# Improving Performance of DDR and HBM Memories

PI: Sriram Vishwanath

Members: Zhengpeng Hu, Casen Hunger, Hardik Jain

Consultant: Ankit Singh Rawat

June 22, 2015

**Abstract**

This project will explore the use of coding theory in DDR and high bandwidth memory (HBM) system. Conventional DDR systems use timing optimization techniques around DDR protocol to improve the efficiency of accesses. In this work we propose increasing redundancy to help distribute access across dram pages. This will help in efficient retrieval of data. These schemes will be evaluated and optimized with several iterations. The aim is to find the best performing code design scheme given the performance requirement compared to alternatives.

## Problem Statement

This project will present solution to two main problems in computing systems. First is low access efficiency for memory read and write. The gap between the memory access speed and CPU speed is widening more than before [1]. The efficient integration of multi-cores combined with high speed architectures increases computation capability. However, slow memory fails to keep up with the request from the cores delaying the overall processing speed. The access latency to the shared memory increases due to contention among requests from computing cores. This results in formation of large access request queues waiting to be served by the slow shared memory. The current solutions to these problems in DDR environment is limited to improving command scheduling and clever data addressing. This project will develop a different solution where by introducing compressed redundancy in data, memory accesses are spread across banks. Our unique approach uses ideas from coding theory to achieve optimal compressing and distribution. We will develop this solution for general memory architecture and implement it for DDR and High Bandwidth Memory (HBM) protocols.

## Proposed Research

In conventional DDR memory system, consecutive memory accesses to different row of one bank require the memory to first PRECHARGE the existing open row and then ACTIVATE 1the required row. The PRECHARGE and ACTIVATE requires 10 cycles compared to 2 cycles for READ or WRITE to an existing open row. Hence there is a need to reduce the PRECHARGE and ACTIVATE cycles during a program. Currently this is done by using column decoders in DRAM’s where when a row is activated, a page amount of data is available in the memory buffers. The column decoders driven by the memory address decides the block of data in this page to be transferred to the requesting core. Once an access is requested to a different page, it triggers a PRECHARGE and ACTIVATE cycle to get that page in the buffer. To reduce the turnaround time of an access, there is a need to increase the page length available in the buffer. A naive solution is to increase the parallel memory DRAM chips. This will linearly increase the available data in buffer. However, in this project we will explore using coding theoretic solution where the cost of increasing the page length increases sub-linearly. From our previous experience developing this at Saltare Systems LLC with Futurewei during Summer 2014, the conclusion is an average increase of 30% access efficiency to the memory with a cost less than 15% [2]. The previous solution is perfect for SoC style memories which employ proprietary protocols and scheduling. This project’s focus will be to use similar principles and design memory coding for the popular DDR and upcoming HBM protocols. This analysis will give an insight on probability distribution function of access performance depending on input request distribution. It will also provide mathematical guarantee on the number of accesses that can be performed in best and worst case. Let’s take an example DRAM memory chip as shown in left figure 1. In this arrangement, we store the addition of data in bank m and bank n in bank o. We call bank o as parity bank. In the conventional memory system, two accesses to different rows of bank m are served sequentially requiring a pre-charge and activate. However, in the scheme described above, the first access is served using bank m and the other access is served using bank n and bank o. This facilitates the pre-charge and activation of rows in parallel and saves significant clock cycles. This simple example certifies the increase in access efficiency by introducing parity. We take this further and design a memory system where the parities are stored in efficient ways to maximize parallelism. The simple example of 2 bank scenario is extended to 8 banks by increasing the number of banks holding the parity. We will also investigate the impact of changing the locality of the code i.e. the number of data elements used to create a parity data element. In the above example, two data elements from bank m and bank n were used to create a data element of bank o. It is possible for more banks to contribute in creation of parity data however, it will result in less memory overhead required to store the parity data. It will also increase the availability of a particular data element in multiple parity data. However, on flip side, it makes the decoding logic complex requiring more data elements to decode a particular data from parity. Increasing the locality helps in certain scenarios where there are burst access patterns to memory. For example, in case of a burst access to 4 consecutive data elements, 3 of the data can be served from the original data elements and the last one can be served using parity. The parity data in memory system increases its efficiency compared to conventional systems. Moreover, this efficiency can be increased by enhancing the access scheduler responsible for scheduling accesses to the memory. The improved access scheduler will look through the pending requests in the queues and form optimized patterns which serve maximum requests.

