A Logic-In-Memory Computer

Harold S. Stone

Published in IEEE Transactions on Computers (1970)

Presented by Quentin Adatte

ETH Zürich

Executive Summary

- **Motivation:** Evolution in number of transistors per chip makes room for improvement for reasonable cost and complexity.
- **Problem:** Pins can only be used in limited quantity to keep cost and complexity reasonable.
- **Goal:** Take advantages of the growing number of transistors in an efficient way to achieve better performance by designing a new computer satisfying restrictions and limitations of advanced microelectronic technology
- **Key Idea:** Put logic were data are to avoid using more pins for communication
- **Challenge:** Main memory would be too slow
- **Approach: Logic-in-Cache:**
  - Move part of the computation within caches
  - Allow to load any bit pattern in cache line to perform parallel operations on these patterns
- **Key takeaways:**
  - Reduce distance between computational units and cache will result in better performance
  - Embedded logic in memory can enable a high level of parallelism by increasing the number of computational unit at reasonable cost and complexity
  - Enable efficient access to arbitrary bit pattern in main memory will improve performance by giving more flexibility about which data set can be treated in parallel

Outline

Motivation

Problem/Goal

Key Insights/Ideas

Novelty

Mechanism

- Logic-in-Memory Arrays
- Explicit Cache Control
- Data Pattern and Access Pattern Improvements

Takeaways/Conclusion

Motivation

Microelectronic industry developed medium-scale integration (100-2000 transistors)

Large-scale integration will be available soon with around ten times more transistors

Taking advantage of these available transistors creates a new challenge for hardware designers

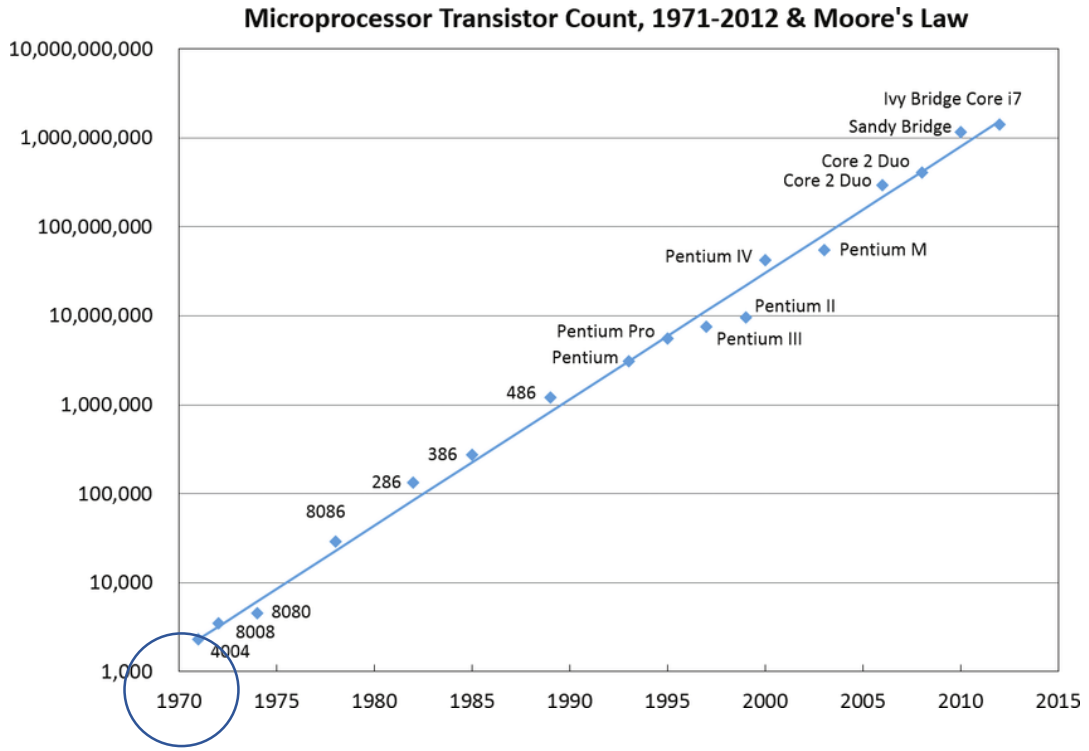
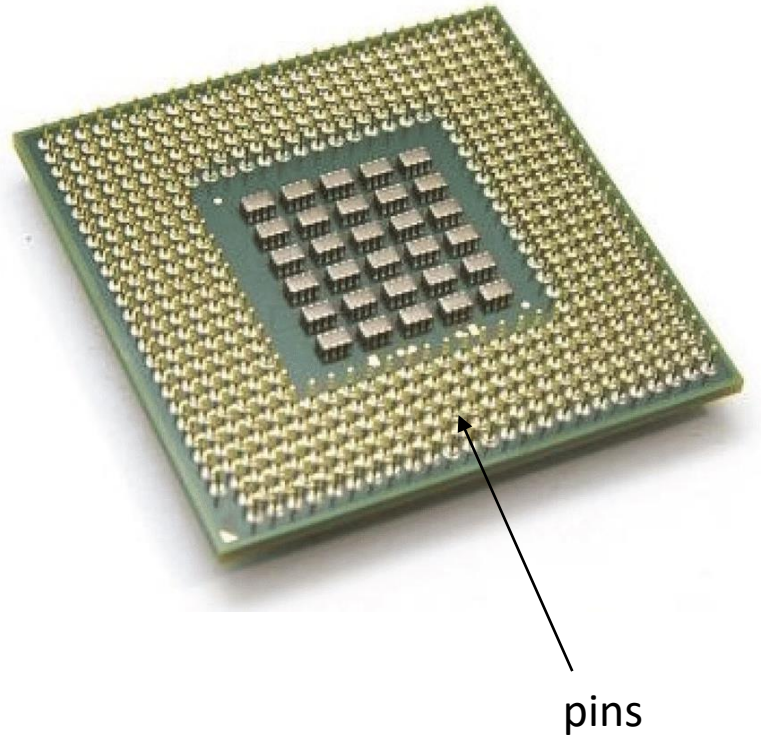


