LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Utilizing Heterogeneous Memory Hierarchies in PGAS

Roger Kowalewski, Tobias Fuchs, Karl Fürlinger

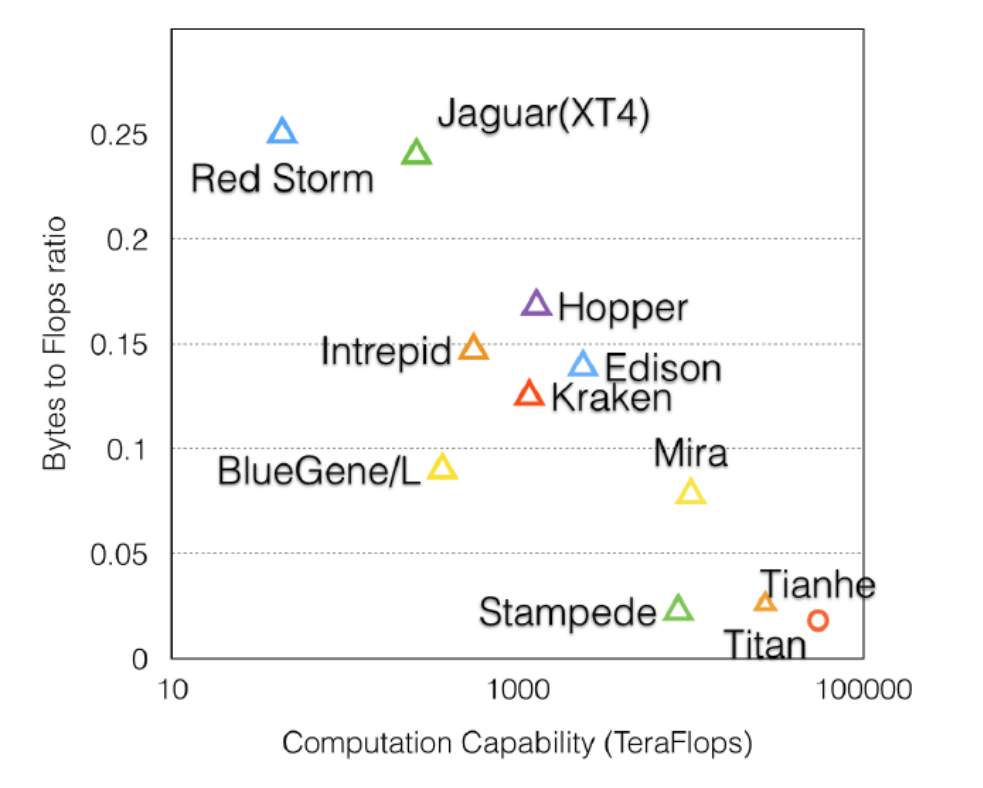
kowalewski@nm.ifi.lmu.de

Ludwig-Maximilians Universität (LMU) Munich

DASH Project
www.dash-project.org

Architectural Trends in HPC Systems

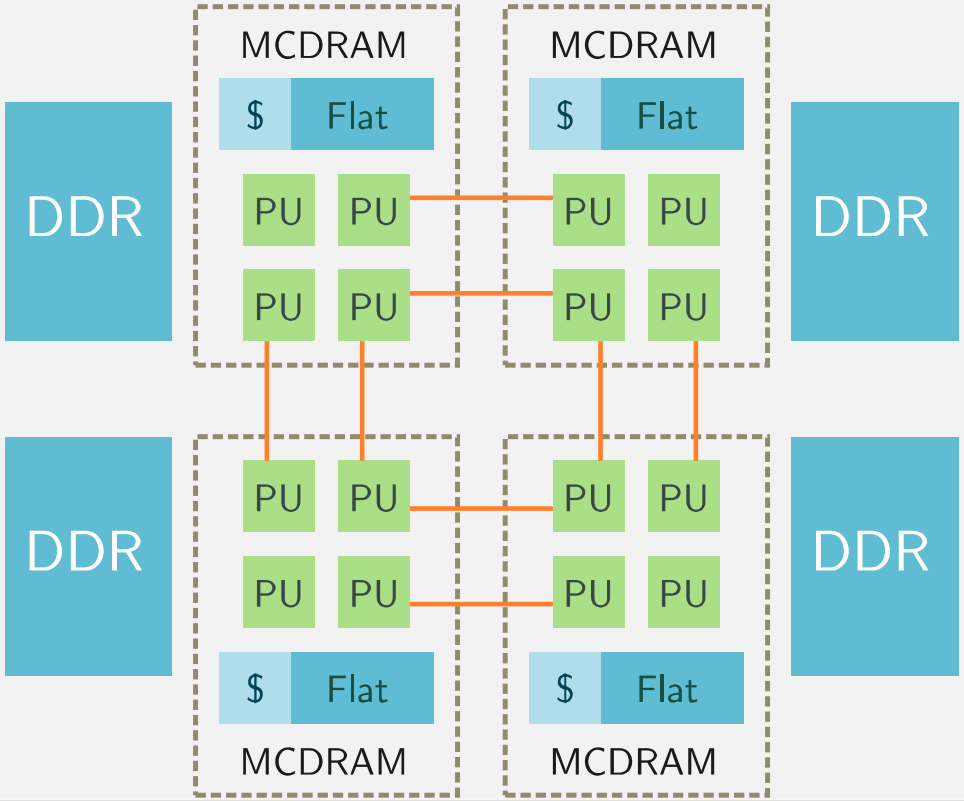
– Capacity and bandwidth scale **significantly** slower than computation



Ni, Xiang, et al. "Runtime Techniques for Programming with Fast and Slow Memory." *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017.

Implications for Modern HPC Architectures

- Emerging Systems feature **heterogeneous** multi-tiered memory spaces
