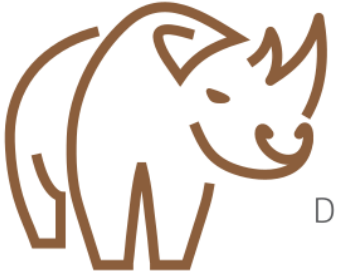
TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



dash

www.dash-project.org

Distributed Data Structures and Parallel Algorithms

HiPEAC 2018 Tutorial, Manchester, 2018-01-22

PARALLEL PROGRAMMING WITH DISTRIBUTED DATA STRUCTURES USING DASH

Andreas Knüpfer (TU Dresden), Tobias Fuchs (LMU Munich)

Overview & Goals

This tutorial contains:

- An introduction to the DASH library for large-scale parallel programing with native C++ look & feel
- A Hello-World example as your first practical steps
- A fill-in application example processing a huge astronomical image in parallel with less than 250 lines of code

Project website
http://www.dash-project.org

GitHub sites
https://github.com/dash-project/
--> dash and dash-apps

Presenters

Andreas: Mathematician and Computer Scientist at TU Dresden. Research in High Performance Computing, parallel programming models, and software tools for parallel performance analysis.

andreas.knuepfer@tu-dresden.de

Tobias: Computer Scientist at LMU Munich, Industry background in embedded systems and real-time applications. Research in High Performance Computing, domain space mappings and dynamic hardware locality.

t.fuchs@mnm-team.org

Agenda

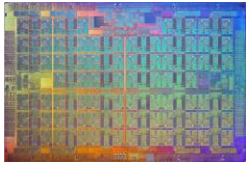
Tutorial 10:00 – 17:30

- Introduction to Parallel and High-Performance Computing, PGAS, DASH
- Tutorial VM image, Hello World example, hands-on exercises
- More about DASH: containers, algorithms, data distribution patterns
- Hands-on with astronomical image processing steps 1 to 5
- DASH halos + astronomical image processing improved with halos
- Preview on multidimensional array views
- Optional: More examples from DASH demo codes

Breaks according to the HiPEAC schedule

- 11:00 – 11:30 Coffee break
- 13:00 – 14:00 Lunch break
- 15:30 – 16:00 Coffee break

The World of Parallel Computing in 2017



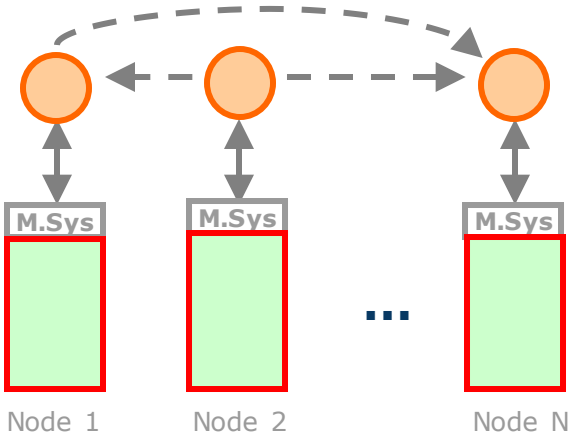
Supercomputer	**Cluster**	**KNL Manycore**	**Server**	**Notebook**	**Mobile**
100 000+ cores	1000s of cores	72 cores	10s of cores	2-4 cores	2-8 cores

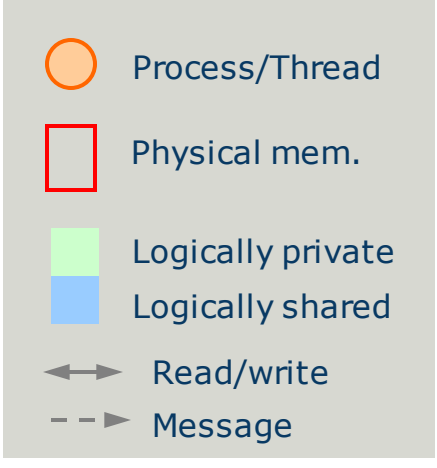
Distributed Memory (DM) ⟵ | ⟶ Shared Memory (SM)

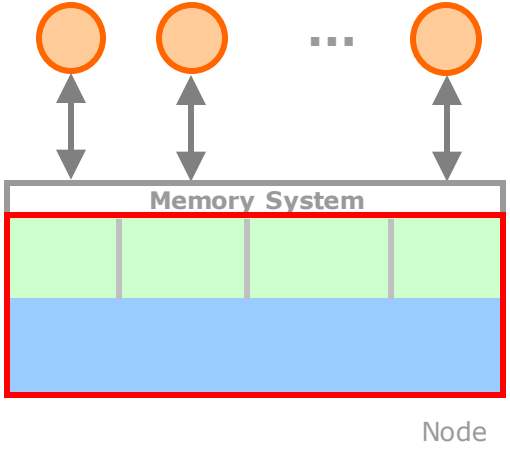
DM programming: MPI, Charm++, …

SM programming: OpenMP, Pthreads, Cilk, TBB, …

Shared Memory vs. Distributed Memory Programming







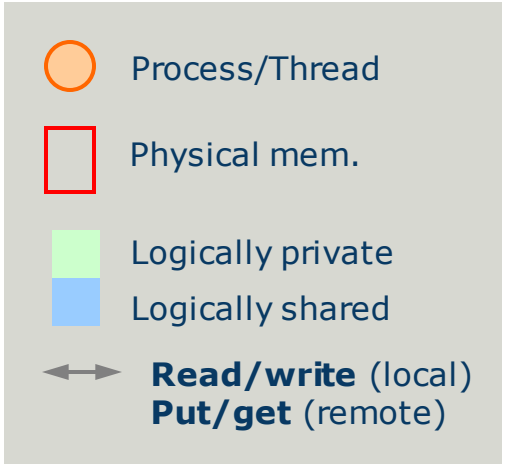
Message Passing

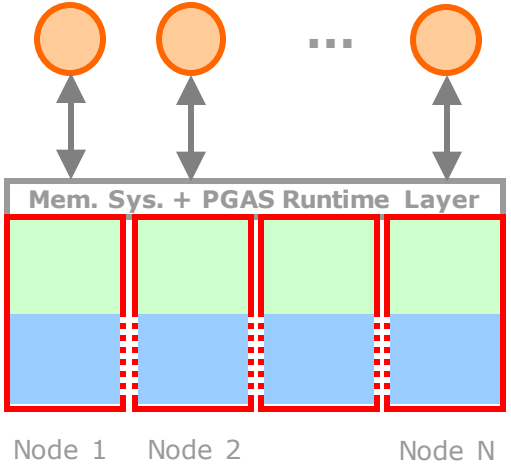
- Performance, runs everywhere
- Productivity

Threading

- Productivity
- Locality control, limited to SM hardware

PGAS – Combining the Advantages of both Approaches





PGAS *Languages*

- Chapel, CoArray Fortran, UPC, …

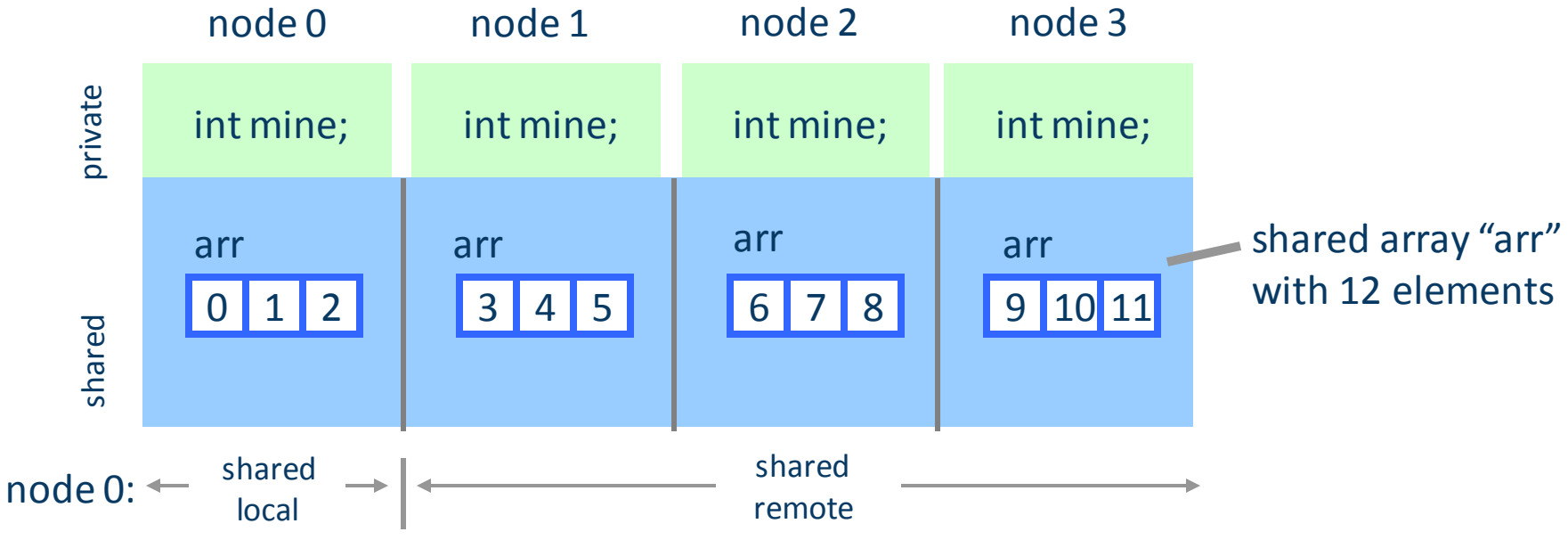
PGAS *Libraries*

- Global Arrays (GA), GASPI, OpenShmem, MPI3.0 RMA
- Used by the DASH runtime layer

Locality control, runs everywhere, performance and productivity

PGAS: Partitioned Global Address Space

PGAS Example with Four Nodes/Processes



PGAS Example in DASH

```
#include <libdash.h>

int main(int argc, char* argv[]) {

  dash::init( &argc, &argv );
  int mine;
  dash::Array<int> arr(12);
  if ( 0 == dash::myid() ) {
    arr.local[2]=2;
    arr[10]=42;
  }
  dash::finalize();
}
```

private

shared

shared local

shared remote

Compilation

```
$ mpicc -L... -ldash
  -o myapp myapp.cc
```

Execution

```
$ mpirun -n 4 ./myapp
```

- DASH is based on MPI, so we need to use mpicc and mpirun
- Note SPMD model (like MPI)

(sometimes one needs to use 'mpirun --oversubscribe')

So you'd like to write parallel codes in C++?

Supercomputing today

- Large scale parallelism
- Heterogeneous architectures
- Hybrid parallelism

--> many sources of complexity

- MPI+X as dominating parallel programming model
- Data distribution, data transfers, and synchronization entangled

MPI disregards C++

- In C++ MPI codes you actually use MPI's C API
- The MPI C++ bindings were deprecated in MPI 2.2 and removed in MPI 3.0*
- C++ concepts like STL containers, iterators, or even data types are incompatible with MPI!

*http://blogs.cisco.com/performance/the-mpi-c-bindings-what-happened-and-why

DASH Goals

Parallel Programming Model

- High-level abstraction for the PGAS (Partitioned Global Address Space) concept
- Keep native C++ look & feel, use C++ 11/14 functionality
- Suitable for large-scale distributed memory parallelism
- High programmer productivity
- Implemented as C++ library, no language extensions, no proprietary compilers needed

Multilevel locality

- Support data locality as the main factor for good performance
- Allow multi-level locality beyond just local vs. remote
- Still provide global access for convenience

The DASH Project and Project Funding

- DASH started in 2013 under the DFG Priority Programme 1648 "Software for Exascale Computing" (SPPEXA)
- Currently funded in SPPEXA phase 2
- Follow-up project "Mephisto" funded by BMBF Germany

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Tobias Fuchs, Karl Fürlinger, Roger Kowalewski, Felix Mößbauer

Colin W. Glass, José Gracia

HLRS

Kamran Idrees, Joseph Schuchart, Huan Zhou

DFG Deutsche Forschungsgemeinschaft

SPPEXA

Bundesministerium für Bildung und Forschung

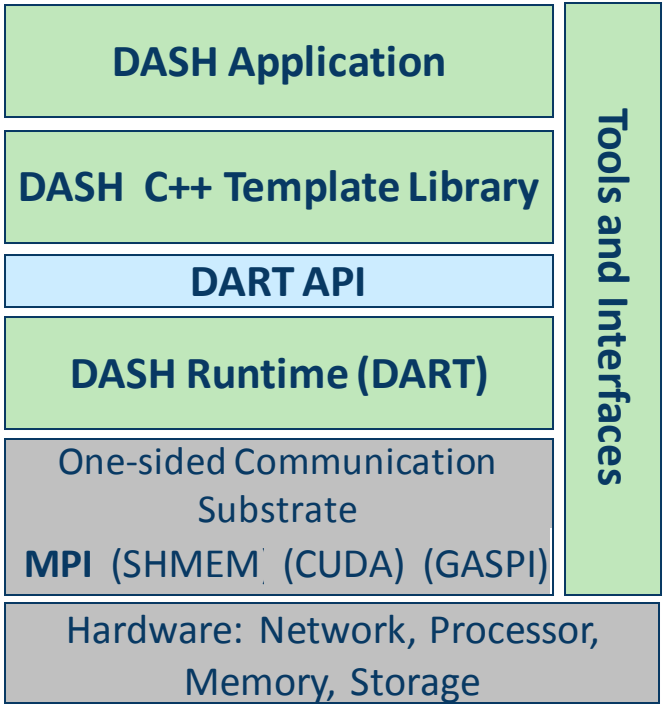
TECHNISCHE UNIVERSITÄT DRESDEN

Denis Hünich, Andreas Knüpfer, Bert Wesarg

Terminology and Library Structure

- **DASH**: The C++ template library for application programmers
- **DART**: The DASH Run Time Library for implementing DASH on different substrates.

