Revisiting Pebble Games for Modeling and Efficient Use of Disaggregated Memory Systems

Anusha Devulapally*^,

Mahantesh Halappanavar^, Bruno Jacob^,

Vijaykrishnan Narayanan*, Andres Marquez^

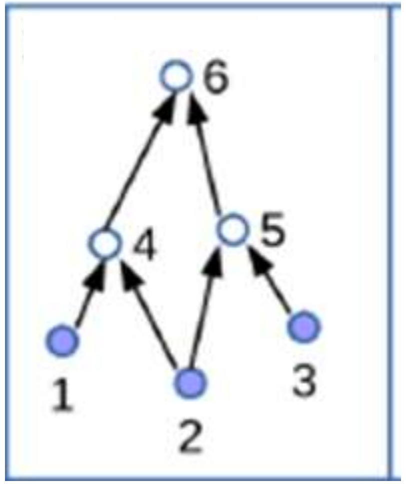
* The Pennsylvania State University,

^ Pacific Northwest National Laboratory

PNNL is operated by Battelle for the U.S. Department of Energy

Hong-Kung Red-Blue Pebble Game

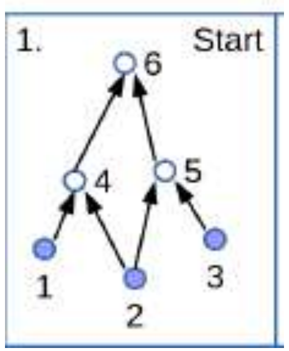
- Abstract model – to understand I/O complexity on 2-level memory hierarchy.
- Pebble Types:
  - **Red** – fast memory capacity ‘r’ (limited)
  - **Blue** – slow memory with unlimited capacity
- Input: DAG, red pebbles ‘r’
- Goal:
  - Compute every node in a DAG while moving few pebbles back and forth b/w red and blue
  - Capture the trade-off between fast-memory size (cache) and total I/O (loads/stores)
  - Prove lower bounds on the number of memory transfers any algorithm must perform



An example with 3 red pebbles and unlimited blue pebbles

Jia-Wei Hong and H. T. Kung. 1981. “I/O complexity: The red-blue pebble game.” In Proceedings of the *Thirteenth Annual ACM Symposium on Theory of Computing* (Milwaukee, Wisconsin, USA) (STOC ’81). Association for Computing Machinery, New York, NY, USA, 326–333. https://doi.org/10.1145/800076.802486

Red-Blue Pebble Game with an Example

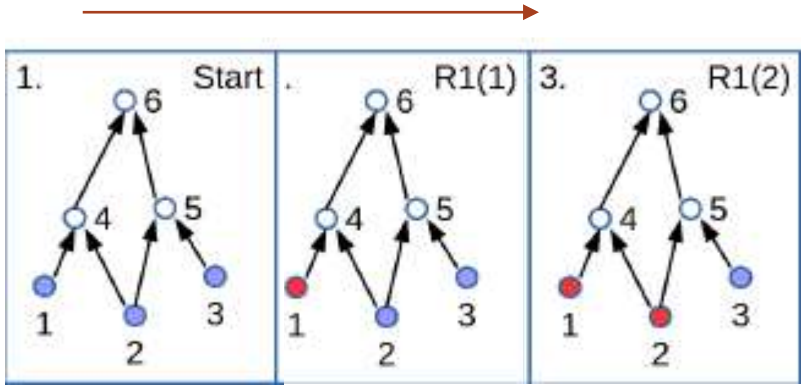


Input: Computational DAGs, red pebbles (3 in this case, unlimited blue pebbles
Output: how many moves between blue <->red (data movement)

Rules:

