Evaluating Hybrid Memory Cube Infrastructure To Support High-performance Sparse Algorithms

Kartikay Garg
Georgia Institute of Technology
Atlanta, Georgia
garg.kartikay@gatech.edu

Jeffrey Young
Georgia Institute of Technology
Atlanta, Georgia
jyoung9@gatech.edu

CCS Concepts •**Hardware → Dynamic memory;** *High-level and register-transfer level synthesis;* •**Computing methodologies → Linear algebra algorithms;**

Keywords PIM, Stacked DRAM, Hybrid Memory Cube (HMC), HPC, SuperLU, FPGA, Heterogeneous Systems

ACM Reference format:
Kartikay Garg and Jeffrey Young. 2017. Evaluating Hybrid Memory Cube Infrastructure To Support High-performance Sparse Algorithms. In *Proceedings of MEMSYS 2017, Alexandria, VA, USA, October 2–5, 2017,* 3 pages.
DOI: 10.1145/3132402.3132435

1 Introduction

This work is focused on analyzing potential performance improvements of HPC applications using stacked memories like the Hybrid Memory Cube, or HMC. We target a HPC sparse direct solver library, SuperLU [4], that performs *LU* decomposition and is a core piece of simulation codes like NIMROD [1]. To accelerate this library, we are interested in mapping both the computationally intense Spare Matrix-Vector (SpMV) kernels that can be implemented using matrix-matrix multiply (GEMM) calls and memory-intensive primitives like Scatter and Gather to a reconfigurable fabric tightly integrated with a 3D stacked memory. Here we provide initial results on mapping GEMM to OpenCL-based devices as well as a trace-driven evaluation of SuperLU's memory accesses with a combined FPGA and HMC platform.

1.1 Application Background - SuperLU

State of the art implementations for sparse direct solvers, like Piyush et al. [7] split the computations into logically independent and isolated phases. The algorithm schedules elementary BLAS operations such as matrix multiply (GEMM) on dense sub-blocks of a sparse matrix. These dense blocks are organized in a tree data structure known as an elimination tree, which can be used to identify independent blocks which may be issued for execution in parallel by both the host and the accelerator. Scheduling a chunk of computations from an independent iteration's phase helps in hiding the memory access latency for requests issued by the host. Lower access latency and increased memory bandwidth helps to alleviate the computation workload on the host, and allows for the transfer of more computation to the accelerator. The high internal bandwidth in a 3D-stacked memory, such as HMC, can be used to improve access times especially for random accesses employed by the Scatter and Gather phases in each iteration.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MEMSYS 2017, Alexandria, VA, USA
© 2017 ACM. 978-1-4503-5335-9/17/10…$15.00
DOI: 10.1145/3132402.3132435

2 Development Approach

Our work thus far has focused on two key pieces to support SuperLU execution and optimization on an FPGA and HMC platform: 1) Migration of core kernels to OpenCL and 2) trace-driven analysis of a multi-threaded, representative memory workload with the test FPGA and HMC platform.

2.1 Migration to OpenCL

For our target application, we have developed an OpenCL "proxy" kernel that implements the Halo optimization for SuperLU [7], which provides an optimized overlapping of computation and data movement for SuperLU's core *LU* decomposition kernel, the Schur Complement. OpenCL gives usversatility across platforms and it can be compiled with newer compilers High-Level Synthesis (HLS) that support FPGA hardware. We employ the clBLAS library for efficient DGEMM calls to offload computations to the accelerator. By using a dense matrix as the input for the proxy, we maximize the number of memory access requests issued by the host, as compared to the data index manipulation overheads in case of a sparse matrix. This worst case scenario model for testing memory accesses provides a counterpoint to the competing baseline model of a GPU accelerator, which can usually hide latency with data parallelism. An analysis of this worst case scenario also helps us to make the best argument for further exploring memory architecture models like near-memory processing with FPGAs or processing in memory (PIM).