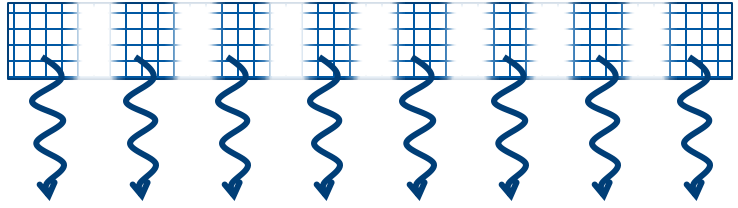
- **Unit**: A process with a separate address space. Usually there is one unit per CPU core.
- **Team**: A group of units. Teams are created in a hierarchy as sub-teams of existing teams.



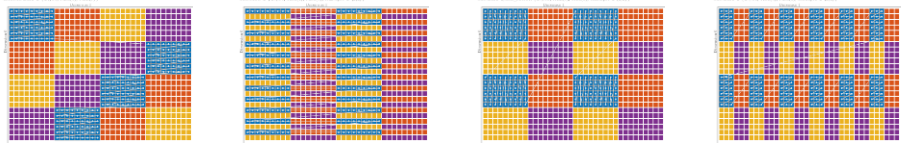
DASH Ingredients

The basic ingredients to DASH are

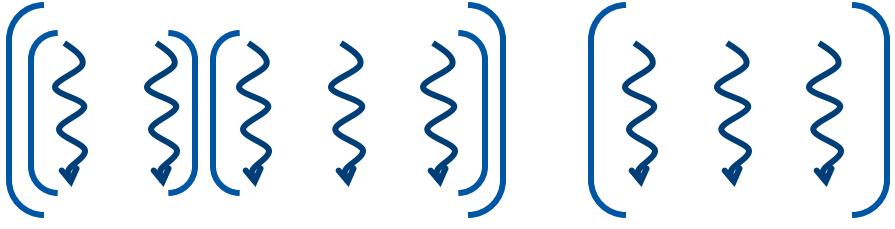
- **Units** (processes/ranks)
- Distributed data **containers**
- Parallel **algorithms** to work on the DASH containers



- **Data distribution patterns** to control data decomposition



- DASH **team hierarchies** to control multi-level decomposition of data and functionality



Examples

Containers

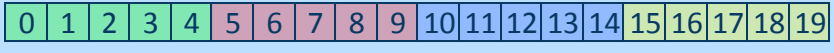
```
dash::Array<double> arr(100); // fixed size 1D array
dash::NArray<int, 2> mat(20,30); // 20x30 matrix
```

Algorithms

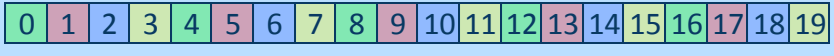
```
dash::fill(arr.begin(), arr.end(), 42);
auto min = dash::min_element(mat.begin(), mat.end());
```

Data Distribution Patterns

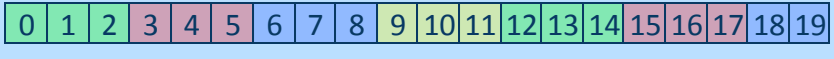
```
dash::Array<int> arr2(20, dash::BLOCKED)
```




```
dash::Array<int> arr3(20, dash::CYCLIC)
```



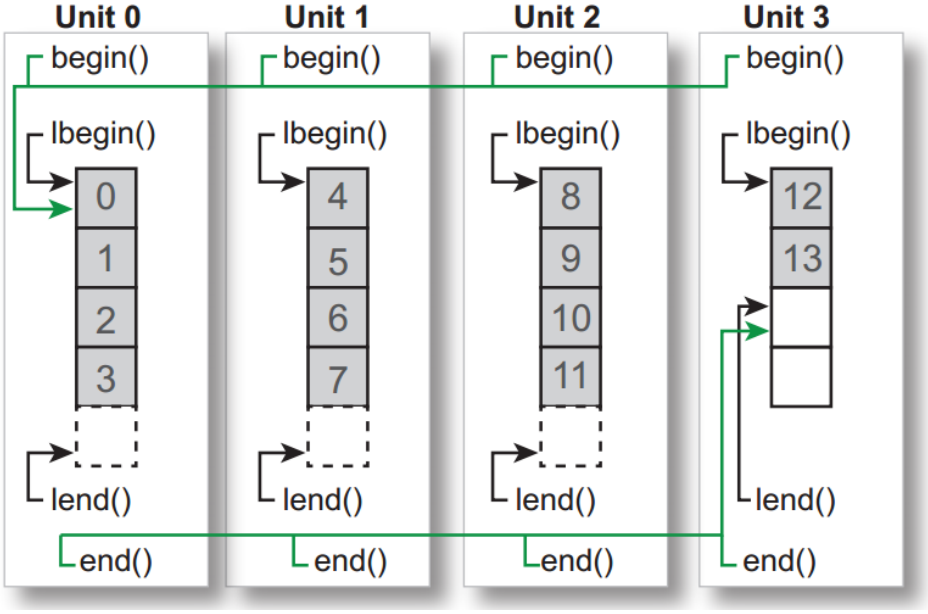

```
dash::Array<int> arr4(20, dash::BLOCKCYCLIC(3))
```



Global vs. Local Access

DASH containers provide global and local views

- Global access for convenience
- Local access for performance



	Global-view	Local-view (LV)	LV shorthand
range begin	`arr.begin()`	`arr.local.begin()`	`arr.lbegin()`
range end	`arr.end()`	`arr.local.end()`	`arr.lend()`
# elements	`arr.size()`	`arr.local.size()`	`arr.lsize()`
element access	`arr[glob_idx]`	`arr.local[loc_idx]`	

TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

dash

www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

THE TUTORIAL VM IMAGE

How to use the tutorial VM image

- Install Virtual Box on your laptop from https://www.virtualbox.org/
- Import VM or
- Use VDI image
  - Create a new VM (Ubuntu 64 bit)
  - all physical CPU cores, 4 recommended
  - Virtual memory < host memory, recommend 6000 MB
  - VDI image file directly from USB drive or a copy on local disk
  - Bidirectional Clipboard and Drag'n'Drop
  - A shared directory if you like
- Boot, log in with dash:dash

- Alternatively install directly
  - git clone https://github.com/dash-project/dash.git
  - Install from distribution packages: libopenmpi-dev, libtiff-dev, libsdl1.2-dev
  - cd dash.git; ./build.mininmal.sh; cd build; make; make install

Please ask instructors; do not spend too long to fix if not working

Fix the VM's keyboard layout

- We tried to think of every little detail but unfortunately kept the German keyboard layout as default. Sorry!

- Please fix like this:
  - Applications menu (top left) -->
  - Settings Manager -->
  - Keyboard --> Layout tab:

  - Unmark "Use systems defaults"
  - Add your preferred layout and move it to the top
  - Log out & log in again

TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

dash

www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

A HELLO WORLD EXAMPLE CODE

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

int main( int argc, char* argv[] ) {
  pid_t pid; char buf[100];

  dash::init( &argc, &argv );
  gethostname(buf, 100); pid = getpid();

  cout<<"'Hello world' from unit "<<
    dash::myid()<<" of "<<dash::size()<<
    " on "<<buf<<" pid="<<pid<<endl;

  dash::finalize();
}
```

Initialize the programming environment

Determine local unit ID and total number of units

Message printed in SPMD mode

```
$ mpirun [--oversubscribe] -n 4 ./hello
'Hello world' from unit 0 of 4 on tardis pid=25785
'Hello world' from unit 1 of 4 on tardis pid=25786
'Hello world' from unit 2 of 4 on tardis pid=25787
'Hello world' from unit 3 of 4 on tardis pid=25788
```

```
…

dash::Array<int> arr(100);

if ( 1 == dash::myid() ) {
  for ( auto i= 0; i < arr.size(); i++ )
    arr[i]= i;
}

dash::barrier();

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

…
```

DASH global array of 100 integers, distributed over all units, default distribution is BLOCKED

Unit 0 writes to the array using the global index i and the [] operator.

Global barrier

Unit 1 executes a global range-based for loop over the DASH array

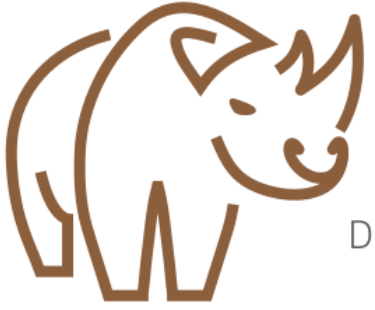
```
$ mpirun -n 4 ./hello_array
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99
```

```
…
dash::Array<int> arr(100);
…

for ( auto it= arr.lbegin(); it != arr.lend(); it++ ) {
  *it= dash::myid();
}

arr.barrier();

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

…
```

Use local iterator via lbegin()

Note container-related barrier instead of global barrier

```
$ mpirun -n 4 ./hello_array
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
www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

FIRST HANDS-ON EXAMPLE

Hands-on assignment 01

- Go to to 01-hello_world subdir and check that everything works

  > make hello_array && mpirun -n 4 ./hello_array

```
$ make && mpirun -n 4 ./hello_array
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
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 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
```

Have a look at the source code with your favorite editor

Hands-on assignment 01 contd.

Assignment 1) Enlarge the distributed array such that every unit has the same number of entries regardless of the number of units. Use values that result in a search time of few seconds for the min_element() calls below for the slower case.

```
dash::Array<int> arr( /* fill me */ );
```

Assignment 2) Replace the following loop with a call to a STL or DASH template algorithm fill that performs the same thing.

```
for ( auto it= arr.lbegin(); it != arr.lend(); it++ ) {
    *it= dash::myid();
}
```

Hands-on assignment 01 contd.

Assignment 3) Call the STL std::min_element() with arr's global iterator on unit 0 just to show that one can.

Assignment 4) Call the dash::min_element() counterpart as a collective operation on all units.

Assignment 5) Run with various numbers of units and watch how the runtimes behave.

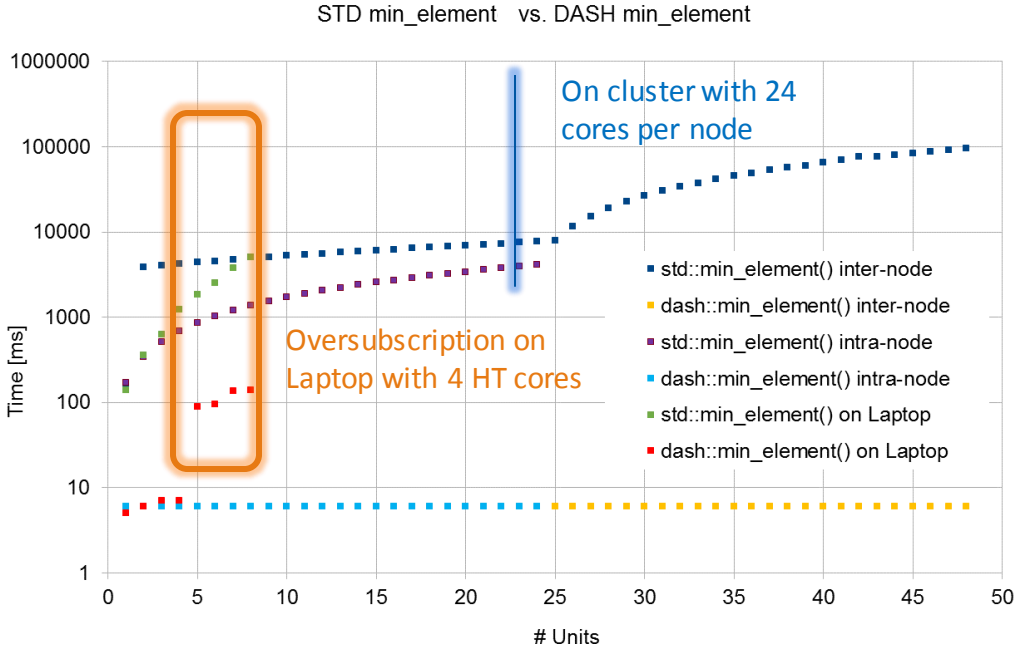
Solution 01

```
…
dash::Array<int> arr( 1000000 * dash::size() );
…
std::fill( arr.local.begin(), arr.local.end(), (int) dash::myid() );
std::fill( arr.lbegin(), arr.lend(), (int) dash::myid() );
dash::fill( arr.begin(), arr.end(), (int) dash::myid() );
…
if ( 0 == dash::myid() ) {

    std::min_element( arr.begin(), arr.end() );
}
…
dash::min_element( arr.begin(), arr.end() );
```

Solution 01

The runtimes of std::min_element() and dash::min_element() differ dramatically!

