Exploiting Heterogeneous Shared Memory Architectures

Christian Terboven <terboven@itc.rwth-aachen.de>
with the H2M team @ INRIA Bordeaux and @ RWTH

September 13th, 2022
PPAM‘22

Inria | NHR4CES | IT Center | RWTH AACHEN UNIVERSITY

Heterogeneous Memory

- The memory subsystem is changing and becoming heterogeneous:
  - Cache hierarchy + new kinds of memory
- High-bandwidth memory (tech. example: HBM)
  - Smaller than “main memory”
  - Faster than “main memory”
- Large-Capacity memory (tech. example: NVM)
  - Larger than “main memory”
  - Slower than “main memory”
- Constant / scratchpad memory (tech. example: GPU memory)
  - Other than “main memory”
  - In the middle between host + device
- We believe: (partial) solution has to be provided by the programming system
  - Compiler + API + Runtime System + …

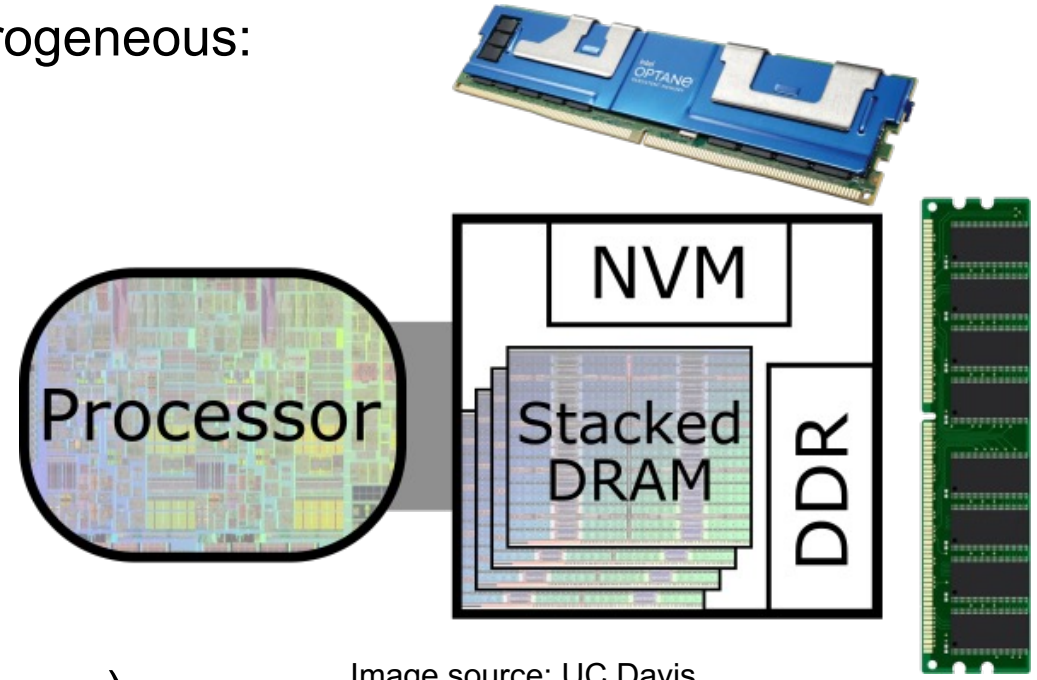


Image source: UC Davis

Research Questions + Agenda

- Challenges & Research Questions
  - Desire to have portable, vendor-neutral solution to expose memory kinds, memory characteristics, and allocate data on it
  - How to decide which data items to place in which kind of memory?
  - What happens when data access patterns change during application run?
  - How to collect knowledge about data items for decision making?

1. Characterization of (new kinds of) memory
2. Review: OpenMP Memory Management
3. H2M: Memory Allocation Abstractions
4. H2M: Early Evaluation
5. Summary

Characterization

Experiments

- **Memory Performance Characteristics: Bandwidth & Latency**
  - Interplay with NUMA effects
  - System: Intel Cascade Lake + Intel Optane

- **Bandwidth Benchmark: STREAM**
  - Clearly displays NUMA effects
  - Using `numactl` to specify
    - Specify where to run (`--cpunodebind`)
    - Specify which memory to use (`--membind`)
  - Evaluated different number of threads

- **Latency Benchmark: Intel Memory Latency Checker or Lmbench**
  - Pointer chasing (avoids HW prefetching)

Bandwidth Results – Cascade Lake + Optane (Regular STREAM Triad)

Architecture

```
CPU: Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
Freq Govenor: performance
--------------------
available: 4 nodes (0-3)
node 0 cpus: 0 2 4 6 8 10 12 14 16 18
             20 22 24 26 28 30 32 34 36 38
node 0 size: 191936 MB
node 0 free: 178709 MB
node 1 cpus: 1 3 5 7 9 11 13 15 17 19 21 23
             25 27 29 31 33 35 37 39
node 1 size: 192016 MB
node 1 free: 179268 MB
node 2 cpus:
node 2 size: 759808 MB
node 2 free: 759794 MB
node 3 cpus:
node 3 size: 761856 MB
node 3 free: 761851 MB
node distances:
node   0   1   2   3
  0:  10  21  17  28
  1:  21  10  28  17
  2:  17  28  10  28
  3:  28  17  28  10
```

DRAM + Optane

Results for CPU-Domain 0 on Socket 0 [MB/s]

Threads	Mem-Domain 0	Mem-Domain 1	Mem-Domain 2	Mem-Domain 3	DRAM - Local vs Remote	NVM / DRAM
1	10484,30	5720,93	5156,73	2817,33	1,8326	2,0331
2	20258,73	11180,27	9700,57	4672,83	1,8120	2,0884
3	29931,40	16419,10	12629,97	6402,63	1,8230	2,3699
4	39393,77	21381,30	14952,13	7777,47	1,8424	2,6347
5	47635,00	26099,27	16738,10	8996,57	1,8251	2,8459
6	56124,63	30449,43	18069,27	9937,73	1,8432	3,1061
7	63814,83	34368,80	19117,40	10682,77	1,8568	3,3380
8	71127,77	37621,47	19992,70	11237,80	1,8906	3,5577
9	77052,30	40462,83	20548,63	11665,90	1,9043	3,7498
10	82760,67	42491,03	21132,23	11578,80	1,9477	3,9163
11	87170,37	43757,17	21255,03	11052,03	1,9921	4,1012
12	90497,07	44515,83	21544,50	10421,80	2,0329	4,2005
13	92723,13	45005,23	21687,73	9807,03	2,0603	4,2754
14	94877,07	45303,67	21752,83	8900,00	2,0942	4,3616
15	96342,97	45459,00	21711,43	7855,93	2,1193	4,4374
16	97184,43	45486,57	21658,70	6677,27	2,1366	4,4871
17	97578,23	45499,37	21555,20	5649,77	2,1446	4,5269
18	97749,70	45490,17	21565,00	4597,50	2,1488	4,5328
19	97817,47	45475,37	21562,40	3602,27	2,1510	4,5365
20	97713,80	45477,97	21374,57	2999,00	2,1486	4,5715

