Practical Mechanisms for Reducing Processor-Memory Data Movement in Modern Workloads

Amirali Boroumand

Committee:
- Prof. Onur Mutlu (Co-Chair)
- Dr. Saugata Ghose (Co-chair)
- Prof. James Hoe
- Dr. Parthasarathy Ranganathan
Data Movement Cost

Challenge: large cost of data movement

Data Movement

Data movement incurs high penalty:

1. Large \textit{energy} and \textit{performance} cost to bring data
2. Limited off-chip bandwidth
Processing-In-Memory (PIM)

Potential Solution: move computation close to data

Recent advancement in 3D-stacked technology enabled Processing-In-Memory (PIM)

- Reduces data movement
- Large in-memory bandwidth
- Shorter access latency to memory
PIM Adoption Challenges

PIM introduces new **system challenges**

1. **What to offload?**
   - Need to **identify** primitives
   - Need to consider **design constraints** (e.g., area, power)

2. **How to program PIM architectures?**
   - Need efficient interfaces and mechanisms to let programs take advantage of PIM
     - Maintain **coherence** between PIM and the system
     - Need to provide **address translation**

3. **How to make the software/algorithm aware of PIM?**
   - To fully benefit from PIM, we need to redesign both **software** and **hardware** to be **aware of PIM**
Our Goal

Our goal is to make PIM effective and practical in conventional computing system.
Overview of Our Approach

1. Identifying **key primitives** by analyzing the benefit of **PIM** for key workloads (*work 1*)

2. Mitigating **key system challenges** for communication with **PIM logic** (*work 2*)

3. Redesigning applications **aware of PIM** using **software-hardware co-design** (*works 3 & 4*)
Processor-memory data movement can be significantly reduced

Using practical mechanisms which are aware of modern workloads and architectural constraints
Thesis Contributions

1. Mitigating data movement bottlenecks in Google Consumer workloads

2. Efficient Cache Coherence Support for Near-Data Accelerators (CoNDA)

3. Efficiently Accelerating Edge ML Inference by Exploiting Layer Heterogeneity (Mensa)

4. Enabling Effective HTAP Databases with Specialized HW/SW Co-Design (Polynesia)
## Thesis Contributions

1. **Mitigating data movement bottlenecks in Google Consumer workloads**
   - Efficient Cache Coherence Support for Near-Data Accelerators (CoNDA)

2. **Efficiently Accelerating Edge ML Inference by Exploiting Layer Heterogeneity (Mensa)**

3. **Enabling Effective HTAP Databases with Specialized HW/SW Co-Design (Polynesia)**
Google Workloads for Consumer Devices: Mitigating data movement bottleneck

(ASPLOS’18)
Popular Google Consumer Workloads

Chrome
Google’s web browser

TensorFlow Mobile
Google’s machine learning framework

Video Playback
Google’s video codec

Video Capture
Google’s video codec
Energy Cost of Data Movement

1st key observation: 62.7% of the total system energy is spent on data movement.

Potential solution: move computation close to data

Challenge: limited area and energy budget
Using PIM to Reduce Data Movement

2nd key observation: a significant fraction of data movement often comes from simple functions

We can design lightweight logic to implement these simple functions in memory

Small embedded low-power core

Small fixed-function accelerators

Offloading to PIM logic, on average, reduces energy by 55.4% and improves performance by 54.2%
Thesis Contributions

1. Mitigating data movement bottlenecks in Google Consumer workloads

2. Efficient **Cache Coherence** Support for Near-Data Accelerators (**CoNDA**)

3. Efficiently Accelerating Edge ML Inference by Exploiting Layer Heterogeneity (**Mensa**)

4. Enabling Effective HTAP Databases with Specialized HW/SW Co-Design (**Polynesia**)

SAFARI
CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

(ISCA’19)
Coherence For Near-Data-Accelerators

Challenge: Coherence between NDAs and CPUs

(1) Large cost of off-chip communication

(2) NDA applications generate a large amount of off-chip data movement

It is impractical to use traditional coherence protocols
Existing Coherence Mechanisms

We extensively study existing NDA coherence mechanisms and make three key observations:

1. These mechanisms eliminate a significant portion of NDA’s benefits.

2. The majority of off-chip coherence traffic generated by these mechanisms is unnecessary.

3. Much of the off-chip traffic can be eliminated if the coherence mechanism has insight into the memory accesses.
An Optimistic Approach

We find that an optimistic approach to coherence can address the challenges related to NDA coherence.

1. Gain insights before any coherence checks happen.
2. Perform only the necessary coherence requests.

We propose CoNDA, a coherence mechanism that lets an NDA optimistically execute an NDA kernel.

Optimistic execution enables CoNDA to identify and avoid unnecessary coherence requests.

CoNDA comes within 10.4% and 4.4% of performance and energy of an ideal NDA coherence mechanism.
Thesis Contributions

1. Mitigating data movement bottlenecks in Google Consumer workloads

2. Efficient Cache Coherence Support for Near-Data Accelerators (CoNDA)

3. Efficiently Accelerating Edge ML Inference by Exploiting Layer Heterogeneity (Mensa)

4. Enabling Effective HTAP Databases with Specialized HW/SW Co-Design (Polynesia)
Efficiently Accelerating Edge ML Inference by Exploiting Layer Heterogeneity

(Submitted to ASPLOS’21)
Why ML on Edge Devices?

Significant interest in pushing ML inference computation directly to the edge devices

Privacy  Connectivity  Latency  Bandwidth
Why Specialized ML Accelerator?

