Computation-in-Memory Accelerators for Secure Graph Database: Opportunities and Challenges

Md Tanvir Arafin[1]

[1]Morgan State University, Baltimore, USA

January 18 2022

Outline

- Accelerating Graph Databases (GDB): Motivation & Background
- Common Hardware Architecture for GDB Acceleration
- Application of CIM in GDB acceleration
- GDB security
- CIM for Secure GDB

Graph Databases

Key Question

How to securely process large-scale graph data?

Why GDBs

- Knowledge graph for AI
- Chemical and biological systems data
- Social network data
- Graph neural networks

Common issues in GDBs

- Data-centric computation
- Under-optimized hardware
- Costly memory operations
- Data privacy, security

Current Solutions

- Move to faster computation
  i.e., arge scale GDBs hosted by cloud providers

Current Solutions

- Move to faster computation
  i.e., arge scale GDBs hosted by cloud providers
- Move to faster memory
  i.e., SSD, NVMe

Current Solutions

- Move to faster computation
  i.e., arge scale GDBs hosted by cloud providers
- Move to faster memory
  i.e., SSD, NVMe
- Graph analysis as a service (GaaS)

Current Solutions

- Move to faster computation
  i.e., arge scale GDBs hosted by cloud providers
- Move to faster memory
  i.e., SSD, NVMe
- Graph analysis as a service (GaaS)
- Data privacy, security etc., provider dependent

Why CIM

Motivation

- Large Memory Bandwidth (~TB/s)
- Application Specific Performance Gain (~10-100x acceleration)
- Novel Memory Primitives (RRAM, PCM, STT-MRAM)
- Efficient computation for security primitives

Goal

Explore the opportunities of CIM for GDBs

Graph Model

Edge-labeled Graph Model

– Edge labels: Representing different relationships between the nodes
– Nodes and edges stored as adjacency matrices or adjacency lists
– Example: Neo4j, OrientDB, and MS Graph Engine

