Memory Processing Units (MPU)

(PhD Defense)

Vijay Thiruvengadam

Committee Members:

Advisor: Karthikeyan Sankaralingam

Mikko Lipasti, Parameswaran Ramanathan, Michael Swift, Jignesh Patel

Executive Summary

- **Problem**: It is unclear if a general architecture can fully exploit Processing-In-Memory (PIM) for high performance and energy efficiency, while supporting wide array of workloads
  - Most existing solutions specialize for particular workload domains
  - Some others incur significant energy overheads to retain generality

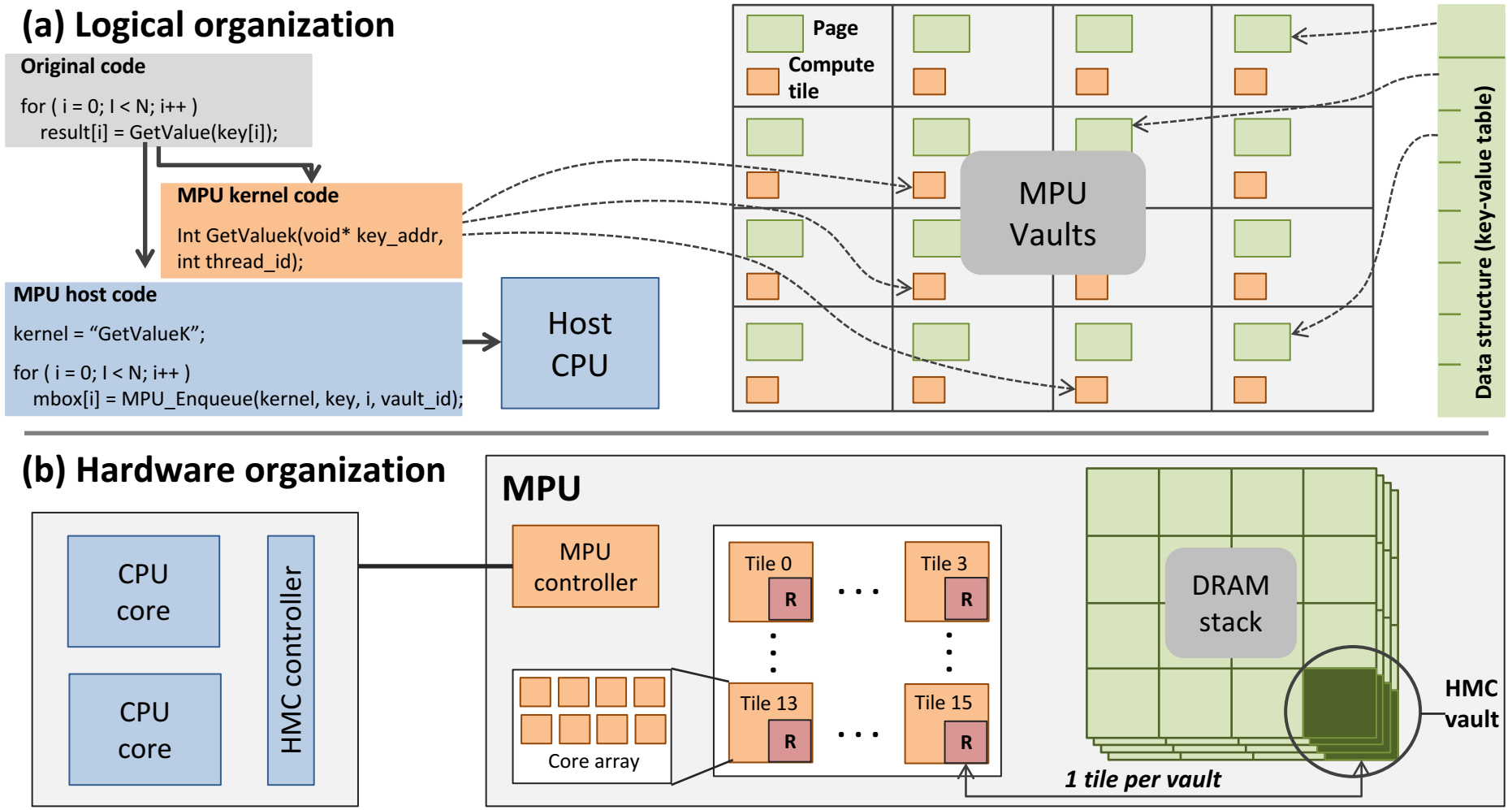
- **Solution**: Memory Processing Units (MPU)
  - More efficient than state-of-art 4-core “Skylake” processor for a wide array of workloads with varying degrees of memory locality
  - More efficient than general PIM architectures targeting low locality workloads
  - Close to efficiency achieved by a domain-specialized PIM architecture, for graph workloads (low locality)

MPU Design Approach

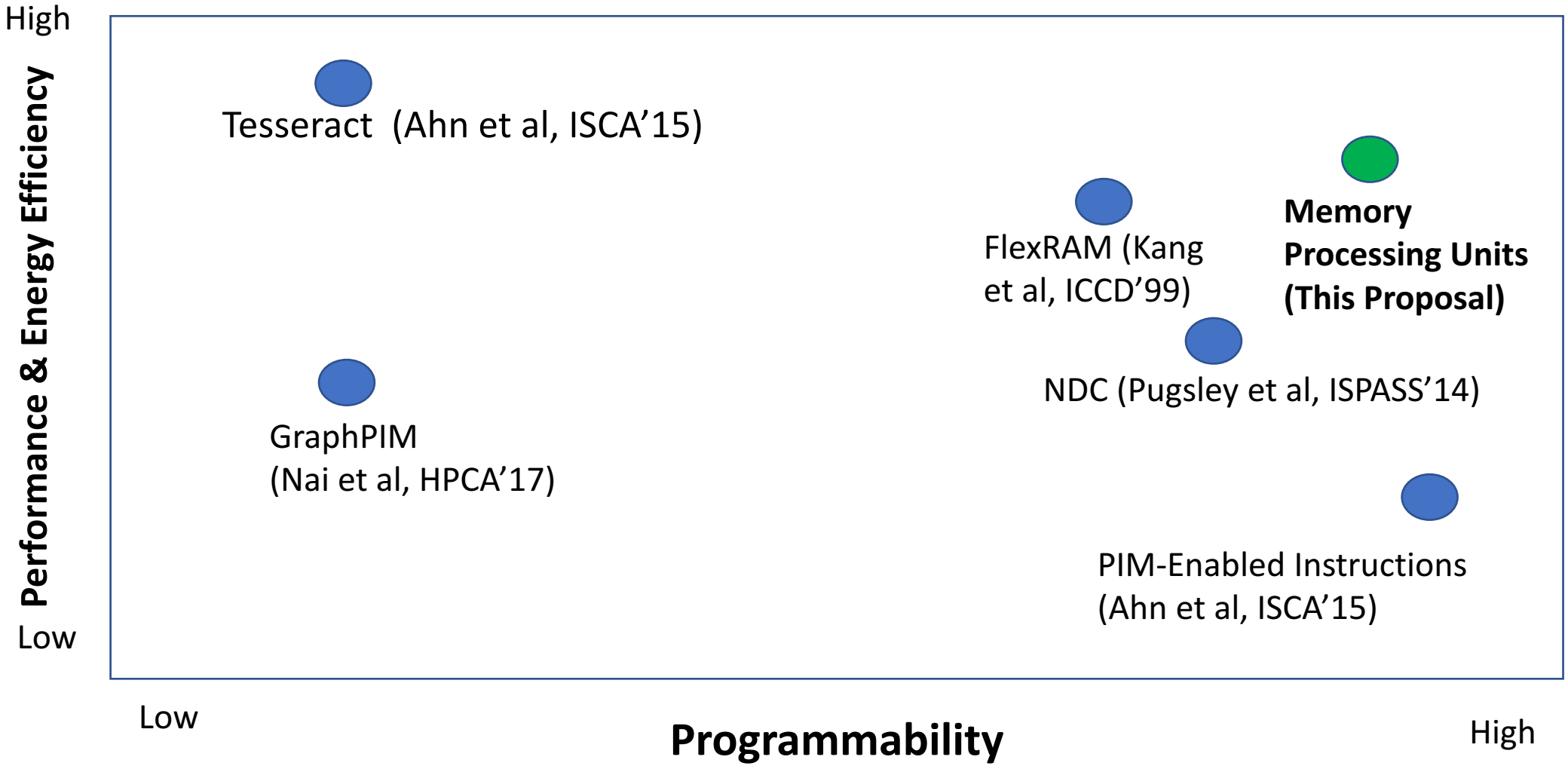
- Adhere to three principles:
  - Performance through massive concurrency to exploit PIM bandwidth
    - Large array of cores placed in memory
  - Energy savings through low-power computation
    - Tiny cores that idle efficiently
  - Flexible Programming Model
    - RPC-like offload mechanism

- Ensure easy integration to commercial OoO processors requiring minimal modifications to the processor hardware structure or the operating system support
  - Stand alone co-processor to the main processor

MPU Overview



Landscape of Solutions in PIM Space targeting low locality workloads



Thesis Statement

Recent works in PIM domain seem to suggest that specialization is necessary to achieve high performance and energy efficiency and a general PIM architecture can only provide modest benefits.

MPU shows it is possible to realize a general PIM architecture that can come close to the efficiency of proposed specialized solutions AND can be significantly more efficient than recent proposals of general PIM architectures.

Contribution of this Thesis

- Memory Processing Units (MPU): A detailed PIM hardware architecture, system architecture and programming model, non-intrusively deployable on today's processors

- Detailed evaluation across a variety of workloads, that
  - Shows that composition of "known" mechanisms, that implement 3 key principles, suffices to achieve general and efficient processing
  - Provides insights into source of performance and energy benefits and scaling trends

Post-Prelim Work

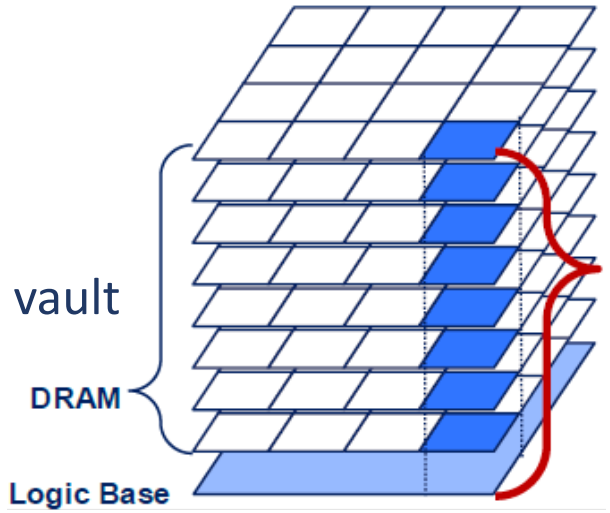
- Had relatively late prelim, with most of the work completed before Prelims

- Post prelims, the following was added to my research and this presentation:
  - Qualitative and quantitative comparison to related work – “Tesseract” and “PIM-enabled Instructions”
  - Baseline for data analysis changed from Westmere to state-of-art Skylake processor
  - 2 text analytics workloads added – dfagroup and htmltok
  - Refinements/optimizations to hardware architecture, system architecture and programming model

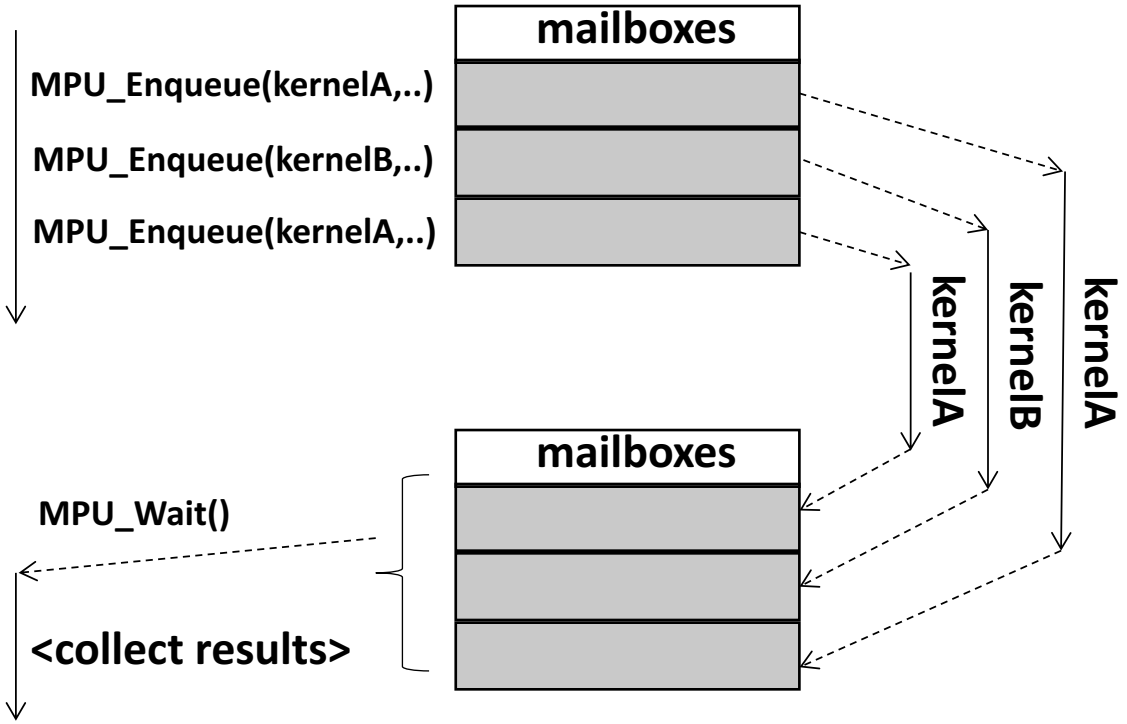
Outline

- **Overview of Programming Model**
- **Overview of Hardware Architecture**
- **Overview of System Design**
- Evaluation Methodology
- Results & Analysis
- Comparison against Related Work

