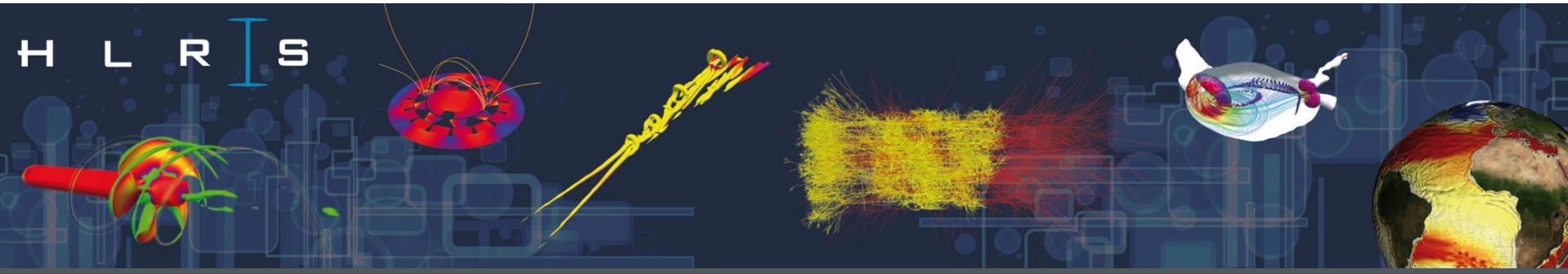
Overview of SmartDASH – Results and Perspectives

Presenter: José Gracia

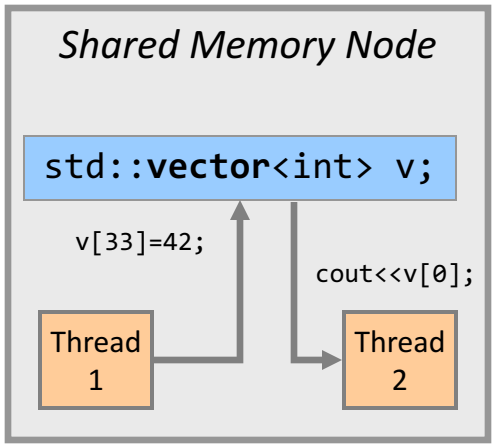
High Performance Computing Center Stuttgart (HLRS)

Roger Kowalewski (LMU), Tobias Fuchs (LMU),
Karl Fürlinger (LMU), Denis Hünich (TUD), Joseph Schuchart (HLRS),
Daniel Rubio Bonilla (IHR)

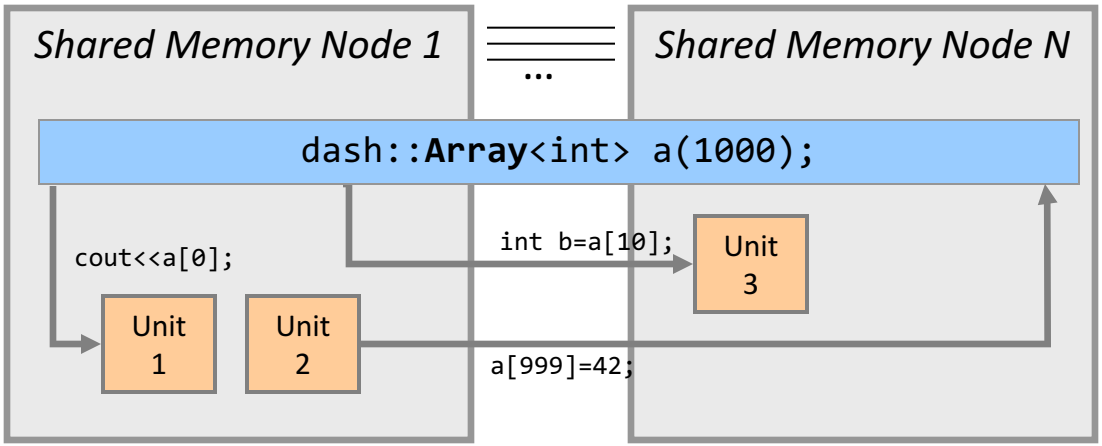


DASH - Overview

- DASH is a C++ template library, that offers
  - Distributed data structures, e.g., `dash::Array<int>`
  - Parallel algorithms, e.g., `dash::sort()`
- Generalizes shared memory programming to distributed memory systems:



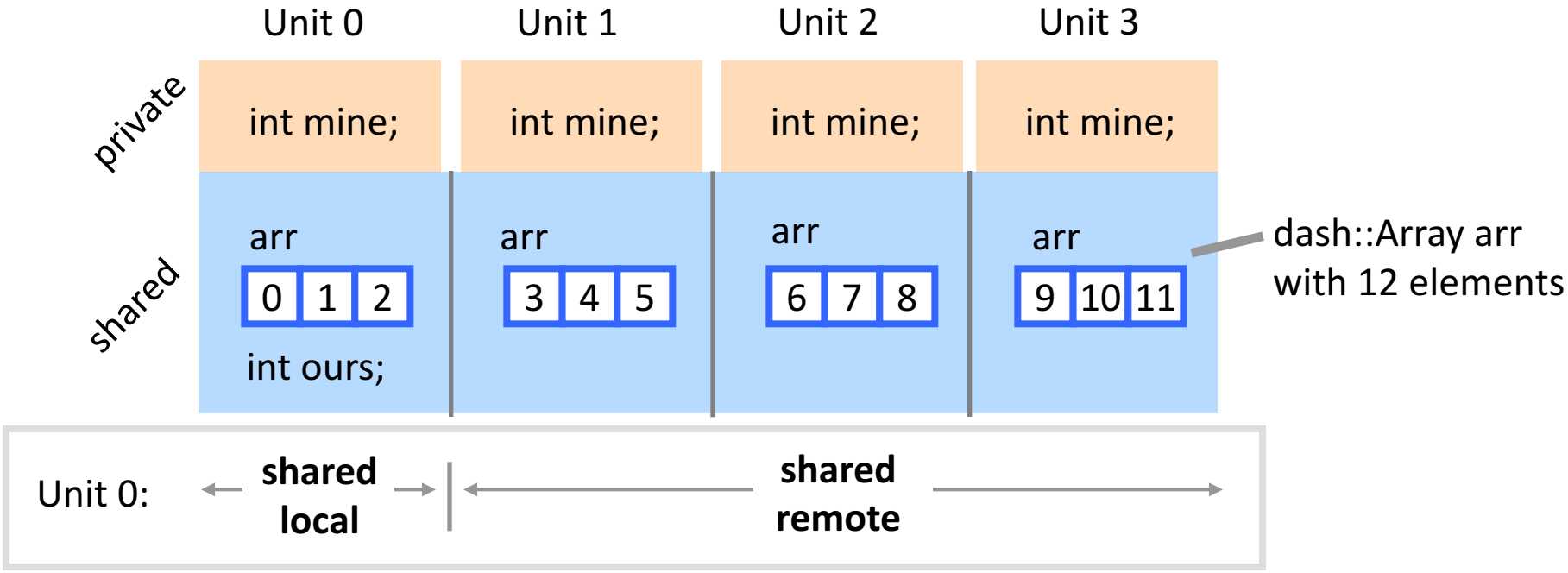
- Multiple threads access **physically** shared memory