Example

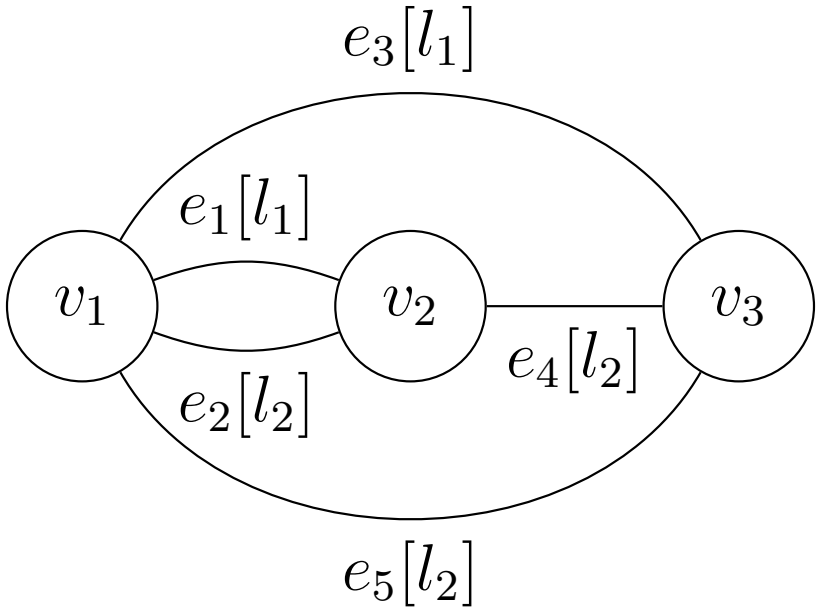


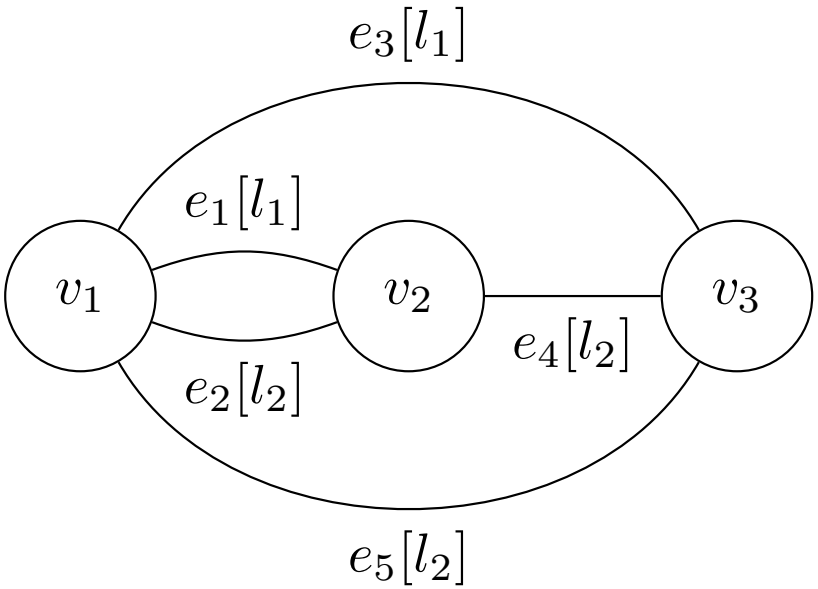
Figure: Example of an edge-labelled graph. Labels for each edge is given inside the square brackets.

Data Store Model

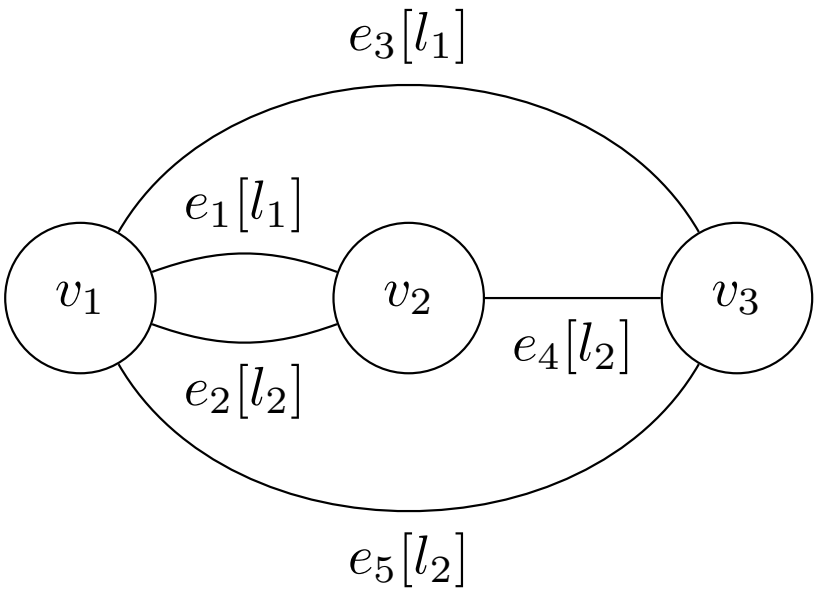
Edge-labeled Graph Model

– Each vertex record contains
    (1) A pointer to the first edge
    (2) Additional metadata such as a pointer to vertex properties
– Each edge record contains
    (1) Label for relationship between the connected vertices
    (2) Pointers to the vertex pairs connected by the edge
    (3) Pointers for previous and next edges for each vertex

Example



Example



Vertex	First Edge, Metadata
v_1	e_1, m_1
v_2	e_1, m_2
v_3	e_3, m_3

Edge	Label, Vertices, (EP_1), (EP_2),
e_1	$l_1, (v_1, v_2), (-, e_2), (-, e_2)$
e_2	$l_2, (v_1, v_2), (e_1, e_3), (e_1, e_4)$
e_3	$l_1, (v_1, v_3), (e_2, e_5), (-, e_4)$
e_4	$l_2, (v_2, v_3), (e_2, -), (e_3, e_5)$
e_5	$l_2, (v_1, v_3), (e_3, -), (e_4, -)$

Graph Traversal

- Datastore model can be implemented with doubly-linked lists
- Vertex and edge values are memory locations
- Fixed-size entry for each row in the vertex and edge table
- Traversing the graph == traversing a doubly-linked list

Graph Traversal

- Datastore model can be implemented with doubly-linked lists
- Vertex and edge values are memory locations
- Fixed-size entry for each row in the vertex and edge table
- Traversing the graph == traversing a doubly-linked list

Pointer chasing $\rightarrow$ fundamental operation for GDB

Graph Traversal

- Datastore model can be implemented with doubly-linked lists
- Vertex and edge values are memory locations
- Fixed-size entry for each row in the vertex and edge table
- Traversing the graph == traversing a doubly-linked list

Pointer chasing $\rightarrow$ fundamental operation for GDB

Pointer chasing is slow

CIM: Key Concepts

Move computation to the memory component.