Image source: https://www.researchgate.net/figure/IntelR-microprocessor-transistor-count-evolution-between-1971-and-2012-For-the-last-40_fig1_323459912

Outline

- Motivation
- **Problem/Goal**
- Key Insights/Ideas
- Novelty
- Mechanism
  - Logic-in-Memory Arrays
  - Explicit Cache Control
  - Data Pattern and Access Pattern Improvements
- Takeaways/Conclusion

Problem



Pins are connecting CPU and memory

Limiting the number of pins is limiting the bandwidth between CPU and memory

But bounding the number of pins from above is important because this contributes a lot to modules complexity

Image source: https://www.researchgate.net/figure/Typical-computer-chip-design-with-connecting-pins-Color-figure-available-online_fig1_237043679

Goal

Taking advantages of the increased number of transistors efficiently to achieve better performance by designing a new computer that satisfy restrictions and limitations of advanced microelectronic technology

Outline

- Motivation
- Problem/Goal
- **Key Insights/Ideas**
- Novelty
- Mechanism
  - Logic-in-Memory Arrays
  - Explicit Cache Control
  - Data Pattern and Access Pattern Improvements
- Takeaways/Conclusion

Key Idea

Put logic near to where data resides to avoid using more pins for communication between memory and CPU

Challenge

There is a performance gap between CPU scaling and DRAM scaling

Embedding logic directly in main memory is not viable

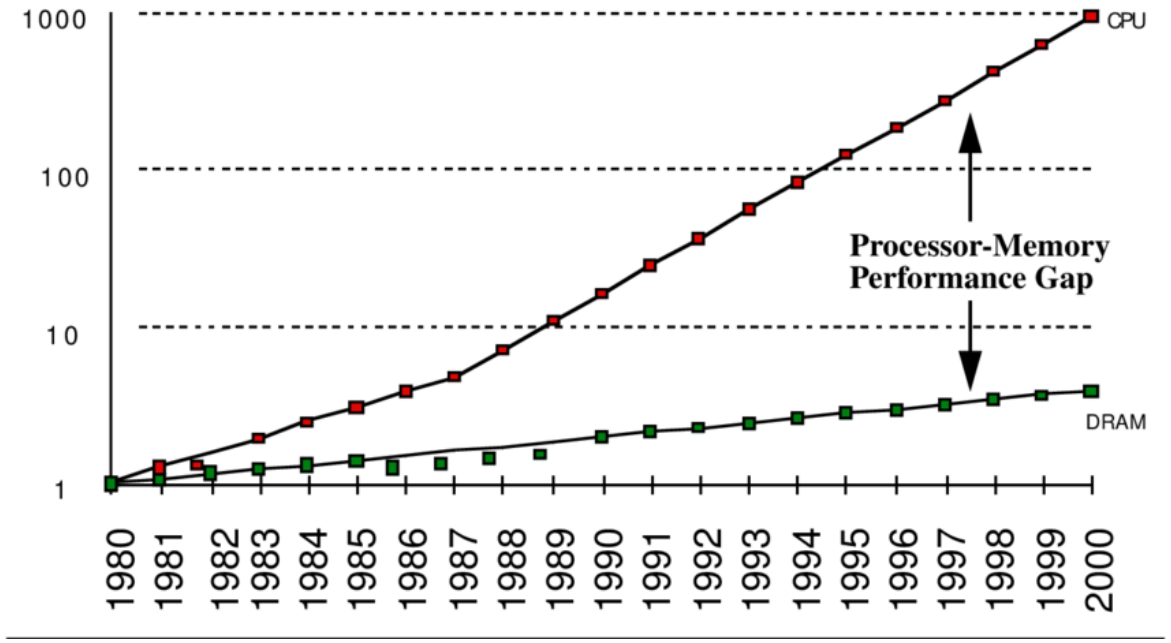


Image source: https://www.researchgate.net/figure/Processor-Memory-Performance-GapHen96_fig1_3214931

Key Insights

Caches are faster than main memory due to two reasons

- Caches use a faster technology (SRAM)
- Caches are smaller

Cellular structure of both memory and caches enables a lot of parallelism

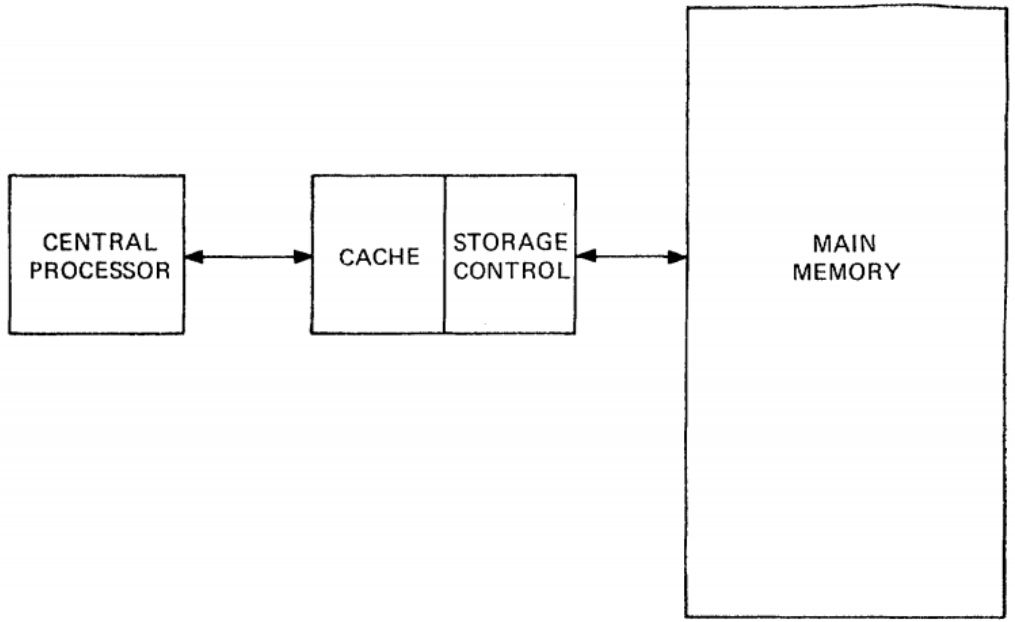


Fig. 1. The structure of a cache-organized computer.

