


清華大學 交叉信息研究院
Institute for Interdisciplinary Information Sciences, Tsinghua University

上海期智研究院
SHANGHAI QI ZHI INSTITUTE

Stream-Based Data Placement for Near-Data Processing with Extended Memory

Yiwei Li, **Boyu Tian**, Yi Ren, Mingyu Gao

Tsinghua University

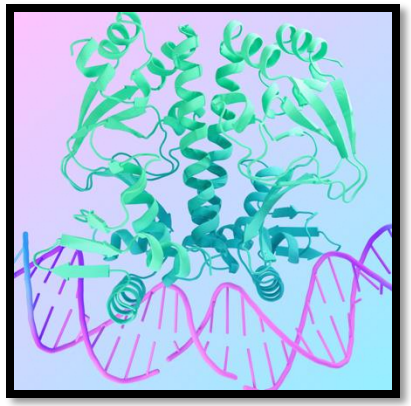
Shanghai Qi Zhi Institute

清華大學
Tsinghua University

MICRO 2024

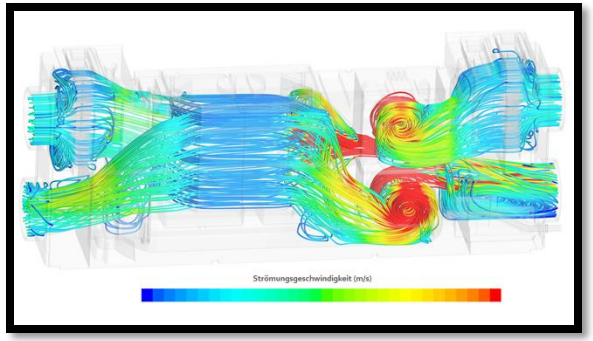
Background

- Data-intensive applications grow rapidly

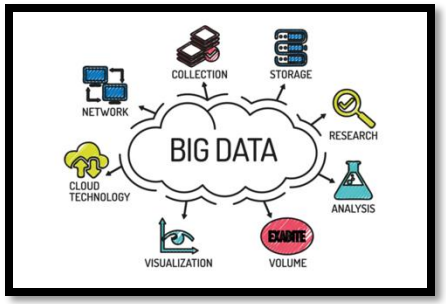


Computational biology

Machine learning



Simulation

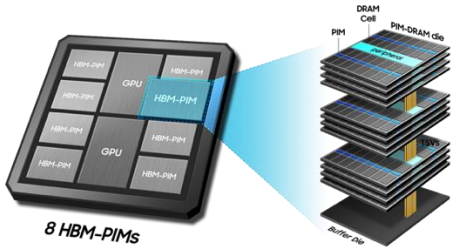


Big data

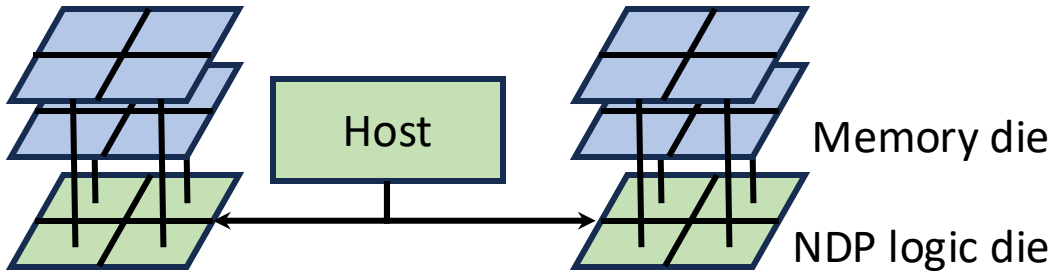
- "Memory wall" bottleneck
  - Limited bandwidth scaling and excessive data migration energy
    - ➡ Near-data processing (NDP) moves computation close to data
  - Limited capacity scaling due to limited CPU pin counts
    - ➡ Compute Express Link (CXL) allows efficient memory capacity extension

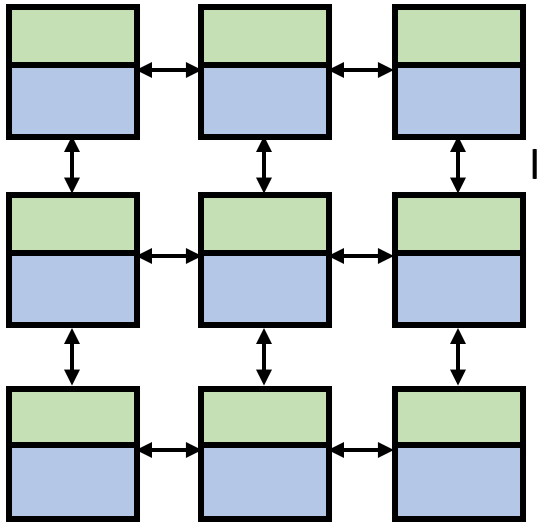
Near-Data Processing

- Near-data processing (NDP): place compute logic near data memory
  - Shorter distance → lower latency and energy
  - Higher bandwidth
- 3D-Stacked NDP architecture:
  - Incorporate compute logics inside 3D-stacked memories
  - Multiple memory stacks interconnect with each other
  - **High bandwidth but Limited capacity**