  - Different characteristics (latency, bandwidth) and capabilities (non-volatility, scratch spaces, …)
- Example: Intel Xeon Phi Knights Landing
  - Discussed throughout this paper

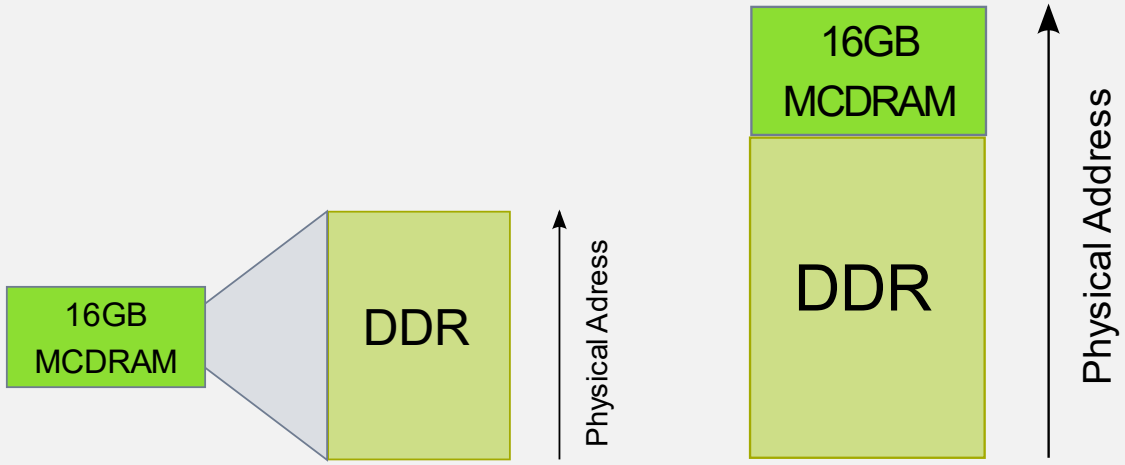


Algorithmic Requirements

- Regular vs. Random Access
- Latency limited / Bandwidth limited
- Temporal scope of data
  - Application life time
  - Scratch Space
- Applied Operations
  - Read-Only
  - Write-Only
  - Read / Write
- Functionality
  - Replication / Migration
  - Checkpoint Restart mechanisms (non-volatile memory)

Managing Heterogeneous Memory

- Transparently managed through hardware or operating system (Cache Mode)
- Software managed “flat” memory model (Flat Mode)



KNL Cache Mode KNL Flat Mode

→ Scientists need **productive** programming abstractions.

