Department Of Computer Science And Engineering
Indian Institute Of Technology Guwahati

Optimizing Memory Performance In Multi-Core Systems Through Data Placement

Akshay Bhosale
(234101006)

Under the Guidance of
Dr. Aryabartta Sahu

Contents

Introduction

- ➢ Efficient memory performance is essential for multi-core systems
- ➢ Modern applications place heavy demands on memory
- ➢ Important to understand how cores interact with memory
- ➢ Involves studying access behaviors, and the hierarchical structure of DRAM (Dynamic Random Access Memory)

Introduction

Address Mapping in DRAM

Virtual to Physical Address Mapping

- ➢ When a program requests data it uses a virtual address
- ➢ This translation uses page tables

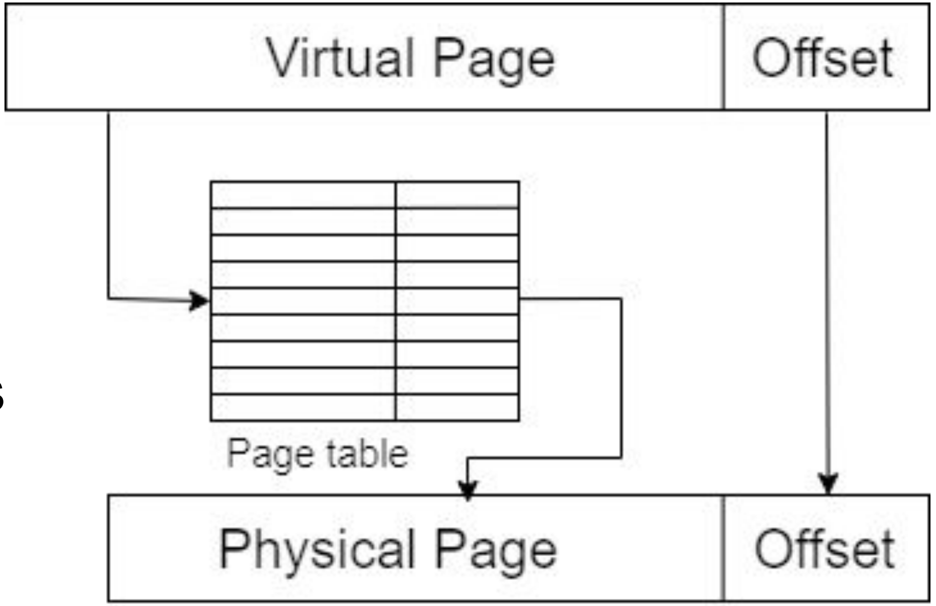


Fig. 1 Virtual Address to Physical Address translation

Introduction

Address Mapping in DRAM

Physical Address To Hardware Address Mapping

Physical address is divided into components that specify the channel, rank, bank, row, and column

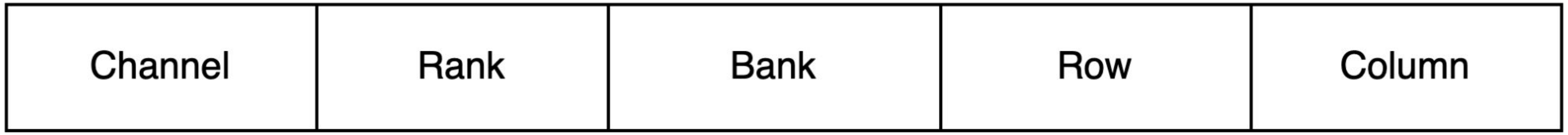


Fig. 2 Physical Address breakdown

Introduction

DRAM Organization

- ➢ Organizes its components in a structured hierarchy
- ➢ Channels, DIMM, ranks, Chips banks, rows, and columns
- ➢ Each bank contains a row buffer