2.2 Generating Memory Trace

Since our FPGA+HMC platform currently has limited support for compiling and running BLAS operations that are critical to our proxy application, we generate memory traces to estimate the response of the HMC memory system. The span of memory requests which are of interest to us are spread out within the execution time because DGEMM calls are distributed across multiple iterations of the LU factorization algorithm. Likewise, a simulation framework modeling the host and the accelerator pipelines through memory stages would simply take too long to simulate the entire program. Thus we dynamically instrument the executable binary with the PIN tool and run our proxy kernel on an Intel CPU which supports multi-threaded implementation for OpenCL runtime calls. An Intel Core(TM) i7-4790K CPU processor clocked at 4.00GHz is used for our experiments.

In order to accurately capture the memory access stream of the DGEMM BLAS calls offloaded to the accelerator, we generate custom PIN tool filters that only trace memory accesses within

BLAS regions. We then trigger instruction level instrumentation for only the period of execution of the proxy OpenCL kernel.

3 Evaluation Platform

3.1 Hardware

For the baseline performance numbers of our Halo implementation, we use an Intel CPU and NVIDIA K40c platform. For the FPGA+HMC platform we use the AC-510 evaluation board by Micron/PicoComputing. The AC-510 [5] features aHMC Consortium Specification 1.1 cube connected to a Xilinx KU060 FPGA. This combined module is connected to the host via PCIe. The vendor firmware on the FPGA exposes 9 stream interfaces to the host software API stack. We use this PicoStream framework (provided by the vendor) [6] to issue requests from the host software to the AXI stream buffers of a port on board the FPGA. The 9 ports drain the requests from their command streams to theHMC controller IP on board the FPGA, which createsHMC command packets and forwards the requests in a batch to theHMC.

3.2 Software for Trace-driven Execution

We use the vendor provided application "GUPS" (Giga Updates Per Second) as a starting point for our trace-driven evaluation on the AC-510 platfom. GUPS is modified to support a multi-threaded trace driven execution where addresses and data are enqueued from the host and then sent over PCIe to the FPGA and HMC module. The application uses hardware registers (local to each stream) implemented on the FPGA, which probe different performance metrics of the HMC that are measured using counters at the FPGA to HMC serialization interface. These include Giga Operations per Second (GOPS), read/write request count, and total time to service memory trace.

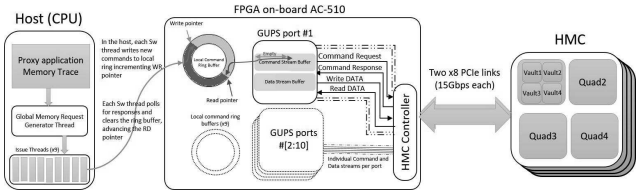
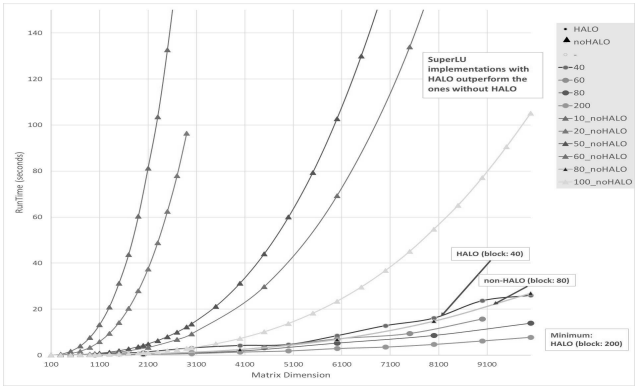


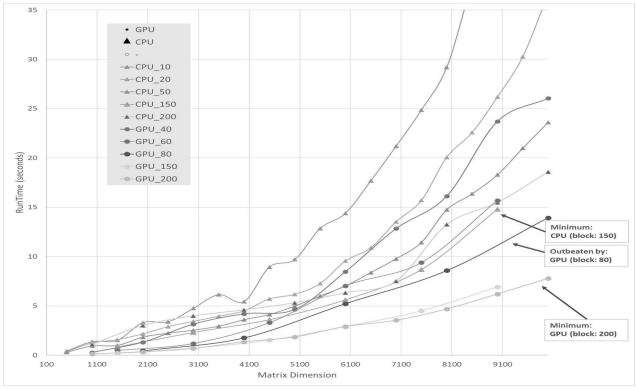
Figure 1. Updated GUPS design to measure multi-port performance of the HMC. Each software thread controls access to the command and data streams of respective GUPS user ports and issues batches of commands as flits are available.

