H2M: Towards Heuristics for Heterogeneous Memory

Jannis Klinkenberg ✉, Anara Kozhokanova, Christian Terboven (RWTH)
Clément Foyer ✉, Brice Goglin, Emmanuel Jeannot (Inria)

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Inria

Motivation & Challenges

- The performance gap between compute units and memory subsystem further spreads as the complexity of HPC systems increases
- Applications' demand for higher memory capacity and faster speed rises

⇒ Emergence of new memory technologies

- *High Bandwidth Memory* (HBM) delivers more bandwidth but has less capacity than DRAM
- *Non-Volatile Memory* (NVM) has larger capacity and consumes less power but has lower bandwidth

⇒ More machines with *Heterogeneous Memory* (e.g. DRAM+NVM) as costs for large HBM-only or DRAM-only machines are too high

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?

Exposing Memory & Characteristics

- ***hwloc*** now capable of exposing available memory and basic scalar characteristics (NUMA effects and scaling behavior not covered)
- Extended analysis to identify precise memory characteristics
  - Different scaling behavior/limits when accessed from multiple threads (Fig. 1)
  - Impact of concurrent accesses and bandwidth saturation on latency (Fig. 2)

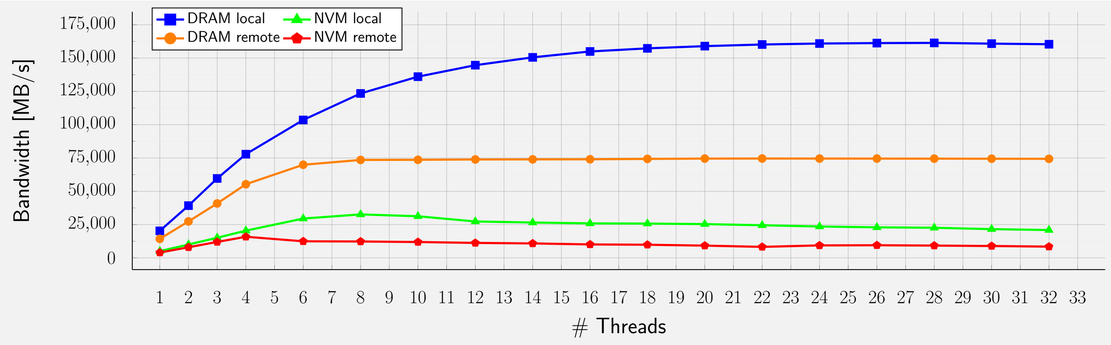


Figure 1: Bandwidth to local and remote DRAM/NVM on a dual-socket Intel Xeon Ice Lake machine

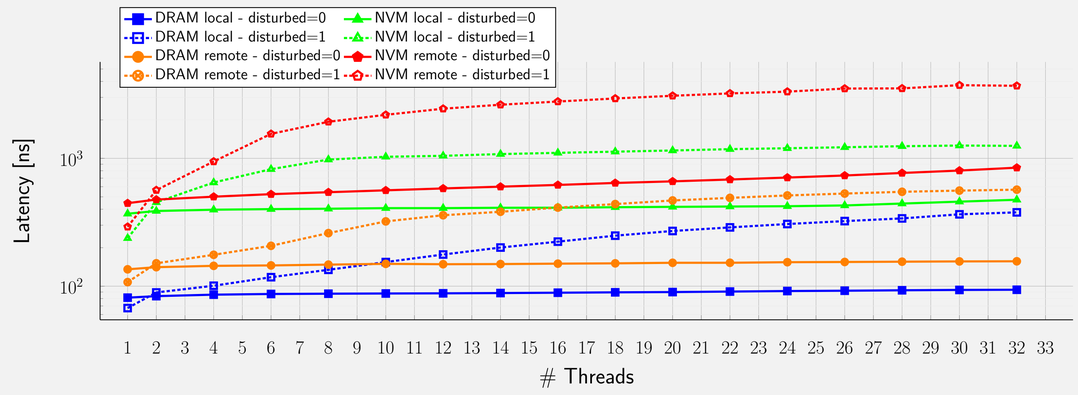


Figure 2: Latency to local and remote DRAM/NVM on a dual-socket Intel Xeon Ice Lake machine. **Disturbed:** 1 thread performing pointer chasing and $n-1$ threads saturate bandwidth. **Non-disturbed:** All threads perform pointer chasing only

