Smart-DASH Update and Status

SPPEXA Annual Plenary Meeting 2018
March 21-22, Garching

www.dash-project.org

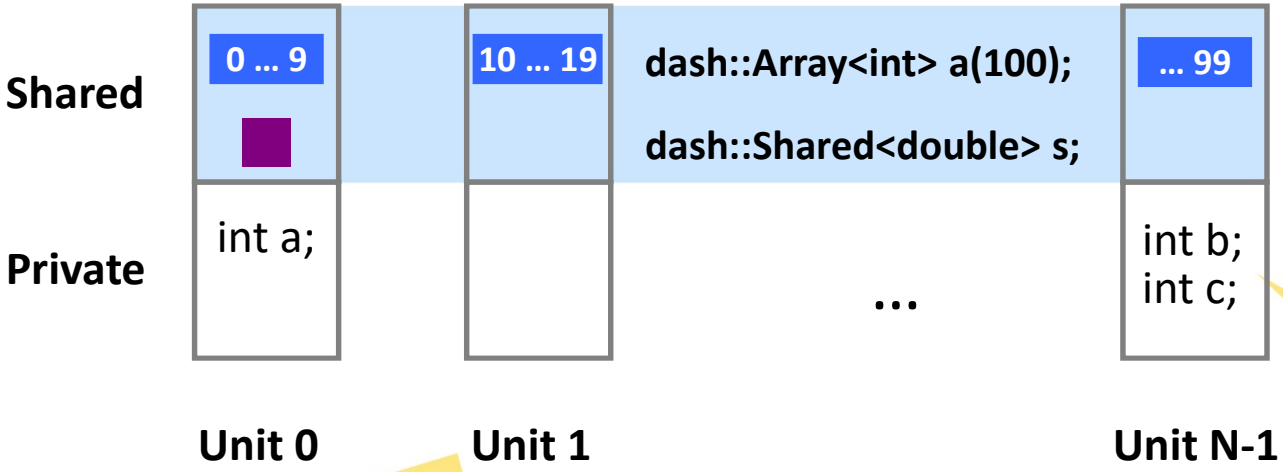
Karl Fürlinger
Ludwig-Maximilians-Universität München

Outline

- Overview of DASH
  - Project partners
  - Hello world in DASH
- Recent additions to DASH
  - The dash::Coarray abstraction
  - Parallel sorting algorithm in DASH
  - (Memory spaces)
- Ongoing and future work
  - Tasking in DASH
  - Managed data replication
  - (“Smart” data structures)
  - Collaborations inside and outside of SPPEXA

DASH — Overview

- DASH is a C++ template library that offers
  - **Distributed data structures** and **parallel algorithms**
  - A complete **PGAS** (part. global address space) programming system without a custom (pre-)compiler
- Terminology:

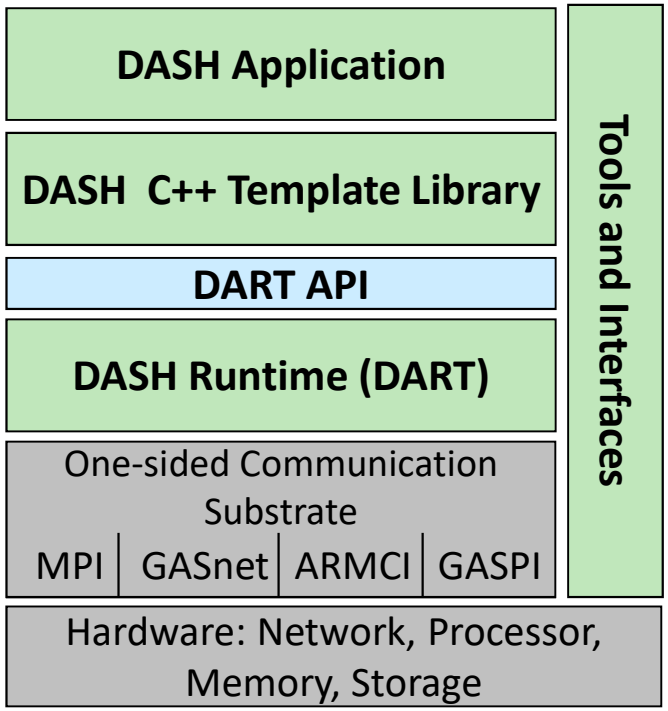


Shared data: managed by DASH in a virtual global address space

Private data: managed by regular C/C++ mechanisms

Unit: The individual participants in a DASH program, usually full OS processes.