Key Approach

Enhancing cache's memory array with logic

Optimising memory system to get as much benefit as possible from logic-in memory arrays

- More precisely, we want to allow as many bit patterns as possible to be loaded into cache line efficiently

Outline

- Motivation
- Problem/Goal
- Key Insights/Ideas
- **Novelty**
- Mechanism
  - Logic-in-Memory Arrays
  - Explicit Cache Control
  - Data Pattern and Access Pattern Improvements
- Takeaways/Conclusion

Novelty

Earlier research works about logic-in-memory arrays focused more on circuit level of logic-in-memory arrays

- For example, enabling operations like AND or OR to be performed in memory

In this paper, logic-in-memory arrays are treated at a higher level of abstraction, namely, at a computer system level. Author wants to discuss:

- Appropriate instructions to exploit logic-in-memory arrays
- Modifications in access pattern of main memory

Outline

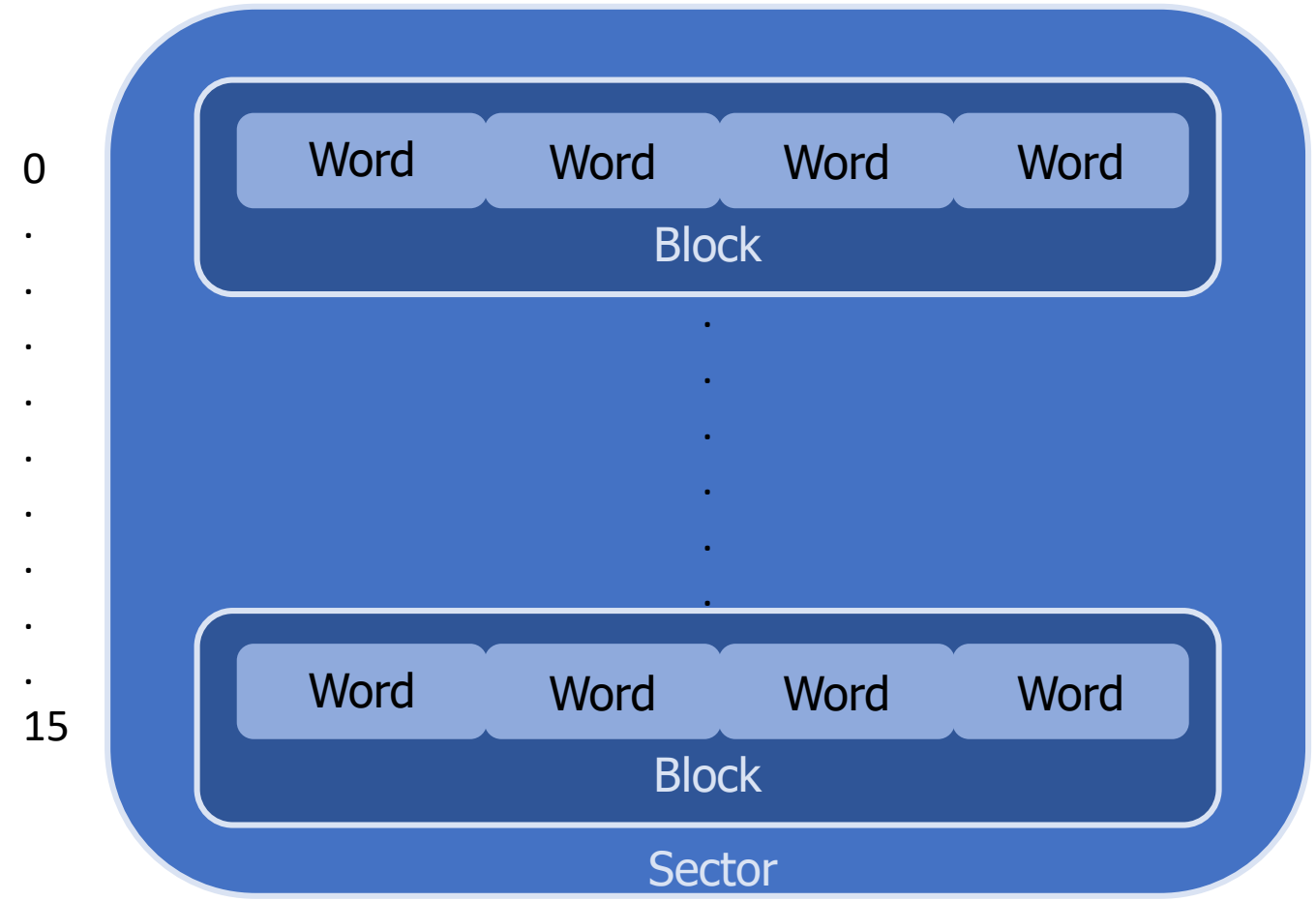
- Motivation
- Problem/Goal
- Key Insights/Ideas
- Novelty
- **Mechanism**
  - **Logic-in-Memory Arrays**
  - Explicit Cache Control
  - Data Pattern and Access Pattern Improvements
- Takeaways/Conclusion