- **Input:** place a blue pebble on any source (input) node

Red-Blue Pebble Game with an Example

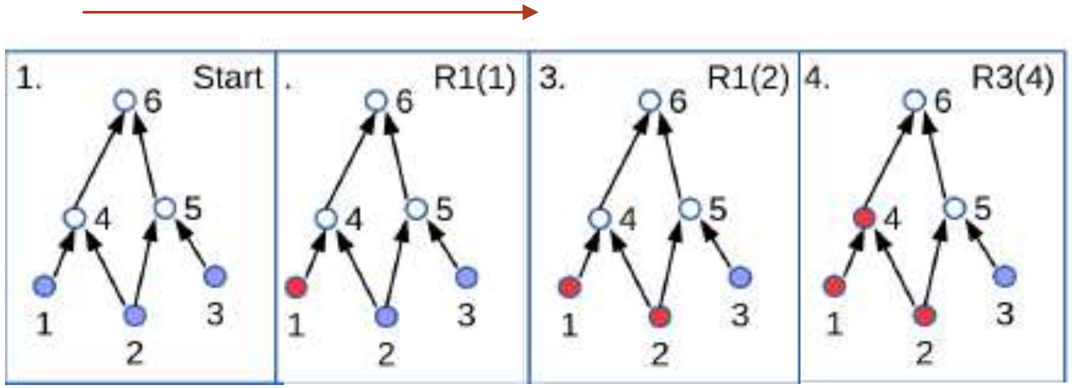


Input: Computational DAGs, red pebbles (3 in this case, unlimited blue pebbles
Output: how many moves between blue <->red (data movement)

Rules:

- **Input:** place a blue pebble on any source (input) node
- **Load:** if a vertex has a blue pebble, you may place a red pebble on it

Red-Blue Pebble Game with an Example

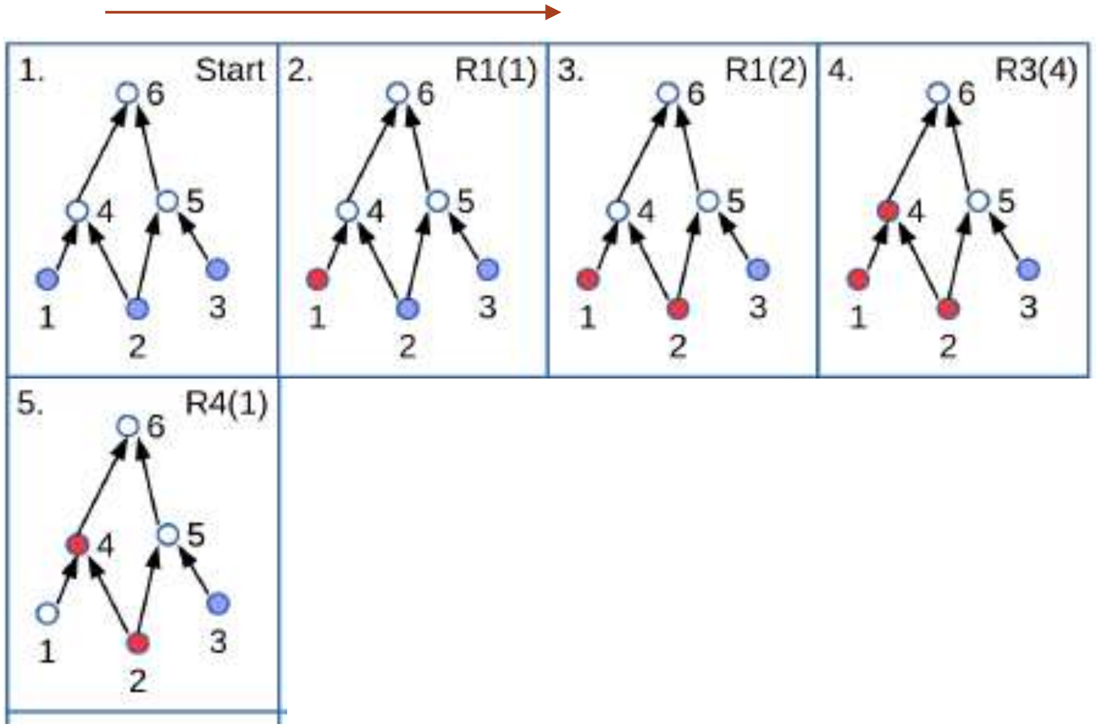


Input: Computational DAGs, red pebbles (3 in this case, unlimited blue pebbles
Output: how many moves between blue <->red (data movement)

