



DASH Status Update

SPPEXA Annual Plenary Meeting 2017

Garching, March 20-22, 2017



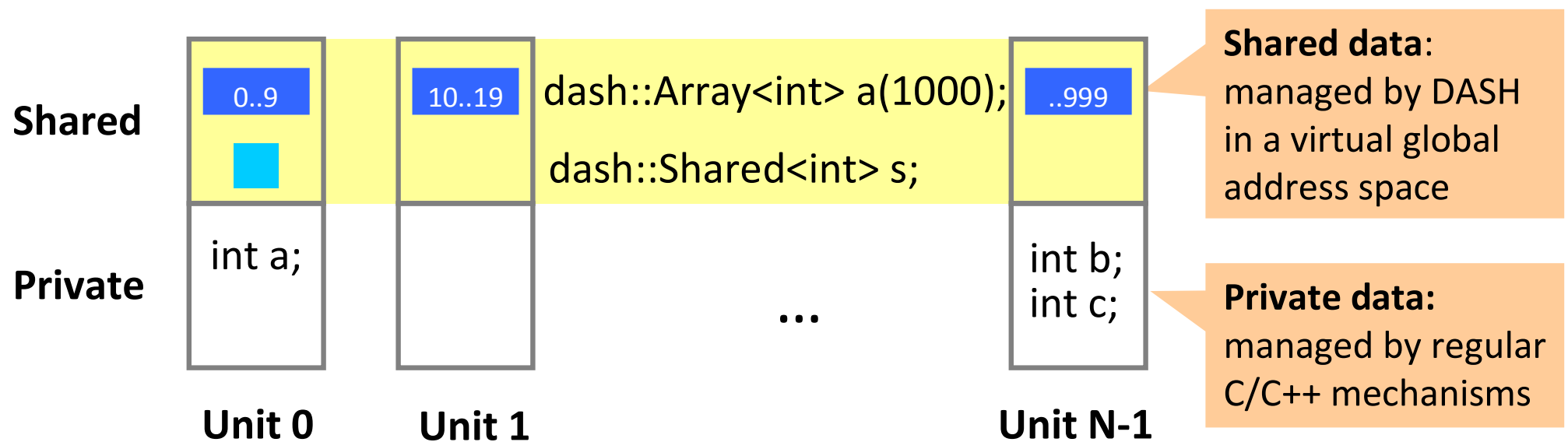
www.dash-project.org

Karl Förlinger

Ludwig-Maximilians-Universität München

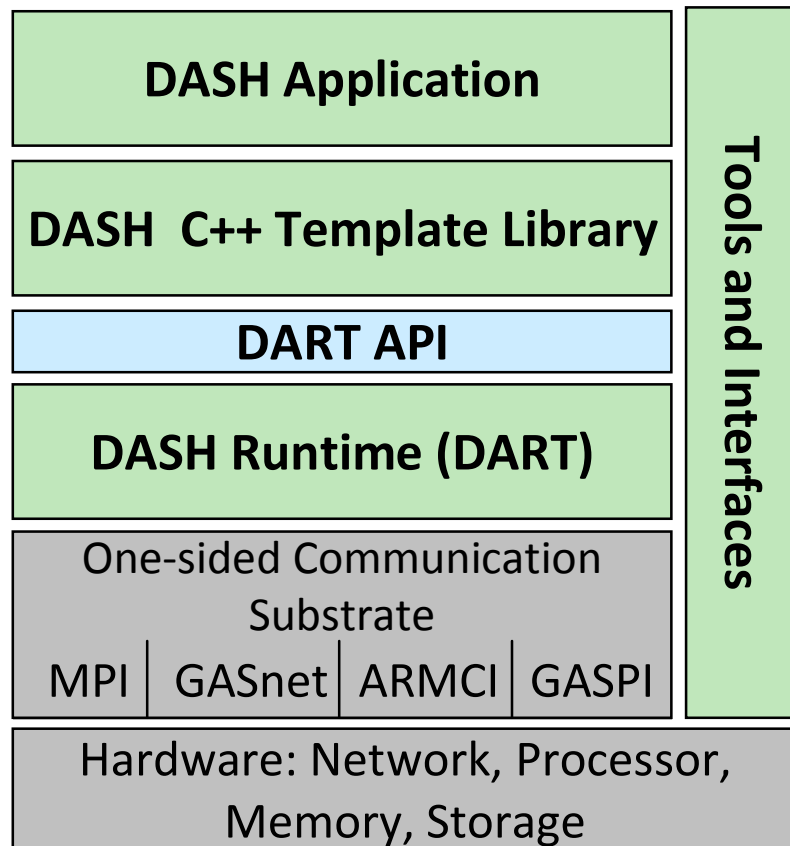


- DASH is a C++ template library that offers
 1. Distributed data structures and parallel algorithms that can be integrated in existing applications
 2. A complete PGAS (part. global address space) programming system without requiring a custom compiler



Units: 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++ template library, DASH data dock
TU Dresden	Libraries and interfaces, tools support	Smart data structures, resilience
HLRS Stuttgart	DASH runtime (DART)	DASH runtime (DART)
KIT Karlsruhe	Application case studies	
IHR Stuttgart		Smart deployment, Application case studies



www.dash-project.org



■ Features:

- **DASH multi-dimensional array (dash::NArray) and data distribution patterns [1],[6]**
- **Application engagement: LULESH, others [2],[5],[7]**
- Support for performance tools and debuggers with DASH [3]
- Correctness tool: Nasty-MPI [4]
- DART progress engine [8]

■ Project management and outreach

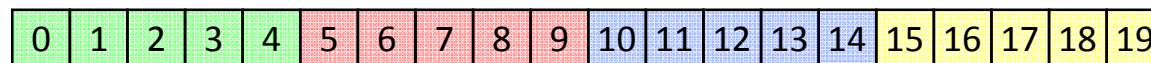
- **Release of DASH v0.2.0 and move to GitHub**
- **DASH Tutorial @HiPEAC in Stockholm**
- DASH presentations at LLNL, LBNL, ORNL, TU Wien, KTH Stockholm
- DASH v0.3.0 release imminent