Overview of Programming Model

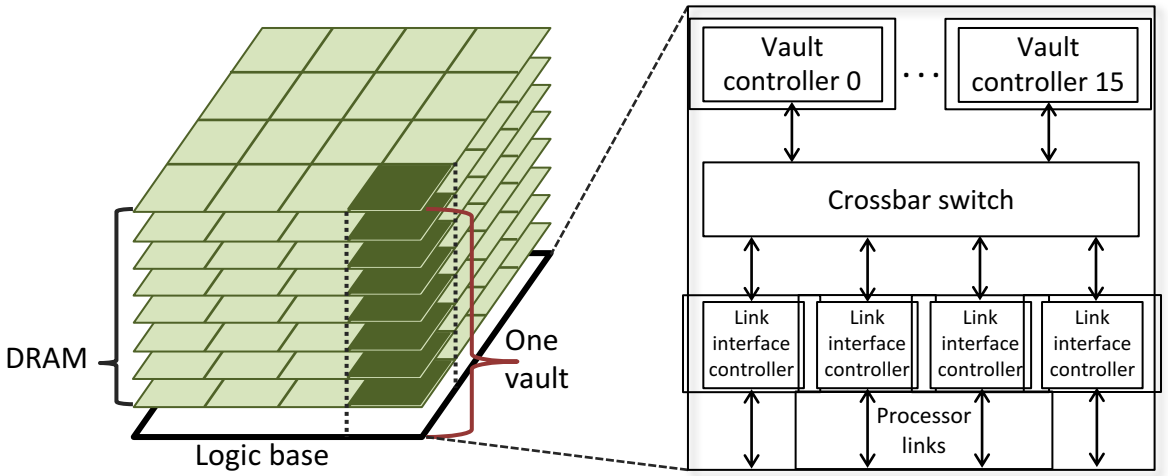


Data Abstraction

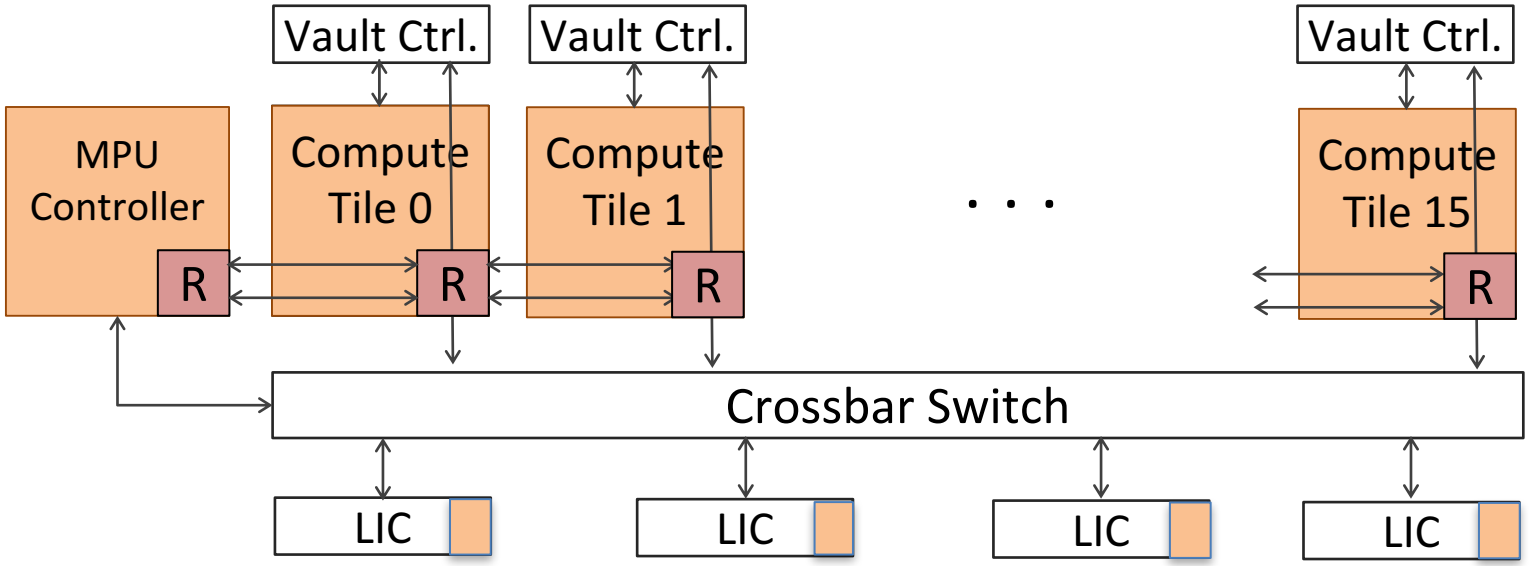


Computation Abstraction

Overview of Hardware Architecture



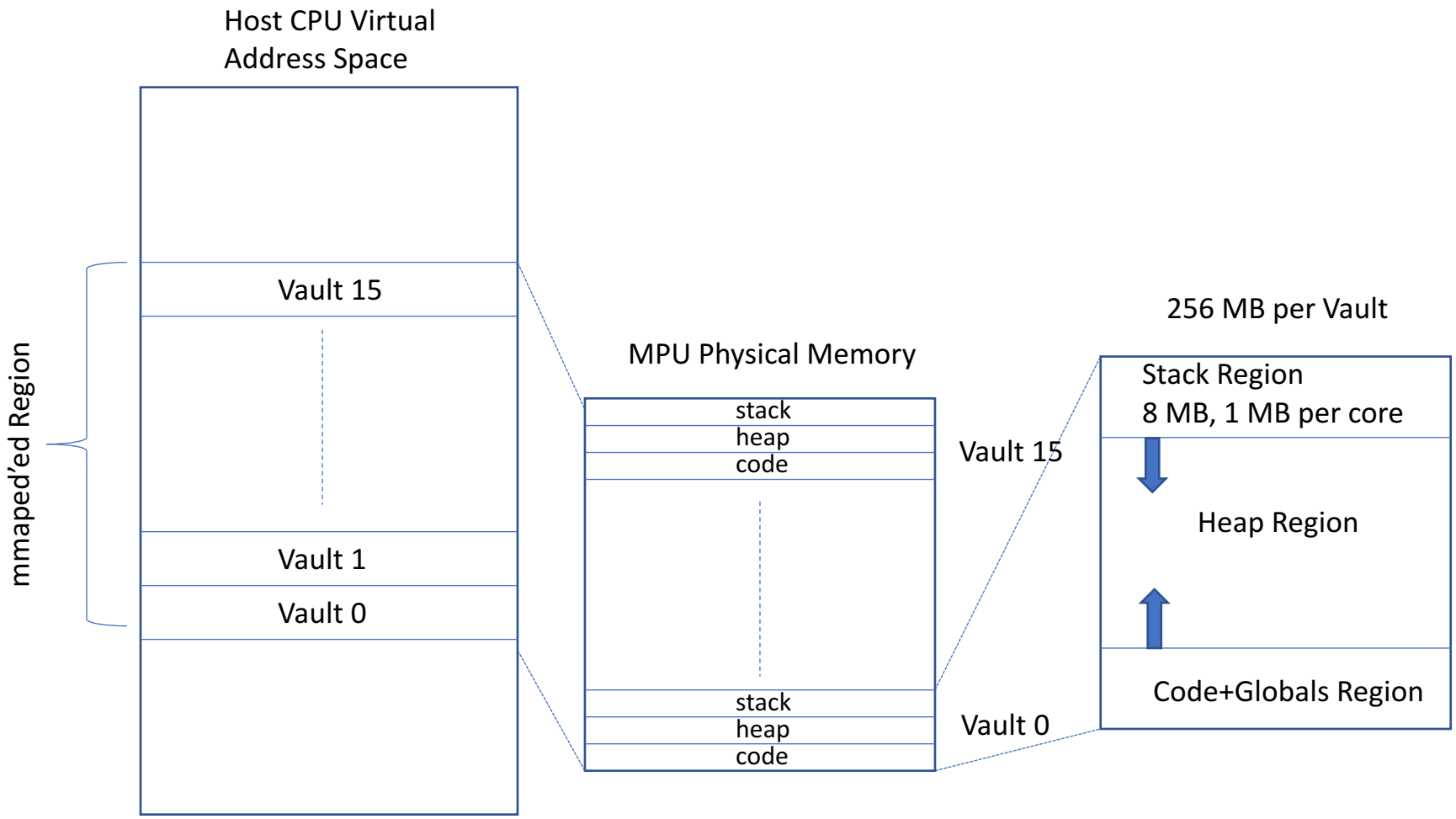
Vanilla HMC



MPU ➔ HMC Augmented with Compute Logic (**shown in orange**)

Overview of System Design

- Consistent pointers
- No TLBs, no page table
- Custom API for memory allocation

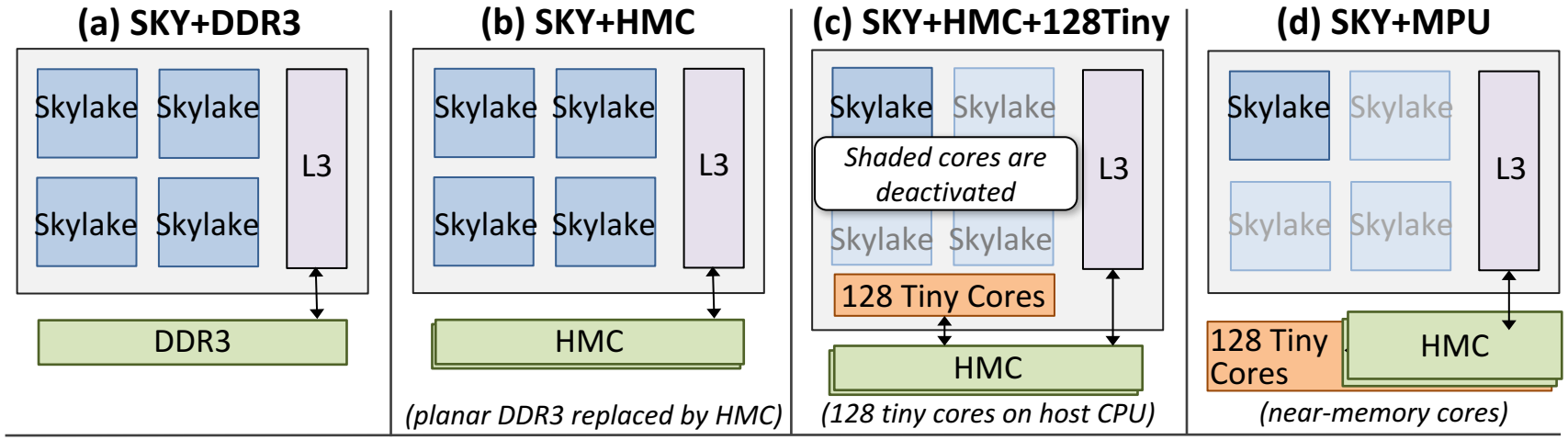


Outline

- Programming Model
- Hardware Architecture
- System Design
- **Evaluation Methodology**
- Results & Analysis
- Comparison against Specialized Architectures

Evaluation Objectives

- What are the dominant factors dictating speedup over a state-of-art multi-core OoO processor?
- What are the dominant sources of power savings?
- Can we get MPU-like benefits with a simpler architecture?



- How scalable is the MPU design?

Choice of Workloads

- Wide array of workloads chosen based on multiple objectives:
  - Commonly used kernels/micro-benchmarks that ease detailed analysis
    - stringmatch, kmeans, histogram, scan, aggregate, hashjoin, buildhashtable
  - **Workloads best suited for MPU**
    - 5 graph analytics workloads (pagerank, shortestpath, averageteenage, vertexcover, conductance)
  - Workloads that allow analyzing impact of load balancing
    - Graph analytics workloads
  - Good mixture of serial and parallel phases to model impact of Amdahl's law
    - 6 queries from database analytics domain (TPCH)
  - Other commercially important workloads that may be a good candidate for MPU
    - Text analytics workloads

- Evaluation uses MPU and pthread (for baseline) programs developed for all 20 workloads

Outline

- Programming Model
- Hardware Architecture
- System Design
- Evaluation Methodology
- **Results & Analysis**
  - What are the dominant factors dictating MPU speedup?
  - What are the dominant sources of power savings?
  - Can we achieve MPU-like benefits with a simpler architecture?
  - How scalable is the MPU design?
- Comparison against Specialized Architectures

Factors Dictating MPU Speedup over Baseline

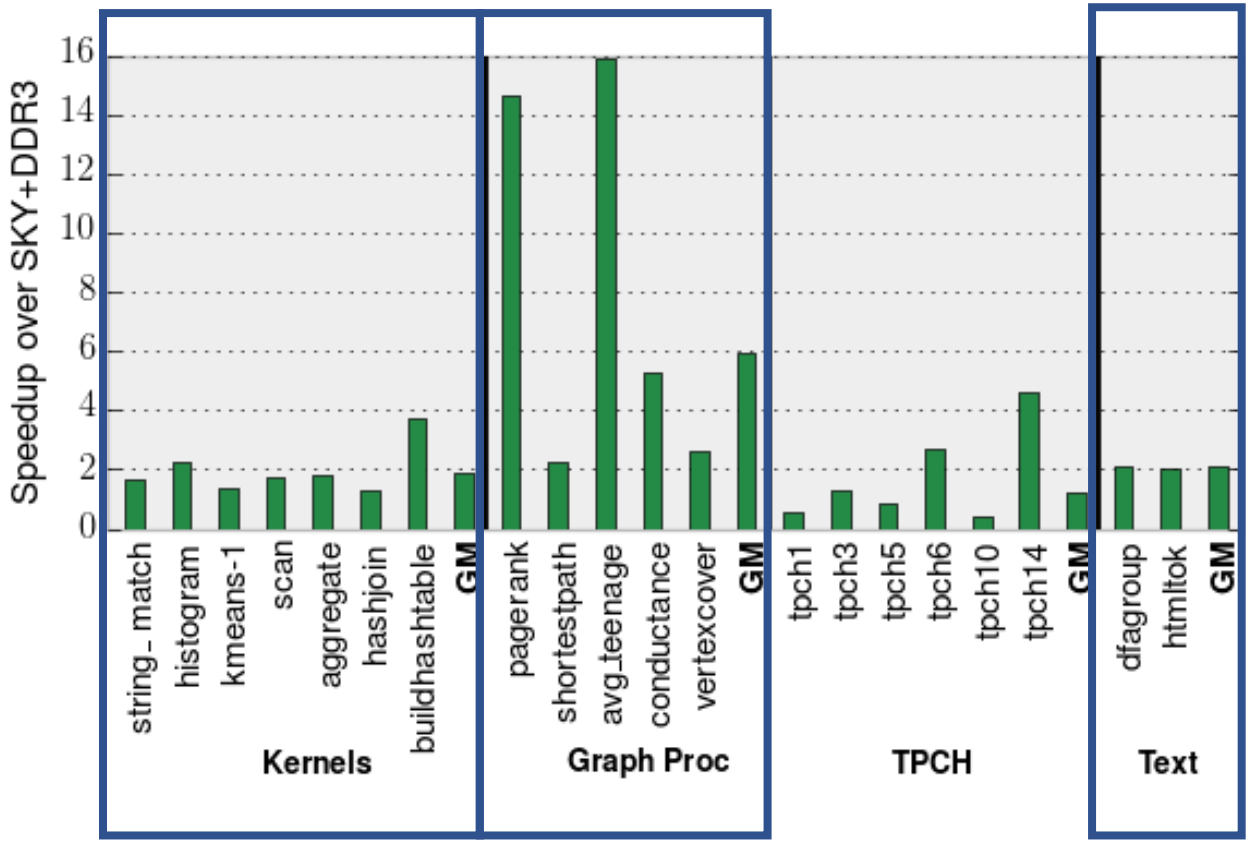
- Compute capability (IPC) of a single MPU core relative to single CPU core
  - $MPU_{speedup} = Frequency_{ratio} \times Concurrency_{ratio} \times IPC_{ratio}$, assuming perfect load balancing across all cores, 100% parallelizable
  - After accounting for workload-independent factors of frequency and core count, to achieve speedup, CPU_IPC must be *less than 4.57x MPU_IPC*
- Load balancing across cores
- Degree of parallelization

