Heterogeneous Data-Centric Architectures for Modern Data-Intensive Applications: Case Studies in Machine Learning and Databases

P&S Processing-in-Memory
Fall 2022
17 January 2023

Geraldo F. Oliveira
Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

P&S Processing-in-Memory
Fall 2022
17 January 2023

Amirali Boroumand  Saugata Ghose  Berkin Akin
Ravi Narayanaswami  Geraldo F. Oliveira  Xiaoyu Ma
Eric Shiu  Onur Mutlu

SAFARI

Carnegie Mellon  University of Illinois Urbana-Champaign  Google  ETH Zürich
Executive Summary

**Context:** We extensively analyze a state-of-the-art edge ML accelerator (Google Edge TPU) using 24 Google edge models
- Wide range of models (CNNs, LSTMs, Transducers, RCNNs)

**Problem:** The Edge TPU accelerator suffers from three challenges:
- It operates significantly below its peak throughput
- It operates significantly below its theoretical energy efficiency
- It inefficiently handles memory accesses

**Key Insight:** These shortcomings arise from the monolithic design of the Edge TPU accelerator
- The Edge TPU accelerator design does not account for layer heterogeneity

**Key Mechanism:** A new framework called Mensa
- Mensa consists of heterogeneous accelerators whose dataflow and hardware are specialized for specific families of layers

**Key Results:** We design a version of Mensa for Google edge ML models
- Mensa improves performance and energy by 3.0X and 3.1X
- Mensa reduces cost and improves area efficiency
Outline

1. Introduction

2. Edge TPU and Model Characterization

3. Mensa Framework

4. Mensa-G: Mensa for Google Edge Models

5. Evaluation

6. Conclusion
Why ML on Edge Devices?

Significant interest in pushing ML inference computation directly to 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 accelerators can significantly improve inference latency and energy consumption

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

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

- **ML Model**
- **DRAM**
- **Input Activation**
- **Parameter**
- **Output Activation**
- **Dataflow**
- **PE Array**
- **Buffer**
- **4MB on-chip buffer**
- **64x64 array**
- **2TFLOP/s**

**Introduction**

**TPU and Model Characterization**

**Mensa Framework**

**Mensa-G**

**Evaluation**

**Conclusion**
Google Edge NN Models

We analyze inference execution using 24 edge NN models

- Speech Recognition
- Face Detection
- Google Edge TPU
- Language Translation
- Image Captioning

- 6 RNN Transducers
- 13 CNN
- 2 LSTMs
- 3 RCNN

SAFARI
Introduction
TPU and Model Characterization
Mensa Framework
Mensa-G
Evaluation
Conclusion
We find that the accelerator suffers from **three major challenges:**

1. Operates significantly below its peak throughput
2. Operates significantly below its peak energy efficiency
3. Handles memory accesses inefficiently
We find that the accelerator operates significantly below its peak throughput across all models.
The accelerator operates far below its upper bound energy efficiency.

**Best CNN model:**
50.7% of upper bound energy efficiency

**LSTMs and Transducers:**
33.1% of upper bound energy efficiency

**Peak = 1.42 TFLOP/J**
Parameter traffic (off-chip and on-chip) takes a large portion of the inference energy and performance.

46% and 31% of total energy goes to off-chip parameter traffic and distributing parameters across PE array.
Major Edge TPU Challenges

We find that the accelerator suffers from three major challenges:

1. Operates significantly below its peak throughput

2. Operates significantly below its peak energy efficiency

3. Handles memory accesses inefficiently

Question: Where do these challenges come from?
Model Analysis:
Let’s Take a Deeper Look
Into the Google Edge NN 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:

**Variation in MAC intensity:** up to 200x across layers

**Variation in FLOP/Byte:** up to 244x across layers.
The key components of Google Edge TPU are completely oblivious to 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.
**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: a new acceleration framework for edge NN inference
The goal of Mensa’s software runtime scheduler is to identify which accelerator each layer in an NN model should run on. Generated once during initial setup of a system. Each of the accelerators caters to a specific family of layers. Layers tend to group together into a small number of families.
The goal of Mensa’s software runtime scheduler is to identify which accelerator each layer in an NN model should run on. 