■ The data distribution pattern is configurable

```
dash::Array<int> arr1(20); // default: BLOCKED  
  
dash::Array<int> arr2(20, dash::BLOCKED)  
dash::Array<int> arr3(20, dash::CYCLIC)  
dash::Array<int> arr4(20, dash::BLOCKCYCLIC(3))
```

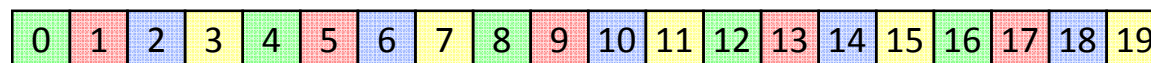
■ Assume 4 units

arr1, arr2



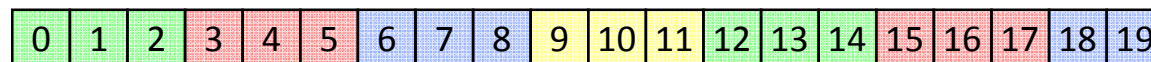
BLOCKED

arr3



CYCLIC

arr4

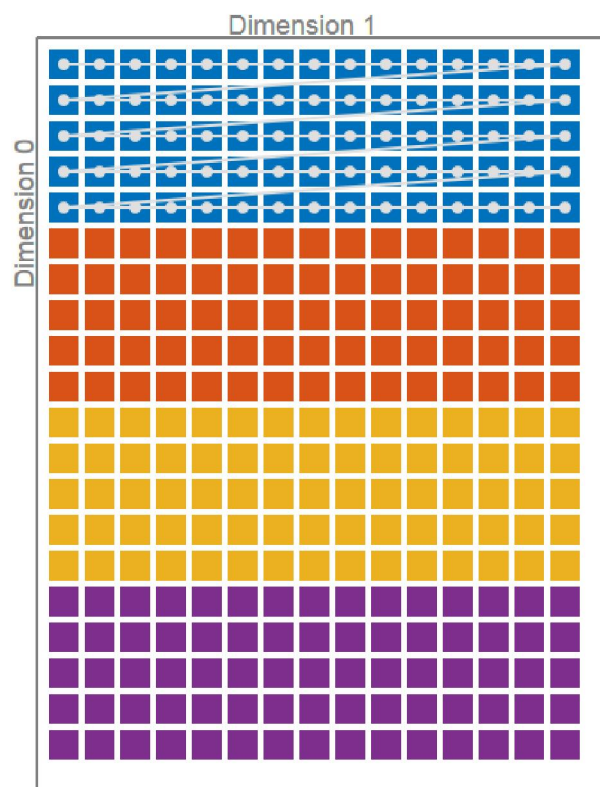


BLOCKCYCLIC(3)

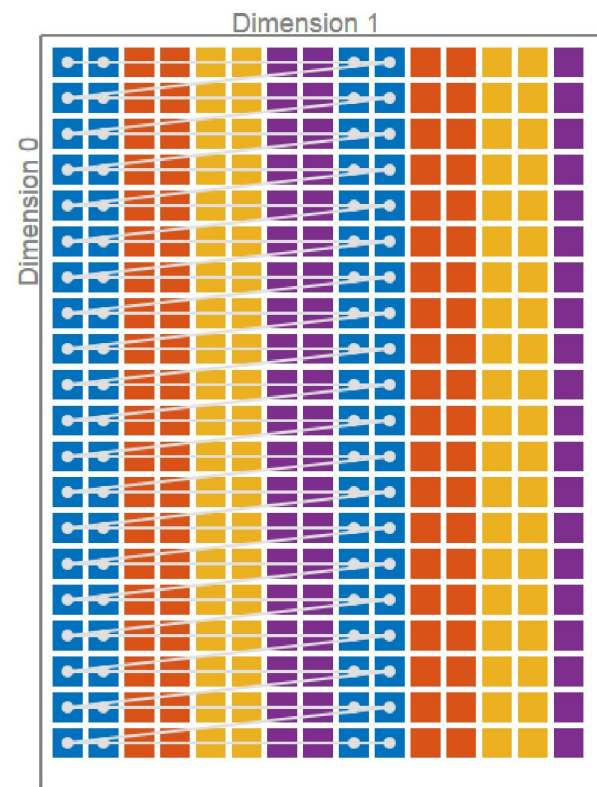
Multi-dimensional Data Distribution (1)

- `dash::Pattern<N>` specifies N-dim data distribution
 - Blocked, cyclic, and block-cyclic in multiple dimensions

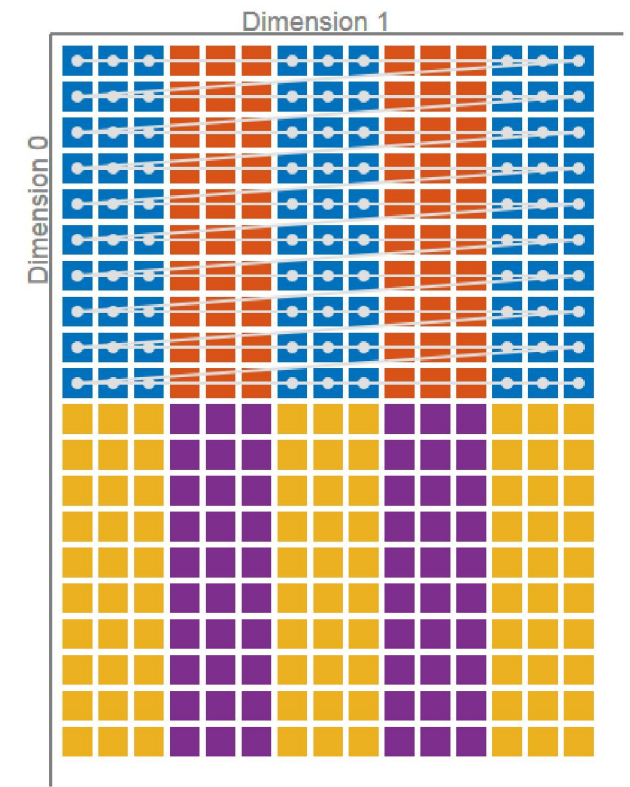
`Pattern<2>(20, 15)`



(BLOCKED,
NONE)



