A Distributed Task Scheduler in the DASH Project

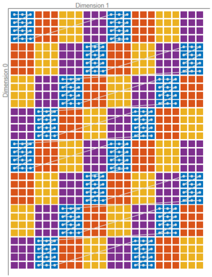
Joseph Schuchart
José Gracia

March 21, 2019

DASH PGAS Library

- ✓ Distributed data structures
- ✓ Locality-awareness
- ✓ Remote Memory Access:
  - ✓ Random one-sided access
  - ✓ Decoupled transfer / synchronization
  - ✓ MPI-3 RMA
- ✓ STL-like algorithms

! Decoupled transfer / synchronization



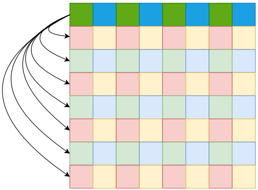
Container	Description	Data distribution
Array<T>	1D Array	static, configurable
NArray<T, N>	N-dim. Array	static, configurable
Shared<T>	Shared scalar	fixed, configurable
Directory*<T>	Variable-size, locally indexed Array	manual
CoArray*<T>	Similar to CAF	uniform

(*) Under construction

Synchronization in DASH

- Collective synchronization

```
dash::Matrix<double> matrix{N, N};

for (size_t j = 0; j < N; ++j) {
  if (matrix(0, j).is_local())
      matrix(0, j) = compute(j);
}

// wait for all blocks to be computed
dash::barrier();

for (size_t i = 1; i < N; ++i) {
  for (size_t j = 1; j < N; ++j) {
    if (matrix(i, j).is_local())
      combine(matrix(i,   j),
              matrix(0, j));
  }
}
```

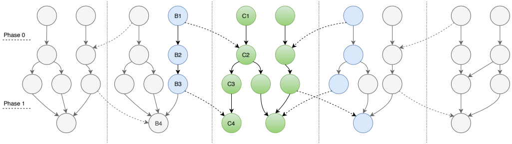


- CoEvents in CoArray port
- Global mutex

Decoupled Synchronization & Transfer

Data-centric

One-sided, Locality-aware Task Discovery

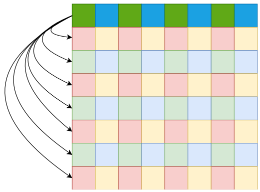


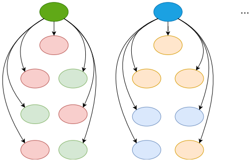
A Humble Classification

	One-sided	Locality-aware	Decoupled	Data-centric
HPX	✓	X	X	X
UPC++	✓	✓	X	X
StarPU (MPI)	X	✓	X	✓
StarPU (STF)	✓	X	X	✓
XscalableMP	X	✓	(✓)	✓
PaRSEC	✓	X	X	✓
MPI+OpenMP	X	✓	X	(✓)
DASH	✓	✓	✓	✓

Global Task Data Dependencies

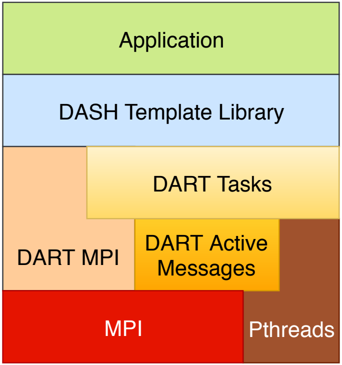
```
dash::Matrix<double> matrix{N, N};


for (size_t j = 0; j < N; ++j) {
  if (matrix(0, j).is_local())

      matrix(0, j) = compute(j);

}

// wait for all blocks to be computed
dash::barrier();

for (size_t i = 1; i < N; ++i) {
  for (size_t j = 1; j < N; ++j) {
    if (matrix(i, j).is_local())

        apply(matrix(i, j),
              matrix(0, j));

  }
}
```