Edge devices have limited battery and computation budget

Limited Power Budget  Limited Computational Resources

Specialized accelerator can significantly improve inference latency and energy consumption

Neural Engine (A12)  Edge-TPU
Myriad of Neural Network Models

Challenge: edge ML accelerators have to execute inference efficiently across a wide variety of NN models
Edge Accelerator Challenges

We study inference execution on a commercialized Edge TPU across 24 state-of-the-art Google edge models

We find that the accelerator suffers from three major challenges:

1. Operating significantly below peak throughput
2. Operating significantly lower than peak energy efficiency
3. Inefficient and ineffective memory system
Per-layer Analysis of NN Models

We perform a comprehensive per-layer analysis of each model and make two key observations:

1. There is significant variation in terms of layer characteristics across the models.

2. Even within each model, there is high variation in terms of layer characteristics (e.g., types, shapes).

The key components of edge ML accelerators are completely oblivious to this heterogeneity.
Mensa distributes the layers from an NN model across a collection of smaller accelerators that are specialized for different layer types.

We design an implementation of Mensa for our Google edge models.

We observe that the layers from our Google models naturally group into a small number of clusters.

Based on key characteristics of clusters, we design three specialized accelerators to efficiently execute inference across our Google models.

Mensa improves inference energy and throughput by 3.0x and 3.1x while reducing hardware cost by 2.8x.
Edge TPU Analysis
Edge TPU: Baseline Accelerator

ML Model

DRAM

Input Activation
Parameter
Output Activation

Dataflow

PE Array

Buffer

4MB on-chip buffer

64x64 array
2TFLOP/s
Google Edge Models

2 LSTMs

13 CNNs

3 RCNNs

6 Transducers (RNN-T)
Major Edge TPU Challenges

We find that the accelerator suffers from three major challenges:

1. Operating significantly below peak throughput

2. Operating significantly lower than peak energy efficiency

3. Inefficient and ineffective memory system
(1) High Resource Underutilization

We find that the accelerator operates significantly lower (75.6%) than the peak throughput across all models.
(2) Low Energy Efficiency

The accelerator operates far below (62.4% on average) the upper bound energy efficiency (TFLOP/j).
(3) Memory System Issues

We find that the memory system suffers from **three major challenges**:

1. A significant portion of dynamic and static energy goes to on-chip memory

2. Despite using **large on-chip buffers**, they are not effective in reducing off-chip accesses

3. Parameter traffic takes a large portion of the inference energy and performance
## Major Accelerator Challenges

We find that the accelerator suffers from **three major challenges:**

1. Operating **significantly below** peak throughput

2. Operating **significantly lower** than peak energy efficiency

3. Inefficient and ineffective memory system

**Question:** Where do these challenges come from?
Model Analysis:
Let’s Take a Deeper Look
Into the Models
Diversity Across the Models

**Insight 1**: there is **significant variation** in terms of **layer characteristics** across the models.
Diversity Within the Models

Insight 2: even within each model, layers exhibit significant variation in terms of layer characteristics.

For example, our analysis of edge CNN models shows:

1. Layers in edge CNNs exhibit significant heterogeneity in terms of type, shape, and size.

2. Layers exhibit significant variation in terms of data reuse patterns for parameters and activations.
Diversity Within the Models

We find that **MAC intensity** varies by **200x** across layers.
Diversity Within the Models

We find that FLOP/Byte varies by $244x$ across layers.
Now we can understand the root cause of accelerator challenges
The key components of Edge TPU are completely oblivious to this layer heterogeneity.

Edge accelerators typically take a monolithic approach: equip the accelerator with an over-provisioned PE array and on-chip buffer, a rigid dataflow, and fixed off-chip bandwidth.

While this approach might work for a specific group of layers, it fails to efficiently execute inference across a wide variety of edge models.
### Key Takeaways From Analysis

| 1 | Significant variation of layer characteristics across and within Google edge models |
| 2 | The monolithic design of edge accelerators is the root cause of their shortcomings |
| 3 | All key components (PE array, dataflow, memory system, off-chip bandwidth/proximity to data) must be customized based on layers characteristics |
Proposal: Mensa Framework
Goal: design an edge accelerator that can efficiently run inference across a wide range of different models and layers.

Instead of running the entire NN model on a monolithic accelerator:

Mensa distributes the layers from an NN model across a collection of smaller hardware accelerators that are specialized for different layer types.

- **On-chip Accelerator**
  - Buffer
  - 32GB/s

- **Near-Data Accelerator**
  - Buffer
  - 256GB/s
The **goal** of Mensa’s software **runtime scheduler** is to **identify which accelerator** each **layer** in an NN model should run on.

**Accelerator characteristics**
- Each of the accelerators caters to a specific cluster of layers

**Layer characteristics**
- Layers tend to group together into a small number of clusters

**Which hardware accelerator** is best suited for each **cluster**

**Generated once during initial setup** of a system

**Scheduler**

**Layers Mapping**

**NN model**
Now let’s talk about implementing an instance of Mensa for our Google edge models
We make a key observation that the majority of layers naturally group into a small number of clusters.
Hardware Design Principles

We study the distinguishing characteristics of each cluster and we make two key observations:

Insight 1: we find that clusters can be categorized into (1) compute-centric clusters, and (2) data-centric clusters

↓

We need to support at least two different accelerator designs: (1) one compute-centric, (2) one data-centric

Insight 2: reuse patterns of layers are a key distinguishing factor between different hardware designs

↓

We need separate accelerators to account for different types of dataflow across the clusters
Hardware Design Principles

Based on key characteristics of clusters, we design three specialized accelerators to efficiently execute inference across our Google models.

