FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching

Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S. Kim, Juan Gómez Luna, Mohammad Sadrosadati, Nika Mansouri Ghiasi, Onur Mutlu

First presented at 53rd International Symposium on Microarchitecture (MICRO), Virtual, October 2020

Seminar in Computer Architecture
Presenter: Alfio Di Mauro
Mentors: Lois Orosa Nogueira, Konstantinos Kanellopoulos, Haocong Luo
Executive summary

- **Problem:** DRAM latency represents a performance bottleneck for a vast majority of applications
- **Goal:** Reduce DRAM latency via in-DRAM cache

- **Existing in-DRAM caches:**
  - **Key Idea:** Enhance DRAMs with fast regions (small) to implement caches
  - Coarse-grained (i.e. multi-kB) in-DRAM data relocation
  - Relocation latency depends on physical distance between slow and fast regions

- **FIGARO substrate:**
  - **Key Idea:** exploit the existing shared global row buffer to support data relocation among subarrays of a DRAM bank
  - Fine-grained (i.e. multi-byte) in-DRAM data relocation and distance independent relocation latency

- **FIGCache:**
  - **Key Idea:** cache only small, frequently-accessed portions of different DRAM rows in a designated region of the DRAM
  - Cache only the portion of a DRAM row that is expected to be accessed in the near future, and increase the row hit ratio

- **Evaluation:**
  - Improve average system performance by 16.3%
  - Reduce DRAM energy by 7.8% on average

- **Results**
  - FIGARO enables fine-grained data relocation in DRAMs at low cost
  - FIGCache outperforms state-of-the-art coarse-grained in-DRAM caches
Outline

• Background, Problems, & goal
• FIGARO substrate
• Fine-grained In-DRAM cache: FIGCache
• Results & Comparison with the SOA
• Summary and conclusion
• Discussion
Background

• Modern applications require to store large amount of data
• On-chip memories can not provide such storage capabilities
• We need to store application data on external memories
• Those memories are typically DRAMs, and they can store several tenths to several hundreds of GBs of data
Problem

• The need for bigger memories pushed DRAM manufacturer to fabricate denser DRAMs, therefore the cost per bit significantly decreased over time.

• However, the **access latency** did NOT decrease at the same pace

• Big **DRAMs are characterized by high access latency**, which became the main bottleneck of many modern applications
Goal

**Reduce the DRAM access latency**

To achieve this goal, we can try to hide the high access latency by implementing in-DRAM caches:

- Exploit existing DRAM circuitry
- Memory reads are served mostly from the in-DRAM cache
- An in-DRAM cache controller relocates hot data to the in-DRAM cache
DRAM organization: chip

- Contains multiple banks
- All the banks operate in parallel
- Each bank can serve an independent request (load/store operation)
DRAM organization: bank

- 32-64 2-dimensional arrays of DRAM cells (subarrays)
- DRAM cells are organized in bitlines and wordlines
- Cell content is read by a sense amplifier

Subarrays can be faster or slower, depending on the length of the local bitlines, as well as the physical distance from the GRB.
DRAM operation: data access

Operations are issued by the memory controller. A data access is performed in multiple steps.

1. **ACTIVATE (row)**
2. **READ**
3. **PRECHARGE**

While the row is “active” the memory controller can issue multiple READ/WRITE commands (row buffer hit)
In-DRAM cache: existing designs

DRAM manufacturers often choose long bitline length to accommodate a large number of rows, thereby increasing the memory capacity.

To alleviate long DRAM latencies caused by such long bitlines, several works proposed DRAMs where slow and fast subarrays coexist.

Data can be relocated from slow subarrays to fast subarrays, depending on their hotness, similarly to what happens in common caches.
Existing (SOA) Heterogeneous subarray Based Designs
Tiered Latency (TL) DRAM [1]
• Divide the subarray into fast and slow segments
• Fast segments serve as in-DRAM cache

+ A row can be quickly copied via back-to-back activation operation

- Intrusive approach requiring inefficient transistors to interrupt the bitline

Existing (SOA) Heterogeneous Bank Based designs without data relocation support

CHARM [2]
- Heterogeneity is introduced within each bank
- Few fast subarray with short bitlines are placed near the chip IO

+ Fast subarrays maintain the same structure as traditional DRAM
- Data are statically allocated depending on program profiling information
- Data are moved through the memory channel

Existing (SOA) Heterogeneous Bank Based designs with bulk data relocation support

- Support In-DRAM data relocation
- Introduce a row of relocation cells

+ low relocation latency
- Large and fixed relocation granularity
- Relocation latency depends on the physical distance from the fast subarray

Existing (SOA) In-DRAM cache: observations

• SOA In-DRAM cache approaches are inefficient because of the very coarse data granularity (~8kB in a single row)

• Interference between different processes, causing the in-DRAM cache to be evicted relatively soon from the fast buffers, thereby resulting in poor data reuse.

