Cross-Layer Memory Management to Achieve Power and Performance Goals

Dr. Michael Jantz

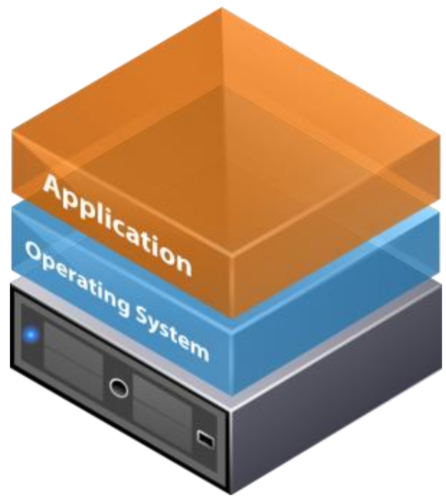
Assistant Professor

University of Tennessee, Knoxville

Introduction

- Assistant Professor at UT since August 2014
- Before UT
  - PhD in Computer Science at KU (July 2014)
  - Intern at Intel Corporation (2012 – 2013)
- Research interests:
  - Compilers (backend optimizations, phase ordering)
  - Operating Systems (kernel instrumentation, memory and power management)
  - Runtime Systems (dynamic compilation, program profiling, heap layout management)

Cross-Layer Memory Management



Publications: VEE ’13 [1], Linux Symposium ’14 [2], OOPSLA '15 [3]

Memory Power Management

- Memory has become a significant player in power and performance
  - Memory power is a dominant factor in servers [4,5,6,7]
- Hardware can automatically *power down* individual memory modules
- Memory power management is challenging
  - Small footprint can reside in multiple devices
  - Different memory regions can have different requirements

Example Scenario

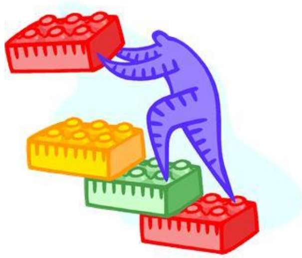
- Server system with database workload with 1TB DRAM
  - All memory in use, but only 2% of pages are accessed frequently
  - CPU utilization is low
- **How to reduce power consumption?**

A Collaborative Approach to Memory Management

- Effective memory management is difficult due to virtualization of memory
- We propose a collaborative approach:
  - Applications – communicate memory usage intent to OS
  - OS – interprets application intent and manages physical memory over hardware units
  - Hardware – communicate hardware layout to the OS to guide memory management decisions