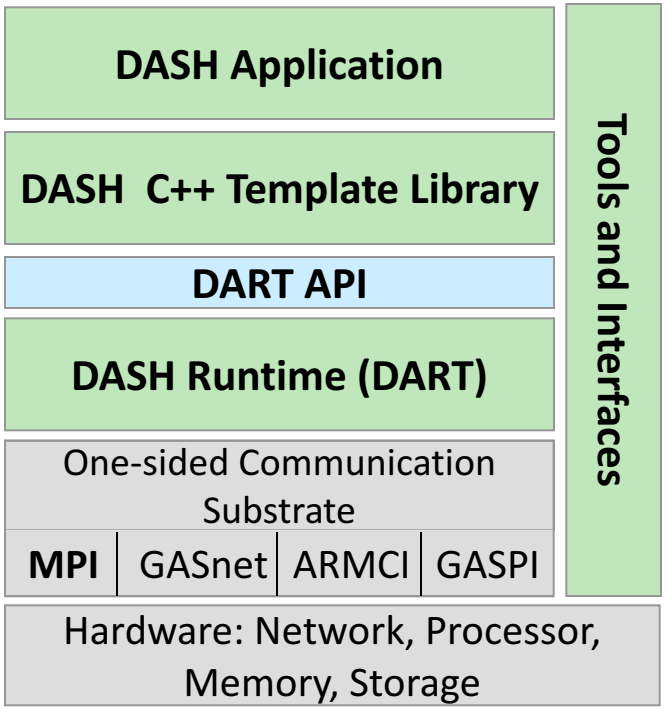
- Multiple nodes connected by a high-speed network
- Multiple threads (“units”) access **logically** shared memory

DASH - A PGAS Programming System

- DASH realizes a PGAS (Partitioned Global Address Space) abstraction
  - SPMD execution model, like MPI
  - Global address space: data accessible from everywhere
  - Partitioned: data distribution is **configurable** and **not hidden**



DASH – Project Overview



	Phase I (2013-2015)	Phase II (2016-2019)
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	DART runtime	DART runtime, Tasking
KIT Karlsruhe	Application case studies	
IHR Stuttgart		Smart deployment, Application case studies

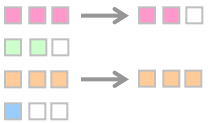
dash

www.dash-project.org

DFG SPPEXA

DASH is one of 16 SPPEXA projects

Global 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	
List$^{(*)}$<T>, **Map**$^{(*)}$<T>	Dynamic data structures (growing/shrinking)	

(*) Under Development

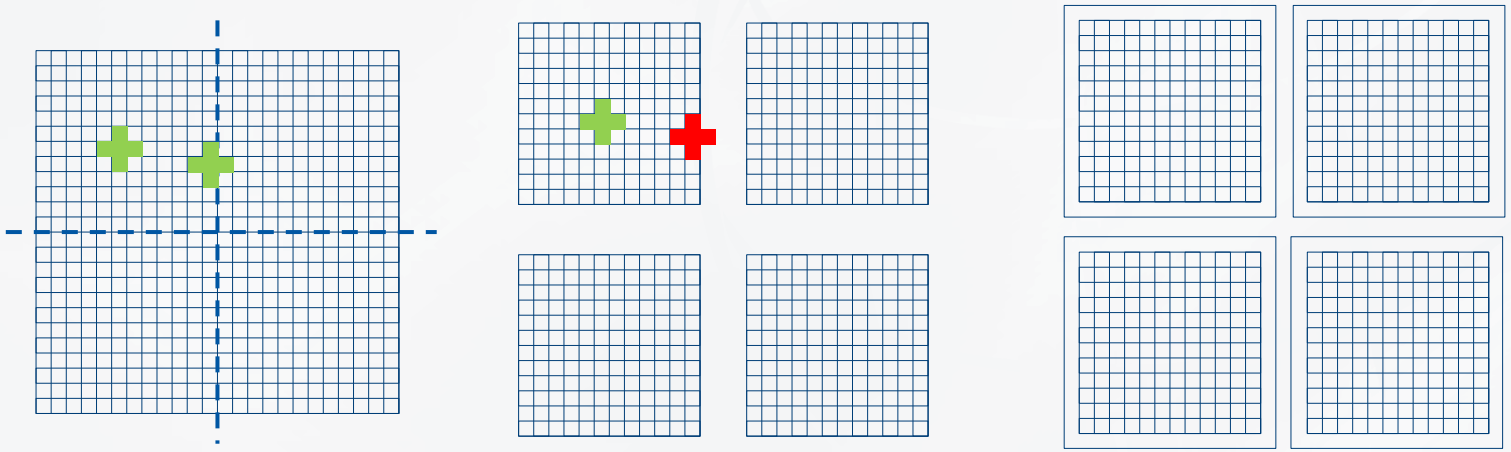
Recent activities

- Distributed tasking (talk by Joseph Schuchart)
- Halo / stencil wrapper for NArray
- Sparse matrix extension
- Partitioned sorting; Replication
- Graph extension; Dyloc
- Performance and productivity evaluation

