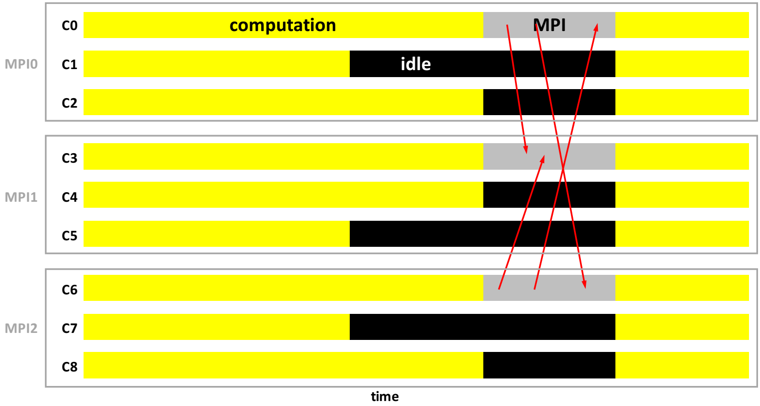
Fine-Grained Synchronization using Global Task Dependencies in DASH

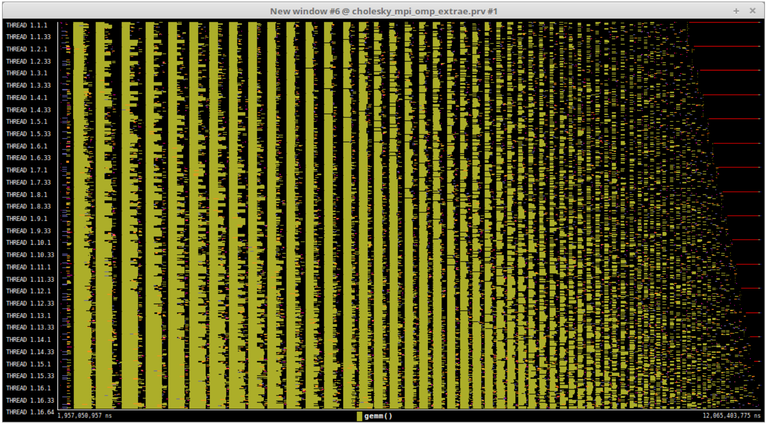
Joseph Schuchart, **José Gracia**

October 2018, WSSP

Motivation: Strong synchronisation in MPI applications



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

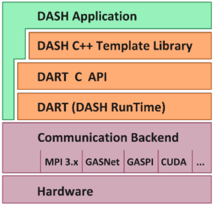
Preliminary Evaluation

Conclusion and Future Work

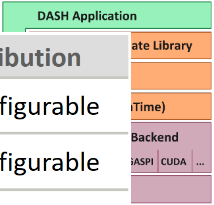
Background

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

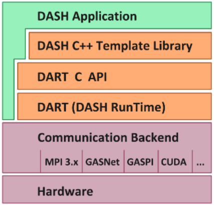


Container	Description	Data distribution
Array<T>	1D Array	static, configurable
NArray<T, N>	N-dim. Array	static, configurable
Shared<T>	Shared scalar	fixed, configurable
Directory*<T>	Variable-size, locally indexed Array	manual
CoArray*<T>	Similar to CAF	uniform

(*) 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]

```
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]

```
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)
```

DASH requires:

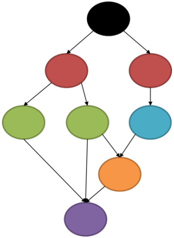
- Scalable distributed task creation
- Implicit, *data-centric* global synchronization
- Recurring dependency patterns

A step back: OpenMP Tasks

OpenMP supports asynchronous tasks since v3.0

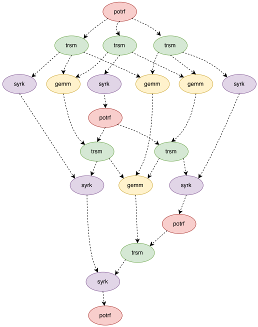
Synchronization: Task data dependencies since v4.0

- Describe data flow to form task graph
- Implicit synchronization among *sibling tasks*
- Strict backward matching



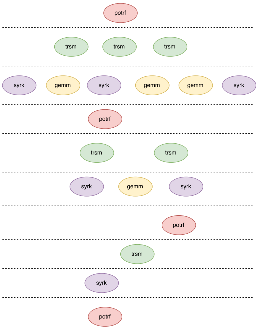
Blocked Cholesky Factorization

