Performance Prediction of Graph Analytics on Persistent Memory

Diego Moura*, Daniel Mossé†, Vinicius Petrucci*†
*Universidade Federal da Bahia, Brazil
†University of Pittsburgh, USA

I. Introduction

Considering a system with heterogeneous memory (DRAM and PMEM, in this case Intel Optane), the problem we address is to decide which application will be allocated on each type of resource. We built a model that estimates the impact of running the application on Intel Optane using performance counters from previous runs on DRAM. Using this model, we present an offline application placement for the context of heterogeneous memories. Our results show that judicious allocation can yield average reduction of 22% and 120% in *makespan* and *degradation* metrics respectively.

A. Intel Optane Technology

Rather than using Intel Optane in *Memory Mode*, where DRAM is a cache for Optane, we consider the *App Direct Mode*, where the user explicitly defines which part of the application will use the Optane memory. Our setup (in Linux 5.1) uses the Optane as a separate memory NUMA (Non Uniform Memory Access) node [1], as a slightly slower memory.

B. Performance of DRAM vs PMEM

Because of the importance of analyzing graph applications (e.g., social networks, maps, etc), GAPBS [2], [3] applications have become very popular. As motivation, we present the slowdown (execution time on PMEM normalized to the execution time on DRAM) for each applications/dataset in Figure 1; values above 1 denote degradation, that is, an increase in the execution time when executing the application in PMEM. The results are calculated using the average from 15 executions.

Since PMEM latency is higher than DRAM, performance using PMEM is worse in all cases. However, we can see that some applications suffer less, like *tc_webU, tc_roadU*, and others suffer more, like *bc_twitterU, pr_twitterU*. In a DRAM-constrained context, we advocate that applications with a higher level of degradation be allocated to DRAM while those with less degradation are allocated to PMEM. For this decision to be made, it is necessary to know the impact of allocating each application/dataset on each memory type.

C. Modeling/Predicting Slowdown

Collecting performance monitor counters (PMCs) through profiling tools such as *perf* has been shown to be a practical way to analyze application performance. This is because there is no need to instrument the application (therefore no need to have access to the source code) and because of its low monitoring overhead (usually through hardware and sampling).

Using PMCs collected while the application runs on DRAM, it is possible to estimate the performance if the application were to run on PMEM. For this, we use the following PMCs: *cycles*, *instructions*, *cas_count_read*, *cas_count_write*, *llc_misses.mem_read*, *llc_misses.mem_write*, *l2_lines_in.all*, *l2_trans.l2_wb* and *unc_m_pre_count.page_miss*. These PMCs were selected because they are considered generic and present in most Intel architectures. In addition, they bring relevant information to our context such as external communication (off core), while distinguishing between read and write operations, which are important when using the Intel Optane as they have different latency costs.

We developed a linear regression model that achieves an accuracy of 88% through Coefficient of Determination (R^2), as shown in Figure 2. We also explored more elaborate models such as neural networks and tree-based; the results are close to those obtained through linear regression. As linear regression is less expensive, we decided to use it.

II. Methodology and Evaluation

We use a similar methodology as presented in [4] using trace information, but differently, we explore performance counters instead of collecting traces of memory accesses.

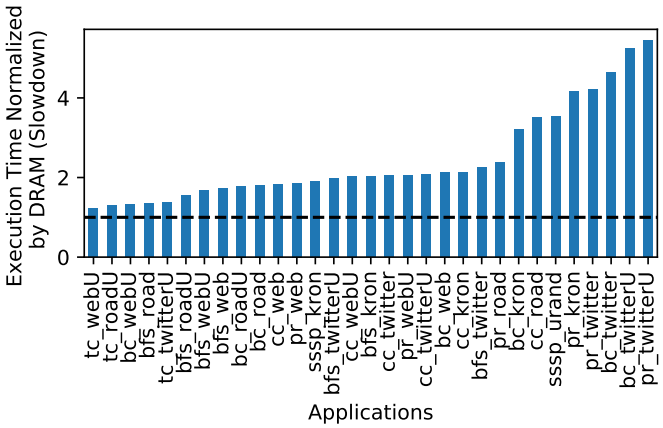

Fig. 1. Performance of GAPBS suite executing on PMEM-only, normalized by DRAM-only, for the different applications and datasets.

* Work done while Diego Braga was a visiting PhD Student at University of Pittsburgh.

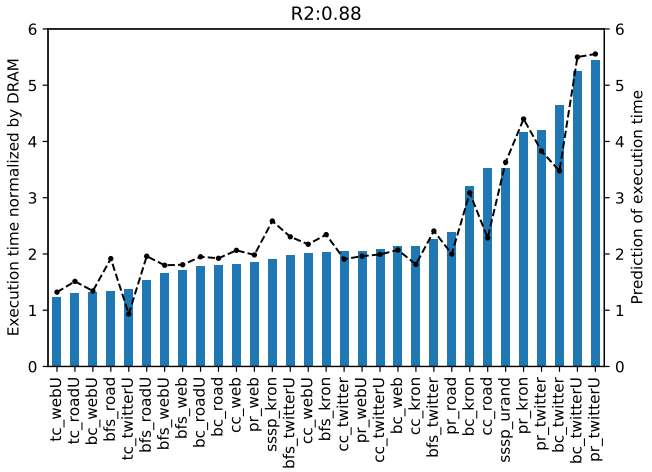

Fig. 2. Accuracy of slowdown prediction using PMCs collected when the application was executing on DRAM and expected to run on PMEM.