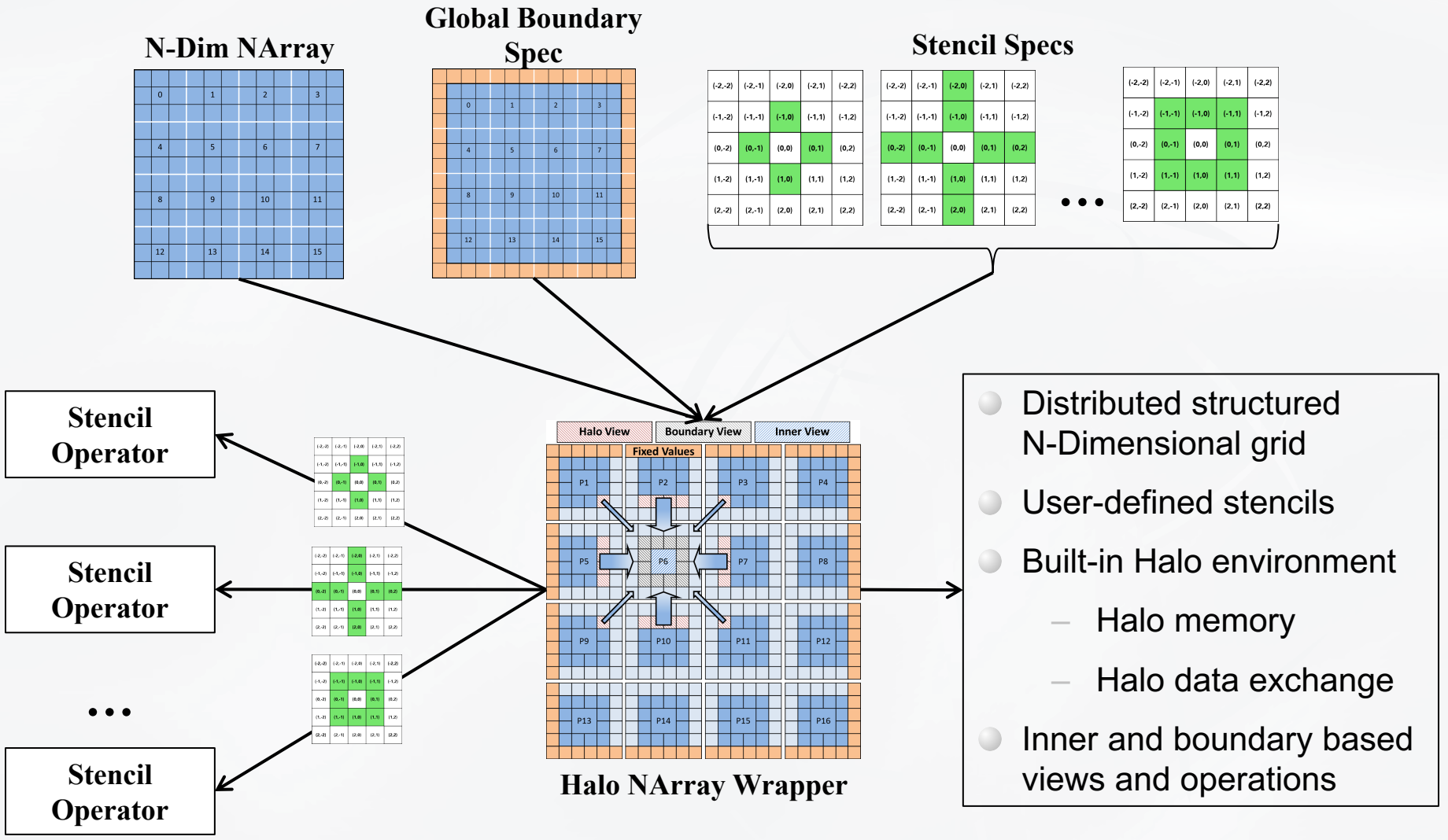
HALO / STENCIL WRAPPER FOR NARRAY

NArray with Support for Stencil and Halo Operation

- Index calculation is simple in the inner region but difficult across distribution borders
- Explicit data transfer for halo exchange
- Halo wrapper for NArray:
  - easy access to neighbor cells across distribution borders
  - asynchronous exchange of halos
  - halo determined user defined stencil operator



Halo NArray Wrapper - Architecture



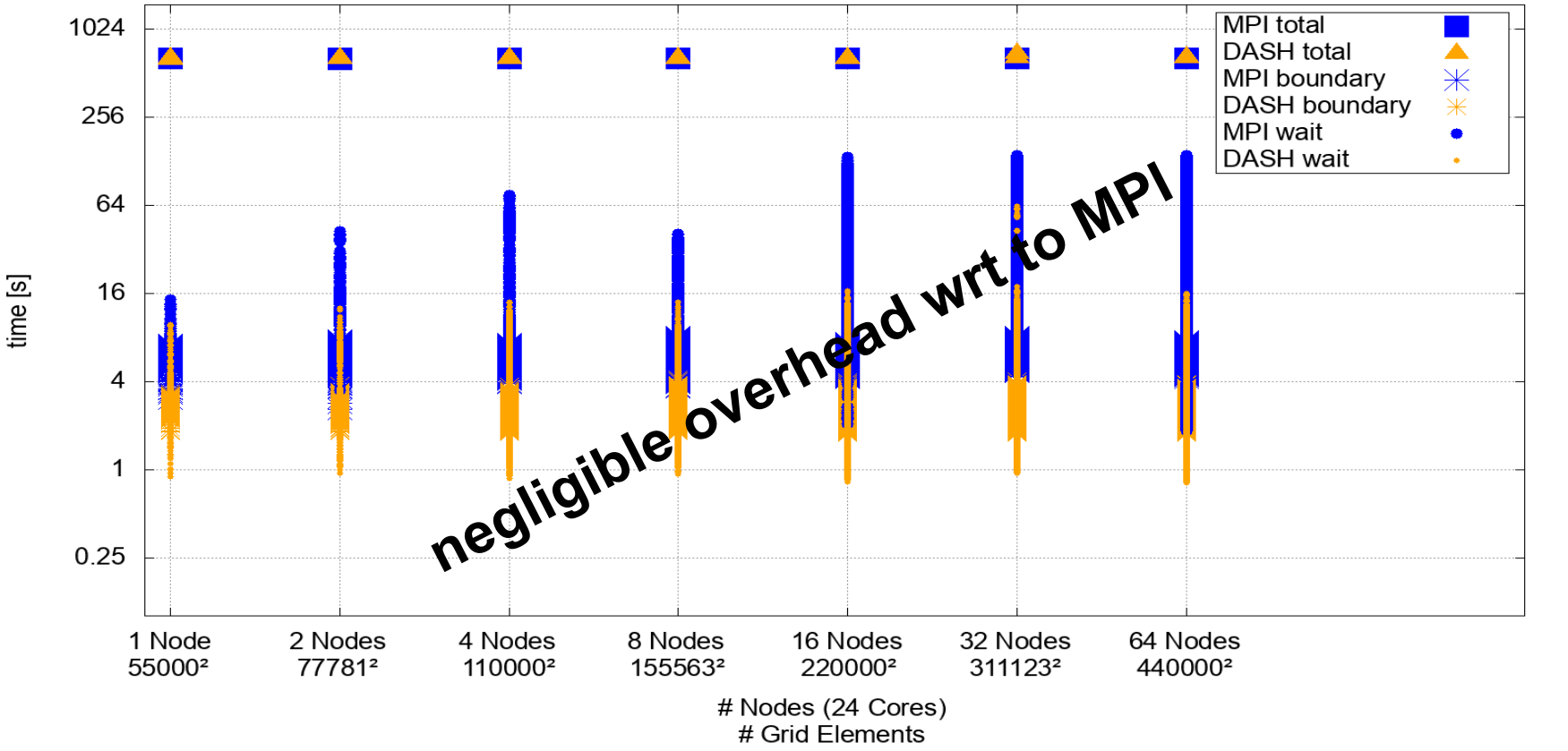
- Distributed structured N-Dimensional grid
- User-defined stencils
- Built-in Halo environment
  - Halo memory
  - Halo data exchange
- Inner and boundary based views and operations

Halo NArray Wrapper - Code Example

```
…
// Stencil points for North, South, West and East
StencilSpecT stencil_spec(StencilT(-1, 0), StencilT(1, 0),
                          StencilT( 0, -1), StencilT(0, 1));
// Periodic/cyclic global boundary values for both dimensions
GlobBoundSpecT bound_spec(dash::halo::BoundaryProp::CYCLIC,
                          dash::halo::BoundaryProp::CYCLIC);
// HaloWrapper for source and destination subgrids
HaloMatrixWrapperT src_halo(src_matrix, bound_spec, stencil_spec);
…
// Stencil specific operator for both subgrids
auto src_stencil_op = src_halo.stencil_operator(stencil_spec);
…
// Iteration loop
for (auto d = 0; d < iterations; ++d) {
…
  // start asynchronous Halo data exchange
  src_halo->update_async();
  // Calculation of all inner subgrid elements via inner stencil operator
  src_op->inner.update(…);
  // Wait until all Halo data exchanges are finished
  src_halo->wait();
  // Calculation of all boundary subgrid elements via stencil iterator
  auto it_bend = src_op->boundary.end();
  for (auto it = current_op->boundary.begin(); it != it_bend; ++it) {
    …
  }
…
}
```

