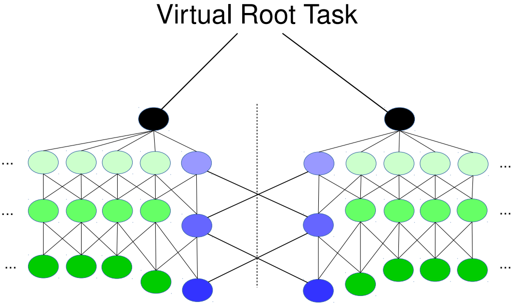


Global Data Dependencies

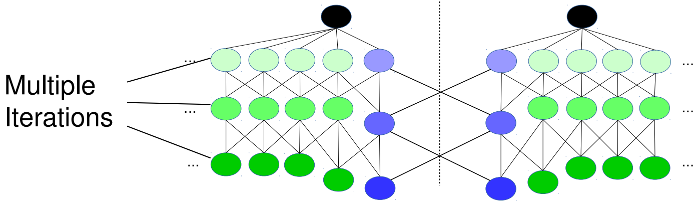
Independent Scheduler Instances



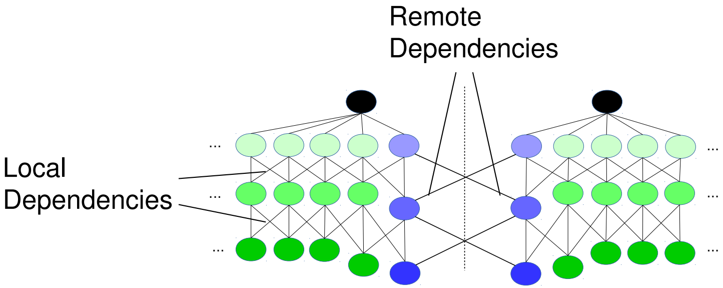
Global Data Dependencies



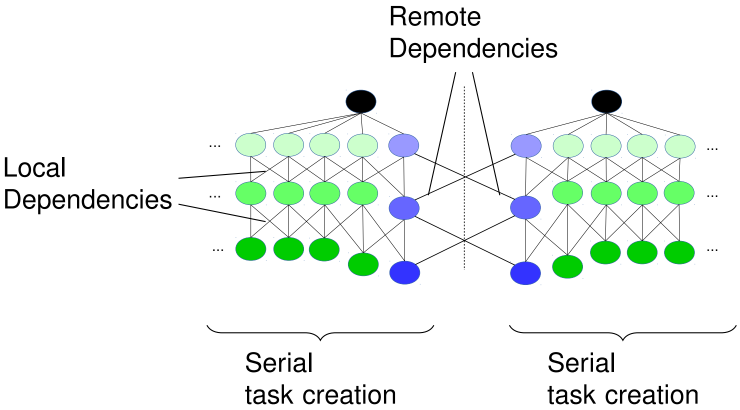
Global Data Dependencies



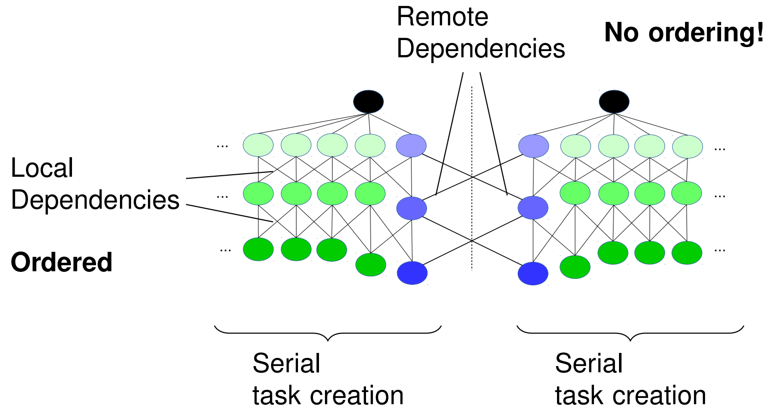
Global Data Dependencies



Global Data Dependencies



Global Data Dependencies



Remote Dependencies

No ordering!