- The former uses one RDMA access per int!
- The latter uses only local accesses, then reduces the parallel sub-results.
- This is not saying that the STL algorithms are badly implemented but a reminder to be careful with global accesses.



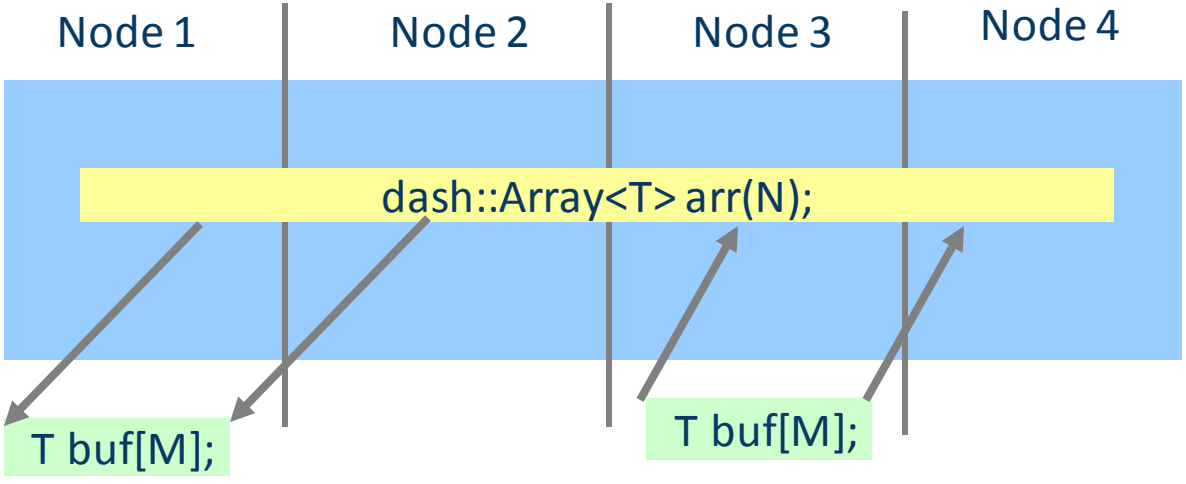
dash
www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

MORE ABOUT DASH

Bulk Data Transfers: dash::copy()

General syntax:
out = dash::copy(in_first, in_last, out_first);



Global to Local:
dash::copy(glob_first, glob_last, loc_first);

Local to Global
dash::copy(loc_first, loc_last, glob_first);

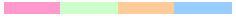
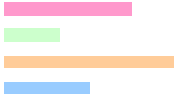
More about DASH – Algorithms

Examples

For all i in range [first, last):

• dash::fill	arr[i] = val
• dash::generate	arr[i] = func()
• dash::for_each	func(arr[i])
• dash::transform	arr2[i] = func(arr1[i])
• dash::min_element	min({arr[i]})
• dash::max_element	min({arr[i]})

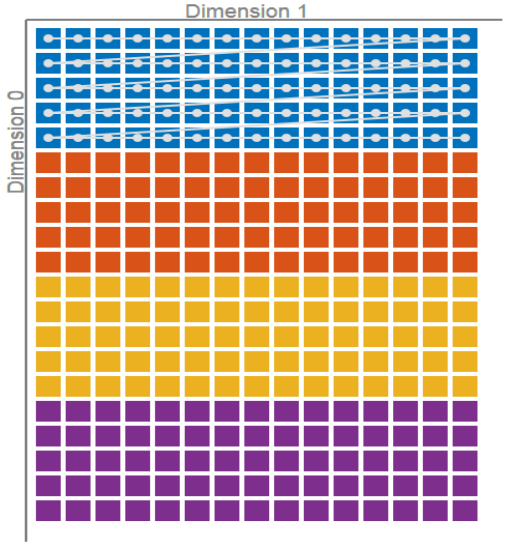
More about DASH – Containers

Container	Description	Data distribution
Array<T>	1D Array 	static, configurable
NArray<T, N>	N-dim. Array 	static, configurable
Shared<T>	Shared scalar 	fixed (at 0)
Directory(*)<T> (*) Under construction	Variable-size, locally indexed array 	manual

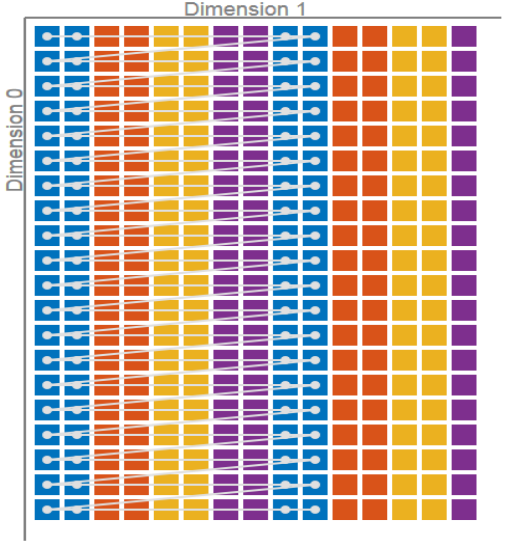
Dynamic data structures (growing/shrinking) are under development
See http://doc.dash-project.org/api/latest/html/group__DashContainerConcept.html

Multidimensional Data Distribution Patterns

Pattern<2>(20, 15)



(BLOCKED, NONE)

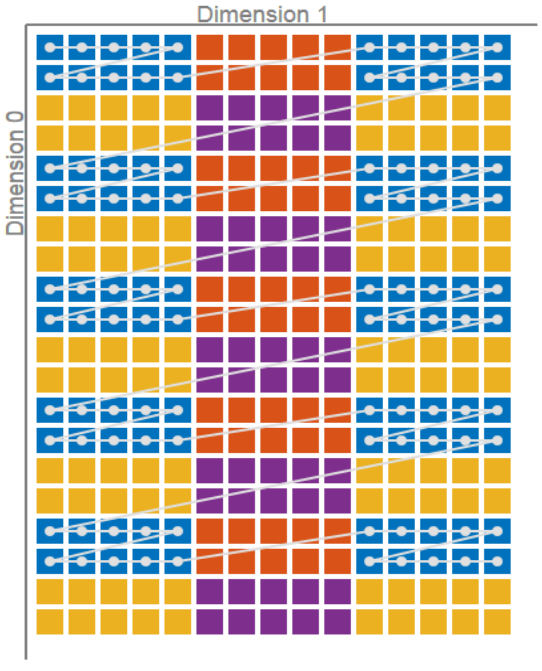


(NONE, BLOCKCYCLIC(2))

(BLOCKED, BLOCKCYCLIC(3))

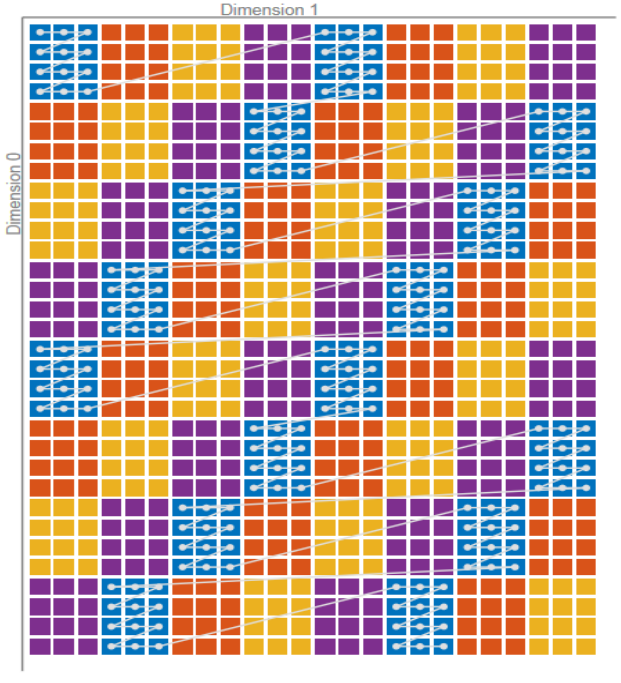
Multidimensional Data Distribution Patterns

TilePattern<2>(20, 15)



(TILE(2), TILE(5))

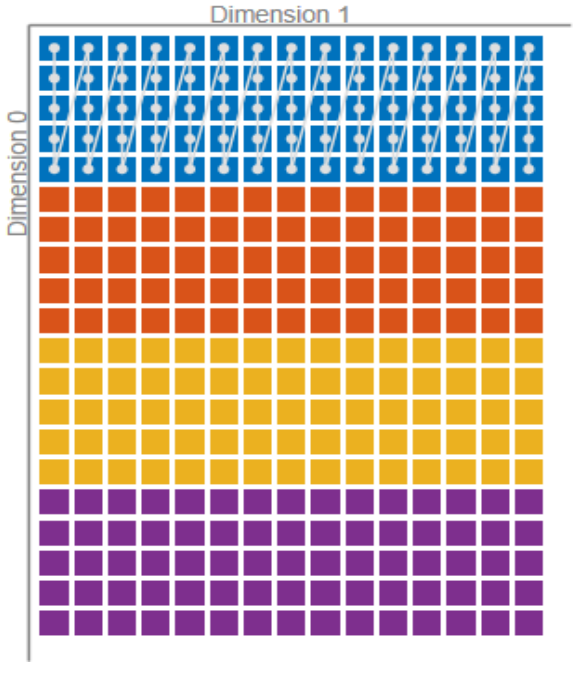
ShiftTilePattern<2>(32, 24)



(TILE(4), TILE(3))

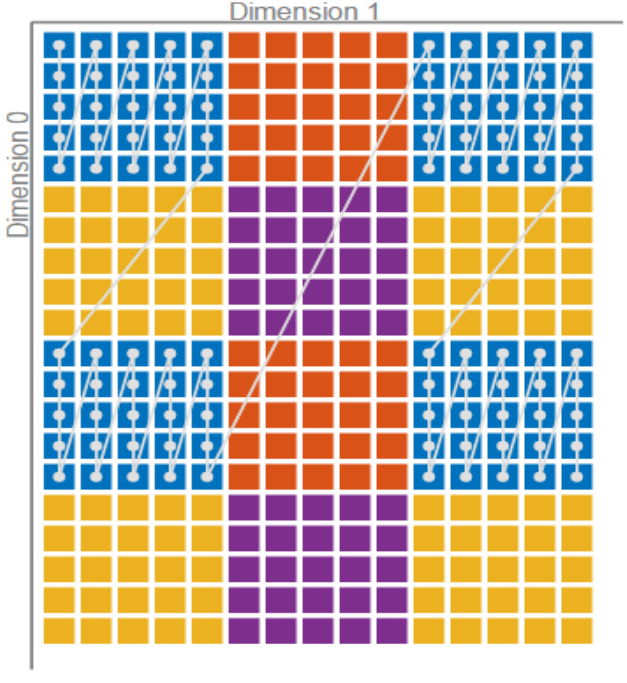
Multidimensional Data Distribution Patterns

Pattern<2, COL_MAJOR>(20, 15)

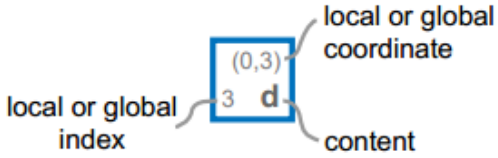


(BLOCKED, NONE)

TilePattern<2, COL_MAJOR>(20, 15)



(TILE(5), TILE(5))



N-dimensional Array Example

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)

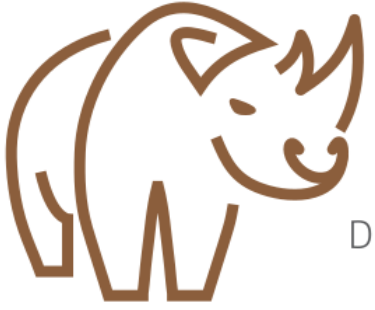
(0,0) 0 e	(0,1) 1 f	(0,2) 2 g	(0,3) 3 h
(1,0) 4 q	(1,1) 5 r	(1,2) 6 s	(1,3) 7 t

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

TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



dash

www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

HANDS-ON: PROCESSING HUGE ASTRONOMICAL IMAGES WITH DASH

Processing A Huge Astronomical Image

Task: Count the objects in this image

- Read image into a distributed DASH data structure
- Get pixel brightness histogram in parallel, define separation between black and bright pixels.
- Find bright pixels, flood-fill all adjacent bright pixels and count it as one object, do in parallel

"GOODS South field" with 31813 x 19425 px from http://www.spacetelescope.org/images/heic1620a/.