- **Pascal**: On-chip Accelerator
  - Clusters 1 & 2
  - Off-chip Bandwidth: 32GB/s

- **Pavlov**: Near-Data Accelerator
  - Clusters 3, 4 & 5
  - Off-chip Bandwidth: 256GB/s

- **Jacquard**: On-chip Accelerator
  - Clusters 1 & 2
  - Off-chip Bandwidth: 256GB/s
A Brief Look at Pascal
Pascal Dataflow

Pascal caters for Cluster 1\&2 layers: majority are **pointwise** or standard **Conv2D** layers with shallow input/output channels

We devise a dataflow that enables **temporal reduction of output activations and spatial multicasting of parameters**:

1. **Spatially** distributing output activation across PEs provide **temporal reduction** $\rightarrow$ **reduce buffer size** (16x) and on-chip traffic
2. Since parameters have **small footprint**, we **temporally** replicate each parameter across PEs at each time step $\rightarrow$ **spatial multicasting**
Pascal Dataflow - Example

Input Activation

Parameters

Output Activation

Different PEs (Spatial Dimension)

Spatial Reuse (multicast)

Temporal Reuse (reduction)

Cycle 1

Cycle 2
Pascal In A Nutshell

Co-design the proposed dataflow with the accelerator’s memory system

- 256KB Act. Buffer (8x Reduction)
- 128KB Param. Buffer (32x Reduction)

Clusters 1&2 include compute-centric layers (e.g., early 2D conv)

Data reuse opportunities exposed by our dataflow enable us to significantly reduce the on-chip buffer sizes

Off-chip bandwidth available to PE array is same as the baseline accelerator

2 TFLOP/s

32GB/s

3.8X Area Reduction
A Brief Look at Pavlov
Pavlov Dataflow

Pavlov caters to Cluster 3: (1) zero param. reuse & large footprint, (2) high act. reuse & small footprint, and (3) major operation is MVM

We devise a dataflow that enables temporal reduction of output activations and sequential access to parameters:

1. Avoiding spatial reduction of output act.: eliminates partial sum traffic
2. Sequential access pattern → use bandwidth without complex hardware
3. Replicates input act. across all PEs while spatially distributing params

Require high bandwidth to achieve high PE utilization
Pavlov Dataflow - Example

MVM operation for an LSTM gate

$\mathbf{W}_x \mathbf{X}_t = \mathbf{O}_t$

Different PEs (Spatial Dimension)

Sequential access

Cycle 1

Cycle 2

Temporal Reuse (reduction)

$H = 16$

$\mathbf{X}_0(\mathbf{t})$

$\mathbf{X}_1(\mathbf{t})$

$\mathbf{W}_{x_0}$ $\mathbf{W}_{x_1}$ $\mathbf{W}_{x_2}$ $\mathbf{W}_{x_3}$

$\mathbf{O}_0$ $\mathbf{O}_1$ $\mathbf{O}_2$ $\mathbf{O}_3$

$\mathbf{PE}_1$ $\mathbf{PE}_2$ $\mathbf{PE}_3$ $\mathbf{PE}_4$
Pavlov In A Nutshell

Designed aware of Cluster 3 characteristics and proposed dataflow

No Parameter Buffer (4MB in Baseline)

128KB Activation Buffer (16x Reduction)

For parameters, we use only one level of memory hierarchy and stream parameters directly from DRAM

Since Pavlov is data-centric and its sequential access parameters can exploit high bandwidth memory, we place it in the logic layer of 3D-stacked memory

Cluster 3 include LSTM-like data-centric layers: low MAC intensity

Buffers

8x8 PE Array

NoC

256GB/s

128 GFLOP/s

DRAM
Let’s Take a Brief Look at our Evaluation
Simply providing high-bandwidth to the baseline accelerator provides only 7.5% reduction.

Mensa reduces energy by 66.0% and improves energy efficiency by 3.0X compared to Baseline.

More beneficial for LSTM/Transducer models (14.2% energy reduction) than CNNs and RCNNs.

Mensa lowers on-chip/off-chip parameter traffic energy by 15.3x by scheduling layers on the accelerator with the most appropriate dataflow and proximity to data.

Base+HB still incurs the high energy costs of (1) on-chip buffers and (2) off-chip traffic to DRAM.

Mensa reduces the dynamic energy of the on-chip buffer and NoC by 49.8x over Base+HB, by avoiding overprovisioning and catering to specialized dataflows.
Wrap-Up

• We extensively analyze a state-of-the-art edge accelerator using 24 Google edge models
  – Wide range of models (CNNs, LSTMs, Transducers, RCNNs)

• We find that the accelerator suffers from three challenges
  – Operating significantly below peak throughput
  – Operating significantly below theoretical energy efficiency
  – Inefficiently handling memory accesses

• We find that these shortcomings arise from the monolithic design of edge accelerators
  – Their design do not account for layer heterogeneity

• We propose a new framework called Mensa
  – Consist of heterogeneous accelerators whose dataflow and hardware are specialized for specific clusters of layers

• We design an implementation of Mensa for Google models
  – Show that it improves performance and energy by $3.0X$ and $3.1X$
  – Reduce cost and improves area efficiency by $2.8X$ and
Contributions

1. Mitigating data movement bottlenecks in Google Consumer workloads

2. Efficient Cache Coherence Support for Near-Data Accelerators (CoNDA)

3. Efficiently Accelerating Edge ML Inference by Exploiting Layer Heterogeneity (Mensa)

4. Enabling Effective HTAP Databases with Specialized HW/SW Co-Design (Polynesia)
Enabling Effective HTAP Databases with Specialized HW/SW Co-Design (Polynesia)

(Submitted to ASPLOS’21)
Real-time Analysis

An explosive interest in many applications domains to perform analytics on the most recent version of data (real-time analysis)

Use transactions to record each periodic sample of data from all sensors.