Local Dependencies

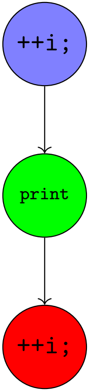
Ordered

Serial task creation

Serial task creation

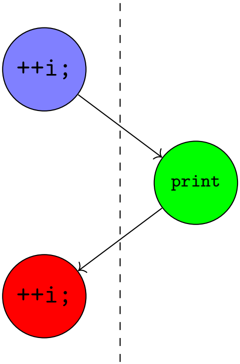
A simple example

```
int i;
[...]
#task inout(i)
{ ++i; }



#task in(i)
{ print(i); }



#task inout(i)
{ ++i; }
[...]
```



A simple example

```
// Unit 0
dash::Shared<int> i;
[...]

#task inout(i)
{ ++i; }




#task inout(i)
{ ++i; }


[...]
```



```
// Unit 1
dash::Shared<int> i;
[...]

#task in(i)
{ print(i); }

[...]
```

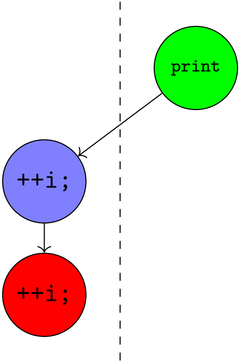
A simple example

```
// Unit 0
dash::Shared<int> i;
[...]
```

```
#task inout(i)
{ ++i; }
```

```
#task inout(i)
{ ++i; }
```

```
[...]
```

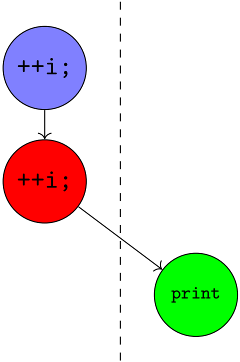


```
// Unit 1
dash::Shared<int> i;
[...]

#task in(i)
{ print(i); }
```

```
[...]
```

A simple example

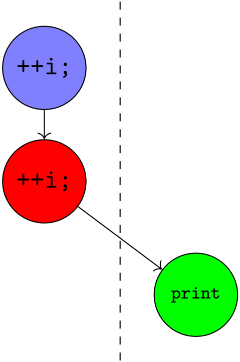
```
// Unit 0
dash::Shared<int> i;
[...]

#task inout(i)
{ ++i; }


#task inout(i)
{ ++i; }


[...]
```



```
// Unit 1
dash::Shared<int> i;
[...]


#task in(i)
{ print(i); }


[...]
```

A simple example

```
// Unit 0
dash::Shared<int> i;
[...]

#task inout(i)
{ ++i; }


#task inout(i)
{ ++i; }

[...]
```



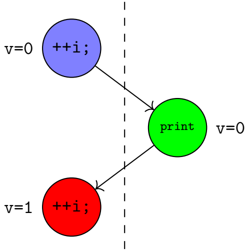
```
// Unit 1
dash::Shared<int> i;
[...]

#task in(i)
{ print(i); }

[...]
```

How to restore global task ordering?

Global Task Ordering

Proposed Solution: **Dependency Versions**

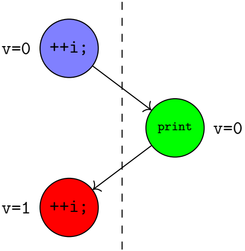
- *Logical clock* for dependencies
- Additional information provided by the user
- Versioning splits execution in *phases*
- Task-based synchronization between phases
- Overlap of phases

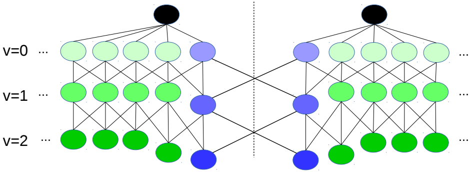


Global Task Ordering

Proposed Solution: **Dependency Versions**

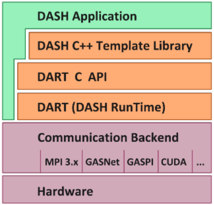
- *Logical clock* for dependencies
- Additional information provided by the user
- Versioning splits execution in *phases*
- Task-based synchronization between phases
- Overlap of phases