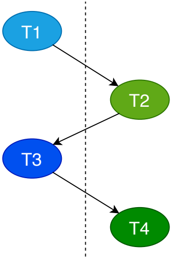


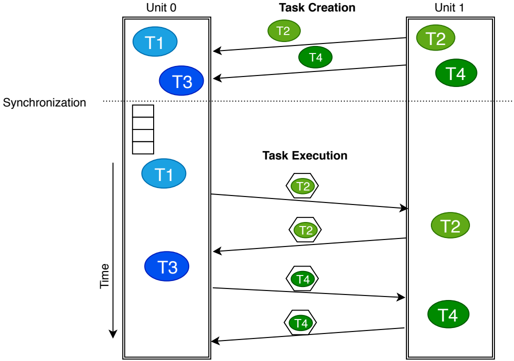

```
dash::Matrix<double> matrix{N, N};
```

Architecture



Inter-Scheduler Communication





Task-loops

```
dash::Matrix<double> matrix{N, N};

#pragma omp parallel for
for (auto iter = matrix.lbegin(); iter != matrix.lend(); ++iter)
{
  *iter *= 2;
}
```

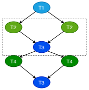
⇓

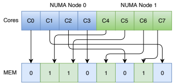
```
dash::taskloop(matrix.lbegin(), matrix.lend(),
  [&](auto begin, auto end){
  for (auto iter = begin; iter != end; ++iter)
  {
    *iter *= 2;
  }


});
```

```
dash::taskloop(matrix.lbegin(), matrix.lend(), dash::chunk_size(S),
  [&](auto begin, auto end){
  for (auto iter = begin; iter != end; ++iter)
```

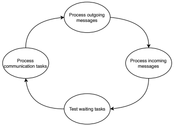
New Features / Improvements

Phase windows





NUMA-aware task scheduling

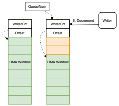


Progress Thread

Detached tasks

Lock-free data structures

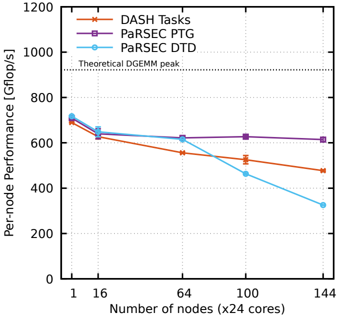
RMA-based active message queue



Example: Blocked Cholesky Factorization

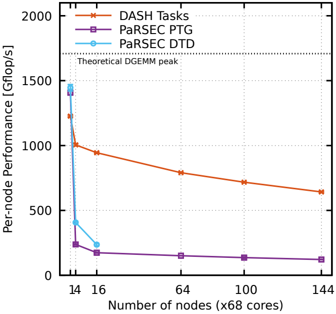
```
for (int k = 0; k < num_blocks; ++k) {
  if (mat.block(k,k).is_local())
    dash::async([&](){ potrf(matrix.block(k,k)); },
                dash::out(mat.block(k,k)));

  dash::async_fence(); // <- advance to next phase
  for (int i = k+1; i < num_blocks; ++i)
    if (mat.block(k,i).is_local())
      dash::async([&](){ trsm(cache[k][k], matrix.block(k,i)); },
                  dash::copyin(mat.block(k,k), cache[k][k]),
                  dash::out(mat.block(k,i)));

  dash::async_fence(); // <- advance to next phase
  for (int i = k+1; i < num_blocks; ++i) {
    for (int j = k+1; j < i; ++j) {
      if (mat.block(j,i).is_local())
        dash::async([&](){ gemm(cache[k][i], cache[k][j], mat.block(j,i)); },
                    dash::copyin(mat.block(k,i), cache[k][i]),
                    dash::copyin(mat.block(k,j), cache[k][j]),
                    dash::out(mat.block(j,i)));
    }

    if (mat.block(i,i).is_local())
      dash::async([&](){ syrk(cache[k][i], mat.block(i,i)); },
                  dash::copyin(mat.block(k,i), cache[k][i]),
                  dash::out(mat.block(i,i)));
  }
  dash::async_fence();  // <- advance to next phase
}
dash::complete(); // <- wait for all tasks to execute
```