Generated once during initial setup.
Identifying Layer Families

Key observation: the majority of layers group into a small number of layer families

Families 1 & 2: low parameter footprint, high data reuse and MAC intensity
   → compute-centric layers

Families 3, 4 & 5: high parameter footprint, low data reuse and MAC intensity
   → data-centric layers
Based on key characteristics of families, we design three accelerators to efficiently execute inference across our Google NN models.
Mensa-G: Mensa for Google Edge Models

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

**Pascal**
- 32x32 PE Array
- 32 GB/s DRAM

**Families 1&2 → compute-centric layers**
- 32x32 PE Array → 2 TFLOP/s
- 256KB Act. Buffer → 8x Reduction
- 128KB Param. Buffer → 32x Reduction
- On-chip accelerator

**Pavlov**
- 8x8 PE Array
- 256 GB/s DRAM

**Jacquard**
- 16x16 PE Array
- 256 GB/s DRAM

SAFARI
- Introduction
- TPU and Model Characterization
- Mensa Framework
- Mensa-G
- Evaluation
- Conclusion
Mensa-G: Mensa for Google Edge Models

Based on **key characteristics** of families, we design **three accelerators** to efficiently execute inference across our Google NN models.

**Pascal**
- DRAM: 32 GB/s
- 32x32 PE Array
- 32 GB/s
- 32x32 PE Array

**Families 1&2 → compute-centric layers**
- 32x32 PE Array → 2 TFLOP/s
- 256KB Act. Buffer → 8x Reduction
- 128KB Param. Buffer → 32x Reduction
- On-chip accelerator

**Pavlov**
- DRAM: 256 GB/s
- 8x8 PE Array
- 256 GB/s
- 8x8 PE Array

**Family 3 → LSTM data-centric layers**
- 8x8 PE Array → 128 GFLOP/s
- 128KB Act. Buffer → 16x Reduction
- **No** Param. Buffer → 4MB in Baseline
- Near-data accelerator

**Jacquard**
- DRAM: 256 GB/s
- 16x16 PE Array
- 256 GB/s
- 16x16 PE Array
Based on **key characteristics** of families, we design **three accelerators** to efficiently execute inference across our Google NN models.

**Pascal**
- Families 1&2 $\rightarrow$ compute-centric layers
  - 32x32 PE Array $\rightarrow$ 2 TFLOP/s
  - 256KB Act. Buffer $\rightarrow$ 8x Reduction
  - 128KB Param. Buffer $\rightarrow$ 32x Reduction
  - On-chip accelerator

**Pavlov**
- Family 3 $\rightarrow$ LSTM data-centric layers
  - 8x8 PE Array $\rightarrow$ 128 GFLOP/s
  - 128KB Act. Buffer $\rightarrow$ 16x Reduction
  - No Param. Buffer $\rightarrow$ 4MB in Baseline
  - Near-data accelerator

**Jacquard**
- Families 4&5 $\rightarrow$ non-LSTM data-centric layers
  - 16x16 PE Array $\rightarrow$ 256 GFLOP/s
  - 128KB Act. Buffer $\rightarrow$ 16x Reduction
  - 128KB Param. Buffer $\rightarrow$ 32x Reduction
  - Near-data accelerator
Based on key characteristics of families, we design three accelerators to efficiently execute inference across our Google NN models.

**Google Neural Network Models for Edge Devices:**

Analyzing and Mitigating Machine Learning Inference Bottlenecks

Amirali Boroumand†, Geraldo F. Oliveira*, Saugata Ghose‡, Xiaoyu Ma§, Berkin Akin§, Ravi Narayanaswami§, Onur Mutlu*†

†Carnegie Mellon Univ.  ○Stanford Univ.  ‡Univ. of Illinois Urbana-Champaign  §Google  *ETH Zürich

---

**Pascal**
- Families 1&2 → compute-centric layers
  - 32x32 PE Array → 2 TFLOP/s
  - 256KB Act. Buffer → 8x Reduction

**Jacquard**
- Families 4&5 → non-LSTM data-centric layers
  - 16x16 PE Array → 256 GFLOP/s
  - 128KB Act. Buffer → 16x Reduction
  - 128KB Param. Buffer → 32x Reduction

---

**SAFARI**
- Introduction
- TPU and Model Characterization
- Mensa Framework
- Mensa-G
- Evaluation
- Conclusion
Outline

1 Introduction

2 Edge TPU and Model Characterization

3 Mensa Framework

4 Mensa-G: Mensa for Google Edge Models

5 Evaluation

6 Conclusion
Energy Analysis

Baseline Google Edge TPU accelerator using a high-bandwidth off-chip memory
Mensa-G lowers on-chip/off-chip parameter traffic energy by 15.3x by scheduling layers on the accelerator with the most appropriate dataflow and memory bandwidth.

