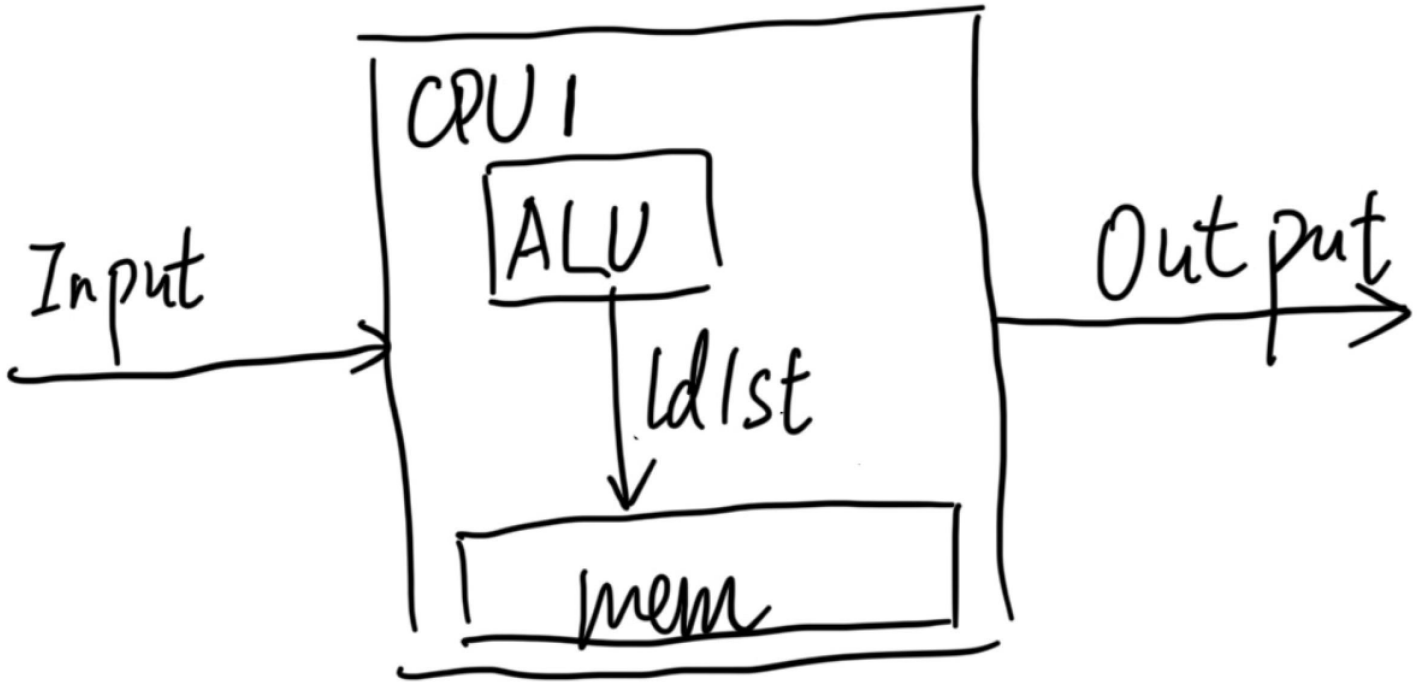
CXLMemUring

Type2 device upgrade

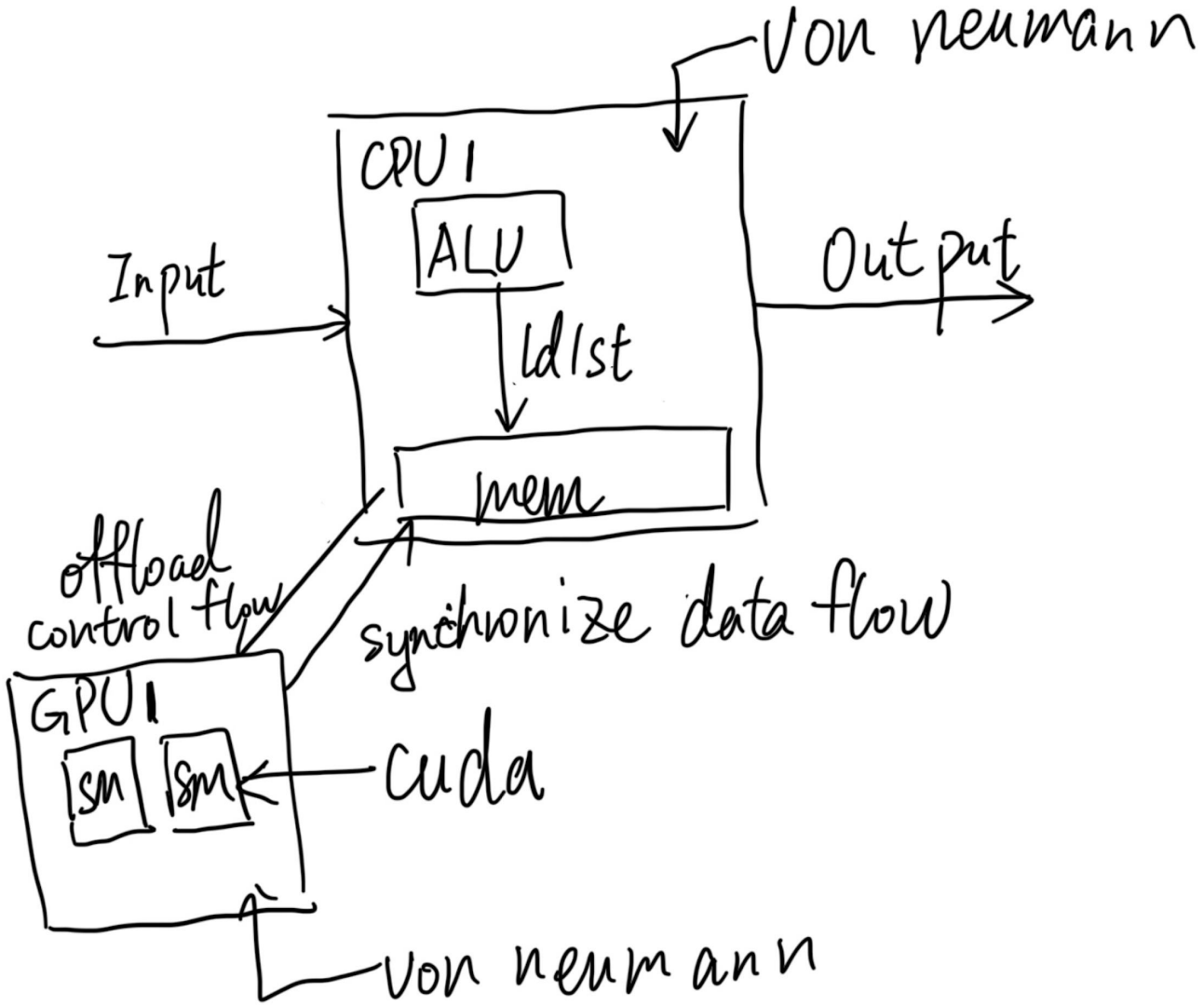
Yiwei Yang, Yang Man, Andrew Quinn

Problems

- Von Neumann Memory wall



Problems


von neumann
CPU
ALU
Input
Output
ldlst
mem
offload
control flow
synchronize data flow
GPU
sm
sm
cuda
von neumann

SoTA solutions

- Async loading engine
  - NVLink
    - Industry Leading fabric in Bandwidth
    - NCCL IOMMU (phys memory translation) conflict ~ No CPU scene
  - TPU async loading engine
    - mpi allreduce for tensor abstraction
    - overlap rate is low when it’s dependency pointer chasing
  - In order-core AMU
    - object back scratchpad