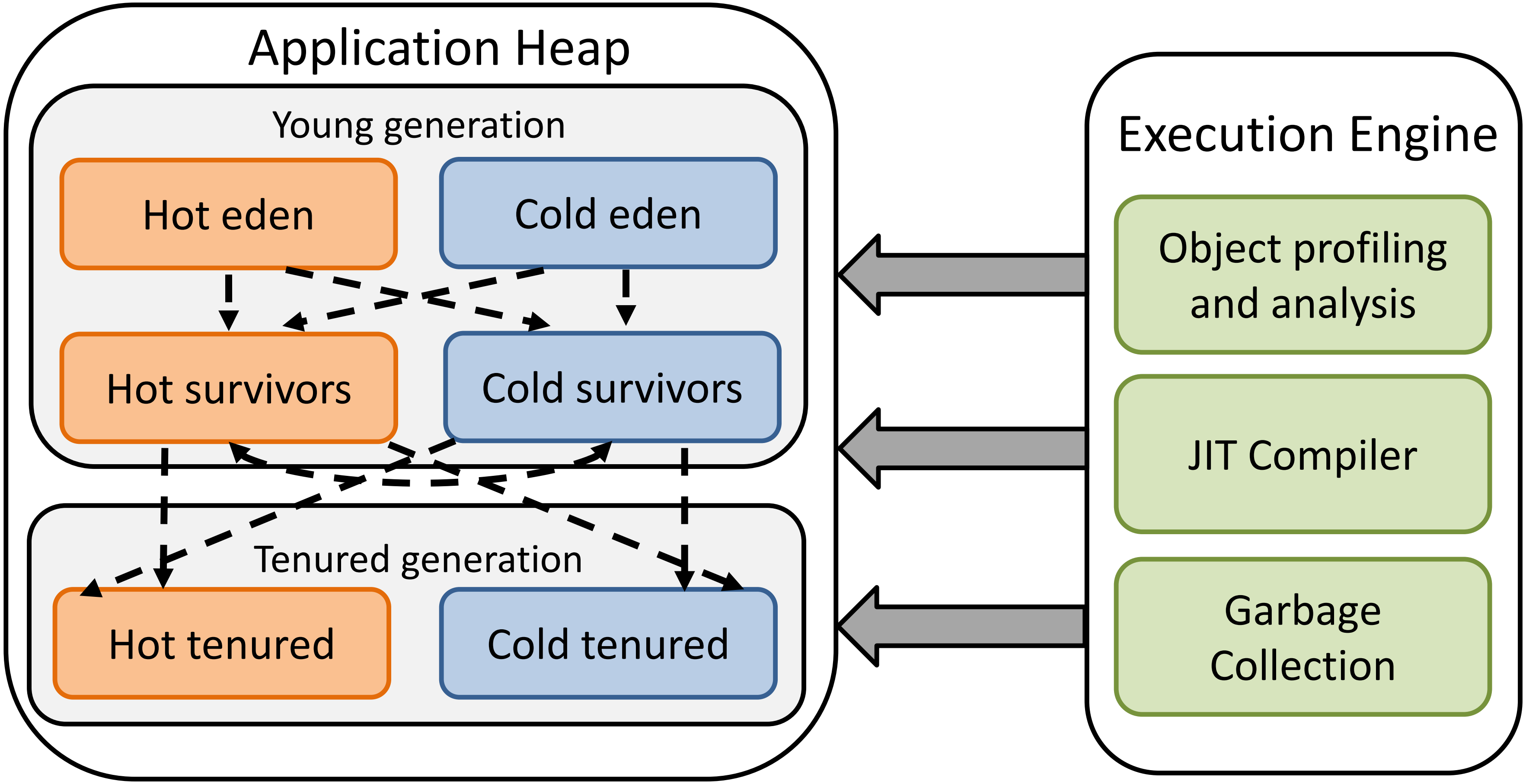
Application Guidance in the Linux Kernel

- Implemented by re-architecting a recent Linux kernel
  - Applications pass guidance to the OS by *coloring* virtual address ranges with a system call interface
  - OS organizes physical memory into software structures that correspond to hardware memory devices (*trays*)
- Limitations of our Linux kernel-based framework:
  - Little understanding of what kind of guidance will be most useful for existing workloads
  - All hints must be manually inserted into source code