```
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);
  }
}
```



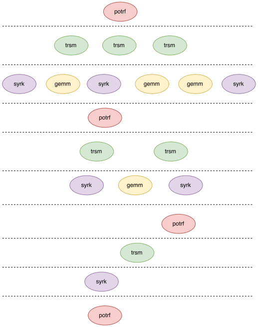
Blocked Cholesky Factorization

```
  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();
  }
```



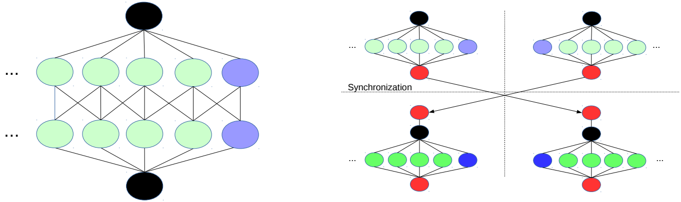
Limitations of DASH + OpenMP

- Synchronization through collectives
  dependencies across nodes have to be enforces by barriers
- Synchronization may lead to imbalances
- Complex/impossible to further taskify



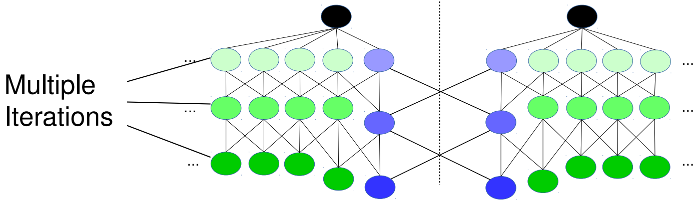
Limitations of DASH + OpenMP

- Synchronization through collectives
  dependencies across nodes have to be enforces by barriers
- Synchronization may lead to imbalances
- Complex/impossible to further taskify

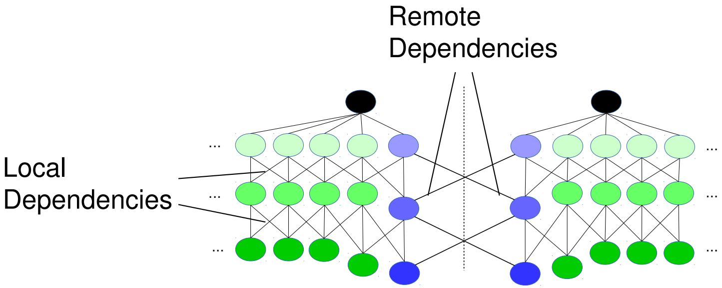


Global Data Dependencies with DASH

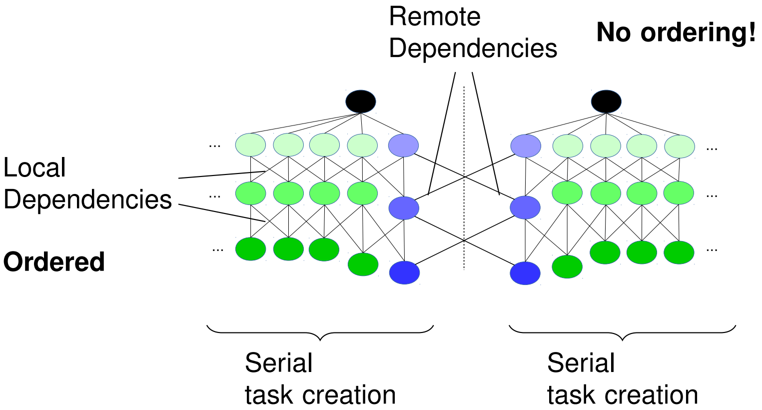
Global Data Dependencies



Global Data Dependencies

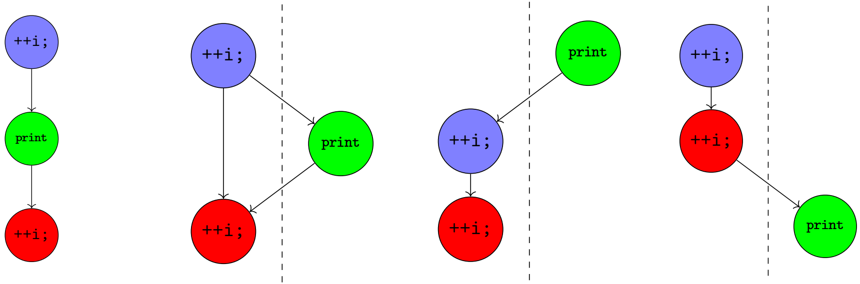


Global Data Dependencies



Global Task Ordering

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

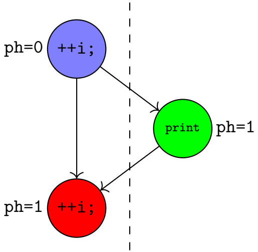


How to restore global task ordering?

Global Task Ordering

Proposed Solution: **Task Phases**

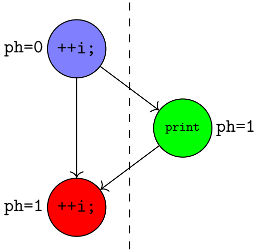
- *Logical clock* for tasks and dependencies
- Virtual barrier in task-based execution
- *Happens before*: `input` dependencies refer to earlier phase
- Additional information provided by the user

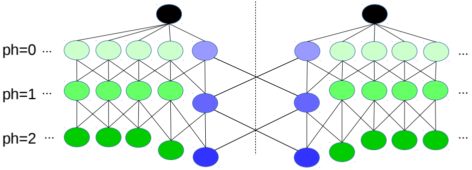


Global Task Ordering

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





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)