Two flavors

- Near-memory processing
- In-memory processing

Near Memory Computing

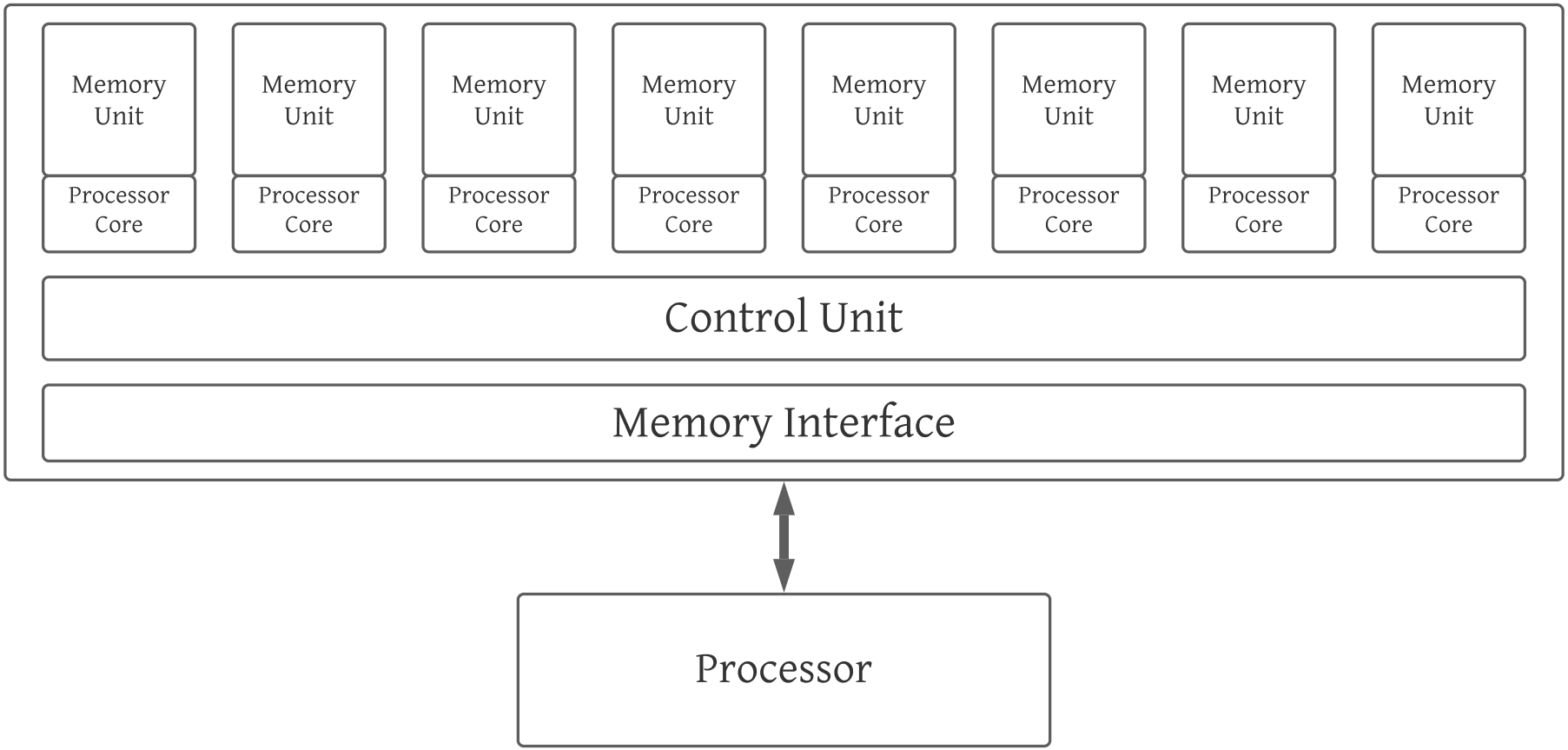


Figure: Near-Memory Computing

Near Memory Computing

- DRAM-based Processing Units (DPU) (i.e., UPMEM)