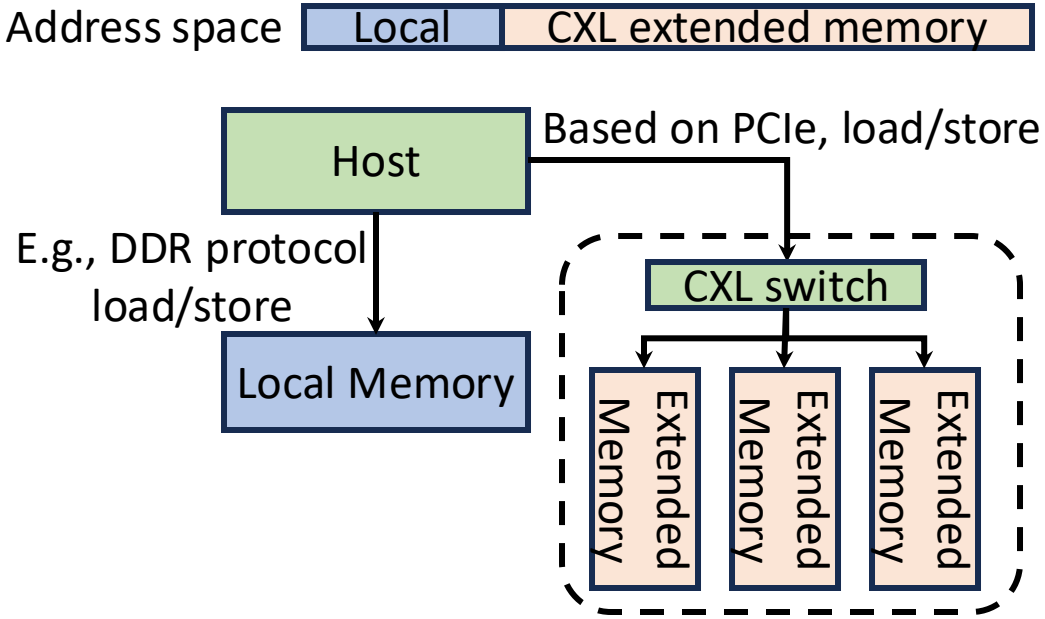
Samsung HBM-PIM (2021)
6 GB capacity each stack
1.23TB/s Off-chip bandwidth

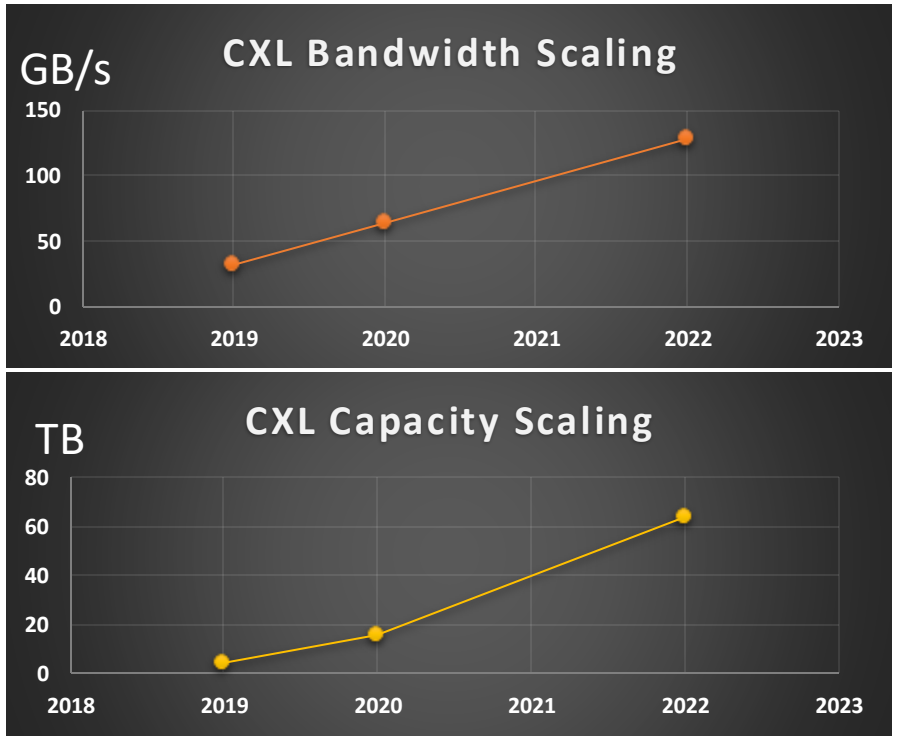




Compute Express Link

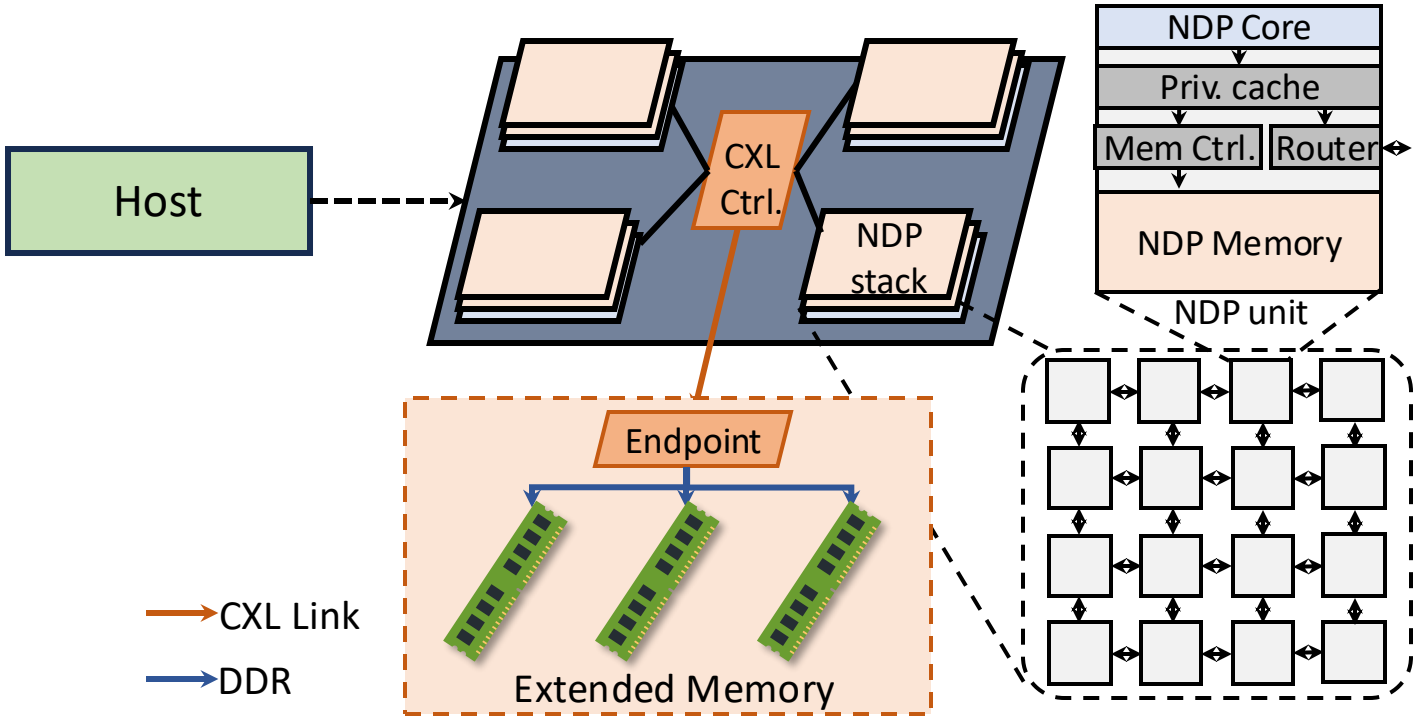
- CXL outlines memory interaction between host and devices
- We primarily focus on CXL.mem for memory extension
  - Direct load/store access in the address space
  - TB-level capacity plus scalable bandwidth support
  - Fewer CPU pins requirement compared to DDR controllers





