SAND2008-5157P

Kitten Lightweight Kernel

August 7, 2008

Kevin Pedretti

Sandia National Laboratories

ktpedre@sandia.gov

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

Motivations for LWK on Capability Platforms

- Scalability
  - Low to no OS noise/jitter
    - Selfish on very large Infiniband cluster showed 0.5% average noise, 2.5% worst case (see SC08 paper)
    - Hard to unintentionally introduce noise with LWK
  - Support full potential of network HW
    - No error-prone memory pinning/unpinning
    - Physically contiguous memory => 2x higher msg. rate
- Deterministic Performance
  - Minimal run-to run variability
  - Simplifies performance tuning/debugging
- Reliability (next slide)

Software MTTI

- 13 Week Period (16 Sep 2007 to 16 Dec 2007)
  - Catamount: ~1735 hours
  - CNL: ~569 hours

System	Hours	Interrupts	S/W MTTI
CNL Site	2136	5	427
CNL Site	2095	7	299
CNL Site	1964	2	982
CNL Site	840	0	-
Sandia (Catamount)	2093	1	2093
Catamount Site	2087	4	522
Catamount Site	2043	0	-
Catamount Site	2162	1	2162
Catamount Site	2164	1	2164
Catamount Site	2110	0	-

Common Criticisms of LWK

- It's not Linux/Solaris/BSD/AIX/Windows/etc.
- It's missing feature X (threads, python, dynamic libs, ...)
- It's too much work to maintain
- It's a proprietary black box
- There's no community around it
- There's no market for it

- We are trying to address (some of) these with Kitten
- Also, time is ripe for innovation
  - Multicore provides lots of resources, OS role changed
  - My opinion is OS should be treated more like an application or library

Sandia National Laboratories

Kitten Defined

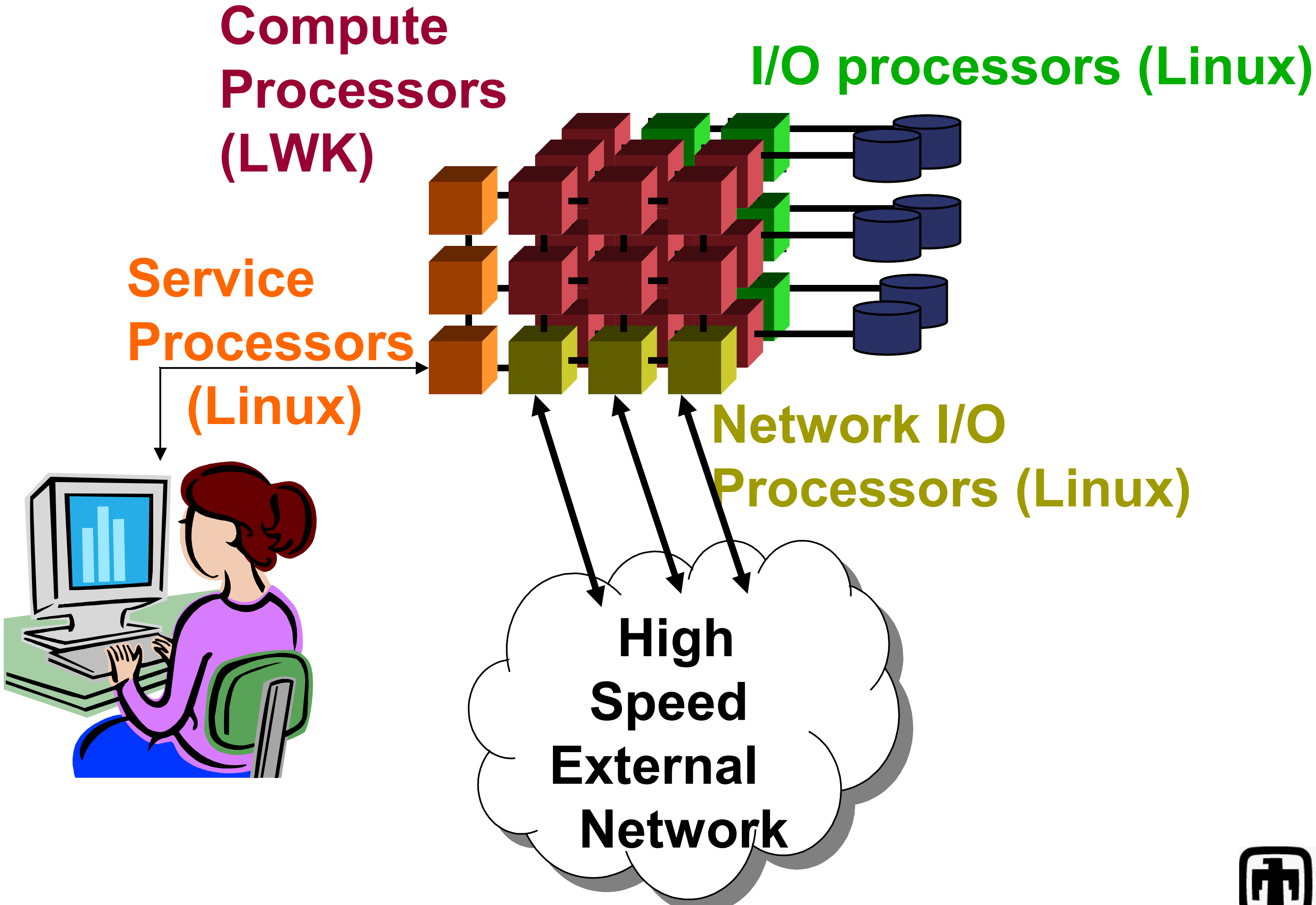
- Kitten is a simple, open-source (GPL) OS kernel that provides basic mechanisms for managing memory, computational tasks, hardware devices, and (in the future) guest operating systems. Kitten does not have an in-kernel file system and instead relies on function-shipping for I/O.
- Kitten is not derived from Catamount
  - Uses no kernel-level code from Catamount or OpenCatamount
  - Essentially same LWK philosophy
- Kitten is derived from Linux, but is not a fork
  - Only leverage pieces of code where it makes sense
  - No expectation of keeping up with Linux