Factors Dictating Speedup - IPC_{ratio}

- Kernels, Text Workloads ➔ Well load balanced + ~100% concurrent
- Graph Workloads ➔ Bad load balancing + `100% concurrent
- Database (TPCH) Workloads ➔ Good load balancing + varying degrees of concurrency

- For kernels & text workloads, mean CPU_IPC ~ **2x** mean MPU_IPC
  - Mean CPU IPC = 1.18
  - Mean MPU IPC = 0.6

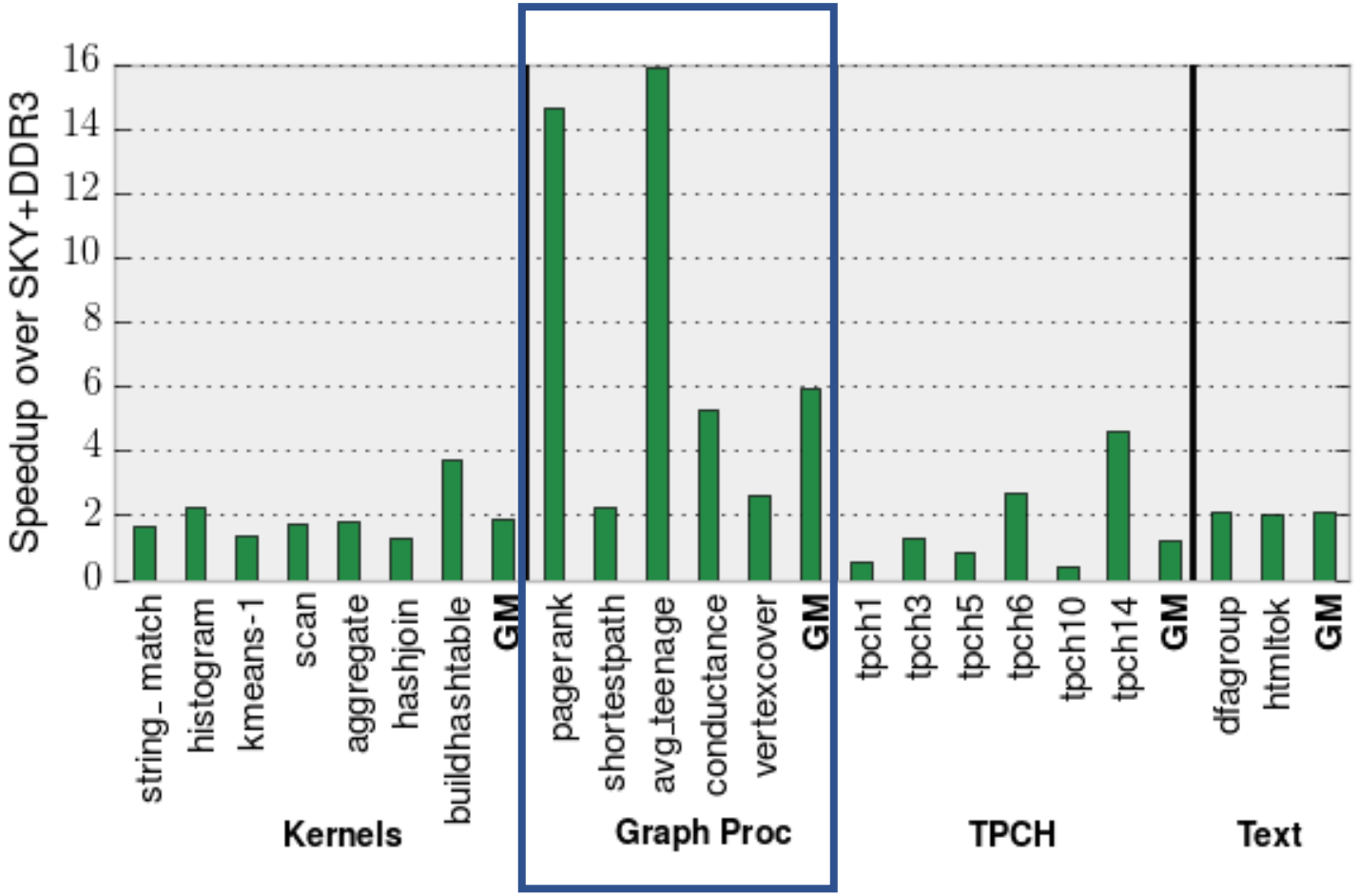
- For graph workloads, mean CPU_IPC **is less than** mean MPU_IPC
  - Mean CPU IPC = 0.13
  - Mean MPU IPC = 0.19



Takeaway: For highly concurrent workloads that we evaluate, OoO+Caching mechanisms in CPU do not help extract enough IPC to outperform MPU

Factors Dictating Speedup – Load Balancing

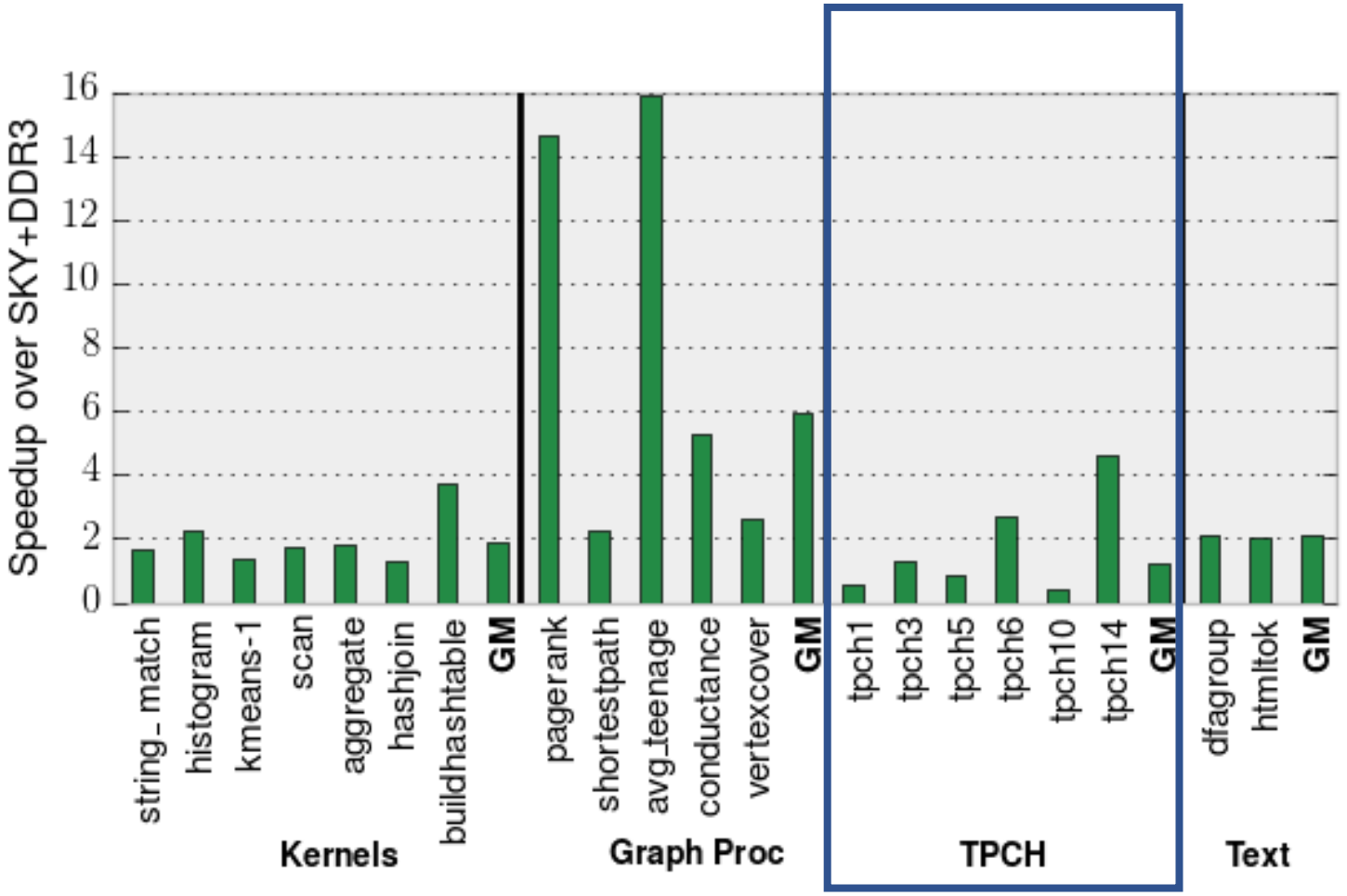
- Perfect load balancing ➔ Instructions equally distributed among all cores
- Slowest core extra load ➔ MPU(~60%) > Baseline(~44%)
- Perfect load balancing ➔ MPU speedup ⬆ up to 40%



Takeaway: Load balancing has non-trivial impact on MPU speedup. The impact is a function of graph structure and partitioning algorithm

Factors Dictating Speedup – Degree of Parallelization

- Degree of Parallelization
  - tpch 5, 6, 14 ➔ Very high (99%)
  - tpch 1 ➔ High (96%)
  - tpch 3, tpch 10 ➔ Low (86%, 73%)



Takeaway: As per Amdahl's law, degree of parallelization has a significant impact on performance and leads to low speedup with MPU

Takeaway from Performance Analysis

Maximum Speedup with MPU over state-of-art OoO processor is achieved with following workload behavior:

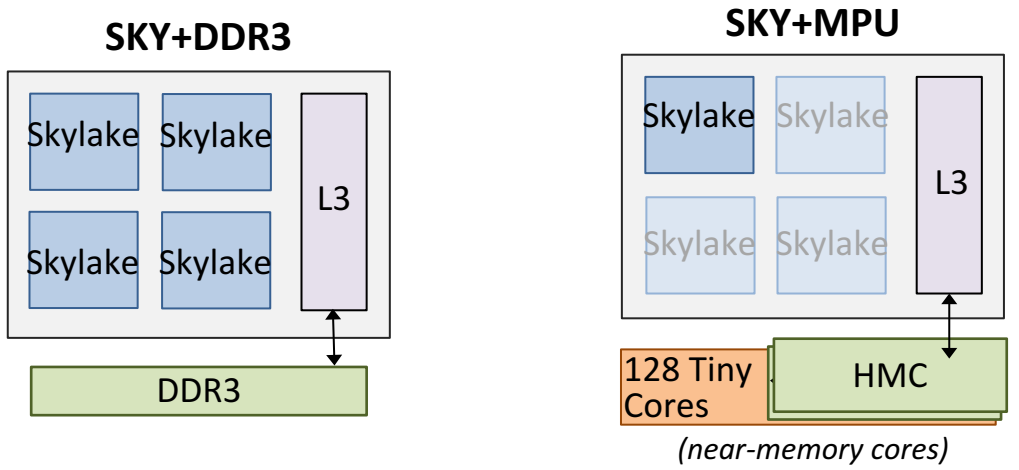
High Thread Level Parallelism
Low instruction level parallelism
Low Cache Locality
Good load balancing

Outline

- Programming Model
- Hardware Architecture
- System Design
- Evaluation Methodology
- **Results & Analysis**
  - What are the dominant factors dictating MPU speedup?
  - What are the dominant sources of power savings?
  - Can we achieve MPU-like benefits with a simpler architecture?
  - How scalable is the MPU design?
- Comparison against Specialized Architectures

Sources of Power Savings

- Four sources of power savings with MPU relative to Baseline:
  - All but 1 host SKY core+cache absent, providing static core+SRAM power reduction (2.9x)
  - 3 of 4 HMC links turned off as only 1 core active, providing static memory power reduction (1.7x)
  - Lower memory access energy per bit (40%) since accesses originate from HMC logic die
  - MPU cores run at much lower power than baseline out-of-order cores (6mW vs ~3W)



Power Savings Analysis

- S*tatic core+SRAM power* accounts for 50-80% baseline power
  - Power savings with MPU ➔ 2.9x

- Static DRAM power accounts for ~10% baseline power
  - Power savings with MPU ➔ 1.7x

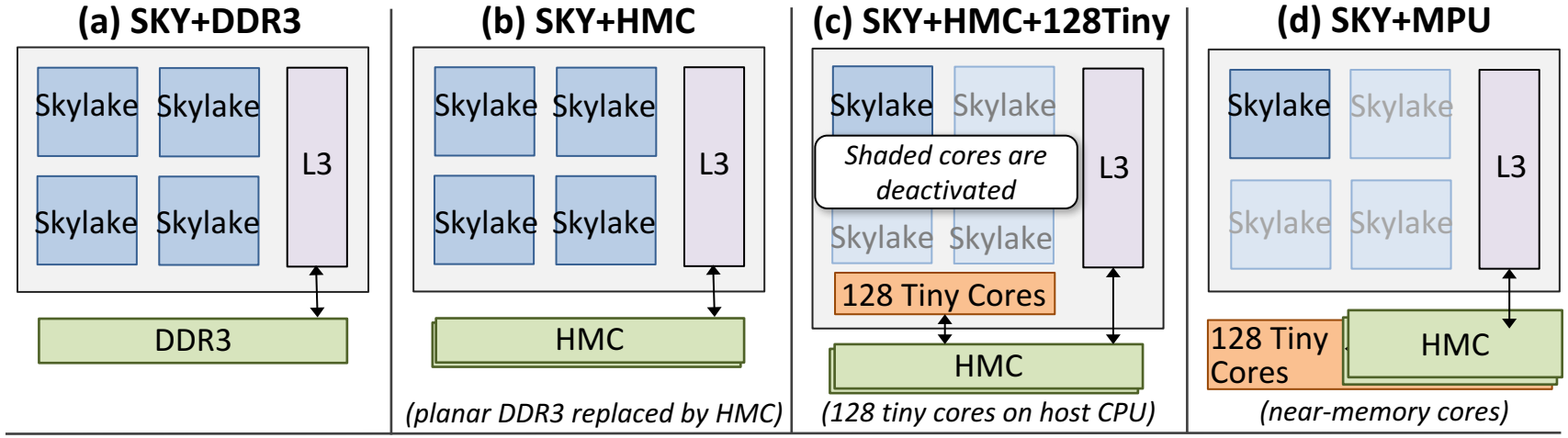
- For highly memory intensive and low locality behavior (graph analytics), dynamic DRAM power accounts for 25-40% baseline power
  - Power savings with MPU ➔ 3%-7%

Takeaway: Most power savings come from the absence of 3 out-of-order cores+SRAM compared to the baseline