DASH — Project Structure



	Phase I (2013-2015)	Phase II (2016-2018)
LMU Munich	Project management, C++ template library	Project management, C++ tempalte library, DASH data dock
TU Dresden	Libraries and interfaces, tools support	Smart data structures, resilience
HLRS Stuttgart	DART runtime	DART runtime
KIT Karlsruhe	Application case studies	
IHR Stuttgart		Smart deployment, Application case studies

www.dash-project.org

DASH is one of 16 SPPEXA projects

Hello World in DASH

```
#include <iostream>
#include <libdash.h>
using namespace std;

int main(int argc, char* argv[])
{
  dash::init(&argc, &argv);
  auto myid = dash::myid();
  auto size = dash::size();
  cout <<"'Hello world' from unit " << myid
       << " of " << size << endl;

  dash::Array<int> arr(100);
  dash::fill(arr.begin(), arr.end(), myid);
  dash::barrier();

  if( myid==0 ) {
    for( auto el: arr )
      cout << (int) el << '
    cout << endl;
  }
  dash::finalize();
}
```

SPMD programming model, like MPI

100 elements, distributed over all units, accessible from all units

Parallel algorithm, executed by all units in parallel (owner-computes model)

Integration with standard C++ features

```
$ mpirun -n 4 ./hello
'Hello world' from unit 0 of 4
...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3
```

DASH Distributed Data Structures Overview

Container	Description	Data distribution
Shared<T>	Shared Scalar	
Array<T>	1D Dist. Array	
NArray<T, N>	N-dim. Dist. Array	
Coarray<T[R][S]>	CAF-like Coarray	New!
List(*)<T>, **Map**(*)<T>	Dynamic data structures (growing/shrinking)	

(*) Under Construction

The DASH Coarray (1)

- Modeled after the coarray concept in Fortran
  - Originally an extension to F95, then part of Fortran 2008
  - Some shortcomings to be addressed in Fortran 2018 (e.g., teams)

```
dash::Coarray<int> x;
int l = x;
int r = x(3);
```

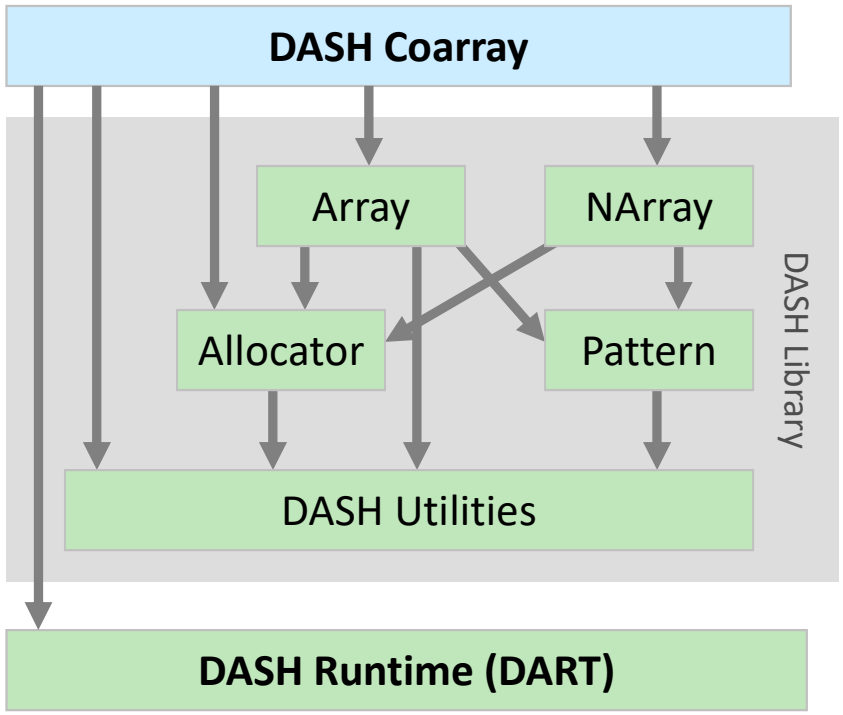
```
dash::Coarray<int[5]> y;
int l = y[1];
int r = y(3)[1];
```

```
dash::Coarray<int[2][4]> z;
int l = y[1][2];
int r = y(3)[1][2];
```