(NONE,
BLOCKCYCLIC(2))



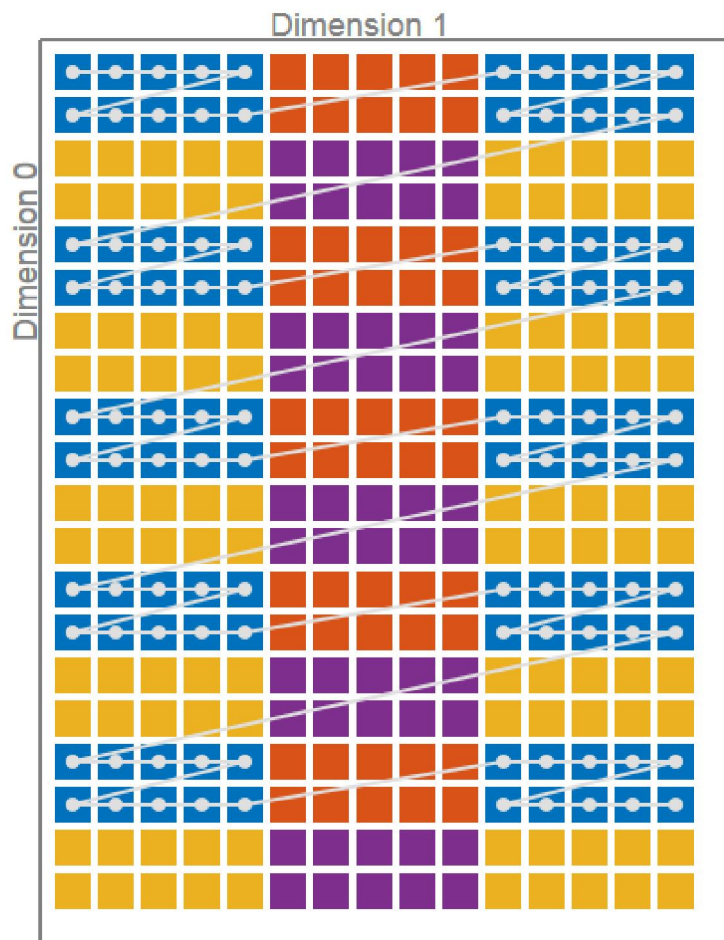
(BLOCKED,
BLOCKCYCLIC(3))



Multi-dimensional Data Distribution (2)

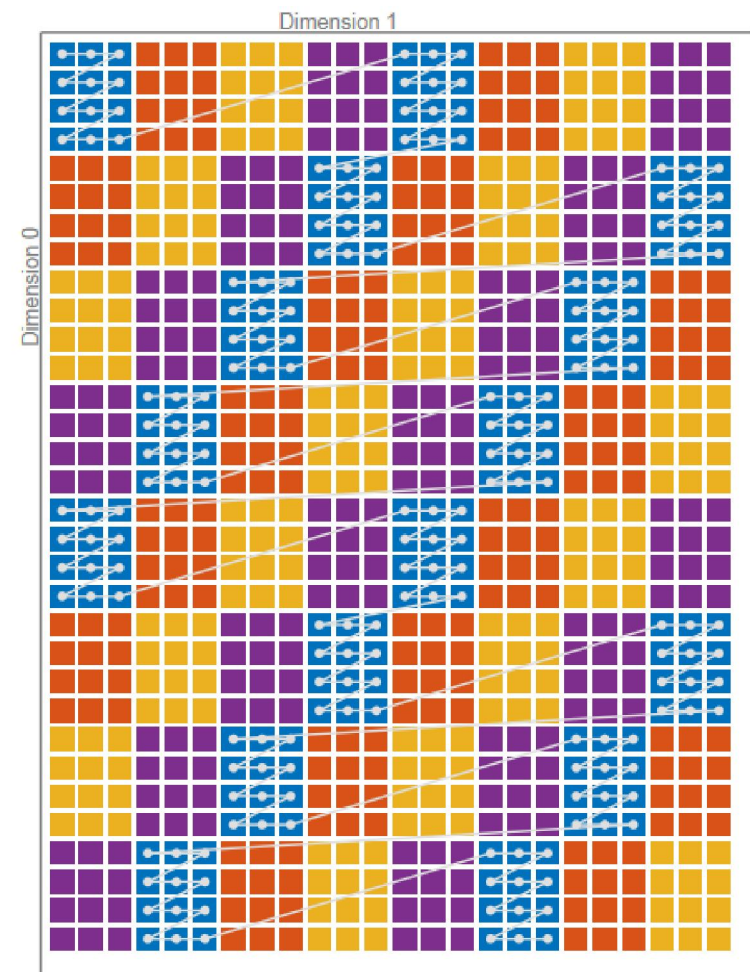
■ Tiled data distribution and tile-shifted distribution

TilePattern<2>(20, 15)



(TILE(2), TILE(5))

ShiftTilePattern<2>(32, 24)