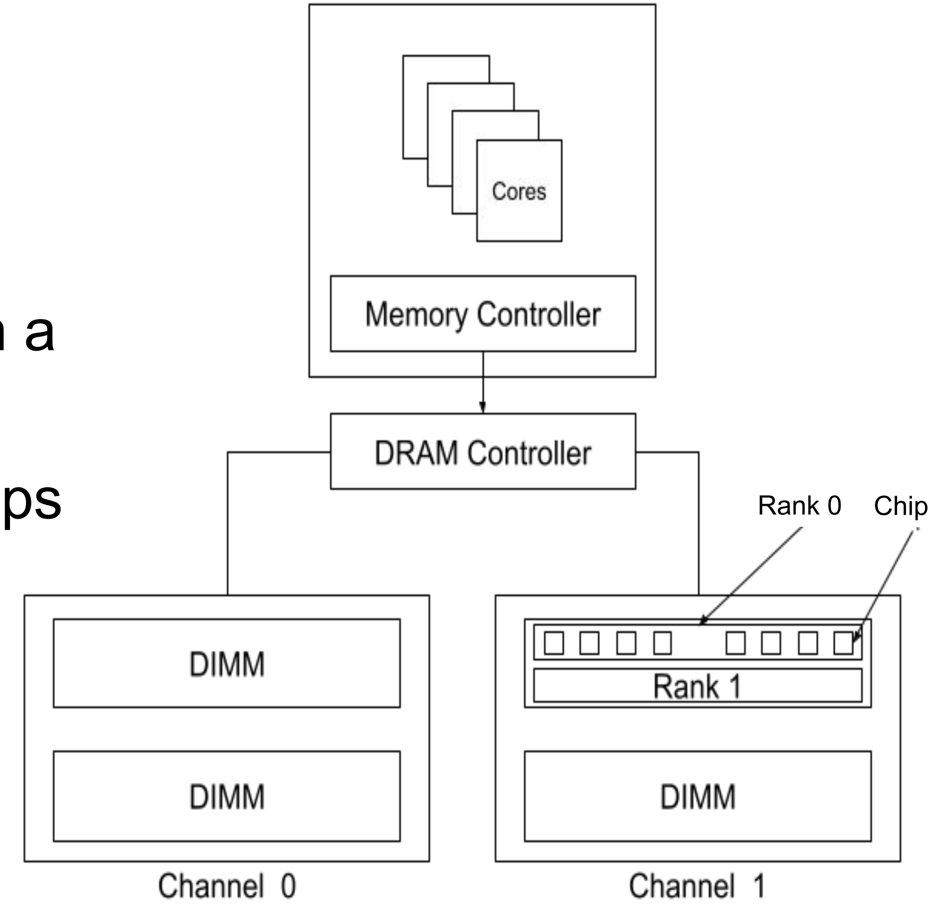


Fig. 3 High-level architecture of DRAM

Introduction

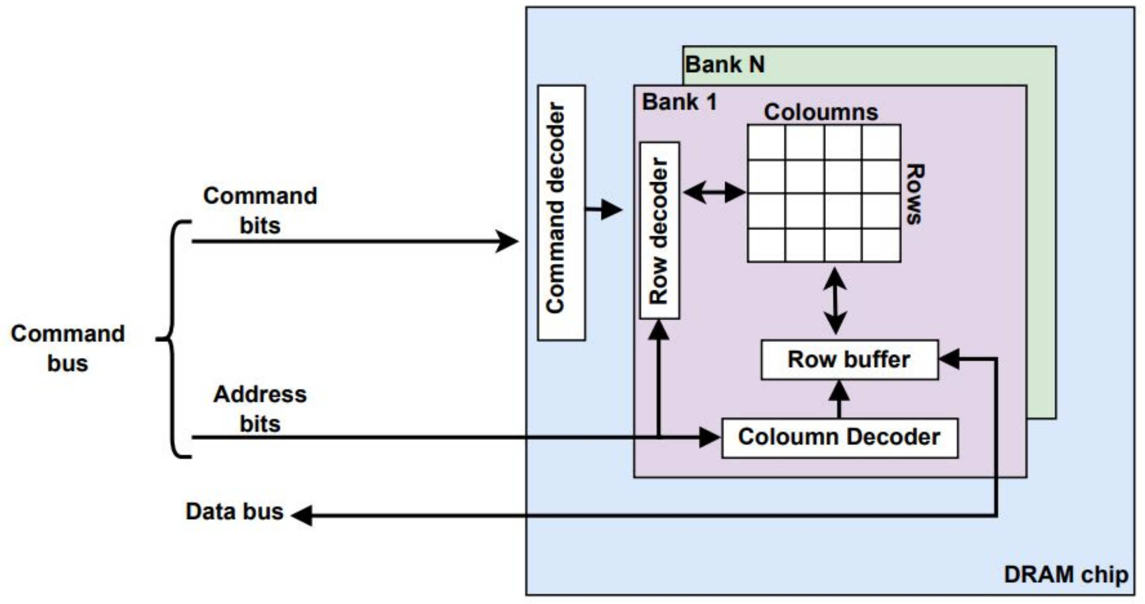


Fig. 4 DRAM Chip architecture

Introduction

Shared Memory

- ➢ All processors share a common physical memory space
- ➢ Two types
  - UMA (Uniform Memory Access)
  - NUMA (Non-Uniform Memory Access)
- ➢ NUMA Improves scalability over UMA

Introduction

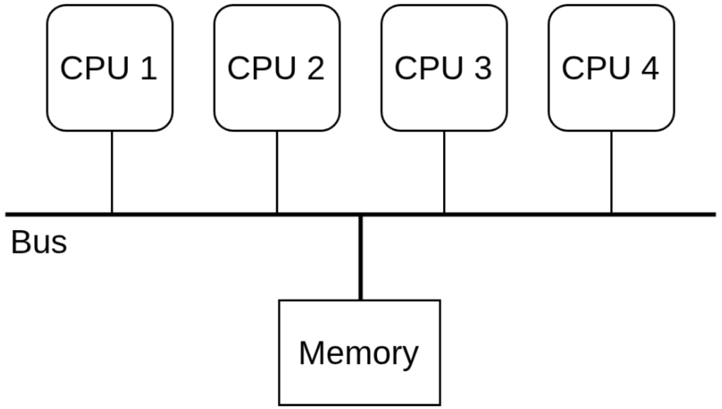
NUMA

- ➢ Memory is split across multiple memory channels
- ➢ Faster access to its local memory region
- ➢ Accessing remote memory is slower
- ➢ Latency varies depending on core-to-channel proximity

Introduction

Uniform Memory Access (UMA)

Non Uniform Memory Access (NUMA)