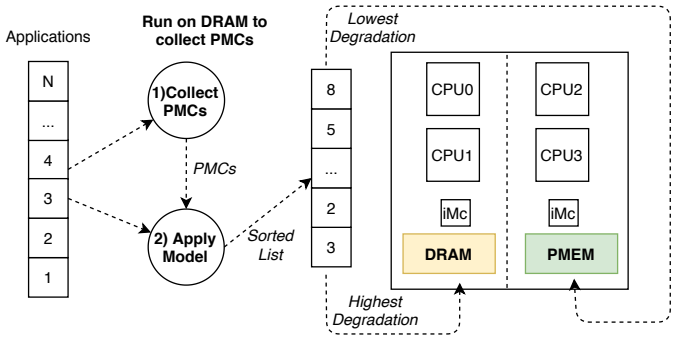

Fig. 3. Model-driven application scheduling in a heterogeneous memory context. A model predicts the performance degradation when applications are expected to be migrated to PMEM, and a scheduler uses this knowledge to map the applications with more (less) degradation to DRAM (PMEM).

A. Experimental Setup

We evaluate our approach using a 2-socket Intel server with 48 cores, 384 GB of RAM, and 3TB of Optane memory. The NVDIMMS are setup to use interleaved regions, which create contiguous physical address space and provides striped reads and writes for better throughput. To eliminate the NUMA effects, we run each application using only one socket. Hyper-threading is disabled. We tested four graph-based algorithms [2], [3]: Betweenness Centrality (BC), Breadth-First Search (BFS), Connected Components (CC) and PageRank (PR). We varied access patterns and size of data structures.

III. Simulations and Results

Given a typical scenario in cluster computing (a load balancer assigns applications to a server), Figure 3 illustrates a potential deployment scenario of our model on a server where there is a queue of applications to be executed on a machine with two memory resources: DRAM and PMEM. We assume all applications from Figure 1 have been executed once on DRAM and PMCs collected. Then, we use our model and create a new application queue sorted from highest to lowest performance degradation (that is, PMEM_exec_time/DRAM_exec_time). The applications are selected to run from both ends: the applications with lowest degradation are allocated to PMEM and highest degradation going to DRAM.

We simulate our model-driven application scheduling and compare it with an approach used by some cluster managers like SLURM [5] that manages a queue with pending jobs. We experiment with 1,000 random application lists/orders, using different random seeds, and we report the *makespan* (i.e., length of time to finish all applications) metric for each scheduling strategy and *degradation* (i.e., sum of the degradation experienced when the applications are executed in PMEM).

Figure 4 shows the results: scheduling using our model, the *average makespan* is reduced by 22% and *maximum makespan* is over 30%. In addition to reducing total execution time, our approach reduces degradation by more than double (up to 120%). Reducing the total degradation is important in preventing any application from taking much longer than it would take to run on DRAM.

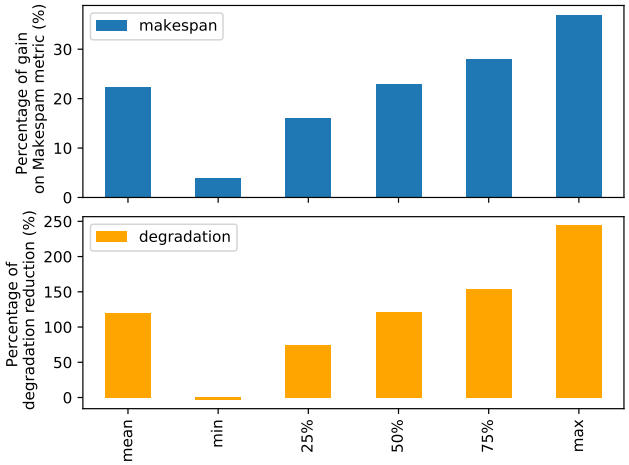

Fig. 4. Simulation results of the *makespan* and *degradation* improvement.

IV. Future Work

We plan to study dynamic memory allocation, that is, without prior knowledge of the application. We will also use real hardware for calculating makespan and degradation (rather than from the model only). Lastly, we will consider migrations.

References

[1] Jonathan Corbet. Persistent memory for transient data. https://lwn.net/Articles/777212/, 2019. Accessed: 2020-06-03.

[2] Scott Beamer, Krste Asanovic, and David A. Patterson. The GAP benchmark suite. *CoRR*, abs/1508.03619, 2015.

[3] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *(SPAA)*, 2018.

[4] Lei Zhang, Reza Karimi, Irfan Ahmad, and Ymir Vigfusson. Optimal data placement for heterogeneous cache, memory, and storage systems. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(1), May 2020.

[5] Slurm workload manager. https://slurm.schedmd.com/. Accessed: 2020-12-12, 2020.