DASH Memory Allocation Model

- **A memory space** abstracts a specific memory resource (e.g., Intel KNL HBW).
  - Unified Allocation / Deallocation Interface
  - Consistency Semantics
  - Memory Traits as an interface to specify and query characteristics and properties
- Multiple Memory Spaces may be constructed from the same memory resource.
- A **memory allocator** provides specific allocation strategies.
  - Organizes memory from the underlying memory space.
  - Error and Fallback strategies.
  - Behavior may be specific for certain use cases.

DASH Memory Traits

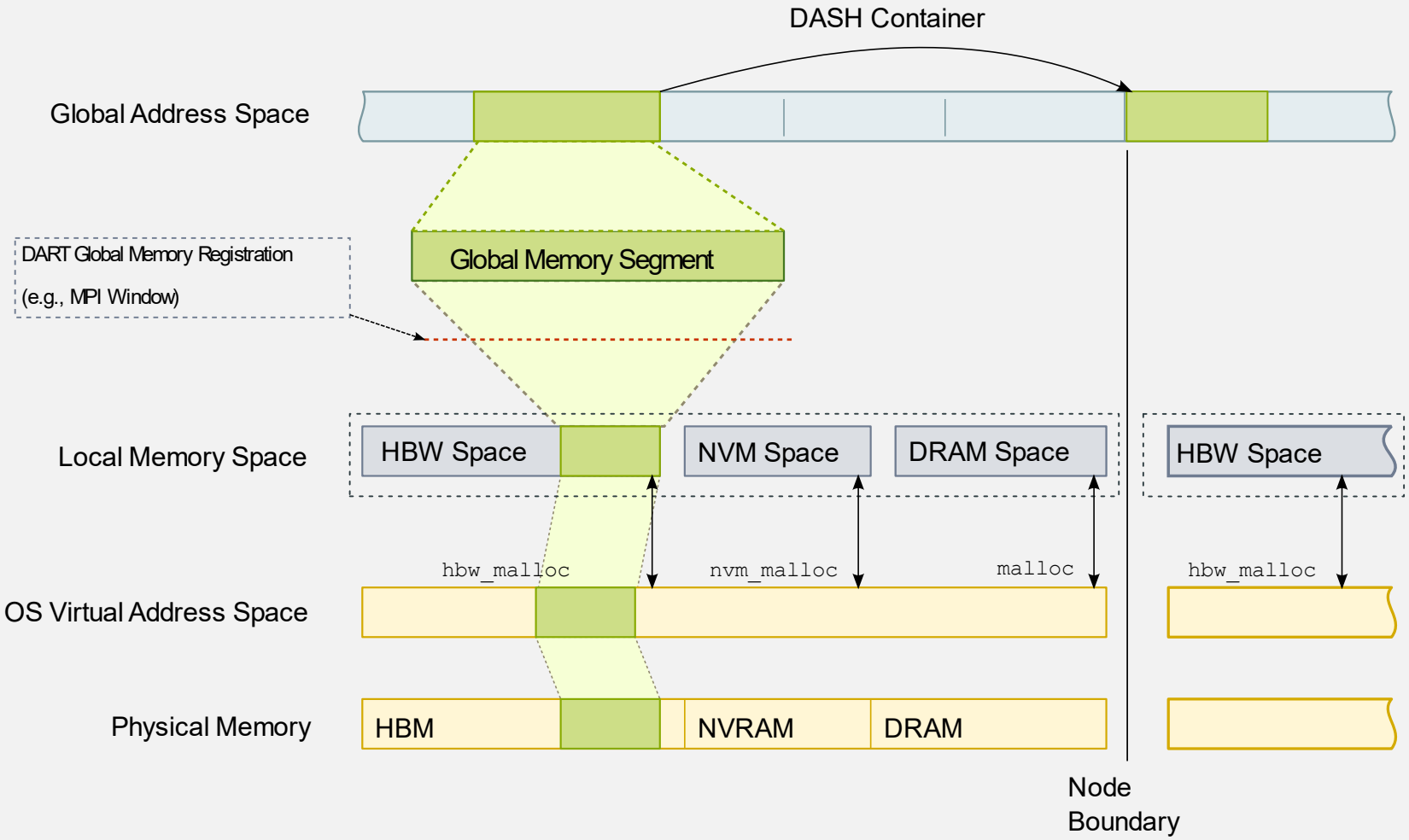
- Prescriptive (required) Traits
  - Domain: Global, Local
  - Location: Host, Device, etc
  - Permissions: Read, Write, Read / Write
- Descriptive (requested) Traits
  - Available Capacity
  - Distance: Near, far
  - Optimization Parameter: Latency, Bandwidth, Capacity
  - Page size

