Tiny but Mighty: Designing and Realizing Scalable Latency Tolerance for Manycore SoCs

Marcelo Orenes-Vera, Aninda Manocha, Jonathan Balkind, Fei Gao, Juan L. Aragón, David Wentzlaff and Margaret Martonosi
Problem: Memory bottlenecks

- Modern system designs employ hardware accelerators, heterogeneity, and parallelism
  - Significantly benefits *compute-bound* workloads

- Applications that are **memory-bound due to irregular memory access patterns** do not scale well with the number of cores
  - *Sparse neural networks*, as a result of network pruning to reduce model storage
  - *Graph algorithms*, recommendation systems, etc.
Opportunity: Mitigating the latency of Indirect Memory Accesses (IMAs)

• Their data footprint is constantly increasing, putting more pressure in the memory system.

• IMAs arise from pointer indirection, e.g. A[B[i]]
  • Since array A is often very big (e.g., millions of edge/nodes in graph analytics) and accesses are unpredictable IMAs often incur in poor cache locality and their latency dominates the runtime

1 Runtimes measured on a simulated in-order core.
**Challenge A: Mitigate IMAs in Manycores**

1. Manycores often have slim cores without OoO structures

2. A prefetcher in each core would cause significant per-core overhead

3. Heterogeneous tiles (e.g. accelerators) might need memory tolerance too.

4. Prefetching in the LLC require changes specific to mem hierarchy
Challenge B: Easy hardware integration

1. **Deep microarchitecture changes are difficult to incorporate** due to the verification burden

2. Faster path to SoC silicon by integrating off-the-shelf IP blocks

3. Easier adoption when **not modifying the memory hierarchy** not existing IP blocks

---

Rapid Prototyping via SoC Integration

- Third-Party Accelerator IP
- Third-Party Processor IP
- In-House Hardware Block
Challenge C: Memory-access specialization without adding new instructions

1. Not modifying the cores IP means no new instructions (no ISA modifications)

2. It’s ideal to bring to L1 the cache-friendly accesses and bypass the cache-averse ones

3. Provide HW advantages but with the illusion of only using SW optimizations with an API

Sparse Matrix-Vector multiplication (SPMV) code

```c
for (i=0;i<N;i++){
    for (k=ptr[i];k<ptr[i+i];k++){
        result[i] += val[k] * A[B[k]];
    }
}
```

```
for (i=0;i<N;i++){
    specialized_prefetch(A,B,ptr[i+1]);
    for (k=ptr[i];k<ptr[i+i];k++){
        result[i] += val[k] * consume();
    }
}
```

Mitigating the IMA
Our Approach: Out-of-core mem. latency tolerance

- Mitigating the latency of **IMAs without modifying cores or mem. hierarchy**
  - Ease to integrate via the NOC
  - ISA-agnostic
  - Provides **memory-level parallelism** to the thin cores of a manycore
- Enables **decoupling** and **prefetching** SW optimizations via an API that only uses existing memory instructions
Contributions

• RTL implementation taped-out into silicon
  • Reusable open-source hardware block
  • Real area numbers
  • Extensive testing using formal verification

• Scalable Latency tolerance
  • Multiple instances
  • Instances shared across cores, protected access

• Real-world OS and compiler support
  • MAPLE’s API supports virtual memory
  • Programmed from SMP Linux
  • Open-source compiler pass targets MAPLE’s API
Outline

- Motivation, challenges and contributions
- Background
- MAPLE
- Evaluation & Results
- Conclusions, contact, and open-source repo
Decoupling for Latency Tolerance

At DAE [Smith ‘82] (Decoupled Access Execute) ideally the Access runs ahead of the Execute

- The Access core issues memory requests early and the return data is enqueued
- The Execute consumes data from the queue and handles complex value computation

Some applications may involve long-latency loads, where the Execute waits for their data to be ready
Layers that Prior Work Modifies

- **Application**
- **OS, Compiler, DSL (Domain Specific Language) backend**
- **Architecture (ISA)**
- **Processor Core Microarchitecture**
- **RTL (e.g. Verilog)**

- **DeSC** [Ham MICRO '15]
  - Compiler targeting new ISA inst.
- **DAE** [Smith ISCA '82]
  - μarch changes visible with new ISA inst.
Layers in which Prior Work Operates

- **Software Prefetching for IMAs** [Ainsworth CGO’18]
- **Clairvoyance** [Tran CGO’17]
- **Programmable Prefetching** [Ainsworth ASPLOS’18]
- **DeSC** [Ham MICRO’15]

  - Compiler-only approach
  - Compiler targeting new ISA inst.
  - μarch changes visible with new ISA inst.

- **DAE** [Smith ISCA ’82], **Pipette** [Nguyen MICRO’20]
- **DROPLET** [Basak HPCA ’19], **IMP** [Yu MICRO’15]

  - Use predictors to trigger prefetches

- **Slipstream** [Sundaramoorthy ASPLOS ’00]

  - Use predictors to orchestrate streams
Layers in which Prior Work Operates

**Our Approach**
- Provides an API that can be targeted by Compiler or DSL backend
- API operations use existing memory instructions, so it is ISA-agnostic
- Our hardware approach doesn’t modify the Core

**Prior Work**
- *Software Prefetching for IMAs [Ainsworth CGO’18]*
- *Clairvoyance [Tran CGO’17]*
  - Compiler-only approach
- *Programmable Prefetching [Ainsworth ASPLOS’18]*
- *DeSC [Ham MICRO’15]*
  - Compiler targeting new ISA inst.
- *DAE [Smith ISCA ‘82], Pipette [Nguyen MICRO’20]*
  - μarch changes visible with new ISA inst.
- *DROPLET [Basak HPCA ‘19], IMP [Yu MICRO’15]*
  - Use predictors to trigger prefetches
- *Slipstream [Sundaramoorthy ASPLOS ‘00]*
  - Use predictors to orchestrate streams
Layers in which MAPLE Operates