Mensa-G improves energy efficiency by 3.0X compared to the Baseline.
Mensa-G improves **throughput by 3.1X** compared to the Baseline.
More in the Paper

• Details about Mensa Runtime Scheduler

• Details about Pascal, Pavlov, and Jacquard’s dataflows

• Energy comparison with Eyeriss v2

• Mensa-G’s utilization results

• Mensa-G’s inference latency results
More in the Paper

- Details about Mensa Runtime Scheduler
- Details about Pascal, Pavlov, and Jacquard’s dataflows
- Energy comparison with Eyeriss v2
- Mensa-G’s utilization results
- Mensa-G’s inference latency results

Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

Amirali Boroumand†,‡, Geraldo F. Oliveira*, Saugata Ghose§, Xiaoyu Ma§, Berkin Akin§, Ravi Narayanaswami§, Eric Shiu§, Onur Mutlu*†

†Carnegie Mellon Univ.  ‡Stanford Univ.  §Univ. of Illinois Urbana-Champaign  $Google  *ETH Zürich
## Outline

1. Introduction
2. Edge TPU and Model Characterization
3. Mensa Framework
4. Mensa-G: Mensa for Google Edge Models
5. Evaluation
6. Conclusion

### MenSA (Mensa for Scalable AI)
- Introduction to Edge TPU and Model Characterization
- Mensa Framework
- Mensa-G: Mensa for Google Edge Models
- Evaluation
- Conclusion
**Conclusion**

**Context:** We extensively analyze a state-of-the-art edge ML accelerator (Google Edge TPU) using 24 Google edge models

- Wide range of models (CNNs, LSTMs, Transducers, RCNNs)

**Problem:** The Edge TPU accelerator suffers from three challenges:

- It operates significantly below its peak throughput
- It operates significantly below its theoretical energy efficiency
- It inefficiently handles memory accesses

**Key Insight:** These shortcomings arise from the monolithic design of the Edge TPU accelerator

- The Edge TPU accelerator design does not account for layer heterogeneity

**Key Mechanism:** A new framework called Mensa

- Mensa consists of heterogeneous accelerators whose dataflow and hardware are specialized for specific families of layers

**Key Results:** We design a version of Mensa for Google edge ML models

- Mensa improves performance and energy by 3.0X and 3.1X
- Mensa reduces cost and improves area efficiency
Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks

P&S Processing-in-Memory
Fall 2022
17 January 2023

Amirali Boroumand  Saugata Ghose  Berkin Akin
Ravi Narayanaswami  Geraldo F. Oliveira  Xiaoyu Ma
Eric Shiu  Onur Mutlu

SAFARI

Carnegie Mellon  University of Illinois Urbana-Champaign  Google  ETH Zürich
Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

P&S Processing-in-Memory
Fall 2022
17 January 2023

Amirali Boroumand
Geraldo F. Oliveira

Saugata Ghose
Onur Mutlu
Executive Summary

- **Context:** Many applications need to perform real-time data analysis using an **Hybrid Transactional/Analytical Processing (HTAP)** system
  - An ideal HTAP system should have **three properties**:
    1. **data freshness and consistency**
    2. **workload-specific optimization**
    3. **performance isolation**

- **Problem:** Prior works cannot achieve all properties of an ideal HTAP system

- **Key Idea:** Divide the system into transactional and analytical **processing islands**
  - Enables **workload-specific optimizations** and **performance isolation**

- **Key Mechanism:** Polynesia, a novel hardware/software cooperative design for in-memory HTAP databases
  - Implements **custom algorithms and hardware** to reduce the costs of **data freshness and consistency**
  - Exploits **PIM** for analytical processing to alleviate **data movement**

- **Key Results:** Polynesia outperforms three state-of-the-art HTAP systems
  - Average transactional/analytical throughput improvements of **1.7x/3.7x**
  - **48%** reduction on energy consumption
1 Introduction

2 Limitations of HTAP Systems

3 Polynesia: Overview

4 Update Propagation Mechanism

5 Consistency Mechanism

6 Analytical Engine

7 Evaluation

8 Conclusion
Real-Time Analysis

An explosive interest in many applications domains to perform data 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 is critical to analyze the transactions in real-time as the data’s value diminishes over time
Traditionally, new transactions (updates) are propagated to the analytical database using a periodic and costly process.