Figure: UPMEM DPU. Source: https://www.upmem.com

In-Memory Computing

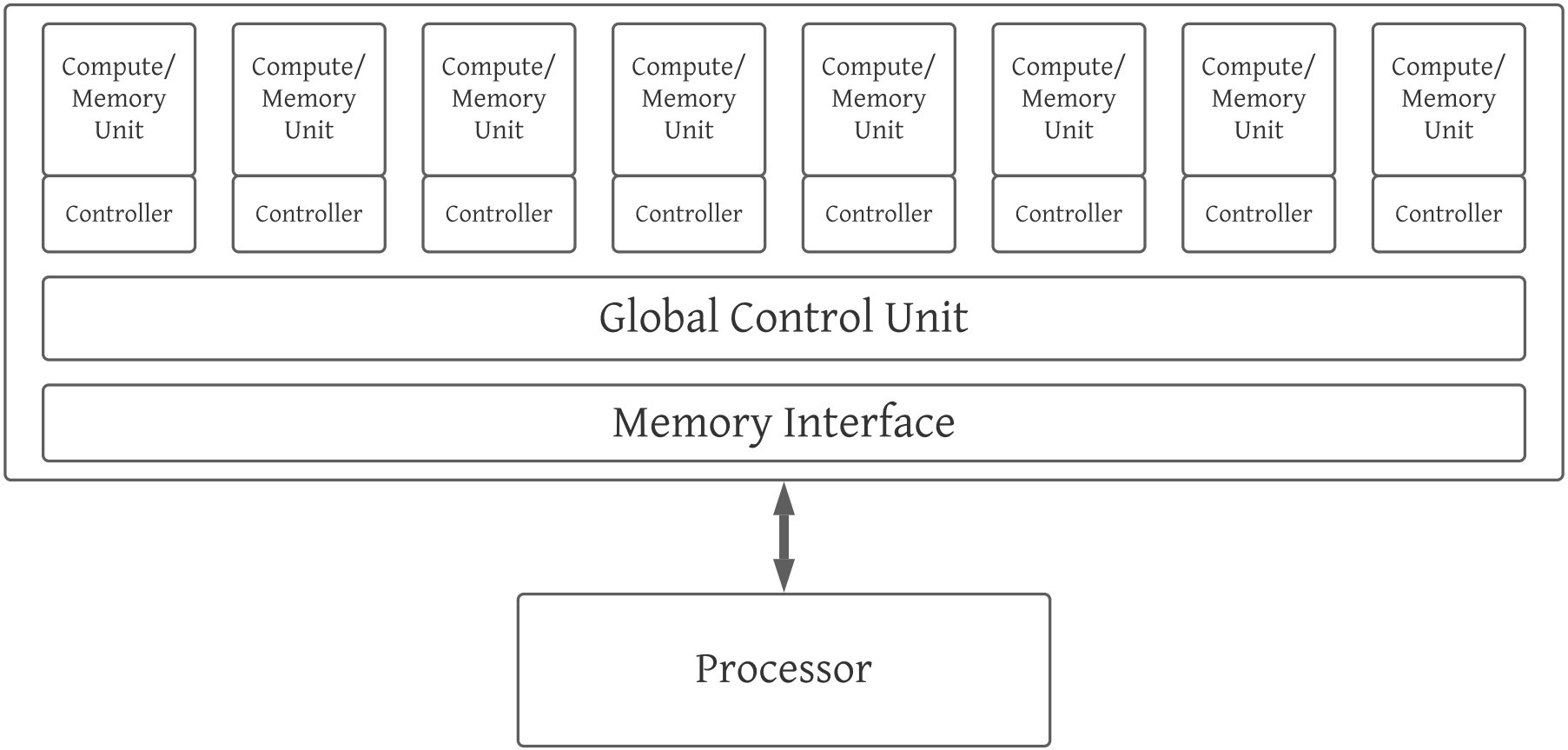


Figure: In-Memory Computing

