HPMMAP: Lightweight Memory Management for Commodity Operating Systems

Brian Kocoloski       Jack Lange
University of Pittsburgh
Lightweight Experience in a Consolidated Environment

• HPC applications need lightweight resource management
  • Tightly-synchronized, massively parallel
  • Inconsistency a huge problem
Lightweight Experience in a Consolidated Environment

- HPC applications need lightweight resource management
  - Tightly-synchronized, massively parallel
  - Inconsistency a huge problem

- Problem: Modern HPC environments require commodity OS/R features
  - Cloud computing / Consolidation with general purpose
  - In-Situ visualization
Lightweight Experience in a Consolidated Environment

• HPC applications need lightweight resource management
  • Tightly-synchronized, massively parallel
  • Inconsistency a huge problem

• Problem: Modern HPC environments require commodity OS/R features
  • Cloud computing / Consolidation with general purpose
  • In-Situ visualization

• This talk: How can we provide lightweight memory management in a fullweight environment?
Lightweight vs Commodity Resource Management

- Commodity management has fundamentally different focus than lightweight management
  - Dynamic, fine-grained resource allocation
  - Resource utilization, fairness, security
  - Degrade applications fairly in response to heavy loads
Lightweight vs Commodity Resource Management

- Commodity management has fundamentally different focus than lightweight management
  - Dynamic, fine-grained resource allocation
  - Resource utilization, fairness, security
  - Degrade applications fairly in response to heavy loads

- Example: Memory Management
  - Demand paging
  - Serialized, coarse-grained address space operations
Lightweight vs Commodity Resource Management

• Commodity management has fundamentally different focus than lightweight management
  • Dynamic, fine-grained resource allocation
  • Resource utilization, fairness, security
  • Degrade applications fairly in response to heavy loads

• Example: Memory Management
  • Demand paging
  • Serialized, coarse-grained address space operations

• Serious HPC Implications
  • Resource efficiency vs. resource isolation
  • System overhead
  • Cannot fully support HPC features (e.g., large pages)
HPMMAP: High Performance Memory Mapping and Allocation Platform

- Independent and isolated memory management layers
- Linux kernel module: NO kernel modifications
- System call interception: NO application modifications
- Lightweight memory management: NO page faults
- Up to 50% performance improvement for HPC apps
Talk Roadmap

• Detailed Analysis of Linux Memory Management
  • Focus on demand paging architecture
  • Issues with prominent large page solutions

• Design and Implementation of HPMMAP
  • No kernel or application modification

• Single-Node Evaluation Illustrating HPMMAP Performance Benefits

• Multi-Node Evaluation Illustrating Scalability
Linux Memory Management

• Default Linux: On-demand Paging
  • Primary goal: optimize memory utilization
  • Reduce overhead of common behavior (fork/exec)

• Optimized Linux: Large Pages
  • Transparent Huge Pages
  • HugeTLBfs
  • Both integrated with demand paging architecture
Linux Memory Management

• Default Linux: On-demand Paging
  • Primary goal: optimize memory utilization
  • Reduce overhead of common behavior (fork/exec)

• Optimized Linux: Large Pages
  • Transparent Huge Pages
  • HugeTLBfs
  • Both integrated with demand paging architecture

• Our work: determine implications of these features for HPC
Transparent Huge Pages

- Transparent Huge Pages (THP)
  - (1) Page fault handler uses large pages when possible
  - (2) `khugepaged` address space merging
Transparent Huge Pages

- **Transparent Huge Pages (THP)**
  - (1) Page fault handler uses large pages when possible
  - (2) `khugepaged` address space merging

- **`khugepaged`**
  - Background kernel thread
  - Periodically allocates and “merges” large page into address space of any process requesting THP support
  - Requires global page table lock
  - Driven by OS heuristics – no knowledge of application workload
Transparent Huge Pages

• Ran miniMD benchmark from Mantevo twice:
  • As only application
  • Co-located parallel kernel build
• “Merge” – small page faults stalled by THP merge operation

<table>
<thead>
<tr>
<th>Added Load</th>
<th>Fault Size</th>
<th>Total Faults</th>
<th>Avg Cycles</th>
<th>Stdev Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Small</td>
<td>136,004</td>
<td>1,768</td>
<td>993</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>367,675</td>
<td>65,663</td>
</tr>
<tr>
<td></td>
<td><strong>Merge</strong></td>
<td><strong>30</strong></td>
<td><strong>1,005,412</strong></td>
<td><strong>503,422</strong></td>
</tr>
<tr>
<td>Yes</td>
<td>Small</td>
<td>135,987</td>
<td>2,206</td>
<td>1,444</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>757,598</td>
<td>61,439</td>
</tr>
<tr>
<td></td>
<td><strong>Merge</strong></td>
<td><strong>45</strong></td>
<td><strong>3,360,292</strong></td>
<td><strong>4,017,001</strong></td>
</tr>
</tbody>
</table>
Transparent Huge Pages

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build
- “Merge” – small page faults stalled by THP merge operation
- Large page overhead increased by nearly 100% with added load