Memory Sector in Baseline System

The cache is a set of such sector

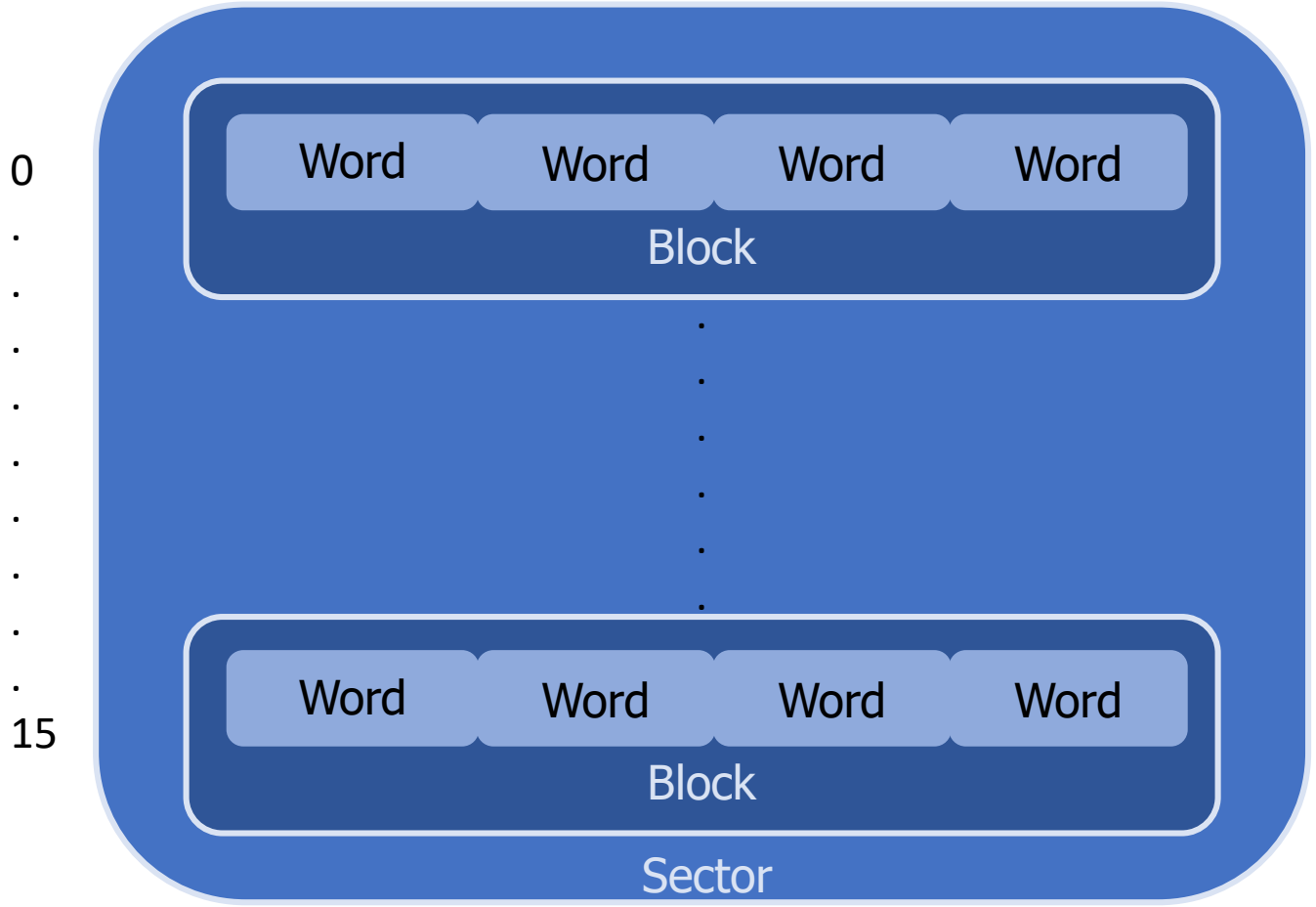
Each sector is an independent logic-in-memory array

