



HAL
open science

H2M: Towards Heuristics for Heterogeneous Memory

Jannis Klinkenberg, Anara Kozhokanova, Christian Terboven, Clément Foyer,
Brice Goglin, Emmanuel Jeannot

► **To cite this version:**

Jannis Klinkenberg, Anara Kozhokanova, Christian Terboven, Clément Foyer, Brice Goglin, et al..
H2M: Towards Heuristics for Heterogeneous Memory. IEEE Cluster 2022 - 2022 IEEE International
Conference on Cluster Computing, Sep 2022, Heidelberg, Germany. hal-03886110

HAL Id: hal-03886110



<https://inria.hal.science/hal-03886110>

Submitted on 6 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

H2M: Towards Heuristics for Heterogeneous Memory

Clément Foyer  , Brice Goglin
Emmanuel Jeannot
Inria, Univ. Bordeaux, LaBRI
Talence, France

Jannis Klinkenberg  , Anara Kozhokanova
Christian Terboven
RWTH Aachen University
Aachen, Germany

Index Terms—Memory, Heterogeneous Memory, Non-Volatile Memory, High-Bandwidth Memory, Heuristics, HPC, Data Placement

I. INTRODUCTION

For the past years, scientific applications and simulations show increasing demand for both memory speed and capacity. The performance gap between compute units and the memory subsystem continues to spread which led to redesigns and the emergence of new technologies. Recent architectures already comprise, next to classical DRAM, portions of *High Bandwidth Memory* (HBM) that has less capacity than DRAM and is solving only one of the requirements. The newly introduced *Non-Volatile Memory* (NVM) shows performance closer to DRAM, while providing terabytes of capacity, consuming less power and having a better price per byte ratio.

Currently, applications have to be heavily modified to use heterogeneous memory, often relying on vendor-specific APIs and software. Expecting that more and more machines in HPC will incorporate heterogeneous memory, there is a need for a portable, vendor-neutral solution to expose available kinds of memory and allocate data on those at runtime. Tools like *memkind* [1] or *hwloc* [2] already provide an API to identify memory (e.g., high bandwidth, large capacity) and allocate data on it.

More questions arise that are not trivial to answer. As memory with high bandwidth is limited in capacity, how to decide which data items to put in which kind of memory? And is it reasonable to consider such allocations optimal for the whole application duration? Further, how can we collect knowledge about data items and make that available for decision making?

In project *H2M* (Heuristics for Heterogeneous Memory), we address these questions and challenges with the following approach. We review and extend methodologies to expose memory and associated memory characteristics. We present a runtime system that enables programmers to express hints about data and how to translate hints into memory placement decisions. Finally, we have been

studying ways to identify memory access characteristics for data items both at compile- and at run-time.

II. EXPOSING ENVIRONMENT AND CHARACTERISTICS

In a first step, we evolved the way memory tiers and their characteristics are exposed in *hwloc* to benefit from the additional information contained in ACPI tables. Although that information is useful, it might only include scalar values for e.g. achievable memory bandwidth and latency of a particular kind of memory. Further, HPC systems usually comprise several sockets with several cores. Consequently, the values might change for parallel applications running multiple threads and NUMA effects can have a strong impact on performance.

We analyzed the latency and achievable memory bandwidth on a dual-socket Intel Xeon Gold 6338 (*Ice Lake*) machine with classical DRAM as well as Optane DC Persistent Memory (NVM). There are clear differences in performance and scaling behavior depending on the memory kind and whether it is accessed from the local or remote socket. Moreover, results reveal how latency is changing when several threads access the memory and when part of the available memory bandwidth is saturated. To sustain satisfying performance, these aspects need to be taken into account later in the decision making process where to place data and when to move data items between memory kinds.

III. SUPPORT FOR HETEROGENEOUS MEMORY

Another challenge is that data items in applications might exhibit different accesses characteristics which additionally affect the performance. Characteristics include e.g. the frequency of accesses, whether the item is mostly read or written, whether it is accessed from multiple threads, and the access pattern (linear, strided, random). As memory with the highest bandwidth and lowest latency is limited in size, these insights can be useful to select the right kind of memory for a particular data item.

Several questions arise. How can a programmer express requirements/hints (also called *traits*) for data items and how can these traits be exploited for decision making? Further, what happens when access patterns change (e.g., several phases) during run-time? Finally, is it possible to identify access characteristics and generate trait recommendations for items?

This work was supported in part by the French National Research Agency (ANR) and the German Research Foundation (DFG) in the frame of the ANR-DFG H2M project (ANR-20-CE92-0022-01, DFG project number 446185093).