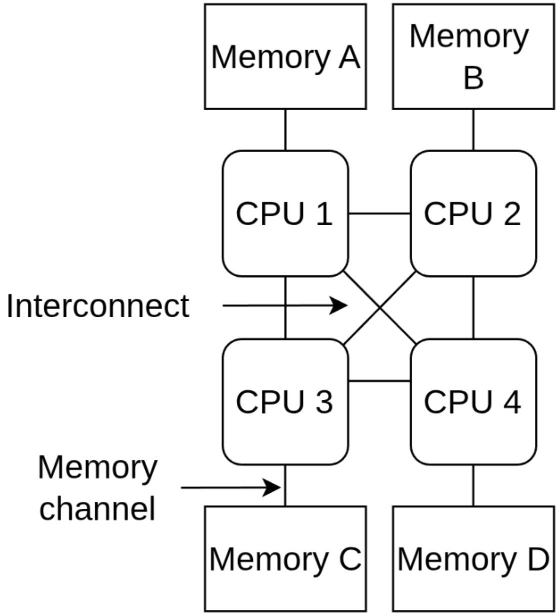


Fig. 5 Differences between a NUMA and UMA

Challenges

- ➢ Frequently accessed data may reside on high-latency channels
- ➢ Changing access patterns reduce placement effectiveness
- ➢ Overall number of remote accesses increases
- ➢ Results in suboptimal memory performance

Objectives

- ➢ Reduce latency by minimizing remote accesses
- ➢ Place data closer to the cores that access it most frequently
- ➢ Adapt to changing access patterns in real-time
- ➢ Design a solution that is lightweight, modular, and practical

Review Of Prior Work

- ➢ ML based memory and thread placement [1]
  - ○ Uses profiling
- ➢ Replicate page table across nodes in NUMA [2]
  - ○ targets metadata, not actual data
- ➢ Traffic-Aware Memory Placement in NUMA [3]
  - ○ Reduce congestion
  - ○ Collocation, replication and interleaving

Review Of Prior Work

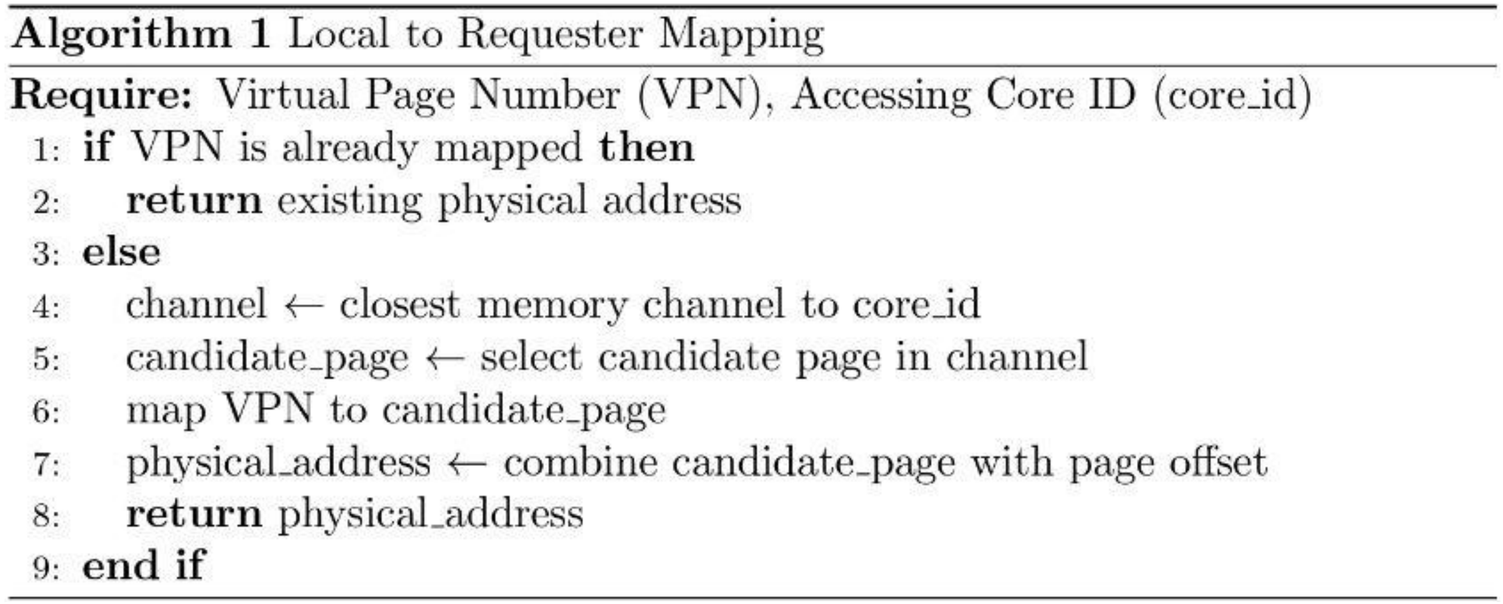
- ➢ Adaptive data placement in 3D-stacked DRAM [4]
  - ○ Uses a small SRAM buffer
- ➢ Dynamically remaps physical address bits using access patterns [5]
  - ○ Counters are used to monitor changes in bits
- ➢ Required hardware/kernel modifications or are tied to specific DRAM architectures

Proposed Approach

- ➢ Map VPN to PPN in the memory channel closest to its dominant accessing core
- ➢ Periodically track and migrate hot pages to better channels
- ➢ Use Local to Requester and Dynamic Migration mapping
- ➢ Entirely at the translation layer no hardware changes needed