Outline

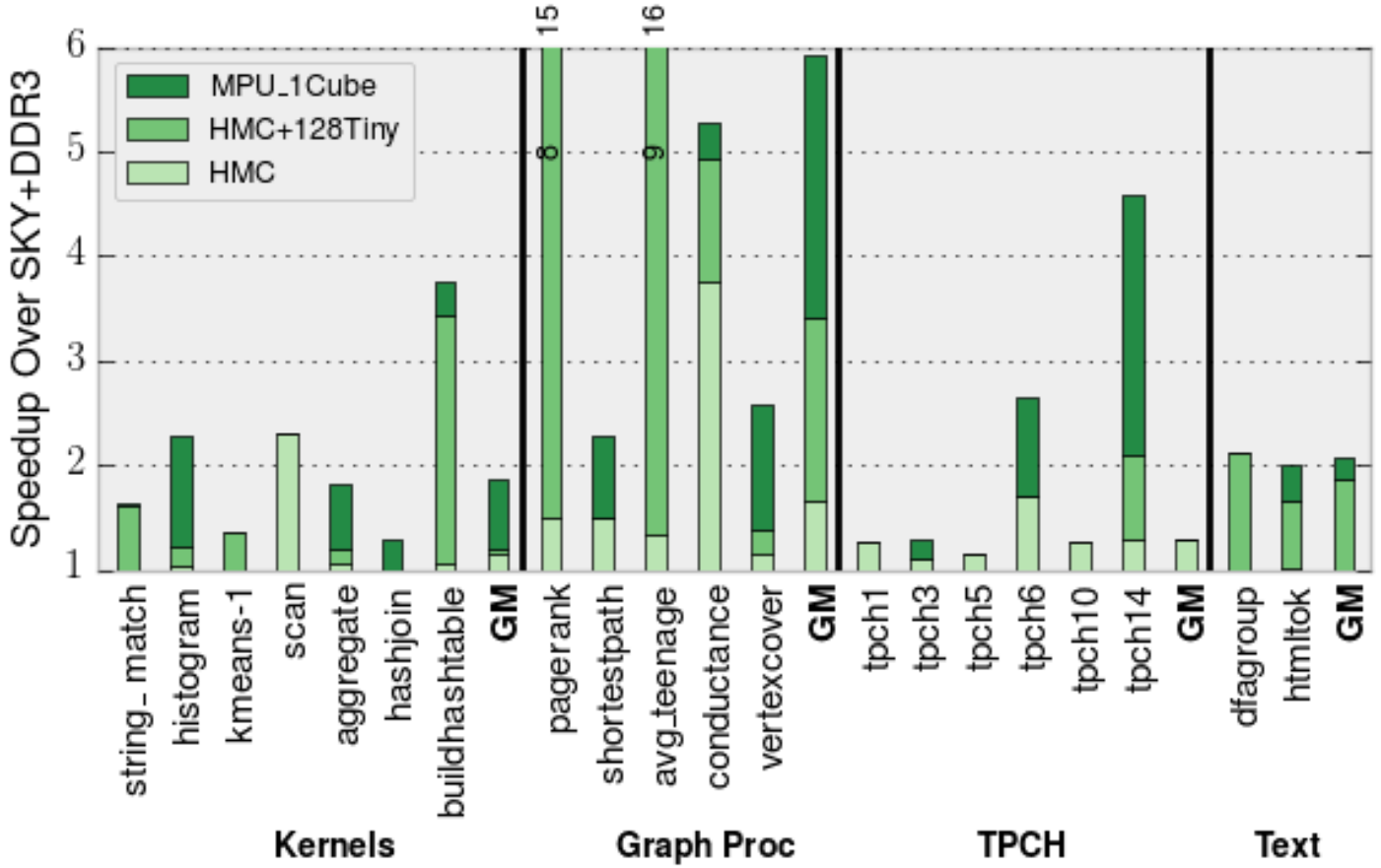
- Programming Model
- Hardware Architecture
- System Design
- Evaluation Methodology
- **Results & Analysis**
  - What are the dominant factors dictating MPU speedup?
  - What are the dominant sources of power savings?
  - Can we achieve MPU-like benefits with a simpler architecture?
  - How scalable is the MPU design?
- Comparison against Specialized Architectures

Can we achieve MPU-like benefits with a simpler architecture?



- **SKY+MPU Design (Proposed Design)** – Provides high concurrency, highest bandwidth availability, lowest memory access energy, no change required to host chip
- **SKY+HMC+128Tiny** – Provides equivalent concurrency, lower bandwidth and higher memory access energy than proposed design
  - With good cache locality, and thus lower bandwidth demand, this can be as almost as good as SKY+MPU, both from performance and energy standpoint
- **SKY+HMC** – Baseline design with HMC. Provides higher bandwidth availability than baseline
  - With low concurrency and/or high ILP in the workload, this can be as good (or better) as SKY+MPU from a performance standpoint

Can we achieve MPU-like benefits with a simpler architecture?

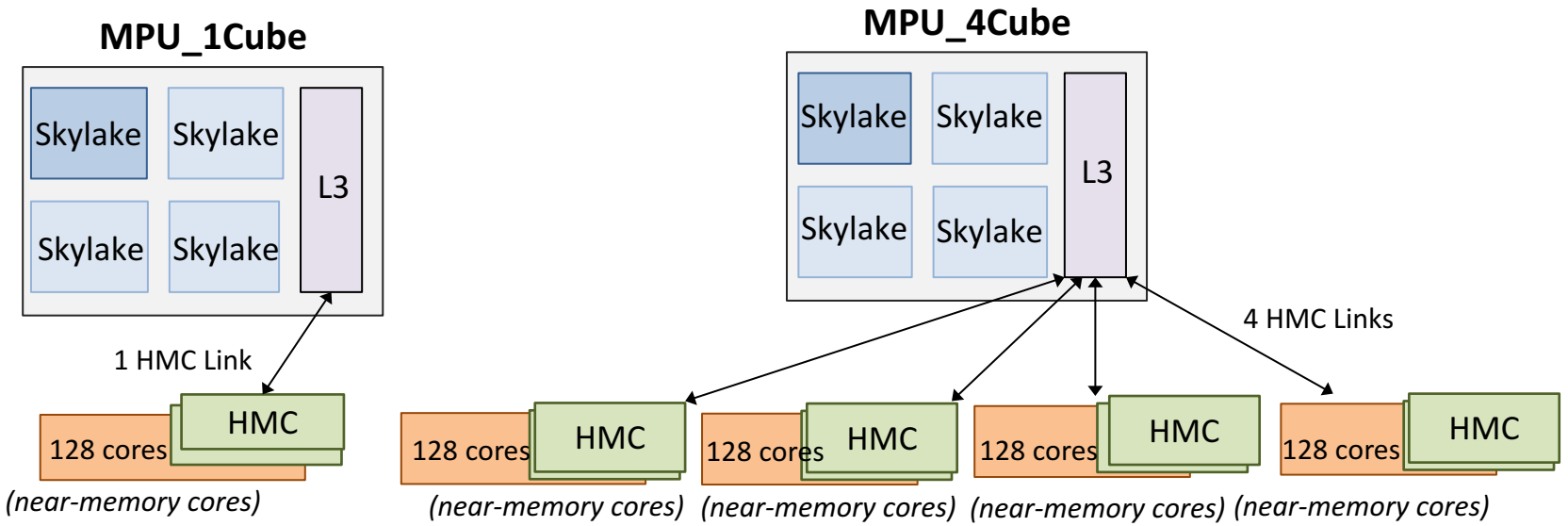


Takeaway: For most workloads (11 of 20), MPU proves to be the best design by significant margin. For remaining 9 workloads, either HMC+128Tiny or HMC designs proves sufficient

Outline

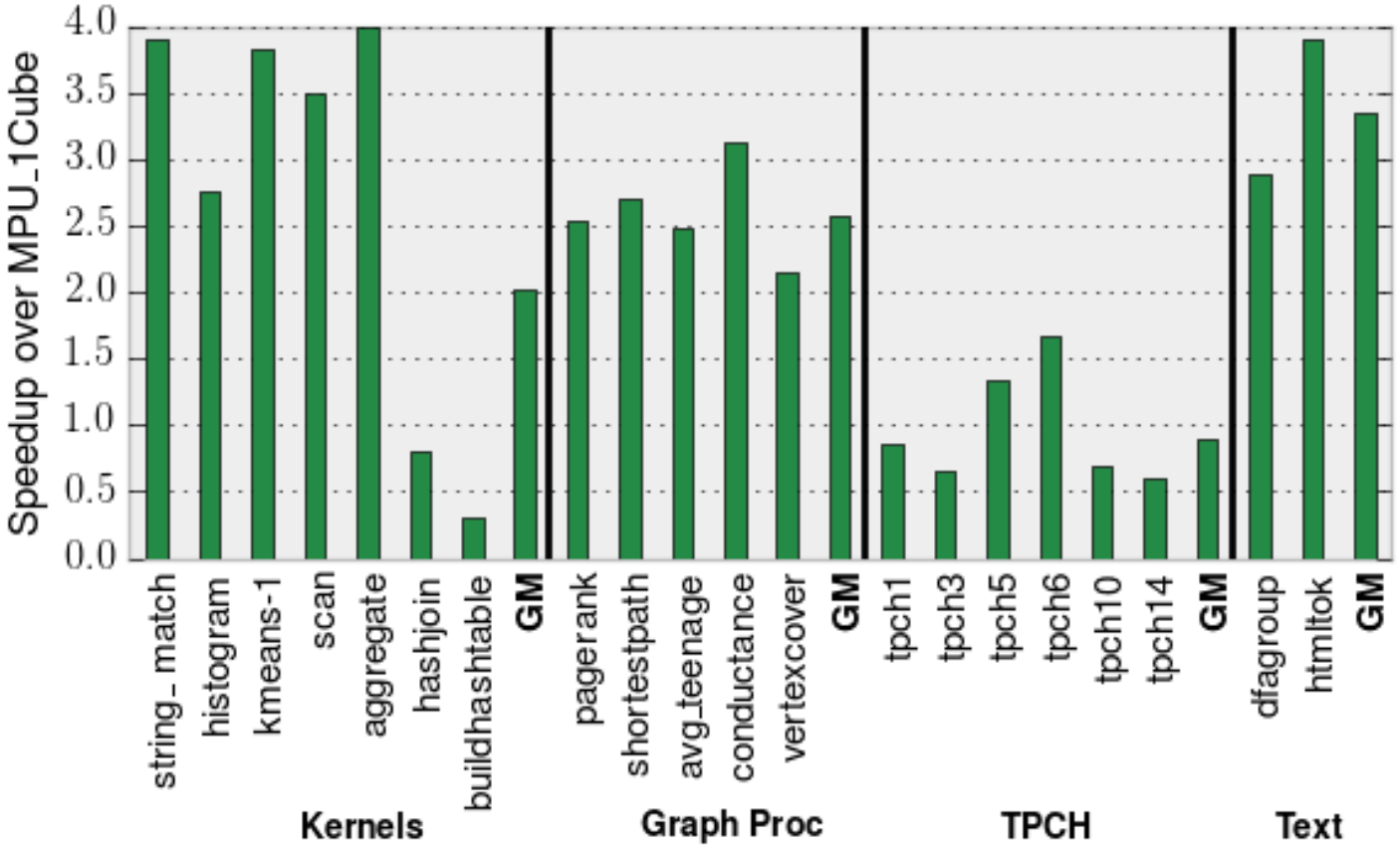
- Programming Model
- Hardware Architecture
- System Design
- Evaluation Methodology
- **Results & Analysis**
  - What are the factors dictating MPU speedup over baseline?
  - What are the dominant sources of power savings?
  - Can we achieve MPU-like benefits with a simpler architecture?
  - How scalable is the MPU design?
- Comparison against Specialized Architectures

Scalability of MPU Benefits - multi-MPU System



- 4 MPU system compared against single MPU
- 1 SERDES channel active per MPU

Scalability of MPU Benefits - multi-MPU System

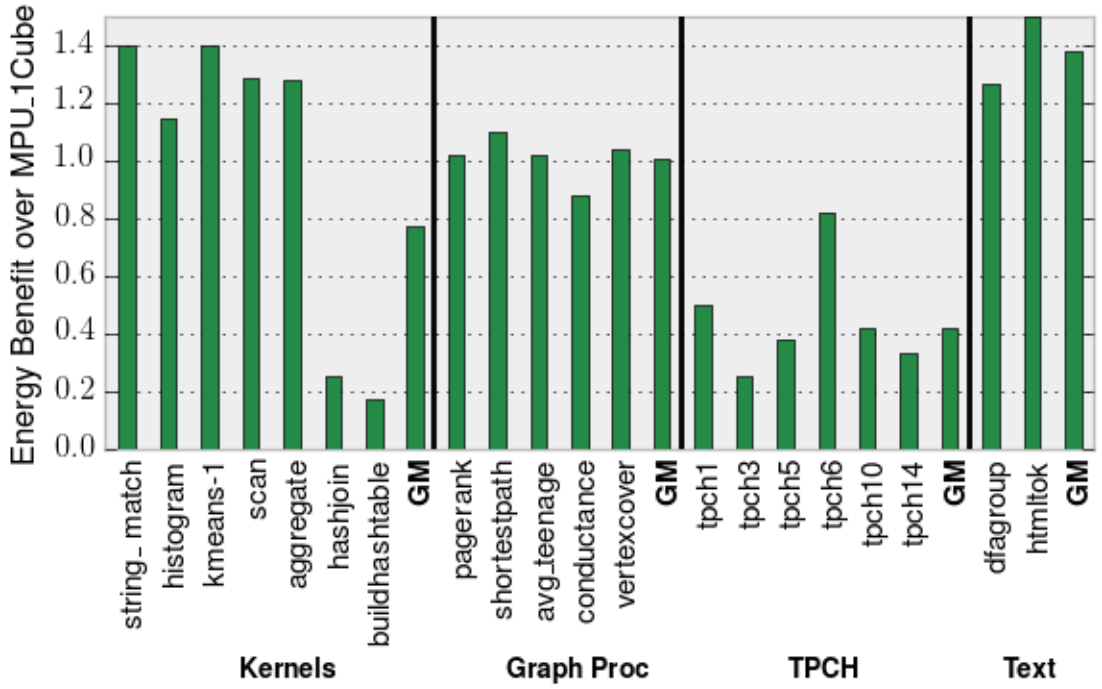


Takeaway: Increase in BW (4x) significantly improves IPC (2-3x) for many workloads. Not reflected in overall speedup due to serialization, load balancing, other algorithmic effects

Scalability of MPU Benefits – multi-MPU System

Maximum energy savings only 1.4x, despite higher speedup

- Primarily due to increase in all power components, with dynamic DRAM power contributing the most