Released 2016-10-13 by ESA/Hubble under Creative Commons Attribution 4.0 Intl. license, see http://www.spacetelescope.org/copyright/.

Disclaimer

This is close enough for computer science tutorial purposes to actual algorithms used by astronomers.

In the end we will demonstrate an extended version with object indices which requires more memory.

Still, do not use this tutorial as your only source in case you pursue a Ph.D. in astronomy.

Hands-on assignment 02

Got to the 02-astro_load_image/ subdir and familiarize yourself with the fill-in-the-gaps code in astro-assignment.cpp.

- DASH init etc.
- Unit 0 opens the image with libtiff, determines it's width and height.
- Communicates image width and height to other units.
- **Assignment 1)** Create the distributed DASH NArray of proper size.
- Unit 0 reads the actual image line by line into a small local array.
- **Assignment 2)** Copy lines from the local byte array to the DASH NArray.

Hands-on 02 Detail

Unit 0 communicates image width and height to other units using a dash::Shared.

- It doesn't look like parallel communication but it is.
- It is not the most efficient way to do it but here it just doesn't matter.
- Do not access "shared" all the time, instead use local copy.

```
struct ImageSize {
    uint32_t height;
    uint32_t width;
};
…
dash::Shared<ImageSize> shared {};

if ( 0 == myid ) {

    …

    tif = TIFFOpen( argv[1], "r" );
    TIFFGetField(tif, … , &w );
    TIFFGetField(tif, … , &h );

    shared.set( {h, w} );
}

shared.barrier();

ImageSize imagesize= shared.get();
```

Hands-on assignment 02 + solution

Assignment 1) Declare and allocate a distributed 2D DASH NArray container with given width and height with the RGB data type.
Select BLOCKED and NONE distribution for the first resp. second dimension.

Distributing rows

```
dash::TeamSpec<2> teamspec{};
…
auto distspec= dash::DistributionSpec<2>( dash::BLOCKED, dash::NONE );

dash::NArray<RGB, 2> matrix(
        dash::SizeSpec<2>( imagesize.height, imagesize.width ),
        distspec, dash::Team::All(), teamspec );
```

Hands-on assignment 02 contd.

Assignment 2) Copy the next 'rowsperstrip' lines from the local buffer 'rgb' with width 'w' each to the proper target position in the DASH matrix.

Solution 02

```
auto iter= matrix.begin();
…
for ( uint32_t l= 0; ( l<rowsperstrip ) && (line+l<imagesize.height); l++ ) {

    iter = dash::copy( rgb, rgb+imagesize.width, iter );
    rgb += imagesize.width;
}
```

- dash::copy() does the trick, nothing else needed.
- There is no index calculation for the DASH matrix.
- Note that it uses a global iterator for the DASH matrix but that's okay because it is not dereferenced directly, only passed to dash::copy().

Hands-on assignment 03

Got to the 03-astro_histogram/ subdir and look at astro-assignment.cpp which continues the previous code.

Assignment 1) Compute a pixel brightness histogram for the entire image with BINS (constant) histogram bins. Create a distributed DASH array for the histogram of size (BINS x team size) which is block distributed over all units. Fill it with '0' using a DASH algorithm.

Assignment 2) Iterate over all local pixels in the distributed matrix using the local iterator. For every pixel determine the brightness bin in the histogram of BINS bins. Then increment the corresponding histogram bin in the local part of the histogram by 1.

Hands-on assignment 03 contd.

Assignment 3) For all units except unit 0, add their histogram bins to the values of the histogram of unit 0. Try using a DASH algorithm for that.

Assignment 4) Add a barrier for the histogram data structure at the specified place.

Assignment 5) Use the given function 'print_histogram()' to print out the global histogram.

Assignment 6) Near the end put the limit where the pixel color is considered part of a bright object instead of the dark background. Look at the histogram you produced to find a good limit.

Solution 03

For assignment 1) Create a distributed DASH array for the histogram of size 'BINS' x team size which is block distributed over all units. Fill it with '0' using a DASH algorithm.

```
dash::Array<uint32_t> histogram( BINS * numunits, dash::BLOCKED );
dash::fill( histogram.begin(), histogram.end(), (uint32_t) 0 );
```

For assignment 2) Iterate over all local pixels in the distributed matrix using the local iterator. For every pixel determine the brightness bin in the histogram of 'BINS' bins. Then increment the corresponding histogram bin in the local part of the histogram by 1.

```
for ( auto it= matrix.lbegin(); it != matrix.lend(); ++it ) {

    histogram.local[ it->brightness() * BINS / MAXKEY ]++;
}
```

Solution 03

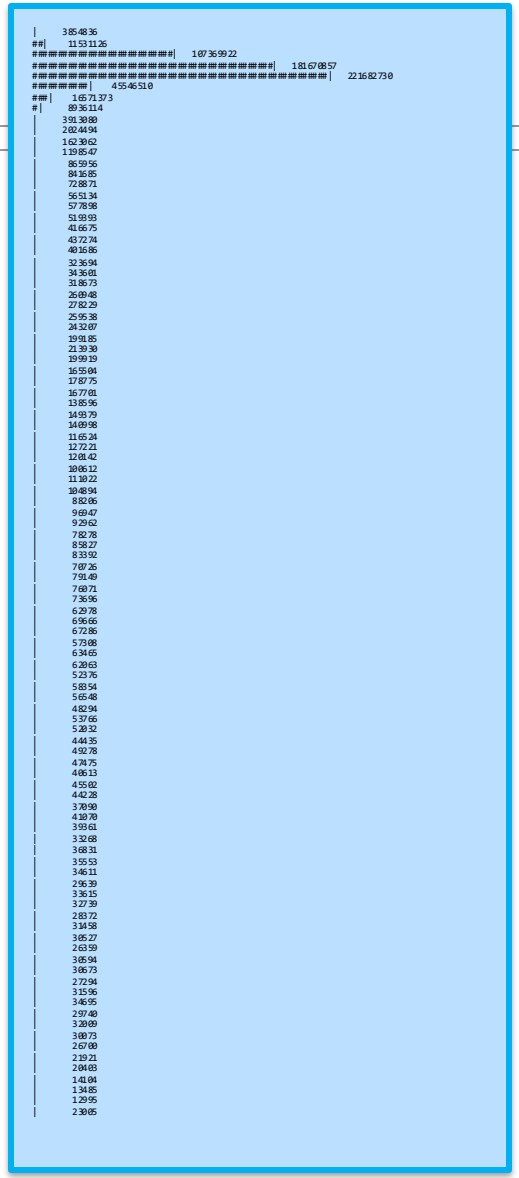
For assignments 3 and 4)

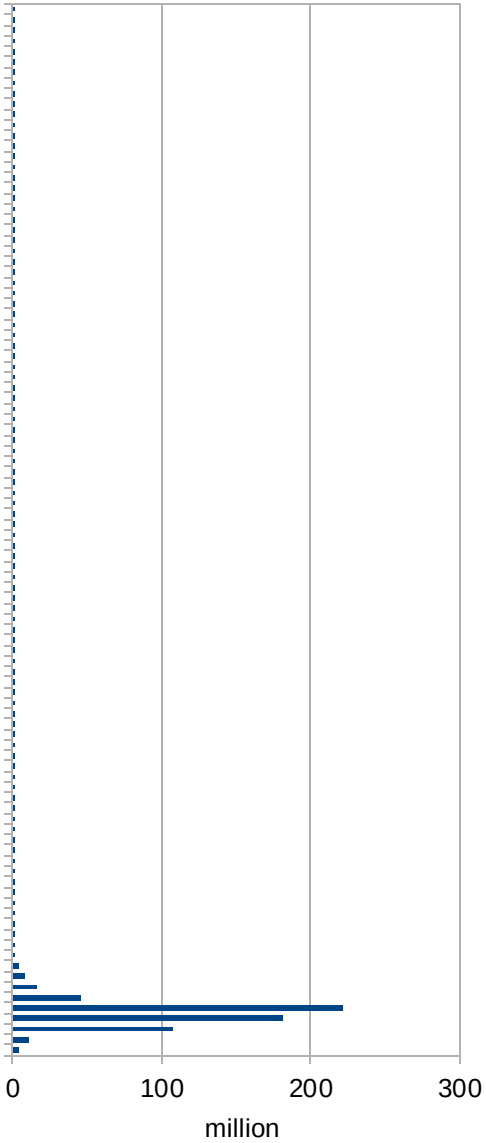
```
histogram.barrier();
if ( 0 != myid ) {
    dash::transform<uint32_t>( histogram.lbegin(), histogram.lend(),
        histogram.begin(), histogram.begin(), dash::plus<uint32_t>() );
}
histogram.barrier();
```

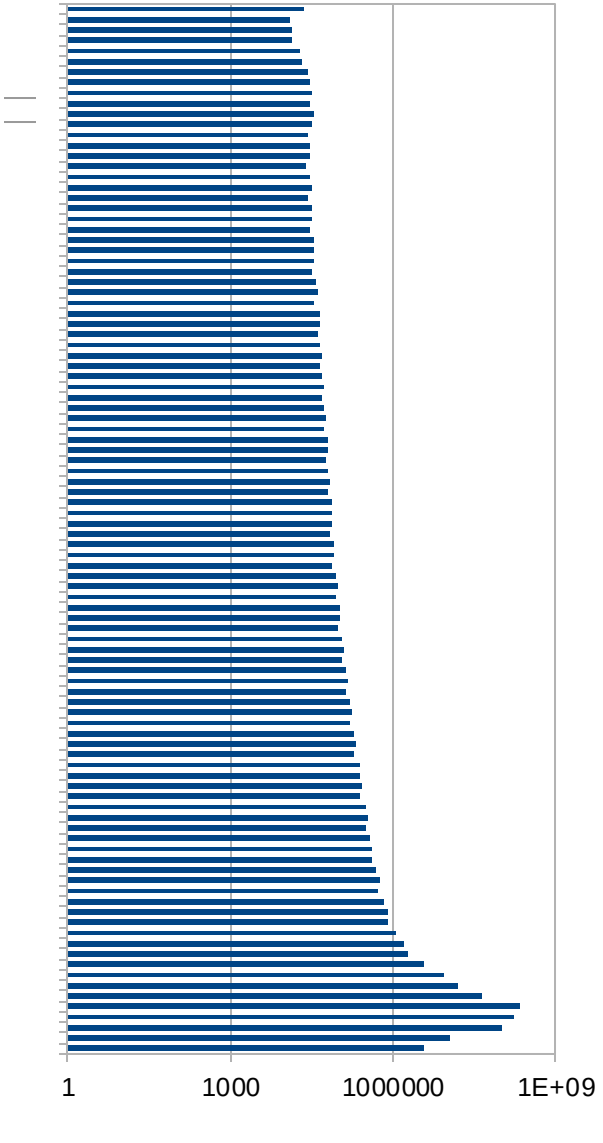
For assignment 5 and 6)

```
print_histogram<uint32_t*>( histogram.lbegin(), histogram.lend() );
…
const uint32_t limit= 255*3*2/17;
```

```
computed parallel histogram in 15 seconds

##################################################################|   568321679
##|    37327661
|     4183916
|     2273500
|     1479900
|     1039184
|      726282
|      554720
|      448845
|      376511
|      317329
|      274541
|      221885
|      188508
|      172852
|      180186
|      102717
```