On the host, nine independent threads are spawned. Each thread monitors attributes of a distinct GUPS port and the local command and data streams. A ring buffer local to each thread stores the list of commands allocated to the respective port to be issued to the HMC. Each thread reads from the ring buffer, starting at the read pointer, and issues a batch of commands to the HMC controller IP. This continues till the read pointer catches up with the write pointer, and then the thread is idle, unless the memory trace has been exhausted. In that case, the thread exits.

Another producer thread on the host parses the memory trace file and creates HMC command and data packets, and allocates them to each port ring buffers in a round robin order. This thread has the responsibility of remapping the host address space to the HMC address space.



(a) Performance comparison of SuperLU kernel with HALO vs. SuperLU without HALO on NVIDIA Tesla K40c GPU.



(b) Performance comparison of HALO kernel on CPU (Intel 4790K) and GPU (Tesla K40c) platforms.

Figure 2. Profiling of different phases of Halo

4 Results

4.1 Halo Measurements for CPU and GPU

We first investigate the effects of block size, matrix dimension, and the HALO optimization versus a baseline (ie non-HALO) implementation, as shown in Figure 2. We sweep a wide range of values for matrix dimension size (N), where N varies from 500 up to 10, 000 in steps of 1, 000. We sweep the block size (M) from 50 to 200, in steps of 20. A careful analysis confirms our understanding wherein data transfer overheads dominate lower matrix dimension and smaller block size test cases; this situation matches when GPU occupancy is low. As matrix dimensions are increased the GPU becomes better utilized and overall runtime is decreased. These results mostly confirm that dense, accelerated GEMM operations are limited by data transfer and that using too small a block size can result in worse performance for a high-end GPU than a mid-grade CPU, as is the case for GPU, block size 40 versus CPU, block size 150 in Figure 2b.

4.2 Memory Trace Experiments

We next study the effect of FPGA firmware design decisions on the performance of our target kernel on the 3D-stacked memory system. A detailed report on the results from runs of the "proxy" kernel on different platforms and measurements from the memory trace simulation on the FPGA+HMC evaluation board have

been presented in related work [2]. By varying the batch size of read/write commands, we are able to understand the optimum queue size for the host thread that is contending for access to the common HMC link. If we reduce the batch size to 1, we can study the implications of issuing requests in-order one by one to the memory system. These variations in dispatching memory requests to the HMC platform help us to evaluate expected performance for a CPU-style processing core that makes blocking memory access calls. Alternatively, by issuing commands serially on a single user port, we can simulate memory contention between threads. Note that while nine ports are available to interact with the FPGA and HMC, our trace-driven simulation uses a maximum of seven ports.

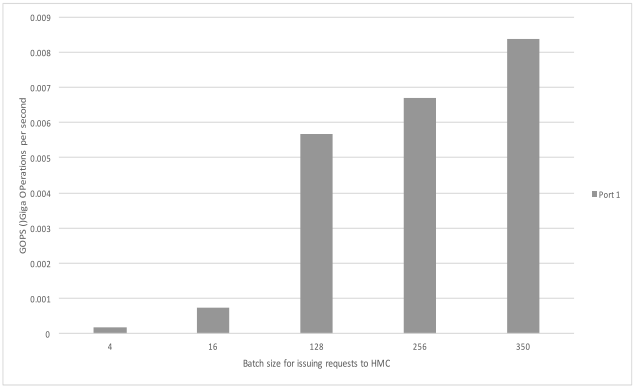


Figure 3. Aggregate GOPS for Serial Issue

Figure 3 shows the measured performance of our proxy kernel using the serial issue implementation with one port. In general, increasing batch sizes correlate with aggregate port throughput increases and reduced service times, with an improvement of almost 8x Giga-Operations per Second (GOPS) when scaling from batch size of 16 to 250.