- Access to local data “as usual”
- Access to remote data using round-bracket (d) notation holding co-dimension

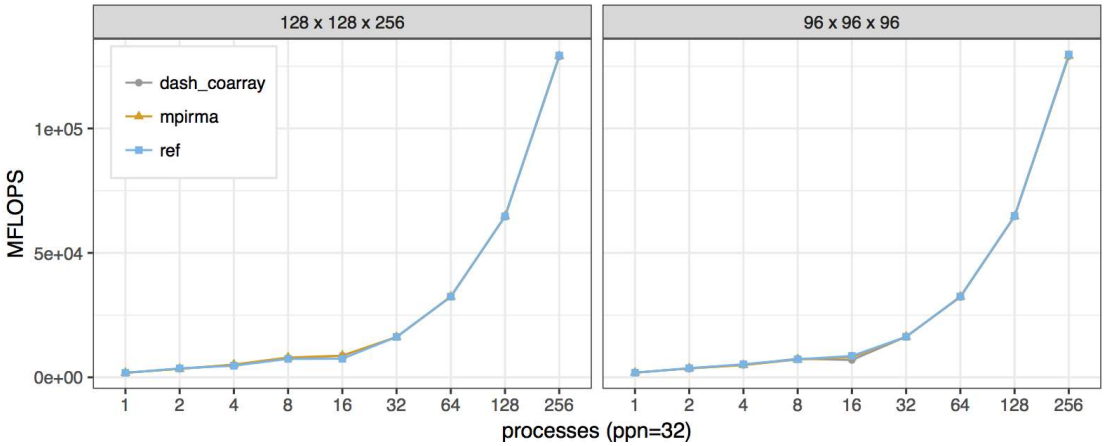
The DASH Coarray (2)

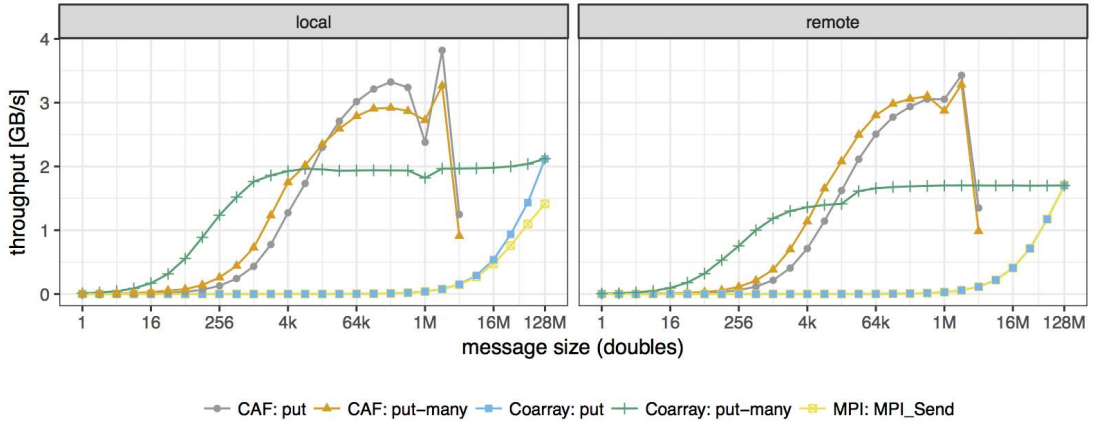
- Coarray builds on the existing DASH infrastructure
  - Implements the container concept
  - Fully inter-operable with existing DASH functionality




```
dash::Coarray<int[10][20]> x;

//global iterators to full range
GlobIter<int> gbegin = x.begin();
GlobIter<int> gend   = x.end();

// global iterator to remote data
GlobIter<int> rbegin = x(1).begin();
GlobIter<int> rend   = x(1).end();

// local it to local range (pointer)
int *lbegin = x.local.begin();
int *lend   = x.local.end();

dash::for_each(x(1).begin(),
               x(3).end(), ...);
```

The DASH Coarray (3) — Early Results





- HPCCG
  - CG solver on 3D chimney domain
  - NERSC Cori, Haswell Nodes
- CAF-Bench
  - Benchmark of communication patterns
  - LRZ SuperMUC, IntelMPI