Run analytics across sensor data to make real-time steering decisions.

For these applications, it’s critical to analyze the transactions in real-time as the data’s value significantly diminishes over time.
Traditionally, **new transactions (updates)** are propagated to the **analytics database** using a **periodic and costly** process.

To support real-time analysis: a single hybrid DBMS is needed to execute both transactions and analytics.
Ideal HTAP System Properties

An ideal HTAP system should have three properties:

1. **Workload-specific optimizations**
   - Each workload must benefit from its own specific optimizations

2. **Data Freshness and consistency guarantee**
   - Guarantee access to the most recent version of data for analytics while ensuring that both workloads have a consistent view of data

3. **Performance Isolation**
   - Latency and throughput of both workloads are the same as if they were run in isolation

Meeting all three properties at the same time is very challenging
Limitations of State-of-the-Art

We extensively study state-of-the-art HTAP systems and observe two key problems:

1. **Data freshness and consistency mechanisms** generate a large amount of data movement which causes a drastic **reduction** in transactional/analytical **throughput**

2. They **fail** to provide **performance isolation** because of the **high resource contention** between transactional and analytical workloads
We propose Polynesia, a hardware/software cooperative design for in-memory HTAP databases. Polynesia improves transactional/analytical throughput and energy by 1.7X/3.7X and 48% compared to prior HTAP systems.
Analysis of State-of-the-art HTAP Systems
Analysis of State-of-the-art HTAP Systems

We study two major types of HTAP systems:

- **Single-Instance**
  - Main Replica
  - Transactions
  - Analytics

- **Multiple-Instance**
  - Replicas
  - Transactions
  - Analytics

We observe **two key problems**:

1. **Data freshness and consistency mechanisms**
   - Are costly and cause a drastic reduction in throughput

2. **These systems fail to provide performance isolation**
   - Because of the high resource contention
Single-Instance Design
Since both analytics and transactions work on the same data concurrently, we need to ensure that the data is consistent.

There are two major mechanisms to ensure consistency:

1. Snapshotting (Snapshot Isolation)
2. Multi-Version Concurrency Control (MVCC)
Several HTAP systems (e.g., HyPer) use snapshotting to provide consistency via Snapshot Isolation (SI).

These systems explicitly create snapshots from the most recent version of data and let the analytics run on the snapshot while transactions continue updating data.
We find that this approach requires frequent snapshot creation to sustain data freshness under high transactional update rate. The overhead comes from Memcpy operation which generates a large amount of data movement and introduces significant interference.
MVCC avoids making full copies of data by keeping several versions of the data (used in HyPer v2)

When updates happen, MVCC creates a new time-stamped version of data and keeps the old version in a version chain alongside the data.
Multi-Version Concurrency Control (MVCC)

We observe that MVCC overhead leads to 42.4% performance loss over zero-cost MVCC

We find that long version chains are the root cause of the issue

1. Frequent transactional updates create lengthy version chains
2. Scan-heavy analytics traverses a lengthy version chain upon accessing a data tuple
   - Expensive time-stamp comparison + a very large number of random memory accesses
Wrap up: Single-Instance Systems

While single-instance design enables high data freshness, we find that it suffers from two major challenges:

1. High **Cost** of Consistency and Synchronization

2. **Limited** Performance Isolation
Multiple-Instance Design
One of the **major challenges** in multiple-instance systems is to keep **analytic replicas** up-to-date.

To maintain data freshness:

1. **Update Shipping**: gather updates from transactional threads and ship them to analytic replica.

2. **Update application**: perform the necessary **format conversation** and apply those updates to analytic replicas.
Maintaining Data Freshness

We find that the transactional throughput reduces by up to **21.2%** and **64.2%** during update shipping process and update application process.

The overhead becomes significantly higher when the transactional queries are more update-intensive.
Data Freshness: Shipping Updates

Throughput reduction is because update shipping generates a large amount of **data movement** and takes **several cycles**.

Timeline → CPU → Update Logs → Memory → Analytic Replica

- **Scan and Merge**
- **Find the target columns for updates**
- **Ship the updates**

Updates from different transactional threads

**High update rate** → **High frequency update shipping** → **Higher data movement overhead**
Data Freshness: Update Application

Analytics engines are optimized for long-running read-only queries and they are not update-friendly.

Transactional Replica

Analytical Replica

Update: Row 2, Column 1 and 3

Compressed Column

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ann</td>
</tr>
<tr>
<td>1</td>
<td>car</td>
</tr>
<tr>
<td>2</td>
<td>cat</td>
</tr>
<tr>
<td>3</td>
<td>ear</td>
</tr>
</tbody>
</table>

A simple tuple update in row-wise layout leads to

These operations generate a large amount of data movement and take many CPU cycles

reconstruct the dictionary, and (2) recompress the column
More insights on Data Freshness Challenges

Our analysis shows that simply providing higher bandwidth (8x) to CPU cores does not address the challenges.

We need to take advantage of PIM logic to reduce data movement and resource contention.

We find that simply offloading them to general purpose PIM cores does not address the challenges.

We need to design custom algorithm and hardware to efficiently execute update shipping/application process.
Major Takeaways from Analysis

1. State-of-the-art HTAP systems do not meet all of the desired HTAP properties

2. Data freshness and consistency mechanisms are data-intensive and cause a drastic reduction in throughput

3. These systems fail to provide performance isolation because of the high resource contention

4. We need to take advantage of custom algorithm and PIM logic to address these challenges
Polynesia
The key idea is to partition the computing resources into two types of isolated, specialized processing islands.