3854836
11531126
10736922
18167857
221682730
45546510
16571373
8936114
3913080
2824494
1623062
1198547
865956
841685
728871
565134
578898
519393
416675
437274
401686
323694
343681
318673
260948
278229
259538
243207
199185
213930
199919
165504
178775
167701
138596
149379
140998
116524
127221
120142
109612
111022
104894
88206
96947
92962
78278
85827
83392
70726
79149
76071
73696
62978
69666
67286
57308
63465
62063
52376
58354
56548
48294
53766
52032
44435
49278
47475
46613
45502
44228
37090
41070
39361
33268
36831
35553
34611
29639
33615
32739
28372
31458
30527
26359
30594
30673
27294
31596
34695
29740
32089
30073
26700
21921
20403
14104
13485
12995
23005


0
100
200
300
million


1
1000
1000000
1E+09

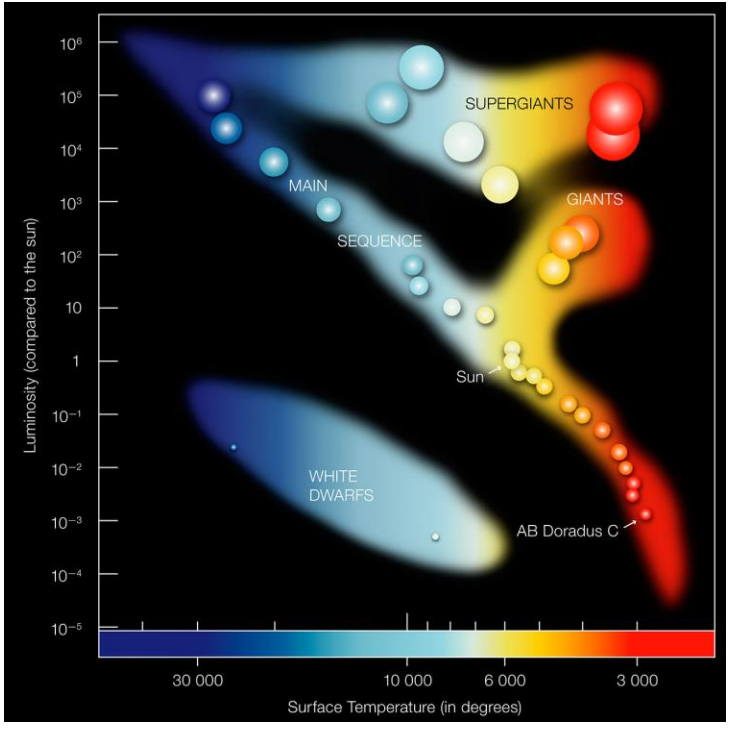
Hands-on assignment 04

Go to the 04-astro_marker_color/ subdir.

The following code for counting the objects relies on marking already visited pixels with a color that is not yet used in the image.

Assignment: Guess a marker color. Then make each unit check if / how often this color appears. Run the example with new guesses until you find an unused color.

Hint: What color is missing in the Hertzsprung-Russel diagram?



https://cdn.eso.org/images/screen/eso0728c.jpg

Solution 04

```
RGB marker(0,255,0);

uint64_t count= 0;
for ( auto it= matrix.lbegin(); it != matrix.lend(); ++it ) {
    if ( marker == *it ) count++;
}
```

```
for ( auto it : matrix.local ) {
    if ( marker == it ) count++;
}
```

```
auto ret= std::find( matrix.lbegin(), matrix.lend(), marker );
bool found= ( matrix.lend() != ret );
```

Hands-on assignment 05

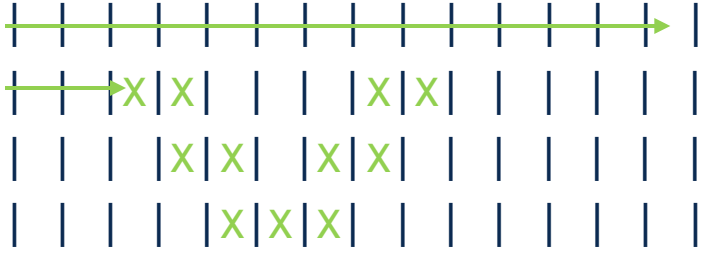
Go to 05-astro_count_objects/

Consider the flood-fill algorithm:

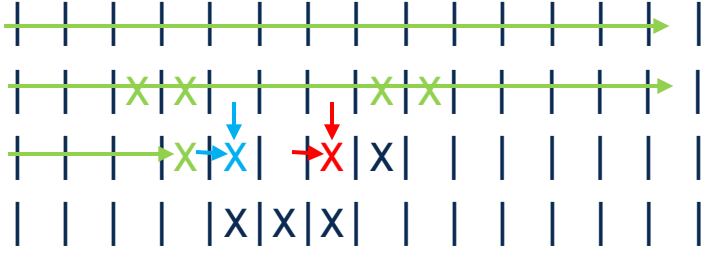
- Iterate over all pixels
- Count every bright pixel, that is unmarked. Then mark all its connected bright pixels with a flood-fill operation.

- A scan-line algorithm checking if the adjacent pixels (top, left) were already marked would not be sufficient.

Flood-fill algorithm



Scan-line algorithm



Hands-on assignment 05

Look at the `checkobject()` routine given in `mish.h`.

- Usea the start ptr of local array and find pixel with pointer arithmetic.
- If already marked or not a bright pixel, return 0
- Otherwise, put position into queue (`std::set`) of pixels to be marked. Later return 1 for one new object detected.

```
uint32_t checkobject( RGB* ptr,
        uint32_t x, uint32_t y,
        uint32_t w, uint32_t h,
        uint32_t limit, RGB marker ) {

    RGB* pixel= ptr + y*w+ x;

    if ( ( *pixel == marker ) ||
         ( pixel->brightness() < limit ) )
        return 0;

    std::set< std::pair<int,int> > queue;
    queue.insert( { x, y } );
    *pixel= marker;

    …

    return 1;
```

```
while ( ! queue.empty() ) {

    auto next= *queue.begin();
    int x= next.first;
    int y= next.second;
    RGB* pixel= ptr + y*w+ x;

    queue.erase( queue.begin() );

    if ( pixel->brightness() < limit )
        continue;

    *pixel= marker;
    if ((0<x)   && (*(pixel-1)!=marker)) queue.insert({x-1,y});
    if ((x+1<w) && (*(pixel+1)!=marker)) queue.insert({x+1,y});
    if ((0<y)   && (*(pixel-w)!=marker)) queue.insert({x,y-1});
    if ((y+1<h) && (*(pixel+w)!=marker)) queue.insert({x,y+1});
}
```

While queue not empty

- Remove first position from queue.
- If it is a bright pixel, then add all adjacent pixels into the queue unless they are already marked.

Wouldn't a recursive implementation be much shorter and better?

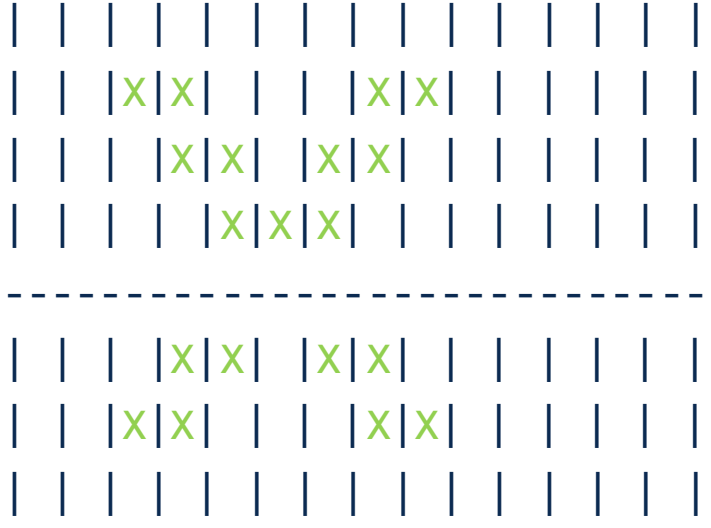
Actually not …

Hands-on assignment 05 contd.

Assignment 1) Iterate over the local pixels using x-y coordinates and the horizontal/vertical local extents.

For every pixel call the given `checkobject()` routine and sum all return values in a local variable.

For now: Ignore that objects will be counted multiple times at the partition borders between distribution blocks.
See later how to correct this.



Hands-on assignment 05 contd.

Assignment 2) Sum the local results of bright objects into the global result.

Use either std::accumulate() together with the correct iterator or use the same manner as for the histogram above.

Then output the global number of bright objects once.

Solution 05

For assignment 1)

```
constexpr uint32_t limit= 256*3*2/17;
uint32_t lw= matrix.local.extent(1);
uint32_t lh= matrix.local.extent(0);
uint32_t found= 0;
for ( uint32_t y= 0; y < lh ; y++ ){
for ( uint32_t x= 0; x < lw ; x++ ){

    found += checkobject(matrix.lbegin(),x,y,lw,lh,limit,marker);
}
}
```

Solution 05

For assignment 2)

```
dash::Array<uint32_t> sums( numunits, dash::BLOCKED );
auto foundobjects = sums.lbegin();
for ( … ) {
for ( … ) {
    foundobjects += checkobject( … );
}
}
sums.barrier();

if ( 0 == myid ) {
    sum_objects = std::accumulate( sums.begin(), sums.end(), 0 );
    cout << "found " << sum_objects << " objects in total" << endl;
}
```

Congratulations! You finished all astro hands-on assignments.

Less than 400 lines of code in total can:

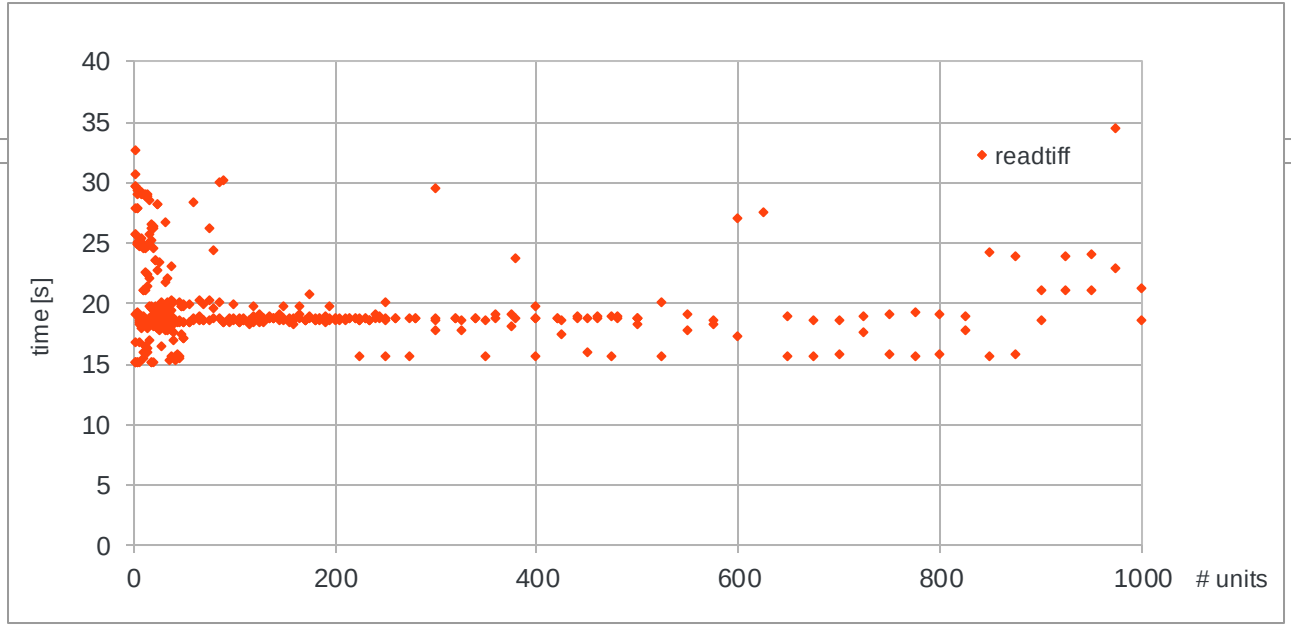
- Read a huge TIFF image and distribute it over all parallel units
- Compute a brightness histogram
- Count the number of bright objects