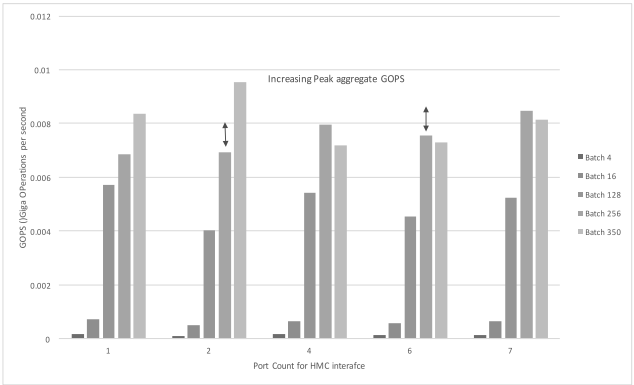


Figure 4. Aggregate GOPS for Parallel Ports

However, for multiple ports contending for the HMC link in parallel (Figure 4), the observed performance is optimal for a batch size of 256. When the software issue batch size is 256, it also matches the length of the command queue FIFO depth in hardware, which reduces overheads to poll for read responses. Taking these two preliminary results together indicates that the HMC interface works best with larger batch sizes but that larger batches must be matched with longer queues in hardware to reduce overheads.

5 Future Work

While these results presented so far are preliminary, we plan to further investigate memory access patterns for SuperLU with the HMC platform. Some challenges we have observed thus far include the following:

1. Currently, HLS firmware packages have limited support for interacting with unusual memory types like HMC. We envision a future OpenCL-based evaluation that can utilize either a wider interface to the HMC or multiple ports as with the GUPS module.
2. Tunable BLAS calls are still not well-supported with FPGA implementations. For this reason, we are currently testing with custom GEMM calls that replicate functionality available in libraries like clBLAS.
3. GPU-based memory traces would reveal better dependency and consistency information and allow for testing out BSP-based accelerators that can be more easily compared to previous GPU and Xeon Phi implementations. Currently PIN does not fully support iGPU trace generation

By focusing on solutions to each of these issues, we hope to further develop and evaluate our SuperLU implementation for FPGA and HMC-based platforms.

6 Conclusion

This work provides a detailed testing of an OpenCL implementation of a core SuperLU proxy kernel with Halo on CPU, GPU, and HMC and FPGA. While we find that the dense GEMM proxy kernel scales as batch size is increased with Halo, the trace-driven simulation shows that a higher batch size of memory accesses to the HMC may not guarantee peak performance since FPGA user ports have to contend for the shared HMC link resource. We conclude that memory accesses for SuperLU will likely need more careful optimizations like those presented in related work [3] as well as further hardware support to match bursty read/write accesses with available HMC link resources.

References

[1] David Abramson, Colin Enticott, and Ilkay Altinas. 2008. Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC '08)*. Article 24, 11 pages.

[2] Kartikay Garg. 2017. *Near-memory Primitive Support and Infrastructure for Sparse Algorithms.* Master's thesis. Georgia Institute of Technology, Atlanta, GA, USA. https://smartech.gatech.edu/handle/1853/58343

[3] Maya Gokhale, Scott Lloyd, and Chris Hajas. 2015. Near Memory Data Structure Rearrangement. In *Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS '15)*. ACM, New York, NY, USA, 283–290. https://doi.org/10.1145/2818950.2818986

[4] Xiaoye S. Li and James W. Demmel. 2003. SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. *ACM Trans. Mathematical Software* 29, 2 (June 2003), 110–140.

[5] Micron. 2014. AC-510 HMC accelerator board product brief. http://picocomputing.com/ac-510-superprocessor-module/. (November 2014). Retrieved 2016-11-05.

[6] Micron. 2014. Pico-Computing software stack and driver framework. http://picocomputing.com/products/framework/. (November 2014). Retrieved 2016-11-05.

[7] Piyush Sao, Xing Liu, Richard Vuduc, and Xiaoye Li. 2015. A sparse direct solver for distributed memory xeon phi-accelerated systems. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 71–81.