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

Geraldo F. Oliveira
Saugata Ghose
Amirali Boroumand
Juan Gómez-Luna
Onur Mutlu

ISVLSI 2022

SAFARI Google UNIVERSITY OF ILLINOIS ETH Zürich
## Outline

### 1 Introduction

### 2 Mensa: Accelerating Google Neural Networks
- Edge TPU and Model Characterization
- Mensa Framework
- Evaluation
- Conclusion

### 3 Polynesia: Accelerating HTAP Systems
- HTAP Systems Characterization
- Polynesia: Overview
- Evaluation
- Conclusion
Data movement bottlenecks happen because of:

- Not enough data locality → ineffective use of the cache hierarchy
- Not enough memory bandwidth
- High average memory access time
Data Movement Bottlenecks (2/2)

Compute-Centric Architecture

Memory-Centric Architecture

1. Abundant DRAM bandwidth
   - Off-Chip Link

2. Shorter memory latency
   - Processing-in-Memory (PIM)

SAFARI
When to Employ PIM

Mobile consumer workloads
(\textit{GoogleWL}^2)

Graph processing
(\textit{Tesseract}^1)

Neural networks
(\textit{GoogleWL}^2)

Databases
(\textit{Polynesia}^5)

DNA sequence mapping
(\textit{GenASM}^3; \textit{GRIM-Filter}^4)

Time series analysis
(\textit{NATSA}^6)

\ldots


SAFARI
**Drawbacks and Limitations of PIM**

PIM designs are restricted by low **area** and **power budgets**, manufacturing challenges, and limited **clock frequencies**.

To avoid **subpar performance**, an **efficient PIM architecture needs** to take into consideration **PIM constraints**.

---

**Co-designing hardware and software** to take advantage of PIM properties while mitigating its **shortcomings** can lead to a **better system design**.
HW/SW Co-Design for PIM

We follow a two-step approach to co-design software and hardware to efficiently take advantage of PIM paradigm.

**Step 1:** Application Profiling

- **HW/SW requirements**
- **performance bottleneck**
- **energy bottleneck**

**Step 2:** Co-design SW and HW

- high-performance and energy-efficient PIM architecture

**Target Application**

We showcase our two-step approach for two applications:

1. **Machine learning inference models** for edge devices
2. **Hybrid transactional/analytical processing databases** for cloud systems
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.

Speech Recognition

Face Detection

RNN

Transducers

CNN

LSTMs

Language Translation

Image Captioning

RCNN

SAFARI

Introduction

TPU and Model Characterization

Mensa Framework

Evaluation

Conclusion
Outline

1 Introduction

2 Mensa: Accelerating Google Neural Networks
   Edge TPU and Model Characterization
   Mensa Framework
   Evaluation
   Conclusion

3 Polynesia: Accelerating HTAP Systems
   HTAP Systems Characterization
   Polynesia: Overview
   Evaluation
   Conclusion
Edge TPU: Baseline Accelerator

ML Model

Input Activation

Parameter

Output Activation

Dataflow

PE Array

4MB on-chip buffer

64x64 array

2TFLOP/s

DRAM
We analyze inference execution using 24 edge NN models
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
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:

![Graphs showing variation in MACs and FLOP/Byte across layers for CNN5 and CNN13]

Variation in MAC intensity: up to 200x across layers

Variation in FLOP/Byte: up to 244x across layers
Root Cause of Accelerator Challenges

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: Accelerating Google Neural Networks

1. **Introduction**

2. **Mensa Framework**

3. **Evaluation**

4. **Conclusion**

# Polynesia: Accelerating HTAP Systems

1. **Introduction**

2. **Polynesia: Overview**

3. **Evaluation**

4. **Conclusion**
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
Mensa High-Level Overview

Edge TPU Accelerator

Model A  Model B  Model C

Mensa

Model A  Model B  Model C

Buffer  NoC

Monolithic Accelerator

Family 1  Family 2  Family 3

Acc. 1  Acc. 2  Acc. 3

SAFARI

Introduction  TPU and Model Characterization  Mensa Framework  Evaluation  Conclusion

23
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.
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.