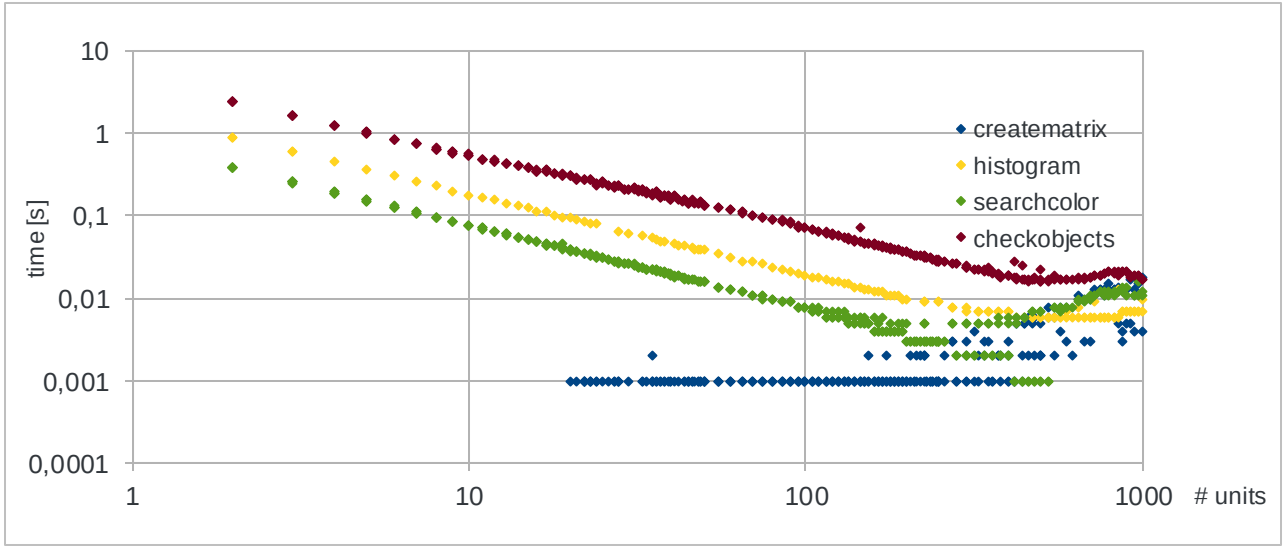
Furthermore it includes:

- Print histogram
- Simple graphical output of a section of the images
- Comments

Benchmarks

- On cluster with dual Xeon E5-2680v3 Haswell, 2.5 GHz, 24 cores per node
- Constant time for sequential reading as expected
- Perfect parallel scaling for pixel operations up to approx. 300 units (approx. 64 lines or 2M pixels per unit)





TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

dash

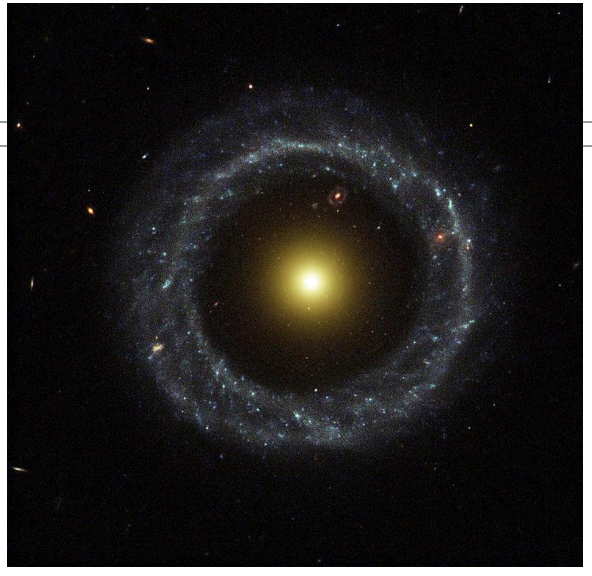
www.dash-project.org

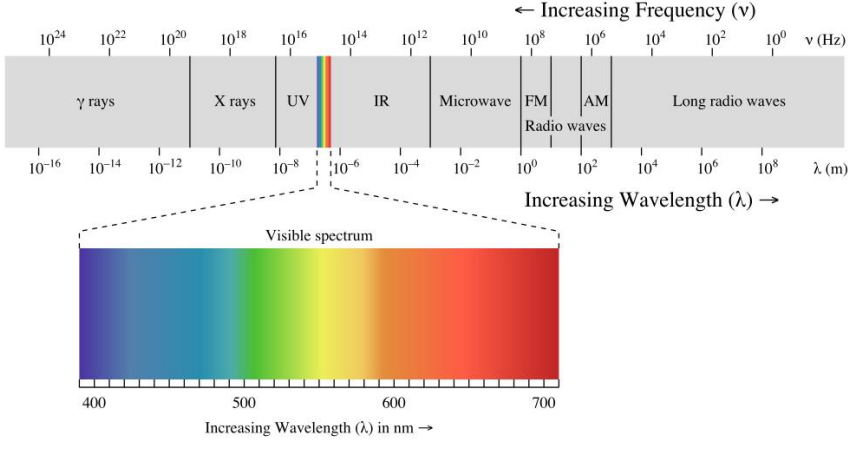
Distributed Data Structures and Parallel Algorithms

DASH HALOS + ASTRONOMICAL IMAGE PROCESSING IMPROVED WITH HALOS

What an astronomer would do differently

- Single bright pixels are ignored.
- Do histograms by color channel.
- Detect objects by color channel, then register different results.





- Do color clustering to discern objects behind other objects.
- Assign ascending IDs to clusters of pixels, not merely a marker color.
  --> Compensate for objects counted multiple times at the block borders.
  --> Needs 4 extra bytes per pixel.

https://commons.wikimedia.org/wiki/File:EM_spectrum.svg
https://commons.wikimedia.org/wiki/File:Hoag's_object.jpg

Corrected code a.k.a. a bit closer to what astronomers would do

- Assign IDs to object pixels
- Need a separate 2D array of same dimensions for the IDs
- Thus do not change any pixels

- Step 1: detect objects locally and assign globally unique IDs
- Step 2: check for neighbor IDs at distribution borders
- Step 3: for every neighboring pair of IDs reduce the number of objects by one

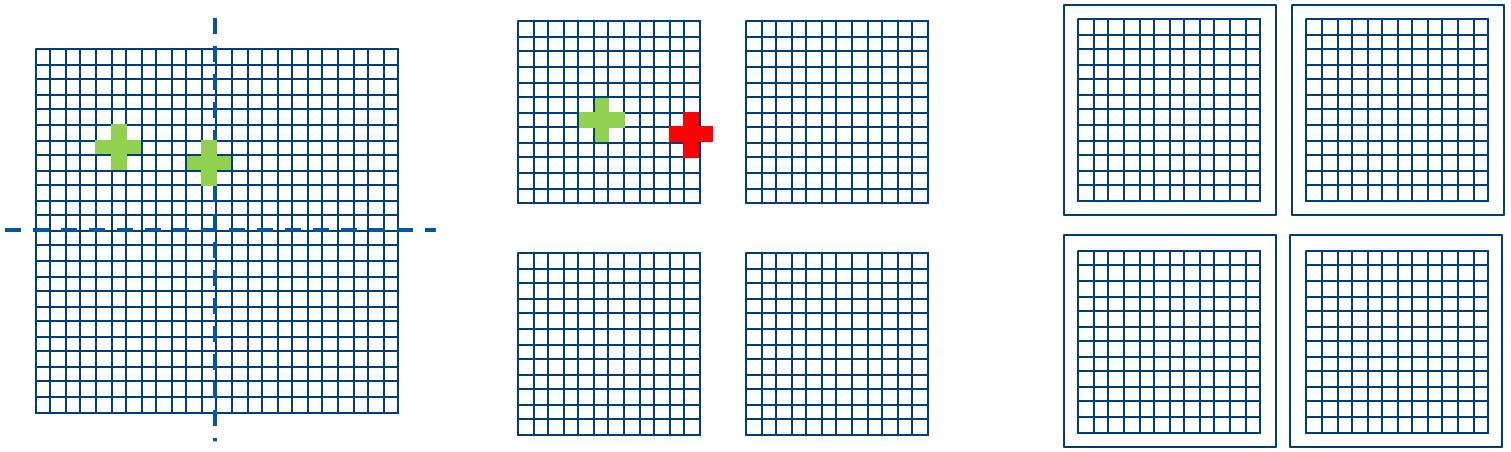
Use the most convenient DASH halo class for data exchange at the borders …

--> go to 06-astro_count_with_ids/

The DASH halo wrapper class

- A typical access pattern uses a cell and some of its neighbors
- Simple in the inner area but not at distribution borders – depends on global data distribution

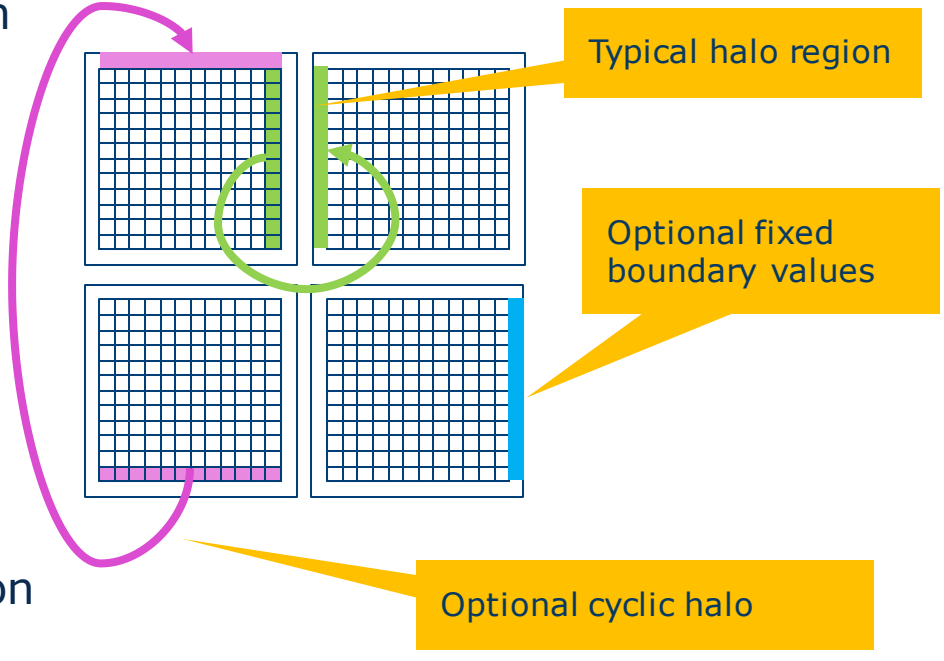
- A halo wrapper can be added to a DASH array. It provides easy access to neighbor cells across distribution borders



The DASH halo wrapper class

- In 1/2/3/4/… n dimensions
- Flexible halo width per dimension
- Halo areas between partitions
- Optionally at global borders:
  - cyclic halos or
  - fixed boundary values

- One call for collective halo value update or two for async update
- Provide extra iterators for inner area, edges, and halo region



Using halo to correct object count at distribution borders

Step 1: Add ID array ids next to pixel array. The original code was:

```
dash::TeamSpec<2> teamspec{};
auto distspec= dash::DistributionSpec<2>( dash::BLOCKED, dash::NONE );
dash::NArray<RGB, 2> matrix(
        dash::SizeSpec<2>( imagesize.height, imagesize.width ),
        distspec, dash::Team::All(), teamspec );
```

… use identical extents and the same distribution pattern for ids array:

```
dash::NArray<uint32_t,2> ids(
        dash::SizeSpec<2>( imagesize.height, imagesize.width ),
        distspec, dash::Team::All(), teamspec );
dash::fill( ids.begin(), ids.end(), 0 );
```

Using halo to correct object count at distribution borders (contd.)

Step 2: Use same algorithm to identify objects locally as before

- Do not mark pixels with green color anymore
- Mark them in ids array with id != 0, use unique ID ranges per unit

```
countobjects( matrix, ids, 1 + (1<<30) / numunits, limit );
```

Pixel array

ID array

Start id != 0 per unit

Limit for bright pixels

- countobjects() will return the local object count as before
  … see given code
- Again, accumulate local number into the global result

Using halo to correct object count at distribution borders (contd.)

Step 3: Add a halo wrapper to the DASH array ids:

```
constexpr dash::StencilSpec<2,4> stencil_spec({
    dash::Stencil<2>(-1, 0), dash::Stencil<2>( 1, 0),
    dash::Stencil<2>( 0,-1), dash::Stencil<2>( 0, 1)});
constexpr dash::CycleSpec<2> cycle_spec(
    dash::Cycle::NONE, dash::Cycle::NONE );

dash::HaloMatrixWrapper< dash::NArray<uint32_t,2>, dash::StencilSpec<2,4> >
    halo( ids, stencil_spec, cycle_spec );
halo.update();
```

StencilSpec defines the access width +/-1 per direction and thus the halo widths

CycleSpec: no cyclic boundaries

Add halo to existing DASH array ids

Update all halo values from the original fields, also avail in an async manner

Using halo to correct object count at distribution borders (contd.)