    - With context fetching to local to compute

SoTA solutions's problem

- Polluting the cache
- Side channel
- Not instantly consumed cacheline lead to mispredict
  - timing sequence is hard even with sophisticated prefetcher
    - VMT - Todd Austin's Graph prefetching dependency offloading to Little core
    - Leap - Husan's software solution of profiling guided prefetching
    - FastSwap - Software defined cost model for disaggregation
  - The right target address is hard to predict if it's random or long dependency chain
    - Especially when the 4kB R/W is 3000ns

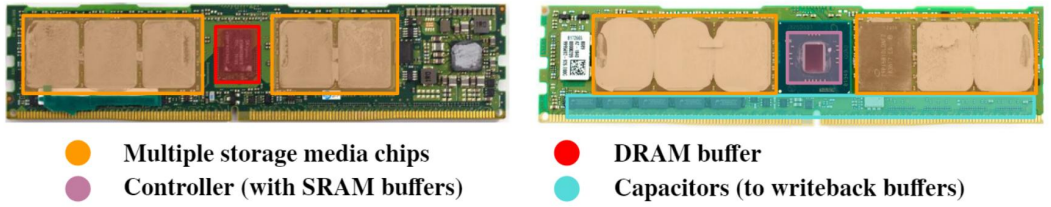
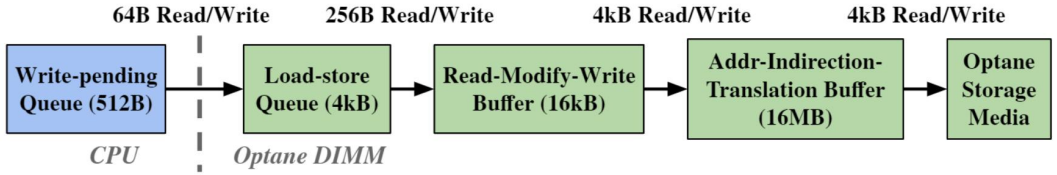