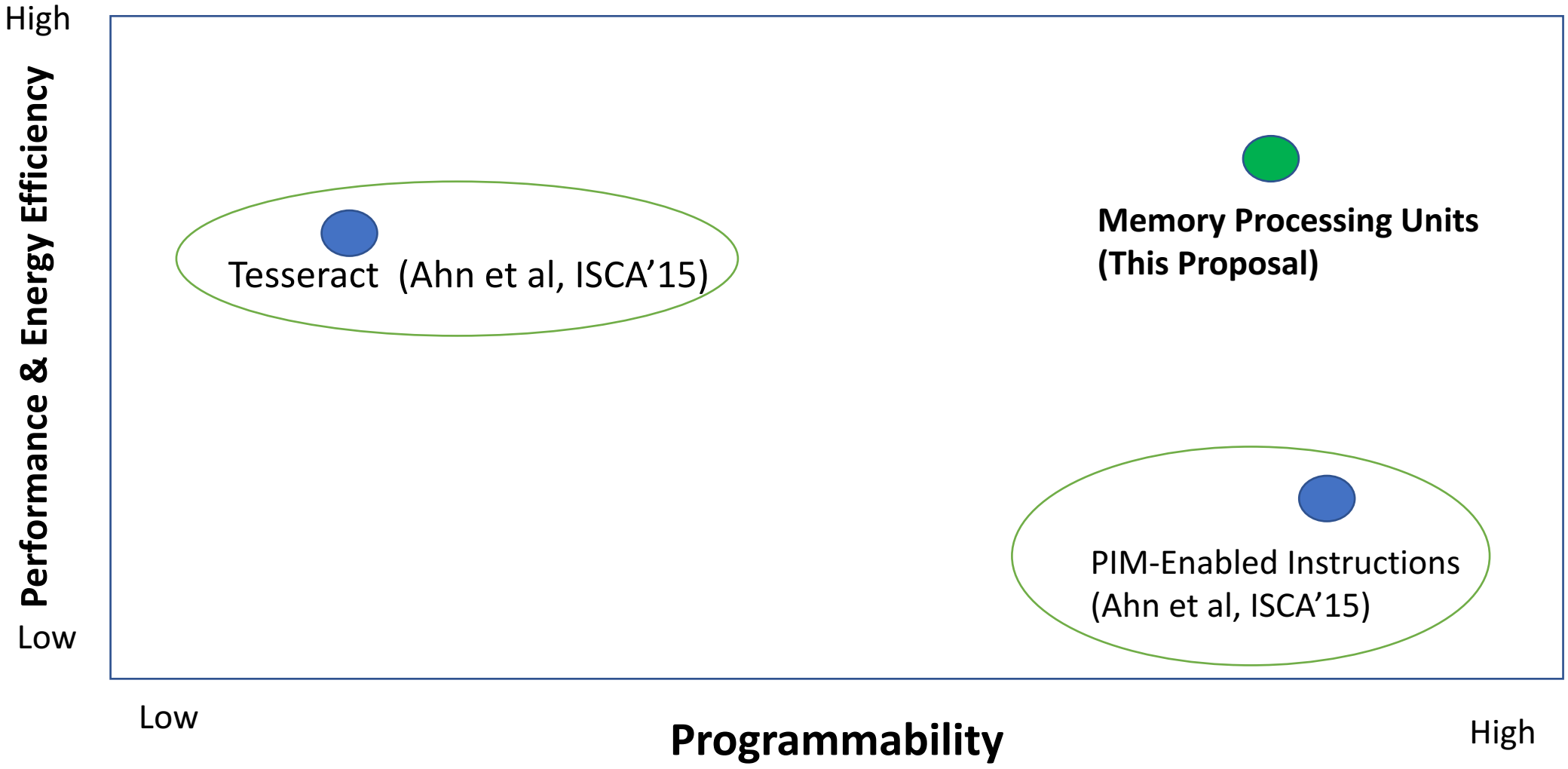


Takeaway: On scaling MPU system, energy savings does not keep up with speedup due to large power overhead of more simultaneous DRAM accesses

Outline

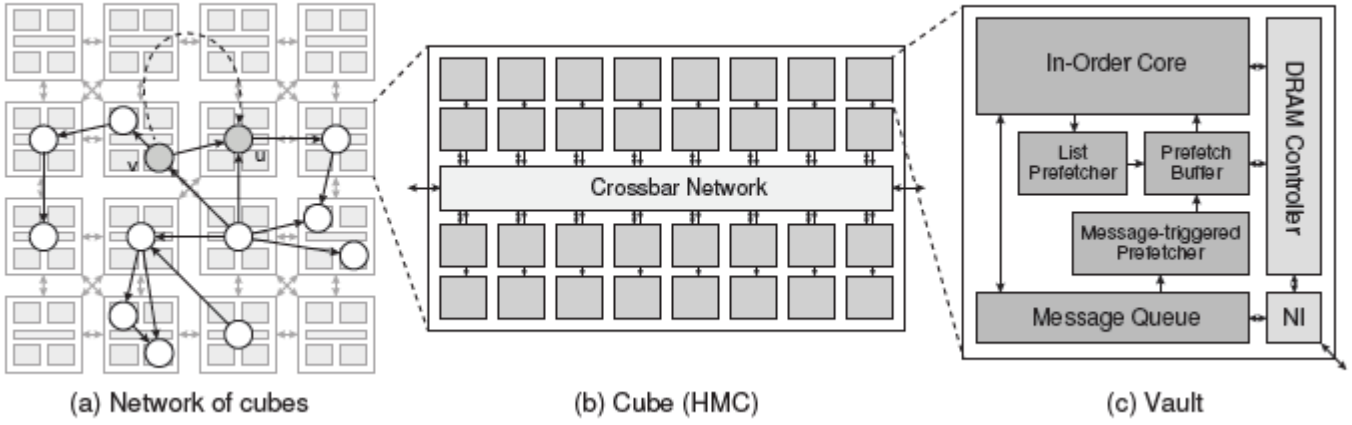
- Programming Model
- Hardware Architecture
- System Design
- Evaluation Methodology
- Results & Analysis
  - What are the factors dictating MPU speedup over baseline?
  - Can we achieve similar benefits with a simpler design?
  - How much energy reduction does MPU achieve? What are the sources?
  - How does MPU speedup and energy reduction scale with a large MPU system?
- **Comparison against other PIM Architectures**

Landscape of Solutions in PIM Space targeting low locality workloads



Tesseract (ISCA'15)

- "Tesseract" by Ahn et al features a specialized hardware architecture and specialized programming API for graph processing
  - 1 core per vault
  - Per-vault List Prefetcher ➔ to retrieve neighboring vertices
  - Per-vault Message Prefetcher ➔to retrieve vertex data upon inter-vault computation transfer
  - Requires message passing API for out-of-vault computations

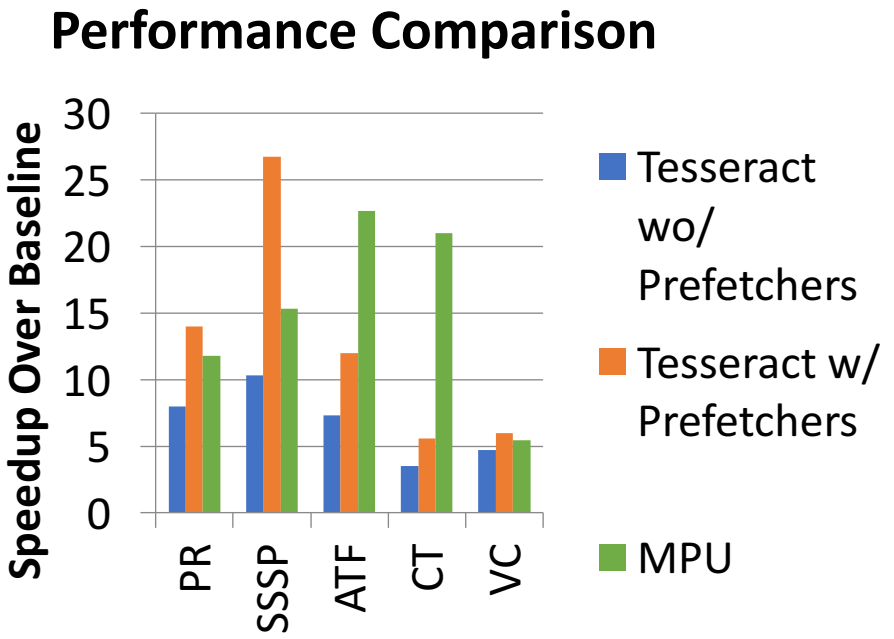


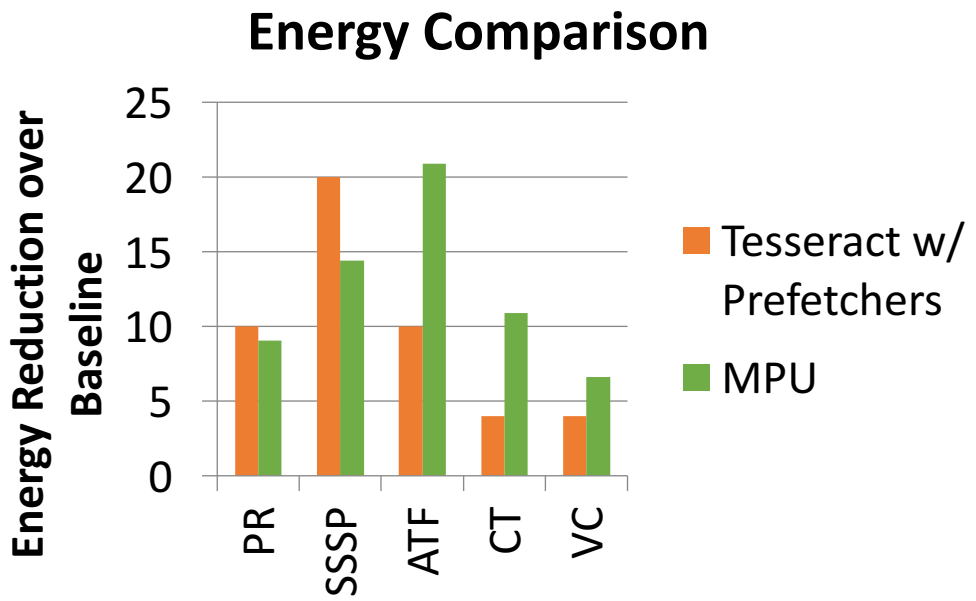
Tesseract Hardware Architecture

```
1  ...
2  count = 0;
3  do {
4    ...
5    list_for (v: graph.vertices) {
6      value = 0.85 * v.pagerank / v.out_degree;
7      list_for (w: v.successors) {
8        arg = (w, value);
9        put(w.id, function(w, value) {
10         w.next_pagerank += value;
11       }, &arg, sizeof(arg), &w.next_pagerank);
12     }
13   }
14   barrier();
15   ...
16 } while (diff > e && ++count < max_iteration);
```

PageRank Computation in Tesseract

MPU versus Tesseract (ISCA'15)





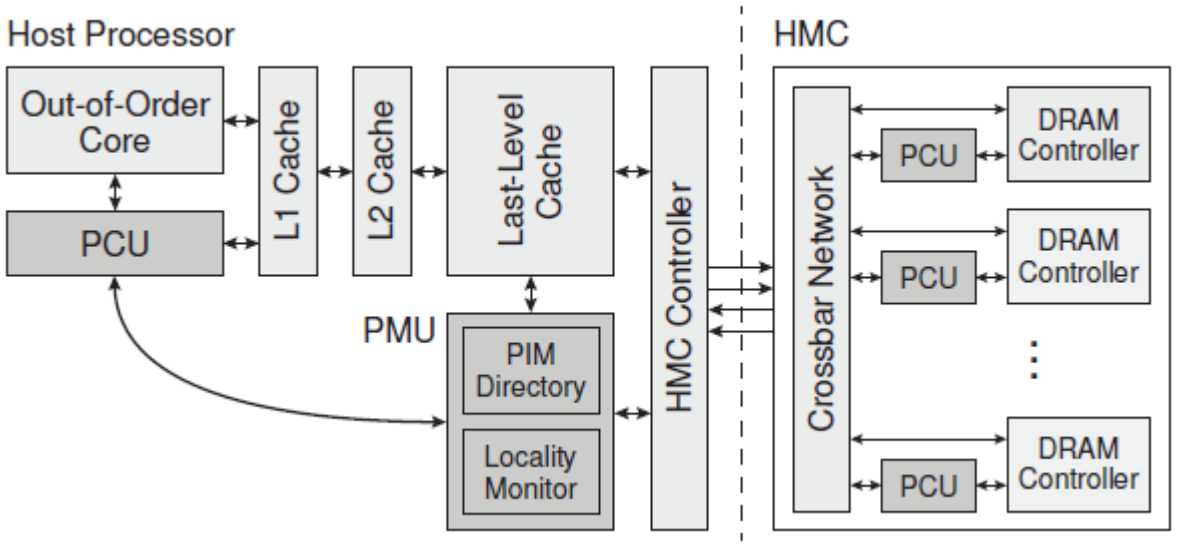
*MPU provisioned with same capacity and vault count as Tesseract

Baseline: 32 4GHz 4-issue OoO + 128GB HMC

Takeaway: MPU comes close in performance and energy efficiency to a specialized architecture for graph analytics

PIM-Enabled Instructions (ISCA'15)

- "PIM-Enabled Instructions" by Ahn et al features a general architecture that requires minimal modifications to programming API, operating system, coherence support, etc
  - 1 PCU/ALU per vault
  - Offloads **atomic** basic-blocks to PIM ALU units
  - Requires modifications to host HW to identify low locality memory accesses, so it could only offload those blocks of work that are likely to miss the cache hierarchy