DRAM Sockets **Optane Socket 0** **Optane Socket 1**

Latency Results – Cascade Lake + Optane (Intel Memory Latency Checker)

Architecture

```
CPU: Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
Freq Govenor: performance
--------------------
available: 4 nodes (0-3)
node 0 cpus: 0 2 4 6 8 10 12 14 16 18
             20 22 24 26 28 30 32 34 36 38
node 0 size: 191936 MB
node 0 free: 178709 MB
node 1 cpus: 1 3 5 7 9 11 13 15 17 19 21 23
             25 27 29 31 33 35 37 39
node 1 size: 192016 MB
node 1 free: 179268 MB
node 2 cpus:
node 2 size: 759808 MB
node 2 free: 759794 MB
node 3 cpus:
node 3 size: 761856 MB
node 3 free: 761851 MB
node distances:
node   0   1   2   3
  0:  10  21  17  28
  1:  21  10  28  17
  2:  17  28  10  28
  3:  28  17  28  10
```

DRAM + Optane

Results for CPU-Domain 0 on Socket 0 [ns]

Threads	Mem-Domain 0	Mem-Domain 1	Mem-Domain 2	Mem-Domain 3	DRAM - Local vs Remote	NVM / DRAM
1	75,70	121,60	245,80	298,70	1,6063	3,2470
2	92,93	150,92	398,16	462,27	1,6240	4,2845
3	97,33	159,17	481,21	535,27	1,6354	4,9441
4	101,91	167,28	553,87	676,81	1,6414	5,4349
5	108,24	179,38	622,06	722,30	1,6572	5,7470
6	114,68	196,22	675,97	730,87	1,7110	5,8944
7	121,12	218,34	751,61	820,22	1,8027	6,2055
8	127,32	244,76	808,12	874,56	1,9224	6,3472
9	134,66	268,03	855,67	917,77	1,9904	6,3543
10	143,19	287,20	919,06	994,00	2,0057	6,4185
11	151,71	302,67	943,30	1042,94	1,9951	6,2178
12	160,30	315,32	970,63	1085,39	1,9671	6,0551
13	170,52	327,16	994,46	1132,31	1,9186	5,8319
14	180,22	337,80	1004,35	1129,78	1,8744	5,5729
15	191,27	346,65	1001,74	1137,59	1,8124	5,2373
16	203,71	355,55	1006,11	1135,21	1,7454	4,9389
17	214,38	362,42	1022,54	1146,80	1,6905	4,7698
18	223,68	369,44	1014,38	1174,31	1,6516	4,5350
19	233,62	377,90	1020,73	1206,89	1,6176	4,3692
20	244,60	385,06	1027,22	1205,71	1,5742	4,1996

DRAM Sockets | Optane Socket 0 | Optane Socket 1

Results

- **Latency and bandwidth depend on**
  - Origin of access in relation to target of access
  - Number of threads
  - Kind of memory

- **Note 1:** Memory “performance” ordering regarding bandwidth different than regarding latency
  - “Performance” becomes even more application-specific (if one can say so)
    - As always: some applications might not be affected
  - “Fast” is relative: bandwidth, latency, …

- **Implication for performance modeling**
  - Successful (but simple) models such as Roofline might not work anymore
  - Need to take access pattern into account

- **Note 2:** Situation becomes even more complicated if energy usage is taken into account
  - Not considered here!

Review: OpenMP Memory Management

OpenMP Memory Management: Memory Spaces & Traits

- Allocator := OpenMP object that fulfills requests to allocate and deallocate storage for prog. variables
- **Memory Spaces**

Memory space name	Storage selection intent
`omp_default_mem_space`	Represents the system default storage
`omp_large_cap_mem_space`	Represents storage with large capacity
`omp_const_mem_space`	Represents storage optimized for variables with constant values
`omp_high_bw_mem_space`	Represents storage with high bandwidth
`omp_low_lat_mem_space`	Represents storage with low latency

- **Predefined Allocators**

Allocator name	Associated memory space	Non-default trait values
`omp_default_mem_alloc`	`omp_default_mem_space`	`fallback:null_fb`
`omp_large_cap_mem_alloc`	`omp_large_cap_mem_space`	(none)
`omp_const_mem_alloc`	`omp_const_mem_space`	(none)
`omp_high_bw_mem_alloc`	`omp_high_bw_mem_space`	(none)
`omp_low_lat_mem_alloc`	`omp_low_lat_mem_space`	(none)
`omp_cgroup_mem_alloc`	Implementation defined	`access:cgroup`
`omp_pteam_mem_alloc`	Implementation defined	`access:pteam`
`omp_thread_mem_alloc`	Implementation defined	`access:thread`

Fortran

- **Allocator Traits**

Allocator trait	Allowed values	Default value
`sync_hint`	`contended`, `uncontended`, `serialized`, `private`	`contended`
`alignment`	A positive integer value that is a power of 2	1 byte
`access`	`all`, `cgroup`, `pteam`, `thread`	`all`
`pool_size`	Positive integer value	Implementation defined
`fallback`	`default_mem_fb`, `null_fb`, `abort_fb`, `allocator_fb`	`default_mem_fb`
`fb_data`	an allocator handle	(none)
`pinned`	`true`, `false`	`false`
`partition`	`environment`, `nearest`, `blocked`, `interleaved`	`environment`