Our Approach

Provides an API that can be targeted by Compiler or DSL backend

API operations use existing memory instructions, so it is ISA-agnostic

Our hardware approach doesn’t modify the Core

Application

OS, Compiler, DSL (Domain Specific Language) backend

Architecture (ISA)

Out-of core HW: MAPLE

Processor Core Microarchitecture

RTL (e.g. Verilog)
Outline

- Motivation, challenges and contributions
- Background
- MAPLE
- Evaluation & Results
- Conclusions, contact, and open-source repo
Software API for Decoupling with MAPLE

- Compiler pass for decoupling (e.g. similar to DeSC) divides the program into Access and Execute threads and targets MAPLE’s API for Produce/Consume
  - Decoupling by itself doesn’t give latency tolerance
  - Need Memory-Level Parallelism

- Targeting MAPLE’s hardware achieves better performance due to its memory-parallelism
Decoupling with MAPLE

Produce path (steps 1-6)

Core 1 (behaving as the Access core) will supply data to Core2 (Execute)

‘Access’ or ‘Execute’ are roles taken by software threads rather than a core-type (as in prior art)
Decoupling with MAPLE

**Consume** path (A-C)

Core 1 (behaving as the Access core) will supply data to Core2 (Execute)

‘Access’ or ‘Execute’ are roles taken by software threads rather than a core-type (as in prior art)
Decoupling with MAPLE

Core 1 (behaving as the Access core) will supply data to Core2 (Execute)

‘Access’ or ‘Execute’ are roles taken by software threads rather than a core-type (as in prior art)
MAPLE Hardware Design

- **Load Pipeline**: Consume data
- **Configuration Pipe**: manage queues, config MMU, debug
- **Store Pipe**: Push data and pointers (to fetch)
Prefetching Loops of IMAs: LIMA

Loading $A[B[i]]$ for a range

- Base address of arrays $A$ and $B$ are configured
- Fetches $B$ in chunks, which are then accessed word by word to calculate the index to array $A$. 

![Diagram showing the process of prefetching loops of IMAs using LIMA.](image-url)
Prefetching with MAPLE

- Prefetch IMAs in tight inner loops with a single instruction and then consume from MAPLE

The LIMA subunit prefetches Loops of IMAs

- Can also do individual prefetching

- Advantages over the hardware and software state of the art (see full-paper)

```
Original SPMV code Snippet

for (i=0;i<N;i++){
    for (k=ptr[i];k<ptr[i+i];k++){
        y += val[k]*A[B[k]];  //IMA
    }
    result[i]=y;
}

Prefetching version with MAPLE

LIMA(A,B,ptr[i],ptr[i+1];
for (i=0;i<N;i++){
    for (k=ptr[i];k<ptr[i+i];k++){
        y += val[k]*CONSUME();
    }
    result[i]=y;
    LIMA(A,B,ptr[i+1],ptr[i+2];
}
```
OS support

• MAPLE can be instantiated many times, e.g., in a tiled architecture.
  • Each unit is addressed as a separate memory-mapped page (protected access)
  • A process can map multiple MAPLE units

• The API implementation hides the management of physical MAPLE units
  • The software interface only deals with the abstract concept of queues
Hardware Integration with OpenPiton

- We integrated MAPLE into the open-source OpenPiton [Balkind ASPLOS’16] manycore, on its own tile

- We use in-order, OS-capable RISC-V cores: Ariane
  - Using the API, loads/stores are routed to MAPLE via the NoC
  - Mem-mapped address range
Evaluating MAPLE full-system on FPGA

- **Experimental setup:** SoC prototype on FPGA VC707 composed of 2 Ariane Core and 1 MAPLE Tile. We evaluate applications full-stack on top of Linux v5.6-rc4
MAPLE for decoupling

MAPLE decoupling provides \textbf{2.3x speedup over SW-only decoupling}, and outperforms traditional parallelism across the board, 1.5x over 2-cores do-all
MAPLE for programmable prefetching

- Geomean speedup 1.7× over no prefetching
- Up to 2.4× for SPMV

Decreases the average load latency by 1.9x
Scaling core counts sharing a MAPLE unit

- Evaluating 8 cores sharing the same MAPLE instance
  - 4 decoupling queues
  - Can handle twice that
  - Area-efficient
MAPLE enables prefetching and decoupling SW optimizations with specialized HW to make it effective even with slim, in-order cores.

✓ Can be used on a SoC generator framework, as a *plug-n-play latency tolerance mechanism*

Full-stack SoC prototype evaluation shows geomean speedups of 2.3x over software-only decoupling and prefetching

*Our HW-SW co-design benefits from program knowledge and hardware specialization.*
Contributions

- RTL implementation taped-out into silicon
  - Reusable open-source hardware block
  - Real area numbers
  - Extensive testing using formal verification
- Scalable Latency tolerance
  - Multiple instances
  - Instances shared across cores, protected access
- Real-world OS and compiler support
  - MAPLE’s API supports virtual memory
  - Programmed from SMP Linux
  - Open-source compiler pass targets MAPLE’s API

Contact

- Marcelo Orenes-Vera, movera@princeton.edu
- https://decades.cs.princeton.edu/

Project Repositories

- github.com/PrincetonUniversity/maple
- github.com/PrincetonUniversity/openpiton
- github.com/PrincetonUniversity/DecadesCompiler

MAPLE demos on FPGA

- Decoupling with four tiles
  https://youtu.be/elkQcMFSvoo
- Decoupling and prefetching on top of Linux
  https://youtu.be/YRbsjqlTOM