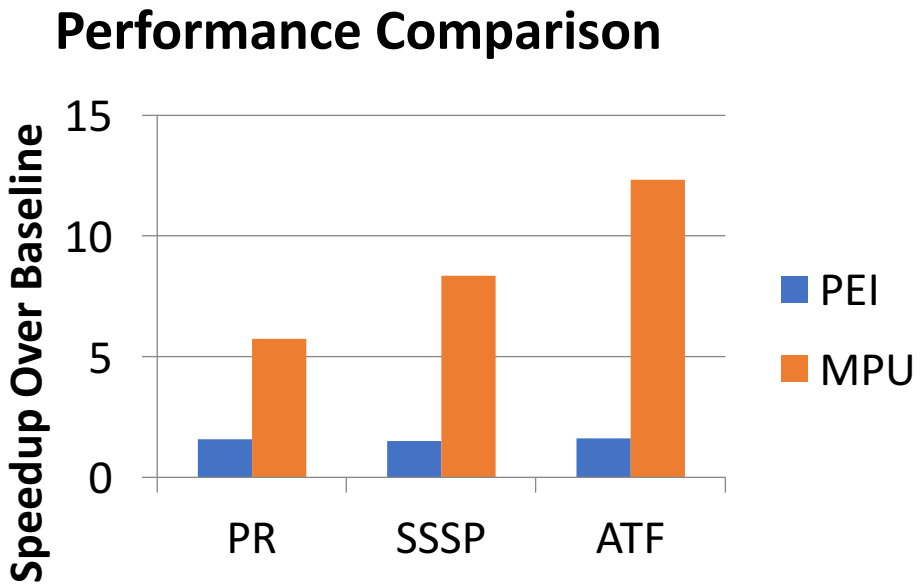


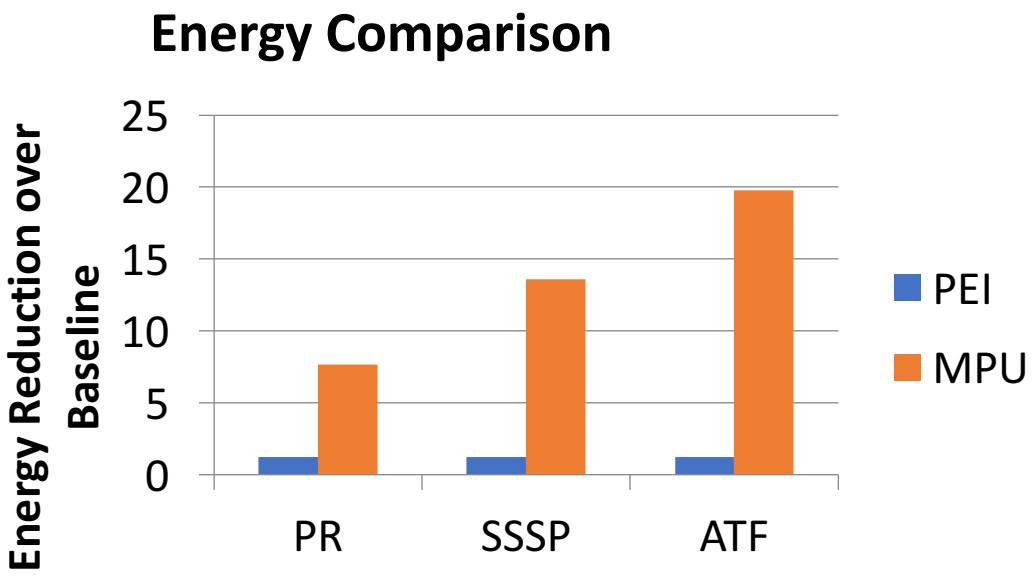
PEI Hardware Architecture

```
1  parallel_for (v: graph.vertices) {
2    v.pagerank = 1.0 / graph.num_vertices;
3    v.next_pagerank = 0.15 / graph.num_vertices;
4  }
5  count = 0;
6  do {
7    parallel_for (v: graph.vertices) {
8      delta = 0.85 * v.pagerank / v.out_degree;
9      for (w: v.successors) {
10       atomic w.next_pagerank += delta;
11     }
12   }
13   diff = 0.0;
14   parallel_for (v: graph.vertices) {
15     atomic diff += abs(v.next_pagerank - v.pagerank);
16     v.pagerank = v.next_pagerank;
17     v.next_pagerank = 0.15 / graph.num_vertices;
18   }
19 } while (++count < max_iteration && diff > e);
```

PageRank Computation in PEI

MPU versus PIM-Enabled Instructions (ISCA'15)





*MPU provisioned with same capacity and vault count as PEI

Baseline: 16 4GHz 4-issue OoO + 32GB HMC

Takeaway: MPU significantly outperforms and reduces energy compared to a recently proposed general PIM architecture

Comparison Against Other PIM Architectures

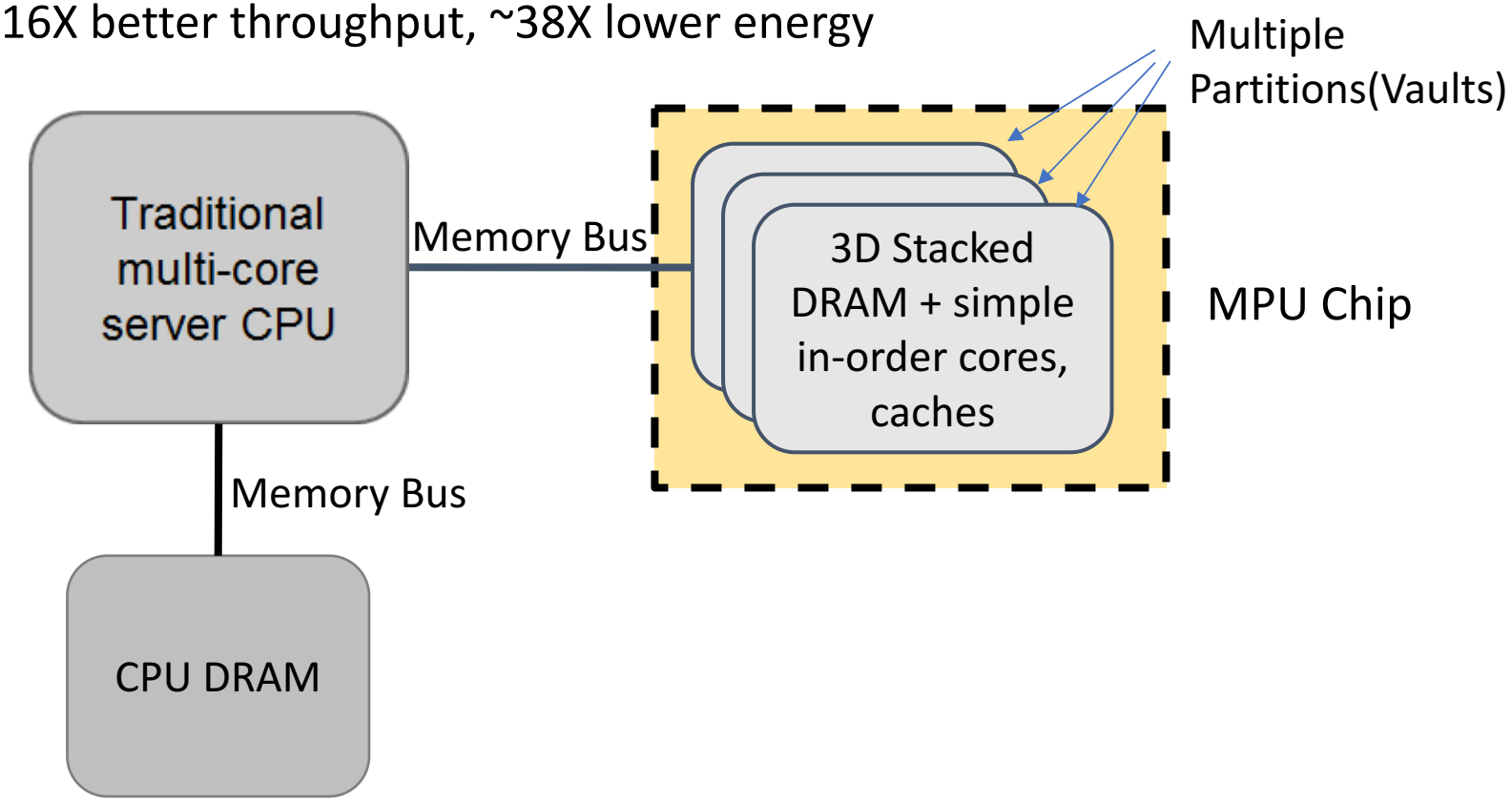
Takeaway: MPU shows it is possible to realize a general PIM architecture that can come close to the efficiency of proposed specialized solutions AND can be significantly more efficient than recent proposals of general PIM architectures.

Thank You – Questions?

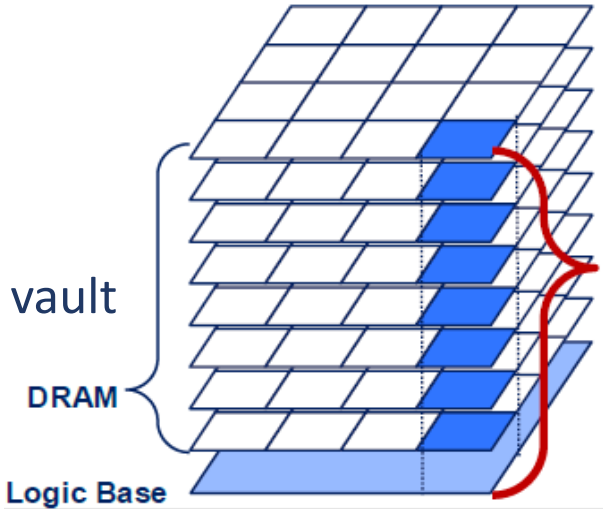
BACKUP SLIDES

MPU: New “Offload” Chip

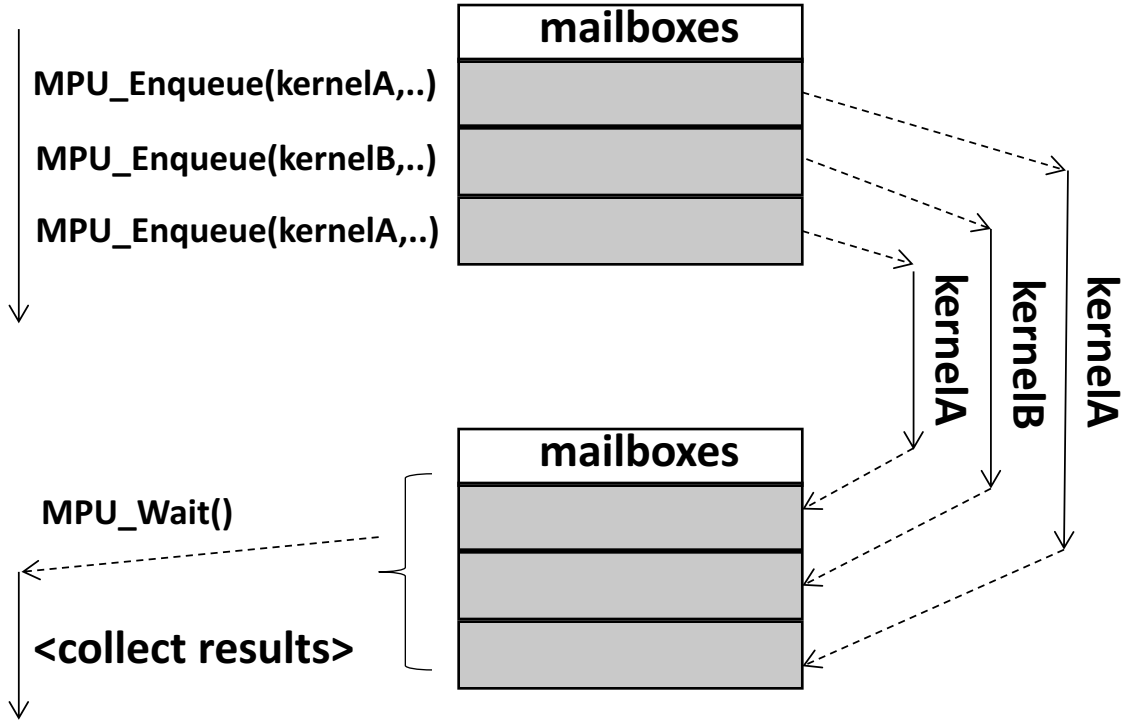
- Co-processor
- 3D Memory (HMC from Micron) + Simple Cores on logic base
- Enables work offload from CPU
- Up to ~16X better throughput, ~38X lower energy



Overview of Programming Model



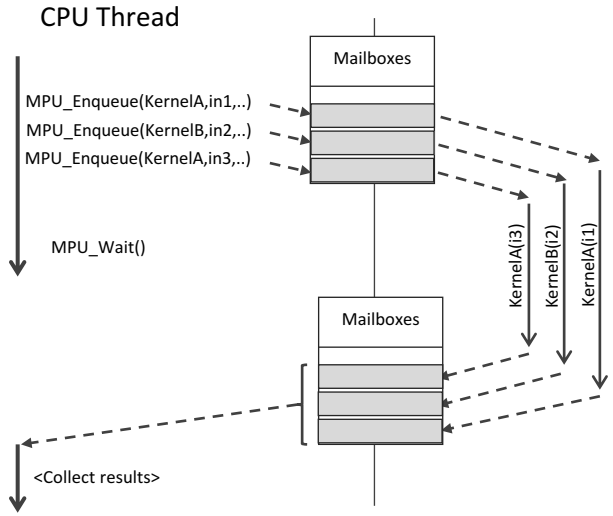
Data Abstraction



Computation Abstraction

- CUDA-like offload model, with some differences:
- CUDA focuses on organizing computation with thread hierarchies, directly supporting SIMD computation
- MPU model focuses on data layout and irregular code, allowing non-SIMD independent computations

Computation Abstraction



(a) Execution flow for MPU applications

host.c

```
void KVApplication(Input in[N]) {
  MPU_Init("mpu_kernels.o")
  MPUMailbox mbox[1024];
  kernel1 = "GetValueK";
  kernel2 = "SetValueK";

  for ( int c = 0, i = 0; i < N; i++ )
    vid = <assign vault number>
    if ( data[i]->type == GET )
      mbox[c] = MPU_Enqueue(kernel1, in, i, vid);
    else
      mbox[c] = MPU_Enqueue(kernel2, in, i, vid);
    if ( ++c == 1024 ) {
      MPU_Wait();
      CollectResults(mbox);
      c = 0;
    }
  }
}
```

(b) Key-Value store host CPU code

mpu_kernels.c

```
HTValue GetValueK(Input* in, int tid)
{
  Bucket* b = in[tid]->bucket;
  Key k = in[tid]->key;
  for ( unsigned i = 0; i < B_SIZE; i++ )
  {
    if ( b[i].key == k )
      return b[i].value;
  }

  return DEFAULT_VALUE;
}
```

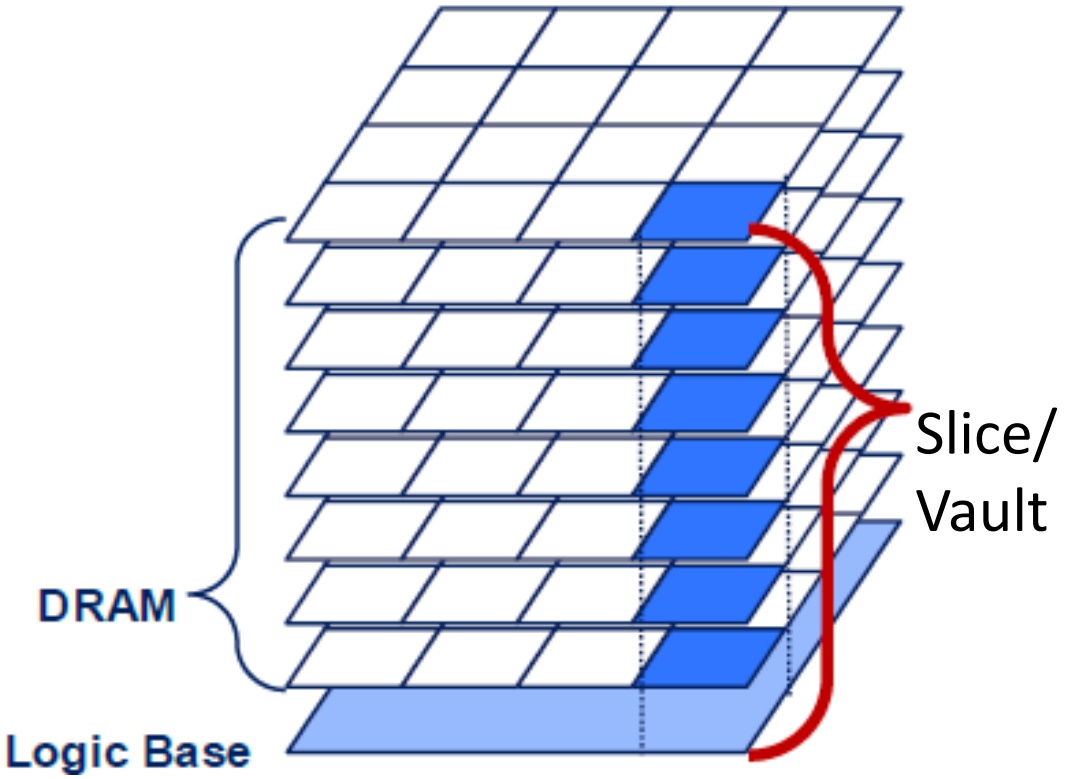
```
int SetValueK(Input* in, int tid)
{
  Bucket* b = in[tid]->bucket;
  Key k = in[tid]->key;
  Value v = in[tid]->value;
  for ( unsigned i = 0; i < B_SIZE; i++ )
  {
    if ( b[i].key == k ) {
      b[i].value = v;
      return SUCCESS;
    }
  }

  return NOT_FOUND;
}
```

