Concepts and Models of Parallel and Data-centric Programming

Distributed Shared Memory

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

Outline

DASH Overview

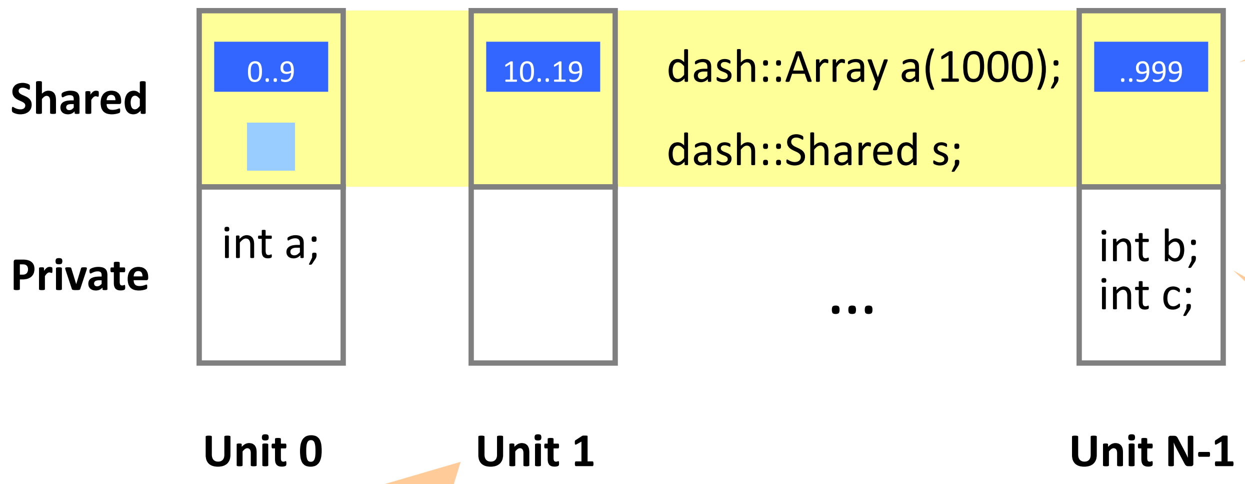
Material provided by Karl Fürlinger, LMU

www.dash-project.org

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



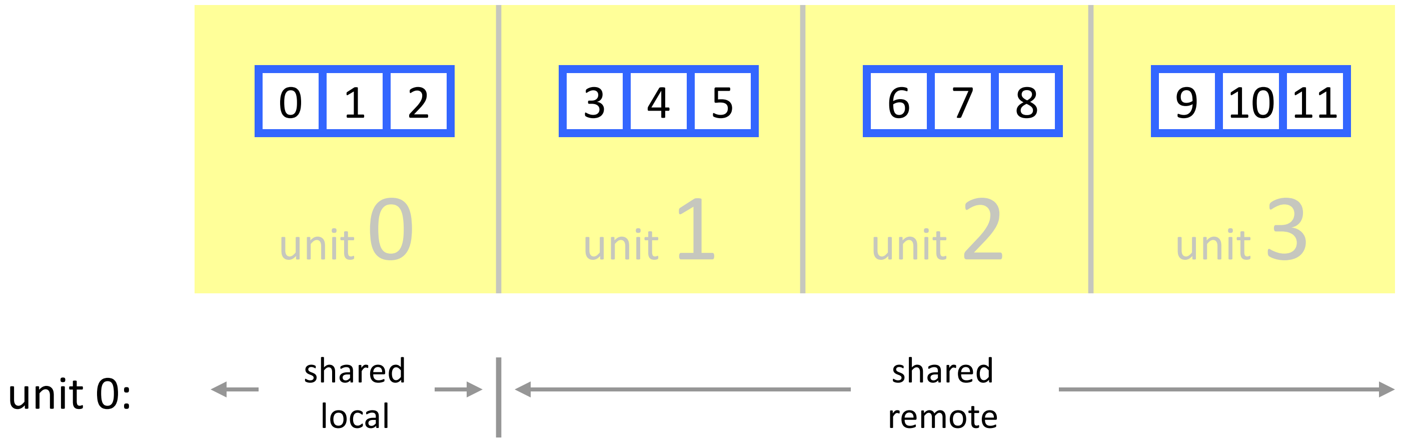
Shared data: managed by DASH in a virtual global address space

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

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

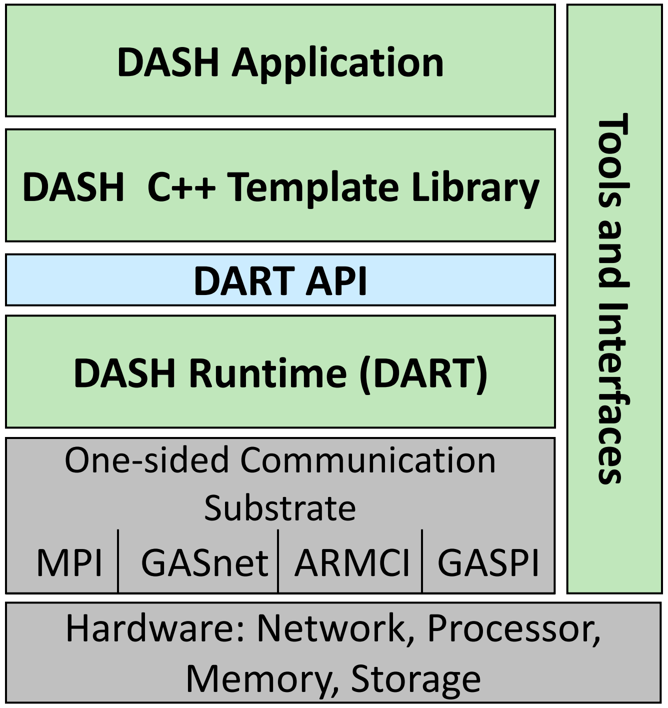
The “Partitioned“ in PGAS

- Example: dash::Array with 12 elements



- Data affinity
  - Elements can be accessed by any unit, but each one has an explicit and well defined home (owner) unit
  - Data locality important for performance
  - Support for the *owner computes* execution model

DASH Components


DASH Application
DASH C++ Template Library
DART API
DASH Runtime (DART)
One-sided Communication Substrate
MPI
GASnet
ARMCI
GASPI
Hardware: Network, Processor, Memory, Storage
Tools and Interfaces

DART: The DASH Runtime

- The DART Interface
  - Plain-C based interface (“dart.h”)
  - Follows the SPMD execution model
  - Provides global memory abstraction and global pointers
  - Defines one-sided access operations (puts and gets) and synchronization operations
- Several implementations
  - **DART-SHMEM**: shared-memory based implementation
  - **DART-CUDA:** supports GPUs, based on DART-SHMEM
  - **DART-GASPI**: Initial implementation using GASPI
  - **DART-MPI**: MPI-3 RMA based “workhorse” implementation

DASH: Hello World

```
#include <iostream>
#include <libdash.h>

using namespace std;

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

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

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

  dash::finalize();
}
```

Initialize the programming environment

Determine total number of units and our own unit ID

Print message. Note SPMD model, similar to MPI.

```
$ mpirun -n 4 ./hello
'Hello world' from unit 2 of 4 on nuc03 pid=30964
'Hello world' from unit 0 of 4 on nuc01 pid=25422
'Hello world' from unit 3 of 4 on nuc04 pid=32243
'Hello world' from unit 1 of 4 on nuc02 pid=26304
```