To support real-time analysis: a single hybrid DBMS is used to execute both transactional and analytical workloads.
An ideal HTAP system should have three properties:

1. **Workload-Specific Optimizations**
   - Transactional and analytical workloads must benefit from their own specific optimizations

2. **Data Freshness and Consistency Guarantees**
   - Guarantee access to the most recent version of data for analytics while ensuring that transactional and analytical workloads have a consistent view of data

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

Achieving all three properties at the same time is very challenging
## Outline

1. **Introduction**

2. **Limitations of HTAP Systems**

3. **Polynesia: Overview**

4. **Update Propagation Mechanism**

5. **Consistency Mechanism**

6. **Analytical Engine**

7. **Evaluation**

8. **Conclusion**
We study two major types of HTAP systems:

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

**Multiple-Instance**
- Transactions
- Analytics
- Analytics
- Replica
- Replica
- Replica

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 **high main memory contention**.
We study two major types of HTAP systems:

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

2. **Multiple-Instance**
   - Transactions
   - Analytics
   - Analytics
   - **Replica**

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 high main memory contention**
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**
   - Main Replica → Transactional Data → Analytical Snapshot
   - Main Replica

2. **Multi-Version Concurrency Control (MVCC)**
   - Main Replica → Transaction Updates
   - Time-stamped version chain

---

**SAFARI**
- Introduction
- Motivation
- Polynesia
- Update Propagation
- Consistency Mechanism
- Analytical Engine
- Evaluation
- Conclusion
Drawbacks of Snapshotting and MVCC

We evaluate the **throughput loss** caused by Snapshotting and MVCC:

**Throughput loss comes from `memcpy` operation:**
- Generates a large amount of data movement

**Throughput loss comes from long version chains:**
- Expensive time-stamp comparison and a large number of random memory accesses
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 high main memory contention.
Maintaining Data Freshness

One of the **major challenges** in multiple-instance systems is to keep **analytical replicas up-to-date**

---

**Transaction**

**Transactional queries**

- Updates

**Replica**

- Updates

**Analytical Replica**

---

**Multiple-Instance HTAP System**

---

**To maintain data freshness (via **Update Propagation**):**

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

2. **Update Application**: perform the necessary **format conversation** and apply those updates to analytical replicas
Cost of Update Propagation

We evaluate the **throughput loss** caused by Update Propagation:

Transaction throughput reduces by up to **21.2%** during the update gathering & shipping process.

Transaction throughput reduces by up to **64.2%** during the update application process.

---

**SAFARI**

- Introduction
- Motivation
- Polynesia
- Update Propagation
- Consistency Mechanism
- Analytical Engine
- Evaluation
- Conclusion
Problem and Goal

**Problems:**

1. State-of-the-art HTAP systems do not achieve 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 high main memory contention

**Goal:**

Take advantage of custom algorithm and processing-in-memory (PIM) to address these challenges
<table>
<thead>
<tr>
<th>#</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Limitations of HTAP Systems</td>
</tr>
<tr>
<td>3</td>
<td>Polynesia: Overview</td>
</tr>
<tr>
<td>4</td>
<td>Update Propagation Mechanism</td>
</tr>
<tr>
<td>5</td>
<td>Consistency Mechanism</td>
</tr>
<tr>
<td>6</td>
<td>Analytical Engine</td>
</tr>
<tr>
<td>7</td>
<td>Evaluation</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
**Polynesia**

*Key idea:* partition computing resources into two types of isolated and specialized processing islands

Isolating transactional islands from analytical islands allows us to:

1. **Apply workload-specific optimizations to each island**
2. **Avoid high main memory contention**
3. **Design efficient data freshness and consistency mechanisms without incurring high data movement costs**
   - Leverage processing-in-memory (PIM) to reduce data movement
   - PIM mitigates data movement overheads by placing computation units nearby or inside memory
Polynesia: High-Level Overview

Each island includes (1) a **replica** of data, (2) an **optimized** execution engine, and (3) a set of **hardware resources**.