Isolating transactional islands from analytical islands allows us to:

1. Apply workload-specific optimizations to each island
2. Avoid high resource contention
3. Design efficient data freshness and consistency mechanisms where we can maintain data freshness and consistency without incurring high data movement costs
Each island includes (1) a replica of data, (2) an optimized execution engine, and (3) a set of hardware resources designed to sustain the bursts of writes.

- **Transactional Island:**
  - Exec. Engine
  - Replica
  - CPU Cores
  - Conventional multicore CPUs with multi-level caches
  - Designed to sustain the bursts of writes

- **Analytical Island:**
  - Exec. Engine
  - Replica
  - PIM Logic
  - Data Freshness Mechanism
  - Consistency Mechanism
  - Designed to provide high read throughput
  - Take advantage of a customized version of PIM to mitigate data movement bottleneck
We co-design new algorithms and efficient hardware support for the three key components of an analytical island.

Design two algorithms:
1. update shipping and
2. update application

Design custom PIM logic for both algorithms.
We co-design new algorithms and efficient hardware support for the three key components of an analytical island.

Develop an algorithm relies on a combination of versioning and snapshotting to maintain data consistency.

Design an in-memory copy unit that enables highly efficient snapshot creation.
Analytical Islands Key Components

We co-design new algorithms and efficient hardware support for the three key components of an analytical island.

A custom data placement and task scheduler aware of 3D-stacked memory.

Simple PIM cores to execute execution engine.
We implement an instance of Polynesia that supports *relational transactional* and *analytical* workloads.
Data Freshness Mechanism
Data Freshness Mechanism:

1. **Update Shipping**: gather updates from transactional islands, find the target location in analytical island, and ship them.

2. **Update Application**: performs format conversion and applies the update to the analytical replica.
Our update shipping algorithm has **three major** stages:

1. **Scan and Merge**
   - Update Log1
   - Update Log2
   - Update LogN
   - Merge / Sort

2. **Find Target Column**
   - Update_{k}
   - Target Column
   - Hash Unit

3. **Transfer Updates**
   - Update_{k}
   - Copy
   - Column Buffer

**Two major bottlenecks** that keep us from meeting **data freshness** and **performance isolation**

These primitives generate a large amount of data movement and account for 87.2% of our algorithm’s execution time.
To avoid these **bottlenecks**, we design a new hardware accelerator, called **update shipping unit**.

- A **3-level comparator tree** to merge updates.
- **Decoupled hash computation** from the bucket traversal to allow for **concurrent** lookups.
- **Multiple fetch and write-back units** to issue multiple memory accesses concurrently.
Now let’s talk about Update Application
Like other relational analytical DBMSs, our analytical engine uses the **column-wise data layout** and **dictionary encoding**.

**Analytical Replica**

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Compressed Column**

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ann</td>
</tr>
<tr>
<td>1</td>
<td>car</td>
</tr>
<tr>
<td>2</td>
<td>cat</td>
</tr>
<tr>
<td>3</td>
<td>ear</td>
</tr>
<tr>
<td>4</td>
<td>man</td>
</tr>
</tbody>
</table>
We design our update application algorithm to be aware of **PIM logic** characteristics and constraints.

Since the number of updates are fixed (unlike a column), this allows us to keep power/area overhead of hardware sorter within logic layer’s budget.

We maintain a hash index that links the old encoded value in a column to the new encoded value. This avoids the need to decompress the column and add updates, eliminating data movement and random accesses to 3D DRAM.
We design a hardware implementation of our algorithm, and add it to each in-memory analytical island.

A 1024-value bitonic sorter, whose basic building block is a network of comparators.

Similar design from our update shipping unit.
Other Key Components of Analytical Islands

- **Analytics**
- **Execution Engine**
- **Replica**
- **Consistency Mechanism**
- **Data Freshness Mechanism**
- **Updates**
- **PIM Logic**

Analytical Islands
Brief Look at Evaluation
While SI-MVCC is the best baseline for transactional throughput, it degrades analytical throughput by 63.2%, due to its lack of workload-specific optimizations and poor consistency mechanism.

Both MI+SW and MI+SW+HB fall significantly short of Ideal-Txn because of lack of performance isolation and overhead of update propagation.

MI+SW+HB is the best software-only HTAP for analytics, because it provides workload-specific optimizations, but it still loses 35.3% of the analytical throughput due to high resource contention.

Overall, Polynesia achieves all three properties of HTAP system and has a higher transactional/analytical throughput (1.7X/3.74X) over prior HTAP systems.

Polynesia comes within 8.4% of ideal Txn because it uses custom PIM logic for data freshness/consistency mechanisms which significantly reduce resource contention and data movement.