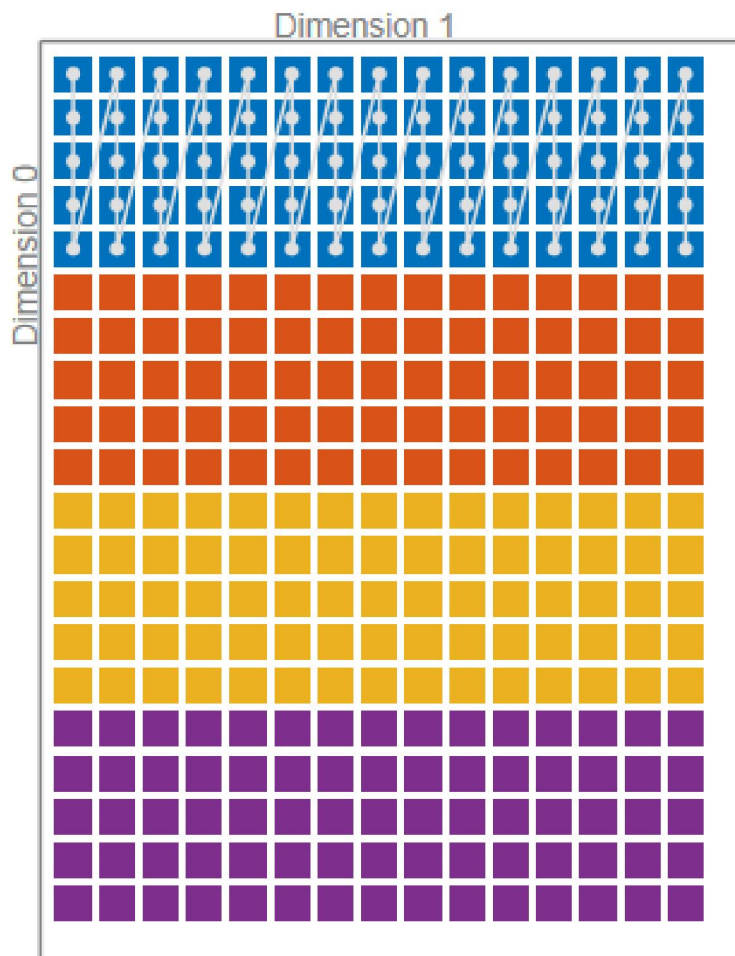
(TILE(4), TILE(3))



Multi-dimensional Data Distribution (3)

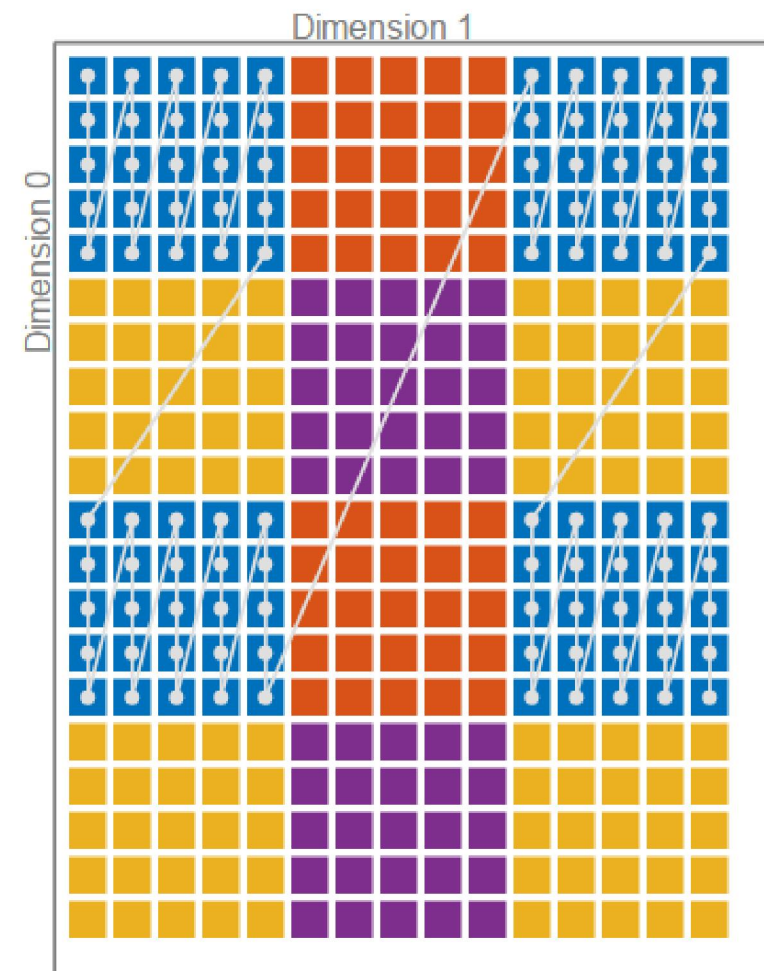
■ Row-major and column-major storage

Pattern<2, COL_MAJOR>(20, 15)



(BLOCKED, NONE)

TilePattern<2, COL_MAJOR>(20, 15)



(TILE(5), TILE(5))

Unit 0
Unit 1
Unit 2
Unit 3

The Multi-Dimensional Array

- `dash::NArray` (`dash::Matrix`) offers a distributed multi-dimensional array abstraction
 - Dimension is a template parameter
 - Element access using coordinates or linear index
 - Support for custom index types
 - Support for row-major and column-major storage

```
dash::NArray<int, 2> mat(40, 30); // 1200 elements

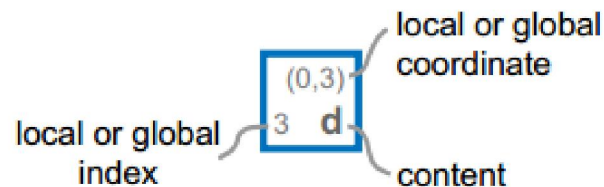
int a = mat(i,j);    // Fortran style access
int b = mat[i][j];   // chained subscripts

auto loc = mat.local;

int c = mat.local[i][j];
int d = *(mat.local.begin()); // local iterator
```

DASH NArray Global View and Local View

Local view works similar to 1D array



Global View

(0,0) 0 a	(0,1) 1 b	(0,2) 2 c	(0,3) 3 d
(1,0) 4 e	(1,1) 5 f	(1,2) 6 g	(1,3) 7 h
(2,0) 8 i	(2,1) 9 j	(2,2) 10 k	(2,3) 11 l
(3,0) 12 m	(3,1) 13 n	(3,2) 14 o	(3,3) 15 p
(4,0) 16 q	(4,1) 17 r	(4,2) 18 s	(4,3) 19 t
(5,0) 20 u	(5,1) 21 v	(5,2) 22 w	(5,3) 23 x
(6,0) 24 y	(6,1) 25 z	(6,2) 26 A	(6,3) 27 B

```
dash::NArray<char, 2> mat(7, 4);
cout << mat(2, 1) << endl; // prints 'j'

if(dash::myid()==0 ) {
    cout << mat.local(2, 1) << endl; // prints 'z'
}
```

