Asynchronous memory access unit for general purpose processors

Luming Wang, Xu Zhang, Tianyue Lu, Mingyu Chen

Institute of Computing Technology, Chinese Academy of Sciences

Nov 7, 2022 @Bench 2022

Far Memory

The prevalence of in-memory computing applications has caused massive growth in memory demand

Far Memory

NVM Main Memory:
3D Xpoint/Optane

New Medium besides DRAM

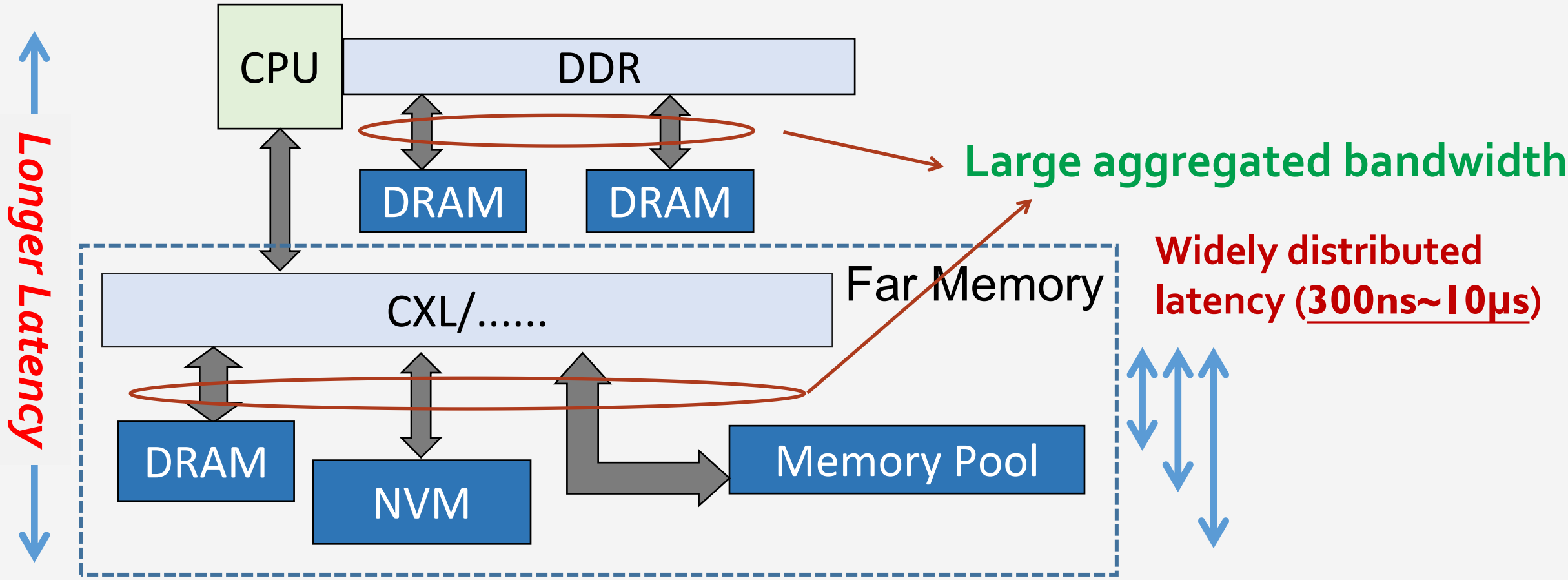
Compute Express Link

New Interconnect Techologies and Protocols

To meet the growing demand for memory,
far memory is introduced into data centers

Far Memory

The future servers will use a variety of emerging far memory to meet the growing memory demands



Leave a challenge for CPU to fully utilize the far memory resources!