Rules:

- **Input:** place a blue pebble on any source (input) node
- **Load:** if a vertex has a blue pebble, you may place a red pebble on it
- **Compute:** if all immediate predecessors carry red pebbles, place a red pebble on that vertex

Red-Blue Pebble Game with an Example

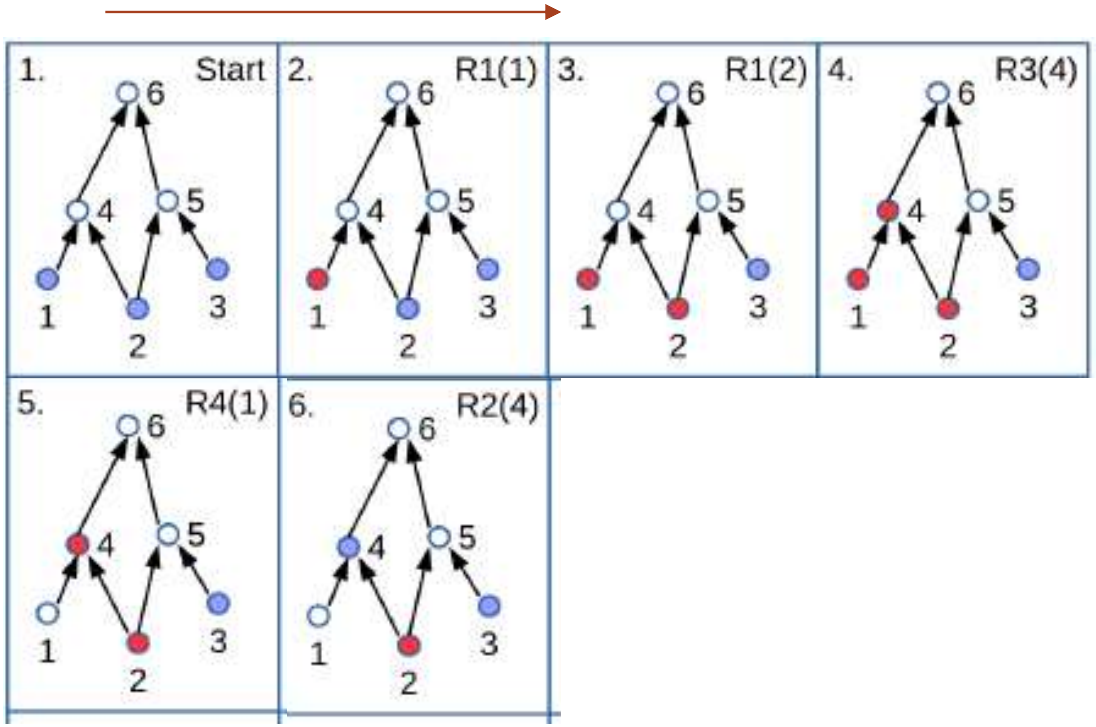


Input: Computational DAGs, red pebbles (3 in this case, unlimited blue pebbles
Output: how many moves between blue <->red (data movement)

Rules:

- **Input:** place a blue pebble on any source (input) node
- **Load:** if a vertex has a blue pebble, you may place a red pebble on it
- **Compute:** if all immediate predecessors carry red pebbles, place a red pebble on that vertex
- **Evict:** remove any red or blue pebble at any time