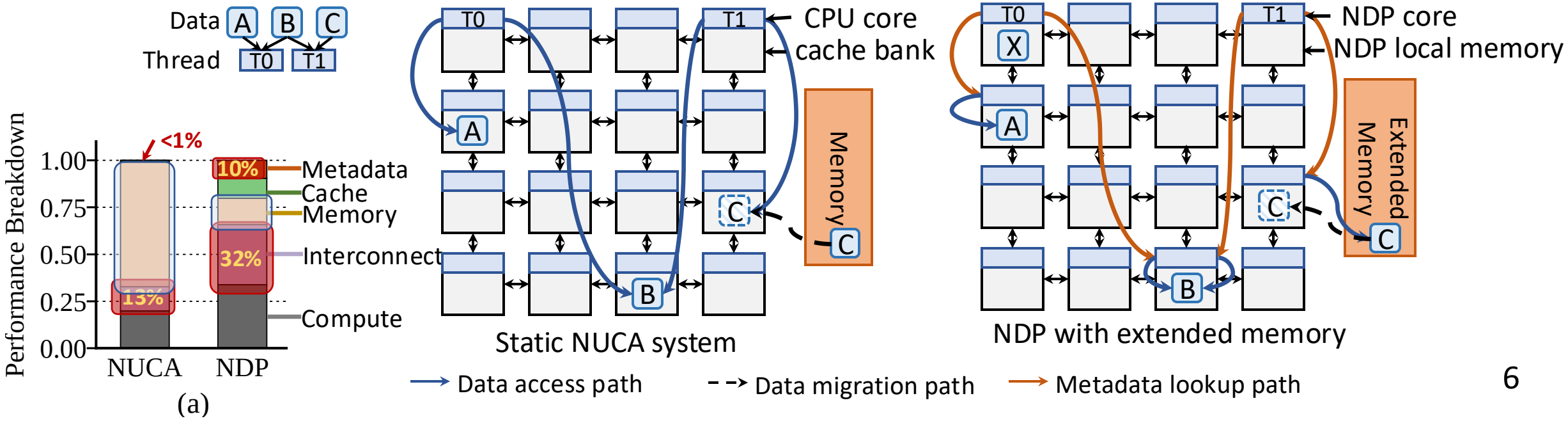
NDP with Extended Memory

- 3D stacked NDP provides TB/s bandwidth while CXL provides TB capacity
- We add extended memory to NDP to accelerate large-scale applications
  - Host offloads large-scale data-intensive tasks to NDP accelerator
  - NDP manages data between local stacks and extended CXL memory
  - For simplicity we use NDP as a distributed cache of extended memory space



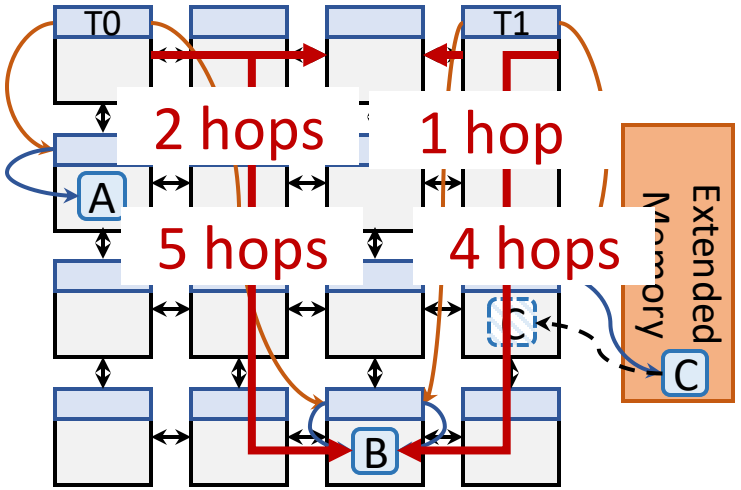
Design Challenges

- Existing systems already incorporate a distributed cache system: NUCA
- What are the differences?
  - D1: Fewer misses to extended memory in NDP thanks to larger cache capacity
  - D2: NDP has higher interconnect overheads due to high off-chip/on-chip latency
  - D3: NDP has higher metadata overheads since tags cannot store fully on-chip

