Emulating In-Memory Data Reorganization for HPC Applications

Chris Hajas, Scott Lloyd, Maya Gokhale

Lawrence Livermore National Lab, University of Florida

Abstract

High bandwidth memory architectures such as the Hybrid Memory Cube (HMC) will contain a logic layer below the DRAM to aid memory transactions. We propose adding a Data Reorganization Engine (DRE) to the logic layer to reduce memory latency in data intensive applications. We modified and tested Berkley's Sparse-Matrix, Dense-Vector multiplication (SpMV) application and found over 1.4x speedup using our in-house DRE emulation framework.

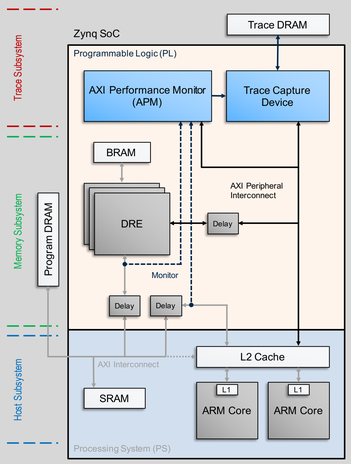
Background and Framework

Hybrid Memory Cube:

- Memory package containing stacked DRAM on top of logic layer for memory intensive computing.
- Currently on some FPGAs and Intel's Knight's Landing; shows promise for use on next-gen systems as we move toward Exascale
- Logic layer is not reconfigurable, which makes experimenting with logic functions difficult

Emulation Framework:

- Zynq SoC with ARM + FPGA
- Application runs on hard core and makes calls to DRE in programmable logic (PL)
- DRE creates a new view by sending messages using DMA to fetch data according to stride or indexed pattern specified by application
- Trace subsystem monitors hardware counters to measure performance and model energy



Case Study: Accelerating Sparse Matrix-Vector Multiply (SpMV)

What is SpMV? SpMV [1] is the multiplication of a sparse matrix in Compressed Storage Row format (CSR) by a dense vector. SpMV is commonly used to solve large scale linear systems in many scientific applications. These sparse matrices range in size from hundreds to millions of rows/columns and are typically banded.

How does it work? CSR format only stores non-zero elements and the associated column of each element. This data structure can easily be streamed through and will take advantage of the cache. However, the dense vector cannot! Each multiplication will access the value in the dense vector pointed by the column index array, which will be very irregular, making this application memory latency bound.

$$\begin{bmatrix} 2 & 4 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 3 & 0 & 9 \end{bmatrix} \times \begin{bmatrix} 3 \\ 7 \\ 1 \\ 4 \end{bmatrix}$$

Matrix Values: [2, 4, 3, 7, 3, 9]
Row Start: [0, 2, 3, 4]

Col Index: [0, 1, 1, 2, 1, 3]

Dense Vector: [3, 7, 1, 4]

View of dense vector after DRE: [3, 7, 7, 1, 7, 4]

Is it likely to benefit from the DRE?

1. Bulk of runtime spent accessing DRAM ☑
2. Accesses to memory are irregular ☑ **Yes!**
3. Disjoint memory accesses can be coalesced ☑

How is the application modified to use the DRE? Pointers to the column index array and dense vector array are sent to the DRE in PL. The DRE traverses the arrays and uses DMA logic to set up a view in SRAM of the dense vector in the order it will be accessed. The application code is modified by defining the access pattern in an array and using the **setup** and **fill** functions to create the view. Cache coherency must be manually managed due to the DMA in the logic. The program then operates on the returned cache-friendly contiguous block of data.

setup: Sets up DRE with source array and index array.

dre.setup(src_array, sizeof(dbl), idx_array, idx_size)

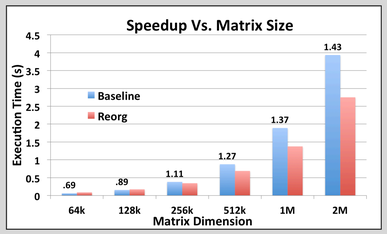
fill: Fills view buffer with coalesced memory accesses according to pattern defined in setup.

```
cache_flush ()
dre.fill(view_ptr, view_size, view_offset)
cache_invalidate()
```

How is this better? Instead of incurring a cache miss and high DRAM latency on every multiplication, the dense vector can be streamed through and uses every value in the cache line. Cache misses are significantly reduced, improving program performance considerably.

