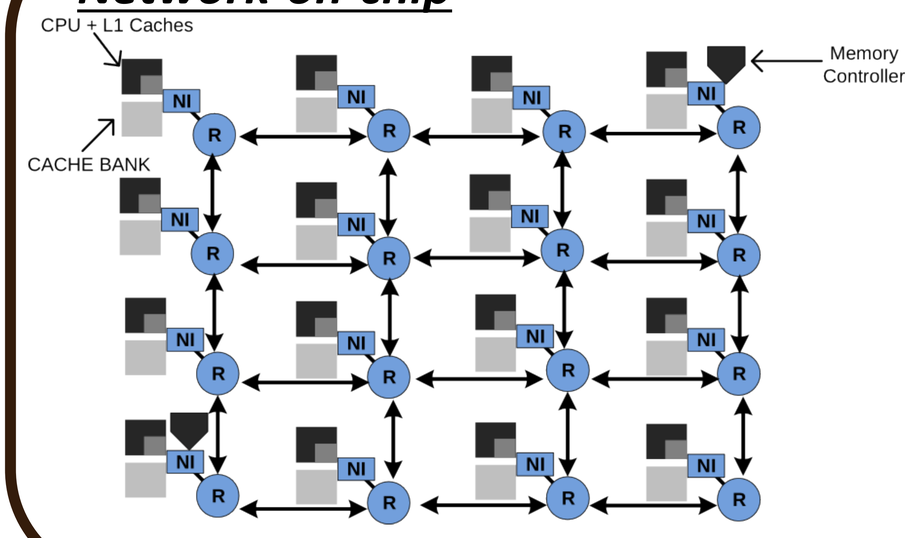
Application-to-Core Mapping Policies to Reduce Memory Interference in Multi-Core Systems

Reetuparna Das§ Rachata Ausavarungnirun† Onur Mutlu† Akhilesh Kumar‡ Mani Azimi‡

§University of Michigan †Carnegie Mellon University ‡Intel Labs

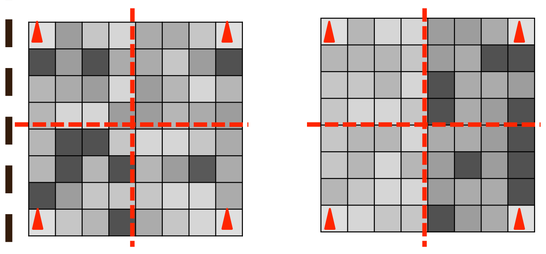
Background and Problems

Network-on-chip

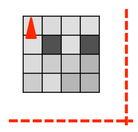


Problems

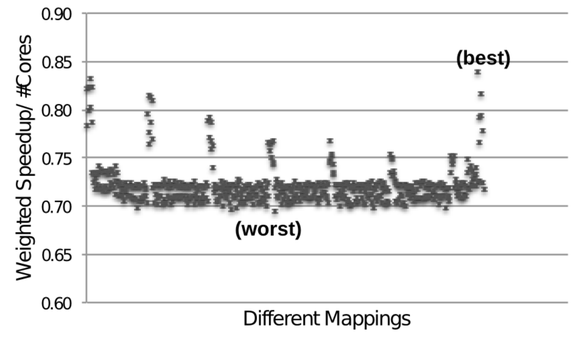
- Current operating systems are **unaware** of:
  - •On-chip interconnect topology
  - •Application interference characteristics



Unbalanced Network Load



Unaware of the location of the memory controller



System performance varies with different mappings

Our Solution

Key insights

1. Network and memory load are not balanced across the network
2. Overall performance **degrades** when applications that **interfere significantly** with each other get mapped to closeby cores
3. **Some applications benefit** significantly from being mapped close to **a shared resource**

Identifying Sensitive Applications

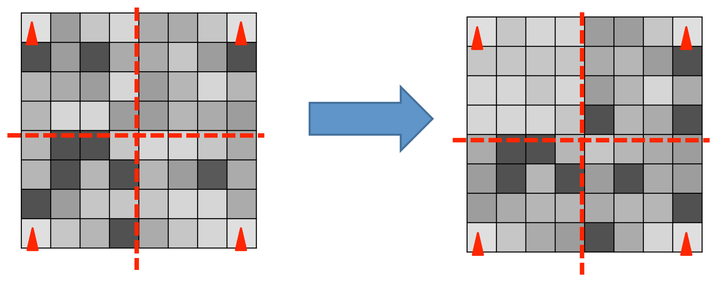
- Stall Time per Miss (STPM): average number of cycles a core is stalled because of a cache miss
→Applications with **high STPM are interference-sensitive**