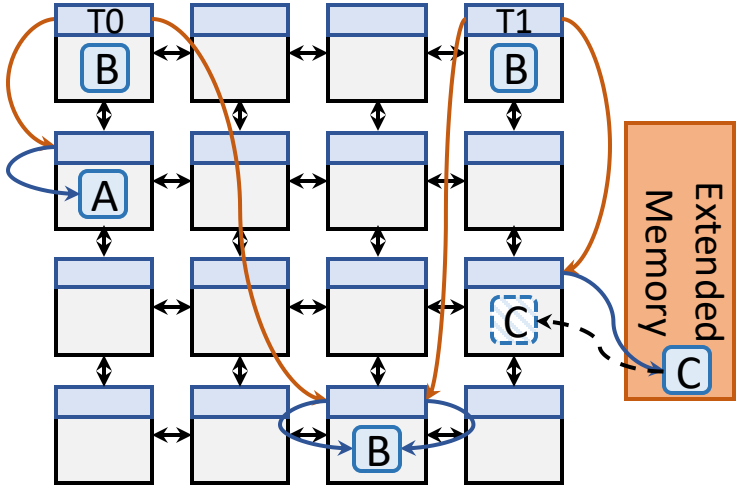


Prior NUCA Solutions for Interconnect Overheads

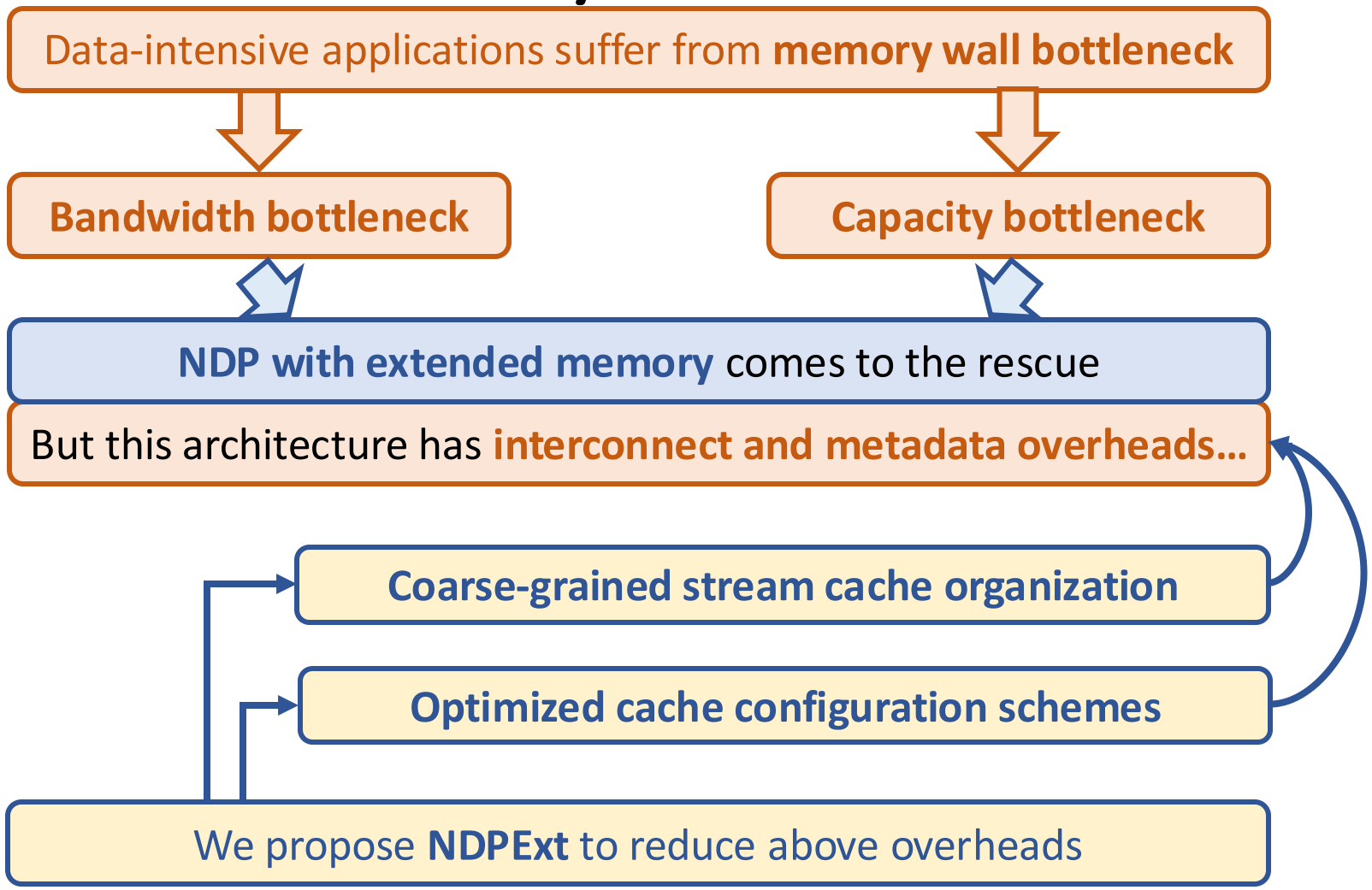
- Placing common data to the "center-of-mass"?
  - Related work: Jigsaw [PACT'13]
  - B's interconnect hops are reduced
  - But this scheme always favors cache space at the center than corner ones
  - Per cacheline placement is challenging



- Replicating every data to units
  - Related work: Nexus [PACT'17]
  - B's interconnect hops are reduced
  - But this scheme usually wastes cache space
  - Metadata overheads disallow flexible replication options