Polynesia improves over MI+SW+HB by 63.8%, by eliminating data movement, and using custom logic for update propagation and consistency.
Wrap Up

• Many application domains have a critical need to perform real-time data analysis, and make use of HTAP systems
  – An ideal HTAP system should have three properties: (1) data freshness and consistency, (2) workload-specific optimization, (3) performance isolation

• We extensively study state-of-the-art HTAP systems
  – We find that neither of them can meet all HTAP properties

• We propose Polynesia, a novel hardware/software cooperative design for in-memory HTAP databases
  – Divides the system into transactional and analytical processing islands
  – Implements custom algorithms and hardware to reduce the costs of update propagation and consistency
  – Exploits PIM for the analytical islands to alleviate data movement

• Polynesia outperforms three state-of-the-art HTAP systems, with average transactional/analytical throughput improvements of 1.7X/3.7X, educes energy consumption by 48%
Conclusion
Conclusion

• **Problem:** data movement cost is a critical challenge
• **PIM** is a potential solution to address this problem
  – **Challenge:** there are many practical system-level challenges that need to be solved to enable the widespread adoption of PIM
• **Goal:** make PIM effective and practical in computing systems
• Toward this end, we propose a series of practical mechanisms to reduce processor-memory data movement
  – (1) Examine the suitability of PIM across key Google workloads
  – (2) Address a major system challenge (coherence) for adopting PIM in computing systems
  – (3) Propose a **HW/SW co-design approach** aware of PIM for designing an accelerator for Google edge models
  – (4) Propose a **HW/SW co-design approach** aware of PIM for in-memory hybrid databases
• We conclude that the proposed mechanisms provide promising solutions to make PIM effective and practical
<table>
<thead>
<tr>
<th>Thesis Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> Mitigating <em>data movement</em> bottlenecks in <em>Google Consumer workloads</em></td>
</tr>
<tr>
<td><strong>2</strong> Efficient <em>Cache Coherence</em> Support for <em>Near-Data Accelerators</em> (<em>CoNDA</em>)</td>
</tr>
<tr>
<td><strong>3</strong> Efficiently Accelerating Edge ML Inference by Exploiting <em>Layer Heterogeneity</em> (<em>Mensa</em>)</td>
</tr>
<tr>
<td><strong>4</strong> Enabling Effective <em>HTAP</em> Databases with Specialized <em>HW/SW Co-Design</em> (<em>Polynesia</em>)</td>
</tr>
</tbody>
</table>
Publication

- (1) **Amirali Boroumand**, Saugata Ghose, Minesh Patel, Rachata Ausavarungrunrun, Hasan Hassan, Kevin Hsieh, Brandon Lucia, Nastaran Hajinazar, Krishna Melladi, Hongzhong Zheng, Onur Mutlu, “**CoNDA: Enabling Efficient Near-Data Accelerator-CPU Communication by Optimizing Data Movement**” (ISCA’19)

- (2) **Amirali Boroumand**, Saugata Ghose, Youngsok Kim, Rachata Ausavarungrunrun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, Onur Mutlu, “**Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks**”. (ASPLOS 2018)

- (3) **Amirali Boroumand**, Saugata Ghose, Hasan Hassan, Minesh Patel, Kevin Hsieh, Brandon Lucia, Krishna Melladi, Hongzhong Zheng, Onur Mutlu, ”**LazyPIM: Efficient Coherence Mechanism For Processing In Memory**” (CAL 2017)


Publication


- (7) Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin Chang, Amirali Boroumand, Saugata Ghose, Onur Mutlu, ”Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation” (ICCD 2016)


Publication


Practical Mechanisms for Reducing Processor-Memory Data Movement in Modern Workloads

Amirali Boroumand

Committee: Prof. Onur Mutlu (Co-Chair)
            Dr. Saugata Ghose (Co-chair)
            Prof. James Hoe
            Dr. Parthasarathy Ranganathan
Backup
Future Directions

- **Applying our workload analysis methodology to Other Key Consumer Workloads**
  - In our analysis, we focus on applications that run on CPU cores in consumer devices
  - There are other important consumer applications that relay on GPU cores or Camera systems: maps, 3D games, VR applications

- **Automating the Cluster Identification in Mensa**
  - Mensa’s scheduler relies on two pieces of information to generate a mapping between layers and accelerators: (1) Characteristics of each cluster (2) Which hardware accelerator is best suited for each cluster
  - However, generating cluster identification info could be challenging as it involves a comprehensive analysis across all layers from all models
  - One future research direction is how we can automate the process of identifying clusters across a wide range of models
  - One potential solution to address this challenge is to employ automated Neural Architecture Search (NAS) methods to identify clusters
Future Directions

- **Extending CoNDA to Non-NDA Systems**
  - Coherence for specialized accelerators is still an open problem
  - We believe that the core idea of CoNDA can be extended to address coherence challenge between CPUs and on-chip accelerators

- **Extending Polynesia to Support Non-Relational Analytical Execution Engine**
  - We focus on designing an instance of Polynesia that supports relational (SQL) transactional and analytical workloads
  - However, the term analytics is no longer limited to SQL analytics → ML, graph processing, NoSQL
  - A modern HTAP system needs to support both high update rates as well as the ability to run diverse analytics on the data
  - One potential future research direction is to examine how we can use Polynesia framework to support various types of analytics workloads
Comparison with EyerissV2 (1)

• We compare Mensa with EyerissV2 for 7 representative models:
  – 1 LSTM, 1 Transducers, 4 CNNs, 1 RCNN

• We find that compared to EyerissV2, Mensa reduces total inference energy by 53.2%, and improves throughput by 3.3X

• Similar to the Edge TPU accelerator, EyerissV2 suffers from high energy inefficiency for the LSTM and Transducer models

• Mensa, on the other hand, lowers the energy spent on on-chip and off-chip parameter traffic, by scheduling layers on the accelerators with the most appropriate dataflow for LSTM and Transducer layers
Comparison with EyerissV2 (2)

- For CNN models, EyerissV2 performs better than the baseline accelerator as it has a smaller on-chip buffer, which enables EyerissV2 to reduce dynamic energy.