Local View (Unit 0)

(0,0) 0 a	(0,1) 1 b	(0,2) 2 c	(0,3) 3 d
(1,0) 4 m	(1,1) 5 n	(1,2) 6 o	(1,3) 7 p
(2,0) 8 y	(2,1) 9 z	(2,2) 10 A	(2,3) 11 B

Local View (Unit 1)

(0,0) 0 e	(0,1) 1 f	(0,2) 2 g	(0,3) 3 h
(1,0) 4 q	(1,1) 5 r	(1,2) 6 s	(1,3) 7 t

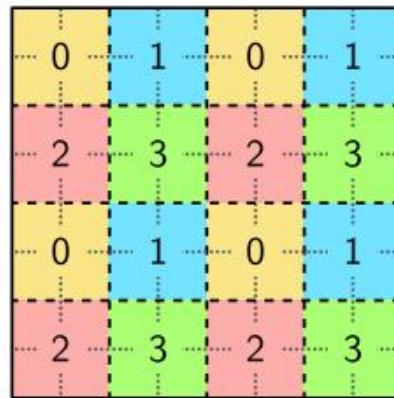
Local View (Unit 2)

(0,0) 0 i	(0,1) 1 j	(0,2) 2 k	(0,3) 3 l
(1,0) 4 u	(1,1) 5 v	(1,2) 6 w	(1,3) 7 x

Application Example: S(R)UMMA Algorithm

■ Block matrix-matrix multiplication algorithm with block prefetching

Decomposition



Prefetching

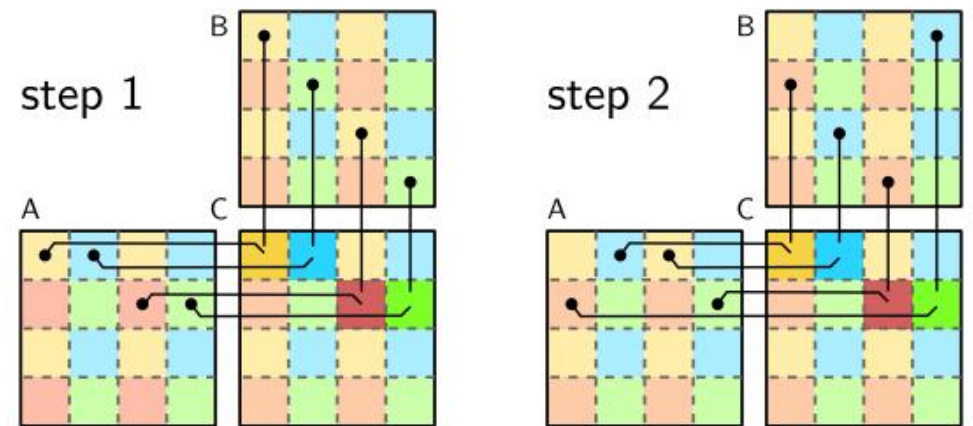


Image source: [1]

```
while(!done) {
    blk_a = ...
    blk_b = ...
    // prefetch
    auto get_a = dash::copy_async(blk_a.begin(), blk_a.end(), lblk_a_get);
    auto get_b = dash::copy_async(blk_b.begin(), blk_b.end(), lblk_b_get);
    // local DGEMM
    dash::multiply(lblk_a_comp, lblk_b_comp, lblk_c_comp);
    // wait for transfer to finish
    get_a.wait(); get_b.wait();
    // swap buffers
    swap(lblk_a_get, lblk_a_comp); swap(lblk_b_get, lblk_b_comp);
}
```


DGEMM on a Single Shared Memory Node

- LRZ SuperMUC, phase 2: Haswell EP, 1.16 Tflop/sec peak

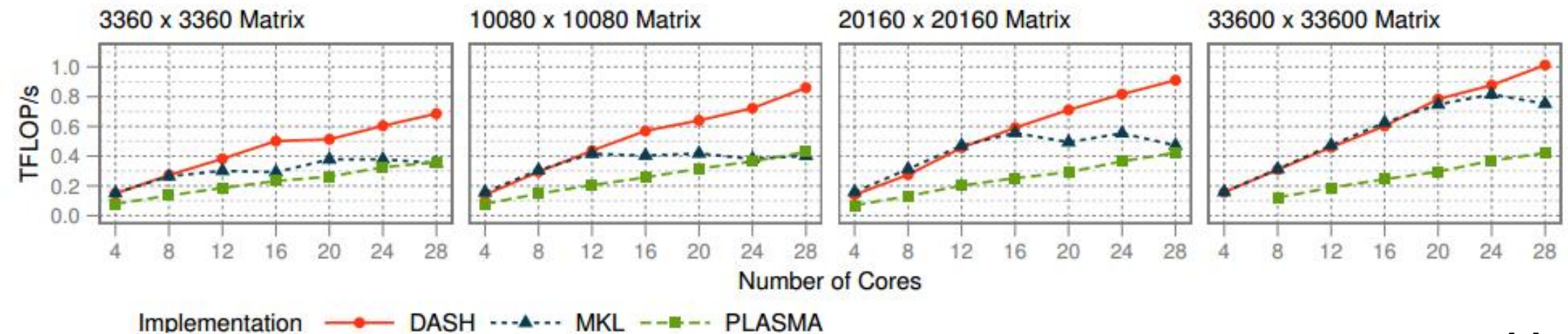


Image source: [1]

- DASH: Multi-process, using one-sided communication and single-threaded MKL
- MKL: Multithreaded, using OpenMP
- PLASMA: Multithreaded tile-algorithms on top of sequential BLAS