Halo NArray Wrapper – Evaluation w/ 2D Heat-Equation

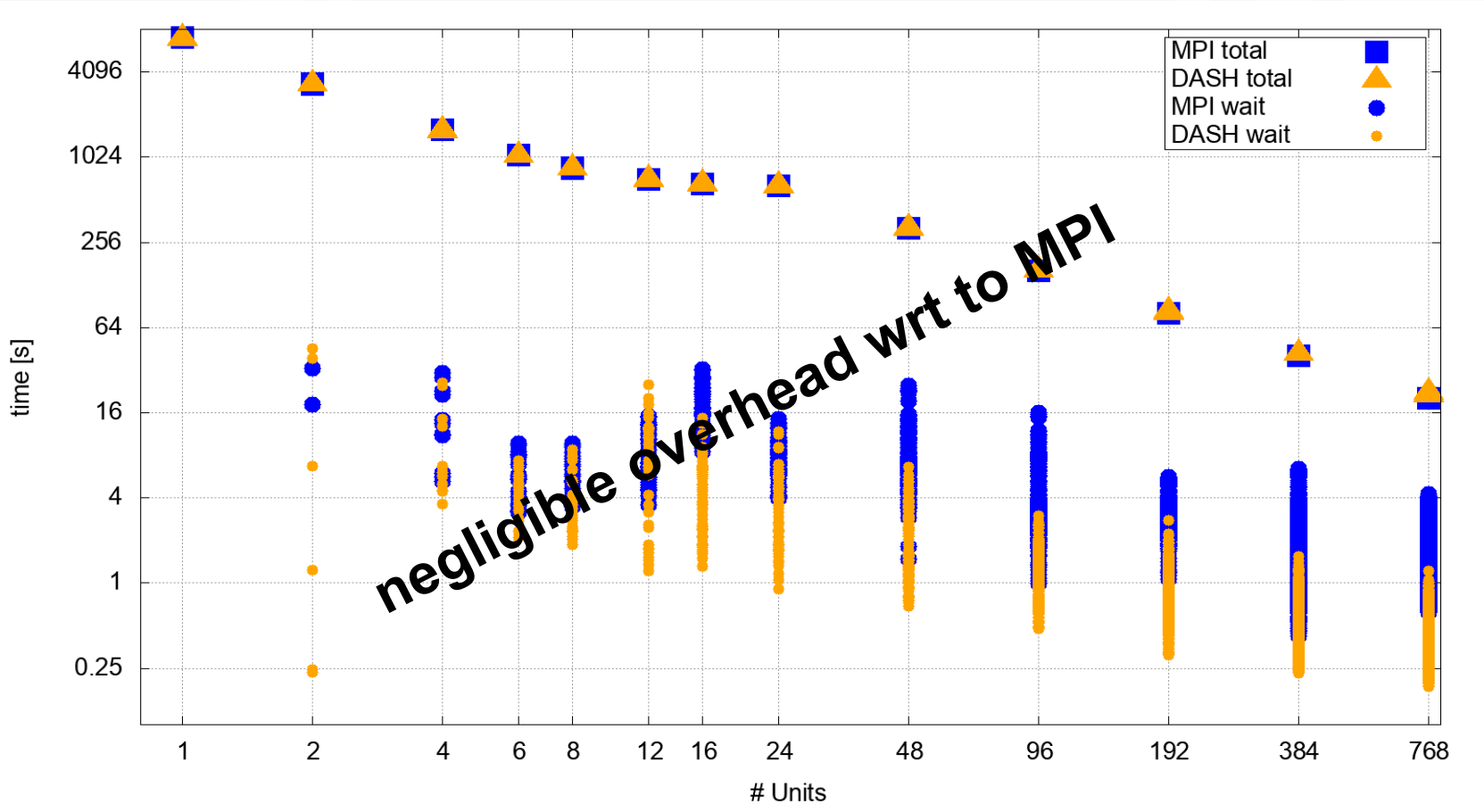
- gcc 7.1.0 and OpenMPI 3.0.0
- Each compute node has two Haswell E5-2680 v3 CPUs at 2.50GHz with 12 physical cores each and 64 GB memory



"A Halo Abstraction for Distributed N-Dimensional Structured Grids within the C++ PGAS Library DASH ", D. Hünich, A. Knüpfer; submitted to EuroPar 2019

Halo NArray Wrapper - Strong Scaling 2D Heat-Equation

- gcc 7.1.0 and OpenMPI 3.0.0
- Each compute node has two Haswell E5-2680 v3 CPUs at 2.50GHz with 12 physical cores each and 64 GB memory
- 55000² grid elements

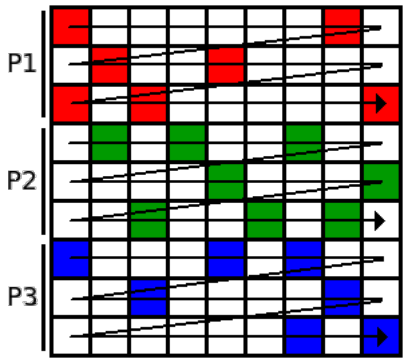


SPARSE MATRIX EXTENSION

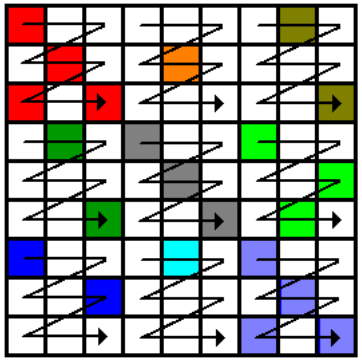
Sparse Matrix Extension

- Dash container for sparse matrices
- Based on CSR format
- Distribution of the Matrix can be 1D or 2D

- Offers methods for common matrix operations with:
  - sparse matrices,
  - dense matrices,
  - vectors
  - and scalars