DASH Memory Allocator Traits

Allocation strategies are configurable through allocator traits.

- Memory Domain
  - Global (distributed)
  - Local Memory
- Thread Safety
- Fallback Policy
- Memory Alignment
- Pinning

DASH Memory Organization



Memory Spaces and DASH Containers

- DASH provides a set of Memory Spaces
  - Default Space (usually DRAM)
  - HBW Space (Intel KNL)
  - NVRAM (will be available in future)
  - Device Memory (will be available in future)
- … along with a set of memory allocators
  - Both `global` and `local`
  - Default strategies for various DASH containers
- Integrated as template parameter into the DASH Container Concept
- `Local` allocators to the C++11 allocator concept

```
template<typename Domain,
         typename MSpaceCategory>
class MemorySpace {
  public:
   //void_pointer is obtained by dash::Memory_Traits
   void_pointer allocate(size_t nbytes,
                         size_t alignment);
   void   deallocate(void_pointer p, size_t nbytes);
   bool   is_equal(const MemorySpace & other);
   size_t available();
   size_t capacity();
   bool   flush(void_pointer p, size_t bytes);
}
```

Listing 1: Memory Spaces interface

Integration into DASH Containers


```
1  // type alias for an array in default memory space
2  using array_t = dash::Array<int>;
3  // type alias for an array in HBW memory space
4  using array_hbw_t = dash::Array<int,
5                                  dash::mspace_hbw_tag>;
6
7  int main(int argc , char * argv []) {
8    dash::init(&argc, &argv)
9    size_t  global_capacity = ...;
10   array_t arr_dram(global_capacity);
11   dash::fill(arr_dram.begin(), arr_dram.end(), ...);
12   // Initial processing in DRAM...
13
14   // Migrate data from DRAM to HBM
15   array_hbw_t arr_hbw(array_dram.size());
16   dash::copy(arr_dram.begin(), arr_dram.end(),
17                arr_hbw.begin());
18   // Further processing in HBM...
19   dash::finalize();
20   return 0;
21 }
```

Listing 2: Example with DASH Memory Spaces

Case Study: Cowichan Problems on Intel KNL

- Use Case: Cowichan Problems
  - Set of small kernels to assess the usability of parallel programming languages
  - Basic data structures: 1D and 2D Arrays
  - Algorithms
    - Matrix Multiply
    - Parallel Sort
    - Prefix-Sum
- Platform: Intel Xeon Phi 7210 (KNL Family), LRZ Test System
  - Only 1 Node, 64 Cores
  - 384 GB DRAM
  - 16 GB MCDRAM (400 GB/s memory bandwidth)
- First experiences with DASH memory spaces
- Evaluating the impact of Intel KNL MCDRAM