In-Memory Computing Example

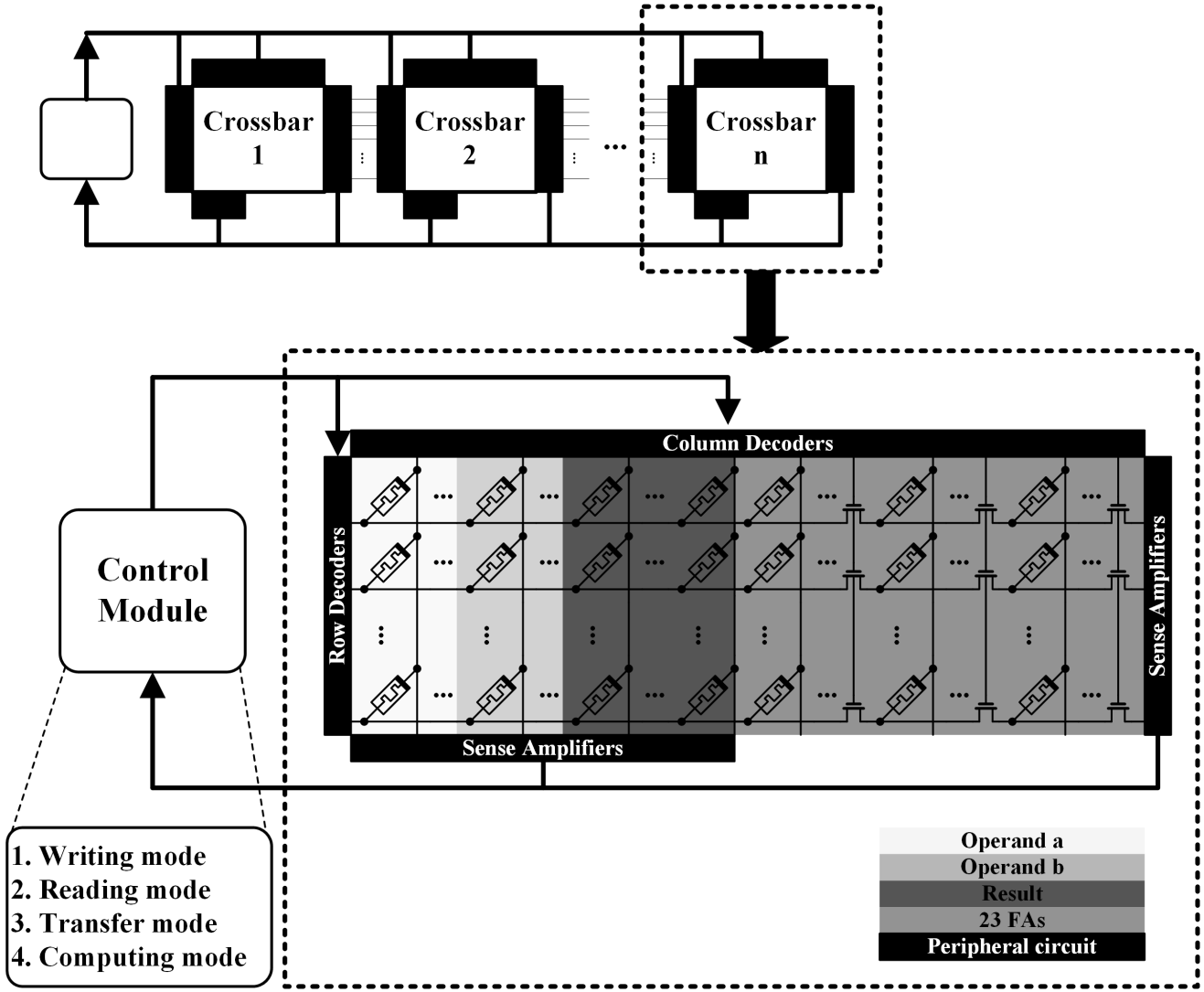


Figure: In-Memory computing example: details of a compute/memory unit.

