Leveraging MPST in Linux with Application Guidance to Achieve Power and Performance Goals

Michael R. Jantz\textsuperscript{1}, Kshitij A. Doshi\textsuperscript{2}, Prasad A. Kulkarni\textsuperscript{1}, and Heechul Yun\textsuperscript{1}

\textsuperscript{1} University of Kansas, Lawrence, Kansas
\textsuperscript{2} Intel Corporation, Chandler, Arizona
Introduction

• Memory has become a significant player in power and performance
• Memory power management is challenging
• Propose a collaborative approach between applications, operating system, and hardware:
  – Applications – insert instructions to communicate to OS memory usage intent
  – OS – re-architect memory management to interpret application intent and manage memory over hardware units
  – Hardware – communicate hardware layout to the OS to guide memory management decisions
• Implemented framework by re-architecting recent Linux kernel
• Experimental evaluation with industrial-grade JVM
Why

• CPU and Memory are most significant players for power and performance
  – In servers, memory power == 40% of total power [1]

• Applications can direct CPU usage
  – threads may be affinitized to individual cores or migrated b/w cores
  – prioritize threads for task deadlines (with nice)
  – individual cores may be turned off when unused

• Surprisingly, much of this flexibility does not exist for controlling memory
Example Scenario

• System with database workload with 512GB DRAM
  – All memory in use, but only 2% of pages are accessed frequently
  – CPU utilization is low

• How to reduce power consumption?
Challenges in Managing Memory Power

• Memory refs. have temporal and spatial variation
• At least two levels of virtualization:
  – Virtual memory abstracts away application-level info
  – Physical memory viewed as single, contiguous array of storage
• No way for agents to cooperate with the OS and with each other
• Lack of a tuning methodology
A Collaborative Approach

• Our approach: enable applications to guide mem. mgmt.

• Requires collaboration between the application, OS, and hardware:
  – Interface for communicating application intent to OS
  – Ability to keep track of which memory hardware units host which physical pages during memory mgmt.

• To achieve this, we propose the following abstractions:
  – Colors
  – Trays
Communicating Application Intent with Colors

- Color = a hint for how pages will be used
  - Colors applied to sets of virtual pages that are alike
  - Attributes associated with each color

- Attributes express different types of distinctions:
  - Hot and cold pages (frequency of access)
  - Pages belonging to data structures with different usage patterns

- Allow applications to remain agnostic to lower level details of mem. mgmt.
Power-Manageable Units Represented as Trays

- Tray = software structure containing sets of pages that constitute a power-manageable unit
- Requires mapping from physical addresses to power-manageable units
- ACPI 5.0 *memory power state table* (MPST):
  - Phys. address ranges --> mem. hardware units
• Application with two distinct sets of memory
  – Large set of infrequently accessed (cold) memory
  – Small set of frequently accessed (hot) memory

• Specify guidance as a set of standard intents
  – MEM-INTENSITY (hot or cold)
  – MEM-CAPACITY (% of dynamic RSS)

• Intents enable OS to manage mem. more efficiently
  – Save power by co-locating hot / cold memory
  – Recycle large span of cold pages more aggressively
Configuration File to Specify Intents

# Specification for frequency of reference:
INTENT MEM-INTENSITY

# Specification for containing total spread:
INTENT MEM-CAPACITY

# Mapping to a set of colors:
MEM-INTENSITY RED 0 // hot pages
MEM-CAPACITY RED 5 // hint - 5% of RSS
MEM-INTENSITY BLUE 1 // cold pages
MEM-CAPACITY BLUE 3 // hint - 3% of RSS

• Associate colors with intents in configuration files
• Parses config file to create and structure data passed to the OS
Memory Coloring System Calls

<table>
<thead>
<tr>
<th>System Call</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcolor</td>
<td>addr, size, color</td>
<td>Applies <em>color</em> to a virtual address range of length <em>size</em> starting at <em>addr</em></td>
</tr>
<tr>
<td>get_addr_mcolor</td>
<td>addr, *color</td>
<td>Returns the current color of the virtual address <em>addr</em></td>
</tr>
<tr>
<td>set_mcolor_attr</td>
<td>color, *attr</td>
<td>Associates the attribute pointed to by <em>attr</em> with <em>color</em></td>
</tr>
<tr>
<td>get_mcolor_attr</td>
<td>color, *attr</td>
<td>Returns the attribute currently associated with <em>color</em></td>
</tr>
</tbody>
</table>