We assume that data pattern in these sectors is the same as in main memory

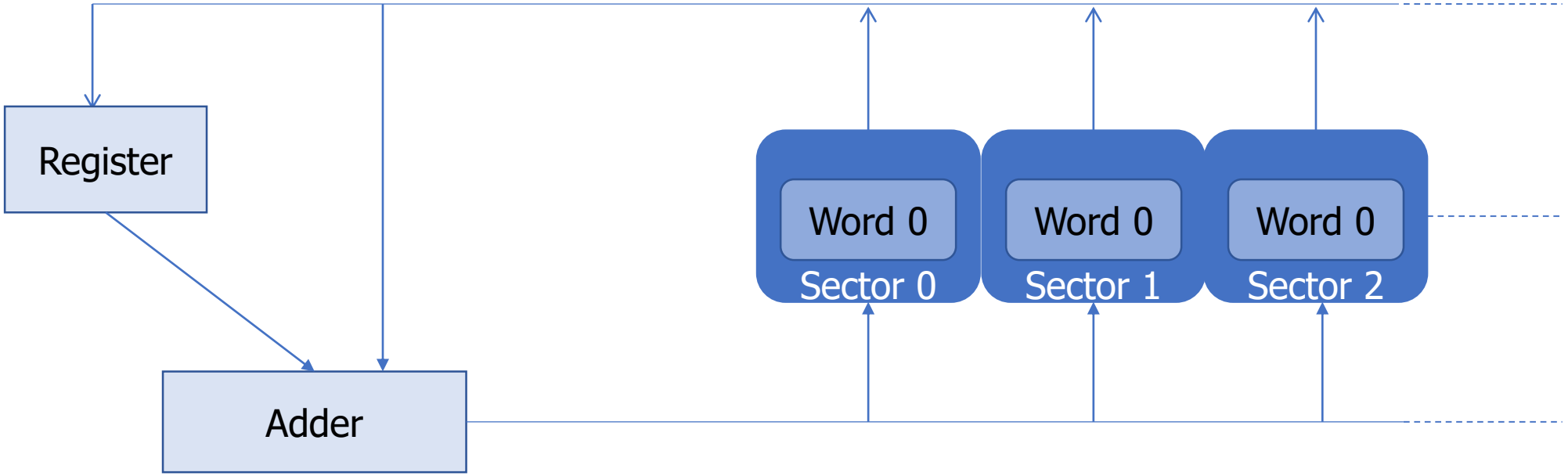


Memory Sector

Only entire such sectors can be loaded from main memory



Hardware Description for Sector Add



Sector addition consists in adding two sectors word-wise

There is one adder per word index

We can process the n words of a sector in parallel

Cost and complexity remains manageable

Instructions for Logic-in-Memory Arrays

Sector ADD: corresponding words of two sectors are added together and results is stored in first sector.

Search on Masked Equality: set the tag bit of each word in a given sector matching a certain pattern

Copy Tag Bit: tag bits of all word in a sector are stored at a specific location

Instructions for Logic-in-Memory Arrays

Tag Bit AND: ANDed two bits of each word in sector and store results at first bit location. OR, XOR, NOT also possible

Sector Scale: each word in one sector is multiplied by a given factor. Adding a factor also possible, called sector bias.