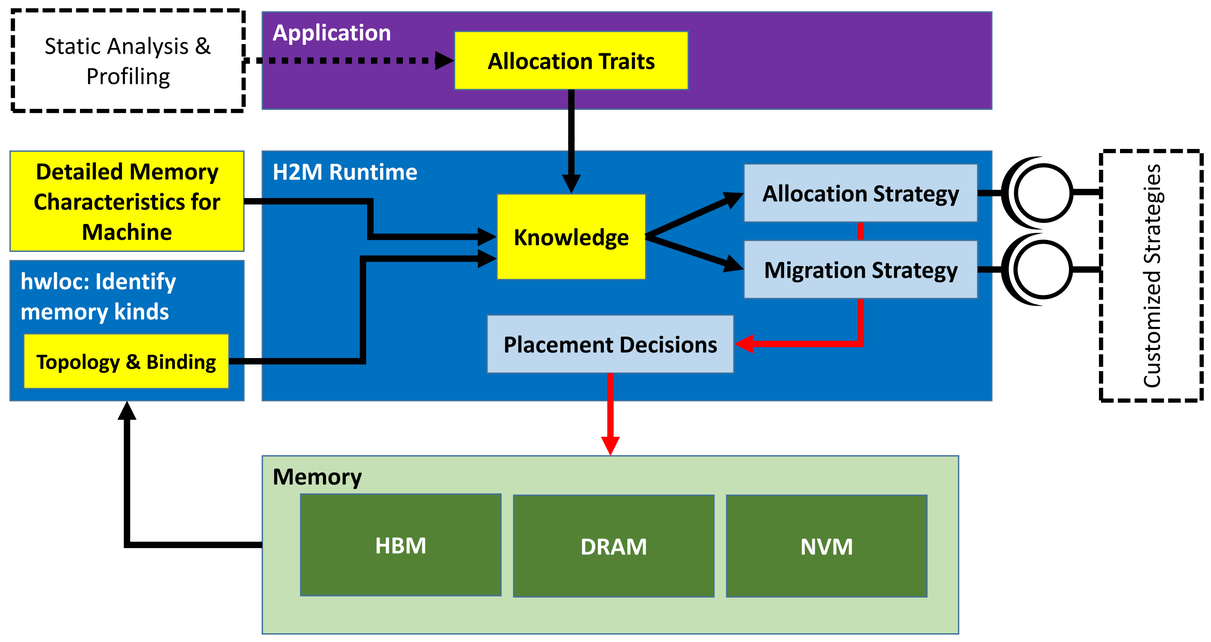
Approach & Components

Runtime & Allocation Abstraction

- Abstract memory allocation and allow to express traits for data items
- Gather additional knowledge about memory kinds and characteristics
- Heuristics/Strategies exploit knowledge to generate placement decision

Memory Access Analysis & Profiling

- Static compiler analysis to detect data accesses patterns
- Employ sampling approach with NumaMMA and Precise Event-Based Sampling (PEBS)
- Profiling based on allocation interception and compiler instrumentation using LLVM and AddressSanitizer



Preliminary Evaluation & Results

Evaluated with

- Dual-socket Intel Xeon Gold 6338 machine (codename *Ice Lake*) with 512 GB DRAM and 2 TB NVM (Optane DC Persistent Memory)
- Synthetic multi-phase naive DGEMM benchmark (details: see Fig. 3)

Note: Limited capacity for memory with highest bandwidth
⇒ initial data placement and movement crucial to sustain performance

