The Kitten LWK
A More Practical Lightweight Kernel

Kevin Pedretti
Scalable System Software Department
Sandia National Laboratories
ktpedre@sandia.gov

Fast OS Phase II Workshop
June 12, 2009

SAND Number: 2009-3410P
Sandia Has a Long History in MPP Architectures and System Software

- **1990**: nCUBE2
  - Sandia’s first large MPP
  - Achieved Gflops performance on applications

- **1993**: Paragon
  - Tens of users
  - First periods processing MPP
  - World record performance
  - Routine 3D simulations
  - SUNMOS lightweight kernel

- **1997**: ASCI Red
  - Production MPP
  - Hundreds of users
  - Red & Black partitions
  - Improved interconnect
  - High-fidelity coupled 3-D physics
  - Puma/Cougar lightweight kernel

- **1999**: Red Storm
  - 41 Tflops
  - Custom interconnect
  - Purpose built RAS
  - Highly balanced and scalable

- **2004**: Cplant
  - Commodity-based supercomputer
  - Hundreds of users
  - Enhanced simulation capacity
  - Linux-based OS licensed for commercialization

Red Storm
- **41 Tflops**
- Custom interconnect
- Purpose built RAS
- Highly balanced and scalable
- Catamount lightweight kernel

Cplant
- Commodity-based supercomputer
- Hundreds of users
- Enhanced simulation capacity
- Linux-based OS licensed for commercialization

Paragon
- Tens of users
- First periods processing MPP
- World record performance
- Routine 3D simulations
- SUNMOS lightweight kernel

nCUBE2
- Sandia’s first large MPP
- Achieved Gflops performance on applications

Sandia Has a Long History in MPP Architectures and System Software
LWK Advantages

- **Improved scalability and performance**
  - Derived from taking full advantage of hardware and providing a more deterministic runtime environment
  - Primary motivator for use on capability systems
- **Simple resource management**
  - Deterministic virt->phys memory mapping, no demand paging
    - Simplifies network stack, no page pinning or page table walks
    - Enables single-copy intra-node MPI at user-level (SMARTMAP)
    - Enables transparent use of large pages, reduces TLB pressure
  - Simple task scheduling and affinity
- **Better platform for research**
  - Less complex code-base, slower moving target
  - Focus on what’s important rather than working around problems

LWK is focused on “doing the right thing” for HPC on MPPs, being special-purpose provides freedom to innovate
LWK Disadvantages/Complaints

• It’s not Linux
  – Everybody loves Linux! (myself included)
  – Is lightweight Linux still “Linux”?

• It’s missing feature X (threads, python, …)
  – Can be mitigated by adding feature or through running guest OS
  – Users care about the environment, not implementation details

• It’s too much work to maintain
  – All OSes will encounter issues on capability platforms
  – Low batting average getting HPC patches into Linux (.000?)

• There’s no community around it
  – Existing LWKs are closed-source or closed-development
  – Bugs at scale are anything but shallow, not many eyeballs

Our position is LWK advantages will be required on future MPPs, trying to address or mitigate disadvantages with Kitten
Kitten: A More Practical LWK

• **Open-source from start, open development**
  – Kernel is GPLv2, leverages code from Linux
  – User libraries are LGPL

• **Better match for user expectations**
  – Provides mostly Linux-compatible user environment, including threading
  – Supports unmodified compiler toolchains and resulting ELF executables
  – Leverages virtualization hardware to load full guest OSes on-the-fly
    (VMM capability provided by Palacios from V3VEE project, [http://v3vee.org](http://v3vee.org))

• **Better match vendor expectations**
  – Modern code-base with familiar Linux-like organization
  – Drop-in compatible with Linux (e.g., boots like CNL on Cray XT)
  – Infiniband support (currently in private tree)
  – Engaging with vendors and getting feedback

• **End-goal is deployment on future capability system**
Improved Out-of-box Experience

make menuconfig; make isoimage
qemu-system-x86_64 -cdrom arch/x86_64/boot/image.iso -smp 4