Proposed Approach

Local to Requester

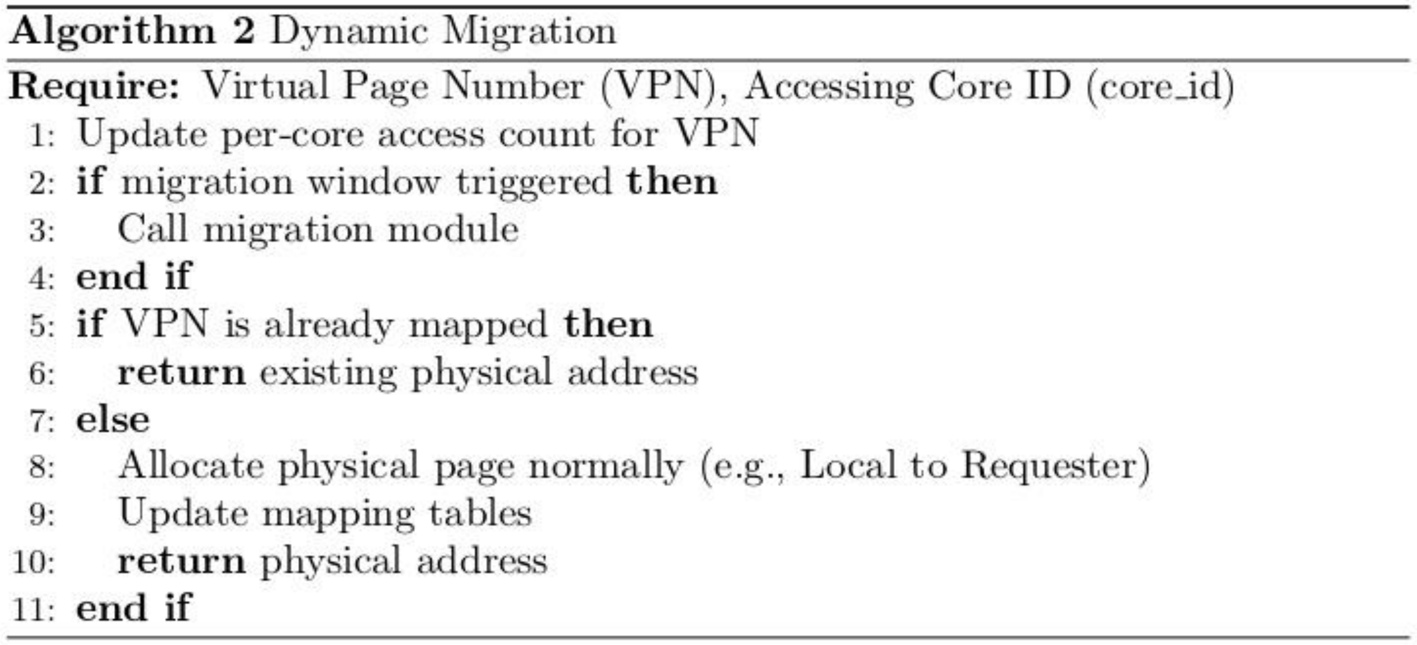


Algorithm 1 Local to Requester Mapping

```
Require: Virtual Page Number (VPN), Accessing Core ID (core_id)
1: if VPN is already mapped then
2:     return existing physical address
3: else
4:     channel ← closest memory channel to core_id
5:     candidate_page ← select candidate page in channel
6:     map VPN to candidate_page
7:     physical_address ← combine candidate_page with page offset
8:     return physical_address
9: end if
```

Proposed Approach

Dynamic Migration



Algorithm 2 Dynamic Migration

```
Require: Virtual Page Number (VPN), Accessing Core ID (core_id)
Update per-core access count for VPN
if migration window triggered then
    Call migration module
end if
if VPN is already mapped then
    return existing physical address
else
    Allocate physical page normally (e.g., Local to Requester)
    Update mapping tables
    return physical address
end if
```

Proposed Approach

Migration module

```
for all hot pages do
    Determine dominant core and access distribution
    if a single dominant core exists then
        best_channel $\leftarrow$ channel closest to dominant core
    else
        best_channel $\leftarrow$ channel minimizing total latency across major cores
    end if
    if page is already in best_channel or cooldown is active then
        continue
    end if
    Calculate estimated gain and cost of migration
    if gain $>$ cost then
        Migrate page to best_channel
        Update page table and cooldown timestamp
    end if
end for
```

Experiment Setup

System Configuration

- ➢ System with eight cores and eight DRAM channels.
- ➢ Each core is assigned to a nearby DRAM channel to capture NUMA memory access behavior
- ➢ DRAM type DDR4
- ➢ Simulated 1 billion instructions per benchmark to capture long-term behavior