Red-Blue Pebble Game with an Example



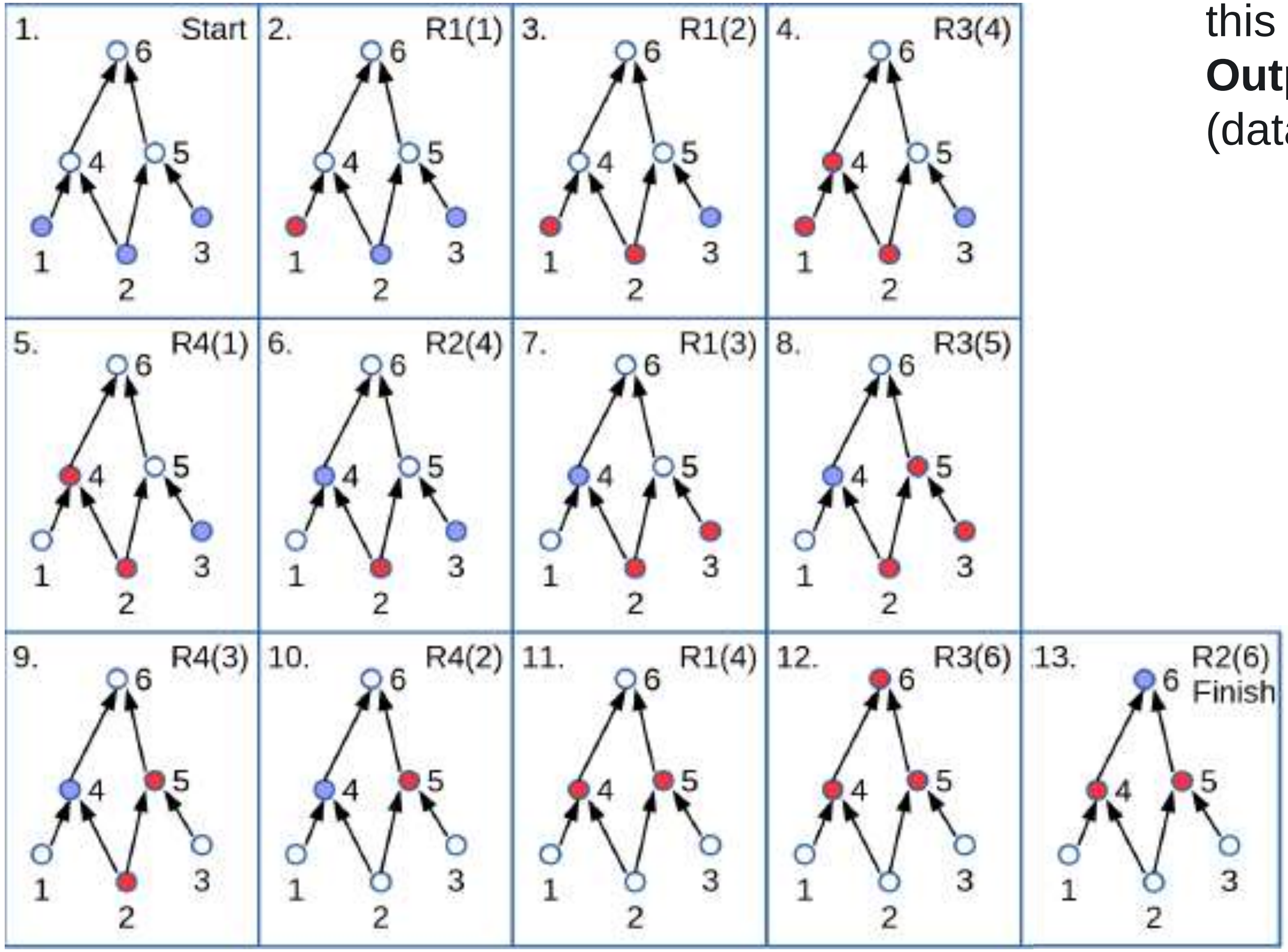
Input: Computational DAGs, red pebbles (3 in this case, unlimited blue pebbles

Output: how many moves between blue <->red (data movement)

Rules:

- **Input:** place a blue pebble on any source (input) node
- **Load:** if a vertex has a blue pebble, you may place a red pebble on it
- **Compute:** if all immediate predecessors carry red pebbles, place a red pebble on that vertex
- **Evict:** remove any red or blue pebble at any time
- **Store:** convert a red pebble back to blue on any pebbled vertex

Red-Blue Pebble Game with an Example



Input: Computational DAGs, red pebbles (3 in this case, unlimited blue pebbles
Output: how many moves between blue <->red (data movement)

Rules:

- **Input:** place a blue pebble on any source (input) node
- **Load:** if a vertex has a blue pebble, you may place a red pebble on it
- **Compute:** if all immediate predecessors carry red pebbles, place a red pebble on that vertex
- **Evict:** remove any red or blue pebble at any time
- **Store:** convert a red pebble back to blue on any pebbled vertex
- **Termination:** The game ends when the sink node is blue.