(c) Key-Value store application kernels

- Offload using MPU_Enqueue(), an RPC-like memory procedure call
- Wait for return value using MPU_Wait()
- Read mailboxes to gather results

Data abstraction

- No flat memory view
  - Large **data** sets must be **sharded**
- Each MPC tied to 256MB DRAM slice/vault
  - All data and code “ideally” should be local
  - Out-of-vault accesses possible and allowed, though at a higher cost
  - MPU kernel code operates on virtual addresses

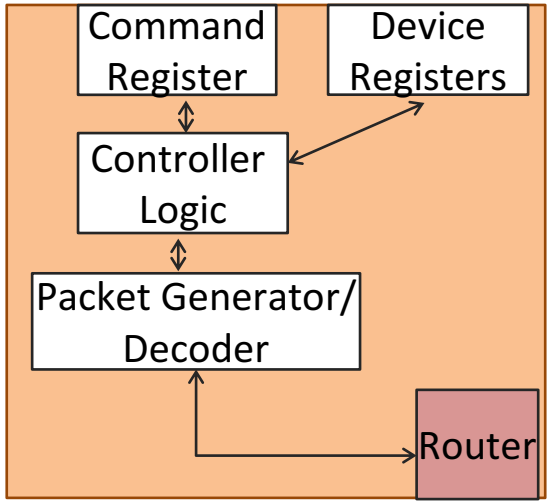


MPU Controller

- Command Register – Identify command type (Init, Enqueue, Wait) + other information

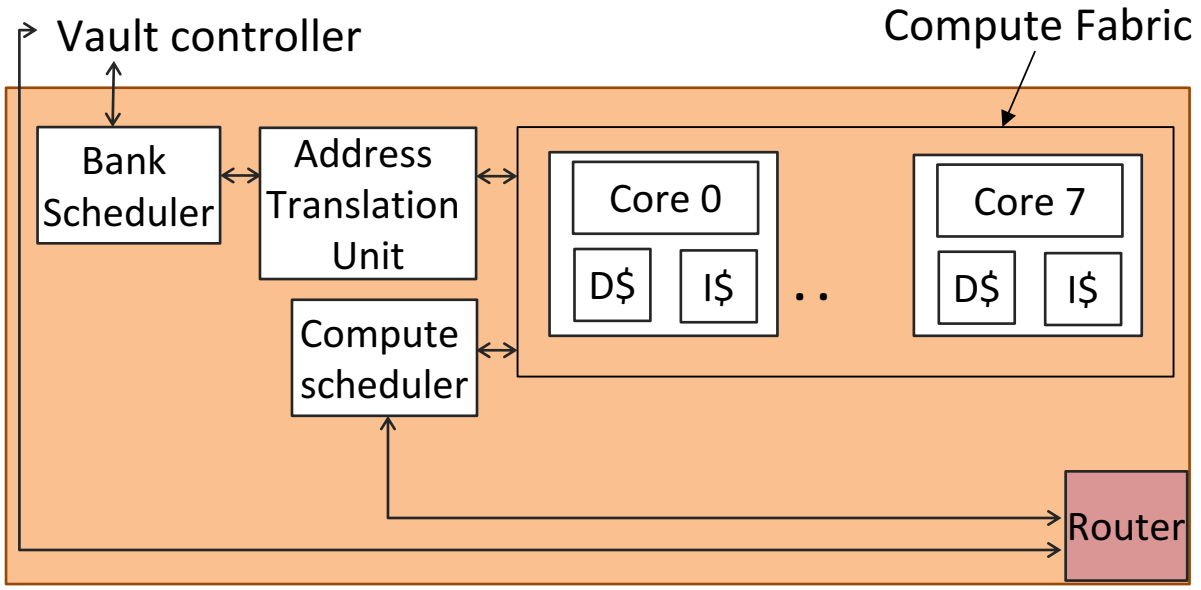
- Device Registers
  - 1024 Mailbox Registers – store incoming MPU_Enqueue data
  - 6 Initialization Registers – store information for device init

- Mailbox Registers
  - 24 bytes each – kernel address, argument address, thread ID, vault ID



MPU Compute Tile (s)

- Compute Fabric
  - 8 Tensilica LX3 cores + private L1 data & instruction cache
  - LX3 core ➔ Single issue, in-order (600x smaller than Intel Ivybridge)
  - No hardware cache coherence support



Ensuring Coherence w/o HW Cache Coherence

- Hardware cache coherence eschewed to reduce design complexity and traffic on interconnect network

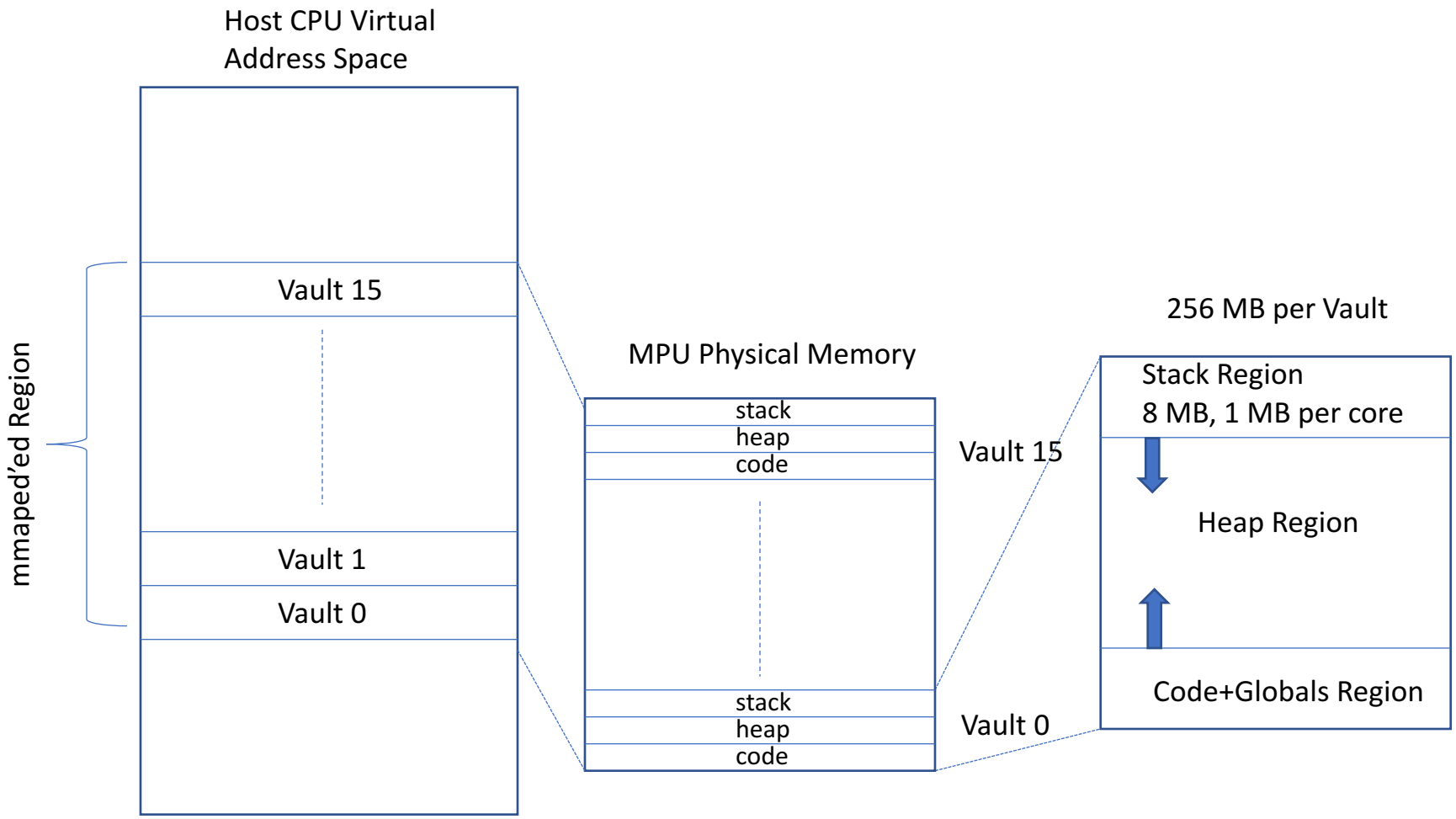
- Coherence ensured with two mechanisms, both of which require some programmer involvement:
  - Global variables always bypass L1 cache. Achieved by placing global variables in specific region (code section) in vault 0 and having each core monitor if a load/store access falls within that region
  - Coherence for read-write shared heap variables ensured by enclosing accesses within critical section. At end of critical section, cache lines made dirty within critical section are written-back and all lines read or written are invalidated.

Guiding Principles of MPU System Design

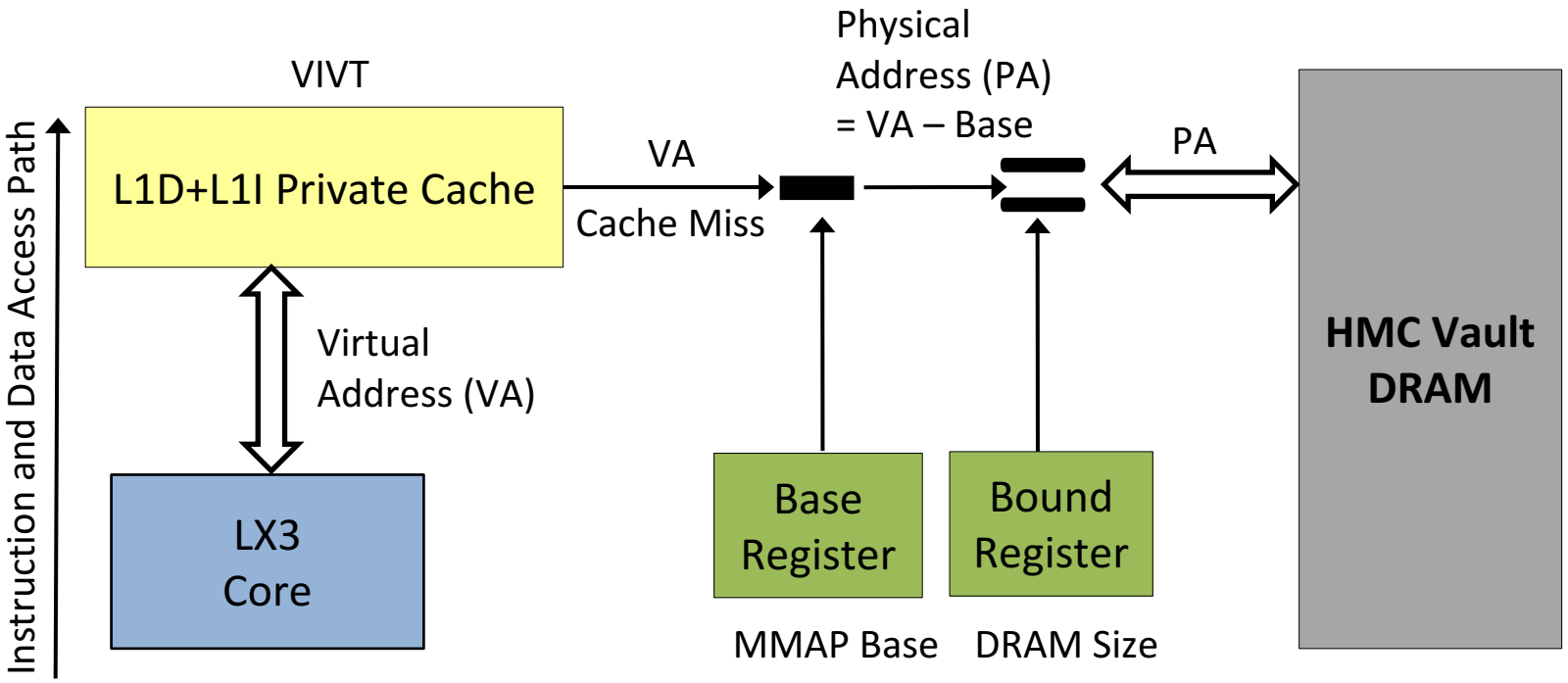
- No modifications to host CPU, to ensure easy integration
- Minimal or no modifications to the host operating system
- Consistent pointers between host code and kernel code to ease program development
- Minimal hardware overhead on MPU for address translation

Address Space Management

- To enable consistent pointers, MPU physical memory linearly mapped to host virtual address space using *mmap()*
- MPU API for memory allocation inside mmap'ed range



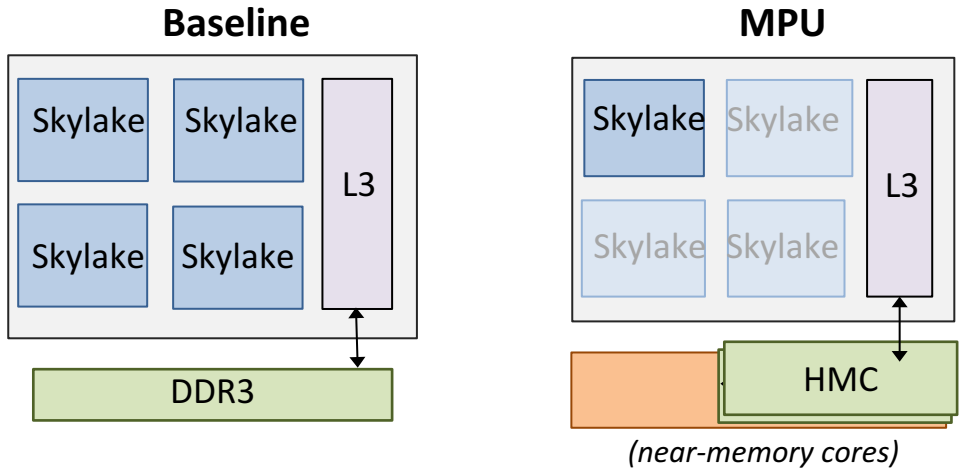
Code/Data Access Mechanism



Performance Model

- Built upon ZSim, a fast multi-core simulator
  - Instruction driven simulation, based on dynamic binary translation (DBT)
  - Modified to model MPU architecture
  - ~150-200x slower than real hardware