Experiment Setup

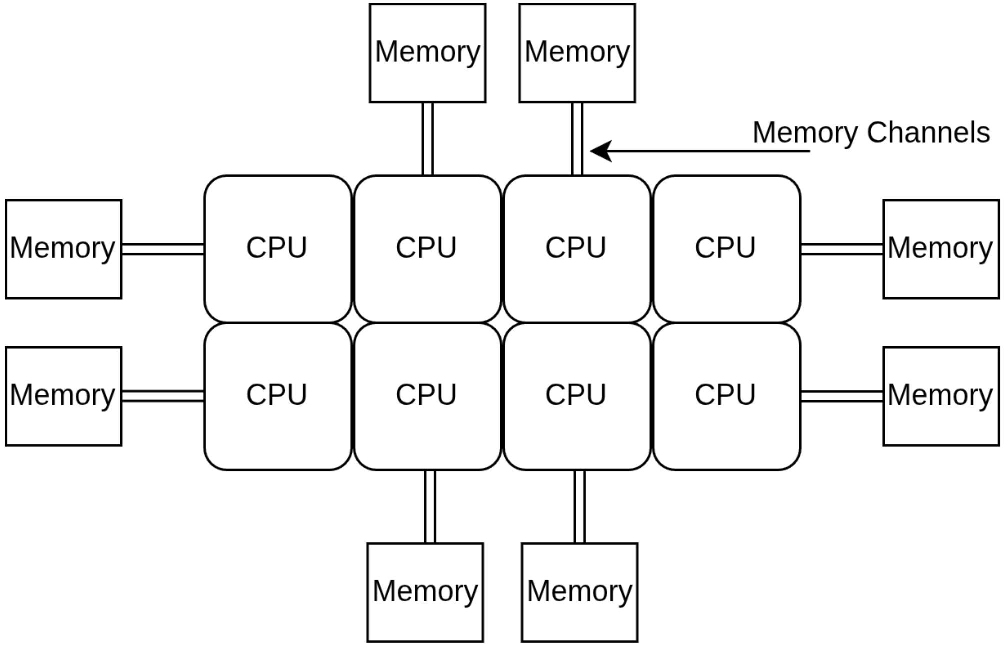


Fig. 6 Eight-core, eight-channel system used for simulation

Experiment Setup

Simulation Framework

- ➢ SniperSim performs simulations for multi-core and shared-memory workloads
- ➢ Ramulator2 is a detailed DRAM simulator modeling memory system behavior
- ➢ Benchmarks run on SniperSim to generate traces
- ➢ Traces are fed into Ramulator2 for simulation

Experiment Setup

Benchmarks

- ➢ Workloads from SPEC OMP2001
  - ○ Art, Equake and FFT
- ➢ Other parallel programs
  - ○ Gauss, PCNN and Matmul
- ➢ Covers diverse memory access patterns

Experiment Setup

Translation Policies Compared

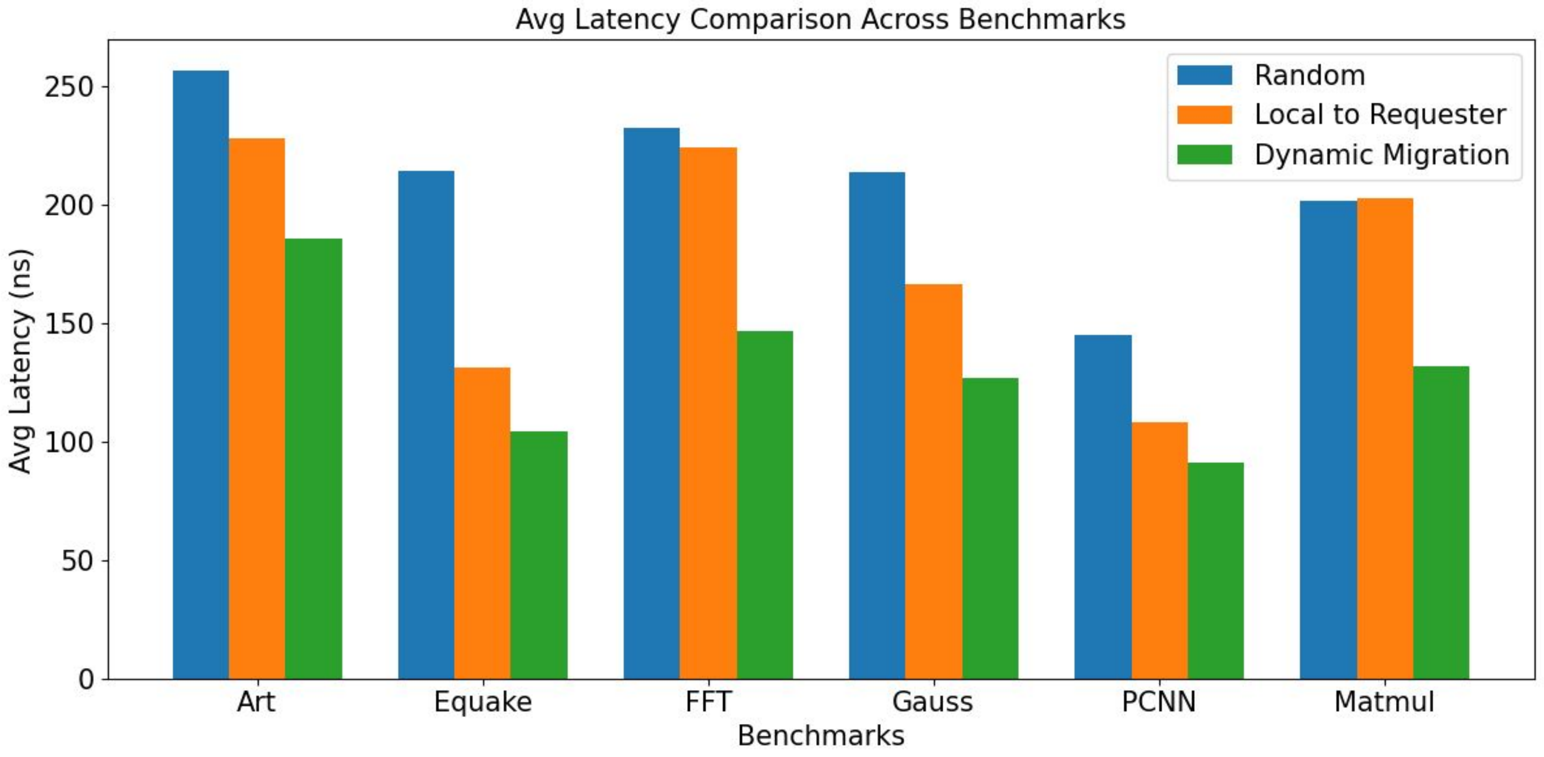
- ➢ Random Mapping
  - ○ Serves as a simple baseline for comparison
- ➢ Local to Requester
  - ○ Captures the effect of initial placement near first accessing core
- ➢ Dynamic Migration
  - ○ Tests the benefit of adaptive page migration
  - ○ Nine migration configs by varying key parameters