OpenMP allocator traits

- `fallback`: describes the behavior if the allocation cannot be fulfilled
  - `default_mem_fb`: system's default
  - Other options: null, abort, different allocator
- `pinned`: pinned memory, i.e. for GPUs
- `partition`: partitioning of allocated memory of physical storage resources (think of NUMA)
  - `environment`: use system's default behavior
  - `nearest`: closest memory
  - `blocked`: partitioning into approx. same size with at most one block per storage resource
  - `interleaved`: partitioning in a round-robin fashion across the storage resources

- Construction of allocators with traits via
  - `omp_allocator_handle_t omp_init_allocator( omp_memspace_handle_t memspace, int ntraits, const omp_alloctrait_t traits[] );`
    - Selection of memory space mandatory
    - Empty traits set: use defaults
- Allocators have to be destroyed with `*_destroy_*`

Memory Management Status

- **LLVM OpenMP runtime internally already uses libmemkind (libnuma, numactl)**
  - Support for various kinds of memory: DDR, HBW and Persistent Memory (Optane)
  - Library loaded at initialization (checks for availability)
  - If requested memory space for allocator is not available → fallback to DDR (aka “main memory”)

- **Memory Management implementation in LLVM still not complete**
  - Some allocator traits not implemented yet
  - Some `partition` values not implemented yet (environment, interleaved, nearest, blocked)
  - Semantics of `omp_high_bw_mem_space and omp_large_cap_mem_space` unclear. Which memory should be used?
    - Option 1: Explicitly target HBM → currently implemented in LLVM
    - Option 2: Target memory with the highest available bandwidth

H2M: Memory Allocation Abstractions

H2M := Heuristics for Heterogeneous Memory

Abstracting Memory Allocation / 1

- **Memory Allocation in C/C++ as an example**
  - Currently specification contains only size of chunk

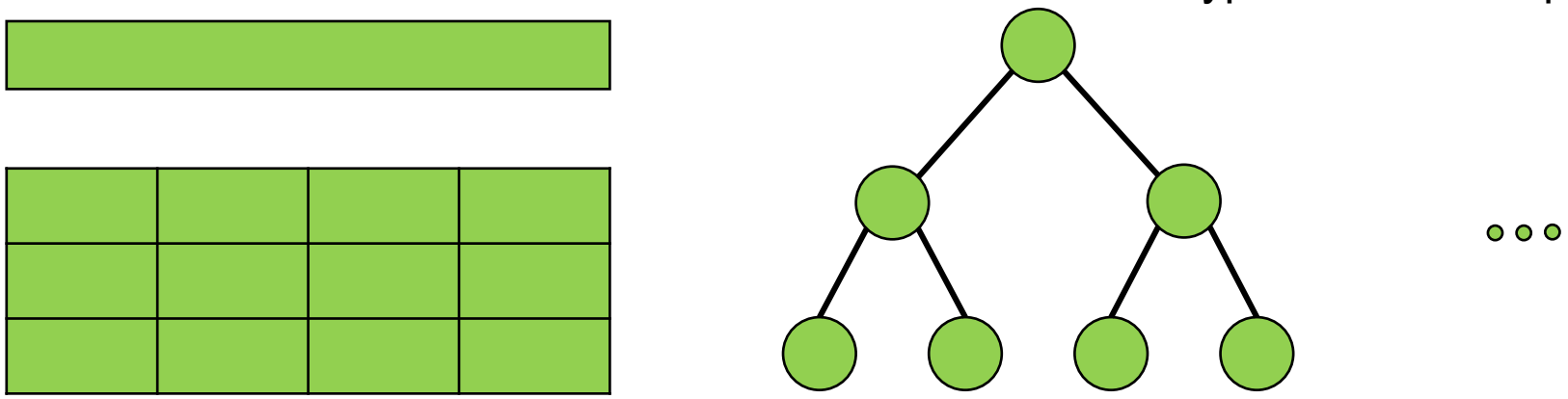
```
double* a = (double*) malloc(N * sizeof(double));
```

- **Problem 1:** No way to target different kinds of memory in the system
  - OpenMP 5.0 introduced memory management
  - Possibility to define requirements to target desired memory

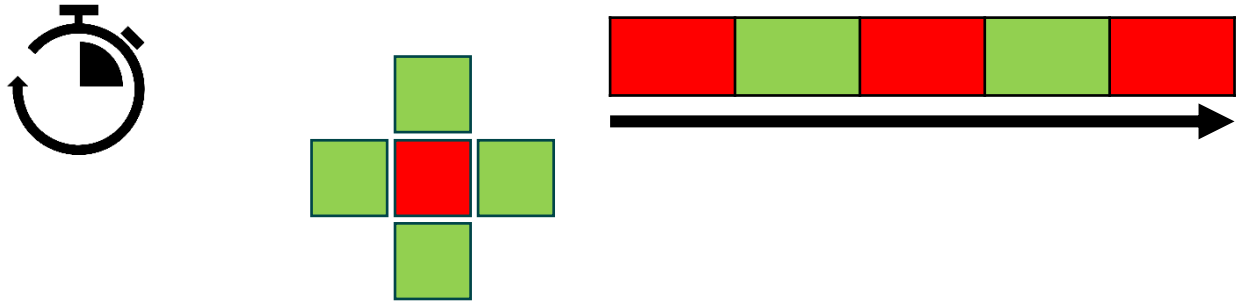
```
// use predefined allocator type
double* a = (double*) omp_alloc(N * sizeof(double), omp_high_bw_mem_alloc);

// possibility to define custom allocator with traits
omp_memspace_handle_t xy_memspace = omp_default_mem_space;
omp_alloctrait_t xy_traits[1]     ={omp_atk_alignment, 64};
omp_allocator_handle_t xy_alloc   = omp_init_allocator(xy_memspace,1,xy_traits);
X =(float *)omp_alloc(N * sizeof(float), xy_alloc);
```

Abstracting Memory Allocation / 2

- **Problem 2:** No information what kind of data structure or type the chunk represents



- **Problem 3:** No Information about access pattern and frequency
  - How frequent is data used?
  - Linear access?
  - Strided or random accesses?
  - Stencil?
  - Read/write or both?