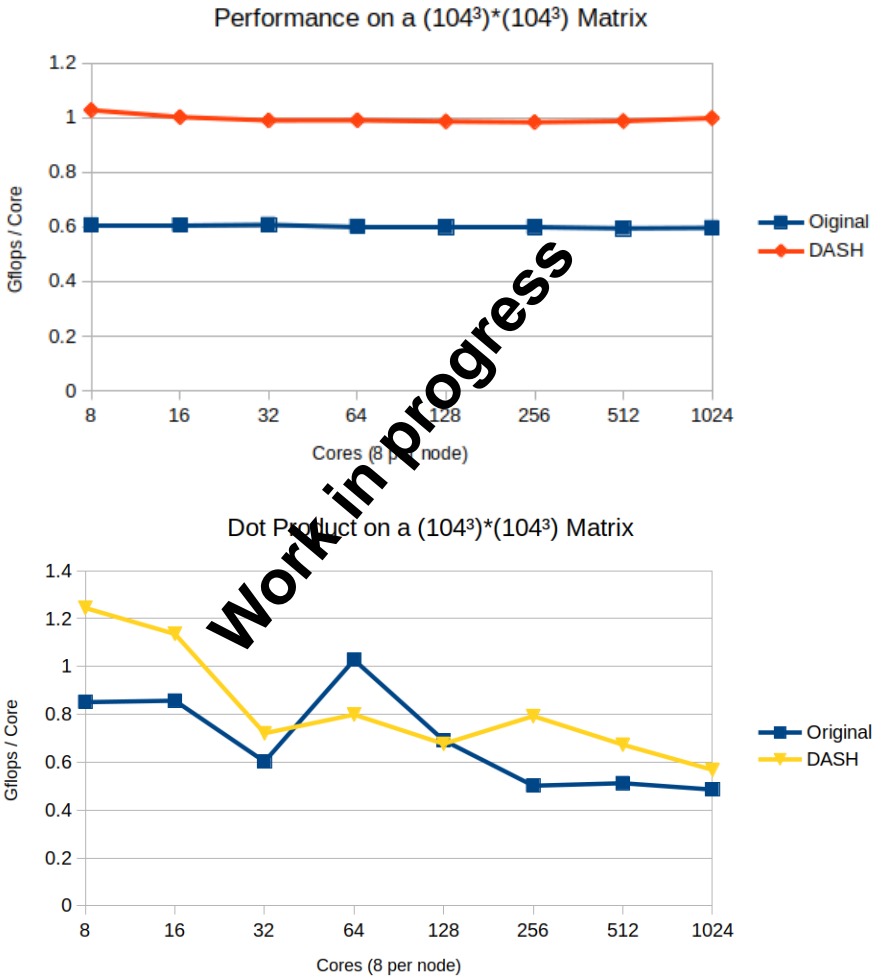
2D matrix with 1D partitioning



2D matrix with 1D partitioning

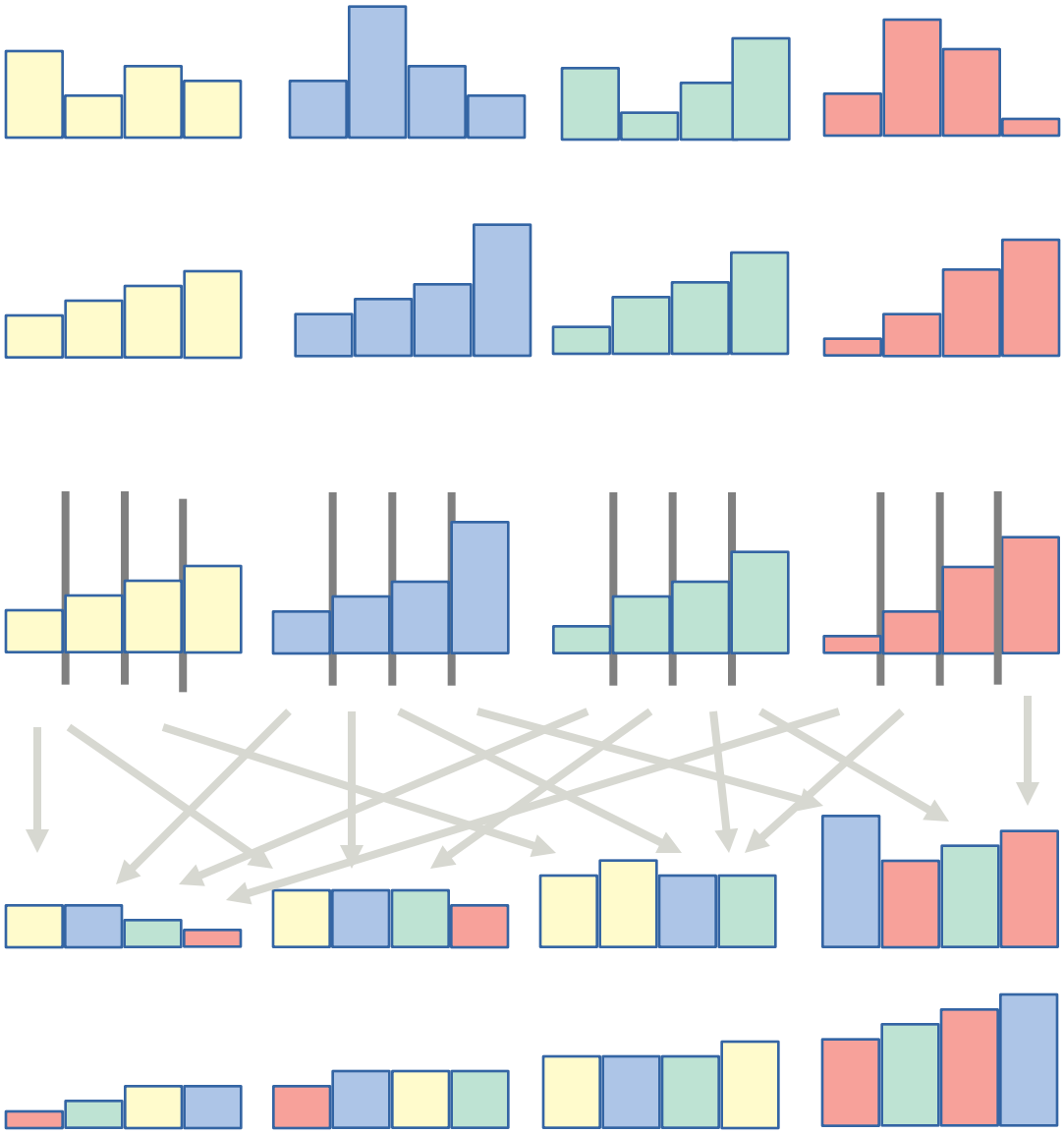
Sparse Matrix Extension – Evaluation with CG

- Conjugate gradient solvers usually employ sparse matrices
- Reference version uses MPI
- Main operations:
  - sparse matrix-vector mult,
  - global dot-product
- DASH-only implementation using sparse matrix extension
- Expects performance gain mainly due to:
  - less memory indirection
  - dot product with dash::reduce()