First Results (1)

- Matrix Size: 20k x 20k
- Data set fits completely into High Bandwidth Memory

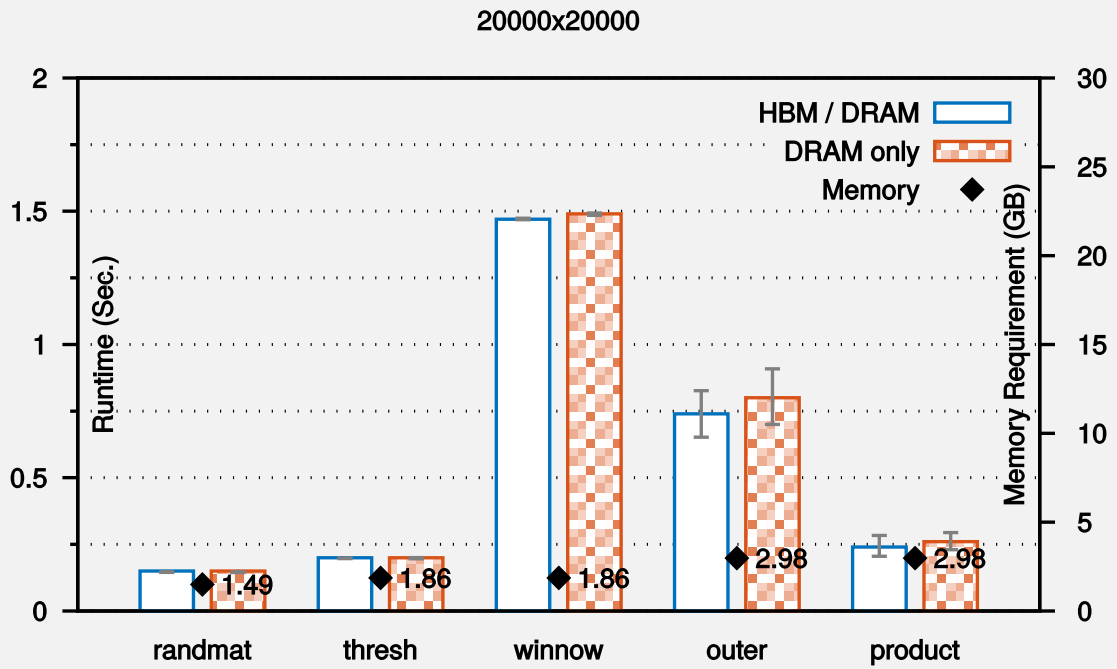


Figure 1a: Static Memory Allocation

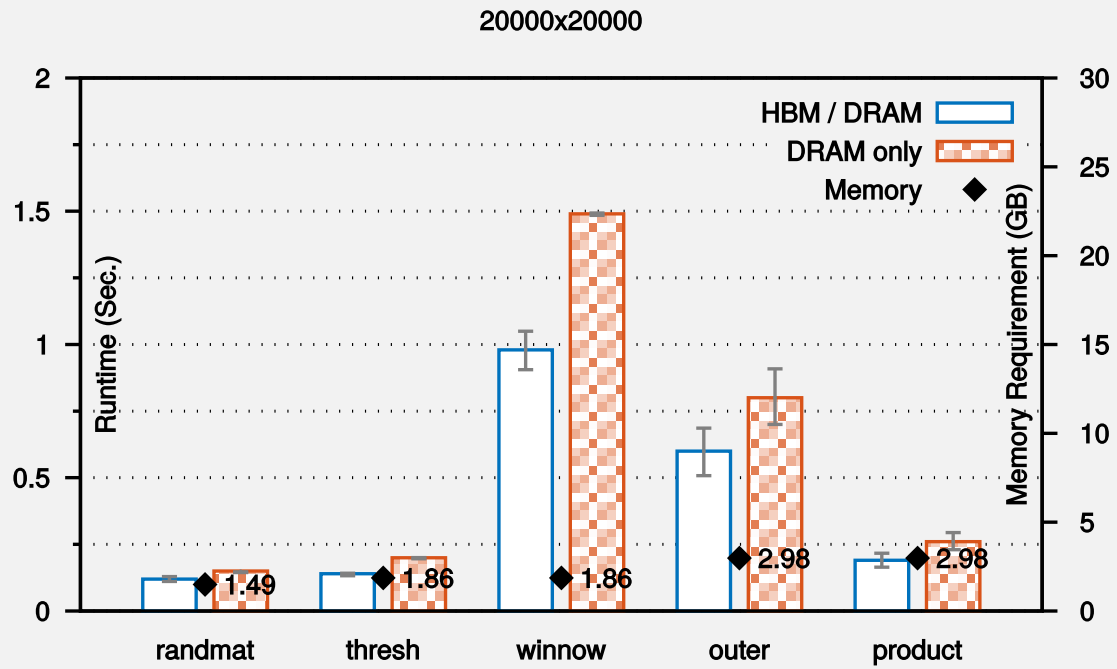


Figure 1b: Strategic Allocation in HBM / DRAM

First Results (2)

