Hardware-Software Co-Design for Efficient Graph Application Computations on Emerging Architectures

The DECADES Team
Princeton University
Columbia University
The DECADES Project

• Software Defined Hardware (SDH)
  • Design runtime-reconfigurable hardware to accelerate data-intensive software applications
    • Machine learning and data science
    • Graph analytics and sparse linear algebra

• DECADES: heterogeneous tile-based chip
  • Combination of core, accelerator, and intelligent storage tiles
    • Princeton/Columbia collaboration led by PIs Margaret Martonosi, David Wentzlaff, Luca Carloni

• Our tools are open-source!
  • https://decades.cs.princeton.edu/
Graphs and Big Data

• Machine learning and data science process large amounts of data
  • Huge strides in dense data (e.g. images)

• Graph databases and structures can efficiently represent big data
  • What about sparse data (e.g. social networks)?

• Graph applications in big data analytics
  • E.g. recommendation systems

Images from TripSavvy, Neo4j, and Twitter
Modern Technology Trends and Big Data

- Modern system designs employ specialized hardware (e.g. GPUs and TPUs), accelerator-oriented heterogeneity, and parallelism
  - Significantly benefit **compute-bound** workloads
- Amdahl’s Law perspective: faster compute causes relative memory access time to increase
  - Leads to memory latency bottlenecks
- Many graph applications are **memory-bound**
- Datasets are massive and growing exponentially
  - The ability to process modern networks has not kept up

*Image from Synopsys*

We need efficient graph processing techniques that can scale!
Graph Applications: Access Patterns are Irregular

- Iterative, frontier-based graph applications
  - Describes many graph processing workloads (e.g. BFS, SSSP, PR)
- *Indirect* accesses to neighbor data
  - Conditionally populate next frontier

```python
for node in frontier:
    val = process_node(node)
    for neib in G.neighbors(node):
        update = update_neib(node_vals, val, neib)
        if(add_to_frontier(update)):
            new_frontier.push(neib)
```

Indirect memory access due to neighbor locations

Frontier nodes processed in parallel

Stores IDs of nodes to process

stores node property data

Updates are irregular!
LLAMAs: The Problem

• Irregular accesses experience cache misses
• Long-LATency Memory Accesses (LLAMAs): irregular memory accesses in critical path

Programs see disproportionate performance impact from just a few LLAMAs. Our work seeks to address these.
Our Approach: FAST-LLAMAs

FAST-LLAMAs: Full-stack Approach and Specialization Techniques for Hiding Long-Latency Memory Accesses

- A **data supply** approach to provide performance improvements in graph/sparse applications through latency tolerance
  - Programming model to enable efficient producer/consumer mappings by explicitly directing LLAMA dependencies
  - Specialized hardware support for asynchronous memory operations
- Achieves up to an **8.66x** speedup on the DECADES architecture
Outline

Introduction

Decoupling Overview

FAST-LLAMAs

Results

Conclusions
Decoupling for Latency Tolerance

- **Decoupling**: static division of a program into [data] Producer/Consumer pair
  - Cores run independently; heterogeneous parallelism

- Ideally, the Producer runs ahead of the Consumer
  - Issues memory requests early and enqueues data

- The Consumer consumes enqueued data and handles complex value computation
  - Data has already been retrieved by the Producer

### Memory Hierarchy

- **Producer** (memory access, address computation)
- **Consumer** (value computation)

The Producer runs ahead and retrieves data for the Consumer. The Producer has no dependencies, so the Consumer never stalls after warm-up period!
Decoupling for Asynchronous Accesses

- Decoupling into two instruction streams removes dependencies on each slice
  - The *Producer* might have to stall waiting for long-latency loads, but doesn’t use data
  - Usually, only the *Consumer* needs the data

- **Asynchronous accesses**: accesses whose data is not later used on the *Producer*
  - The *Producer* does not occupy pipeline resources waiting for their requests
  - These loads **asynchronously** complete early and are maintained in a **specialized buffer**
  - Asynchronous loads help maintain longer *Producer* runahead and exploit MLP

**Diagram:**
- The *Producer* issues several **non-asynchronous** loads
  - Waits for prev req to return with data
  - Decoupling into two instruction streams removes dependencies on each slice
  - The *Producer* might have to stall waiting for long-latency loads, but doesn’t use data
  - Usually, only the *Consumer* needs the data

**Notes:**
- Asynchronous loads are not later used on the *Producer*; allows *Producer* runahead
- Memory Level Parallelism

**No asynchronous loads; stalling due to long memory latency**
FAST-LLAMAs Tolerates Latency in Graph Applications by Making LLAMAs Asynchronous

Fast-LLAMAs eliminates LLAMA dependencies, so decoupling achieves latency tolerance on graph applications!

LLAMAs dominate runtime

Application data dependency graph

(a) In-Order Execution

Fast-LLAMAs are issued asynchronously after warm-up period

(b) Fast-LLAMAs

for node in frontier:
    val = process_node(node)
    for neib in G.neighbors(node):
        update = update_neib(node_vals, val, neib)
        if add_to_frontier(update):
            new_frontier.push(neib)

Iterative, frontier-based graph application template
FAST-LLAMAs Hardware Support

- Asynchronous access buffer holds data for asynchronous accesses
  - FIFO queue as simple hardware addition compatible with modern processors
    - E.g. in-order RISC-V core tiles

- Asynchronous memory access specialized hardware support
  - Memory request tracked in buffer
  - Returned data enqueued for Consumer
  - Modified (via ALU) data written to memory

Blue arrows indicate datapath additions for asynchronous accesses. The numbers illustrate the order in which data proceeds through the system.
Graph/Sparse Applications

• **Elementwise Sparse-Dense (EWSD):** Multiplication between a sparse and a dense matrix.

• **Bipartite Graph Projections (GP):** Relate nodes in one partition based on common neighbors in the other.

• **Vertex-programmable (VP) graph processing primitives:**
  
  • **Breadth-First Search (BFS):** Determine the distance (number of node hops) to all nodes.

  • **Single-Source Shortest Paths (SSSP):** Determine the shortest distance (sum of path edge weights) to all nodes.

  • **PageRank (PR):** Determine node ranks based on the distributed ranks of neighbors.

Can be efficiently sliced automatically

Currently require explicit annotations for efficient slicing

Images from Wikipedia
FAST-LLAMAs Tolerates Latency for Graph Applications

2 Parallel In-Order RISC-V Core Tiles

1 FAST-LLAMAs Pair of In-Order RISC-V Core Tiles

Speedups range from 2.39-8.66x. 
Memory-bound application performance idealization.
Conclusions

Overview
FAST-LLAMAs: hardware-software co-design for efficient graph application computations
• Applications are sliced and mapped onto producer/consumer pairs
• Achieves up to 8.66x speedup over single in-order core

The DECADES Team
People: Margaret Martonosi, David Wentzlaff, Luca Carloni, Juan L. Aragón, Jonathan Balkind, Ting-Jung Chang, Fei Gao, Davide Giri, Paul J. Jackson, Aninda Manocha, Opeoluwa Matthews, Tyler Sorensen, Esin Türeci, Georgios Tziantzioulis, and Marcelo Orenes Vera
Website: https://decades.cs.princeton.edu/
Presenter: Aninda Manocha
• amanocha@princeton.edu
• https://cs.princeton.edu/~amanocha

Open-Source Tools
Applications:
https://github.com/amanocha/FAST-LLAMAs
Compiler:
https://github.com/PrincetonUniversity/DecadesCompiler
Simulator:
https://github.com/PrincetonUniversity/MosaicSim
DECADES RTL: Coming soon!

This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.