Kitten Defined (cont.)

- Kitten is only a small part of the compute node system software. Suite of user-level libraries provide POSIX environment and interface to the full-system runtime environment (e.g., the job launcher and node allocator).
  - Currently running Kitten with Catamount user-level (PCT, glibc, libsysio, liblustre)
  - Kitten provides subset of Linux ABI system calls, which someday may enable more standard user-level
- Not intended to be a general-purpose OS kernel... simple compute node OS kernel for HPC
- Research platform for system software

Sandia National Laboratories

Kitten is designed for an MPP environment with functional partitions



Project Info

- Funded by Sandia LDRD (Lab Directed Research and Development) and CSRF (Computer Science Research Foundation)
  - Research multi-core and accompanying trends
  - More flexible and open LWK platform
  - Target next-generation capability platforms
- Kitten is ~1 yr. old, based on work from prior CSRF project. Two more years of funding remain.
- 1.25 FTE effort
  - Trammell Hudson, Kurt Ferreira, Sue Kelly, Michael Levenhagen, and Kevin Pedretti
  - Collaborators at Univ. New Mexico and Northwestern
- Key measure of success is having impact on platforms, enabling new capabilities

Sandia National Laboratories

Kitten Kernel-level Functionality

- X86_64 bootstrap (from Linux)
  - Physical memory detection (NUMA)
  - CPU detection (shared cache topology)
  - Kernel runs on all CPUs, locking for shared data
- Physical memory management
  - Tracks contiguous regions of physical memory
    - No page map
    - Each region has type, name, lgroup, + other meta-data
  - Portion of memory set-aside and managed by kernel, remainder managed by user-space init task
    - Default first 8 MB used for kernel dynamic allocations, buddy allocator
    - Large contiguous region(s) available for applications

Kitten Kernel-level Functionality (cont.)

- Virtual memory management
  - Address space object and management API
  - Similar to Linux MM and VMA
  - Contiguous virtual memory maps to contiguous physical memory, possibly via a mapping function
- Task management
  - Per CPU run queues
    - No periodic OS tick
    - Multiple scheduling policies
  - Tasks that share an address space are threads
  - Can support more tasks than CPUs, not usual case
  - No kernel threads yet

Kitten Kernel-level Functionality (cont.)

- Device management
  - Simple console system
    - Drivers for VGA, PC serial port, Cray XT L0
    - KGDB support (ported by Univ. New Mexico)
  - Portals network stack
    - Based on LGPL Portals core + proprietary Cray SeaStar NAL (Interrupt based)
    - Uses Linux IOCTL interface, like Cray Portals
  - Device driver structure similar to Linux
    - External modules not supported initially, maybe later
    - Interrupts are supported
    - Lots missing, but adding functionality as necessary; Platform is the target, not every device out there

Kitten Kernel-level Functionality (cont.)

- Other
  - Sending signals to tasks (currently enough for uClibc's pthreads implementation)
  - Linux clone() interface for creating threads
  - Shared memory regions between tasks (currently enough for PCT)
  - SMARTMAP support (see SC08 paper)
- Future
  - Hypervisor functionality (FY09)
    - Currently investigating, initial plan to use Xen+paravirtualized guests
    - Allocate physically contiguous memory to guests
  - Heterogeneous CPUs, Asymmetric slave CPUs, ...

Current Status

- Nearing initial release
  - X86_64, Cray XT, PC (mostly under Qemu+Bochs)
- Leveraging Catamount user-level
  - Scalable job load
  - User-level I/O libraries (glibc, libsysio, liblustre)
- Will be adding
  - Support for user-level threads (sort of works now)
  - Support for run-time and load-time dynamic libs
  - Use of standard Glibc

Research

- Impact of physically contiguous memory
  - Can result in better best case bandwidth, worse worst-case (due to DRAM bank conflicts)
  - Large performance swings based on alignment/offsets
- SMARTMAP + bandwidth reduction techniques
  - All address spaces mapped into each task
  - Single copy intra-node MPI
  - “Partitioned nodal address space”
- Hooks for lightweight threads and synchronization, advanced architecture capabilities
- More transparent reliability mechanisms
- Virtualization