Experiment Setup

Dynamic Migration Configurations

- Parameters varied:
  - Window size
  - Hot page Threshold
  - Cooldown period
- Groups
  - Small windows: quick reaction
  - Large windows: more stable decisions

Results and Analysis

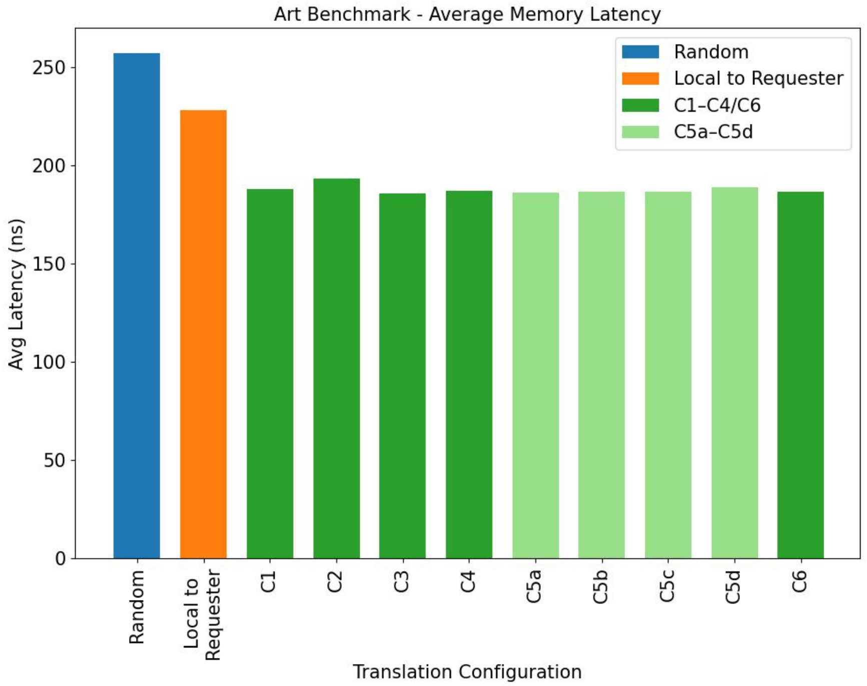


Results and Analysis

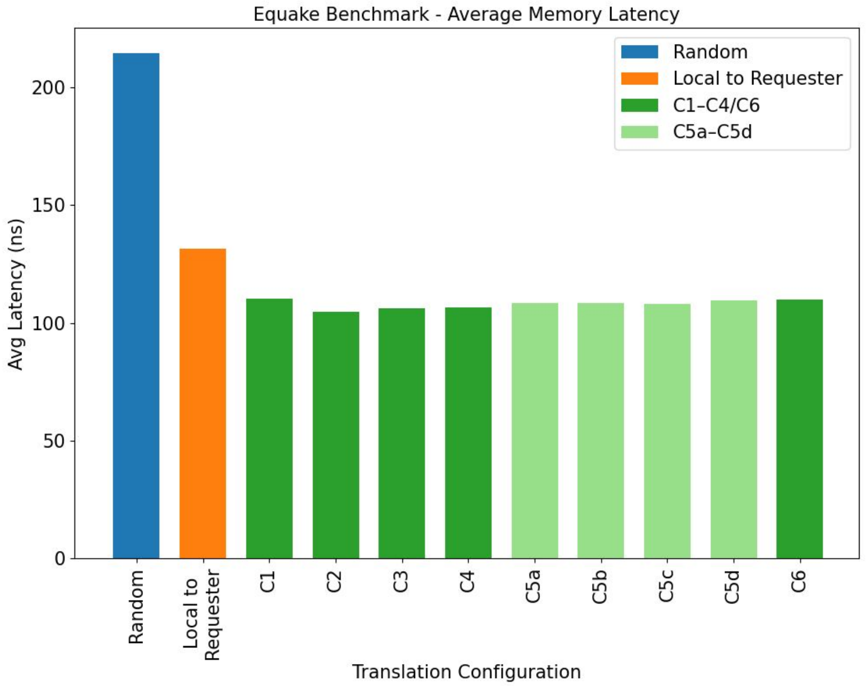
Translation Policies Compared

- ➢ Dynamic Migration consistently beats Random and Local to Requester
- ➢ Latency reduced by 27% to 50% across all benchmarks
- ➢ Biggest gains in Equake
- ➢ Shows benefit of adapting data placement at runtime