Results: Cholesky



Cray XC40, $N = 25k/Node$, $NB = 320$

Oakforest PACS, $N = 25k/Node$, $NB = 320$

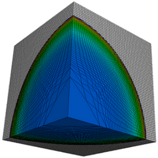


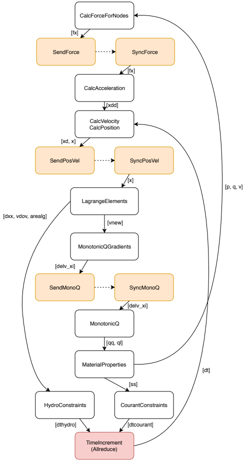
Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)

- 28pt stencil
- DoE CoDesign applications
- Abstraction levels: nodes, elements, regions

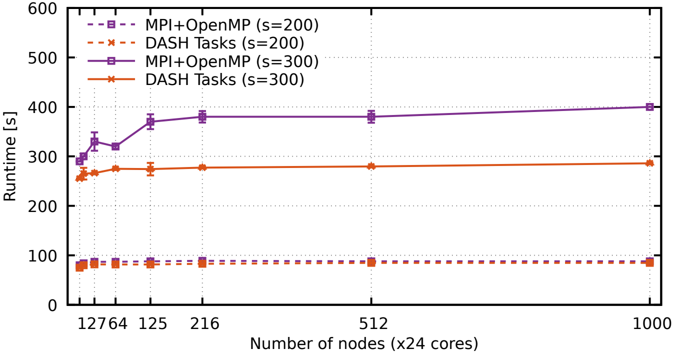
Porting Strategy

1. MPI Send/recv $\Rightarrow$ `dash::copy`*
2. `omp parallel for` $\Rightarrow$ `dash::taskloop`
3. Local dependencies
4. Remote dependencies





Results: Lulesh @ Cray XC40



Conclusion

PGAS synchronization should provide:

- Decoupled transfers ✓
- Complex synchronization patterns ✓
- Locality-aware programming ✓
- Scalability ✓

Future Work: MUST for correctness checking

Global Task Data-Dependencies in PGAS Applications* **

Joseph Schuchart and Jose Grácia

High Performance Computing Center Stuttgart (HLRS)
Nobelstraße 19
70569 Stuttgart, Germany
{schuchart,gracia}@hlrs.de

Abstract. Recent years have seen the emergence of two independent programming models challenging the traditional two-tier combination of message passing and loop-based work-sharing: partitioned global address space (PGAS) and task-based concurrency. In the PGAS programming model, synchronization and communication between processes are decoupled, providing significant potential for reducing communication overhead. At the same time, task-based programming allows to exploit a large degree of shared-memory concurrency. The inherent lack of fine-grained synchronization through communication in PGAS can be addressed through fine-grained task synchronization across process boundaries. In this work, we propose the use of task data dependencies describing the data-flow in the global address space between tasks created in parallel on multiple processes. We present a description of the global data dependencies, describe the necessary interactions between the distributed scheduler instances required to handle them, and discuss our implementation in the context of the DASH C++ PGAS framework. We evaluate our approach using the Blocked Cholesky Factorization and the LULESH proxy app, demonstrating the feasibility and scalability of our approach.

Keywords: Parallel Programming, PGAS, Task Parallelism

1 Introduction

The decoupling of communication and synchronization in the partitioned global address space (PGAS) programming model allows applications to better exploit

* We gratefully acknowledge funding by the German Research Foundation (DFG) through the German Priority Programme 1648 Software for Exascale Computing (SPPEXA) in the SmartDASH project and would like to thank all members of the DASH team.

** Joseph Schuchart is a doctoral student at the University of Stuttgart and the main author, claiming exclusive authorship of the design and implementation of the API and distributed scheduler as well as leading authorship of the global task dependency design, the evaluation scenario selection, and interpretation of experimental results.

Questions?