<table>
<thead>
<tr>
<th>Added Load</th>
<th>Fault Size</th>
<th>Total Faults</th>
<th>Avg Cycles</th>
<th>Stdev Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Small</td>
<td>136,004</td>
<td>1,768</td>
<td>993</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>367,675</td>
<td>65,663</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>30</td>
<td>1,005,412</td>
<td>503,422</td>
</tr>
<tr>
<td>Yes</td>
<td>Small</td>
<td>135,987</td>
<td>2,206</td>
<td>1,444</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>757,598</td>
<td>61,439</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>45</td>
<td>3,360,292</td>
<td>4,017,001</td>
</tr>
</tbody>
</table>
Transparent Huge Pages

• Ran miniMD benchmark from Mantevo twice:
  • As only application
  • Co-located parallel kernel build

• “Merge” – small page faults stalled by THP merge operation

• Large page overhead increased by nearly 100% with added load

• Total number of merges increased by 50% with added load

<table>
<thead>
<tr>
<th>Added Load</th>
<th>Fault Size</th>
<th>Total Faults</th>
<th>Avg Cycles</th>
<th>Stddev Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Small</td>
<td>136,004</td>
<td>1,768</td>
<td>993</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>367,675</td>
<td>65,663</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>30</td>
<td>1,005,412</td>
<td>503,422</td>
</tr>
<tr>
<td>Yes</td>
<td>Small</td>
<td>135,987</td>
<td>2,206</td>
<td>1,444</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>757,598</td>
<td>61,439</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>45</td>
<td>3,360,292</td>
<td>4,017,001</td>
</tr>
</tbody>
</table>
Transparent Huge Pages

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build
- “Merge” – small page faults stalled by THP merge operation

- Large page overhead increased by nearly 100% with added load
- Total number of merges increased by 50% with added load
- Merge overhead increased by over 300% with added load

<table>
<thead>
<tr>
<th>Added Load</th>
<th>Fault Size</th>
<th>Total Faults</th>
<th>Avg Cycles</th>
<th>Stddev Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Small</td>
<td>136,004</td>
<td>1,768</td>
<td>993</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>367,675</td>
<td>65,663</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>30</td>
<td>1,005,412</td>
<td>503,422</td>
</tr>
<tr>
<td>Yes</td>
<td>Small</td>
<td>135,987</td>
<td>2,206</td>
<td>1,444</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>1,060</td>
<td>757,598</td>
<td>61,439</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
<td>45</td>
<td>3,360,292</td>
<td>4,017,001</td>
</tr>
</tbody>
</table>
Transparent Huge Pages

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build
- “Merge” – small page faults stalled by THP merge operation
- Large page overhead increased by nearly 100% with added load
- Total number of merges increased by 50% with added load
- Merge overhead increased by over 300% with added load
- Merge standard deviation increased by nearly 800% with added load
Transparent Huge Pages

- Large page faults green, small faults delayed by merges blue
- Generally periodic, but not synchronized
Transparent Huge Pages

- Large page faults green, small faults delayed by merges blue
- Generally periodic, but not synchronized
- Variability increases dramatically under load
HugeTLBfs

- HugeTLBfs
  - RAM-based filesystem supporting large page allocation
  - Requires pre-allocated memory pools reserved by system administrator
  - Access generally managed through *libhugetlbfs*
HugeTLBfs

• HugeTLBfs
  • RAM-based filesystem supporting large page allocation
  • Requires pre-allocated memory pools reserved by system administrator
  • Access generally managed through *libhugetlbfs*

• Limitations
  • Cannot back process stacks
  • Configuration challenges
  • Highly susceptible to overhead from system load
HugeTLBfs

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build

<table>
<thead>
<tr>
<th>Added Load</th>
<th>Fault Size</th>
<th>Total Faults</th>
<th>Avg Cycles</th>
<th>Stddev Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Small</td>
<td>1,310</td>
<td>1,350</td>
<td>1,683</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>84</td>
<td>735,384</td>
<td>458,239</td>
</tr>
<tr>
<td>Yes</td>
<td>Small</td>
<td>1,777</td>
<td>475,724</td>
<td>16,387,888</td>
</tr>
<tr>
<td></td>
<td>Large</td>
<td>75</td>
<td>615,162</td>
<td>225,726</td>
</tr>
</tbody>
</table>
HugeTLBfs

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build

- Large page fault performance generally unaffected by added load
  - Demonstrates effectiveness of pre-reserved memory pools
HugeTLBfs

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build

- Large page fault performance generally unaffected by added load
  - Demonstrates effectiveness of pre-reserved memory pools

- Small page fault overhead increases by nearly 475,000 cycles on average
HugeTLBfs

- Ran miniMD benchmark from Mantevo twice:
  - As only application
  - Co-located parallel kernel build

- Large page fault performance generally unaffected by added load
  - Demonstrates effectiveness of pre-reserved memory pools

- Small page fault overhead increases by nearly 475,000 cycles on average

- Performance considerably more variable
  - Standard deviation roughly 30x higher than the average!
HugeTLBfs

Page Fault Cycles vs. App Runtime (s) for HPCCG, CoMD, and miniFE with and without parallel kernel build.