Introduction & Motivation

- Disaggregated memory systems separate on-chip cache, fabric-attached pool, and storage, creating complex data-movement trade-offs.
- Pebble games abstract these tiers with colored pebbles and rules that mirror load, compute, and store operations.
- This framework lets us predict I/O costs and derive bounds without exhaustive benchmarking.
- Extending to a red-orange-blue model captures fabric-attached memory explicitly and supports dynamic budget changes under SLAs.
- Enable system design for disaggregated memory systems for given set of applications.

Origin: The Black Pebble Game (Sethi'72)

- Goal: Model space-bounded computation by minimizing the maximum number of pebbles (i.e. registers) placed at once.
- Pebble Rules:
  - Place a pebble on any source (leaf) node at any time.
  - To pebble a non-leaf node, all its immediate predecessors must currently carry pebbles
  - Remove (evict) a pebble from any node at any time.
- By sequencing these rules over a DAG, we “compute” every node while reusing pebbles to keep the peak count low.
- Key Applications
  • Register allocation in compilers (minimize actual CPU registers)
  • VLSI/circuit-layout resource scheduling
  • Analysis of propositional proof complexity
  • Design of memory-hard functions in cryptography

Sethi, R. (1973, April). Complete register allocation problems. In Proceedings of the fifth annual ACM symposium on Theory of computing (pp. 182-195).

Ext: Multi-Level & Multiprocessor Games

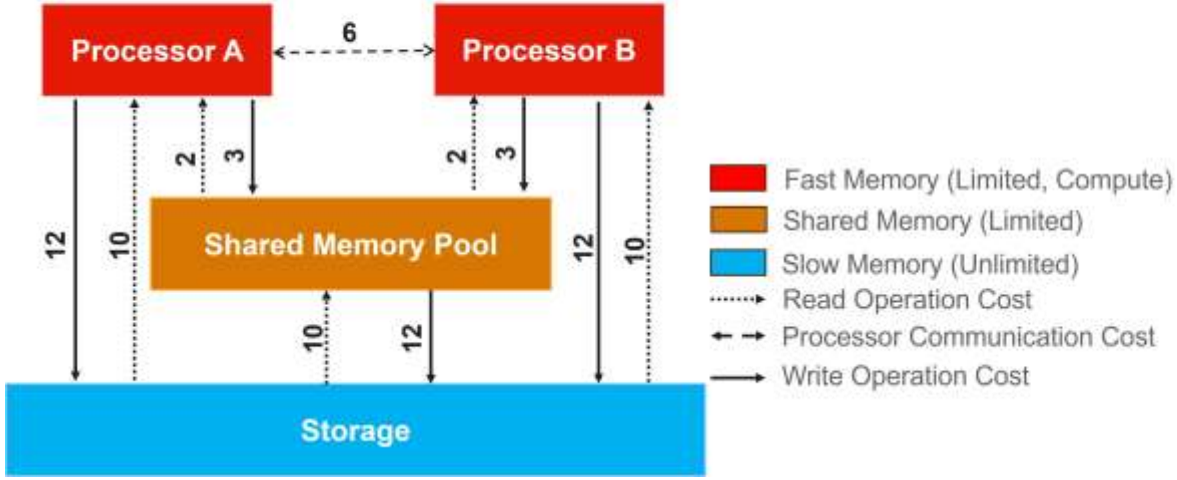
- Savage (1995)
  - k-level Memory Hierarchies
  - S-Partition across L cache levels → matching I/O lower bounds
  - Static budgets, single-processor model
- Elango et al. (2014)
  - Parallel Red-Blue-White Game
  - Multi-node, multi-core with hierarchical caches & interconnects
  - Lower bounds on data movement for parallel CDAGs
- Liu et al. Survey (2020)
  - Comprehensive treatment of sequential & parallel, red vs. black-white games
  - Applications from register allocation to I/O modeling
- Kwasniewski et al. (2021)
  - Greedy S-Partition Strategies
  - Tight sequential & parallel I/O bounds for matrix–matrix multiplication
  - Empirical speedups in HPC kernels
- Böhnlein et al. (2024)
  - Multiprocessor Red-Blue Game with Greedy Heuristic
  - Captures compute ↔ communication ↔ memory trade-offs
  - Static per-processor budgets, no shared-pool tier