The Limitation of Current Out-of-Order Processors

OoO Execution

Dynamical scheduling more load/store instructions

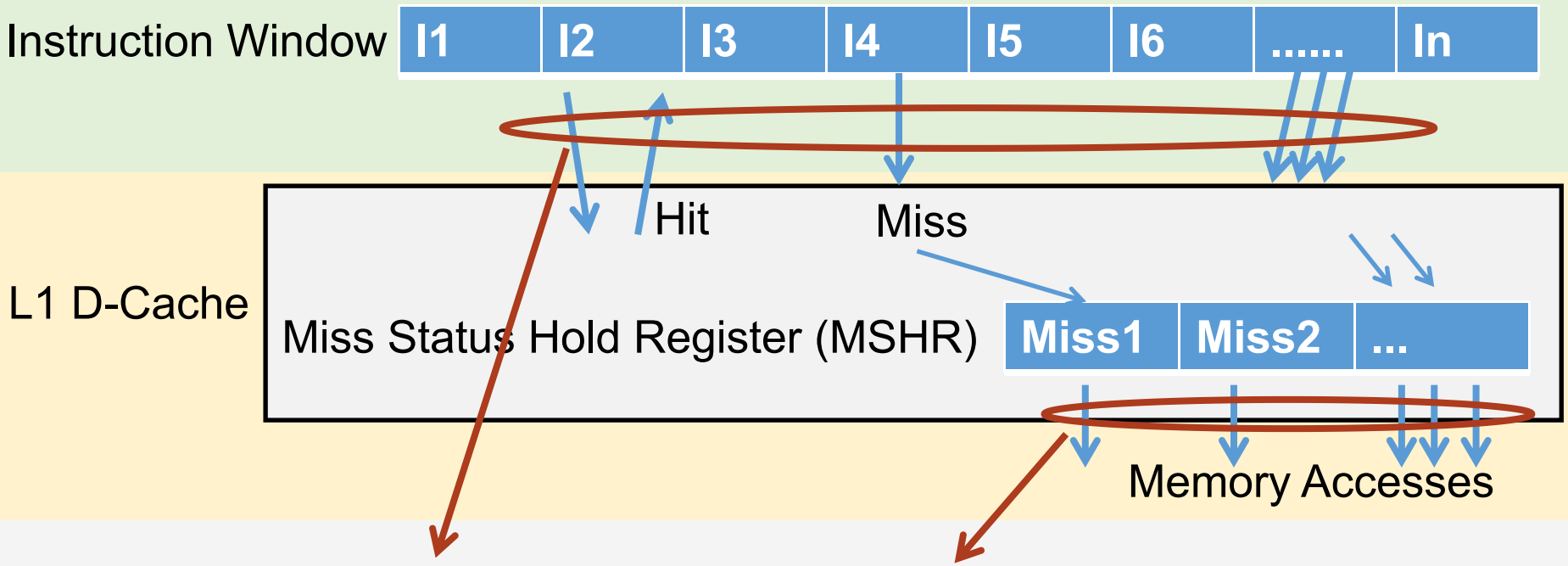
+

Non-Blocking Cache

Handling several miss memory requests

↓

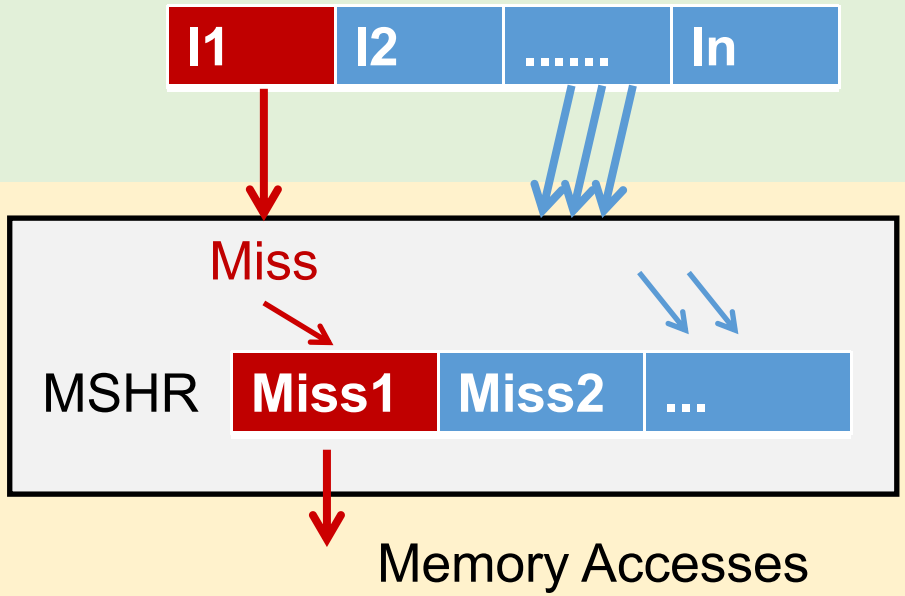
Memory-Level Parallelism (MLP)