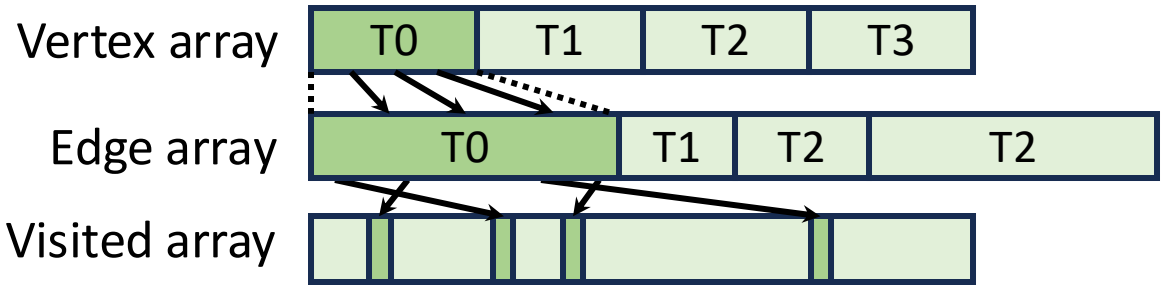


Motivation Summary from Previous Slides



Stream Cache Organization

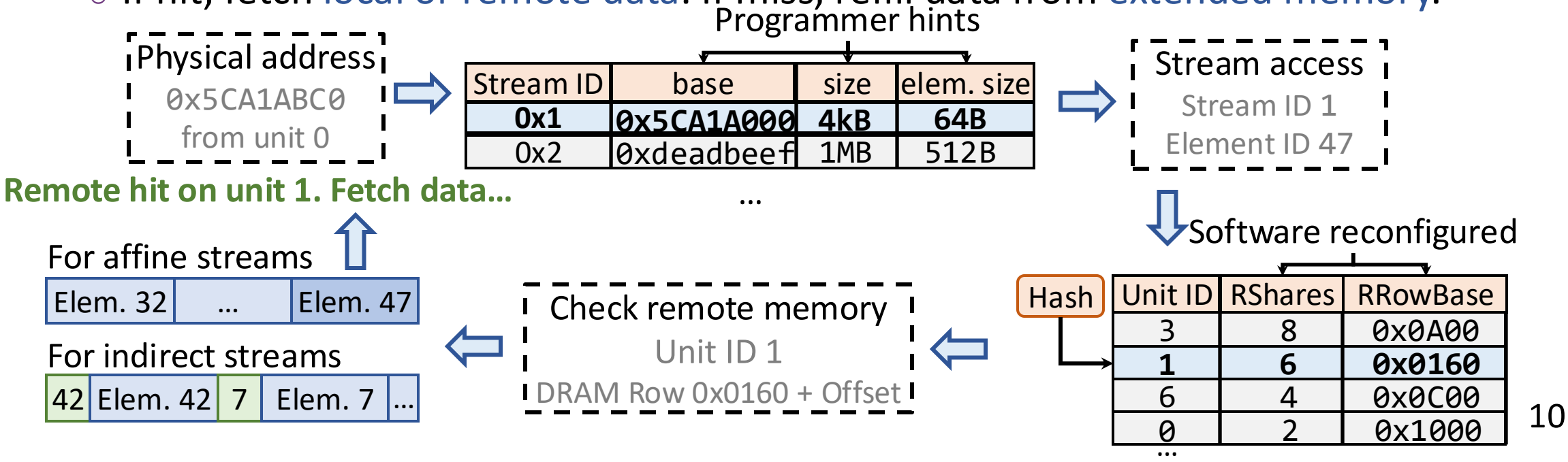
- Observation 1: most NDP applications respect to stream access patterns
- Streams: coarse-grained memory access pattern abstraction
  - Affine pattern, e.g., $addr = ax + b$
  - Indirect pattern, e.g., $addr = s[i]$
  - Used in prior work for prefetching [ISCA'19], instruction offloading [HPCA'22], etc.
- For example, parallelized graph traversal for thread 0
  - Access pattern of vertex array: `i` **Affine stream**
  - Access pattern of edge array : `j + vertex[i]` **Affine stream**
  - Access pattern of visited array: `edge[j + vertex[i]]` **Indirect stream**



- We can use such programming hints for coarse-grained caching!