Gaps:
- Lack of empirical evaluations
- No explicit “shared-pool” (fabric/CXL) tier
- Static, pre-provisioned memory budgets only
- Limited or no support for dynamic, multi-tenant allocation

Multiple Processor Red-Orange-Blue Game

Rules:

- **Remote-Get:** On processor p, if a vertex already has a **blue** pebble, you may place an **orange** pebble there (fetch from pool) or a **red** pebble of **another** shade R□ (n≠p), modelling an inter-processor message
- **Data-Movement:** On processor p, if a vertex has any **blue**, **orange**, or a **red** pebble of a **different** shade, you may load it into your local cache by placing your **red** pebble there
- **Remote-Put:** Convert your **red** pebble to an **orange** pebble (write back to pool) or to another-shade **red** (hand off to another CPU)



Multi-Level multi-processor memory hierarchy and data movement costs in a disaggregated system.

We assume costs:
red ↔ blue = highest cost
blue ↔ orange = medium cost
orange ↔ red = lowest cost

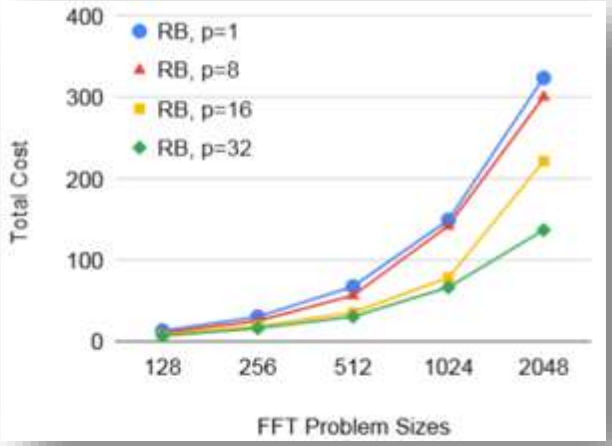
We also assume different costs for Read versus Write tasks. These are only representative numbers for empirical evaluation.

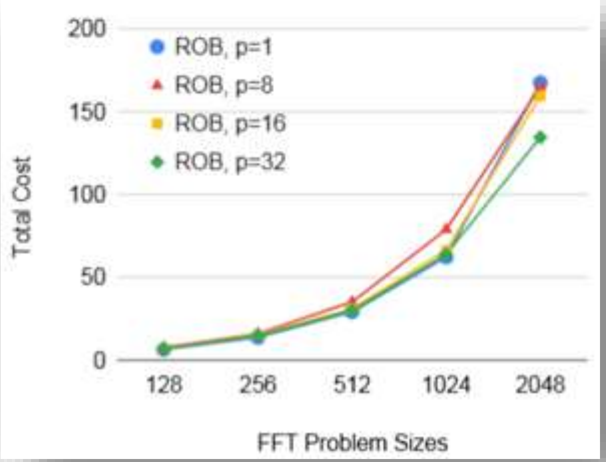
Experimental Setup: Applications & Metrices