**Pascal**
- Act. Buffer
- Param. Buffer
- 32x32 PE Array
- DRAM
- 32 GB/s

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

**Jacquard**
- Act. Buffer
- Param. Buffer
- 16x16 PE Array
- DRAM
- 256 GB/s
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
- 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

**Jacquard**
- 16x16 PE Array
- 256 GB/s
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 → 2 TFLOP/s
- 256KB Act. Buffer → 8x Reduction
- 128KB Param. Buffer → 32x Reduction
- On-chip accelerator

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

**Jacquard**
- 16x16 PE Array

---

**SAFARI**

- Introduction
- TPU and Model Characterization
- Mensa Framework
- Evaluation
- Conclusion

---

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

**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
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**
- DRAM (32 GB/s)
- **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**
- DRAM (256 GB/s)
- **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
- **16x16 PE Array**
- DRAM (256 GB/s)
- **Families 4&5 → non-LSTM data-centric layers**
  - 16x16 PE Array → 256 GFLOP/s
  - 128KB Act. Buffer → 16x Reduction
  - 128KB Param. Buffer → 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**
- 32x32 PE Array → 2 TFLOP/s
- 256KB Act. Buffer → 8x Reduction
- 32 GB/s

**Jacquard**
- 16x16 PE Array → 256 GFLOP/s
- 128KB Act. Buffer → 16x Reduction
- 128KB Param. Buffer → 32x Reduction
- Near-data accelerator

**Families 1&2 → compute-centric layers**

- 32x32 PE Array → 2 TFLOP/s
- 256KB Act. Buffer → 8x Reduction

**Families 4&5 → non-LSTM data-centric layers**

- 16x16 PE Array → 256 GFLOP/s
- 128KB Act. Buffer → 16x Reduction
- 128KB Param. Buffer → 32x Reduction
- Near-data accelerator
Outline

1 Introduction

2 Mensa: Accelerating Google Neural Networks
   - Edge TPU and Model Characterization
   - Mensa Framework
   - Evaluation
   - Conclusion

3 Polynesia: Accelerating HTAP Systems
   - HTAP Systems Characterization
   - Polynesia: Overview
   - Evaluation
   - Conclusion
**Energy Analysis**

- **Total Static**
- **Act Buffer + NoC**
- **PE**
- **Off-chip Interconnect**
- **Param Buffer + NoC**
- **DRAM**

**Normalized Energy**

- **Baseline**
- **Base + HB**
- **Mensa**

**Models**
- LSTM1
- Transd.1
- Transd.2
- CNN5
- CNN9
- CNN10
- CNN12
- RCNN1
- RCNN3
- Average

**Baseline Google Edge TPU accelerator**

**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
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
<table>
<thead>
<tr>
<th>Outline</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Mensa: Accelerating Google Neural Networks</td>
</tr>
<tr>
<td></td>
<td>Edge TPU and Model Characterization</td>
</tr>
<tr>
<td></td>
<td>Mensa Framework</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
</tr>
<tr>
<td>3</td>
<td>Polynesia: Accelerating HTAP Systems</td>
</tr>
<tr>
<td></td>
<td>HTAP Systems Characterization</td>
</tr>
<tr>
<td></td>
<td>Polynesia: Overview</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
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

Self-Driving Cars

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
State-of-the-Art HTAP Systems

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**
State-of-the-Art HTAP Systems

We study two major types of HTAP systems:

- **Single-Instance**
  - Transactions
  - Analytics

- **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**
   - Transactions
     - Main Replica
   - Transactional Data
   - Snapshot
   - Analytical Snapshot

2. **Multi-Version Concurrency Control (MVCC)**
   - Transactions
     - Main Replica
   - Analytics
     - Column
     - Transaction Updates
     - Time-stamped version chain
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
State-of-the-Art HTAP Systems

We study two major types of HTAP systems:

- **Single-Instance**
  - Main Replica

- **Multiple-Instance**
  - Transactions
  - Analytics
  - 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 high main memory contention
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
We evaluate the **throughput loss** caused by Update Propagation:

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

**Transactional throughput reduces by up to 64.2%** during the update application process.
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>1</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Mensa: Accelerating Google Neural Networks</td>
</tr>
<tr>
<td></td>
<td>Edge TPU and Model Characterization</td>
</tr>
<tr>
<td></td>
<td>Mensa Framework</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
</tr>
<tr>
<td></td>
<td>Conclusion</td>
</tr>
<tr>
<td>3</td>
<td>Polynesia: Accelerating HTAP Systems</td>
</tr>
<tr>
<td></td>
<td>HTAP Systems Characterization</td>
</tr>
<tr>
<td></td>
<td>Polynesia: Overview</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
</tr>
<tr>
<td></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

**Analytical Island**

- **Analytical Engine**
  - PIM Core
  - PIM Core
  - PIM Core
  - PIM Core

- **Update Propagation Mechanism**
  - Update Gathering and Shipping Unit
  - Update Application Unit