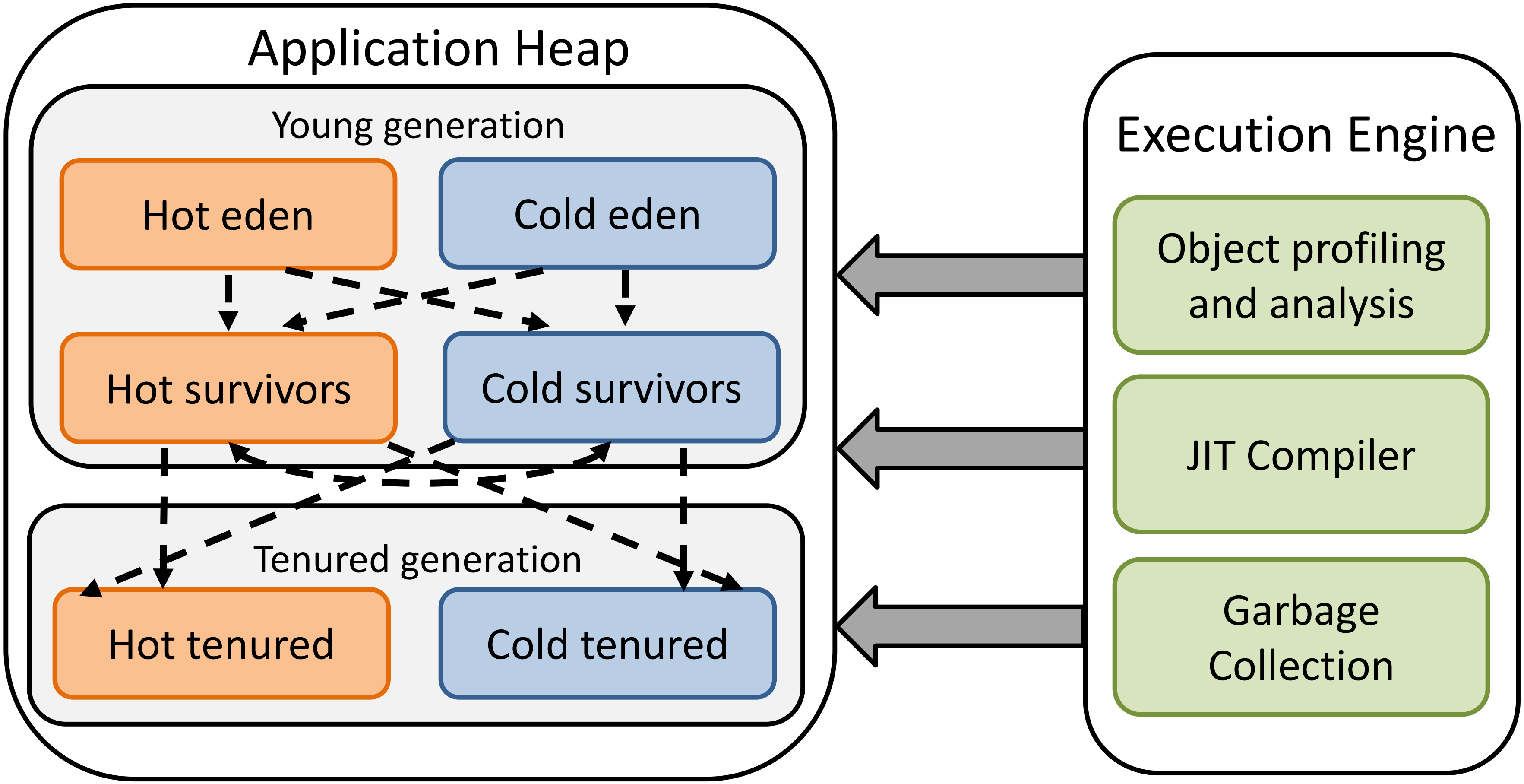


Automatic Guidance in the Application Layer

- Our approach: integrate with *automated* mechanism to generate guidance for the OS
  - No source code modifications or recompilations
- Implemented in the HotSpot JVM
  - Create separate heap regions for different usage patterns
  - Instrumentation and analysis to build memory profile
  - Partition/allocate live objects into separate regions according to partitioning strategy
  - Communicates heap region information to the OS



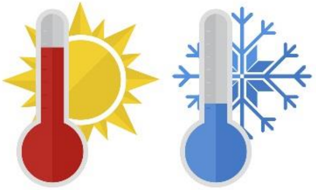
- Employ the default HotSpot config. for server-class applications
- Divide survivor / tenured spaces into spaces for hot / cold objects



- Partition allocation sites and objects into hot / cold sets
- Color spaces on creation or resize

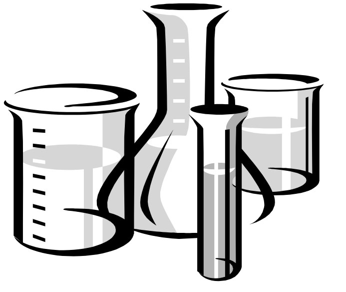
Potential of JVM Framework

- Our goal: evaluate power-saving potential when hot / cold objects are known statically
- MemBench: Java benchmark that uses different object types for hot / cold memory
- “HotObject” and “ColdObject”
  - Contain memory resources (array of integers)
  - Implement different functions for accessing mem.