Source and more details: Felix Mössbauer, Roger Kowalewski, Tobias Fuchs and Karl Fürlinger *"A Portable Multidimensional Coarray for C++"*, to appear in Proceedings of the 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2018)

DASH Algorithms (1)

- STL algorithms can be used with DASH containers
  - Both on the **local view** and the **global view**

```
#include <libdash.h>

int main(int argc, char* argv[])
{
  dash::init(&argc, &argv);

  dash::Array<int> a(1000);

  if( dash::myid()==0 ) {
    // global iterators and std. algorithms
    std::sort(a.begin(), a.end());
  }

  // local access using local iterators
  std::fill(a.lbegin(), a.lend(), 23+dash::myid());

  dash::finalize();
}
```

STL algorithms work with DASH global iterators

STL algorithms work with DASH local iterators

DASH Algorithms (2)

- There are DASH equivalents for STL algorithms, e.g., dash::**fill**, dash::**for_each**, etc.

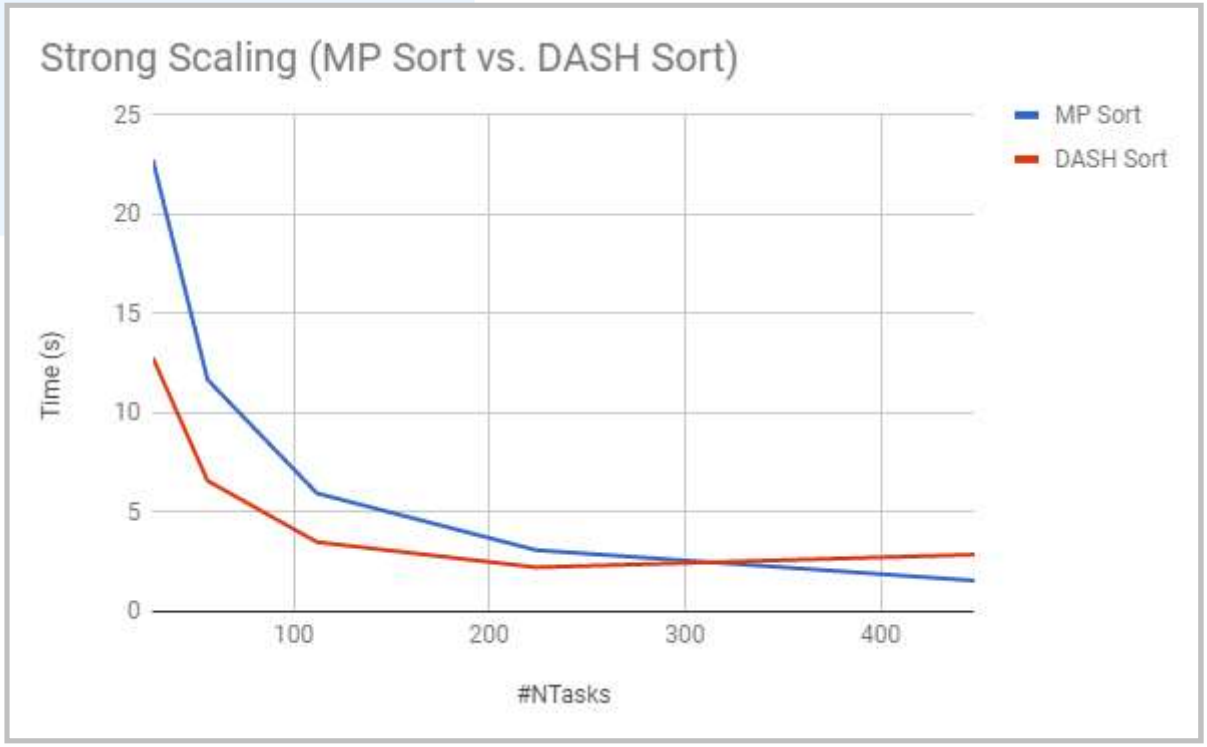
- Examples

– dash::fill	arr[i] <- val
– dash::generate	arr[i] <- func()
– dash::for_each	func(arr[i])
– dash::transform	arr2[i] = func(arr1[i])
– dash::accumulate	sum(arr[i]) (0 <= i < n)
– dash::min_element	min(arr[i]) (0 <= i < n)
– dash::max_element	min(arr[i]) (0 <= i < n)
– dash::sort New!	