PARTITIONED SORTING

Partition-based Sort



0. Initial Data

1. Local Sort

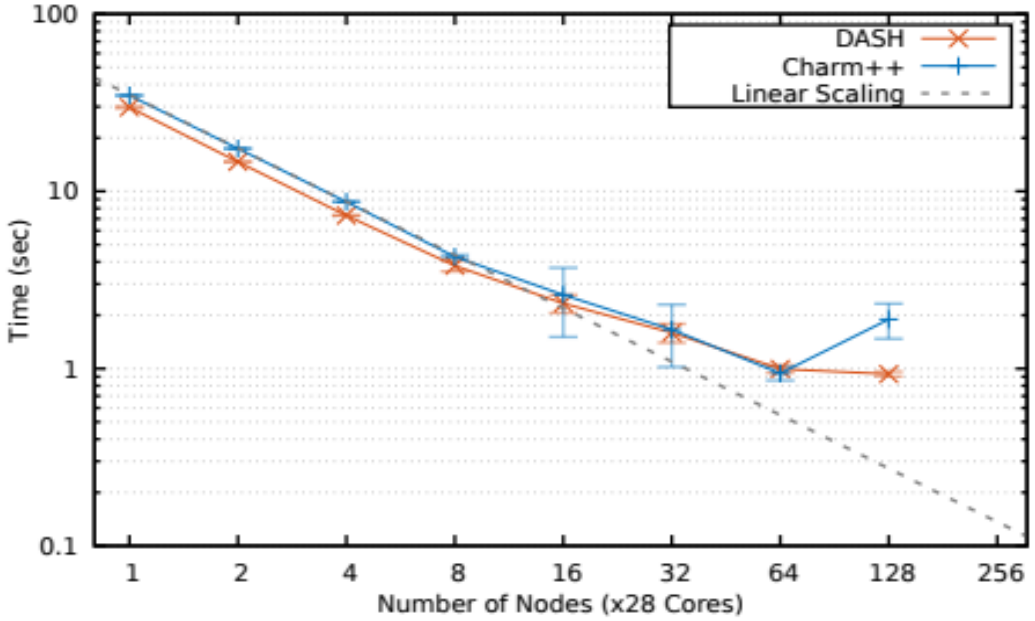
2. Determine Partitions

3. All-to-all

4. Merge Sort

Partition-based Sort – Distributed Memory

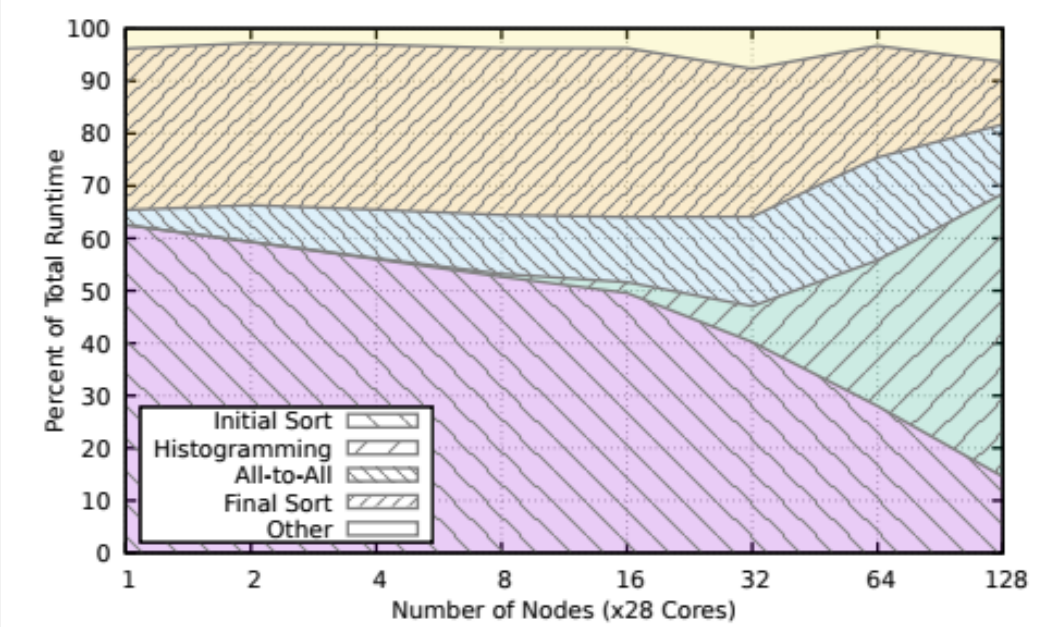
- Competitors:
  - HykSort (SC2013), segfaults for >4GB
  - Charm++, number of elements must be 2^n



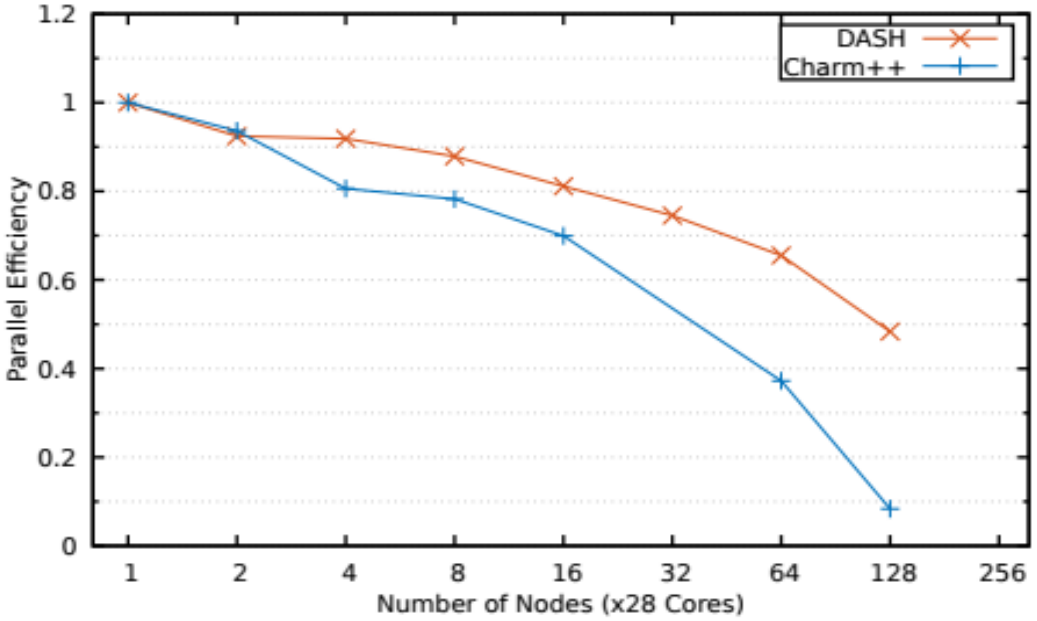
- Strong scaling on SuperMUC
- sorting ~16GB DP numbers

- PGAS approach beneficial because data locality is primary concern
- Partition-based sort algorithm moves data only once

Strong Scaling Area Plot



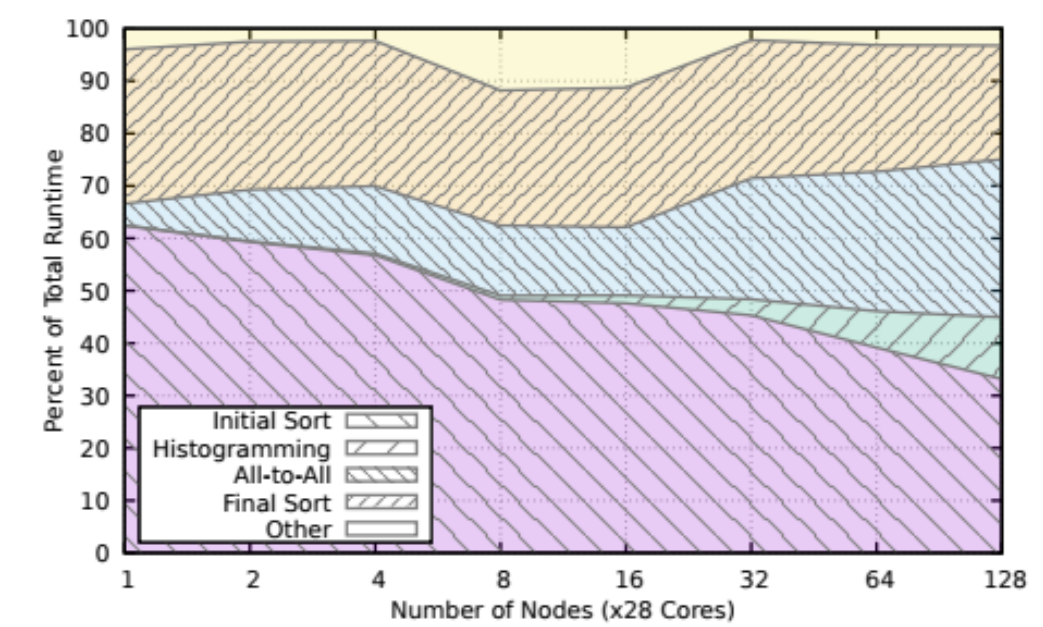
Partition-based Sort – Distributed Memory