• Data relocation has its own latency and energy overhead, the cost is higher for slow subarrays that are far from the fast subarrays.
Outline

• Background, Problems, & goal
• FIGARO substrate
• Fine-grained In-DRAM cache: FIGCache
• Results & Comparison with the SOA
• Summary and conclusion
• Discussion
FIGARO

Key Ideas:

1. FIGARO is a new substrate that enable fine granularity data relocation across subarrays in a bank at a distance independent latency

2. It allows to cache only a portion of the DRAM row. It uses the global buffer in a bank to transfer unaligned column data from a local buffer to another

3. It does not require data to travel through the memory channel
FIGARO: working principle

Transferring data between two local row buffers

1. ACTIVATE (A)
2. RELOC
3. ACTIVATE (B)
4. PRECHARGE
DRAM architecture enhancement to support FIGARO

To be able to implement FIGARO, we need to implement 2 key modification to the DRAM architecture:

1. Support for 2 simultaneous row activation
2. Support for 2 simultaneous column decoding
Simultaneous row activation

Subarray A (Source)

Subarray B (Destination)

1. ACTIVATE (A)
2. RELOC
3. ACTIVATE (B)
4. PRECHARGE
Simultaneous row activation
Multiple column decoding

1. ACTIVATE (A)
2. RELOC
3. ACTIVATE (B)
4. PRECHARGE
Multiple column decoding

1. ACTIVATE (A)
2. RELOC
3. ACTIVATE (B)
4. PRECHARGE
FIGARO: where does the lower latency originate from?

• The Latency of the **RELOC** operation depends mostly on the sensing of the GRB and the driving of the destination LRB

• The latency of the **RELOC** operation is determined by the latency of relocating data between two subarrays that are the furthest away from the GRB (similar to **READ/WRITE** operations)
Outline

• Background, Problems, & goal
• FIGARO substrate
• **Fine-grained In-DRAM cache: FIGCache**
• Results & Comparison with the SOA
• Summary and conclusion
• Discussion
FIGCache: Fine-Grained in-DRAM cache is an in-DRAM caching mechanism based on FIGARO

**Key features:**

1. It increases the performance of in-DRAM caches as a single row can contain segments of “hot” rows. In-DRAM cache hit rate and row buffer hit rate increases substantially.
2. It simplifies the in-DRAM cache design, FIGCache has a distance independent relocation latency.
3. It does not necessarily require fast buffers to provide performance improvement, as it uses a smaller number of rows. More requests can be served with the same number of slow buffers.
FIGCache: caching mechanism

1. A row segment is brought into FIGCache to lower the latency for subsequent accesses to the same segment.

2. The memory controller needs to hold metadata about the cached segments to manage the cache.

3. The cache management relies on the FIGCache Tag Store (FTS), which stores information about the row segments that are brought into the cache.
FIGCache Tag Store (FTS)

The FTS contains 4 fields
1. A tag holding the original address of the row segment
2. A valid bit
3. A dirty bit, that is set to 1 if the request is a write
4. A benefit counter, that is incremented every time a request is a FIGCache hit
FIGCache line insertion policy

FIGCache relies on the *insert-any-miss* policy for handling cache misses

- Every FIGCache miss triggers a row segment relocation
- The goal is to achieve the highest utilization of the in-DRAM cache
- Neither additional circuitry, nor data utilization statistics are required for FIGCache to function
FIGCache line replacement policy

when a new segment needs to be inserted, and there are not free available segment, one row is selected as a candidate for eviction

1. The row is selected by using the cumulative benefit of all segments
2. The row with the lowest score is selected for eviction
3. all the segment are marked for eviction, the one with the lowest benefit counter is evicted
4. Segments remain marked for eviction after the segment replacement
FIGCache line replacement policy

Observations:

• By evicting all the segments of a row, **we can pack data that are accessed close in time to each other on the same cache row**