Acceleration of GDBs

Problems

- Large graph size + poorly designed GDBs → notoriously slow
- Random memory access pattern → traversal and neighborhood-related queries cache-inefficient

Acceleration of GDBs

Problems

- Large graph size + poorly designed GDBs → notoriously slow
- Random memory access pattern → traversal and neighborhood-related queries cache-inefficient

Solutions

- Faster main memory
- Custom hardware accelerators
- Graphical processing units (GPUs)
- Computation-in-memory

Hardware Accelerators

- Dedicated hardware accelerators for fast and parallel graph processing
- Graph problems are highly parallel
- Proper partitioning can provide significant speedup
- Common Designs
  - Gather-scatter accelerators
  - Streaming-based Solutions

Gather-Scatter Accelerators

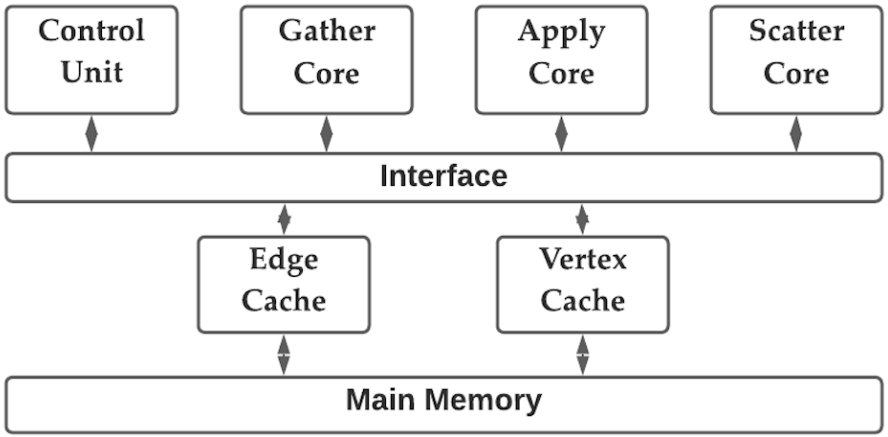
Key Idea

Graph algorithms broken in gather-scatter steps [Ozada *et al.*]

.

Design

- Accelerators for gather, apply, and scatter operations
- Edge and vertex caches



Streaming-based Solution

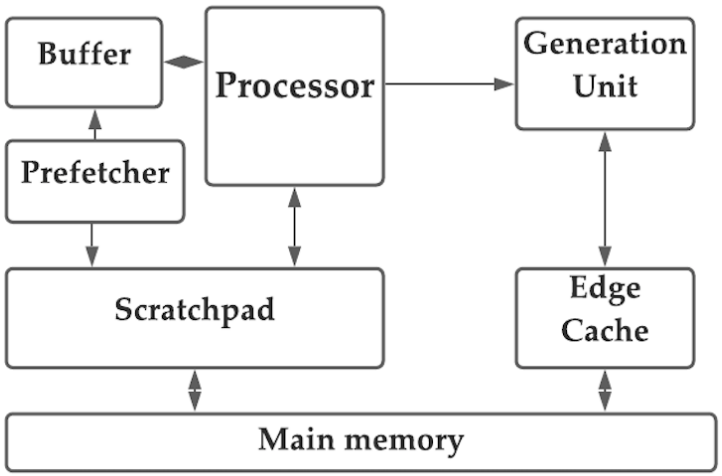
Idea

Exploit the locality of access principle

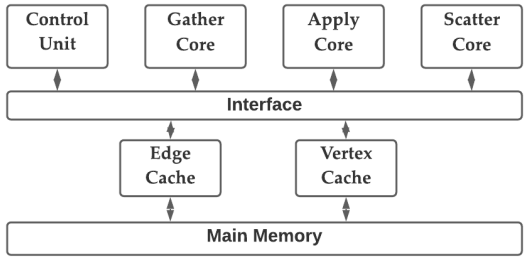
Example: GraphPulse

Design

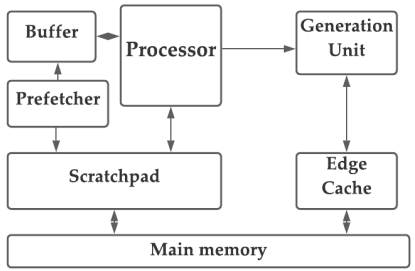
- On-chip memory queues
- Prefetcher
- Edge/vertex caches