Outline

- Motivation
- Problem/Goal
- Key Insights/Ideas
- Novelty
- **Mechanism**
  - Logic-in-Memory Arrays
  - **Explicit Cache Control**
  - Data Pattern and Access Pattern Improvements
- Takeaways/Conclusion

Explicit Cache Control

Observation: programs could inform cache system about which data set to hold and which to evict, as soon as possible

The idea is to exploit access pattern of programs to learn when a data set currently in cache system can be evicted and when it must be held

For example, it will be better to hold it in cache system if it will be accessed soon during the execution of a program

Explicit Cache Control

```
for(int j = 9 ; j > 0 ; j--){
    print(b[j]);
}
for(int i = 0 ; i < 10 ; i++){
    k += a[i];
}
print(k);
for(int j = 9 ; j > 0 ; j--){
    print(b[j]);
}
```

Do not evict b, it is accessed in the last loop

a can be evicted, it isn't accessed later

b won't be accessed anymore, so it can be evicted

Outline

- Motivation
- Problem/Goal
- Key Insights/Ideas
- Novelty
- **Mechanism**
  - Logic-in-Memory Arrays
  - Explicit Cache Control
  - **Data Pattern and Access Pattern Improvements**
- Takeaways/Conclusion

Data Pattern and Access Pattern Improvements

We assumed until now that data pattern in cache needed to be the same as pattern in main memory

- This means that a word in a cache's sector is a part of a single row in main memory

This highly limitates which data set can be treated in parallel

- Considering the sector add instruction, only two corresponding word from two different sectors can be added together
- So only part of main memory rows can support parallelism

Data Pattern and Access Pattern Improvements

First option is modifying data pattern in main memory

- Storing matrices row- and column-major enable storage of column as words
- Columns can now be treated in a highly parallel manner, exactly like rows

Second option consists in using a non-conventional type of memory which is able to efficiently access any bit pattern in main memory

- This will allow to store any pattern in sector's words, enabling more flexibility about which data set can be treated in parallel

Outline

- Motivation
- Problem/Goal
- Key Insights/Ideas
- Novelty
- Mechanism
  - Logic-in-Memory Arrays
  - Explicit Cache Control
  - Data Pattern and Access Pattern Improvements
- **Takeaways/Conclusion**

Takeaways

Logic-in-memory arrays will lead to improvement in performance at reasonable cost and complexity

- A high level of parallelism can be achieved

We can increase the performance by modifying only memory access pattern

Conclusion

- **Motivation:** Evolution in number of transistors per chip makes room for improvement for reasonable cost and complexity.
- **Problem:** Pins can only be used in limited quantity to keep cost and complexity reasonable.
- **Goal:** Take advantages of the growing number of transistors in an efficient way to achieve better performance by designing a new computer satisfying restrictions and limitations of advance microelectronic technology
- **Key Idea:** Put logic were data are to avoid using more pins for communication
- **Challenge:** Main memory would be too slow
- **Approach: Logic-in-Cache:**
  - Move part of the computation within caches
  - Allow to load any bit pattern in cache line to perform parallel operations on these patterns
- **Key takeaways:**
  - Reduce distance between computational units and cache will result in better performance
  - Embedded logic in memory can enable a high level of parallelism by increasing the number of computational unit at reasonable cost and complexity
  - Enable efficient access to arbitrary bit pattern in main memory will improve performance by giving more flexibility about which data set can be treated in parallel

Strengths

- Suggests a very promising concept: people implemented it
- Easy to understand even after 50 years
- Intuitive idea
- Different improvements opportunities to get more performance from logic-in-memory arrays are addressed
- Research is still going on

Weaknesses

- Does not address potential cache coherency issues caused by the embedded logic in memory
- Does not address energy efficiency aspect of logic-in-memory arrays
  - Less data needs to be transmitted to CPU for computation, and this could save energy
- Absence of analytical model for measuring performance in future research works

Research is still going on

2019: "Processing-in-Memory: A Workload-Driven Perspective“

2019: "Processing Data Where It Makes Sense: Enabling In-Memory Computation“

2020: "NATSA: A Near-Data Processing Accelerator for Time Series Analysis"

2020: "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling"

Discussion

Discussion: Research Continues (1)

Research is still active about logic-in-memory arrays or PIM (Processing In Memory)

Nowadays, different mechanisms have been investigated:

- RowClone: move data in main memory with low latency, low bandwidth utilization and with only small changes around DRAM
- Ambit: perform bitwise operations on large vector directly in DRAM using the majority function
- Gather-Scatter DRAM: improve performance for random memory access pattern by remapping parts of each cache line onto multiple chips. We can then access these parts efficiently and concurrently

Discussion: Research Continues (2)

To learn more:

RowClone: https://people.inf.ethz.ch/omutlu/pub/rowclone_micro13.pdf

Ambit: https://people.inf.ethz.ch/omutlu/pub/ambit-bulk-bitwise-dram_micro17.pdf

Gather-Scatter DRAM: https://people.inf.ethz.ch/omutlu/pub/GSDRAM-gather-scatter-dram_micro15.pdf

These ideas give remarkable improvement in performance

What needs to be done for a wide adoption of PIM ?

Discussion: Allow Wide Adoption of PIM

Many key questions need(ed) to be investigated:

- Which function (e.g sorting or matrix multiplication) could be appropriate to be computed with PIM, keeping in mind that we want to avoid intensive communication between PIM and CPU ?
  - Hint: is it better to have intense computation with easy to communicate results (e.g dot product) ? Or simple computation with more complex to communicate results (e.g. vector add) ?
- How could we reduce the number of exchanged coherence messages between PIM and CPU ?
  - Hint: Speculative access to memory and batching
  - LazyPIM: https://people.inf.ethz.ch/omutlu/pub/LazyPIM-coherence-for-processing-in-memory_ieee-cal16.pdf

Discussion: Allow Wide Adoption of PIM

Can you think about other questions ?

Discussion: Computation Where Data Resides

In the paper, it is suggested to place logic in the cache hierarchy, because main memory would be too slow

Do you think it could be a good idea to place logic in main memory ? What could be any advantages in comparison to put logic in cache ?

- Hint: trade-off between reachable level of parallelism and/or energy efficiency and access speed to data

Can you think of any other location than caches or main memory ?

Discussion: Explicit Cache Control

Programmers could improve cache replacement policy by explicitly give information to caches

What are advantages/disadvantages of relying on programmers in this case ? In general ?

What else could programmers tell ?

Discussion: 3D Stacking

Problem with pins is that they are taking much space

At the same time, they are required to ensure communication between memory and CPU

How could we ensure high bandwidth between memory and CPU while not too much increasing complexity ?

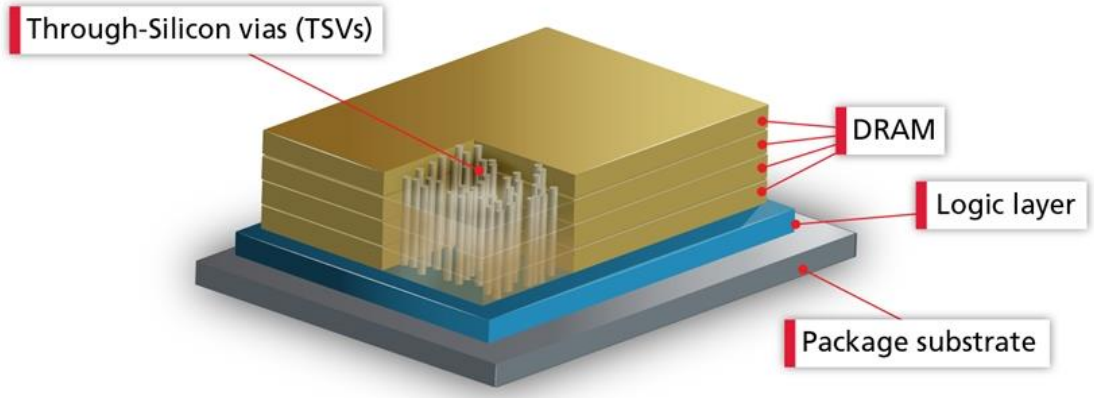
- Use an additional dimension → 3D stacking

Discussion: 3D Stacking

DRAM layers are stacked vertically

TSVs ensure very high-bandwidth between different layers

A logic layer exploits this high-bandwidth by being places below the different layers



HMC Memory Chip Architecture

Image source: https://community.cadence.com/cadence_blogs_8/b/ip/posts/what-s-new-with-hybrid-memory-cube-hmc

Thank You for Listening and Participating