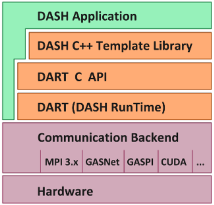


DASH Prototype Implementation

- Nested tasking runtime
- Dependencies:
  - Local input/output dependencies (similar to OpenMP)
  - Remote input dependencies



DASH Prototype Implementation

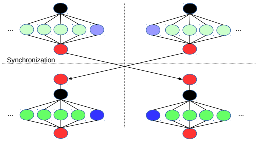


- Nested tasking runtime
- Dependencies:
  - Local input/output dependencies (similar to OpenMP)
  - Remote input dependencies
- Active message queue based on MPI-RMA
- Re-scheduling `task-yield` (using `makecontext (3)`)
- Global task cancellation
- Priorities

Example: Preliminary DASH interface

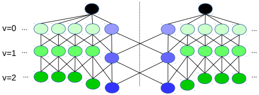
```
using MatrixT = dash::Narray<2, double>;
MatrixT mat1(N, M, dash::BLOCKED, dash::NONE);
MatrixT mat2(N, M, dash::BLOCKED, dash::NONE);
initialize(mat1, mat2);
auto lbegin = mat1.local_offset(0), lend = lbegin + mat1.local_extent(0);


for (size_t ts  = 0; ts < TIMESTEPS; ++ts) {
  auto& mat_old = (ts%2)? mat1 : mat2;
  auto& mat_new = (ts%2)? mat2 : mat1;
  for (auto row = lbegin+1; row < lend-1; ++row) {

             compute_row(mat_old[row], mat_new[row]);






  }
  // handle boundary rows

  dash::barrier(); // wait for all other processes
}
```

Example: Preliminary DASH interface



```
auto tphase = dash::TaskPhase(dash::Team::All());

for (size_t ts  = 0; ts < TIMESTEPS; ++ts) {
  auto& mat_old = (ts%2)? mat1 : mat2;
  auto& mat_new = (ts%2)? mat2 : mat1;
  for (auto row = lbegin+1; row < lend-1; ++row) {
    dash::async(
      [&](){ compute_row(mat_old[row], mat_new[row]); },




    );

  }
  // handle boundary rows
  dash::complete(); // wait for all local tasks
  dash::barrier(); // wait for all other processes
}
```

Example: Preliminary DASH interface

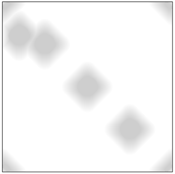


```
auto tphase = dash::TaskPhase(dash::Team::All());

for (size_t ts  = 0; ts < TIMESTEPS; ++ts) {
  auto& mat_old = (ts%2)? mat1 : mat2;
  auto& mat_new = (ts%2)? mat2 : mat1;
  for (auto row = lbegin; row < lend; ++row) {
    tphase.add(
      [&](){ compute_row(mat_old[row], mat_new[row]); },
      dash::in( mat_old[row - 1]), // input:  upper row, prev. iteration
      dash::in( mat_old[row + 1]), // input:  lower row, prev. iteration
      dash::in( mat_old[row]),     // input:  this  row, prev. iteration
      dash::out(mat_new[row])      // output: this  row, this  iteration
    );


  }
  tphase.advance(); // advance to next phase
}
tphase.complete(); // wait for all local tasks
```

Example: Preliminary DASH interface

```
auto tphase = dash::TaskPhase(dash::Team::All());

for (size_t ts  = 0; ts < TIMESTEPS; ++ts) {
  auto& mat_old = (ts%2)? mat1 : mat2;
  auto& mat_new = (ts%2)? mat2 : mat1;
  for (auto row = lbegin; row < lend; ++row) {
   tphase.add(
      [&](){ compute_row(mat_old[row], mat_new[row]); },
      dash::in( mat_old[row - 1]), // input:  upper row, prev. iteration
      dash::in( mat_old[row + 1]), // input:  lower row, prev. iteration
      dash::in( mat_old[row]),     // input:  this  row, prev. iteration
      dash::out(mat_new[row])      // output: this  row, this  iteration
    );
    if (ts%10 == 0) phase.add([&](){ postprocess(mat_new[row]); },
                              dash::inout(mat_new[row])); // dominant dep.
  }
  tphase.advance(); // advance to next phase
}
tphase.complete(); // wait for all local tasks
```