H2M: Abstracting Memory Allocation / 1

- **Design of allocation traits**
  - Key value pairs
    - `h2m_alloc_trait_key_t`
    - `h2m_alloc_trait_value_t`
  - **Intention:** values can be combined using logical OR
  - **Example:** sensitivity
    - Data item could be sensitive to bw and latency

```
typedef struct h2m_alloc_trait_t {
    h2m_alloc_trait_key_t key;
    union h2m_atv_union_t {
        int i;
        unsigned int ui;
        double dbl;
        long l;
        h2m_alloc_trait_value_t atv;
        void *ptr;
    } value;
} h2m_alloc_trait_t;
```

```
typedef enum h2m_alloc_trait_key_t {
    h2m_atk_req_mem_space       = 1,
    h2m_atk_pref_mem_space      = 2,
    h2m_atk_mem_alignment       = 3,
    h2m_atk_sensitivity         = 4,
    h2m_atk_access_mode         = 5,
    h2m_atk_access_freq         = 6,
    h2m_atk_access_prio         = 7,
    h2m_atk_access_pattern      = 8,
    h2m_atk_access_stride       = 9,
    h2m_atk_structure_type      = 10
} h2m_alloc_trait_key_t;
```

```
typedef enum h2m_alloc_trait_value_t {
  h2m_atv_mem_space_hbw          = 0x000001,
  h2m_atv_mem_space_low_lat      = 0x000002,
  h2m_atv_mem_space_large_cap    = 0x000004,

  h2m_atv_bw_sensitive           = 0x000008,
  h2m_atv_lat_sensitive          = 0x000010,

  ...

} h2m_alloc_trait_value_t;
```

H2M: Abstracting Memory Allocation / 2

- **Runnable runtime implementation draft providing**
  - Basic API routines for allocation abstraction
  - Initial set of allocation traits (see previous slide)

  - Possibility to implement / record internal statistics like:
    - Elapsed time for operations
    - Bandwidth when migrating data objects
  - CMake build system + README
  - Tools interface (initial version)

```
// Init / Finalize
int h2m_init();
int h2m_thread_init();
int h2m_post_init_serial();
int h2m_thread_finalize();
int h2m_finalize();
```

```
// High-level memory allocation abstraction
void* h2m_alloc(size_t size, int* err);
void* h2m_alloc_w_traits(size_t size, int* err, int n_traits, const h2m_alloc_trait_t traits[]);
int   h2m_free(void* ptr);

// Updating traits and triggering data migration
int   h2m_update_traits(void* ptr, size_t size, int n_traits, const h2m_alloc_trait_t traits[], int replace_complete);
int   h2m_apply_migration();
int   h2m_apply_migration_for_allocations(void** allocations, int n_allocs);
```

H2M: Mapping to Memory Kinds / 1

- **How will hwloc or the H2M runtime identify / expose available kinds of memory and characteristics?**
  - hwloc 2.7.0 is able to identify different kinds of memory

Example: Intel Optane System

```
$ hwloc-ls -v | grep NUMA

NUMANode L#0 (P#0 local=263845192KB total=263845192KB)
NUMANode L#1 (P#2 local=1040187392KB total=1040187392KB DAXDevice=dax0.0)
NUMANode L#2 (P#1 local=264198452KB total=264198452KB)
NUMANode L#3 (P#3 local=1040187392KB total=1040187392KB DAXDevice=dax1.0)
```

Example: KNL

```
$ hwloc-ls -v | grep NUMA

NUMANode L#0 (P#0 local=49149524KB total=49149524KB)
NUMANode(MCDRAM) L#1 (P#4 local=4128512KB total=4128512KB)
NUMANode L#2 (P#1 local=49535204KB total=49535204KB)
NUMANode(MCDRAM) L#3 (P#5 local=4128512KB total=4128512KB)
NUMANode L#4 (P#2 local=49535204KB total=49535204KB)
NUMANode(MCDRAM) L#5 (P#6 local=4128512KB total=4128512KB)
NUMANode L#6 (P#3 local=49535204KB total=49535204KB)
NUMANode(MCDRAM) L#7 (P#7 local=4128048KB total=4128048KB)
```

  - H2M library queries hwloc API to identify types to build memory space mapping
    - `h2m_atv_mem_space_hbw` → Nodes with highest BW (might be HBM or DRAM)
    - `h2m_atv_mem_space_low_lat` → Nodes with lowest latency (usually DRAM for now)
    - `h2m_atv_mem_space_large_cap` → Nodes with highest capactity (might be DRAM or NVM)

H2M: Mapping to Memory Kinds / 2

- **Examples on Icelake System + Optane:**

```
$ ./allocation_traits.exe

2022-03-16 06:47:47.219.766 H2MLib R#0 T#0 (OS_TID:595593): --> Number of NUMA nodes: 4
2022-03-16 06:47:47.219.800 H2MLib R#0 T#0 (OS_TID:595593): --> NUMANode Idx=0 - local=263845192KB,total=263845192KB
2022-03-16 06:47:47.219.900 H2MLib R#0 T#0 (OS_TID:595593): --> NUMANode Idx=1 - local=1040187392KB,total=1040187392KB,DAXDevice=dax0.0
2022-03-16 06:47:47.219.905 H2MLib R#0 T#0 (OS_TID:595593): --> NUMANode Idx=2 - local=264198452KB,total=264198452KB
2022-03-16 06:47:47.219.917 H2MLib R#0 T#0 (OS_TID:595593): --> NUMANode Idx=3 - local=1040187392KB,total=1040187392KB,DAXDevice=dax1.0
2022-03-16 06:47:47.219.984 H2MLib R#0 T#0 (OS_TID:595593): --> Mapping h2m_atv_mem_space_hbw -> DRAM
2022-03-16 06:47:47.220.070 H2MLib R#0 T#0 (OS_TID:595593): --> Mapping h2m_atv_mem_space_low_lat -> DRAM
2022-03-16 06:47:47.220.079 H2MLib R#0 T#0 (OS_TID:595593): --> Mapping h2m_atv_mem_space_large_cap -> NVM
```

```
$ H2M_NODES_HBW=0,1 H2M_NODES_LOW_LAT=2,3 H2M_NODES_LARGE_CAP=1,3 ./allocation_traits.exe

2022-03-16 06:52:04.613.487 H2MLib R#0 T#0 (OS_TID:596221): --> Number of NUMA nodes: 4
2022-03-16 06:52:04.613.524 H2MLib R#0 T#0 (OS_TID:596221): --> NUMANode Idx=0 - local=263845192KB,total=263845192KB
2022-03-16 06:52:04.613.583 H2MLib R#0 T#0 (OS_TID:596221): --> NUMANode Idx=1 - local=1040187392KB,total=1040187392KB,DAXDevice=dax0.0
2022-03-16 06:52:04.613.592 H2MLib R#0 T#0 (OS_TID:596221): --> NUMANode Idx=2 - local=264198452KB,total=264198452KB
2022-03-16 06:52:04.613.605 H2MLib R#0 T#0 (OS_TID:596221): --> NUMANode Idx=3 - local=1040187392KB,total=1040187392KB,DAXDevice=dax1.0
2022-03-16 06:52:04.613.671 H2MLib R#0 T#0 (OS_TID:596221): --> Mapping h2m_atv_mem_space_hbw -> 0,1
2022-03-16 06:52:04.613.844 H2MLib R#0 T#0 (OS_TID:596221): --> Mapping h2m_atv_mem_space_low_lat -> 2,3
2022-03-16 06:52:04.613.856 H2MLib R#0 T#0 (OS_TID:596221): --> Mapping h2m_atv_mem_space_large_cap -> 1,3
```