- Weak scaling on SuperMUC
- sorting 4GB DP numbers per core

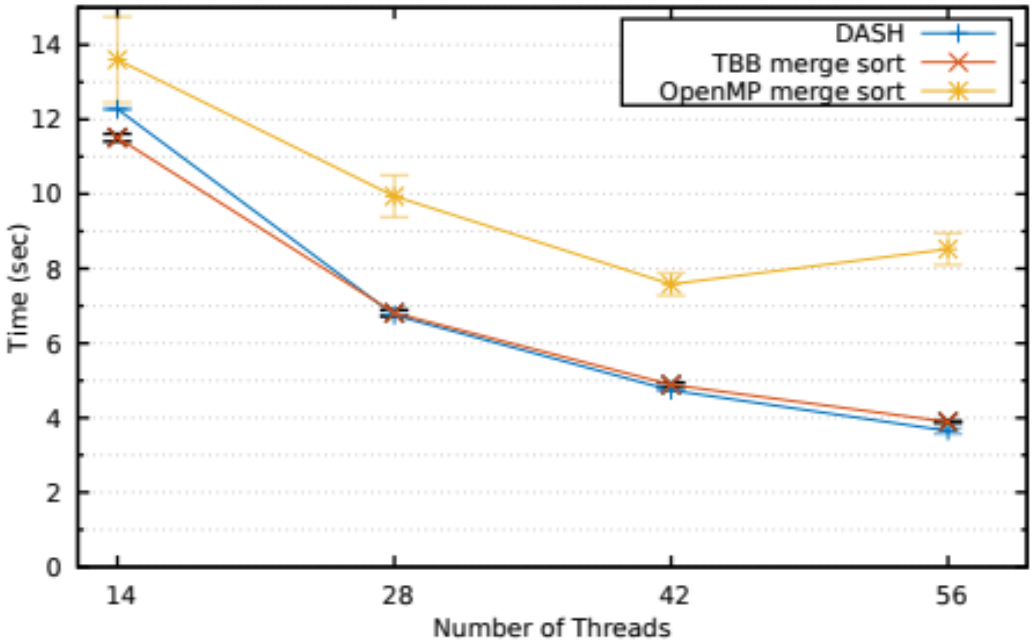
- PGAS approach beneficial because data locality is primary concern
- Partition-based sort algorithm moves data only once

Weak Scaling Area Plot



Partition-based Sort – Shared Memory

- Competitors:
  - TBB
  - OpenMP



- PGAS approach beneficial because data locality is primary concern
- Partition-based sort algorithm moves data only once

- 2 sockets, 28 cores, 4 NUMA domains
- sorting ~5GB of integer numbers

PERFORMANCE AND PRODUCTIVITY EVALUATION

Case Study: The Cowichan Problem Suite

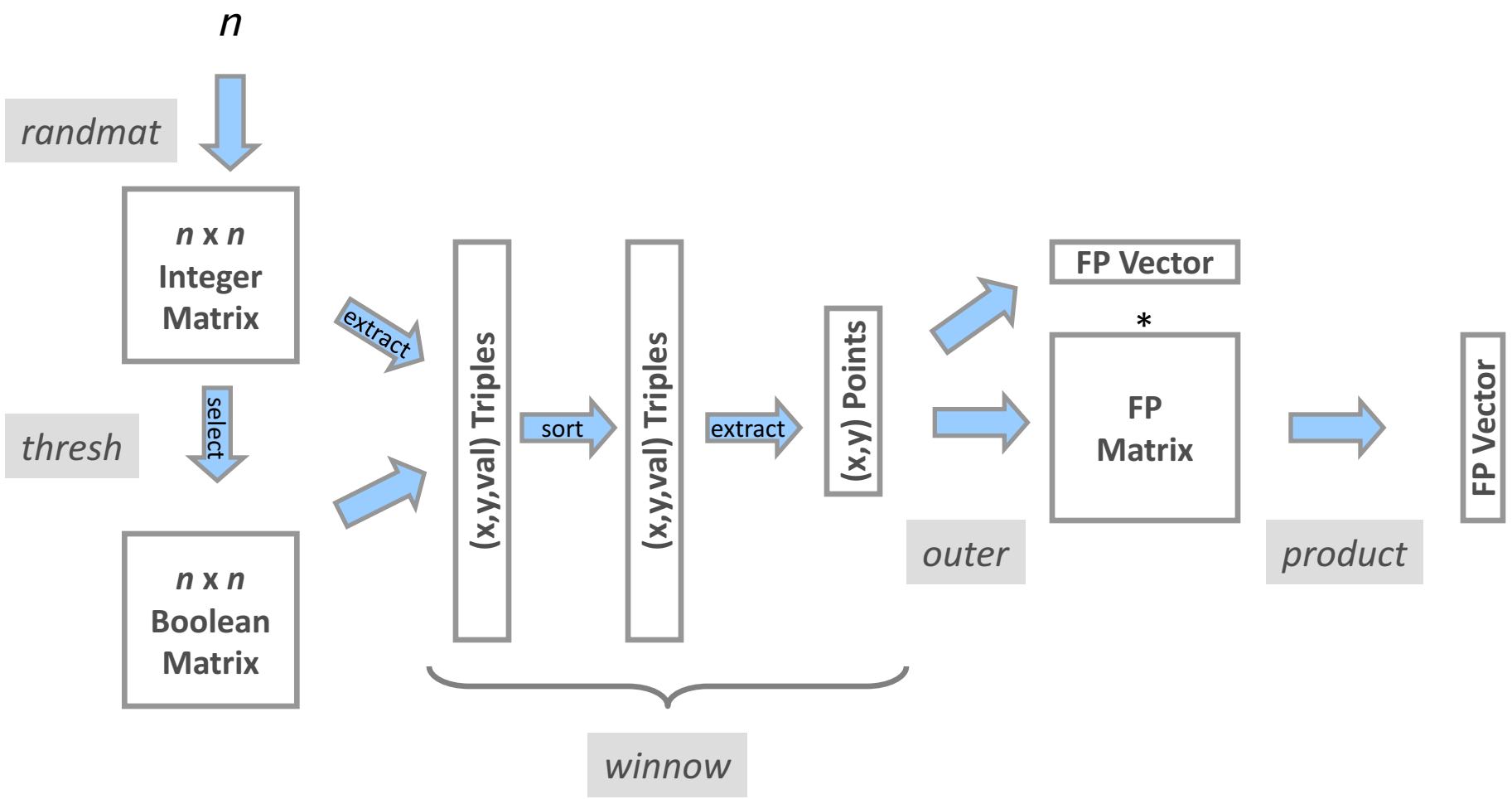
- Cowichan problems
  - A **benchmark suite** designed to investigate the usability of parallel programming systems (1990s)
  - 13 “toy” problems, quick implementation, composable by chaining [1]

- Previous work by Nanz et al. [2] selected **five benchmarks** to evaluate the usability of multicore languages
  - Four programming systems compared:
    - **Go**, **Cilk**, **TBB**, **Chapel**
  - Metrics:
    - **Usability**: LOC, development time
    - **Performance**: execution time and scalability

[1] Wilson, Gregory V., and R. Bruce Irvin. *“Assessing and comparing the usability of parallel programming systems.”* University of Toronto. Computer Systems Research Institute, 1995.

[2] Nanz, Sebastian, Scott West, Kaue Soares Da Silveira, and Bertrand Meyer. *"Benchmarking usability and performance of multicore languages."* In Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on, pp. 183-192. IEEE, 2013.