The maximum number of in-flight memory requests is restricted by critiacl hardware resources (e.g. ROB, MSHRs, etc.)

The Limitation of Current Out-of-Order Processors

ROB



Cache

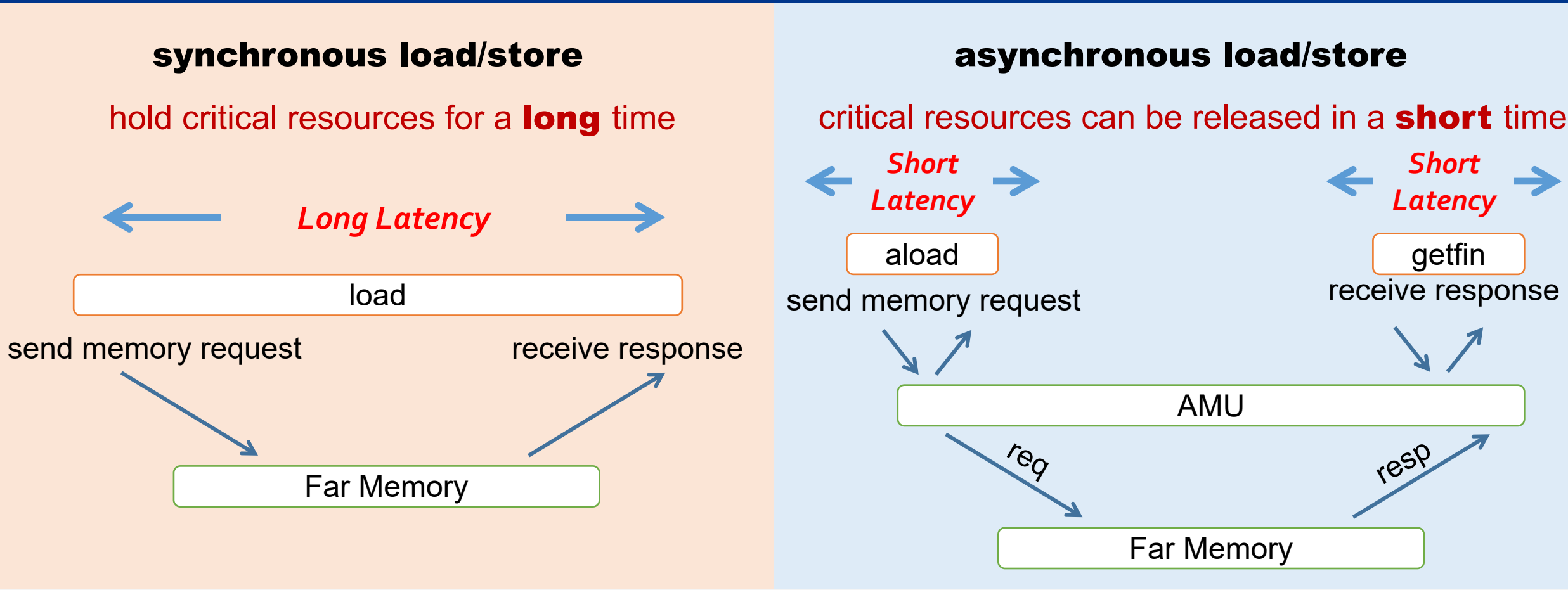
Example: Intel Alder Lake (12th Core)
Golden Cove (P-Core) 3.4GHz

512 ROB entries

L2 cache can handle
48 outstanding misses

A 2GHz CPU faced with $1\mu s$ latency, to avoid stall:
~2000 entries ROB and Physical RF
thousand of items in MSHR
Criticial Resources Limit MLP!