H2M: Mapping to Memory Kinds / 3

- **How do we treat or handle NUMA effects?**
  - During allocation?
  - In heuristics for data accesses?

- **Idea:** NUMA local lists - currently ordered by distance
  - Note: Target close memory nodes but at the same time preserve first-touch semantics (depends on partitioning)

```
$ ./allocation_traits.exe
2022-03-16 06:47:47.219.984 H2MLib R#0 T#0 (OS_TID:595593): --> Mapping h2m_atv_mem_space_hbw -> DRAM
2022-03-16 06:47:47.220.070 H2MLib R#0 T#0 (OS_TID:595593): --> Mapping h2m_atv_mem_space_low_lat -> DRAM
2022-03-16 06:47:47.220.079 H2MLib R#0 T#0 (OS_TID:595593): --> Mapping h2m_atv_mem_space_large_cap -> NVM
2022-03-16 06:47:47.220.481 H2MLib R#0 T#0 (OS_TID:595593): --> Source node 0 - Sorted by distances: (N: 0, D: 10) (N: 2, D: 17) (N: 1, D: 20) (N: 3, D: 28)
2022-03-16 06:47:47.220.527 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 0 and memspace h2m_atv_mem_space_hbw: 0 2
2022-03-16 06:47:47.220.537 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 0 and memspace h2m_atv_mem_space_low_lat: 0 2
2022-03-16 06:47:47.220.540 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 0 and memspace h2m_atv_mem_space_large_cap: 1 3
2022-03-16 06:47:47.220.555 H2MLib R#0 T#0 (OS_TID:595593): --> Source node 1 - Sorted by distances: (N: 1, D: 10) (N: 3, D: 17) (N: 0, D: 20) (N: 2, D: 28)
2022-03-16 06:47:47.220.576 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 1 and memspace h2m_atv_mem_space_hbw: 0 2
2022-03-16 06:47:47.220.581 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 1 and memspace h2m_atv_mem_space_low_lat: 0 2
2022-03-16 06:47:47.220.585 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 1 and memspace h2m_atv_mem_space_large_cap: 1 3
2022-03-16 06:47:47.220.591 H2MLib R#0 T#0 (OS_TID:595593): --> Source node 2 - Sorted by distances: (N: 2, D: 10) (N: 0, D: 17) (N: 1, D: 28) (N: 3, D: 28)
2022-03-16 06:47:47.220.603 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 2 and memspace h2m_atv_mem_space_hbw: 2 0
2022-03-16 06:47:47.220.607 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 2 and memspace h2m_atv_mem_space_low_lat: 2 0
2022-03-16 06:47:47.220.610 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 2 and memspace h2m_atv_mem_space_large_cap: 1 3
2022-03-16 06:47:47.220.616 H2MLib R#0 T#0 (OS_TID:595593): --> Source node 3 - Sorted by distances: (N: 3, D: 10) (N: 1, D: 17) (N: 0, D: 28) (N: 2, D: 28)
2022-03-16 06:47:47.220.629 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 3 and memspace h2m_atv_mem_space_hbw: 0 2
2022-03-16 06:47:47.220.633 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 3 and memspace h2m_atv_mem_space_low_lat: 0 2
2022-03-16 06:47:47.220.637 H2MLib R#0 T#0 (OS_TID:595593): --> Resulting view from numa node 3 and memspace h2m_atv_mem_space_large_cap: 3 1
```