- However, EyerissV2 still falls short of Mensa’s energy efficiency for CNNs:
  - It’s fixed dataflow cannot efficiently expose reuse opportunities across different layers.
  - Some CNN layers have a large parameter footprint and very low data reuse, which generates a large amount of off-chip parameter traffic.
  - Increases static energy because it uses a much smaller PE array, which significantly increases inference latency for many compute-intensive CNN layers in Clusters 1 and 2.
(3.3) High Cost of Moving Parameters

Parameter traffic (Off-chip and on-chip) takes a large portion of the inference energy and performance

50.3% and 30.9% of total energy goes to parameters off-chip traffic and distributing parameters across PE array
Performance Analysis

(1) Properly provisioned PE array, (2)

Utilization is still very low. The major reasons are (1) lack of customized dataflow to properly expose reuse opportunities across layers, (2) over-provisioned PE array for many layers.

The higher bandwidth in Base+HB pushes average utilization up to 34.0%, and improves throughput by 2.5x.

The largest improvements are for LSTMs and Transducers (4.5x), thanks to their low FLOP/B ratio and large footprints.

Overall, Mensa improves utilization on average by 28.8X and 7.5X over Baseline and Baseline+HB, and significantly improves computational throughput (on average by 3.1X) across all models.

The consistency is low across all models (on average 27.3%).
While SI-MVCC is the best software-only DBMS for transactional throughput, it degrades analytical throughput by 63.2%, due to its lack of workload-specific optimizations and poor consistency mechanism.

MI+SW+HB, even with its higher bandwidth, fall significantly short of Ideal-Txn because of lack of performance isolation.

MI+SW+HB is the best software-only HTAP for analytics, because it provides workload-specific optimizations.

Overall, Polynesia has a higher transactional throughput (1.7X), and a higher analytical throughput (3.74X).
Multiple-Stack result for Polynesia

Normalized Analytical Throughput

- **Multiple-Instance**
- **HoCl**

![Bar chart showing throughput comparison between Multiple-Instance and HoCl for 1 Stack, 2 Stack, and 4 Stack configurations.](image-url)
Energy result for Polynesia

The diagram illustrates the energy consumption for different components in Polynesia, categorized by CPU, Caches, Interconnect, and DRAM. The energy values are presented in PJ (petajoules), with logarithmic scales ranging from 1E+13 to 6E+13.

- **SI-SS**:
  - CPU: 1E+13
  - Caches: 2E+13
  - Interconnect: 3E+13
  - DRAM: 4E+13

- **SI-MVCC**:
  - CPU: 1E+13
  - Caches: 2E+13
  - Interconnect: 3E+13
  - DRAM: 4E+13

- **MI+SW**:
  - CPU: 1E+13
  - Caches: 2E+13
  - Interconnect: 3E+13
  - DRAM: 4E+13

- **HOCI**:
  - CPU: 1E+13
  - Caches: 2E+13
  - Interconnect: 3E+13
  - DRAM: 4E+13
Insight 2: while there are different types of accelerators for compute- and data-centric layers, the reuse patterns of layers are a key distinguishing factor between different hardware designs.

Different dataflows can expose different data reuse opportunities:

- **Spatial Multicast**
  - PE1 → O0 → PE2 → O1 → PE3 → O2

- **Spatial Reduction**
  - PE1 → O0 → PE2 → O0 → PE3 → O0

- **Temporal Multicast & Recusion**
  - Cycle 1: PE1 → O0 → PE2 → O1 → PE3 → O2
  - Cycle 2: PE1 → O0 → PE2 → O1 → PE3 → O2

We need separate accelerators to account for different types of dataflow across the clusters.
Diversity Across the Models

**Insight 1**: There is a significant variation in terms of layer characteristics across the models.
We find that **23.8%** of the **CPU cycles** (and **30.8%** of cache misses) go to the **update application** process.

Even providing higher bandwidth (8x) does not address the challenges as it cannot reduce data movement and resource contention.
To enable real-time analysis, we need to support both high data ingest rate and the ability to perform analytics on data.

A hybrid DBMS (HTAP system) is needed that can execute both transactional and analytical queries over all data.
Each island includes (1) a **replica** of data, (2) an **optimized** execution engine, and (3) a set of **hardware resources**

- Designed to sustain the **bursts of writes**
- Designed to provide **high read throughput**

We take advantage of a customized version of PIM to mitigate data movement bottleneck.
Consistency Mechanism
Consistency mechanism must not compromise either the throughput of analytical queries or the update propagation rate.

Consistency mechanism has to satisfy two requirements:

1. Updates must be applied all the time and should not be blocked by analytic queries → Data freshness property

2. Analytics must be able to run all the time should not be blocked by update propagation process → performance isolation property
Consistency Mechanism: Algorithm

Our mechanism relies on a combination of snapshotting and versioning to provide snapshot isolation for analytics.

- Our consistency mechanism is based on two key observations:
  1. Updates are applied at a column granularity
  2. Snapshotting a column is cost effective using PIM logic
For each column, there is a chain of snapshots where each chain entry corresponds to a version of this column.

Polynesia does not create a snapshot every time a column is updated. Instead, Polynesia marks the column as dirty.

Unlike chains in MVCC, each version is associated with a column, not a tuple.

Polynesia creates a new snapshot only if (1) any of the columns are dirty, and (2) no current snapshot exists for the same column.
Our algorithm success at satisfying performance isolation relies on how fast we can do MemCpy to minimize snapshotting latency.

Multiple Fetch and Write-back units to issue multiple accesses concurrently.

Track outstanding reads, as they may come back from memory out of order. Allows to immediately initiate the write after a read is complete.