Step 4: Collect all ID pairs that touch at partition borders:

```
std::set<std::pair<uint32_t,uint32_t>> touchset;
…
/* if there is an upper neighbor partition */
for ( uint32_t x= 0; x < lw ; x++ ) {

    uint32_t l= ids.local[0][x];
    uint32_t r= *halo.halo_element_at( {upperleft[0]-1,x+upperleft[1]} );

    if ( ( 0 != localid ) && ( 0 != remoteid ) && ( localid < remoteid ) )
        touchset.insert( {localid, remoteid} );
}
```

std::set will accept every pair only once, not multiple times

Border field in local coordinates

Neighbor field in halo in global coordinates

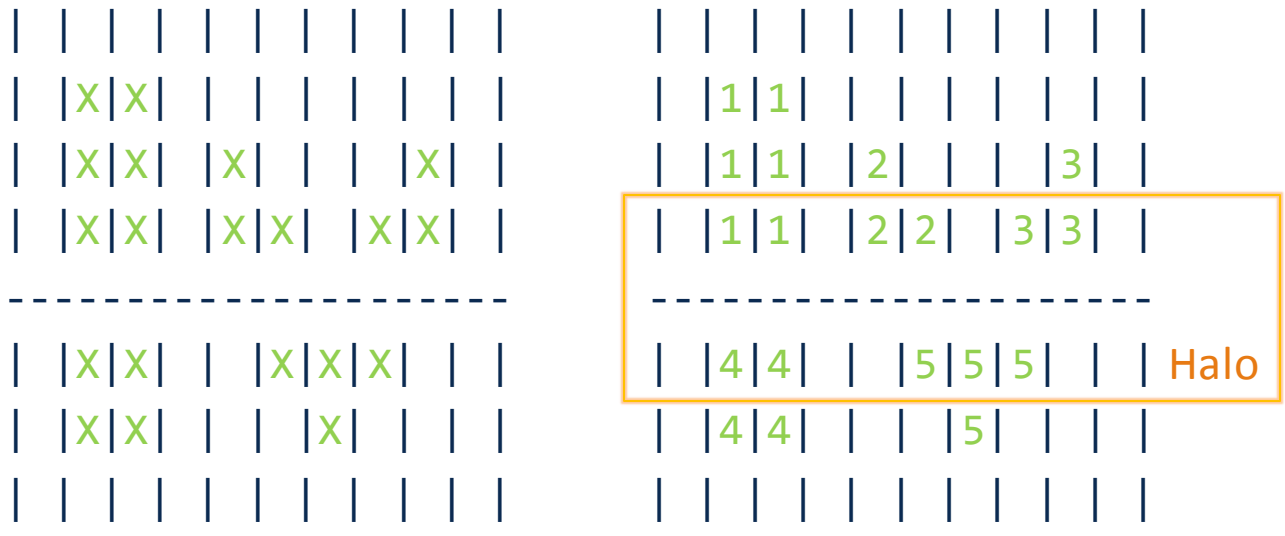
Only once!

Insert real pairs != 0

… do for all four potential neighbors

Using halo to correct object count at distribution borders (contd.)

Now how to correct for split objects counted twice or more?



In total found
L= 5 objects

Pairs in touchset:
(1,4), (2,5), (3,5)
S= touchset.size();

Correct count is
C= L - S

TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

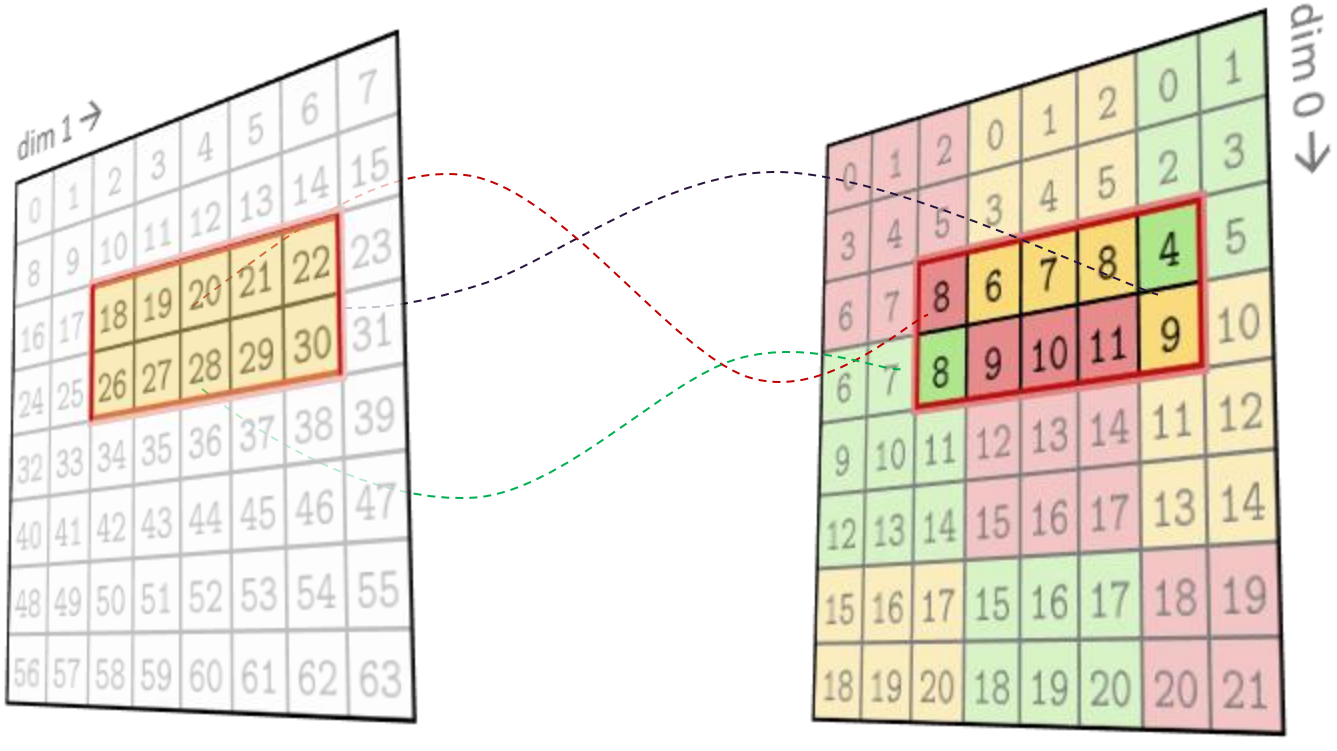
dash

www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

MULTIDIMENSIONAL ARRAY VIEWS

PREVIEW TO UPCOMING RELEASE


dim 1 →
dim 0 ↓

```
dash::Matrix<int, 2> mat(8,8, … );
```

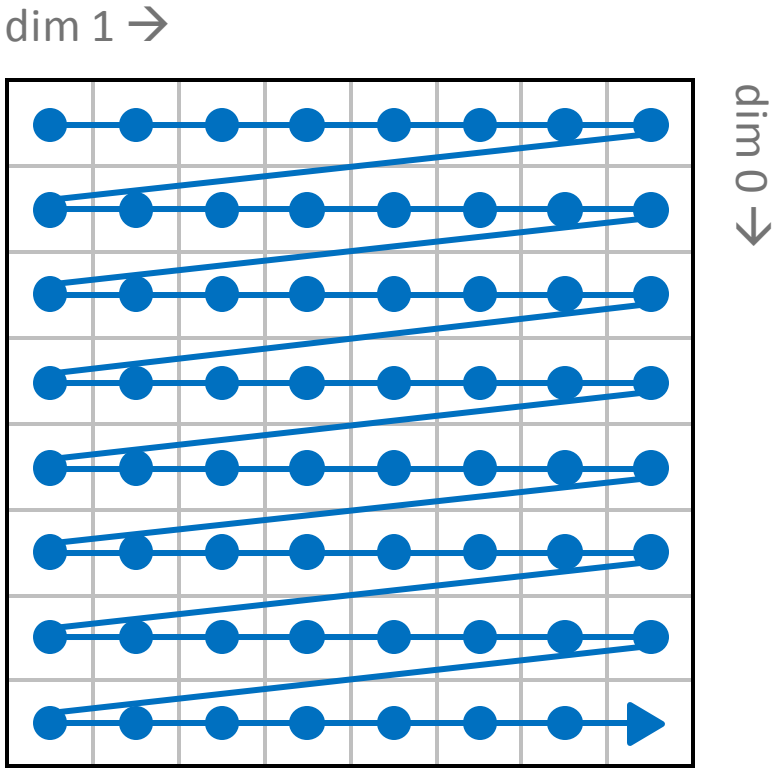
dim 1 →

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

dim 0 ↓

* we use the term *index set* to differentiate from *index sequences*

```
dash::Matrix<int, 2> mat(8,8, … );
```



* we use the term *index set* to differentiate from *index sequences*

```
dash::Matrix<int, 2> mat(8,8, … );
```

dim 1 →

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

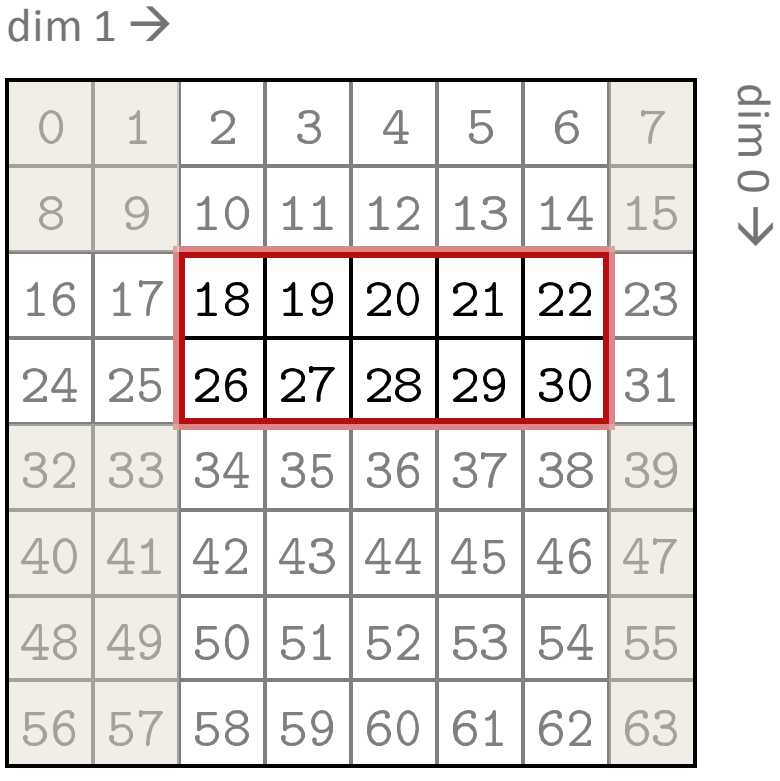
dim 0 ↓

Two iteration orders:

- **canonical order**
  vs.
- **storage order**

* we use the term *index set* to differentiate from *index sequences*

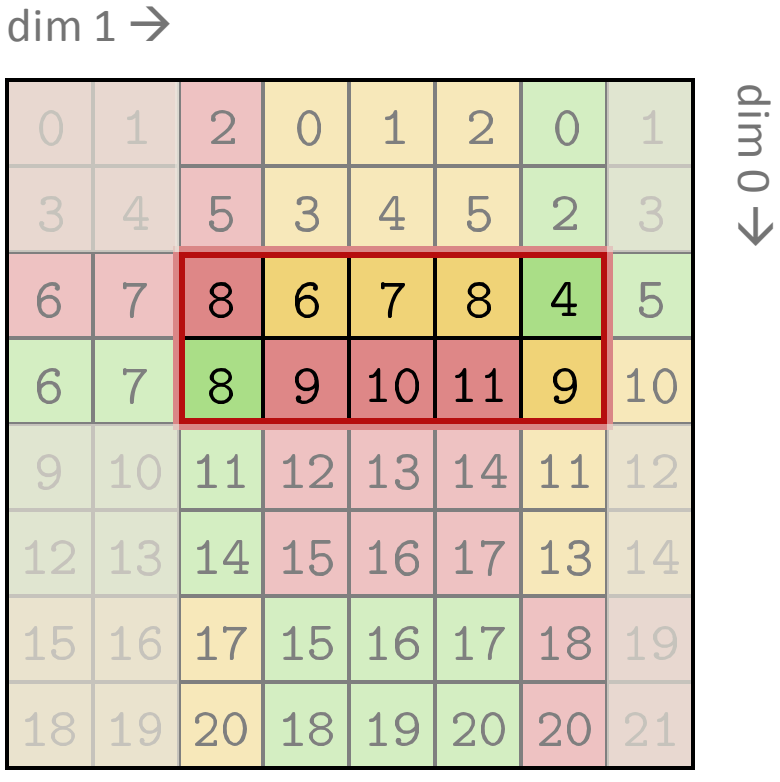
```
dash::Matrix<int, 2>
  mat(8,8, … );

auto mat_rect = mat | sub<0>(2, 4)
                    | sub<1>(2, 7);
```