Experimental Platform

- Hardware
  - Single node of 2-socket server machine
  - Processor: Intel Xeon E5-2620 (12 threads @ 2.1GHz)
  - Memory: 32GB DDR3 memory (four DIMM's, each connected to its own channel)
- Operating System
  - CentOS 6.5 with Linux 2.6.32
- HotSpot JVM
  - v. 1.6.0_24, 64-bit
  - Default configuration for server-class applications

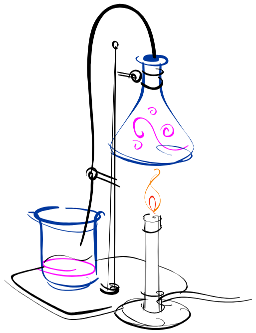


The MemBench Benchmark

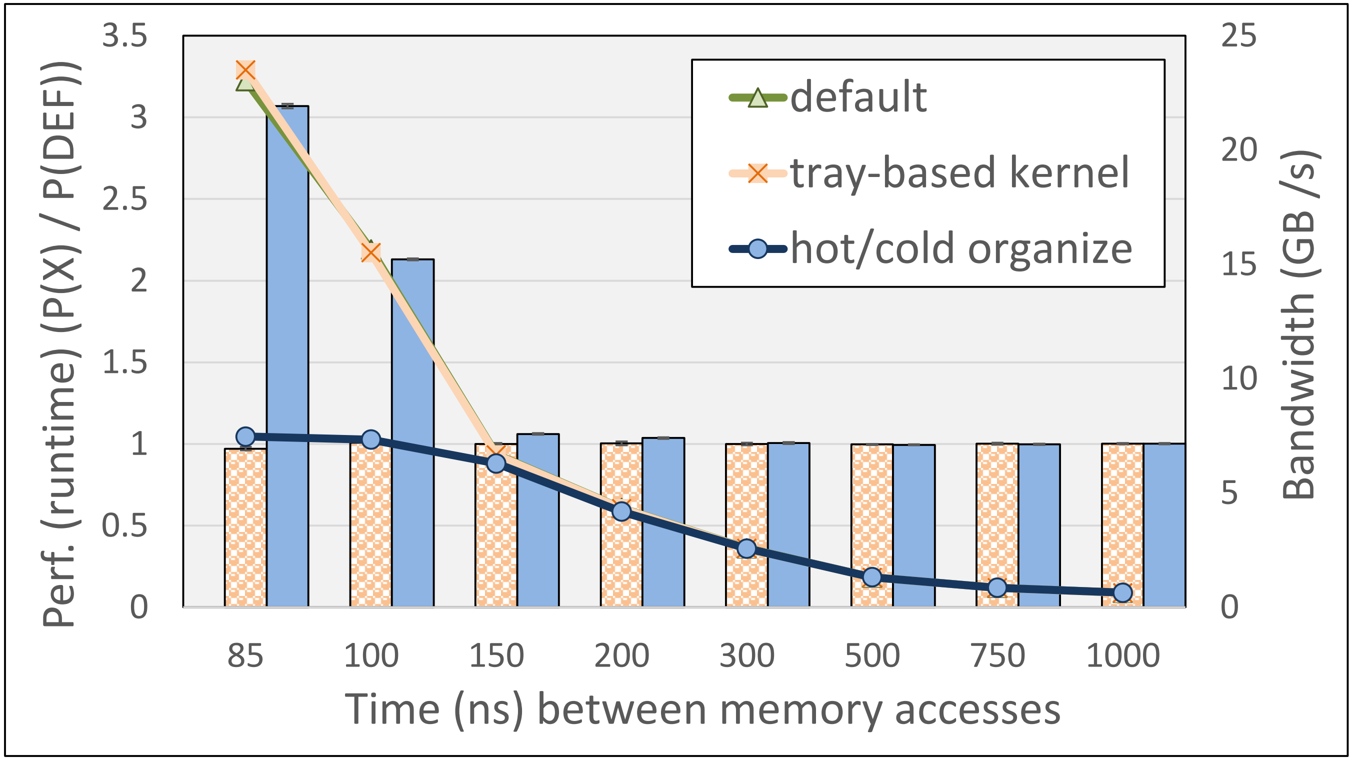
- Object allocation
  - Creates “HotObject” and “ColdObject” objects in a large in-memory array
  - # of hots < # of colds (~15% of all objects)
  - Object array occupies most (~90%) system mem.
- Multi-threaded object access
  - Object array divided into 12 separate parts, each passed to its own thread
  - Iterate over object array, only accessing hot objects
- Optional *delay* parameter