PDGEMM: DASH vs. ScaLAPACK Multinode

- Strong scaling on SuperMUC (57344 × 57344 matrix)

2. IBM MPI

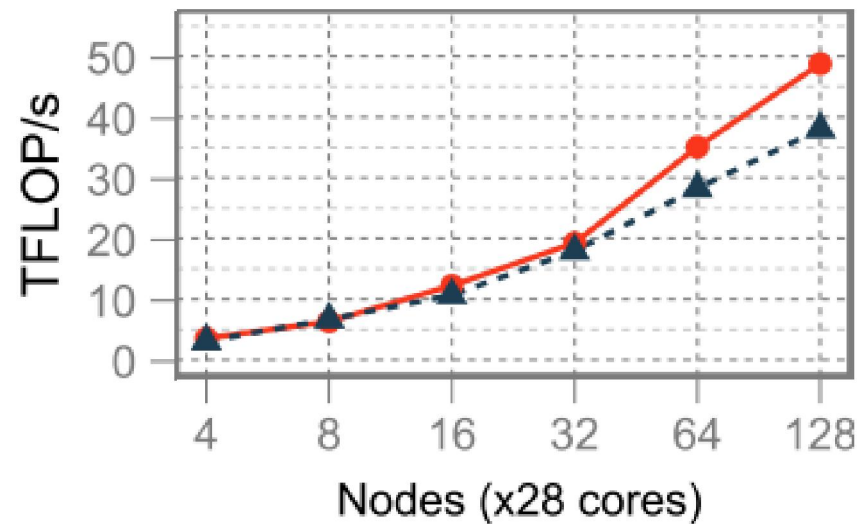


Image source: [1]

Implementation —●— DASH ---▲--- ScaLAPACK

- Trace: Overlapping communication and computation

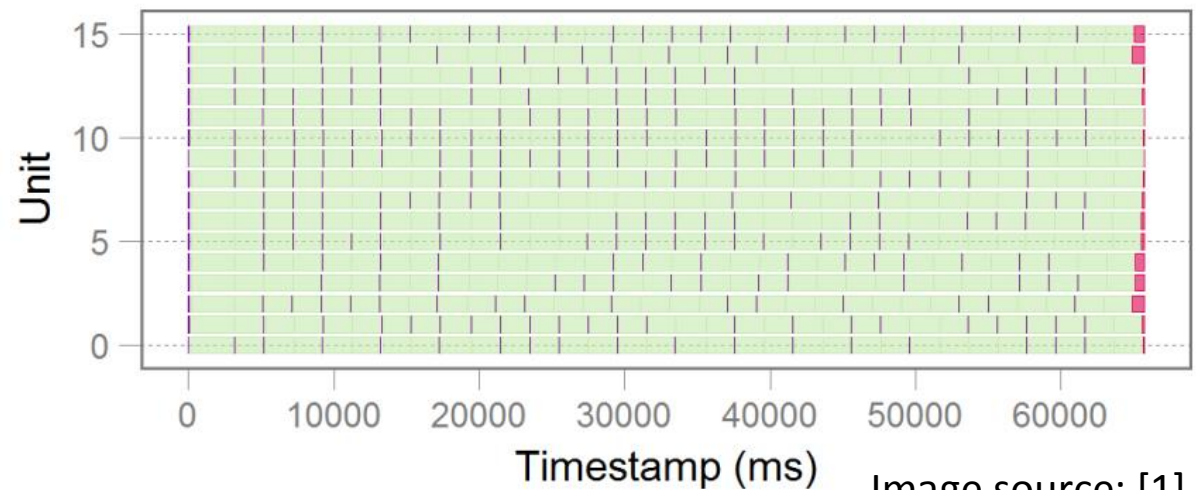


Image source: [1]

Process state ■ barrier ■ multiply ■ prefetch

- LULESH (Livermore **U**nstructured **L**agrangian **E**xplicit **S**hock **H**ydrodynamics code)
 - Widely used mini-app/proxy application
- Goals of the DASH port
 - Remove limitations of MPI domain decomposition approach
 - Avoid replication, manual index calculation, bookkeeping

```
if (rowMin | rowMax) {
  /* ASSUMING ONE DOMAIN PER RANK, CONSTANT BLOCK SIZE HERE */
  int sendCount = dx * dz ;

  if (rowMin) {
    destAddr = &domain.commDataSend[pmsg * maxPlaneComm] ;
    for (Index_t fi=0; fi<xferFields; ++fi) {
      Domain_member src = fieldData[fi] ;
      for (Index_t i=0; i<dz; ++i) {
        for (Index_t j=0; j<dx; ++j) {
          destAddr[i*dx+j] = (domain.*src)(i*dx*dy + j) ;
        }
      }
      destAddr += sendCount ;
    }
    destAddr -= xferFields*sendCount ;

    MPI_Isend(destAddr, xferFields*sendCount, baseType,
              myRank - domain.tp(), msgType,
              MPI_COMM_WORLD, &domain.sendRequest[pmsg])
    ++pmsg ;
  }
}
```