Our Solutions



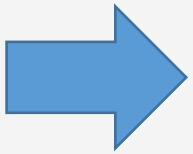
- Asynchronous Memory access Unit (AMU)
  - provide asynchronous semantic for load/store
  - split sending request and getting response as two instructions

Design Challenges of AMU

- How to handle a large number of memory requests?

Register Presure

```
for (i = 0; i < N; ++i) {
    r1 = load A[i];
    // calculation depends on r1
}
```



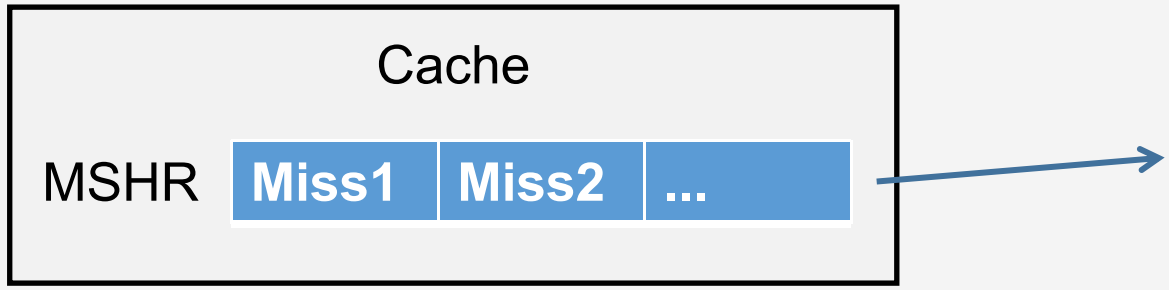
```
for (i = 0; i < N; i+=4) {
    r1 = aload A[i];
    r2 = aload A[i+1];
    r3 = aload A[i+2];
    r4 = aload A[i+4];
    // calculation depends on r1
    // calculation depends on r2
    // calculation depends on r3
    // calculation depends on r4
}
```

Requires large amounts of storage spaces for holding the data of asynchronous memory requests

Design Challenges of AMU

- How to handle a large number of memory requests?