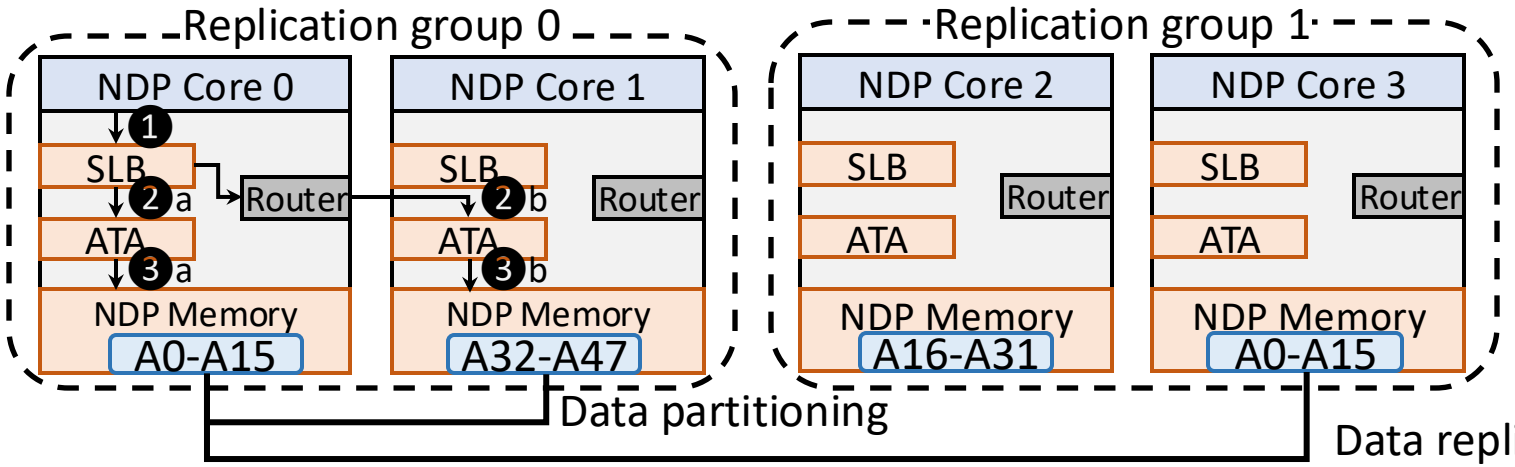
Stream Cache Organization

- Observation 2: high metadata costs due to fine-grained mapping
  - Storing streams instead of cachelines for NDP can reduce metadata overheads
- Basic access flow
  - Lookup stream metadata to identify stream and element ID
  - Fetch stream remap table to identify possible unit ID and DRAM location
  - If hit, fetch local or remote data. If miss, refill data from extended memory.



Stream Cache Organization

- We cache intermediate tables on-chip to reduce lookup overheads
  - Stream lookahead buffer (SLB) is used to cache stream remapping information
  - Affine tag array (ATA) is used to store tags for affine streams
- We support per-stream replication groups
  - Unlike prior work, each stream can be replicated or partitioned to any unit groups
    - For each stream, partial units are tagged with the same replication group (RGroups)
    - Within each RGroup, stream data are partitioned using their allocation shares (RShares)
  - Hardware-efficient lookup procedure

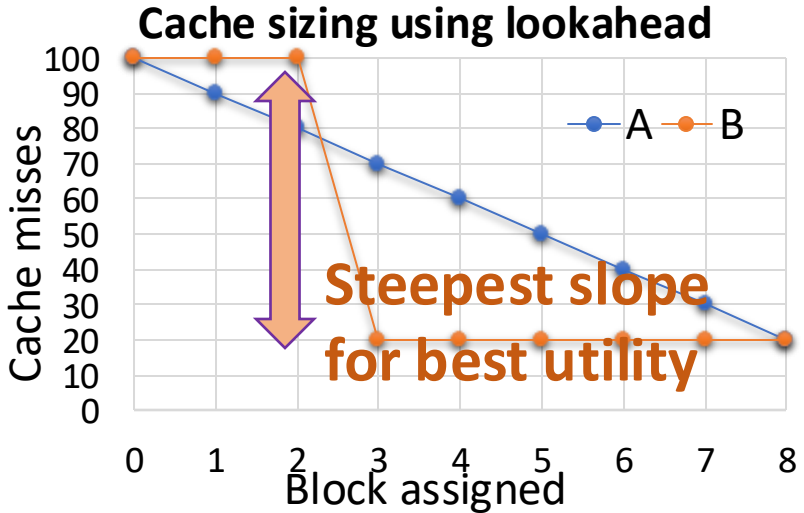


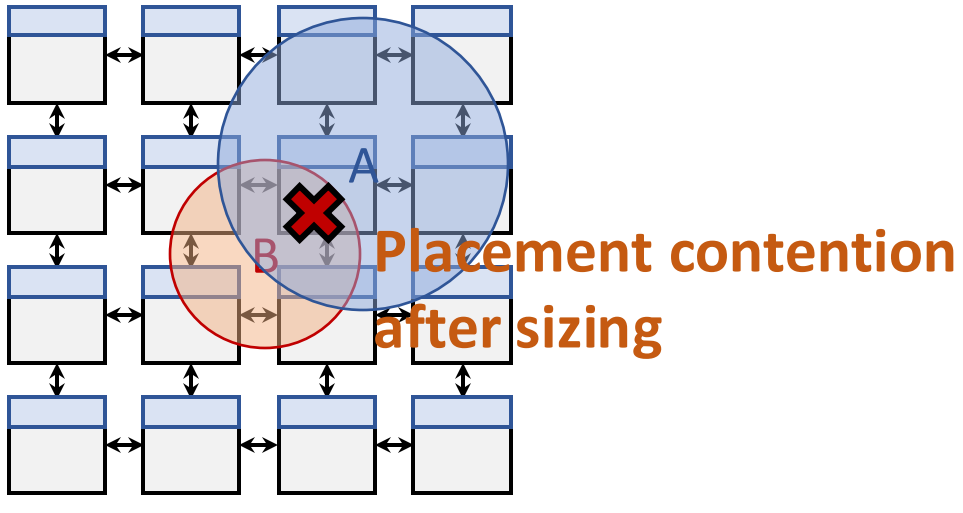
Software reconfigured

sid × unit ID	RShares	RRowBase	RGroups
0x1, Unit 0	8	0x0A00	0
0x1, Unit 1	6	0x0160	0
0x1, Unit 2	4	0x0C00	1
0x1, Unit 3	2	0x1000	1

Cache Configuration

- Goal
  - Host determines stream placement by adjusting per-stream replication group (RGroups) and space partitioning across units within each group (RShares)
  - We sample miss rates and configure data placement periodically
- Prior NUCA work separates sizing and placement
  - For sizing, e.g., using Lookahead in Utility-Based Cache Partitioning [MICRO'06]
  - For placement, e.g., using "center-of-mass" in Jigsaw [PACT'13]
  - Separated sizing and placement results in suboptimal result

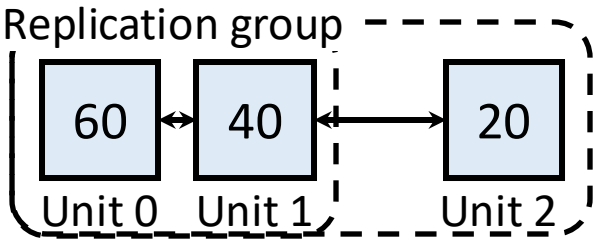




Cache Configuration

- We propose a new configuration algorithm
  - Simultaneous data sizing and placement
  - Flexible data replication
- Key Ideas
  - Compute the next steepest slope and immediately place it
  - Prefer all replication when NDP memory is sufficient
  - If NDP memory full, considering reallocation towards higher utility