implicit assumptions

manual index calculation

manual msg bookkeeping

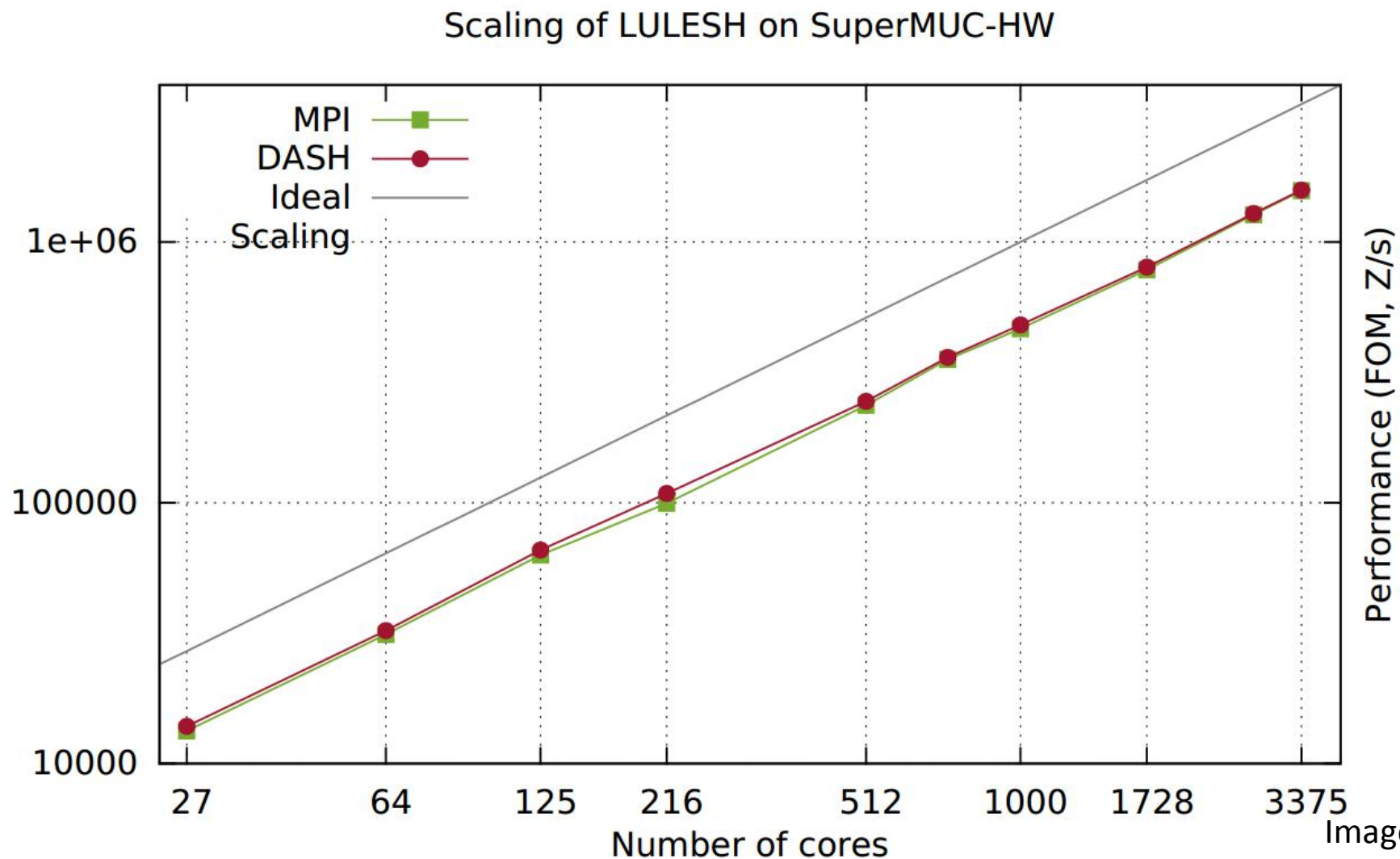
Replicated about 80 times

Porting LULESH (options)

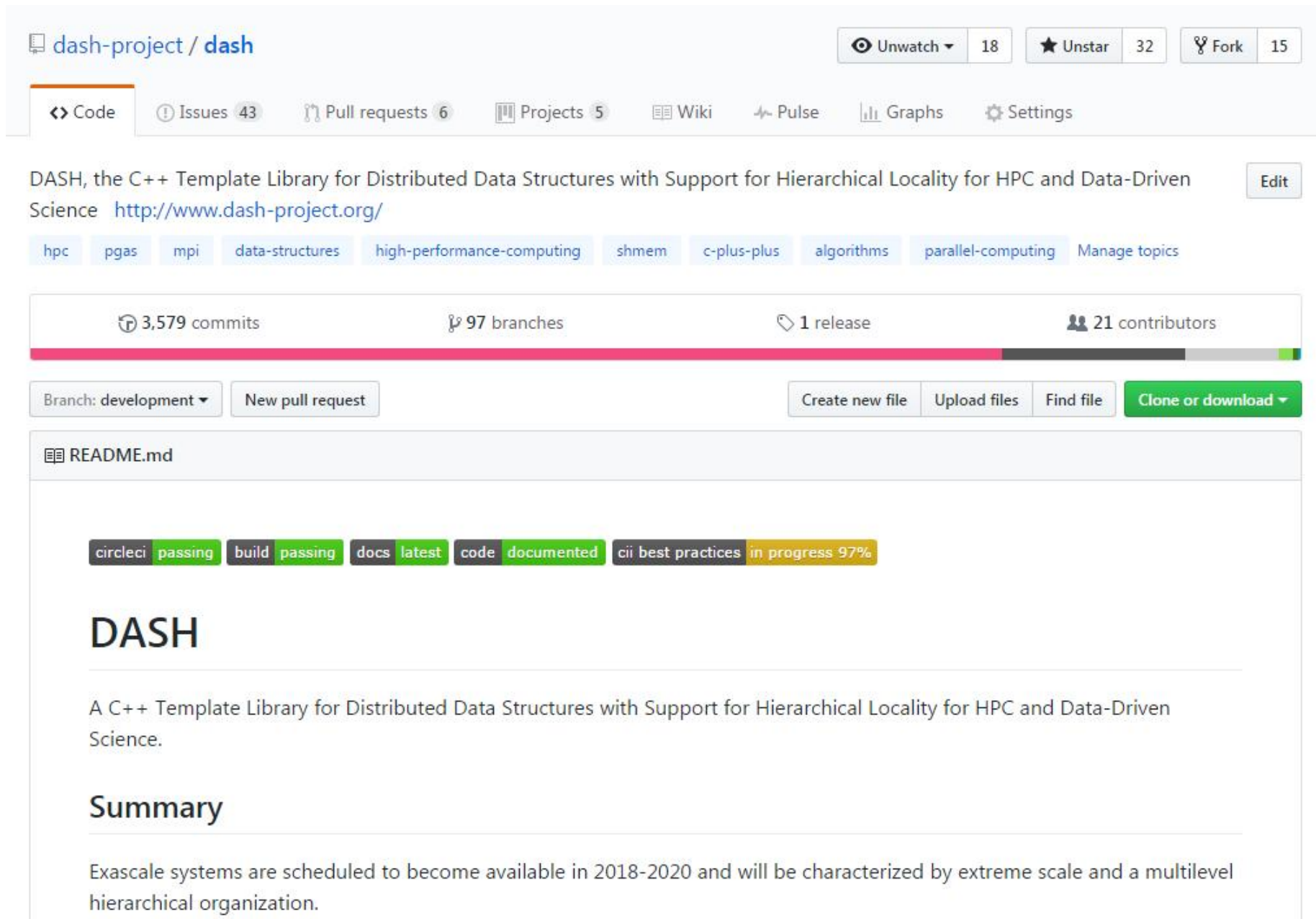
- DASH can be integrated in existing applications and allows for incremental porting
- Porting options:
 1. Port data structures, but keep communication as-is (using MPI two-sided)
 - Can use HDF5 writer for checkpointing
 2. Keep explicit packing code but use one-sided put instead of MPI_Irecv/MPI_Isend
 - Potential performance benefit from one-sided communication
 3. Use DASH for communication directly
 - `auto halo = ...; dash::swap(halo) ...`
 - less replicated code, more flexibility
 4. Use `dash::HaloNArray`
 - Multi-dimensional array with built-in halo areas