Case Study: Data Structures and Algorithms



Cowichan Results – Lines of Code

	DASH	go	Chapel	TBB	Cilk
randmat	16	29	14	15	12
thresh	31	63	30	56	52
winnow	53	94	31	74	78
outer	23	38	15	19	15
product	20	27	11	14	10

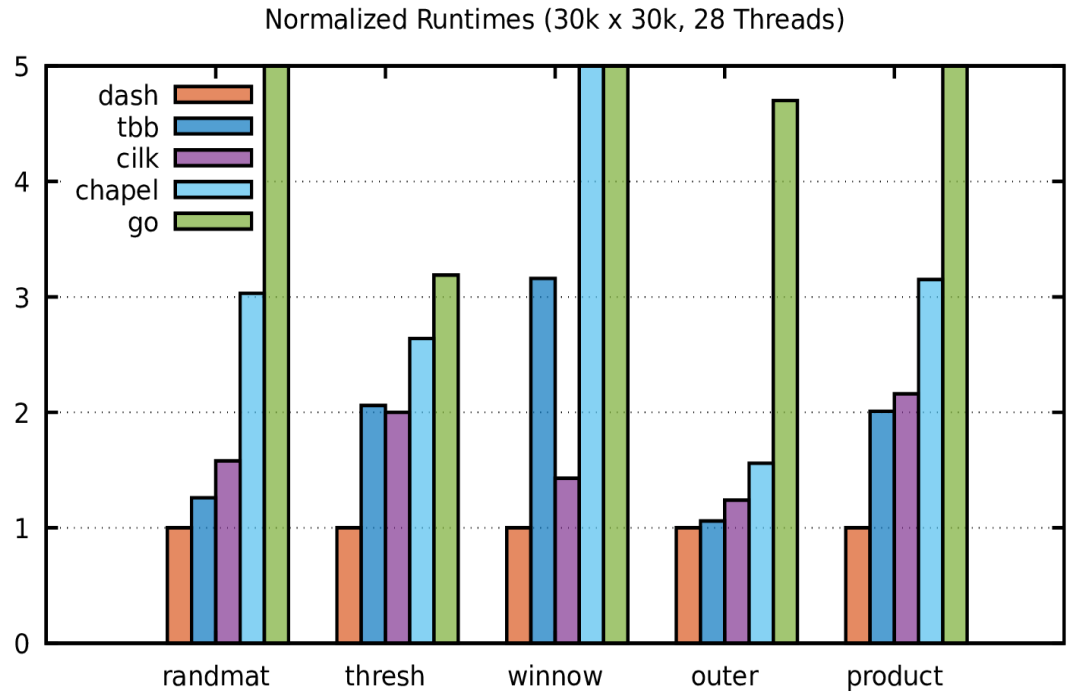
- DASH is not the most concise approach, but not much worse than the best solution
  - DASH is the only case where the same code can be run on shared memory and distributed memory systems!

"Investigating the Performance and Productivity of DASH Using the Cowichan Problems", K. Fürlinger, R. Kowalewski, T. Fuchs, and B. Lehmann; Proc. of the International Conference on High Performance Computing in Asia-Pacific Region, Tokyo Jan. 2018

Cowichan Results – Shared Memory (1)

- Platform: **Single node** of SuperMUC Phase 2 (Haswell)
  - Haswell Xeon E5-2697, 2.6 GHz, 28 cores per node, 64 GB mem
  - 30k x 30k matrix
  - Intel Compiler (icc) v. 18.0.2 used for all programming systems

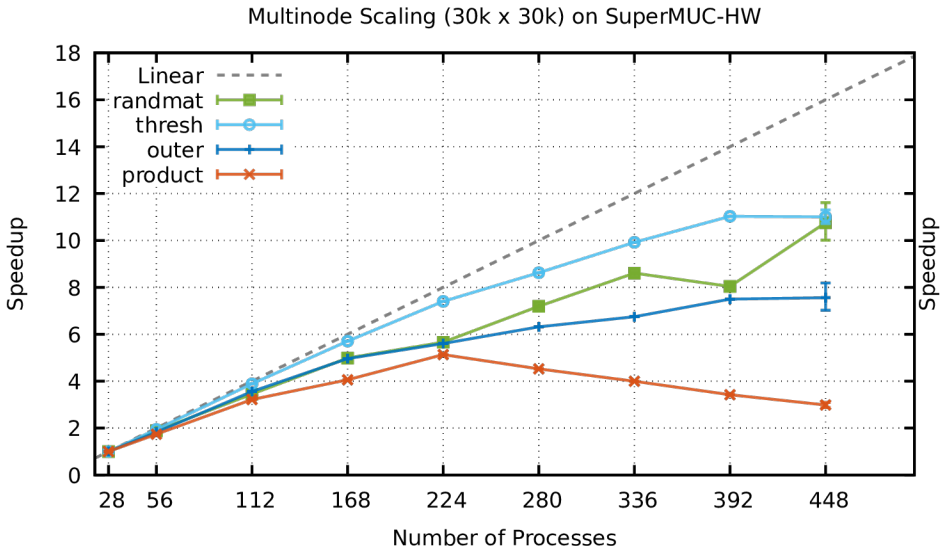
Absolute performance, using all 28 cores, 30k x 30k Matrix



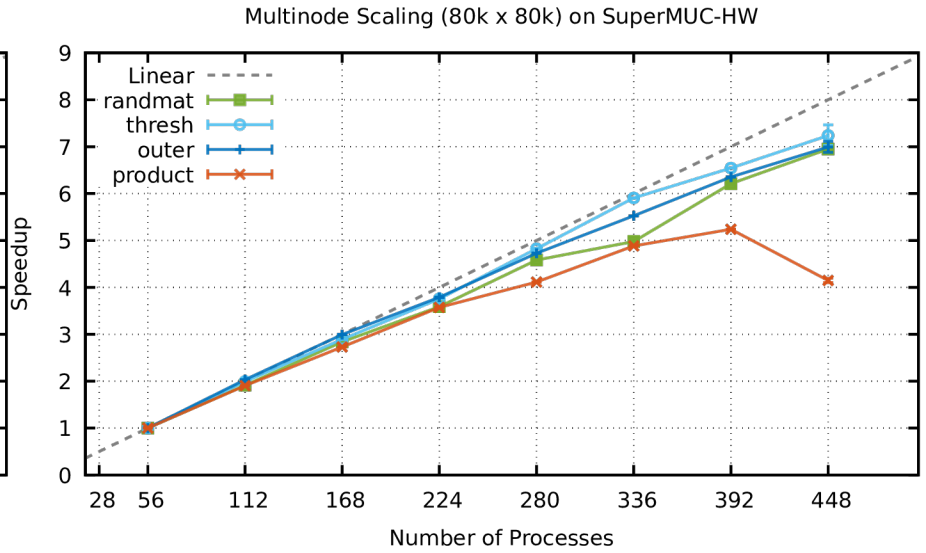
Performance relative to DASH, using all 28 cores, 30k x 30k Matrix

Cowichan Multinode Scaling and Summary

- Platform: **Up to 16 nodes** of SuperMUC
  - Haswell Xeon E5-2697, 2.6 GHz, 28 cores per node
  - 64 GB of main memory
  - DASH is the **only** approach that can also use distributed memory machines (the **same source code**)



30k x 30k, Speedup vs 1 node



80k x 80k, Speedup vs 2 nodes

OUTLOOK & CONCLUSION

Future activities

- Tools interface: OMPT-like
  - performance tools: Scalasca, Vampir, Paraver
  - task graph visualisation / debugging
- Evaluation with graph algorithms
  - complex communication pattern challenging for MPI
- Execution spaces & memory spaces:
  - execute on accelerators
  - access memory on accelerators, NVRAM, etc
- Load balancing
  - requires data-migration and/or task-migration

Conclusions

- DASH has developed into a mature framework for a wide variety of HPC workloads
- it addresses main performance challenges such as: data-locality, multi-level parallelism, overlap of communication and computation, global synchronisation
- performance comparable with established solutions
- incremental porting of C++ code; STL conformity
- interoperable with MPI and OpenMP
- **DASH v0.3 since SC18, DASH v0.4 in Q2/Q3 2019**

Acknowledgements

- Funding

DFG Deutsche Forschungsgemeinschaft

SPPEXA

- The DASH team

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

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

Thank you for your attention