## Research Plan

Our main target for this effort is to develop an optimized memory controller design to solve the problem of access efficiency. We will achieve this target by breaking down our effort in to six phases.

Phase I: **Baseline Characterization** (*1st Sep 2015-15th Jan 2016*): In phase I, we will develop functional and performance metrics of memory code design and controller algorithm optimization. These metrics will help structurally define the problem and develop key evaluation parameters. As an example, these metrics will be applied on our previous code designs. An examination of the problem in regeneration of codes during edits and writes to data memory will also be taken up during this phase.

Phase II: **Code Design** (*16th Jan 2016-15th May 2016*): This phase will focus on theoretical code design for DDR and HBM system. The designs will address two key problems in memory system: reducing the memory access latency in general/specific memory system and improve error correction capability of memory system. These designs would include optimization of memory scheduler. The designs will be developed with theoretical and analytical basis to their benefits and cost parameters.

Phase III: **Design Analysis and Optimization** (*16th May 2016-31st August 2016*): During this phase, our focus will be to derive cost-benefit index using a high level model in Matlab/Python. This will model the memory controller depicting scheduler, input/output arbitrator and reorder buffer. Using this model we will iterate and improve our design based on the requirements. This will serve as a high level evaluation of our design schemes and act as a pseudo algorithm for further implementation. This step will conclude with finding a design which meets all the performance criterion with the least cost.

Phase IV: (*1st Sep. 2016–15th Jan 2017*)D The general model developed in Phase II and Phase III can be enhanced and optimized for specific access patterns in custom application specific hardware systems e.g. network processors etc.

V(16th Jan 2017-15th May 2017): V This will be followed by improvement in code designs and scheduler algorithms based on the access models.

Phase VI: **Further Refinement and Publications** (*16th May 2017–31st August 2017*): During this phase, we will refine our findings and document it for publication and patenting.

## Cost Breakdown

|  |  |  |
| --- | --- | --- |
| **GSRA Costs:** | | |
| Effective # of GSRAs | 1.667 |  |
| GSRA Stipend/month | 2000 |  |
| GSRA fringe cost/month | 778 |  |
| GSRA Stipend + Fringe/month | 2778 |  |
| Net # months: GSRA #1 | 12 |  |
| Net # months: GSRA #2 | 4 |  |
| Net # months: GSRA #3 | 4 |  |
| Net Stipend | 55560 |  |
| Tuition & fees/2 sem | 11500 |  |
| Tuition factor: GSRA #1 | 1 |  |
| Tuition factor: GSRA #2 | 0.33333 |  |
| Tuition factor: GSRA #3 | 0.33333 |  |
| Net tuition/ 2 sem | 19167 |  |
| Net cost of GSRAs | 74727 |  |
| **PI Support Costs:** | | |
| PI Cost/month | 15020 | Includes fringe |
| PI # of months | 1 | Spread over life of project |
| Net cost for PI | 15020 |  |
| **Domestic Air Travel:** | | |
| Cost/domestic airplane trip | 1500 | To Conference or Futurewei |
| # of domestic airplane trips | 1 |  |
| Number of people/trip | 2 |  |
| Total for US airplane trips | 1500 |  |

|  |  |  |
| --- | --- | --- |
| **Totals:** | | |
| Net GSRA cost | 74727 |  |
| Net PI cost | 15020 |  |
| Net Travel cost | 1500 |  |
| Sub-total/year | 91247 |  |
| University fee% | 4.50% | For gift funding: non-gift fee is 60% |
| University fee | 4106 |  |
| Total per year | 95353 |  |
| Number of project years | 2 |  |
| Total over project years | 190706 |  |
| Finalized agreement | 190000 |  |

# References

|  |  |
| --- | --- |
| [1] | J. L. Hennesey and D. A. Patterson, Computer Architecture, A Quantitative Approach, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. |
| [2] | C. Hunger, H. Jain, A. Rawat and S. Vishwanath, "Dynamic Coding for Improved Performance of Memories," 2014. |