Results and Analysis

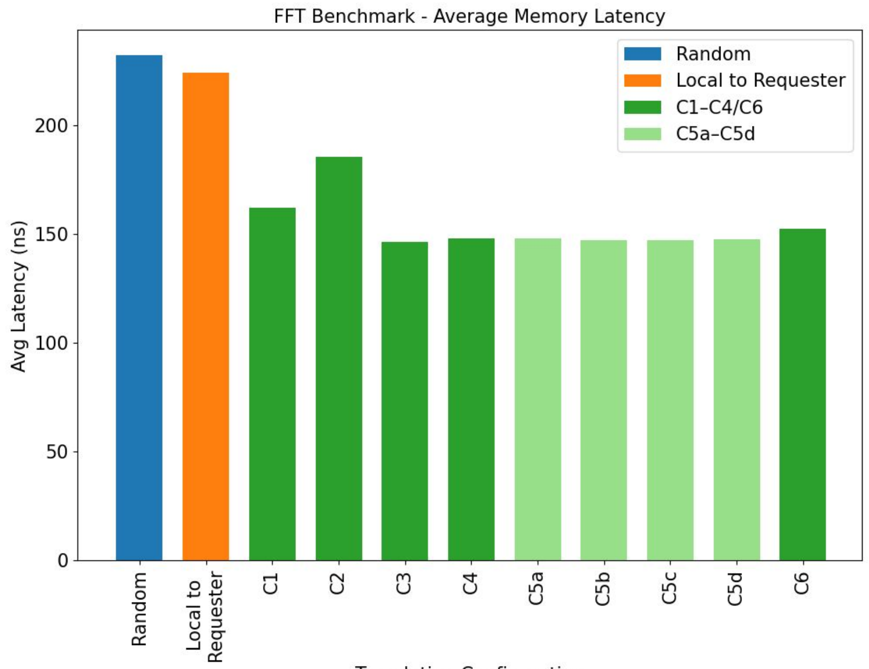


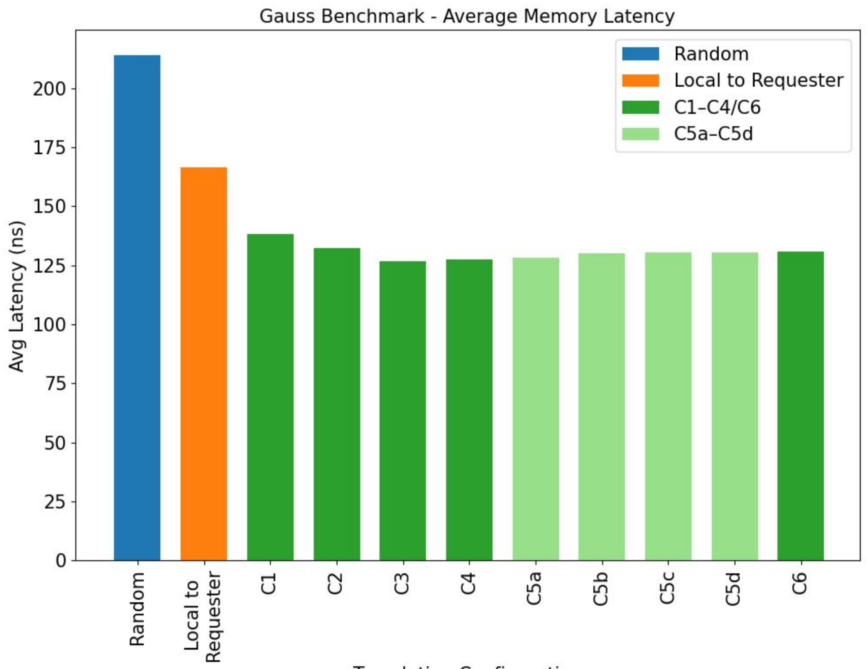
Art Benchmark Avg Memory Latency



Equake Benchmark Avg Memory Latency

Results and Analysis

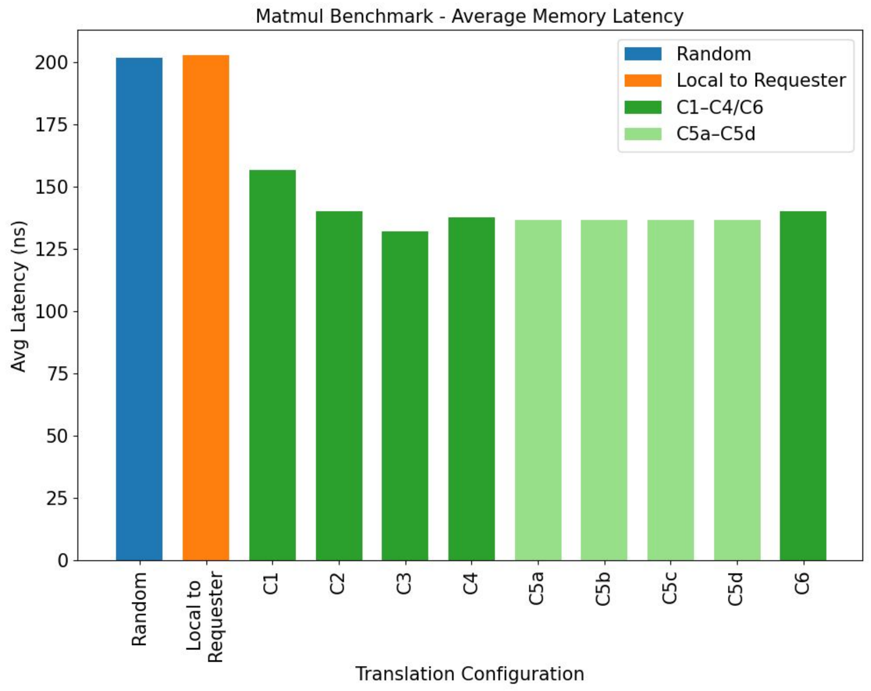




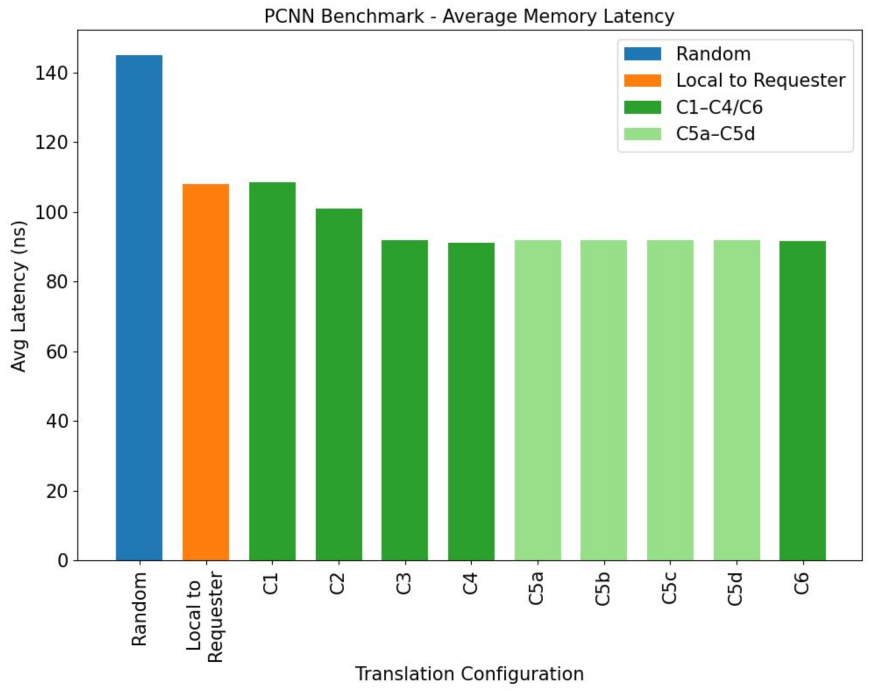
FFT Benchmark Avg Memory Latency

Gauss Benchmark Avg Memory Latency

Results and Analysis



Matmul Benchmark Avg Memory Latency



PCNN Benchmark Avg Memory Latency

Results and Analysis

Sensitivity of Key Parameters

- ➢ All configs outperform Random and Local to Requester
- ➢ No config dominates
- ➢ Ensures meaningful decisions regardless of parameter tuning
- ➢ Fine-tuning yields minimal effect, system is stable and avoids overfitting

Conclusion

- ➢ Lightweight runtime page placement at translation layer
- ➢ Dynamic Migration shows improvements ranging from 27% to over 50% in evaluated benchmark
- ➢ No hardware changes, practical and flexible
- ➢ Maintains performance regardless of different parameter settings

Future Work

- ➢ Implement page replication with coherence support
- ➢ Integrate approach with real OS for practical deployment
- ➢ Explore energy optimization
