Qeios

Peer Review

Review of: "Performant Automatic BLAS Offloading on Unified Memory Architecture with OpenMP First-Touch Style Data Movement"

Petros Anastasiadis[1]

1. Electrical and Computer Engineering, National Technical University of Athens, Greece

The paper introduces SCILIB-Accel, a tool designed for automatic BLAS offloading on NVIDIA Grace-Hopper GPUs utilizing their unified memory architecture. By intercepting CPU BLAS calls through Dynamic Binary Instrumentation (DBI), it enables the execution of legacy code using CPU BLAS without modifications. Then, it evaluates three data movement strategies: Mem-Copy, Counter-Based Migration, and the proposed Device First-Use, which migrates data pages to GPU memory upon first access, utilizing the NUMA hierarchy of Grace-Hopper chips, and retains them for subsequent uses. Performance evaluations using the MuST and PARSEC applications demonstrate that SCILIB-Accel, using the Device First-Use policy, outperforms native CUDA ports and achieves up to a 3× speedup over CPU execution (in a 2X Grace CPU server).

In general, this study provides valuable insights into the backend performance characteristics of NVIDIA Grace-Hopper's unified memory architecture, detailing access bandwidths from the CPU and GPU to both CPU memory and GPU High-Bandwidth Memory (HBM). This information is interesting in the context of understanding and optimizing data movement in high-performance computing applications. Additionally, the paper targets a real problem: automatically intercepting and offloading BLAS calls without requiring code modifications or recompilation significantly simplifies the porting process for legacy codebases, which is often a labor-intensive endeavor. Finally, by comparing data movement strategies—namely Mem-Copy, Counter-Based Migration, and the proposed Device First-Use—the paper provides practical guidance for scientists aiming to optimize performance in unified memory systems for other applications as well.

doi.org/10.32388/NXU60O

However, this paper has some major shortcomings that must be fixed to improve the quality of the publication. I list these in descending order of importance:

First, the paper suffers from numerous grammatical and vocabulary errors, and its data presentation is suboptimal. For instance, certain tables would be more effective as figures, some textual information could be better conveyed in tables, and the captions for tables and figures lack sufficient detail. These issues significantly impair readability, which is particularly problematic for a journal publication exceeding 20 pages. To address this, I recommend the following changes to improve the paper's readability and overall quality.

1. **Proofreading and Language Enhancement:** Utilize spellcheck and grammar tools to identify and correct errors. Additionally, employing a language-improvement tool can enhance clarity and coherence.
2. **Data Presentation Optimization:** Convert the system and benchmark characteristics description (Section 4.1, page ~14) into a table format for better readability.
3. **Enhanced Captions:** Expand table and figure captions to 1-2 sentences that clearly explain the content and significance of each corresponding visual element.

Additionally, the literature review in the paper is notably lacking in coverage of existing work on offloading BLAS operations to GPUs from legacy CPU codebases. In addition to **cuBLASXt/NVBLAS,** offloading a problem from CPU memory + LAPACK layout to GPUs has been a focus of a lot of previous work, such as in **Qilin, MAGMA, SuperMatrix, StarPU, PaRSEC** (older approaches, some with differences in data layout), with the most recent being **BLASX**, **XKBLAS**, and **CoCoPeLia/PARALiA/Uncut-GEMMs**. These approaches primarily utilize memory copy offloading and implement optimizations such as domain decomposition, tiling, overlap, and task scheduling that significantly improve performance and may surpass the page-based strategies proposed in this study. Although the current work's contribution of automated interception and conversion of CPU BLAS calls to GPU operations is valuable regardless, acknowledging these prior efforts would provide a more comprehensive context. Specifically for **XKBLAS,** their implementation offers a complete suite of BLAS routines and supports ***LD_PRELOAD*** for automatic replacement of CPU BLAS calls, so it should be both in the literature review and the evaluation section of the paper.

In a similar direction to the above, I would like some clarifications on the mem-copy approach, which basically treats the Grace-Hopper system as an older CPU-GPU system with discrete memories. My

understanding from Listing 1 is that the author allocates GPU buffers, copies data from the CPU before every BLAS call, and copies back the results afterwards (not sure about deallocation? If the buffers are not reused, it should be done at every invocation). This is expected to be worse than page migration since it is extremely inefficient (see BLASX, XKBLAS, PARALiA) and should be done with a GPU software buffer/heap that is reused to avoid alloc/dealloc entirely. The same is true for data copies: to achieve performance, you must keep the data in the GPU for subsequent invocations (in the same way that device first-touch does) as long as the CPU does not interfere during iterative execution. On top of these, decomposing the domain and adding communication/computation overlap can further improve performance (see BLASx, CoCoPeLia). While all these diverge from the simple "drop-in replacement" that this work offers, it should at least be discussed as a programmability-performance tradeoff and by using **XKBLAS** as a baseline implementation for the mem-copy approach (it should work on Hopper).

Finally, the evaluation section of the paper could be improved. Firstly, converting Tables 3, 4, and 5 into graphical representations would enhance data visualization, making it easier for readers to interpret the results. Secondly, reporting performance metrics solely in terms of execution time is less informative than using metrics like **FLOPS**, which allow for comparisons against theoretical peak performances. For instance, the Grace CPU in the Grace-Grace server (S1) has a peak FP64 performance of approximately 7.1 TFLOPs, while the Hopper GPU in system S2 offers up to 34 TFLOPs (or 67 TFLOPs with tensor cores, which should be utilized automatically by underlying cuBLASDgemm calls). While I am not sure how these translate to zgemm/ztrsm performance, the reported ~3x performance improvement from S1 to S2 warrants a more detailed analysis to understand the underlying factors contributing to this difference.

Declarations

Potential competing interests: No potential competing interests to declare.