Figure 1: Components inside an Optane DIMM.



SoTA solutions

- A profiling guided RDMA/smartNIC way with their own memory metadata managed
  - AIFM
    - Has array library to implement MCF's data structures allocated in continuous memory
  - Carbink
    - Pointer managed as a batch and integrate to TCMalloc
  - STKY
    - Namsong Kim's smartNIC solution, near NIC Arm core handle zswap to compress to OS zpage
  - MIRA
    - Yiying Zhang's Software-implemented Optane Memory mode for managing remote Page metadata
    - Function level Static Analysis for remote offloading and deal with page metadata locally
    - Spec2017 MCF 345MB WSS 12MB Metadata that fits LLC
    - what if 2GB WSS ~100MB Metadata?

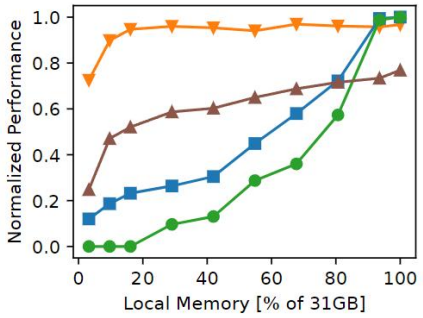


Figure 16. DataFrame Performance.

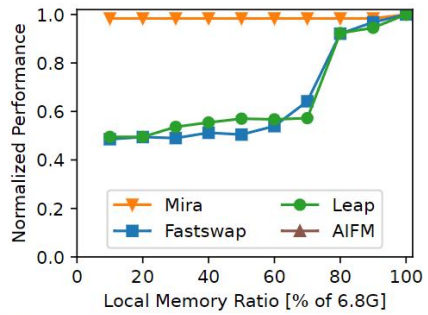


Figure 17. GPT-2 Performance.

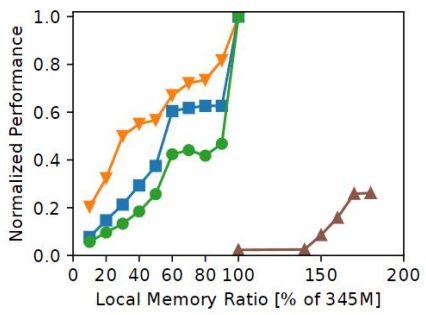


Figure 18. MCF Performance.