DASH Algorithms (3)

- Example: sort a distributed array

```
dash::Array<int> arr(100, dash::BLOCKED);

for( auto i=0; i<arr.lsize(); i++ ) {
  arr.local[i]=rand()%100;
}
arr.barrier();

dash::sort(arr.begin(),
           arr.end());
```

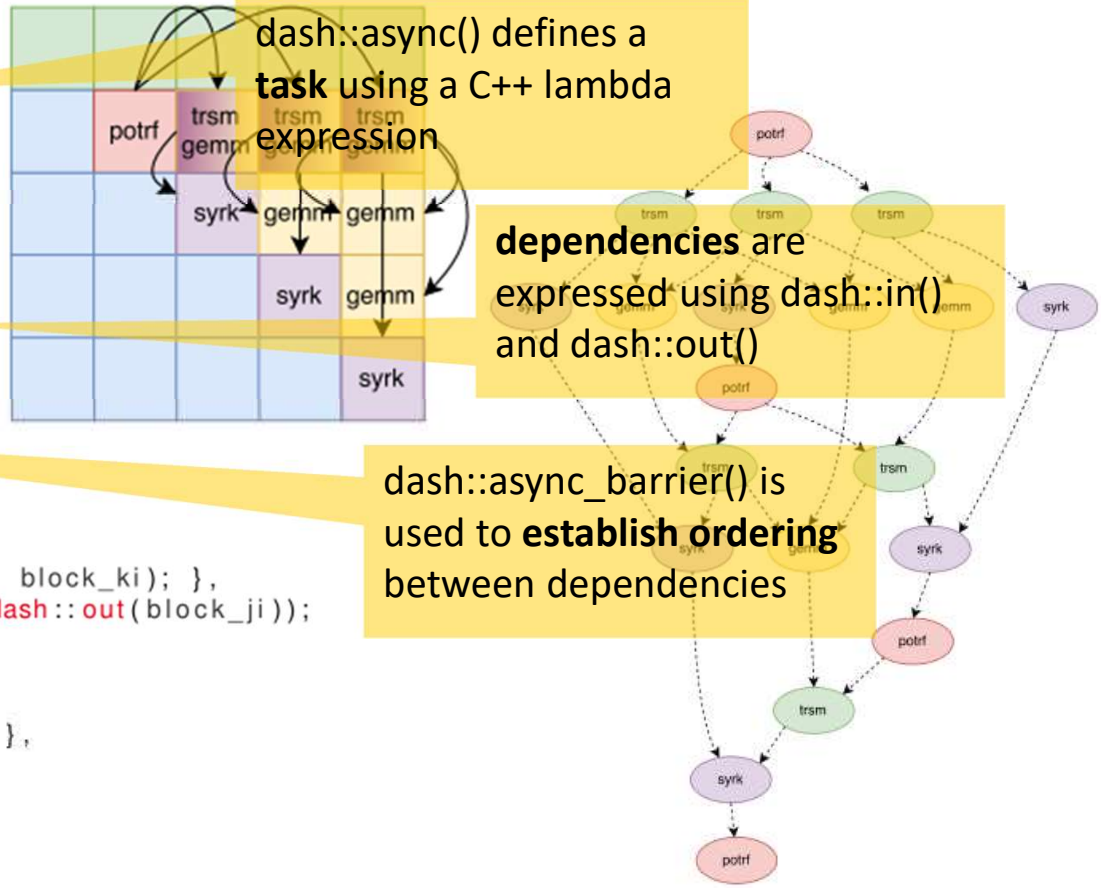


Sorting ~$1.8*10^9$
Elements (64-bit int)