A. Runtime and Allocation Abstraction

Our approach is based on abstracting the memory allocation to allow programmers to specify traits regarding how data is used throughout the application run. It also features methods to update traits for existing data items. An example for such an abstraction is illustrated in Listing 1.

```
int err; double* data_it; size_t N; // size of data item
// specify allocation traits (requirements and hints)
alloc_trait_t traits[5] = {
    atk_access_mode, atv_access_mode_readwrite,
    atk_access_freq, 2000, /* e.g. accesses per sec */
    atk_access_origin, atv_access_origin_single_thread,
    atk_access_pattern, atv_access_pattern_strided,
    atk_access_stride, 4 /* in bytes */};
// allocate data item according to traits specified
data_it = (double*) alloc_w_traits(N, &err, 5, traits);
// or if situation changes -> update traits for items
err = update_traits(data_it, N, 5, traits);
// trigger data migration to respect new hints
err = apply_migration();
```

Listing 1: Allocation abstraction (shortened names). *Code under development but will be available on GitHub soon*

A runtime system then takes care of data allocation or movement by applying desired heuristics/strategies that deliver suitable memory locations for data items. Our runtime provides not only basic default strategies but also an interface to customize strategies and decision making.

B. Memory Access Analysis and Profiling

There are several ways to retrieve information about memory access characteristics. We are currently investigating three directions to gather knowledge in order to recommend allocation traits for data items:

Static Analysis: To a certain extent compilers can detect access patterns or structures in code during compilation.

Sampling: Similar to related works [3] we investigate whether a sampling approach can provide sufficient insight to form recommendations. We build on NumaMMA [4] and Precise Event Based Sampling (PEBS).

Compiler Instrumentation: Relying on the LLVM infrastructure and Sanitizer interfaces we track allocations via interception and memory accesses via compiler instrumentation. Despite higher overheads, it can provide detailed insight about memory accesses.

Based on recorded and static data we are able to generate trait recommendations for data items that can be injected into the runtime system.

IV. PRELIMINARY RESULTS

We conducted first experiments with a synthetic DGEMM benchmark that allows creating controlled scenarios to investigate the trade-off between costs for moving data to faster memory (or vice versa) at run-time and the performance benefits of higher bandwidth and lower latency. The benchmark consists of multiple phases. Each phase is responsible for solving a configurable number of memory-bound (non-blocked), naive DGEMM operations. Before each phase, traits for data buffers of the last

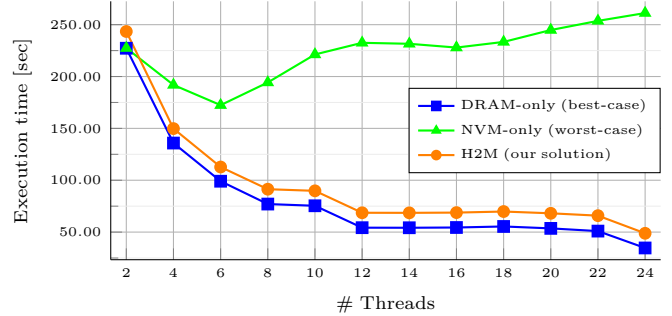


Figure 1: Performance comparison for a synthetic benchmark with multiple naive, memory-bound (non-blocked) DGEMM phases.

and upcoming phase are updated and data migration is triggered. Additionally, we compare against versions where all data stays in NVM (*worst-case*) or DRAM (*best-case*).

Despite current overhead for data movement, we demonstrate that our solution is able to beat the NVM baseline in most cases and gets close to DRAM-only performance as shown in Fig. 1.

V. CONCLUSION

In this work, we present *H2M*, a library for managing data placement and movement on heterogeneous memory systems. We demonstrate how allocation abstraction provides hints to the runtime system that can be exploited by strategies to determine suitable memory locations for data items. We discuss how static analysis and profiling techniques help to understand data access characteristics and to recommend allocation traits.

In our future work, we will develop performance models and heuristics based on the collected information to decide where and how to place data. Further, we investigate software prefetching mechanisms to overlap data movement between memory kinds with useful work. Finally, we will conduct an extensive evaluation with various benchmarks and applications on platforms such as KNL (HBM+DRAM) and Intel Optane (DRAM+NVM).

REFERENCES

- [1] C. Cantalupo, V. Venkatesan, J. R. Hammond, K. Czurylo, and S. Hammond, “User extensible heap manager for heterogeneous memory platforms and mixed memory policies,” *Architecture document*, 2015.
- [2] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, “hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications,” in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Feb. 2010.
- [3] K. Wu, Y. Huang, and D. Li, “Unimem: Runtime Data Management on Non-Volatile Memory-Based Heterogeneous Main Memory,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’17, New York, NY, USA, 2017.
- [4] F. Trahay, M. Selva, L. Morel, and K. Marquet, “NumaMMA: NUMA MeMory Analyzer,” in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018, New York, NY, USA, 2018.