- Why is it fast?
  - Based on DBT (Intel PIN), so no functional simulation required
  - No full system simulation, emulates system calls
  - Bound-Weave algorithm to reduce slowdown due to core-to-core interaction



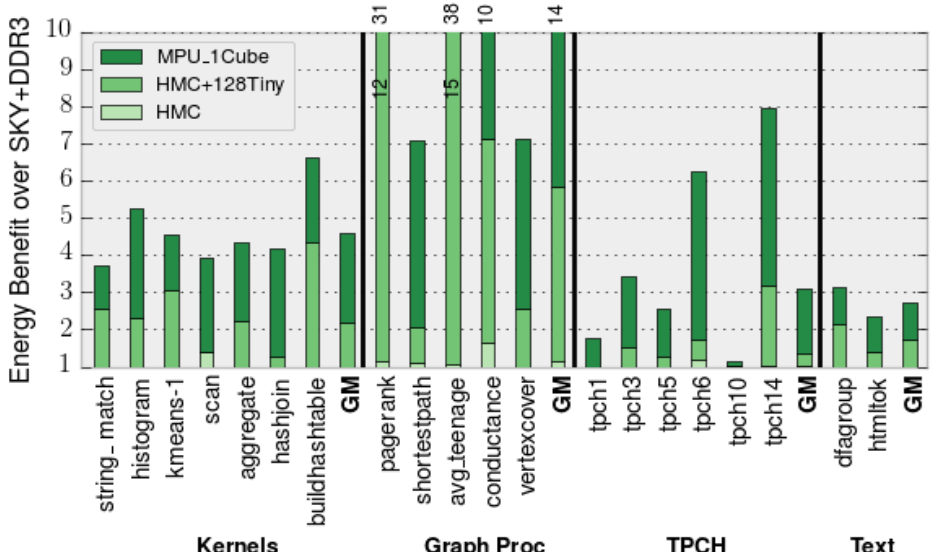
Power Model

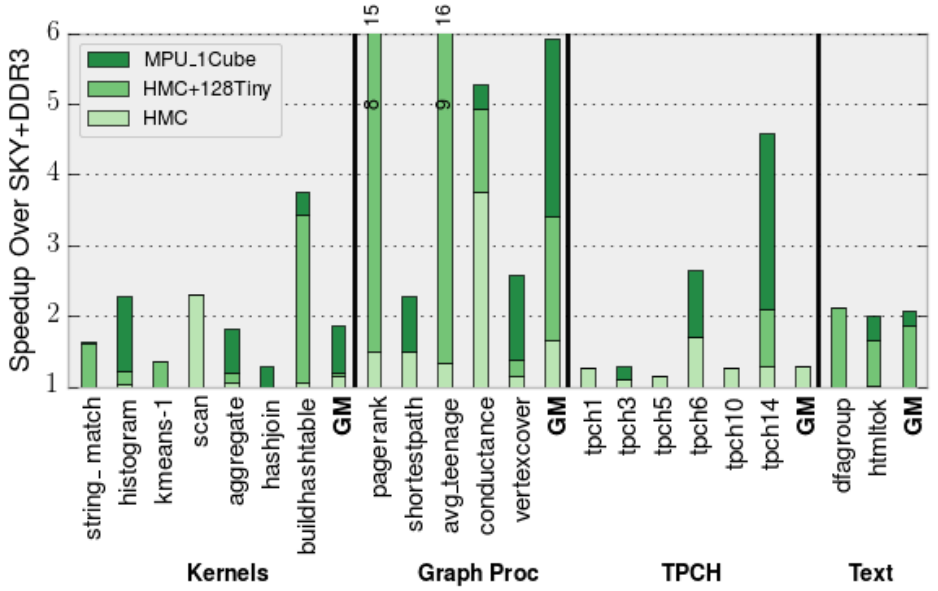
- McPAT-like power model integrated with Zsim
- Built using power/energy data from published literature and datasheets for DDR3, HMC, LX3, Westmere
- Cache power model using CACTI

Main Findings - Energy Reduction Analysis

- *HMC* versus *Baseline*
  - HMC provides substantially lower (12x) access energy per bit than DDR3 and higher (6x) bandwidth at the cost of higher (2.5x) interface power
  - Still, HMC provides energy reduction for very few workloads – scan, conductance, tpch6, over baseline. For other workloads, 10% more energy consumed than baseline
  - Reason: **Insufficient bandwidth utilization**

- *HMC_128Tiny* versus *HMC*
  - Most power savings come from turning off the 3 out-of-order cores+cache on the baseline (2.9x).
  - Overall, 1.3x to 3.1x power reduction.

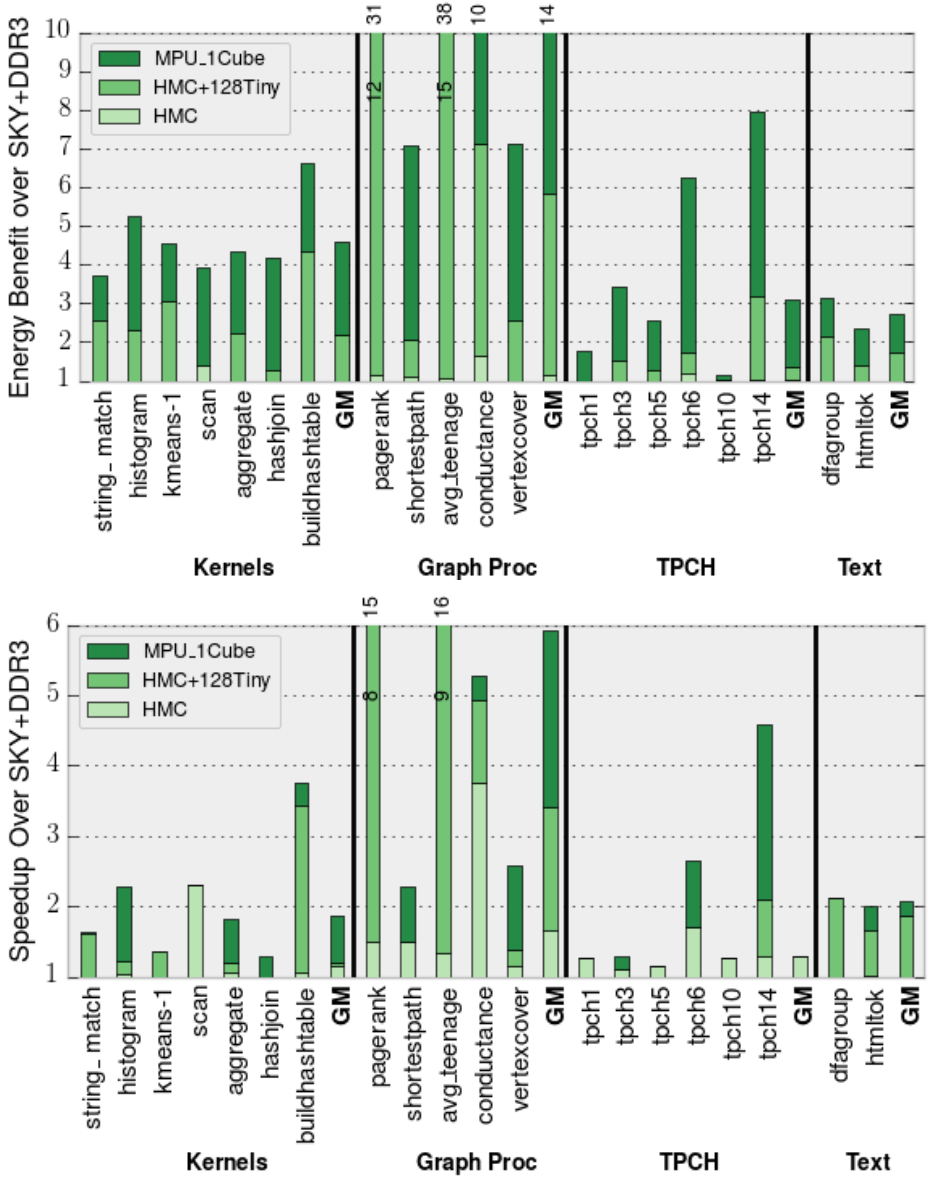




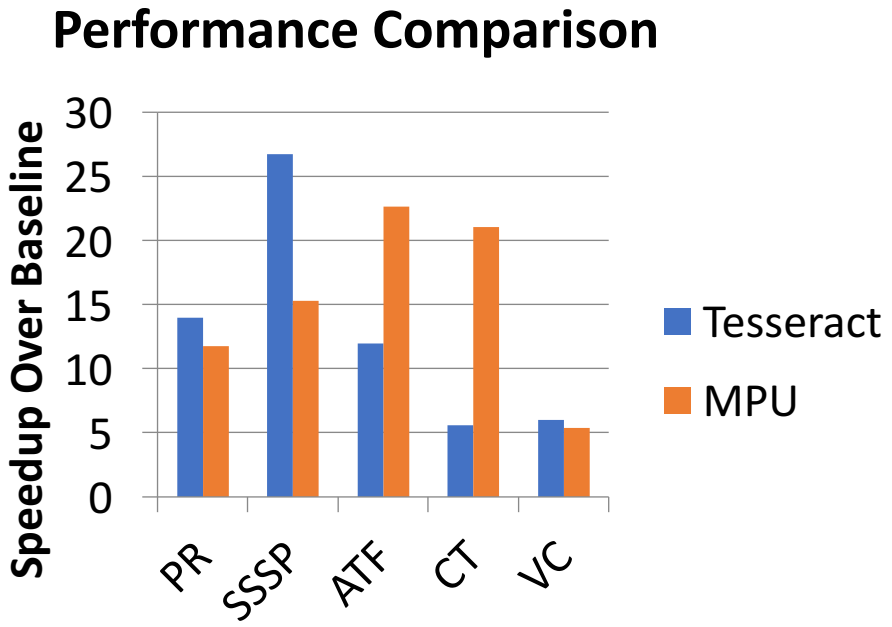
Main Findings - Energy Reduction Analysis

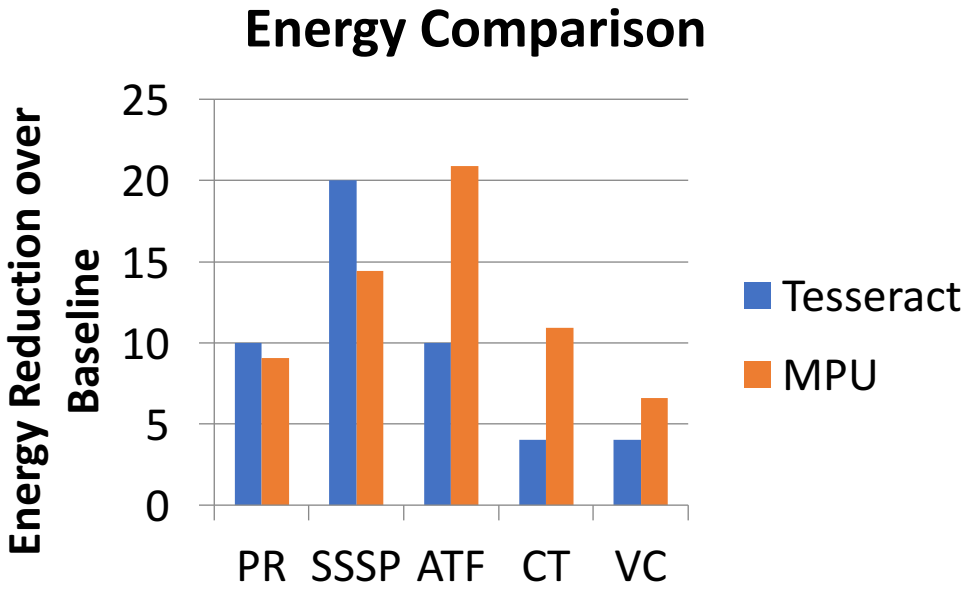
- *MPU_1Cube* versus *HMC_128Tiny*
  - Main source of power saving is the 4x reduction in static DRAM interface power due to fewer SERDES links

- *MPU_1Cube* versus *Baseline*
  - Most power savings come from turning off the 3 out-of-order cores+cache on the baseline
  - Baseline spends most power on *static core power* (50-80%). MPU_1Cube achieves most power reduction (2.9x) on this component



MPU versus Tesseract (ISCA'15)





*MPU provisioned with same capacity and vault count as Tesseract

Baseline: 32 4GHz 4-issue OoO + 128GB HMC

Hardware Specification

Specification	Value
Skylake Specifications	4 cores, 4-issue, 3.5 GHz, 32KB 8-way L1D, 256 KB 8-way L2, 8MB 16-way shared L3
MPU Specifications	128 cores, in-order, 500 MHz, 16 KB private L1D & L1-I
MPU Latencies	1 cycle hit, 20 cycle miss (40ns), 1 cycle non-memory insts,25-cycle out-of-vault latency
MPU Power	0.0056W dynamic [25], 0.0014W static (per core),0.03W SRAM static power

Power/Energy Simulation Parameters

Parameter	Value
Skylake Power Factor	1.1; 1-core DynamicPower=1.1*IPC
Skylake Static Power	11.88W; 2.97W per core x 4; assuming 35% uncore
SRAM Static Power	0.03W; lstp devices assumed
DDR3 Static Power	2.5; static power at 12.8 GB/sec
DDR3 Access Energy	23.3nJ per 64-byte; 70 pJ/bit at 12.8 GB/sec & 2.5W static power
HMC Static Power	6W; all 4 HMC links ON
HMC Internal Access Energy	1.95nJ; per 64-byte access, 3.8 pJ/bit
HMC External Access Energy	3.06nJ; per 64-byte access at 5.98 pJ/bit
HMC Internal Read Bandwidth	160 GB/sec
HMC External Read Bandwidth	80 GB/sec; Assuming all 4 HMC links are ON

MPU Area/Power

Specification	Value
Area and Power of LX3 Core	0.044mm^2, 7.1mW
Area of LX3 core including caches	0.257mm^2
Area and Power of 1 Compute Tile	2.06mm^2, 57mW
Total Area and Power of MPU Logic Die	33.4mm^2, 3.5W

Previously Completed Research

- ISA Wars: Understanding the relevance of ISA in modern processors being RISC or CISC to Performance, Power and Energy on Modern Architectures
  - Undertook this research in first year as PhD student
  - **Main Takeaway**: Decades of compiler and hardware research has enabled efficient handling of both RISC and CISC ISAs. Thus, ISA being RISC or CISC is not relevant to performance or energy.
  - Published in ACM Transactions on Computer Systems (**TOCS**), March **2015**

- Design and Analysis of an APU for Exascale Computing
  - Undertook this research over Summer 2016 as a Co-Op at AMD Research
  - Presents quantitative and qualitative analysis of the various aspects of the APU architecture (chiplets, interposers, 3D die stacking, multi-level memories, among others) for a future Exascale supercomputer
  - **Main Takeaways**: 3D stacked DRAM critical to meet area, performance, power constraints set by DOE, additional level of planar DRAM necessary to meet capacity, chiplet-based design to ensure high die yield and re-usability across market segments
  - Published in High Performance Computer Architecture (**HPCA**), February **2017**