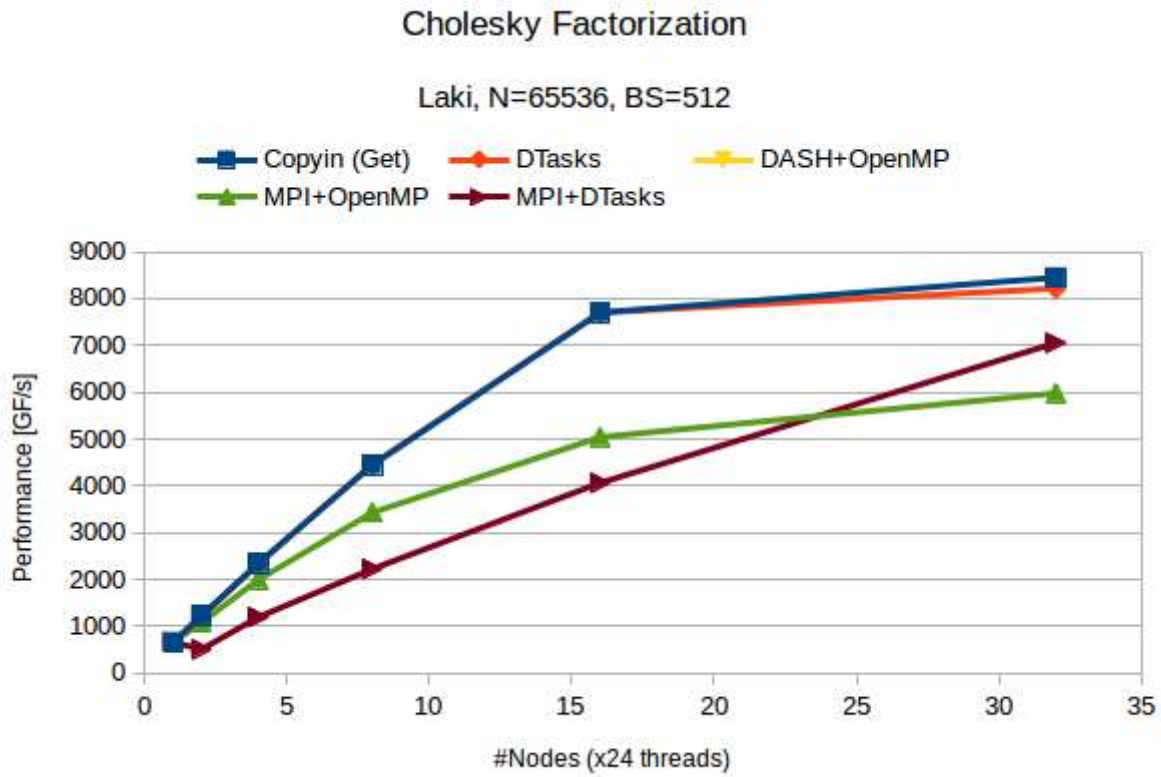
Ongoing and Future Work - Tasking

- Tasking abstraction inspired by OpenMP but with remote dependencies, example: Cholesky factorization

```
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();
```



DASH Tasking – Early Results



More details: "Global Task Data-Dependencies in PGAS Applications", Joseph Schuchart and Jose Gracia, upcoming publication

Managed Data Replication in DASH

- Goal: allocate a copy of (parts of) a data structure in a specified memory space

```
dash::Replica<int, NVRAM, READ_ONLY> rep(arr.begin(),
                                          arr.begin()+10);
```

Where the data is replicated to

How the data will be used

Which data is replicated

- Usage scenarios / benefits
  - Get working-set close to compute cores
  - Utilize bulk-transfers instead of element-wise copies
  - Specify intent that can be checked/supported at runtime/compile time

Specifying Intent

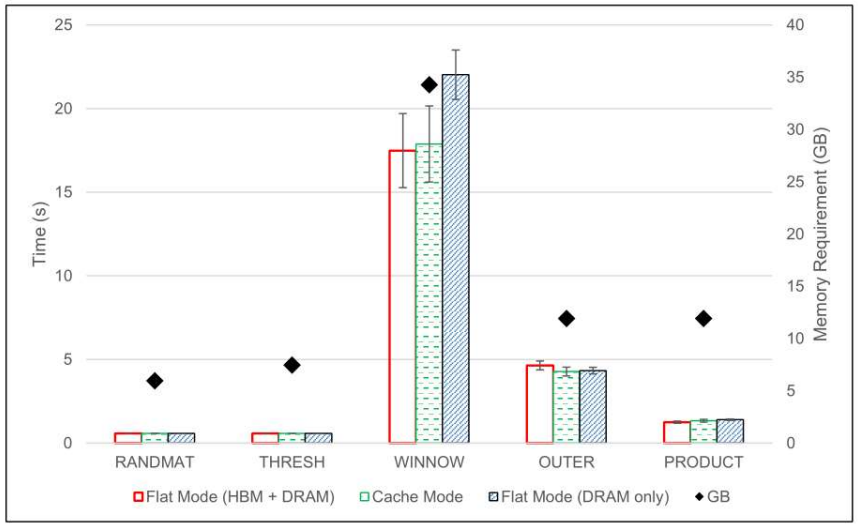
- Local copies with specified intent

```
// local copy of the first 10 elements
dash::Replica<int, READ_ONLY> rep(arr.begin(),
                                  arr.begin()+10);

cout << rep[0]; // OK
rep[0] = 33;    // ERROR lc is a read only copy
```