- Applications
  - Fast Fourier Transform (FFT) (sizes 128…2048)
  - ResNet-Coarse (5 variants of resnet models)
- Metrices
  - Total number of moves
  - Total Cost
- Experiments:
  - **Red-Blue Greedy** (baseline with minimum red pebbles and 2x minimum red pebbles)
  - **Red-Orange-Blue Greedy:** Uses minimum red pebbles + minimum orange multiples (64×…512× minimum red)
  - Varying Processors: p – 1 to 32
- Analysis:
  - Per-processor memory constant (if no. of red pebbles is r, each processor will have 'r' red pebbles)
  - Total memory constant (if no. of red pebbles is r, then r is divided across the processors equally)
- Note:
  - Theoretical minimum red: which is max in-degree+1 for DAG
  - **For experiments**, minimum red pebbles in the experimental number via trial &error as algorithms doesn't give optimal output. (it is much higher than theoretical minimum red)

Results-1: Fast Fourier Transform (FFT)

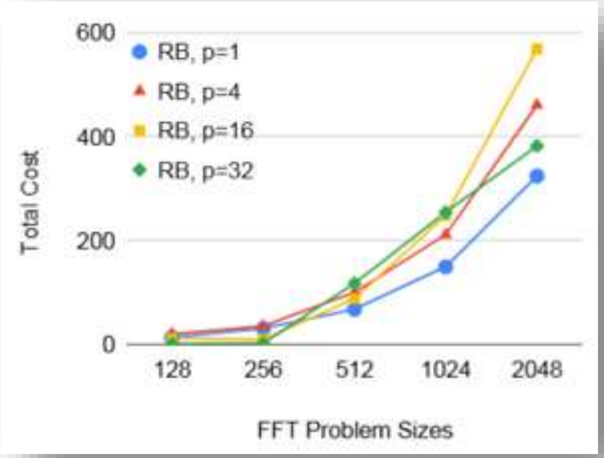
Analysis -1 (Per-processor Memory Constant)

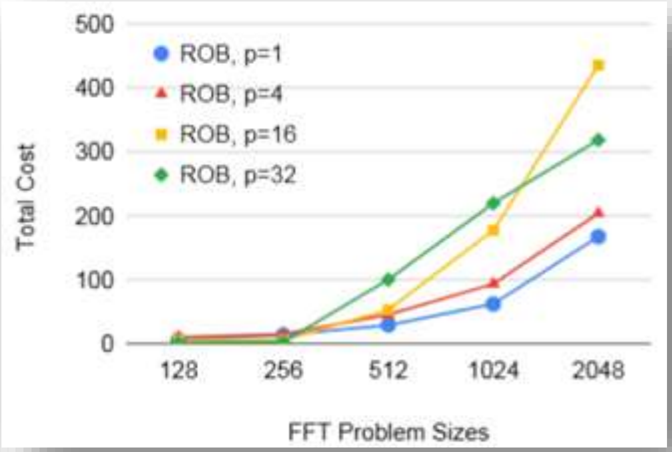




- RB cost drops by 56% from processor 1 to 32 for larger problem size (n)
- ROB cost drops by 21% from processor for n.
- ROB is 47% lower than RB at p=1 and 4% lower at p=32

Analysis -2 (Total Memory Constant)