• This maximize the chance that a cache row will contain segments that are accessed together, thereby having a **higher row buffer hit rate**
Outline

• Background, Problems, & goal
• FIGARO substrate
• Fine-grained In-DRAM cache: FIGCache
• Results & Comparison with the SOA
• Summary and conclusion
• Discussion
Experimental methodology: setup

• FIGCache has been evaluated on a DRAM equipped with slow subarrays, and 2 fast subarrays per bank (512/32 rows respectively)

• Evaluations are based on a cycle-Accurate DRAM simulator (Ramulator [5]) coupled to a process simulator

• User level application traces have been collected using an architecture independent software instrumentation tool (PIN [6])

• Default row segment size has been set as 1/8th of the entire row

• Fast subarrays characterization was based on the open-source SPICE-MODEL developed in [7] (activation and pre-charge times are reduced by ~40% )

Experimental methodology: benchmark suites

**Suites:** TPC, MediaBench, Memory Scheduling Championship, Biobench, SPEC CPU 2006

**Benchmarks were divided in:**

- Memory intensive: greater than 10 Misses Per Kilo Instruction (MKPI)
- Memory non-intensive: smaller than 10MPKI

The ratio of memory intensive benchmark was set to 50%, 100%
Results: Weighted speedup in the case of a multi thread application

The benefits of FIGCache-Fast and FIGCache-Slow increase as workload memory intensity increases

FIGCache-Fast approaches the ideal performance improvement of both FIGCache-Ideal and LL-DRAM
Benefits are more evident when executing memory-intensive applications.

The energy savings originate from:
- Higher DRAM buffer hit rate
- Reduced execution time, which reduced the energy consumption proportionally

Energy consumption of FIGCache-slow and FIGCache-fast consume less energy than the base implementation.
Results: in-DRAM cache hit rate

FIGCache-slow and FIGCache-fast shows significantly improves the Row buffer hit rate and the in-DRAM cache hit rate

Higher hit rate are achieved for higher memory intensity
Outline

• Background, Problems, & goal
• FIGARO substrate
• Fine-grained In-DRAM cache: FIGCache
• Results & Comparison with the SOA
• **Summary and conclusion**
• Discussion
Summary

- **Problem:** DRAM latency represents a performance bottleneck for a vast majority of applications.
- **Goal:** Reduce DRAM latency via in-DRAM cache.
- **Existing in-DRAM caches:**
  - **Key Idea:** Enhance DRAMs with fast regions (small) to implement caches.
  - Coarse-grained (i.e., multi-kB) in-DRAM data relocation.
  - Relocation latency depends on physical distance between slow and fast regions.
- **FIGARO substrate:**
  - **Key Idea:** Exploit the existing shared global row buffer to support data relocation among subarrays of a DRAM bank.
  - Fine-grained (i.e., multi-byte) in-DRAM data relocation and distance independent relocation latency.
- **FIGCache:**
  - **Key Idea:** Cache only small, frequently-accessed portions of different DRAM rows in a designated region of the DRAM.
  - Cache only the portion of a DRAM row that is expected to be accessed in the near future, and increase the row hit ratio.
- **Evaluation:**
  - Improve average system performance by 16.3%.
  - Reduce DRAM energy by 7.8% on average.
- **Results:**
  - FIGARO enables fine-grained data relocation in DRAMs at low cost.
  - FIGCache outperforms state-of-the-art coarse-grained in-DRAM caches.
Outline

• Background, Problems, & goal
• FIGARO substrate
• Fine-grained In-DRAM cache: FIGCache
• Results & Comparison with the SOA
• Summary and conclusion
• Discussion
Strengths

• The paper exploits a very simple, yet very effective idea: reducing the granularity of relocated segments
• The latency insensitive in-DRAM caching mechanism presented in the paper eliminates the data relocation overhead by construction
• Simple and easy to read
• The result are presented clearly, evaluation is done in a very extensive way
Weaknesses

• For this approach to work, the DRAM architecture needs to be modified

• Timing degradation introduced by the additional column and row address muxes could be discussed more extensively
Thoughts

• The approach presented in the paper has a very broad applicability, as it relies on circuitry that is commonly available in DRAMs

• It can be fine-tuned on the specific application, as several aspects can be changed/optimized at run-time
Discussion

Can we use FIGARO when the DRAM is equipped with an ECC chip?
What are the security implications of using FIGARO?
Discussion

Do you envision other use cases for the FIGARO substrate?
Thank you