The limited number of MSHRs



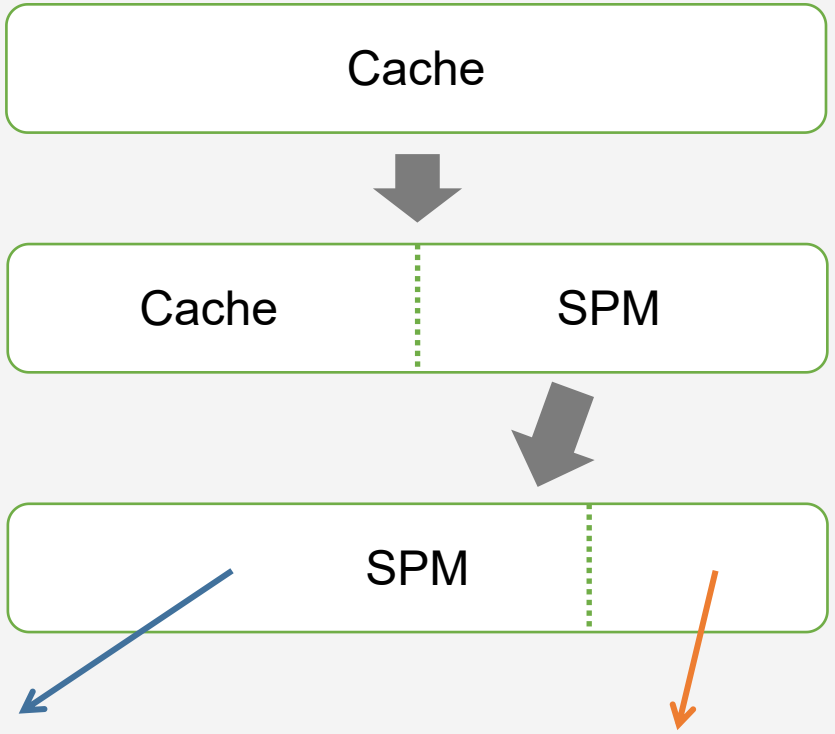
Entries < 100

Not Enough for tracking the states of in-flight asynchronous memory requests

Requires large amounts of spaces for holding the states of asynchronous memory requests

AMU design

- Challenge: How to handle a large number of memory requests?
  - Register presure
  - The limited number of MSHRs
- Solution:
  - split part of L2 Cache as ScratchPad Memory (SPM)
  - use SPM to replace the role of physical register files and MSHRs.



SPM Data Area
- replacing the role of register files
- application accessable
- holding the data of memory operations

SPM Metadata Area
- replacing the role of MSHRs
- hardware managed
- holding the states of in-flight memory operations

AMU design

- Instructions
  - **`aload/store` *`Rd, Rs1, Rs2`***
    - invoking a data movement request between SPM (***`Rs1`***) and memory (***`Rs2`***)
    - a request id is allocated for tracking the request (***`Rd`***)
  - **`getfin` *`Rd`***
    - getting an completed request's id (***`Rd`***).
    - if there is no finished request, the instruction returns a failure code.
- Configuration Registers
  - access granularity: enabling varies access granularity
  - access pattern: for complex asynchronous memory access requests
  - priority: for QoS

Programming model

Step 1: Config AMU

- Max parallelism (length of metadata area)
- Access granularity

```
#define MAX_PARALLELISM 256
int *mem_to_access; // memory to be accessed

// AMU Configuration
acfgwr(MAX_PARALLELISM, AMQLEN);
acfgwr(sizeof(int), RWLEN);

int *spm_data_area = (int *)alloc_spm_addr(sizeof(int));
// issue an aload request
int id = aload(spm_data_area, &far_mem_to_access);

// process other
while(id != getfin()) { /* process other */ }
// access SPM via standard load/store
printf("%d\n", *spm_space);
```

Programming model

Step 1: Config AMU

- Max parallelism (length of metadata area)
- Access granularity

Step 2: Issue an aload/astore request

- Allocate SPM space

```
#define MAX_PARALLELISM 256
int *mem_to_access; // memory to be accessed

// AMU Configuration
acfgwr(MAX_PARALLELISM, AMQLEN);
acfgwr(sizeof(int), RWLEN);

int *spm_data_area = (int *)alloc_spm_addr(sizeof(int));
// issue an aload request
int id = aload(spm_data_area, &far_mem_to_access);

// process other
while(id != getfin()) { /* process other */ }
// access SPM via standard load/store
printf("%d\n", *spm_space);
```

Programming model

Step 1: Config AMU

- Max parallelism (length of metadata area)
- Access granularity

Step 2: Issue an aload/astore request

- Allocate SPM space

Step 3: Wait for finish

- Use *getfin* for checking

```
#define MAX_PARALLELISM 256
int *mem_to_access; // memory to be accessed

// AMU Configuration
acfgwr(MAX_PARALLELISM, AMQLEN);
acfgwr(sizeof(int), RWLEN);

int *spm_data_area = (int *)alloc_spm_addr(sizeof(int));
// issue an aload request
int id = aload(spm_data_area, &far_mem_to_access);

// process other
while(id != getfin()) { /* process other */ }
// access SPM via standard load/store
printf("%d\n", *spm_space);
```