- **Consistency Mechanism**
  - Copy Unit

**Transactional Island**

- **Transactional Engine**
  - CPU
  - CPU
  - CPU
  - CPU

- **Shared Last-Level Cache (LLC)**

- **Processor**

- **Off-Chip Link**

- **3D-Stacked Memory**

- **Memory Controller**
Designed to sustain bursts of updates

Designed to provide high read throughput

Conventional multicore CPUs with multi-level caches

Take advantage of PIM to mitigate data movement bottleneck

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

Transational Island

Analytical Island

Update Propagation Mechanism

Update Gathering and Shipping Unit

Update Application Unit

Consistency Mechanism

Copy Unit

SAFARI

Introduction

HTAP Systems Characterization

Polynesia

Evaluation

Conclusion
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.

Designed to provide high read throughput.

Conventional multicore CPUs with multi-level caches.

Take advantage of PIM to mitigate data movement bottleneck.

Polynesia: High-Level Overview

- Designed to sustain bursts of updates
- Designed to provide high read throughput
- Conventional multicore CPUs with multi-level caches
- Take advantage of PIM to mitigate data movement bottleneck
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 replicas.

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

1. **Scan and Merge Transactional Updates**
   - Update Logs
     - Tnx. 1
     - Tnx. 2
     - ... (omitted)
     - Tnx. N
   - Merge + Sort

2. **Find Target Column at Analytical Replica**
   - Final Update Log
   - Update \( k \)
   - Hash Table
   - Target Column

3. **Transfer Updates to Analytical Replica**
   - Update \( k \)
   - Copy
   - Column \( i \)
   - Buffer

**2^{nd} and 3^{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
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

Designed to provide high read throughput

Conventional multicore CPUs with multi-level caches

Take advantage of PIM to mitigate data movement bottleneck

Polynesia

TAKE A D VANTAGE OF PIM TO MITIGATE DATA MOVEMENT BOTTLENECK
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

Analytical Island

Polynesia: High-Level Overview

DRAM Banks

Off-Chip Link

TSV

3D-Stacked Memory

Analytical Engine

Update Propagation Mechanism

Update Application Unit

Update Gathering and Shipping Unit

Consistency Mechanism

Memory Controller

Copy Unit

Introduction

HTAP Systems Characterization

Polynesia

Evaluation

Conclusion
## Outline

### 1 Introduction

### 2 Mensa: Accelerating Google Neural Networks
- Edge TPU and Model Characterization
- Mensa Framework
- Evaluation
- Conclusion

### 3 Polynesia: Accelerating HTAP Systems
- HTAP Systems Characterization
- Polynesia: Overview
- Evaluation
- Conclusion
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

• 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
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.
End-to-End System Analysis (2/3)

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 consumes $0.4x/0.38x/0.5x$ the energy of SI-SS/SI-MVCC/MI+SW since Polynesia eliminates a large fraction (30%) of off-chip DRAM accesses.

Polynesia is an energy-efficient HTAP system, reducing energy consumption by 48%, on average across prior works.
1 Introduction

2 Mensa: Accelerating Google Neural Networks
   - Edge TPU and Model Characterization
   - Mensa Framework
   - Evaluation
   - Conclusion

3 Polynesia: Accelerating HTAP Systems
   - HTAP Systems Characterization
   - Polynesia: Overview
   - Evaluation
   - 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**, and
    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
Heterogeneous Data-Centric Architectures for Modern Data-Intensive Applications: Case Studies in Machine Learning and Databases

Geraldo F. Oliveira
Saugata Ghose
Juan Gómez-Luna
Amirali Boroumand
Onur Mutlu

ISVLSI 2022

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

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

PACT 2021
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**
- **64x64 array**
- **2TFLOP/s**
- **4MB on-chip buffer**

**Introduction**

- TPU and Model Characterization
- Mensa Framework
- Mensa-G
- Evaluation
- Conclusion
We analyze inference execution using 24 edge NN models

- **Speech Recognition**
  - 6 RNN Transducers
- **Face Detection**
  - 13 CNN
- **Image Captioning**
  - 3 RCNN
- **Language Translation**
  - 2 LSTMs

Google Edge TPU
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
We find that the accelerator operates significantly below its peak throughput across all models.