- L1 Misses per Thousand Instruction (MPKI)
→Applications with **high MPKI are network-intensive**

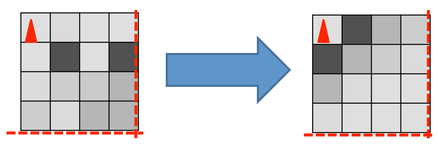
- Sensitive applications are applications with **high STPM and high MPKI**

Application-to-Core Mapping Policy

1. Clustering: A sub-network where applications mapped to a cluster predominantly access resources within that same cluster
2. Mapping policy across clusters:
   - **Equally divides the network load** among clusters
   - **Protects interference-sensitive applications** from others by **assigning them their own cluster**
3. Mapping policy within a cluster: Maps **network-intensive** and **interference-sensitive** applications **close to the memory controller**
4. **Dynamically migrate** applications between cores



Balanced Mapping with Reduced Interference



Radial Inter-cluster Mapping

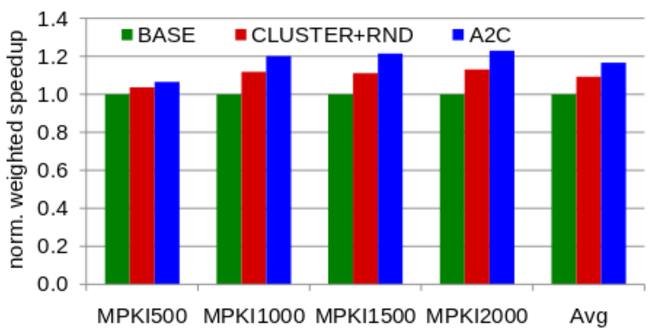
Key Results

Methodology

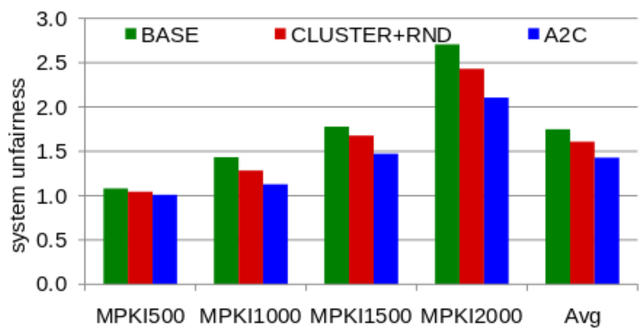
Three systems:
- Baseline with random mapping (BASE),
- Random mapping of applications to cores (CLUSTER+RND)
- Our final system with application-to-core (A2C)

Number of Cores	60
L1 Cache	32KB per core. 4 ways, 2-cycle latency
L2 Cache	256KB per core, 16 ways, 6-cycle latency
MSHR	32 entries
Main Memory	4GB. 160-cycle latency 4 channels at 16GB/s
Network Router	4 VCs per port, 4 flits per VC 2-stage wormhole
Network Topology	8x8 mesh, 128 bit bi-directional links
Memory Management	4KB physical and virtual page 512 entries TLB CLOCK page allocation and replacement

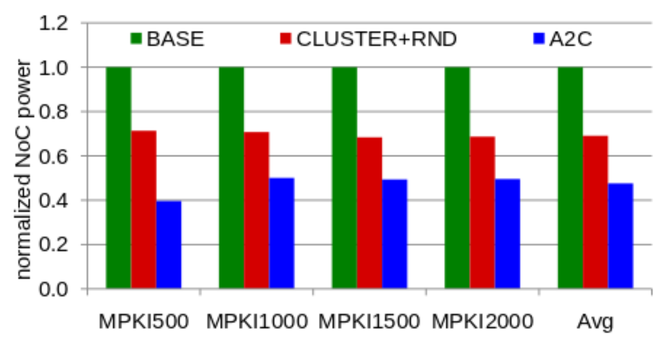
Results



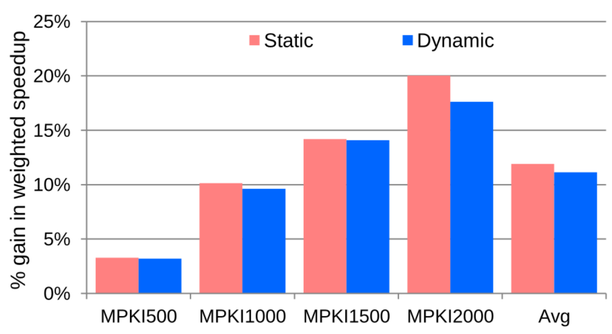
Performance



Fairness



NoC Power



Static A2C vs Dynamic A2C