README includes directions for booting on PC hardware and Cray XT
Included “Hello World” Initial User Task

```plaintext
(init_task) TEST BEGIN: Task Management
(init_task) My task ID is 65535
(init_task) I'm executing on CPU 0
(init_task) The following CPUs are in my cpumask:
  0 1 2 3
(init_task) Creating a thread on each CPU:
  thread 0: rc=0
  thread 1: rc=0
  thread 2: rc=0
  thread 3: rc=0
(init_task) TEST END: Task Management
(init_task) TEST BEGIN: Hypervisor API
(init_task) Starting a guest OS...
(init_task) Failed, no ISO image available.
(init_task) TEST BEGIN: Sockets API
kfs_mkdirent: Creating 'sock' (parent NONE)
socket_allocate: New socket fd 3 file ffff8100002bc710
(init_task) Going into mainloop: server fd 3 on port 80
(init_task_thread) 3: Hello from a thread on cpu 0
(init_task_thread) 0: Hello from a thread on cpu 1
(init_task_thread) 1: Hello from a thread on cpu 2
(init_task_thread) 2: Hello from a thread on cpu 3
```
Running NPB CG.A as Initial Task (OpenMP, Fortran)

Environment:
“GFORTRAN_UNBUFFERED_ALL=y OMP_NUM_THREADS=4”
NPB print_results() modified to stop early for clarity
Running Puppy.ISO Linux as Guest OS
Kitten console on serial port, Puppy Linux outputs to VGA

puppy.iso is 95 MB image embedded in hello_world ELF executable, hello_world starts up then makes syscall to launch guest OS
Motivations for Virtualization in HPC

- **Provide full-featured OS functionality in LWK**
  - Custom tailor OS to application’s needs (ConfigOS, JeOS)
  - Allow users to chose compute node OS on-the-fly
  - Potential to mix native LWK apps and guest OSes on same node

- **Dynamically assign compute node roles**
  - Currently service/compute ratio is fixed
  - Some jobs may benefit if allowed to specify a different balance

- **Runtime system replacement**
  - Capability runtimes often poor match for high-throughput serial
  - Runtime environment for guest OSes need not be the same

- **Improve resiliency**
  - Backdoor debug capabilities without special hardware
  - VM-based techniques for job checkpointing and migration
VM Overhead Investigation on Cray XT

- Two configurations:
  - Native: Compute Node Linux (CNL) or Catamount on raw HW
  - Guest: Kitten+Palacios on raw HW, guest CNL or Catamount

- For guest config, two paging schemes tested:
  - Shadow: Software-based VMM control of guest page tables
  - Nested: Hardware-based control of guest page tables

- Hardware:
  - 48 node Cray XT4, 2.2 GHz AMD quad-cores (Budapest)
  - Planning larger-scale testing with more applications

- Goal was to provide as thin a virtualization layer as possible to achieve best case VM performance
  - SeaStar mapped directly into guest, driver para-virtualized
  - Guest allocated physically contiguous memory
Shadow vs. Nested Paging
No Clear Winner

**Shadow Paging**
- $O(N)$ memory accesses per TLB miss
- Palacios managed page tables used by the CPU
- Page tables the guest OS thinks it is using

**Nested Paging**
- $O(N^2)$ memory accesses per TLB miss
- Palacios managed guest phys to host phys page tables
- Guest OS managed guest virt to guest phys page tables

Page Faults
CPU MMU
IMB PingPong Latency: Nested Paging has Lowest Overhead

Still investigating cause of poor performance of shadow paging on Catamount. Likely due to overhead/bug in emulating guest 2 MB pages for pass-through memory-mapped devices.
IMB PingPong Bandwidth: All Cases Converge to Same Peak Bandwidth

For 4KB message:
Native: 285 MB/s
Nested: 123 MB/s
Shadow: 100 MB/s

For 4KB message:
Native: 381 MB/s
Nested: 134 MB/s
Shadow: 58 MB/s

Compute Node Linux

Catamount
16-byte IMB Allreduce Scaling: Native and Nested Paging Scale Similarly
HPCCG Scaling
Sparse CG Solver, ~400 MB/node
5-6% Virtualization Overhead

Poor performance of shadow paging on CNL due to context switching. Could be avoided by adding page table caching to Palacios.

Catamount is essentially doing no context switching, benefiting shadow paging (2n vs. n^2 page table depth issue discussed earlier)

Sandia HPCCG miniapplication available from:
https://software.sandia.gov/mantevo/download.html
CTH Scaling
Shock Physics Application (Weak Scaling, No AMR)
< 5% Virtualization Overhead

Lower is Better

32 node runtime:
Native: 294 sec
Nested: 308 sec (-4.8%)
Shadow: 628 sec

32 node runtime:
Native: 281 sec
Nested: 294 sec (-4.6%)
Shadow: 308 sec

Compute Node Linux
Catamount

Poor performance of shadow paging on CNL due to context switching. Could be avoided by adding page table caching to Palacios.
Future Virtualization Work