```
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 synchro-nization to ensure happens-before relation

Blocked Cholesky Factorization (DASH Tasks)

```
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();
```

`async` introduces new task using C++ lambda notation

Blocked Cholesky Factorization (DASH Tasks)

```
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_ji); },
          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)

```
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();
```

`input` dependency refers to `output` dependency in previous phase

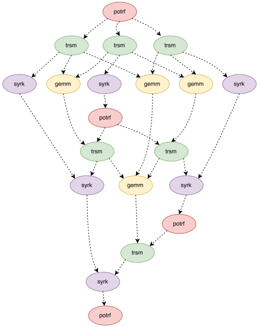
Blocked Cholesky Factorization (DASH Tasks)

```
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();
```

Wait for completion of all tasks

Blocked Cholesky Factorization (DASH Tasks)

```
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

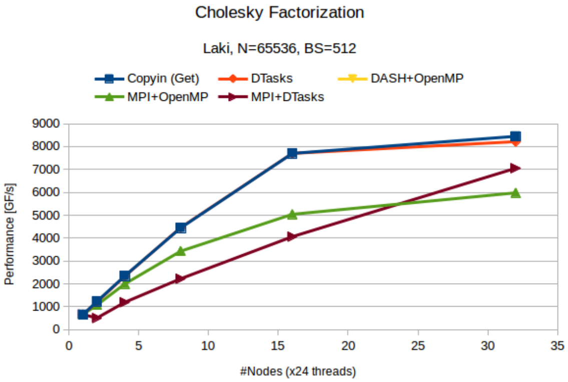
Systems under test:

- 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

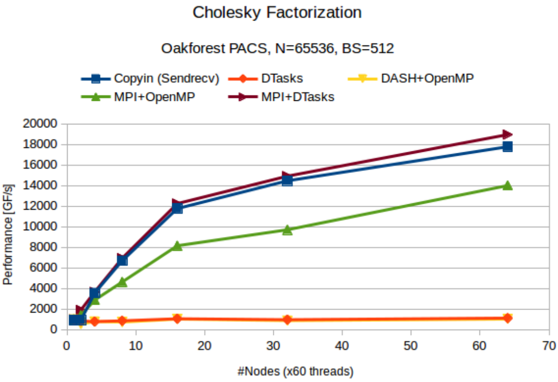
Evaluation Kernel

- 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 commuication task
  - `MPI+DTasks`: block-wise communication tasks (test-yield-loop)

Results: Laki



Results: Oakforest PACS



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

dash

SPPEXA

Deutsche Forschungsgemeinschaft DFG

Bibliography

B. L. Chamberlain, D. Callahan, and H. P. Zima.
Parallel programmability and the Chapel language.
International Journal of High Performance Computing Applications, 21, 2007.

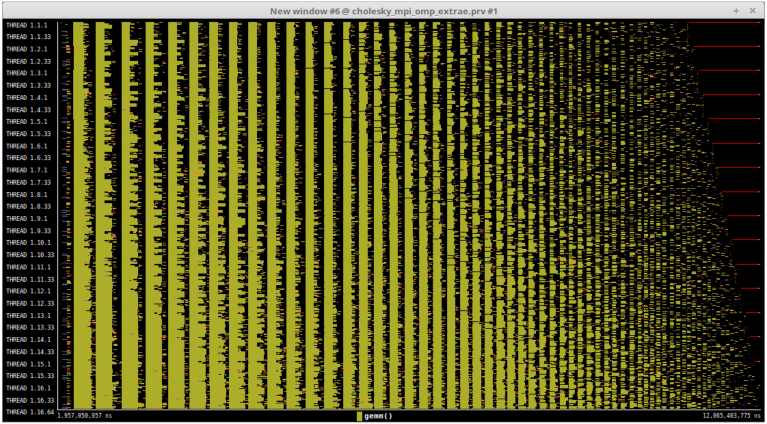
R. Hoque, T. Herault, G. Bosilca, and J. Dongarra.
Dynamic Task Discovery in PaRSEC: A Data-flow Task-based Runtime.
In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '17, 2017.

H. Kaiser, T. Heller, B. Adelstein-Lelbach, A. Serio, and D. Fey.
HPX: A Task Based Programming Model in a Global Address Space.
PGAS '14. ACM, 2014.

V. Kumar, Y. Zheng, V. Cavé, Z. Budimlić, and V. Sarkar.
Habaneroupc++: A compiler-free pgas library.
In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14. ACM, 2014.

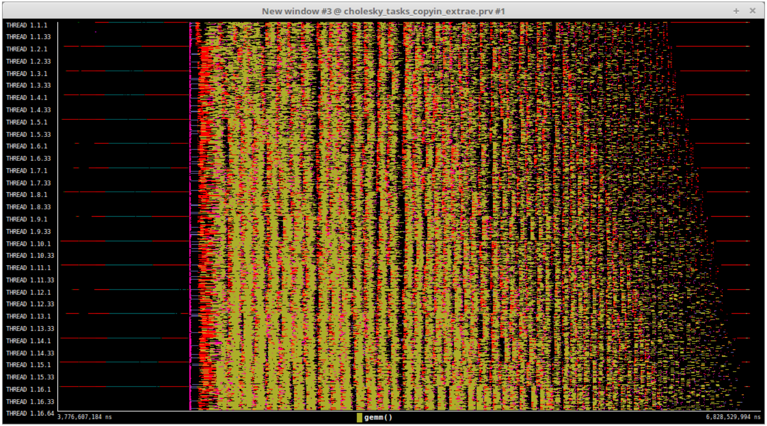
K. Tsugane, J. Lee, H. Murai, and M. Sato.
Multi-tasking Execution in PGAS Language XcalableMP and Communication Optimization on Many-core Clusters.
In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, 2018.

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