- Matrix Size: 50k x 50k
- Data set does not fit into High Bandwidth Memory

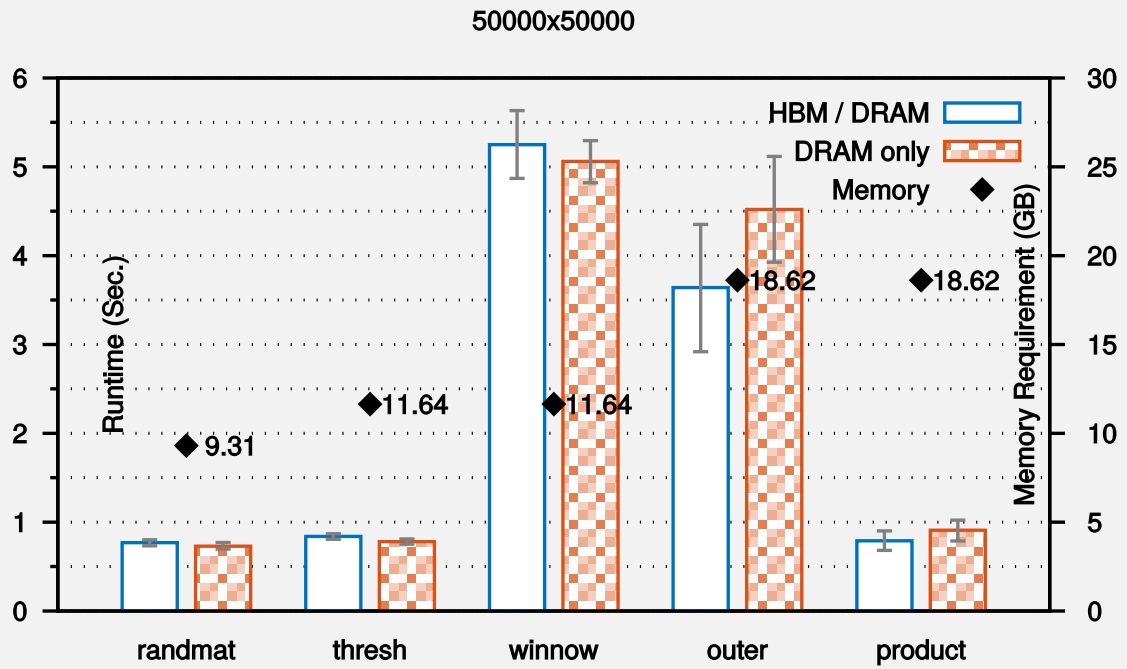


Figure 2a: Static Memory Allocation

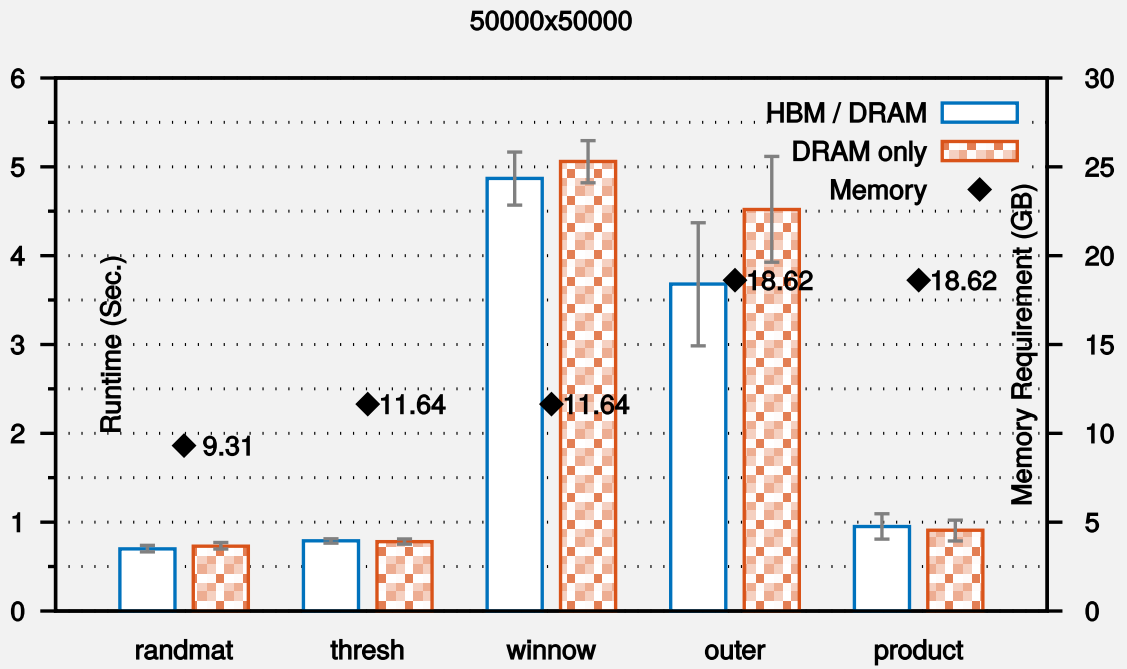


Figure 2b: Strategic Allocation in HBM / DRAM

Future Work

- More detailed evaluation to study the impact of Intel‘s MCDRAM
  - Scaling to multiple nodes
  - More „classic“ HPC applications (e.g. ‚stencils, iterative solvers, etc.)

- More advanced Mechanisms for Replication (Mirroring), considering
  - Resource-aware parameters like capacity
  - Depending on the algorithmic requirements
  - Similar to concepts which can be found in other programming languages as well

Related Work

- C++17 Memory Technical Specification
- OpenMP Technical Report: Memory Management (v5.0)
- Kokkos Memory Spaces
- Other PGAS approaches try to support heterogeneous memory spaces as well.

Acknowledgements

dash-project.org
github.com/dash-project
dash-project.slack.com

Funding

DFG Deutsche Forschungsgemeinschaft

Bundesministerium für Bildung und Forschung

Team

T. Fuchs [LMU], **R. Kowalewski** [LMU], **J. Schuchart** [HLRS],
D. Hünich [TUD], A. Knüpfer [TUD], **J. Gracia** [HLRS], C. Glass [HLRS],
H. Zhou [HLRS], K. Idrees [HLRS], F. Mößbauer [LMU],
K. Fürlinger [LMU]