Designed to sustain **bursts of updates**

Conventional **multicore CPUs** with **multi-level caches**

Designed to provide **high read throughput**

Take advantage of **PIM** to mitigate data movement bottleneck

**SAFARI**
- Introduction
- Motivation
- Polynesia
- Update Propagation
- Consistency Mechanism
- Analytical Engine
- Evaluation
- Conclusion

- 60
Outline

1 Introduction

2 Limitations of HTAP Systems

3 Polynesia: Overview

4 Update Propagation Mechanism

5 Consistency Mechanism

6 Analytical Engine

7 Evaluation

8 Conclusion
One of the **major challenges** in multiple-instance systems is to keep **analytical replicas** **up-to-date**

To maintain data freshness (via **Update Propagation**):

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

2. **Update Application**: perform the necessary **format conversation** and apply those updates to analytical replicas
Update gathering & shipping algorithm has three major stages:

- **Scan and Merge** Transactional Updates
- **Find Target Column** at Analytical Replica
- **Transfer Updates** to Analytical Replica

2\textsuperscript{nd} and 3\textsuperscript{rd} stages 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 gathering & shipping unit.

A 3-level comparator tree to merge updates

Decoupled hash computation from the hash bucket traversal to allow for concurrent hash lookups

Multiple fetch and write-back units to issue multiple memory accesses concurrently
Update Propagation: Update Application

**Goal:** perform the necessary **format conversation** and apply transactional updates to analytical replicas

**Transactional Replica**

**Analytical Replica**

**Update:** Row 2, Column 1 and 3

A simple tuple update in row-wise layout leads to multiple random accesses in column-wise layout.

Updates change encoded value in the dictionary \(\rightarrow (1)\) Need to reconstruct the dictionary, and \( (2)\) recompress the column.
We design our update application algorithm to be aware of **PIM logic** characteristics and constraints.

- **Build Update Dict.**
  - Updates → Sort → Update Dict.
  - We maintain a hash index that links the old encoded value in a column to the new encoded value.

- **Build New Dict. and Index**
  - Dict. + Update Dict. → New Dict. → Index
  - Avoids the need to decompress the column and add updates, eliminating data movement and random accesses to 3D DRAM.

- **New Compressed Col.**
  - Old Col. Value → Location in New Dict. → New Dict. → Encoded Value

---

**SAFARI**

- Introduction
- Motivation
- Polynesia
- Update Propagation
- Consistency Mechanism
- Analytical Engine
- Evaluation
- Conclusion
We design a **hardware implementation of our algorithm**, and add it to each **in-memory analytical island**

---

**Sort Unit**

- FIFOs
- 1024-Bitonic Sorter Network

**Hash Lookup Unit**

- Front-End Engine
- Probe Units

**Merge Unit**

- FIFOs
- Comparator Tree

---

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

---

**Update Application: Hardware**

- **Introduction**
- **Motivation**
- **Polynesia**
- **Update Propagation**
- **Consistency Mechanism**
- **Analytical Engine**
- **Evaluation**
- **Conclusion**
### Outline

1. Introduction
2. Limitations of HTAP Systems
3. Polynesia: Overview
4. Update Propagation Mechanism
5. **Consistency Mechanism**
6. Analytical Engine
7. Evaluation
8. Conclusion
Consistency Mechanism: Algorithm

For each column, there is a chain of snapshots where each chain entry corresponds to a version of the 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 row.

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 writeback units to issue multiple memory accesses concurrently.

Look-ups at the tracking buffer limit performance → use a hash index to alleviate performance bottlenecks.

Track outstanding reads, as they may come back from memory out of order. Allows to immediately initiate a write after a read is complete.
Efficient analytical query execution strongly depends on:

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

The execution of physical operators of analytical queries significantly benefit from PIM.

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

**Problem:** how to partition analytical data across vaults of the 3D-stacked memory

Limits the **area/power/bandwidth** available to the analytical engine inside a vault

Creates **inter-vault communication overheads**

- **Local**
  - Vault 1
  - Vault 2
  - Vault 3
  - Vault 4

- **Distributed**
  - Vault 1
  - Vault 2
  - Vault 3
  - Vault 4

- **Hybrid**
  - Vault 1
  - Vault 2
  - Vault 3
  - Vault 4

Increases the **aggregate bandwidth** for servicing each query by 4 times, and provides up to 4 times the power/area for PIM logic compared to Local
Analytical Engine: Query Execution

Other details in the paper:

Task scheduling policy

We design a pull-based task assignment strategy, where PIM threads cooperatively pull tasks from the task queue at runtime.