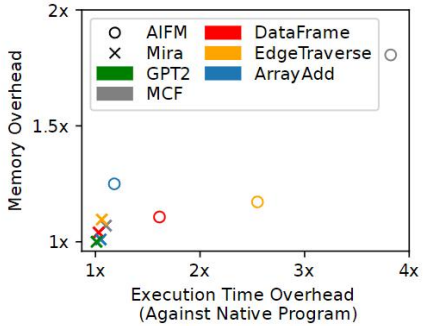


Figure 19. Runtime Overhead. *when running on full local memory.*

SoTA solutions's problem

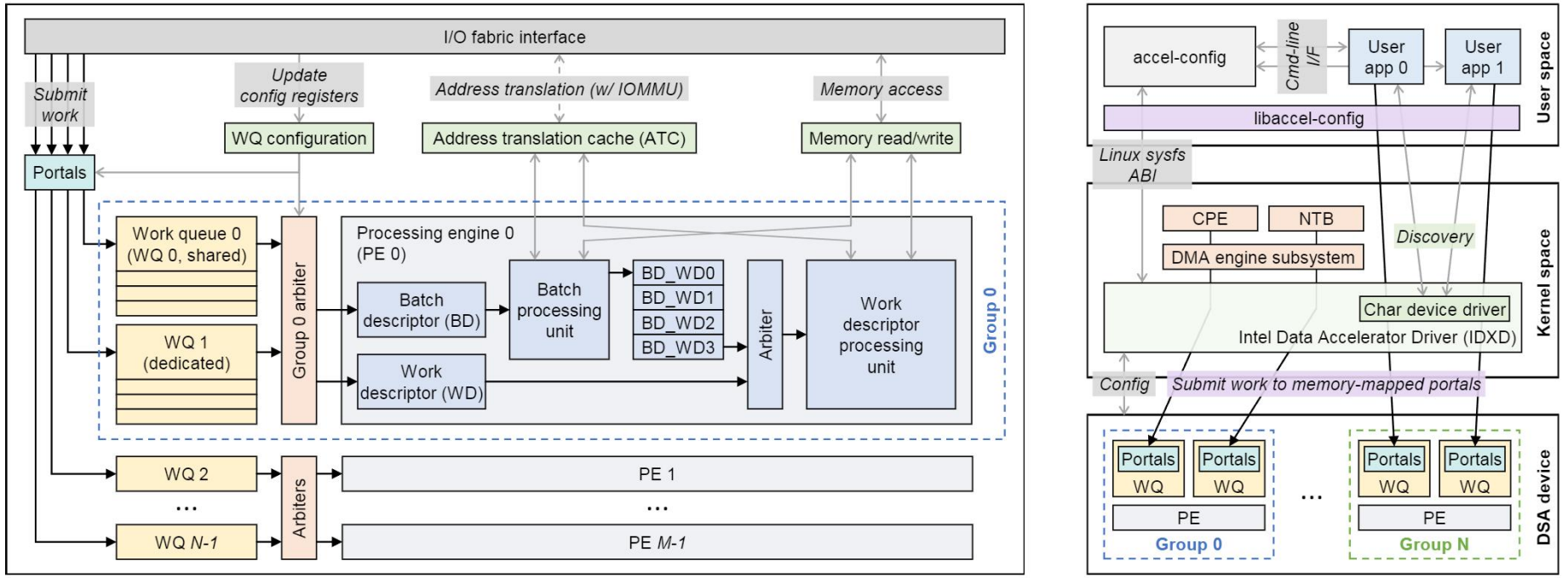
- C++ object >64B cacheline < 4KB page
- WSS's metadata is bigger than LLC, don't scale, like Optane Memory mode 2RTT

```
1  %SEdge = rmem.cache_section {#type = "direct", #line = 2M, ...}
2  %SNode = rmem.cache_section {#type = "full", #line = 128B, ...}
3
4  func.func @trvs_graph_opt(%arg0: !remotable<struct<edge>>){
5    scf.for %i <- %0 to %num_edges step %elements_per_line {
6      // prefetch %n_ahead elements ahead from far memory
7      rmem.fetch %SEdge, %arg0 + %i + %n_ahead
8      // wait for current requested data (at %i) to be in cache
9      rmem.wait %SEdge, %arg0 + %i
10     // get corresponding phyiscal address (paddr) of cache line
11     %wide_cache_line = rmem.paddr %SEdge, %arg0 + %i
12
13     scf.for %j = %0 to %elements_per_line {
14       // directly load element in (already resolved) cache line
15       %1 = memref.load %wide_cache_line[%j]
16
17       // use later element in the line to prefetch node elements
18       %2 = memref.load %wide_cache_line[%j + %n_ahead_node]
19       // node elements may be in cache already, fetch if not
20       rmem.fetch_if_not_in_cache %SNode, %2 -> from
21       rmem.fetch_if_not_in_cache %SNode, %2 -> to
22
23       // wait for node elements to be in cache and access
24       rmem.wait %SNode, %1 -> from
25       %3 = rmem.paddr %SNode, %1 -> from
26       rmem.wait %SNode, %1 -> to
27       %4 = rmem.paddr %SNode, %1 -> to
28       func.call @update_node (%1, %3, %4)
29     }
30     // flush used %i element for eviciton hint
31     rmem.flush %SEdge, %i
32   }
33 }
```

Figure 14. Mira Optimizations for Graph Example. *We show optimizatios of prefetching and eviction flush, not showing others for simplicity.*

SoTA solutions

- Data Streaming Accelerator (SPDK DMAEngine but new in GNR/Zen5 for CXL.mem)
  - A ~400M-1GHz little core near CPU, use the movdir64 async for backend data movement
    - interrupt for callbacks, could do not flush ROB
  - It can put memory back LLC 400ns->20ns(Demisifying by Nam)