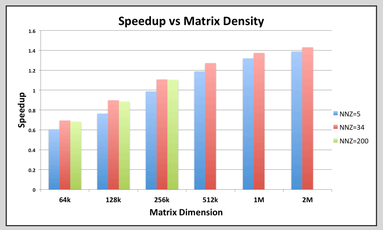
What's the catch? Only certain applications will see performance gain with DRE. There is also some overhead with DRE calls, the code may require significant refactoring to take full advantage of the DRE, and cache coherency must be manually managed.

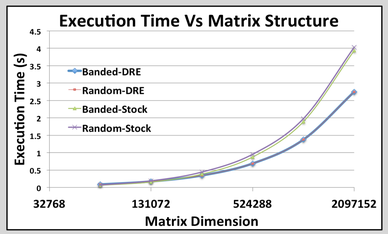
Results and Analysis



Reorganization is only beneficial for matrices greater than 2^{17} x 2^{17} on this architecture

Lower density matrices see lower speedup due to the smaller problem size. Some higher density matrices see reduced speedup due to advantageous caching in the baseline version.



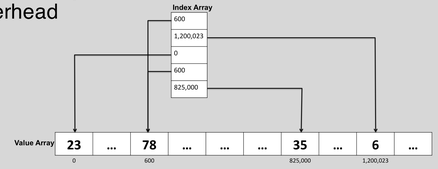


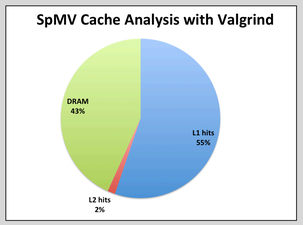
The "bandedness" of the matrix has no effect on performance with the DRE

Application Amenability to the DRE

Applications should have:

1. Many memory accesses that take a significant portion of total program runtime
2. Irregular, non-predictable memory accesses that blow cache
3. The ability to coalesce the accessed locations to minimize overhead





A quick check through Valgrind shows that SpMV is not cache friendly. Over 40% of memory accesses miss cache and go to DRAM, making this a good candidate for reorganization.

Future Work

- 64-bit emulator with larger DRAM and updated SIMD unit for more realistic emulation
- Additional applications/architectures to demonstrate the applicability of the DRE
- Compiler support to automatically modify an application to use the DRE

Conclusion

- DRE shows significant promise for use within HMC logic layers
- The SpMV app is highly amenable to the DRE, with speedup over 1.4 on our emulator
- Only applications and code with specific memory access characteristics will benefit from the DRE

References

[1] H. Gahvari, M. Hoemmen, J. Demmel, and K. Yelick. Benchmarking sparse matrix-vector multiply in five minutes. In SPEC Benchmark Workshop (January 2007), 2007.

[2] Lloyd, Scott, and Maya Gokhale. "In-Memory Data Rearrangement for Irregular, Data-Intensive Computing." *Computer* 8 (2015): 18-25.

[3] R. Nair, S. Antao, C. Bertolli, P. Bose, et al. Active Memory Cube: A processing-in-memory architecture for exascale systems. IBM Journal of Research and Development, 59(2/3):17:1–17:14, March-May 2015.

[4] Yuan, E., Zhang, Y. Q., & Sun, X. (2008, September). Memory Access Complexity Analysis of SpMV in RAM (h) Model. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on* (pp. 913-920). IEEE.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-675559