How each physical operator is executed

We employ the top-down Volcano (Iterator) execution model to execute physical operations (e.g., scan, filter, join) while respecting operator’s dependencies.
Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand†  Saugata Ghose◊  Geraldo F. Oliveira‡  Onur Mutlu‡

†Google  ◊Univ. of Illinois Urbana-Champaign  ‡ETH Zürich

We employ the top-down Volcano (Iterator) execution model to execute physical operations (e.g., scan, filter, join) while respecting operator’s dependencies.
Methodology

- We adapt previous transactional/analytical engines with our new algorithms
  - DBx1000 for transactional engine
  - C-store for analytical engine

- We use **gem5** to simulate Polynesia
  - Available at: [https://github.com/CMU-SAFARI/Polynesia](https://github.com/CMU-SAFARI/Polynesia)

- We compare **Polynesia** against:
  - Single-Instance-Snapshotting (**SI-SI**)
  - Single-Instance-MVCC (**SI-MVCC**)
  - Multiple-Instance + Polynesia’s new algorithms (**MI+SW**)
  - **MI+SW+HB**: MI+SW with a 256 GB/s main memory device
  - **Ideal-Txn**: the peak transactional throughput if transactional workloads run in isolation
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 consistency mechanism.
Polynesia comes within 8.4% of ideal Txn because it uses custom PIM logic for data freshness/consistency mechanisms, significantly reducing main memory contention and data movement.
MI+SW+HB is the best software-only HTAP for analytical workloads, because it provides workload-specific optimizations, but it still loses 35.3% of the analytical throughput due to high main memory contention.
Polynesia improves over MI+SW+HB by 63.8%, by eliminating data movement, and using custom logic for update propagation and consistency.
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 is an energy-efficient HTAP system, reducing energy consumption by 48%, on average across prior works.
More in the Paper

• Real workload analysis

• Effect of the update propagation technique

• Effect of the consistency mechanism

• Effect of the analytical engine

• Effect of the dataset size

• Area Analysis
More in the Paper

• Real workload analysis
• Effect of the update propagation technique
• Effect of the consistency mechanism
• Effect of the analytical engine
• Effect of the dataset size
• Area Analysis

Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand†  Saugata Ghose◊  Geraldo F. Oliveira‡  Onur Mutlu‡
†Google  ◊Univ. of Illinois Urbana-Champaign  ‡ETH Zürich
1. Introduction

2. Limitations of HTAP Systems

3. Polynesia: Overview

4. Update Propagation Mechanism

5. Consistency Mechanism

6. Analytical Engine

7. Evaluation

8. Conclusion
Conclusion

- **Context:** Many applications need to perform real-time data analysis using an Hybrid Transactional/Analytical Processing (HTAP) system
  - An ideal HTAP system should have **three properties:**
    1. data freshness and consistency,
    2. workload-specific optimization,
    3. performance isolation

- **Problem:** Prior works cannot achieve all properties of an ideal HTAP system

- **Key Idea:** Divide the system into transactional and analytical **processing islands**
  - Enables workload-specific optimizations and performance isolation

- **Key Mechanism:** Polynesia, a novel hardware/software cooperative design for in-memory HTAP databases
  - Implements **custom algorithms and hardware** to reduce the costs of data freshness and consistency
  - Exploits **PIM** for analytical processing to alleviate data movement

- **Key Results:** Polynesia outperforms three state-of-the-art HTAP systems
  - Average transactional/analytical throughput improvements of **1.7x/3.7x**
  - **48%** reduction on energy consumption
Polynesia:
Enabling High-Performance and Energy-Efficient
Hybrid Transactional/Analytical Databases
with Hardware/Software Co-Design

P&S Processing-in-Memory
Fall 2022
17 January 2023

Amirali Boroumand
Geraldo F. Oliveira

Saugata Ghose
Onur Mutlu

SAFARI Google University of Illinois ETH Zürich
Heterogeneous Data-Centric Architectures for Modern Data-Intensive Applications: Case Studies in Machine Learning and Databases

P&S Processing-in-Memory
Fall 2022
17 January 2023

Geraldo F. Oliveira