- ➢ Explore integration with caching techniques for DRAM

References

1. Denoyelle et al., “Data and Thread Placement in NUMA Architectures: A Statistical Learning Approach,”
 Proc. ICPP, 2019.
2. Panwar, “Efficient Virtual Memory and Page Table Management in Large NUMA and Virtualized Systems,” Ph.D. Thesis, IIT Kanpur, 2022.
3. Fedorova et al., “Carrefour: Traffic-Aware Memory Placement in Modern NUMA Systems,”
 ACM Queue, vol. 13, no. 7, 2015.
4. R. Pandey and A. Sahu, “Run-time adaptive data page mapping: A comparison with 3d-stacked dram cache,” Journal of Systems Architecture, vol. 110, p. 101798, 2020.
5. Qureshi et al., “DReAM: Dynamic Re-arrangement of Address Mapping to Improve Access Locality in DRAMs,” Proc. ISCA, 2010.
6. H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, , and O. Mutlu, “Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator,” 2023.
7. T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, “An evaluation of high-level mechanistic core models,” ACM Trans. Archit. Code Optim., vol. 11, Aug. 2014.

THANK YOU

Submitted By:
Akshay Bhosale (234101006)

Under the Guidance of:
Dr. Aryabartta Sahu (Associate Professor, CSE Dept, IIT Guwahati)

Indian Institute of Technology, Guwahati