(a) Internal block diagram **(b)** Software architecture

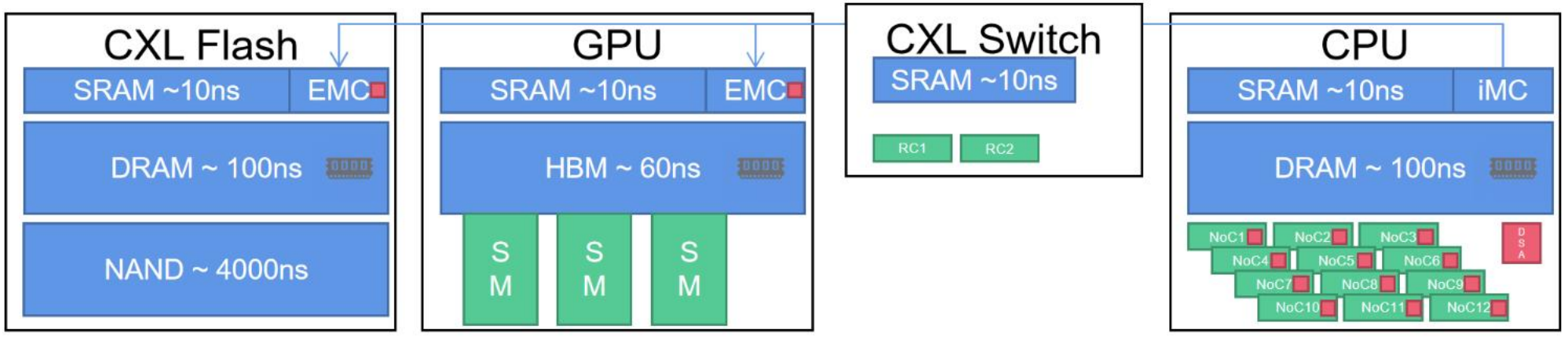
Figure 1: Architectural overview of DSA. Job descriptors are directly submitted to memory-mapped portals in each device. IOMMU allows in-device address translation, and thus memory pinning is not required.

SoTA solutions's problem

- bulk memory
- doesn't support memory operators
  - Possibly evolve with IAA in Emerald Rapids or later
- single rooted

High level idea

- Offload the Control Flow to near memory device with small data coherency
  - Not prefetching, prefetching fails
  - Not Memory metadata, Memory metadata doesn’t scale
- Run compiling once and update the remote through AOT
- Adaptively update the AOT, after knowing the cost model dynamically



1a Hardware- ISA change

- Async Load - aload
  - Basic Block id offload and coherency cacheline back to retire the aload
  - Assumption, the offloaded BB's updated cacheline is instantly consumed in CPU
  - Do not hurt existing ROB, MSHR and on chip devices.
- Remote eBPF accelerator?
  - Lower security caused by new types of side-channels. [[1]](http://www.fan-yao.com/paper/2018_HPCA_caco.pdf)
  - For the CXL.cache device, if some cache line being modified state resides in this device, and the cache line cannot be safely recovered from the dead device, which causes data loss and even kernel panic or system shutdown.
  - Solution: Adding bus-level transactional memory support, so the device will have the ability to pack a set of instructions for control plane operations and processed by bus.
    - Faster than DMA (can be protected by CXL ATS, which provide a new bit to refuse CXL.cache operation in some pages), slower than being modified state in device in some cases.
    - No need for device to make the cache line being modified, so there will no data loss if device disconnected or stop functioning. The bus can rollback the entire transaction (which is small, not huge as current Intel TSX design)

1b Address Translation between local and remote

- Full coherence and no full pagetable local or remote
- Only take minor coherency BB remote (will adaptively update)
- The dynamic running learns the panelty of remote weak cores and minor coherency will beat pure local for 3000ns latency. We seeks to hide the latency RTT.

2 Compiler Change

- Compile 2 kind of ISA based on MLIR frontend analysis
  - local intel + aload (mwait to emulate)
  - remote intel FPGA weak RISCV core RV64-ilp32 etc
- Basic Block id to store the context
  - Profiling results
  - Next time cost function re-calculate and recompile

3a Evaluation

- Proposed to find out 4 types of workloads - Metadata don’t fit local LLC
  - MRMW(Multi Reader Multi Writer)
    - Distributed OLAP ~ cassendra/ bigquery
  - SRSR
    - Ring data structure ~ Alias? Back invalidation?
  - MRSW
    - VectorDB
    - local OLAP
    - Transformer
  - Reader Only
    - MoE Inference
    - MCF

3b Evaluation

- Serving dicisions for Local NOC vs. Near endpoint SMT
  - 40 local + 8SMT*4 remote
  - 40 local * 20 node + 8SMT*4 * 20 endpoint