MemBench Configurations

- Three configurations
  - Default
  - Tray-based kernel (custom kernel, default HotSpot)
  - Hot/cold organize (custom kernel, custom HotSpot)
- Delay varied from "no delay" to 1000ns
  - With no delay, 85ns between memory accesses

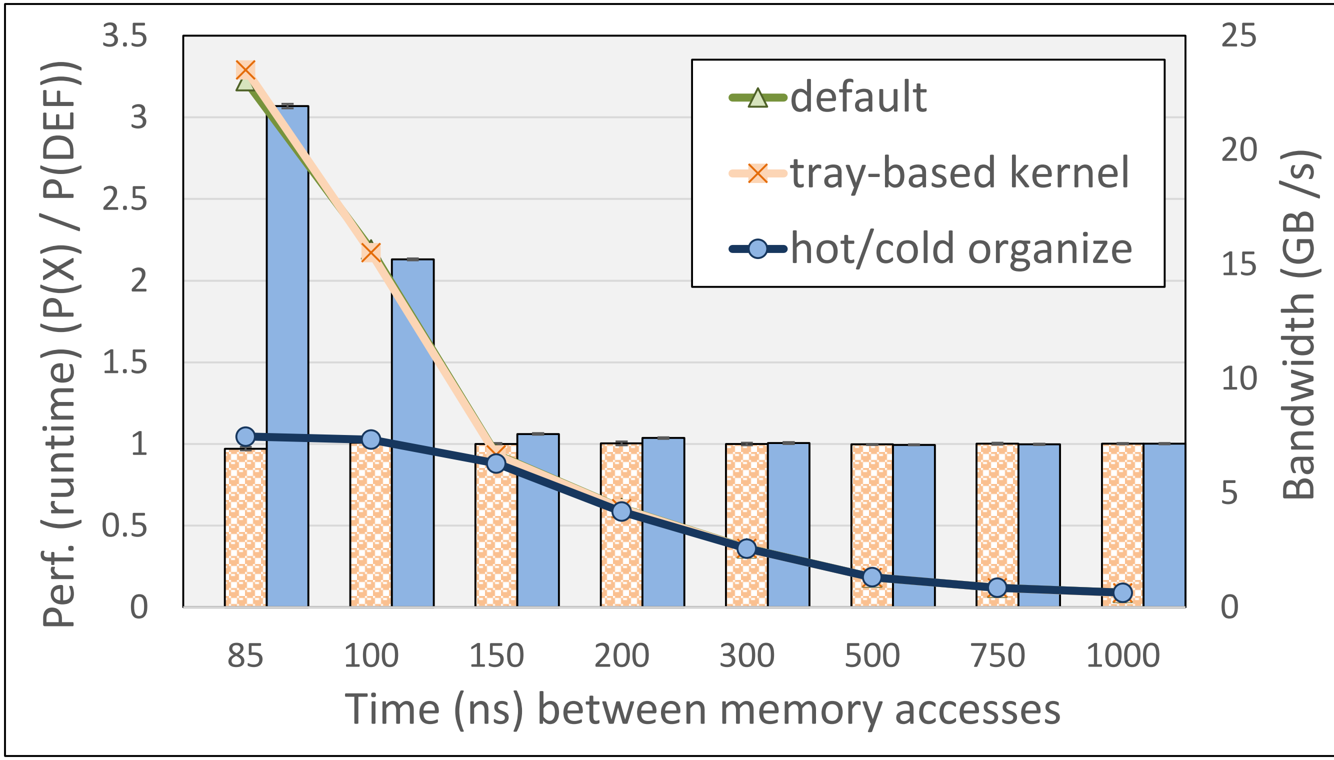


MemBench Performance



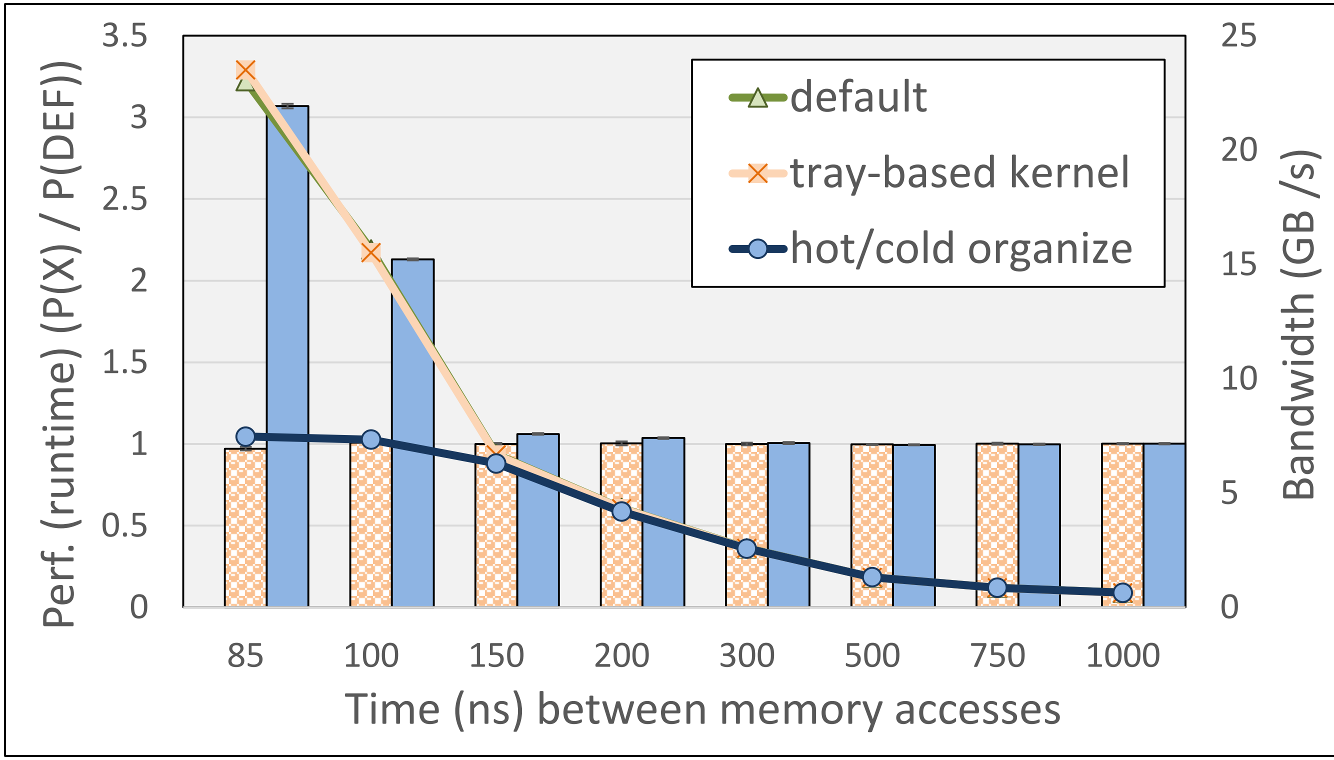
- Tray-based kernel has about same performance as default
- Hot/cold organize exhibits poor performance with low delay