Programming model

Step 1: Config AMU

- Max parallelism (length of metadata area)
- Access granularity

Step 2: Issue an aload/astore request

- Allocate SPM space

Step 3: Wait for finish

- Use *getfin* for checking

Step 4: Access

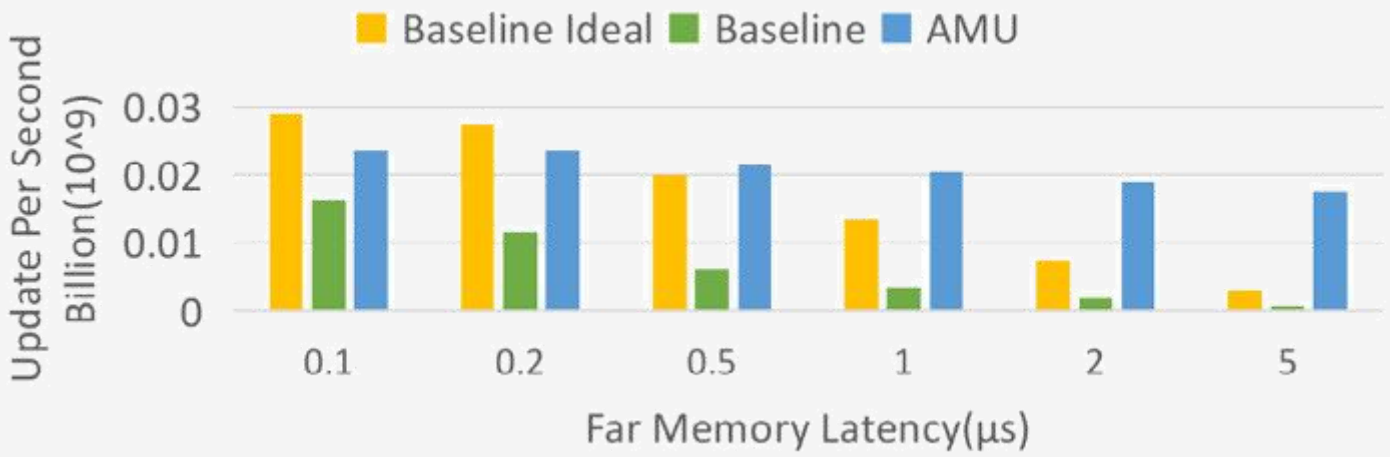
- Via standard load/store.

```
#define MAX_PARALLELISM 256
int *mem_to_access; // memory to be accessed

// AMU Configuration
acfgwr(MAX_PARALLELISM, AMQLEN);
acfgwr(sizeof(int), RWLEN);

int *spm_data_area = (int *)alloc_spm_addr(sizeof(int));
// issue an aload request
int id = aload(spm_data_area, &far_mem_to_access);

// process other
while(id != getfin()) { /* process other */ }
// access SPM via standard load/store
printf("%d\n", *spm_space);
```

AMU Design Discussion

- Efficient ID management
  - hardware id management instead of software id allocation
  - integrated with out-of-order and speculation mechanisms
- Consistency
  - SPM does not provide consistency checking as MSHRs
  - relaying on software to handle the consistency
    - Multi-thread software generally adhere Data-Race-Free model
    - For other programs, it is possible to use a combination of hardware and software solution to handle the consistency problem of asynchronous access to far memory.

Evaluation



- A modified GEM5 for cycle-accurate simulation of AMU model
- Random access benchmark from HPCC
- AMU can better tolerate a large range of access latency

Asynchronous memory access unit for general purpose processors

Luming Wang, Xu Zhang, Tianyue Lu, Mingyu Chen

Nov 7, 2022 @Bench 2022

Q & A

中国科学院大学
University of Chinese Academy of Sciences

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

先进计算机系统研究中心
Center for Advanced Computer Systems