Hardware Accelerators



(a)

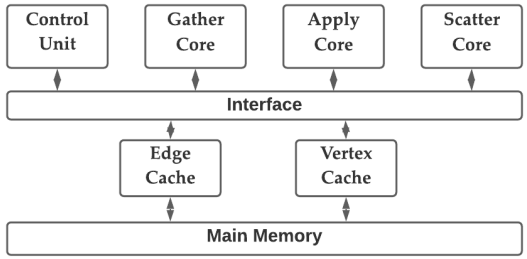


(b)

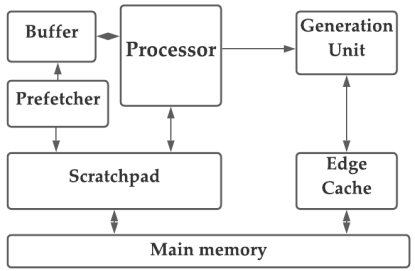
Benefits

- Low power
- Significant speedup from the native CPU-based solution

Hardware Accelerators



(a)



(b)

Benefits

- Low power
- Significant speedup from the native CPU-based solution

Problem

Effectiveness dependent on communication speed with the main memory

GPU

GPUs suffer worse from long latency memory operations in graph applications than non-graph applications

GPU

GPUs suffer worse from long latency memory operations in graph applications than non-graph applications Solutions

- Moving the memory near to the GPU kernels
- Heterogeneous memory access for CPU and GPU
- Improving cache operation
- Replacing the PCIe bus with a faster and direct memory access

CIM-based Solutions

Recent Advances

- Ahn *et al.*: CIM offer faster and more efficient pointer chasing
- Choe *et. al.*: Concurrent data structures link-lists, skip lists, and FIFO queues enjoy significant throughput improvement in CIM

Issues

- No solution for traditional GDBs
- Security issues are ignored

Security Issues in GDB

GDBs contain a wide range of issues from basic authentication vulnerability to remote code executions

Table: List of recently discovered vulnerabilities in commodity GDBs

Attack Type	GDB	Publication
Privilege Escalation	Neo4j v4.2 & v4.3	CVE-2021-34802
Remote code execution	Neo4j 3.4.18 OrientDB 2.2.22	CVE-2021-34371 CVE-2017-11467
Authentication	Neo4j 3.4.x OrientDB	CVE-2018-18389 CVE-2019-25021
Code injection	SAP OrientDB v3.0	CVE-2020-6230
Denial-of-service	RedisGraph v2.2.1	CVE-2020-35668

Threat Model

Multiple Stakeholders

- Data Owner: Provides private data
- Database Owner: Maintains the GDB
- Cloud Service Provider: Queries the DB to provide relevant information to the user
- User

Issue

How can different stakeholders provide efficient computation and security guarantee to each other?

Security Solutions

Secure GDB

Provide privacy-preserving data store and query response

Security Solutions

- CryptDB : SQL Based
- Fully Homomorphic Encryption
- SecGDB: Structural Encryption
- Oblivious Filter

Open Problems in Securing GBDs

C1. Computation overhead

C2. Weak-security models

C3. Non-uniform standards

Proposed Solution

CIM for Securing GDBs

Proposed Solution

CIM for Securing GDBs

Key Idea

- CIM provide in-memory multiplication and parallel logic operation for FHE or multiparty computation-based security primitives
- CIM supports faster pointer chasing