MemBench Bandwidth



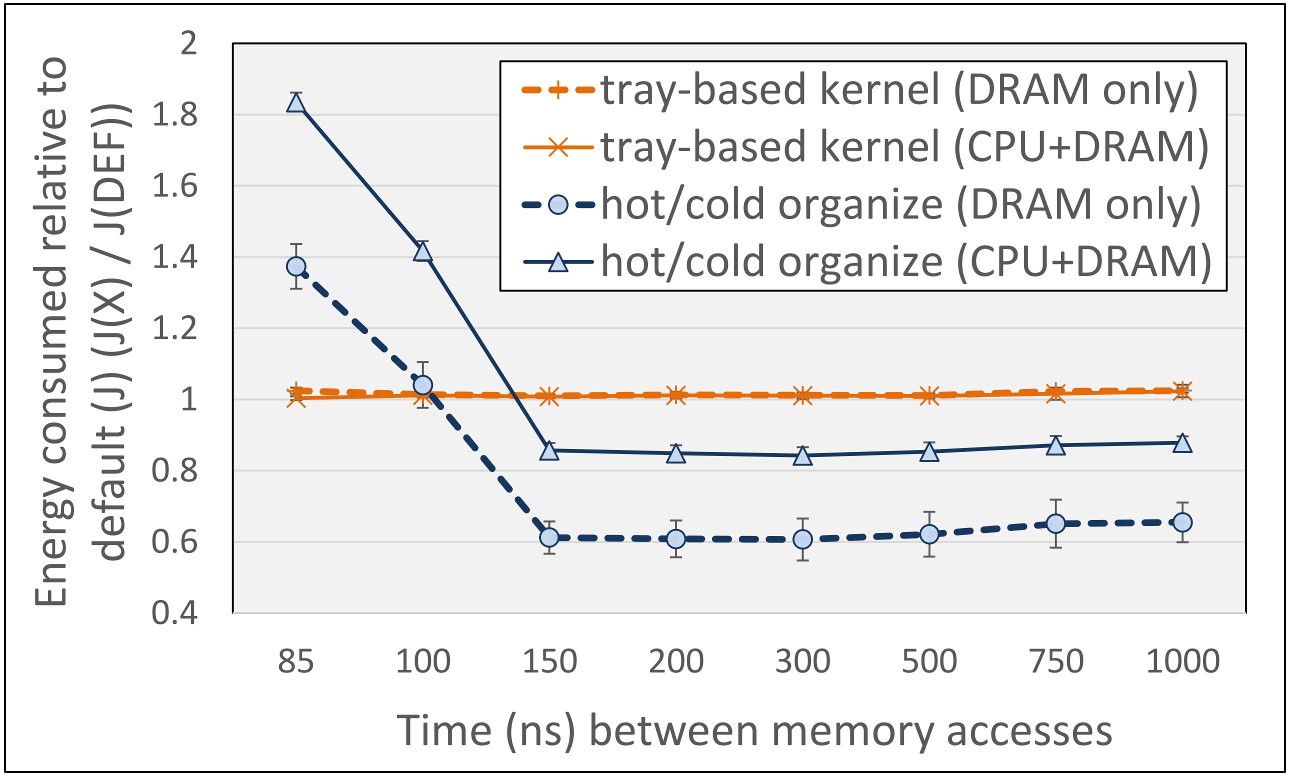
- Default and tray-based kernel produce high memory bandwidth when delay is low
- Placement of hot objects across multiple channels enables higher bandwidth

MemBench Bandwidth



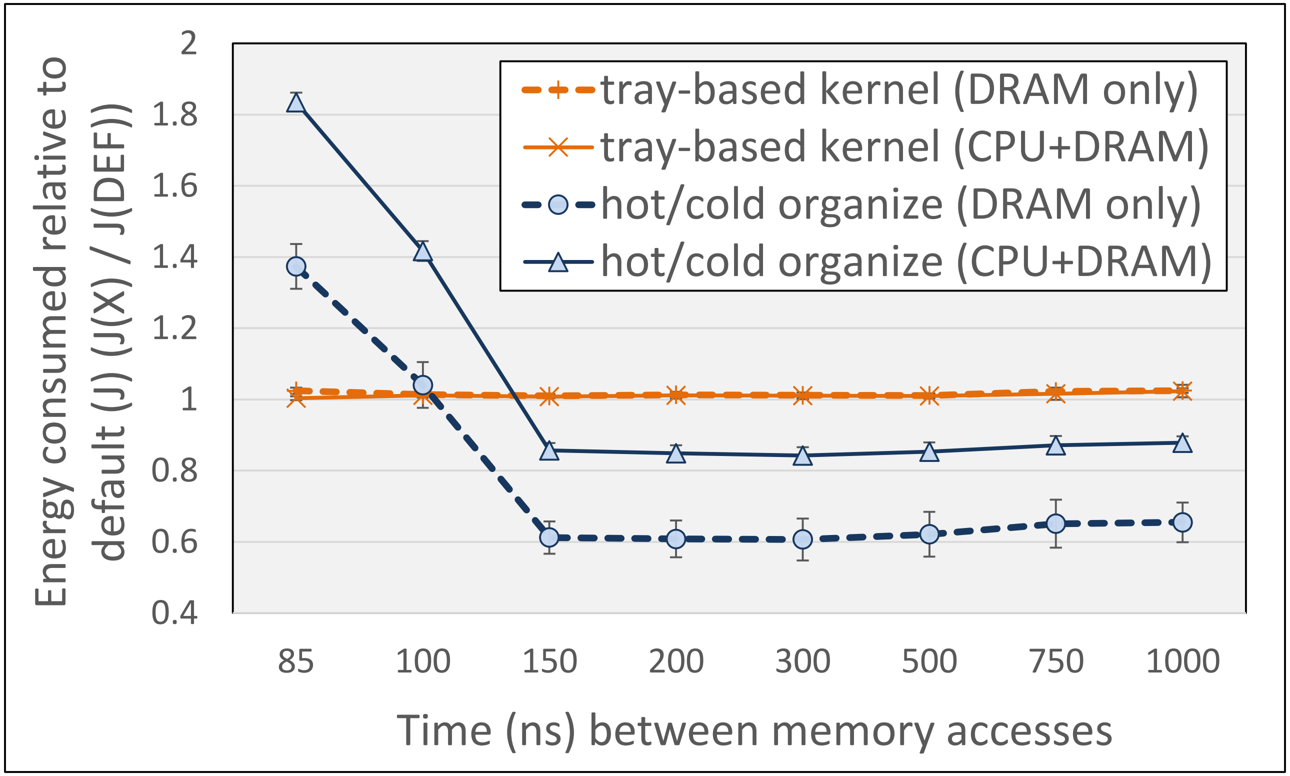
- Hot/cold organize - hot objects co-located on single channel
- Increased delays reduces bandwidth reqs. of the workload