First Results

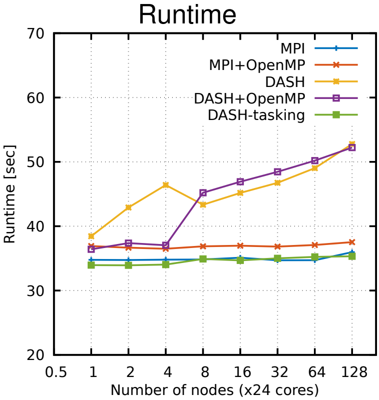


Benchmark:

- 2D heat diffusion
- Row-wise domain decomposition
- 100 iterations
- 409600 elements per row, double-buffered
- MPI, MPI+OpenMP, DASH, DASH+OpenMP, DASH Tasks
- System under test: Haswell-based Linux Cluster (IB)[1]
  - GCC 6.3.0
  - Intel MPI 18.0

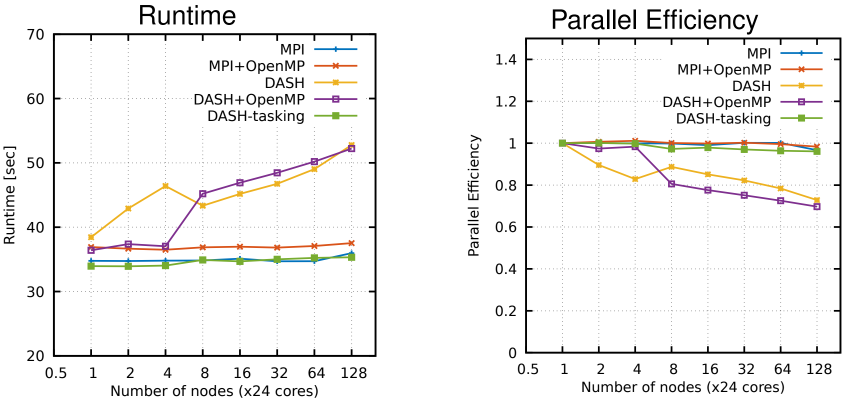
[1] The only stable configuration at the time.

First results: Weak Scaling



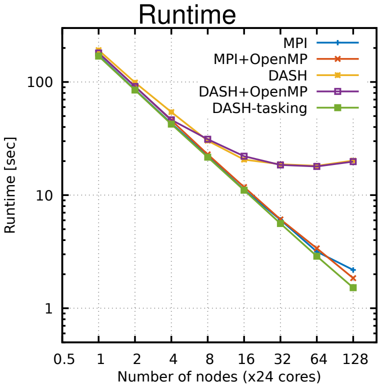
- 4800 rows per node

First results: Weak Scaling



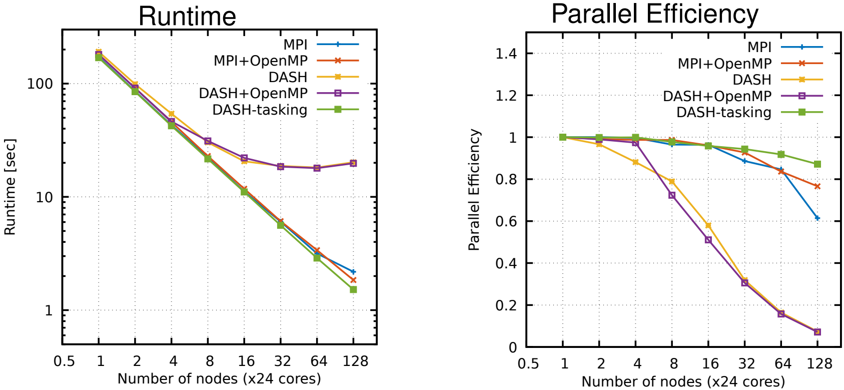
- 4800 rows per node

First results: Strong Scaling



- 24000 rows

First results: Strong Scaling



- 24000 rows

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

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 `dash::barrier()` with non-blocking `dash::TaskPhase::advance()`

Future Work

- Improvements to the scheduler
- Finalize C++ interface (execution policies, anyone?)
- Tool support
- Interoperability with OpenMP (OmpSs?)
- Adapt applications

Questions?

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