Performance of the DASH Version (using put)

- Performance and scalability (weak scaling) of LULESH, implemented in MPI and DASH



- <https://github.com/dash-project/dash>



dash-project / dash

Unwatch 18 Unstar 32 Fork 15

Code Issues 43 Pull requests 6 Projects 5 Wiki Pulse Graphs Settings

DASH, the C++ Template Library for Distributed Data Structures with Support for Hierarchical Locality for HPC and Data-Driven Science <http://www.dash-project.org/> Edit

hpc pgas mpi data-structures high-performance-computing shmem c-plus-plus algorithms parallel-computing Manage topics

3,579 commits 97 branches 1 release 21 contributors

Branch: development New pull request Create new file Upload files Find file Clone or download

README.md

circleci passing build passing docs latest code documented cii best practices in progress 97%

DASH

A C++ Template Library for Distributed Data Structures with Support for Hierarchical Locality for HPC and Data-Driven Science.

Summary

Exascale systems are scheduled to become available in 2018-2020 and will be characterized by extreme scale and a multilevel hierarchical organization.

- Available at <http://www.dash-project.org/tutorial/tutorial.html>



#HiPEAC17

January 23-25, 2017, Stockholm, Sweden

[Programme](#)[Paper track](#)[Keynotes](#)[Venue and travel](#)[Recruitment](#)[3 keynotes >](#)[25 workshops >](#)[11 tutorials >](#)[34 papers >](#)

- Memory Spaces
 - Support for NVRAM, high-bandwidth memory, replica mgmt
- Task-Based Execution
 - Compute kernels and dependency mgmt
- Dynamic Data Structures (growing/shrinking)
 - List and hashmap as first goals
- Hierarchical Locality Information System
 - Make working with locality easier
- Halo Matrix Smart Data Structure
 - NArray with built-in support for halo exchange

■ DASH is

- A complete data-oriented PGAS programming system (i.e., entire applications can be written in DASH),
- A library that provides distributed data structures (i.e., DASH can be integrated into existing MPI applications)

■ The DASH Team

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

■ More information

- <http://www.dash-project.org/>
- <https://github.com/dash-project/dash/>



References (1)

- [1]: Tobias Fuchs and Karl Furlinger. **A Multi-Dimensional Distributed Array Abstraction for PGAS**. In *Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016)*, pages 1061-1068. Sydney, Australia, December 2016.
- [2]: Karl Furlinger, Tobias Fuchs, and Roger Kowalewski. **DASH: A C++ PGAS Library for Distributed Data Structures and Parallel Algorithms**. In *Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016)*, pages 983-990. Sydney, Australia, December 2016.
- [3]: Denis Hünich, Andreas Knüpfer, Sebastian Oeste, Karl Furlinger, and Tobias Fuchs. **Tool Support for Developing DASH Applications**. In Hans-Joachim Bungartz, Philipp Neumann, and E. Wolfgang Nagel, editors, *Software for Exascale Computing - SPPEXA 2013-2015*, pages 361-377. Springer. Garching, Germany, 2016.
- [4]: Roger Kowalewski and Karl Furlinger. **Nasty-MPI: Debugging Synchronization Errors in MPI-3 One-Sided Applications**. In *Proceedings of the 22nd International Conference on Parallel and Distributed Computing (Euro-Par 2016)*, pages 51-62. Grenoble, France, 2016.
- [5]: Huan Zhou and José Gracia. **Towards performance portability through locality-awareness for applications using one-sided communication primitives**. In *Forth International Symposium on Computing and Networking, CANDAR Hiroshima, Japan, November 22-25, 2016*. IEEE Computer Society, 2016.

References (2)

- [6]: Tobias Fuchs and Karl Furlinger. **Expressing and Exploiting Multi-Dimensional Locality in DASH**. In Hans-Joachim Bungartz, Philipp Neumann, and E. Wolfgang Nagel, editors, *Software for Exascale Computing - SPPEXA 2013-2015*, pages 341-359. Springer. Garching, Germany, 2016.
- [7]: Kamran Idrees, Tobias Fuchs, and Colin W Glass. **Effective use of the PGAS Paradigm: Driving Transformations and Self-Adaptive Behavior in DASH-Applications**. In *Proceedings of the First International Workshop on Program Transformation for Programmability in Heterogeneous Architectures, held in conjunction with the CGO Conference*.
- [8]: Huan Zhou and José Gracia. **Asynchronous progress design for a MPI-based PGAS one-sided communication system**. In *22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016*, 2016.