MemBench Energy



- Hot/cold organize consumes much less power with low delay
- Even when BW reqs. are reduced, hot/cold organize consumes less power than other configurations

MemBench Energy



- Significant energy savings potential with custom JVM
- Max. DRAM energy savings of ~39%, max. CPU+DRAM energy savings of ~15%

Results Summary

- Object partitioning strategies
  - Offline approach partitions allocation points
  - Online approach uses sampling to predict object access patterns
- Evaluate with standard sets of benchmarks
  - DaCapo, SciMark
- Achieve 10% average DRAM energy savings, 2.8% CPU+DRAM reduction
- Performance overhead
  - 2.2% for offline, 5% for online

Current and Future Projects in Cross-Layer Memory Management

- Improve performance and efficiency
  - Reduce overhead of online sampling
  - Automatic bandwidth management
- Applications for heterogeneous memory architectures
- Exploit data object placement *within* each page to improve efficiency

Conclusions

- Research focuses on software systems
  - Compilers, operating systems, and runtime systems
- Cross-layer memory management
  - Achieving power/performance efficiency in memory requires a cross-layer approach
  - First framework to use usage patterns of application objects to steer low-level memory mgmt.
  - Approach shows promise for reducing DRAM energy
  - Opens several avenues for future research in collaborative memory management

Questions?

References

1. Michael Jantz, Carl Strickland, Karthik Kumar, Martin Dimitrov, and Kshitij A. Doshi. A Framework for Application Guidance in Virtual Memory Systems. In the *ACM SIGPLAN Conference on Virtual Execution Environments (VEE '13)*, March 16-17, 2013.
2. Michael Jantz, Kshitij Doshi, Prasad Kulkarni, and Heechul Yun. Leveraging MPST in Linux with Application Guidance to Achieve Power-Performance Goals. To appear in *Linux Symposium*, Ottawa, Canada. May 2014.
3. Michael Jantz, Forrest J. Robinson, Prasad A. Kulkarni, and Kshitij A. Doshi. Cross-Layer Memory Management for Managed Language Applications. To be published in the *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '15)*, October 25 - October 30, 2015.
4. C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. Computer ,36 (12):39–48, Dec. 2003
5. Urs Hoelzle and Luiz Andre Barroso. The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan and Claypool Publishers, 1st edition, 2009.
6. Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA '09, pages 267--278, New York, NY, USA, 2009. ACM.
7. Krishna T. Malladi, Benjamin C. Lee, Frank A. Nothaft, Christos Kozyrakis, Karthika Periyathambi, and Mark Horowitz. Towards energy-proportional datacenter memory with mobile dram. In Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12, pages 37--48, Washington, DC, USA, 2012. IEEE Computer Society.