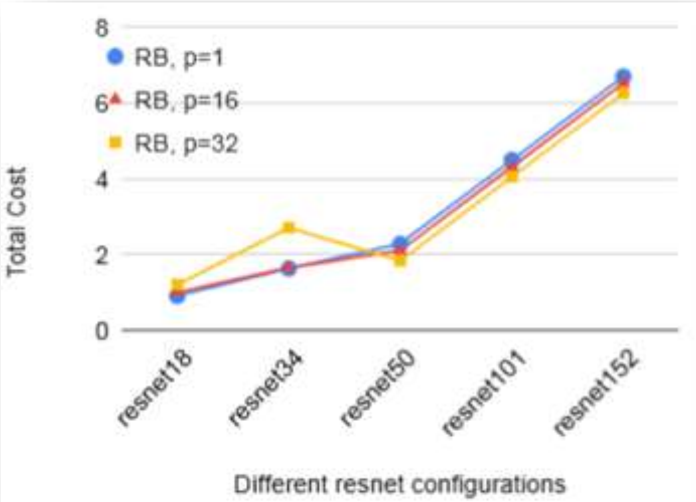


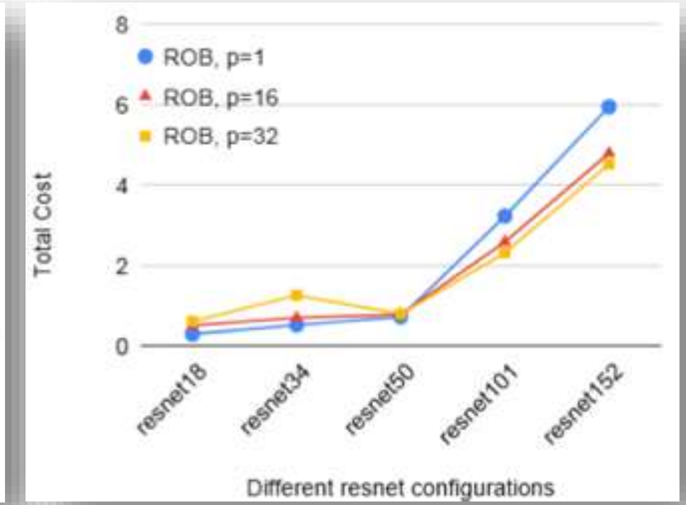


- RB cost increases by 19% from processor 1 to 32 for larger problem size (n)
- ROB cost increases by 88% from processor for n. (cost raises at p=16)
- ROB is 47% lower than RB at p=1 and 16% lower at p=32

Results-2: ResNet-Coarse

Analysis -1 (Per-processor Memory Constant)





- RB cost increases by 18% from processor 1 to 32 for larger problem size (n)
- ROB cost decreases by 20% from processor for n. (cost raises at p=16)
- ROB is 12% lower than RB at p=1 and 40% lower at p=32

Analysis -2 (Total Memory Constant)

The number of red pebbles for resnet-coarse models is just 3, so we cannot divide across processors.

Limits of Static Allocation

- **Fixed working-set budget** requires accurate estimate of peak footprint; misestimation leads to thrashing or wasted memory
- **No adaptability:** cannot respond to dynamic workload spikes, phase changes, or idle periods
- **Poor multi-tenant fairness:** static quotas ignore job priority and SLA requirements, causing performance imbalance
- **Inefficient sharing:** dedicated allocations block reusable orange memory, while idle resources remain unused

Dynamic Single-Tenant Extension (DyPeG)

- **Runtime Budget:** red & orange allocations can grow or shrink in response to workload needs and SLA guarantees
- **Prefetch Rules:** when budget increases, fetch data from blue→orange or orange→red proactively
- **Eviction Rule:** on budget drop, dead red pebbles spill to orange or blue to free space
- **Adaptive I/O Control:** rules D1–D9 govern Input, Remote-Get/Put, Data-Move, Compute, Eviction, Output, and Termination under dynamic quotas
- **SLA Enforcement:** ensures minimum red/orange reservations per job, allowing predictable performance while adapting to phase changes

Summary & Future Work

- **Contributions**
  - Formulated a three-tier Red-Orange-Blue pebble game (ROBP) and its dynamic extension (DyPeG) for modeling CXL-style disaggregated memory
  - Developed and analyzed both greedy heuristics schedulers under static and dynamic budgets
- **Empirical Findings**
  - Multi-processors: smooth scaling and **50%+** savings for $p \le 8$
- **Key Takeaways**
  - Static budgets risk inefficiency; pooling (orange) bridges local vs. remote but needs right sizing
  - Greedy is lightweight and near-optimal
- **Future Work**
  - Build an RL-driven allocator for dynamic orange-pool management
  - Validation on a hardware testbed with real CXL devices with diverse applications
  - **Extend DyPeG to multi-tenant scenarios** with SLA-aware dynamic memory sharing.

Acknowledgment

Pacific Northwest National Laboratory

PennState

AMAIS
ADVANCED MEMORY TO SUPPORT ARTIFICIAL INTELLIGENCE (AI) FOR SCIENCE
@PNNL

Semiconductor Research Corporation

Thank you