- CNNs and RCNNs: only 52.2% of peak throughput
- LSTMs and Transducers: less than 1% of peak throughput
(2) Low Energy Efficiency

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.
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

![Graph showing FLOP/Byte vs Parameter Footprint (MB) for different 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:

![Graphs showing variation in MACs and FLOP/Byte across layers]

Variation in MAC intensity: up to 200x across layers

Variation in FLOP/Byte: up to 244x across layers
Root Cause of Accelerator Challenges

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
<table>
<thead>
<tr>
<th>1</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Edge TPU and Model Characterization</td>
</tr>
<tr>
<td>3</td>
<td>Mensa Framework</td>
</tr>
<tr>
<td>4</td>
<td>Mensa-G: Mensa for Google Edge Models</td>
</tr>
<tr>
<td>5</td>
<td>Evaluation</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
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.

Mensa Runtime Scheduler

Accelerator characteristics

Layer characteristics

Scheduler

NN model

Layer Mapping
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.

Each of the accelerators caters to a specific family of layers. Layers tend to group together into a small number of families.

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

Amirali Boroumand\textsuperscript{†} Geraldo F. Oliveira* Saugata Ghose\textsuperscript{‡} Xiaoyu Ma§ Berkin Akin§ Ravi Narayanaswami\textsuperscript{§}

\textsuperscript{†}Carnegie Mellon Univ. \textsuperscript{‡}Stanford Univ. \textsuperscript{‡}Univ. of Illinois Urbana-Champaign \textsuperscript{§}Google \textsuperscript{*}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
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
- DRAM 32 GB/s

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

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

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

**Pascal**
- 32x32 PE Array → 2 TFLOP/s
- 256KB Act. Buffer → 8x Reduction
- 128KB Param. Buffer → 32x Reduction
- On-chip accelerator

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

**Jacquard**
- 16x16 PE Array

---

**Mensa-G: Mensa for Google Edge Models**

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

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

**Jacquard**
- 16x16 PE Array → 256 GFLOP/s
- 128KB Act. Buffer → 16x Reduction
- 128KB Param. Buffer → 32x Reduction
- Near-data accelerator
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
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
- Near-data accelerator

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§

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

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
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
Outline

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

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

PACT 2021
Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design

Amirali Boroumand
Geraldo F. Oliveira

Saugata Ghose
Onur Mutlu

ICDE
2022
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

SAFARI
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

Self-Driving Cars

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.
Ideal HTAP System Properties

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 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**
State-of-the-Art HTAP Systems

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**
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

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

**T1:** 54
**T2:** 13
**T3:** 84
**T1:** 12
**T2:** 7
**T1:** 10
**T2:** 8

**Column**
- 4
- 1
- 8
- 3
- 2
- 3

**Transaction Updates**
Drawbacks of Snapshotting and MVCC

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

### Throughput loss

**Snapshotting**

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

**MVCC**

Throughput loss comes from **long version chains**:
- Expensive time-stamp comparison and a large number of random memory accesses
State-of-the-Art HTAP Systems

We study two major types of HTAP systems:

- **Single-Instance**
  - Main Replica

- **Multiple-Instance**
  - 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.
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.
Cost of Update Propagation

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

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

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

Designed to provide high read throughput

Take advantage of PIM to mitigate data movement bottleneck
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

Update gathering & shipping algorithm has **three major stages:**

1. **Scan and Merge**
   - Transactional Updates
   - **Update Logs**
   - Tnx. 1
   - Tnx. 2
   - ... (ellipses)
   - Tnx. N

2. **Find Target Column**
   - at Analytical Replica

3. **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
**Goal:** perform the necessary **format conversation** and **apply** transactional updates to analytical replicas

**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 → (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.

We maintain a hash index that links the old encoded value in a column to the new encoded value.

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 as our update gathering & shipping unit.
| 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.
Consistency Mechanism: Hardware

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.
# 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**
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

Creates inter-vault communication overheads

Local

Distributed

Hybrid

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

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

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

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
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.
Analytical Engine: Query Execution

Other details in the paper:

Task scheduling policy

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

Amirali Boroumand†
†Google

Saugata Ghose○
○Univ. of Illinois Urbana-Champaign

Geraldo F. Oliveira‡
‡ETH Zürich

Onur Mutlu‡

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.
End-to-End System Analysis (2/5)

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 consumes $0.4x/0.38x/0.5x$ the energy of SI-SS/SI-MVCC/MI+SW since Polynesia eliminates a large fraction (30%) of off-chip DRAM accesses.

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

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

Amirali Boroumand†
†Google

Saugata Ghose◊
◊Univ. of Illinois Urbana-Champaign

Geraldo F. Oliveira‡
‡ETH Zürich

Onur Mutlu‡
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
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

Amirali Boroumand
Geraldo F. Oliveira
Saugata Ghose
Onur Mutlu

ICDE 2022