Proposed Solution

Architecture

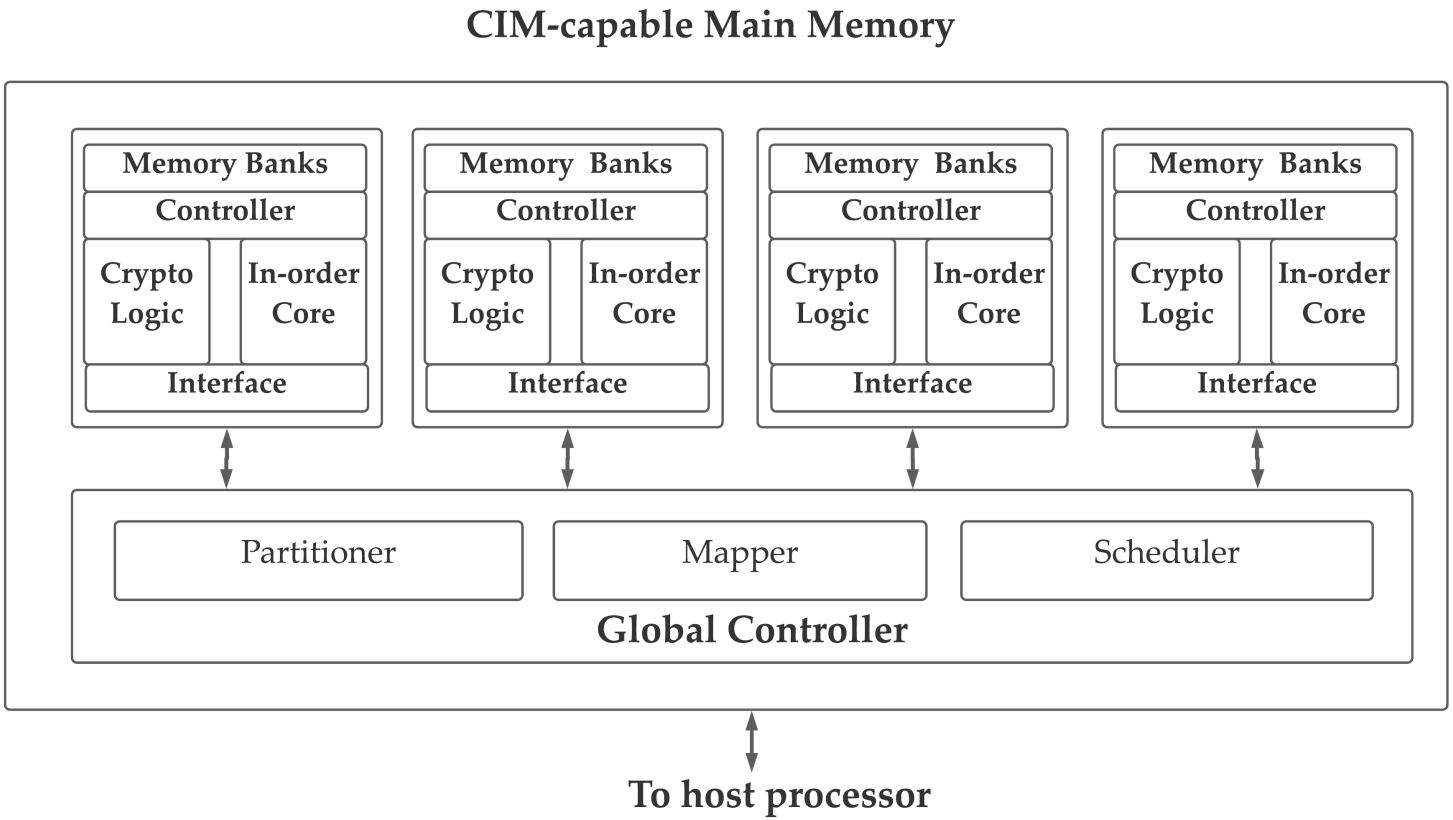


Figure: System architecture for CIM-based memory system for graph acceleration

Proposed Solution

Key Features

- Redesigned hybrid memory cube (HMC) for the main memory
- Logic core in each vault consists of
  - A simple in-order processor for essential data operation
  - Cryptographic logic accelerator for key-storage or NTT-acceleration
- Memory interfaces connected to a crossbar and provide communication capabilities between the cores
- Global controller to support graph partitioning, mapping, and operation scheduling to multiple vaults

Advantages of the Proposed Solution

1. Hardware Root-of-Trust
2. Support for Homomorphic Computation
3. Secure Enclaves
4. Ease of Integration

Challenges

1. Load Balancing,
2. Concurrency
3. Hardware-software co-design
4. Security

Conclusions

- GDBs are becoming incresingly popular
- CIM based data-processing and cryptography can offer viable solutions for secure graph computation in GDB

Conclusions

- GDBs are becoming incresingly popular
- CIM based data-processing and cryptography can offer viable solutions for secure graph computation in GDB
- NOW is the right time