We find that the buffer lookup limits the performance, as each lookup results in a full scan, and multiple fetch units perform lookups concurrently. To alleviate this, we design a hash index.
Analytical Engine
Analytical Engine: Query Execution

Query

**Select** A.id, B.id  
**From** A **JOIN** B  
**ON** A.id = B.id  
**Where** A.value > 55

Algebraic Query Plan

Parser

Volcano execution model

High degree of inter- and intra-operator parallelism

Task

Operator 1  
Operator 2

A

B

\( \sigma \)

\( \Pi \)
Efficient analytical query execution strongly depends on:

1. Data layout and data placement
2. Task scheduling policy
3. How each physical operator is executed

We find that the execution of physical operators of analytical queries significantly benefit from PIM.

Without a HTAP and PIM-aware data placement/task scheduler, PIM logic for operators alone cannot provide throughput improvements.
Analytical Engine: Data Placement
Analytical Engine: Data Placement

DSM Data Layout

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Compressed Column
- Dictionary
  - Id | Value
  - 0  | ann
  - 1  | car
  - 2  | cat
  - 3  | ear
  - 4  | man

Data Placement

Vaults

DRAM Layer

Logic Layer

Limited power and area budget
We propose a hybrid strategy where we create small vault groups, and partition a column across the vaults in a vault group.

- Allows us to increase the aggregate bandwidth for servicing each query by 4 times, and provides up to 4 times the power and area for PIM logic.

- The number of vaults per group is critical for efficiency: too many vaults can complicate the update application process, while not enough vaults can degrade throughput.
Analytical Engine:
Task Scheduler
For each query, the scheduler makes **three key** decisions:

1. Decides how many tasks to create
2. Finds how to map these tasks to the available resources (PIM threads)
3. Guarantees that dependent tasks are executed in order
Task Scheduler: Initial Hueristic

Our scheduler heuristic that generates tasks by disassembling the operators of the query plan into operator instances.

Query

```
Select A.id, B.id
From A JOIN B
ON A.id = B.id
Where A.value > 55
Where B.value < 70
```

Query Plan

- \( \Pi \) (projection)
- \( \sigma \) (selection)
- \( \sigma \) (selection)

Global Work Queue

- Task 1: \( \sigma \) (selection) \( A_1 \)
- Task 2: \( \sigma \) (selection) \( A_2 \)
- Task 3: \( \sigma \) (selection) \( A_3 \)
- Task 4: \( \sigma \) (selection) \( A_4 \)
- Task 5: \( \sigma \) (selection) \( B_1 \)
- Task 6: \( \sigma \) (selection) \( B_2 \)

Scheduler

(1) which vault groups the input tuples reside in, (2) the number of available PIM threads in each vault group.
We find that this heuristic is **not optimized** for PIM and leads to **sub-optimal** performance due to **three reasons:**

1. **The heuristic requires a dedicated runtime component to monitor and assign tasks**
   - The runtime component must be executed on a general-purpose PIM core

2. **The heuristic’s static mapping is limited to using only the resources available within a single vault group**
   - Can lead to performance issues for queries that operate on very large columns

3. **This heuristic is vulnerable to load imbalance**
   - Some PIM threads might finish their tasks sooner and wait idly for straggling threads
Task Scheduler: Optimized Hueristic

We optimize our heuristic to address these challenges:

1. We design a pull-based task assignment strategy, where PIM threads cooperatively pull tasks from the task queue at runtime
   - We introduce a local task queue for each vault group
   - This eliminates the need for a runtime component (first challenge) and allows PIM thread to dynamically load balance (third challenge)

2. We optimize the heuristic to allow for finer-grained tasks
   - Partition input tuples into fixed-size segments (i.e., 1000 tuples) and create an operator instance for each partition

3. We optimize the heuristic to allow a PIM thread to steal tasks from a remote vault if its local queue is empty
   - This enables us to potentially use all available PIM threads to execute tasks
Analytical Engine: Hardware Design
Given area and power constraints, it can be difficult to add enough PIM logic to each vault to saturate the available vault bandwidth. Our new data placement strategy and scheduler enables us to expose greater intra-query parallelism.

Simple programmable in-order PIM cores to exploit the available vault bandwidth.
Evaluation
Evaluation Methodology

- We heavily extend state-of-the-art transactional and analytical engines to implement HTAP baselines
  - We use **DBx1000** as the starting point for our transactional engine
  - We implement an in-house analytical engine similar to **C-store**

- We model both single- and multiple-instance
  - The system consists of **16 tables, 256K** tuples per table

- **Performance**
  - We simulate **Polynesia** using gem5, integrated with **DRAMSim2** to model an HMC-like 3D-stacked DRAM

- **Area and Energy**
  - We use **Calypto Catapult** to determine the area of the accelerators for a **22nm** process
  - We model energy as sum of the energy consumption within the **CPU, on-chip buffers, off-chip/on-chip interconnects, and DRAM**
Other Results in the Paper

- **Results for real workload analysis**
  - 1.76X/3.48X higher transactional/Analytical throughput

- **Study of each component in isolation**
  - Update propagation
  - Consistency mechanism
  - Analytical engine

- **Multiple memory stacks**
  - Polynesia significantly outperforms MI (up to 3.0X) and scales well as we increase the stack count

- **Energy analysis**
  - 48% energy reduction over MI+SW
Multiple-Instance degrades transactional throughput on average by 49.5% as it severely suffers from resource contention and data movement cost.

27.7% of the degradation comes from the update shipping latencies (data movement and merging updates from transactional threads), the remaining is from the update application process (column compression and dictionary reconstruction).

Our update propagation mechanism improves throughput by 1.8X compared to Multiple-Instance, and comes within 9.2% of Ideal.