- HPCCG: 10M page fault cycles vs. 51 seconds runtime.
- CoMD: 3M page fault cycles vs. 248 seconds runtime.
- miniFE: 3M page fault cycles vs. 54 seconds runtime.

No Competition vs. With Parallel Kernel Build
HugeTLBfs

- Overhead of small page faults increases substantially
  - Ample memory available via reserved memory pools, but inaccessible for small faults
  - Illustrates configuration challenges
Linux Memory Management: HPC Implications

• Conclusions of Linux Memory Management Analysis:
  • Memory isolation insufficient for HPC when system is under significant load
  • Large page solutions not fully HPC-compatible

• Demand Paging is not an HPC feature
  • Poses problems when adopting HPC features like large pages
  • Both Linux large page solutions are impacted in different ways

• Solution: HPMMMAP
HPMMAP: High Performance Memory Mapping and Allocation Platform

- Independent and isolated memory management layers
- Lightweight Memory Management
  - Large pages the default memory mapping unit
  - 0 page faults during application execution
Kitten Lightweight Kernel

- Lightweight Kernel from Sandia National Labs
- **Mostly Linux-compatible user environment**
- Open source, freely available
  
  https://software.sandia.gov/trac/kitten

- Kitten Memory Management
  - Moves memory management as close to application as possible
  - Virtual address regions (heap, stack, etc.) statically-sized and mapped at process creation
  - Large pages default unit of memory mapping
  - No page fault handling
HPMMAP Overview

- **Lightweight versions** of memory management system calls (*brk, mmap, etc.*)
- “On-request” memory management
  - 0 page faults during application execution
- **Memory offlining**
  - Management of large (128 MB+) contiguous regions
- Utilizes vast unused address space on 64-bit systems
  - Linux has no knowledge of HPMMAP’d regions
Evaluation Methodology

• Consolidated Workloads
  • Evaluate HPC performance with co-located commodity workloads (parallel kernel builds)
  • Evaluate THP, HugeTLBfs, and HPMMAP configurations

• Benchmarks selected from the Mantevo and Sequoia benchmark suites

• Goal: Limit hardware contention
  • Apply CPU and memory pinning for each workload where possible
Single Node Evaluation

• Benchmarks
  • Mantevo (HPCCG, CoMD, miniMD, miniFE)
  • Run in weak-scaling mode

• AMD Opteron Node
  • Two 6-core NUMA sockets
  • 8GB RAM per socket

• Workloads:
  • *Commodity profile A* – 1 co-located kernel build
  • *Commodity profile B* – 2 co-located kernel builds
    • Up to 4 cores over-committed
Single Node Evaluation -
Commodity profile A

- Average 8-core improvement across applications of 15% over THP, 9% over HugeTLBfs
Single Node Evaluation - Commodity profile A

- Average 8-core improvement across applications of 15% over THP, 9% over HugeTLBfs
- THP becomes increasingly variable with scale
Single Node Evaluation - Commodity profile B

- Average 8-core improvement across applications of 16% over THP, 36% over HugeTLBfs
Single Node Evaluation - Commodity profile B

- Average 8-core improvement across applications of 16% over THP, 36% over HugeTLBfs
- HugeTLBfs degrades significantly in all cases at 8 cores – memory pressure due to weak-scaling configuration
Multi-Node Scaling Evaluation

• Benchmarks
  • Mantevo (HPCCG, miniFE) and Sequoia (LAMMPS)
  • Run in weak-scaling mode

• Eight-Node Sandia Test Cluster
  • Two 4-core NUMA sockets (Intel Xeon Cores)
  • 12GB RAM per socket
  • Gigabit Ethernet

• Workloads
  • *Commodity profile C* – 2 co-located kernel builds per node
    • Up to 4 cores over-committed
Multi-Node Evaluation - Commodity profile C

- 32 rank improvement: HPCCG - 11%, miniFE – 6%, LAMMPS – 4%

- HPMMAP shows very few outliers
- miniFE: impact of single node variability on scalability (3% improvement on single node)
- LAMMPS also beginning to show divergence
Future Work

• Memory Management not the only barrier to HPC deployment in consolidated environments
  • Other system software overheads
  • OS noise

• Idea: Fully independent system software stacks
  • Lightweight virtualization (Palacios VMM)
  • Lightweight “co-kernel”
    • We’ve built a system that can launch Kitten on a subset of offlined CPU cores, memory blocks, and PCI devices
Conclusion

• Commodity memory management strategies cannot isolate HPC workloads in consolidated environments
  • Page fault performance illustrates effects of contention
  • Large page solutions not fully HPC compatible

• HPMMAP
  • Independent and isolated lightweight memory manager
  • Requires no kernel modification or application modification

• HPC applications using HPMMAP achieve up to 50% better performance
Thank You

• Brian Kocoloski
  • briankoco@cs.pitt.edu
  • http://people.cs.pitt.edu/~briankoco

• Kitten
  • https://software.sandia.gov/trac/kitten