H2M: Abstracting Memory Allocation

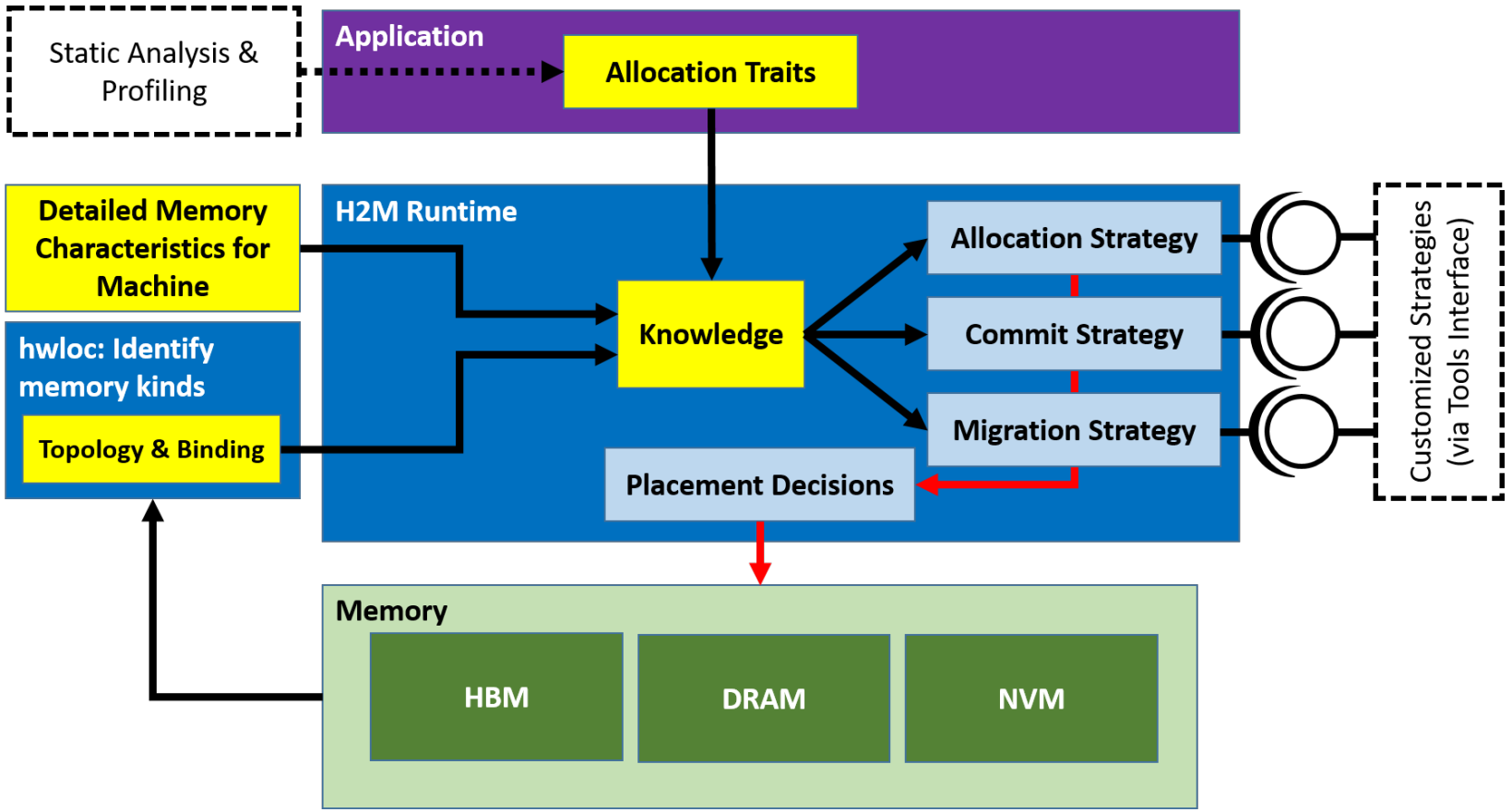
- **Goals (excerpt):**
  - Extend memory allocation interface with an option to specify information about
    - Data structure or data representation
    - Alignment, partitioning or placement of data
    - Access patterns

- **Questions (under discussion):**
  - How to exploit that information to form a clever decision where to place data and how to bind threads
  - What happens when the access pattern changes over time?
    - Add means to specify a change of access pattern
    - Decide if and where to move data?

H2M: Early Evaluation

H2M := Heuristics for Heterogeneous Memory

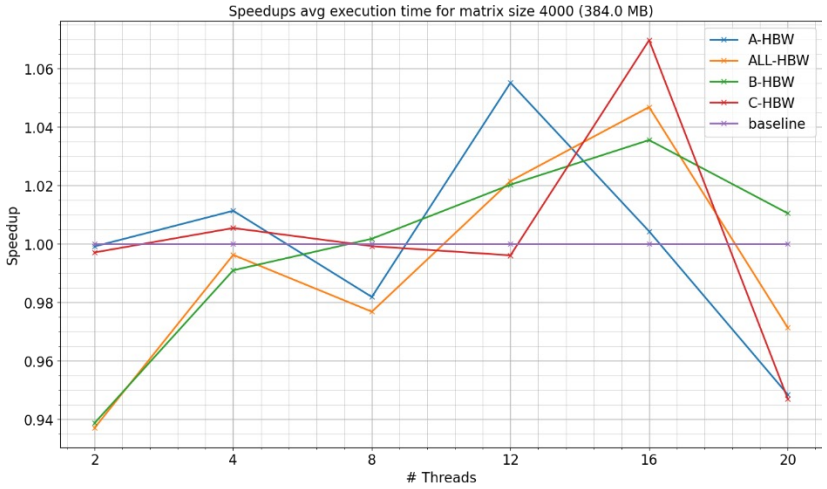
H2M: Workflow and Concept

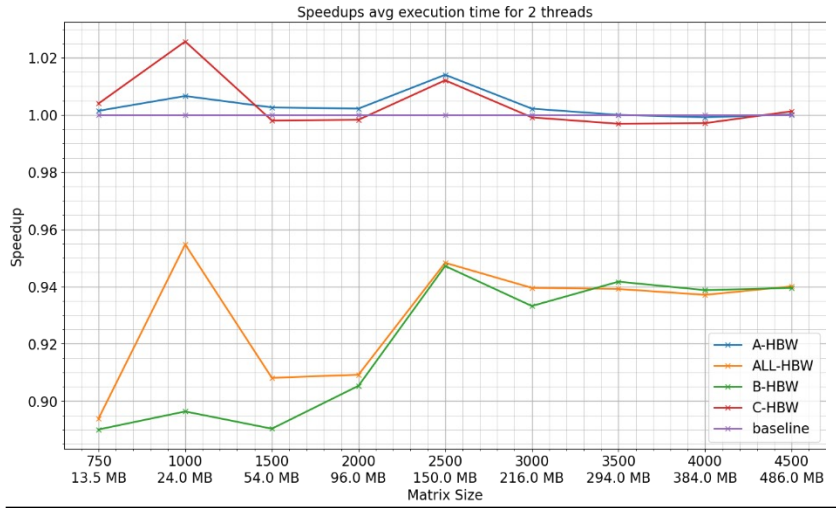


- **Allocation Strategy**
  - Generates placement decision / recommendation based on trait set for a single entity
- **Commit Strategy**
  - Receives multiple allocation declarations (with traits) and decides about the final allocation order
  - Can make use of allocation strategy but is also able to modify traits or placement decision
- **Migration Strategy**
  - Receives information about registered allocations and their current trait sets (that might have been updated in the meantime) and decides how and where to move data