* we use the term *index set* to differentiate from *index sequences*

```
dash::Matrix<int, 2>
  mat(8,8, … );

auto mat_rect = mat | sub<0>(2, 4)
                    | sub<1>(2, 7);
```

```
mat_rect | index()
         | join();
// result, depending on unit:
unit 0:  { 8, 9, 10, 11 }
unit 1:  { 6, 7, 8, 9 }
unit 2:  { 4, 8 }
```



* we use the term *index set* to differentiate from *index sequences*

```
begin()              size()
end()                rank()

offsets()            extents()

domain()             local()
origin()             global()
index()              pattern()

blocks()             join()
chunks()             stride(i0…id)

sub<d>(b,e)
expand<d>(ob,oe)     shift<d>(o)

halo()               border()
```

```
is_local_at(u)
is_contiguous()
is_global()
is_local()
```

```
view_traits<V>
  iterator
  value_type
  ...
  is_local_view
  is_global_view
  is_origin
  rank
```

* we use the term *index set* to differentiate from *index sequences*

TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

dash

www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

MORE DASH EXAMPLE CODES

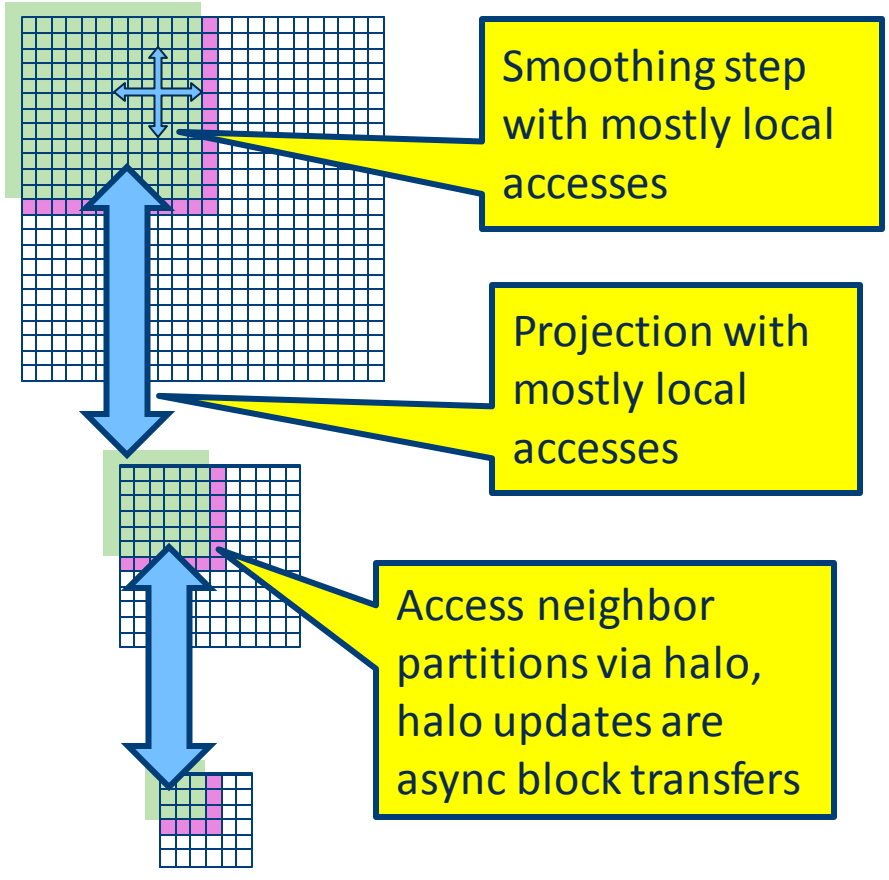
More DASH Example codes

see more example codes https://github.com/dash-project/dash-apps.git:

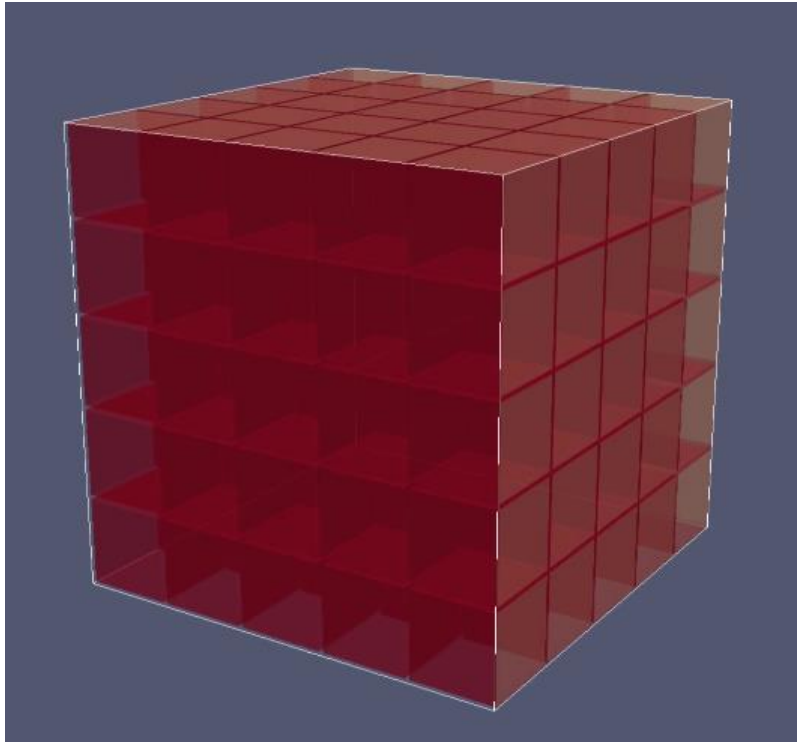
- all-pairs: benchmark pairwise communication between units
- cowichan: set of Cowichan benchmarks ported to DASH
- hpccg: Simple conjugate gradient benchmark code, several versions
- isx: Scalable integer sort application
- lulesh: incremental port of lulesh benchmark
- minife: Finite element demo code
- multigrid: multigrid PDE solver for heat equation with visualization
- NPB-3.3.1: three NPB benchmark codes ported to DASH

Example: Multigrid PDE solver

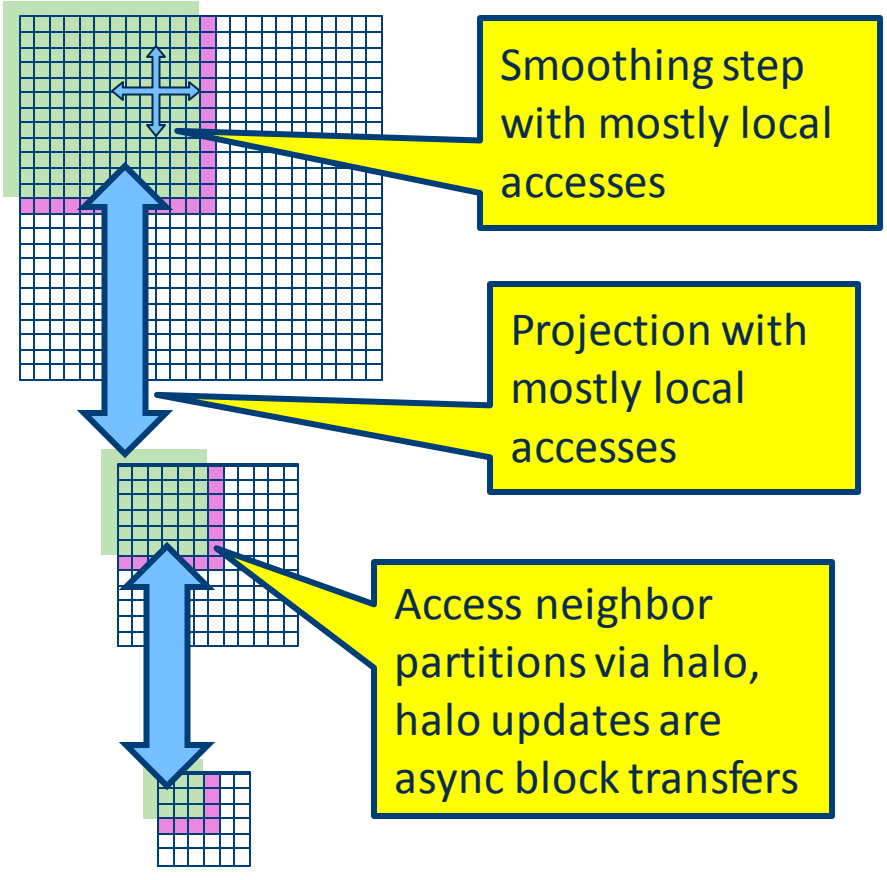
- Implement a multigrid solver with a set of DASH arrays of different resolution
- Solve PDE on coarse grid, then map to finer grid and smooth out remaining errors … get steady state solution much quicker than iterative smoothing on finest grid
- Use matching grid partitioning
- Use halo on partition borders
- https://github.com/dash-project/dash-apps.git --> multigrid



Example: Multigrid PDE solver



(video)



dash
www.dash-project.org

Distributed Data Structures
and Parallel Algorithms

MORE RESOURCES ABOUT DASH

DASH Components as of v0.3.0

Static Containers

- Array
- Matrix / NArray

Dynamic Containers

- Set / MultiSet
- List

DASH Algorithms

- copy, fill, for_each, generate, reduce, transform, generate, max_element, min_element

More at http://doc.dash-project.org/

DASH Concepts

- Dimensional
- DistributionSpec : Dimensional
- CartesianSpace = SizeSpec
- CartesianIndexSpace : CartesianSpace
- TeamSpec : CartesianSpace
- Pattern (TilePattern, BlockPattern)
- Pattern Iterator, Pattern Block Iterator
- Container
- Global Iterator / Pointer
- Global Asynchronous Reference

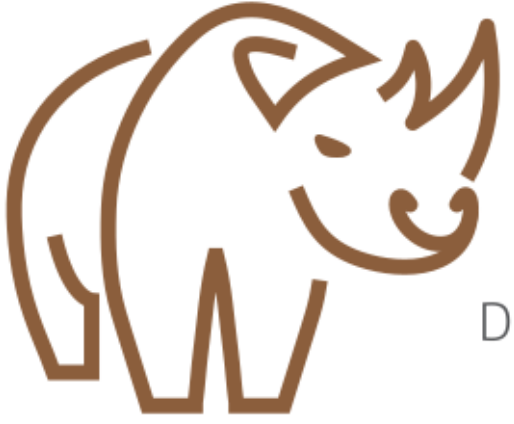
How to learn more

- Small code examples under https://github.com/dash-project/dash.git --> dash/example, see also http://doc.dash-project.org/Examples
- More extensive examples and partial ports of HPC benchmarks under https://github.com/dash-project/dash-apps.git
- Contributor guide under http://doc.dash-project.org/
- Publications under http://www.dash-project.org/publications.html

app.01.mindeg
bench.01.igups
bench.02.memhog
bench.02.meminit
bench.03.gups
bench.04.histo
bench.04.histo-tf
bench.05.array-range
bench.05.pattern
bench.06.jacobi-1d
bench.07.local-copy
bench.08.min-element
bench.08.transform
bench.09.multi-reader
bench.10.summa
bench.11.hdf-io
bench.12.for_each
bench.h
ex.01.hello
ex.01.hello-mpi
ex.02.array
ex.02.array-copy
ex.02.array-csr
ex.02.array-ctors
ex.02.array-delayed
ex.02.array-local
ex.02.halo-swap
ex.02.matrix
ex.02.matrix-multiply
ex.02.matrix.halo.heat_equation
ex.02.matrix_distribution_patterns
ex.02.matrix_global_access
ex.02.matrix_printed_in_different_ways
ex.02.unordered_map
ex.03.globref
ex.04.memalloc
ex.05.min_element
ex.06.pattern-block-visualizer
ex.06.pattern-visualizer
ex.06.std-algo
ex.07.locality
ex.07.locality-group
ex.07.locality-split
ex.07.locality-threads
ex.08.io-hdf5
ex.08.io-hdf5-stream
ex.09.view-1dim
ex.10.psort
ex.10.radixsort
ex.11.halo-stencil
ex.11.simple-stencil

Thank you very much for your attention!

- We hope, you enjoyed the tutorial and found the DASH perspective on parallel programming appealing.
- Please send feedback to andreas.knuepfer@tu-dresden.de and tobias.fuchs@nm.ifi.lmu.de



dash
www.dash-project.org

Distributed Data Structures
and Parallel Algorithms