Extending the current replication group

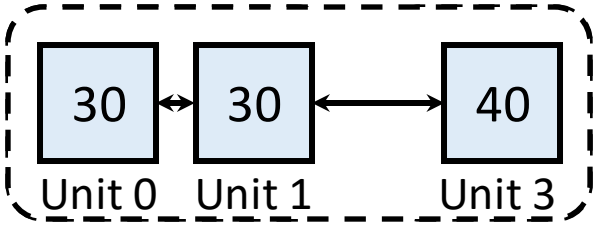


$$\text{Utility} = U_0 + U_1 = (60 + 40k_{01}) + (40 + 60k_{10})$$

$$\text{New utility} = U'_0 + U'_1 = (60 + 40k_{01} + 20k_{02}) + (40 + 60k_{10} + 20k_{12})$$

Relates to interconnect overheads

Merging two existing groups



Recalculate the utility accordingly
Choose the scheme with higher utility

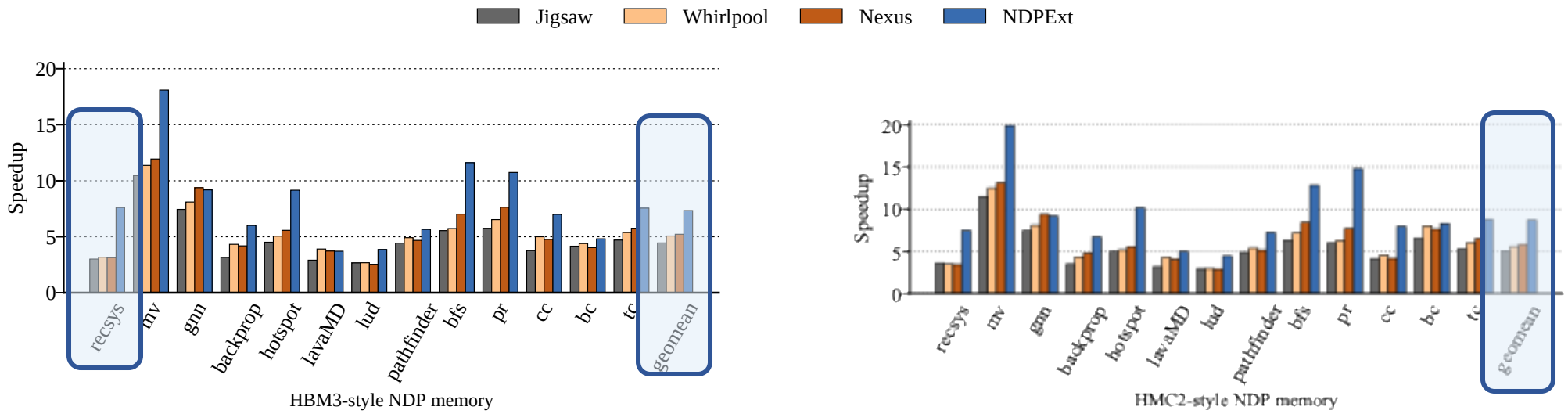
Methodology

- Simulated platform
  - 128 NDP cores with 8 stacks
  - Both HBM3-style and HMC-style NDP evaluated

NDP system	4 × 2 inter-stack mesh, 16 NDP cores per stack; 128 NDP cores in total
NDP core L1I L1D	2 GHz, in-order 2-way, 32 kB per core, 64 B cachelines, LRU 4-way, 64 kB per core, 64 B cachelines, LRU
NDP HBM	16 GB HBM 3.0, 1600 MHz, 256 MB/unit; RCD-CAS-RP: 24-24-24; RD/WR: 1.7 pJ/bit, ACT/PRE: 0.6 nJ
NDP HMC	16 GB HMC 2.1, 1250 MHz, 256 MB/unit; RCD-CAS-RP: 14-14-14;
Extended memory	DDR5-4800, 4 channels × 2 ranks × 16 banks; RCD-CAS-RP: 40-40-40; RD/WR: 3.2 pJ/bit, ACT/PRE: 3.3 nJ
Intra-stack network Inter-stack network CXL link	128-bit link, 1.5 ns/hop [65], [69]; 0.4 pJ/bit 32 GB/s per dir., 10 ns/hop [20], [22], [69]; 4 pJ/bit 16-lane; 200 ns link latency; 11.4 pJ/bit

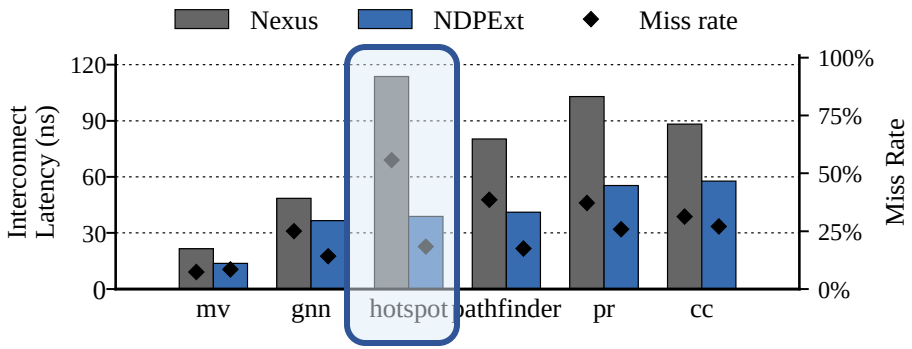
- Evaluated designs
  - Baseline: non-NDP host execution
  - NUCA design: Jigsaw
  - NUCA + data classification: Whirlpool
  - NUCA + global data replication: Nexus
- Workloads
  - Tensor workloads: e.g., DLRM recommendation system
  - Rodinia workloads
  - Graph computing benchmark GAP