- Specify colors / intents using system calls
- Use *mcolor, set_mcolor_attr* to color application pages
Memory Management in Linux

- Default Linux kernel organizes physical memory *hierarchically*
  - Nodes --> zones --> lists of physical pages (free lists, LRU lists)
- Distinction for pages on different nodes, but not different ranks
Tray Implementation

Memory management with tray structures in our modified Linux kernel

- Trays exist as a division between zones and physical pages
- Each tray corresponds to a rank, maintains its own lists of pages
- Kernel memory mgmt. routines modified to operate over trays
Evaluation

• Emulating NUMA API’s
• Enabling power consumption proportional to the active footprint
Emulating NUMA API’s

• Modern server systems include API for managing memory over NUMA nodes

• Our goal: demonstrate that framework is flexible and efficient enough to emulate NUMA API functionality

• Experimental Setup
  – Oracle’s HotSpot JVM includes optimization to improve DRAM access locality (implemented w/ NUMA API’s)
  – Modified HotSpot to control memory placement using mem. coloring
  – Compare performance with the default configuration and with optimization implemented w/ NUMA API’s and w/ memory coloring
Memory Coloring Emulates the NUMA API

- Performance of SciMark 2.0 benchmarks with “NUMA-optimized” HotSpot implemented with (1) NUMA API’s and (2) memory coloring framework
- Performance is similar for both implementations
Memory Coloring Emulates the NUMA API

- % of memory reads satisfied by NUMA-local DRAM for SciMark 2.0 benchmarks with each HotSpot configuration.
- Performance with each implementation is (again) roughly the same
Enabling Power Consumption Proportional to the Active Footprint

• Our goal: demonstrate potential of our custom kernel to reduce power in memory

• Experimental setup:
  – Custom workload that incrementally increases memory usage in 2GB steps
  – Compare three configurations on single node of server machine with 16GB of RAM
    • Default kernel with physical address interleaving
    • Default kernel with no interleaving
    • Custom kernel with tray-based allocation
Enabling Power Consumption Proportional to the Active Footprint

- Default kernel yields high power consumption even with small footprint
- Custom kernel – tray-based allocation enables power consumption proportional to the active footprint
Future Improvements

• Problems:
  – Little understanding of which colors or coloring hints will be most useful for existing workloads
  – All colors and hints must be manually inserted

• Developing a set of tools to profile, analyze and control memory usage for applications

• Capabilities we are working on:
  – Detailed memory usage feedback over colored regions
  – On-line techniques to adapt guidance to feedback
  – Compiler / runtime integration to automatically partition and color address space based on profiles of memory usage activity
Conclusion

• A critical first step in meeting the need for a fine-grained, power-aware flexible provisioning of memory.

• Initial implementation demonstrates value
  – But there is much more to be done

• Questions?
References

Backup
Default Linux Kernel

Pages of different types

- Frequently referenced
- Infrequently referenced

Application

Node’s Memory

Problem

Operating system does not see a distinction between:

- different types of pages from the application
- different units of memory that can be independently power managed
Custom Kernel with Memory Containerization

- Pages of different types
  - Frequently referenced
  - Infrequently referenced

Application

Node’s Memory

- Less power management
- More power management
- Self refresh (idle) state

Note: not drawn to scale - $10^6$ 4kB pages can be contained in a 4GB DIMM
Analysis to Automatically Generate Memory Coloring Hints

• Advantages to memory coloring:
  – Broad spectrum of hints can be overlapped
  – Hints can adapt to changes in the system

• Specific tasks
  – Build post-processing to search profiling data for regions to color
  – Construct analysis to relate objects that should be colored to source code
  – Manually insert coloring hints into application to apply ideal guidance and evaluate its impact
Novel System Tools

• Memory usage statistics over colored regions
  – Similar to /proc tools that enable users to query system-wide or per-application memory usage
  – Example: monitor page faults over a particular data structure
  – Will further improve memory usage guidance

• Monitoring memory usage over trays
  – Benefits applications such as whole-system virtualization
  – Provide user-level access to trays through /proc
More Workloads and Usage Scenarios

• Evaluate approach with complex, multi-tier workloads at the realistic scale of server systems
  – Potential applications: open source database, web server, J2EE software packages

• Explore maximizing performance by distributing high-value data widely across memory channels

• Hints for expected access patterns
  – Application guided read ahead or fault ahead with structures with expected sequential access

• Different page recycling policies for trays