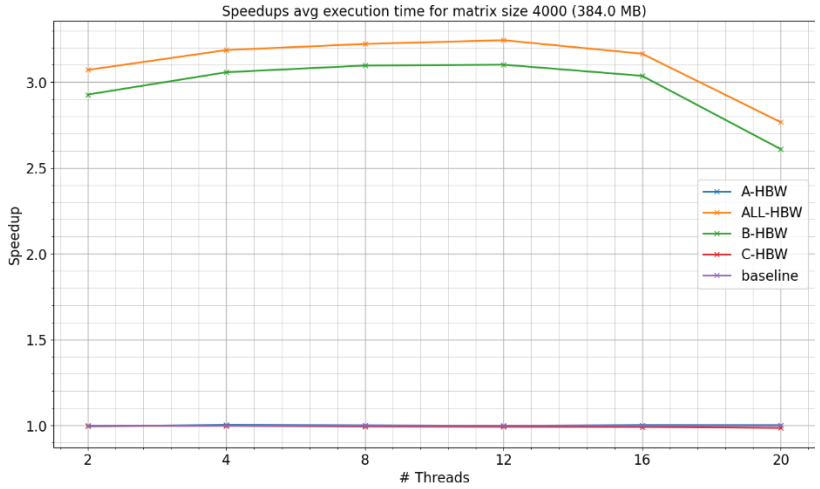
Evaluation – Empirical Results for Kernels – DGEMM (w/o blocking!)

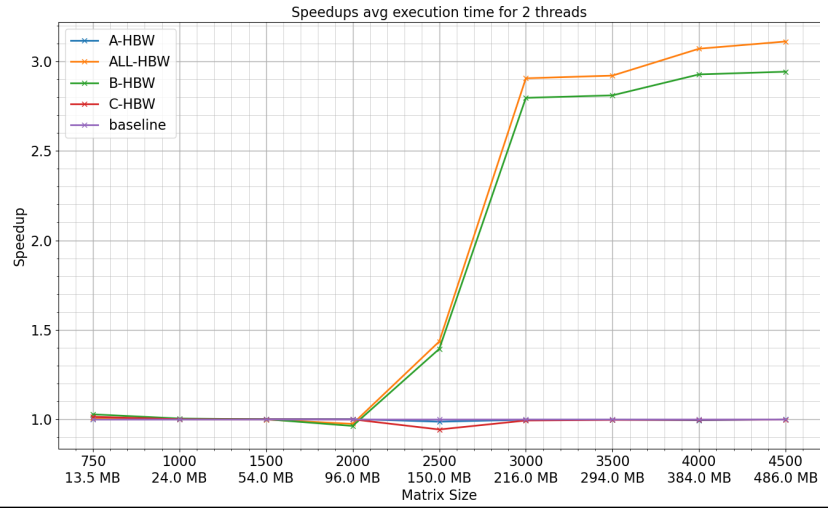
KNL (Note: HBM has higher latency)