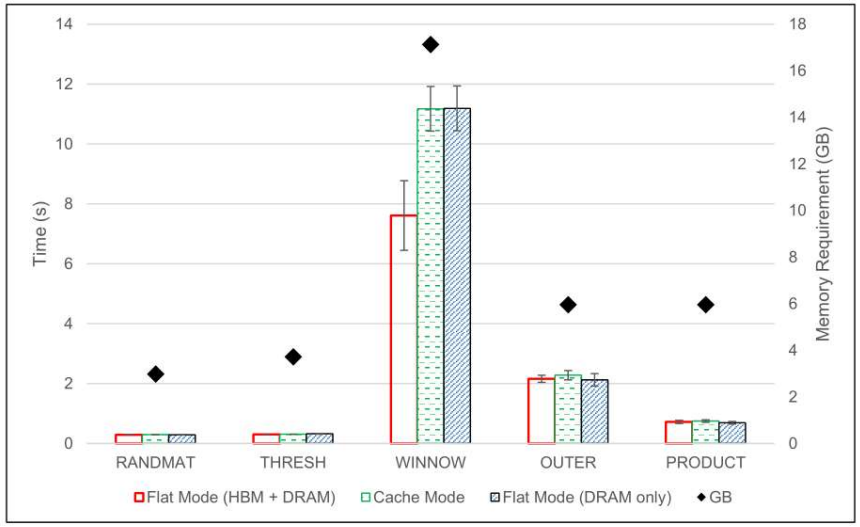
  - READ_WRITE I want a read and write copy (default)
  - READ_ONLY I want a copy for reading from only
  - WRITE_ONLY I want a copy for writing to only
  - EXCLUSIVE I want an element-wise exclusive ownership

First Results (Manually Managed)

- Five benchmarks from the Cowichan suite
  - Platform: Intel Xeon Phi Knights Landing (KNL)
  - Manuall data replication in HBM can give great benefits



(a) $80k \times 80k$ matrix, 4 nodes



(b) $80k \times 80k$ matrix, 8 nodes

Source: Kowalewsi R., Fuchs T., Fürlinger K., Utilizing Heterogeneous Memory Hierarchies in the PGAS Model, to appear in *Proceedings of the 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2018)*

Collaborations (Inside and Outside of SPPEXA)

- Collaboration with the **MYX** and **ESSEX-II** projects: Three Workshops on "*Parallel Programming Models - Productivity and Applications*"
  - Tokio, April 6 2017 / Versailles, October 18 2017 / Aachen, March 15 2018
  - Plans to use **DART** as a use case for the **MUST** correctness checking tool

- BMBF Project **MEPHISTO** (02/2017 – 01/2020)
  - TU Dresden (lead) + HZDR + LMU Munich
  - Integration of ALPAKA (portable compute kernel abstraction) with DASH (data structure abstraction)

Acknowledgements

- Funding

DFG Deutsche Forschungsgemeinschaft

Bundesministerium für Bildung und Forschung

- The DASH Team

  T. Fuchs (LMU), R. Kowalewski (LMU), D. Hünich (TUD), A. Knüpfer (TUD), J. Gracia (HLRS), C. Glass (HLRS), J. Schuchart (HLRS), F. Mößbauer (LMU), K. Fürlinger (LMU)

- DASH is on GitHub
  - https://github.com/dash-project/dash/

Fork me on GitHub

dash

The DASH Coarray

CAF	DASH Native	DASH Compatibility
image	*unit*	
this_image()	dash::myid()	dash::coarray::this_image()
num_images()	dash::size()	dash::coarray::num_images()
sync all	dash::barrier()	dash::coarray::sync_all()
sync images		dash::coarray::sync_images()