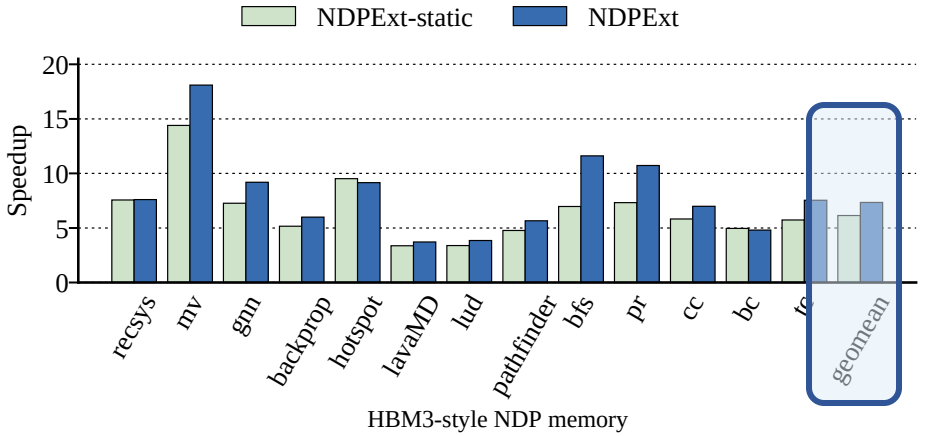
Evaluation



- HBM-style comparison
  - NDPExt outperforms SOTA by 1.41x on average and up to 2.43x over recsys
- Similar result trends in HMC-style NDP systems

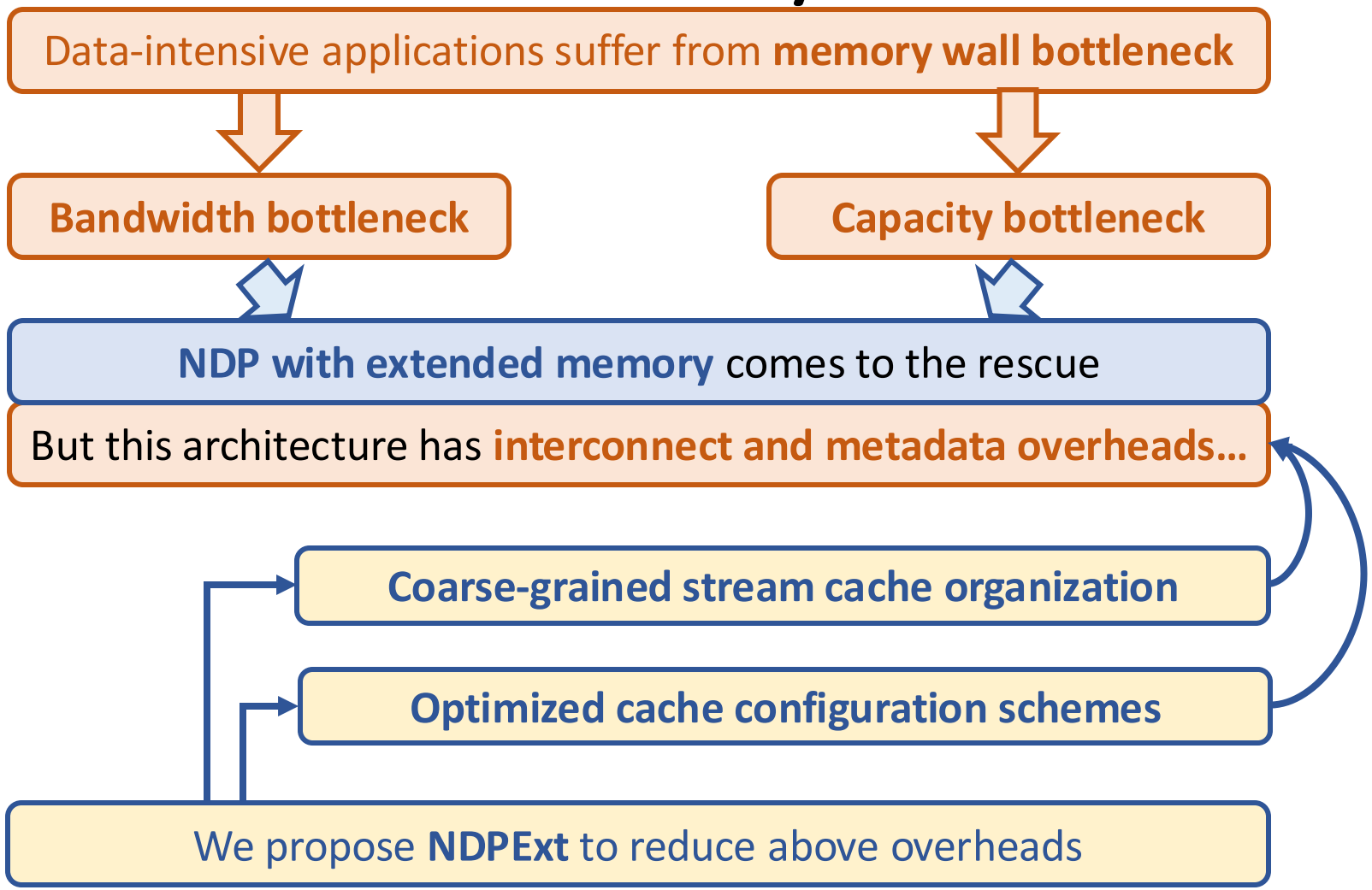
Evaluation





- NDPExt gains speedup for two reasons
  - Prior work shows low metadata hit rate for irregular graph applications, while NDPExt avoids metadata overheads using stream programming
  - Compared to a heuristic static placement (NDPExt-static), NDPExt reduces interconnect overheads with better data placement and flexible data replication

Summary



Thank you!