Ice Lake + Optane

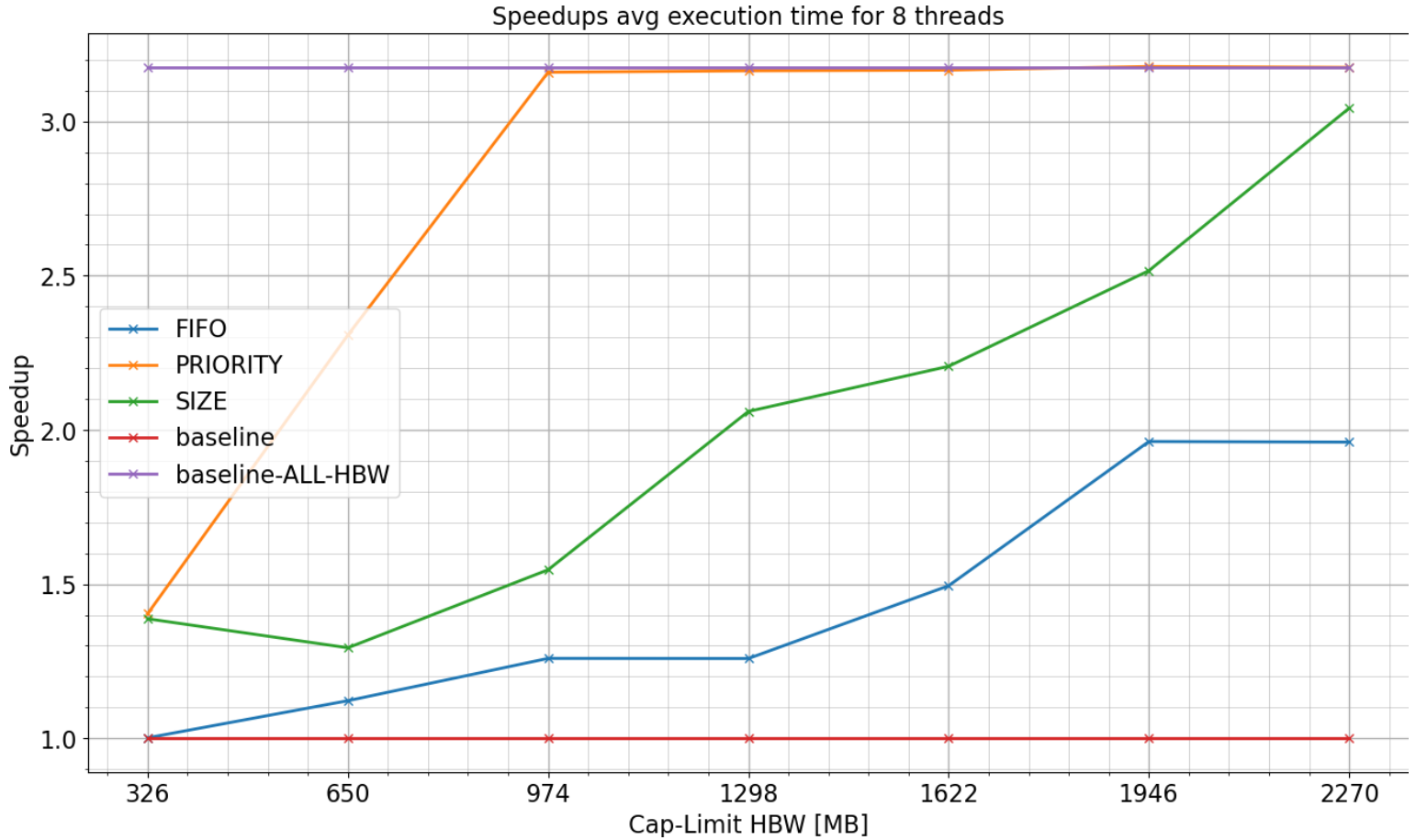




- **Takeaways**
  - Not much improvement on KNL (+/- 5%) despite higher bandwidth
  - Array B seems to be highly latency sensitive due to frequent accesses
- **Profile-based detection of access pattern**
- Resulting traits for A
  - `h2m_atv_mem_space_hbw`
  - `prio = 1.0`
- Resulting traits for B
  - `h2m_atv_mem_space_low_lat`
  - `prio = 15.0`
- Resulting traits for C
  - `h2m_atv_mem_space_hbw`
  - `prio = 5.0`

Evaluation – H2M PoC for Application with Multiple Kernels – Ice Lake + Optane

- **Here:** Only using DGEMM (wo/ blocking!) kernel with various problem sizes

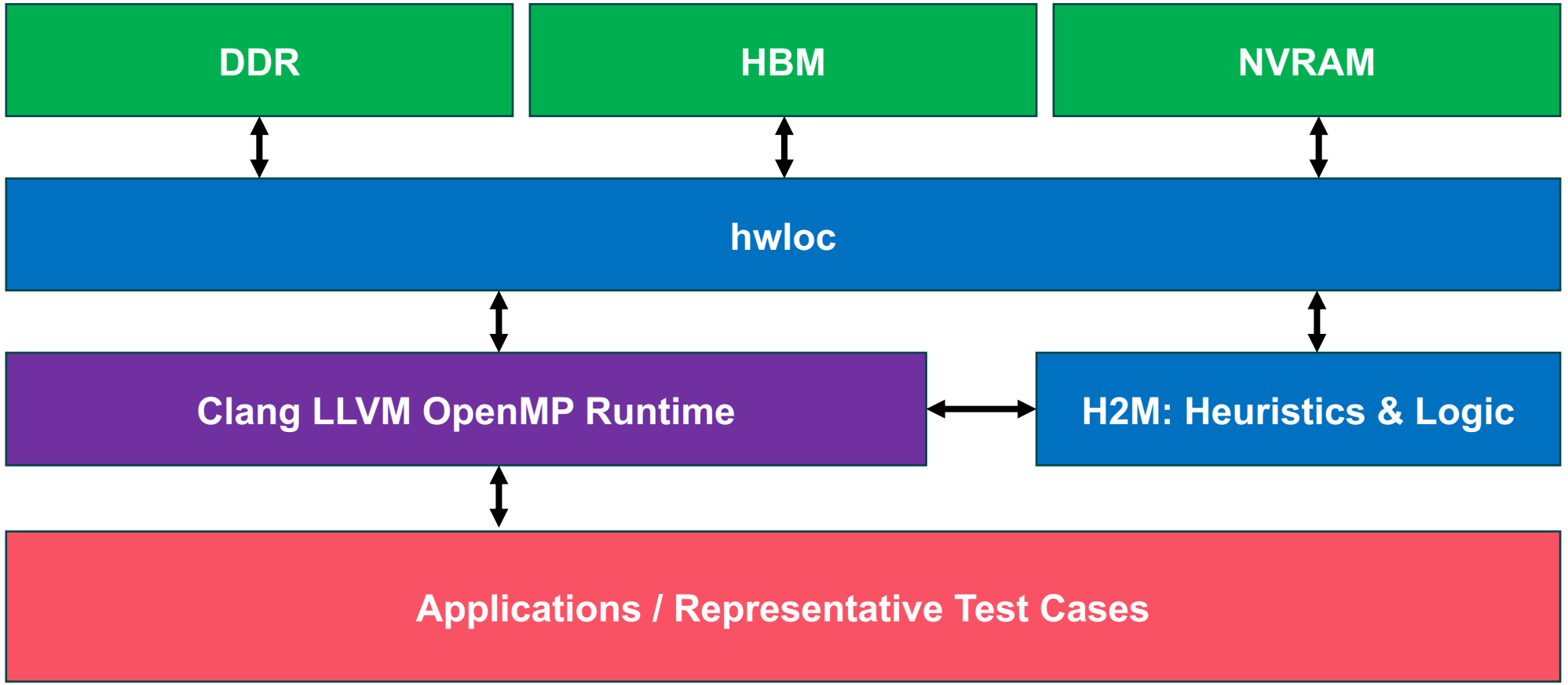


- **Setting**
  - Max memory usage: 2592 MB
  - Capacity of `HBW` memory limited with various thresholds
  - Additionally baselines for all `LARGE_CAP` and HBW (no limit)

- **Takeaways**
  - Strategy that prefers large buffers first (*SIZE*) gets good speedup
  - *PRIORITY* strategy that first orders by priority and then by size almost reaches best case quickly
  - Both strategies outperform default *FIFO* strategy that depends on how buffers/kernels are created in the application

Summary

Prototype: Sketch of Combined Solution



- **Our medium- to long-term goal: integration into programming system (hwloc, OpenMP, …)**

Summary

- **The memory subsystem is changing …**
  - … and “memory performance” becomes even more complex to assess
- **Portable, (semi-)automatic approach managing data in heterogeneous memory**
  - Promising results for memory-bound scenarios
  - Not covered here: Static analysis and profiling help to understand access characteristics of data items and to recommend allocation traits …
- **Remaining questions and action items:**
  - How can we ***detect sensitivity*** of data items efficiently?
  - How to specify more ***access patterns*** in the program code / detect those at compile time?
  - Development of ***performance models*** and ***heuristics*** that exploit information about both platform and data items to make placement decisions

Thank you for your attention

- If you are interested, we are happy to talk and share ideas!

- People on the picture
  - Christian Terboven (RWTH)
  - Brice Goglin (INRIA)
  - Emmanuel Jeannot (INRIA)
  - Clement Foyer (INRIA)
  - Anara Kozhokanova (RWTH)
  - Jannis Klinkenberg (RWTH)