• Support Accelerated Portals on Cray XT
  – Eliminates interrupt overhead
  – Makes thin virtualization layer even thinner

• Add support for SMP guests to Palacios

• Perform larger scale experiments
  – Run on thousands of Cray XT nodes
  – Test more applications and more diverse guest Linux workloads
  – Evaluate newer CPUs (Shanghai, Istanbul, Nahalem)

• Create user interface for loading guest OS/app
Kitten supports SMARTMAP
Simple Mapping of Address Region Tables for Multi-core Aware Programming

- Direct access shared memory between processes on a node
  - Access to “remote” data by flipping bits in the virtual address
- Each process still has a separate virtual address space
  - Everything is “private” and everything is “shared”
  - Processes can be threads
- Allows MPI to eliminate all extraneous memory-to-memory copies
  - Single-copy MPI messages
  - No extra copying for non-contiguous datatypes
  - In-place and threaded collective operations
- Not just for MPI
  - Emulate POSIX shared memory regions
  - One-sided PGAS operations
  - Can be used by applications directly
- Leverages lightweight kernel page table layout

For more information see SC’08 paper on SMARTMAP
Current Status

- **Current release (May 1, 2009)**
  - Single node, multi-core, x86_64 only
  - Support for multi-threaded applications
    - Glibc NPTL POSIX threads
    - GCC OpenMP
    - Sandia Qthreads
  - Supports loading guest OSes via Palacios VMM
    - Boots guest Cray Linux Environment, Catamount, Kitten, and Puppy Linux
    - Supports AMD SVM only
    - Tested on PC hardware, Cray XT, and under Qemu

- **Development trees**
  - Support for unmodified OFA Infiniband stack (mthca and ml4x drivers)
  - Catamount user-level for multi-node testing (temporary scaffolding)
  - Exploring simplified job load mechanism via SLURM compatibility
  - Port of Kitten to NEC SX architecture
Conclusion

• **Kitten is a more practical LWK**
  – Fully open-source, open development
  – Better matches user, vendor, and researcher expectations
  – Straightforward out-of-box experience
  – Supports commodity hardware and (soon) Infiniband

• **Kitten provides a thin virtualization layer via Palacios**
  – Increases flexibility of LWK environment
  – Collaboration with separately funded V3VEE project (v3vee.org)

• **LWK enables novel optimizations for multi-core**
  – SMARTMAP
  – Exploring more opportunities
Acknowledgements

• **Kitten**
  – Michael Levenhagen (SNL)
  – Trammell Hudson (SNL)
  – Ron Brightwell (SNL)

• **SMARTMAP**
  – Ron Brightwell (SNL)

• **Palacios VMM**
  – Northwestern Univ: Peter Dinda and Jack Lange
  – U. of New Mexico: Patrick Bridges and Patrick Widener
# Lines of Code in Kitten and Palacios

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of Code sloccount</th>
<th>Lines of Code wc *.c *.h *.s</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kitten</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kitten Core (C)</td>
<td>17,995</td>
<td>29,540</td>
</tr>
<tr>
<td>Kitten x86_64 Arch Code (C+Assembly)</td>
<td>14,604</td>
<td>22,190</td>
</tr>
<tr>
<td>Misc. Contrib Code (Kbuild + lwIP)</td>
<td>27,973</td>
<td>39,593</td>
</tr>
<tr>
<td>Palacios Glue Module (C)</td>
<td>286</td>
<td>455</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>60,858</td>
<td>91,778</td>
</tr>
<tr>
<td><strong>Palacios</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Palacios Core (C+Assembly)</td>
<td>15,084</td>
<td>24,710</td>
</tr>
<tr>
<td>Palacios Virtual Devices (C)</td>
<td>8,708</td>
<td>13,406</td>
</tr>
<tr>
<td>XED Interface (C+Assembly)</td>
<td>4,320</td>
<td>7,712</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>28,112</td>
<td>45,828</td>
</tr>
<tr>
<td><strong>Grand Total</strong></td>
<td>88,970</td>
<td>137,606</td>
</tr>
</tbody>
</table>
Kitten Leverages Linux

• **Repurposes basic functionality from Linux kernel**
  – Hardware bootstrap
  – Basic OS kernel primitives (locking, data structures, context switch, …)
  – Device driver API (supports unmodified OFA Infiniband stack)
  – User-level ABI and API (stack layout, thread-local storage, /sys, …)

• **Innovates in key areas**
  – Memory management
  – Network stack
  – Multi-core messaging optimizations (SMARTMAP)
  – Tick-less operation, OS work split into small pieces