Results

- Promising perspective for memory-bound scenarios
- *H2M* clearly beats NVM baseline (*worst-case*) in most cases
- Close to DRAM performance (*best-case*) despite migration overhead

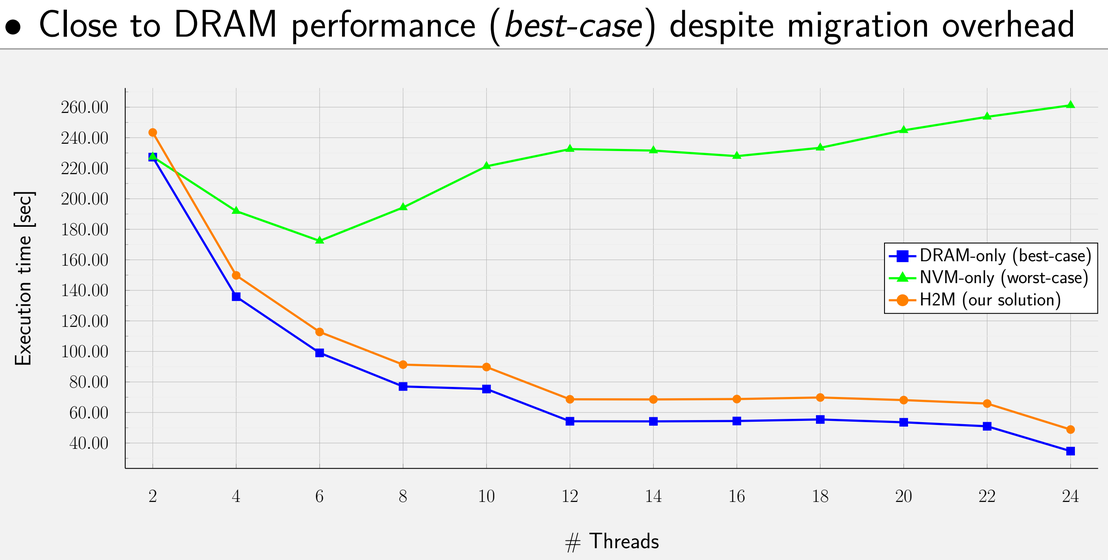


Figure 3: Performance comparison for benchmark that performs multiple naive DGEMM phases on an Intel Xeon Ice Lake machine. Each phase consists of 24 separate DGEMM operations (non-blocked, memory-bound) with matrix-size $ms = 1400 \times 1400$ (≈16 MB). Data of the last and upcoming phase are moved to slower and faster memory respectively.

Code Example for Allocation Abstraction[1]

```
int err; size_t N; // size of data item
double* data_item;
// specify allocation traits (requirements and hints)
h2m_alloc_trait_t traits[5] = {
    h2m_atk_access_mode,      h2m_atv_access_mode_readwrite,
    h2m_atk_access_freq,      2000, /* e.g. accesses per sec */
    h2m_atk_access_origin,    h2m_atv_access_origin_single_thread,
    h2m_atk_access_pattern,   h2m_atv_access_pattern_strided,
    h2m_atk_access_stride,    4 /* in bytes */ };
// allocate data item according to traits specified
data_item = (double*) h2m_alloc_w_traits(N, &err, 5, traits);
// or if situation changes -> update traits for items
err = h2m_update_traits(data_item, N, 5, traits);
// trigger data migration to respect new hints/requirements
err = h2m_apply_migration();
```

[1] *Code under development but will be available on GitHub soon*

Conclusion & Future Work

- Portable, (semi-)automatic approach managing data in heterogeneous memory that greatly improves performance compared to NVM baseline
- Static analysis and profiling help to understand access characteristics of data items and to recommend allocation traits

Future Work

- Further investigation of static analysis and development of performance models and heuristics that exploit information about both platform and data items to make placement decisions
- Prefetching approaches to hide/overlap costs for data movement
- Extensive evaluation with various applications and platforms

j.klinkenberg@itc.rwth-aachen.de clement.foyer@inria.fr http://www.i12.rwth-aachen.de/