Analyzing manycore OS design aspects in NIX

Francisco J. Ballesteros
Gorka Guardiola
Enrique Soriano
Universidad Rey Juan Carlos
{nemo,paurea,esoriano}@lsub.org

Noah Evans
Jim McKie
Alcatel-Lucent Bell Labs
{npe,jmk}@plan9.bell-labs.com

Charles Forsyth
Vita Nuova
forsyth@vitanuova.com

Ron Minnich
Google
rminnich@gmail.com

Website: http://lsub.org/ls/nix.html **Source code:** http://code.google.com/p/nix-os/

This material is based upon work supported in part by the Department of Energy under Award Number DE-FC02-10ER25997/DE-SC0005158, by the Comunidad de Madrid (grant S2009/TIC-1692), and the Spanish Government (grant TIN2010-17344).

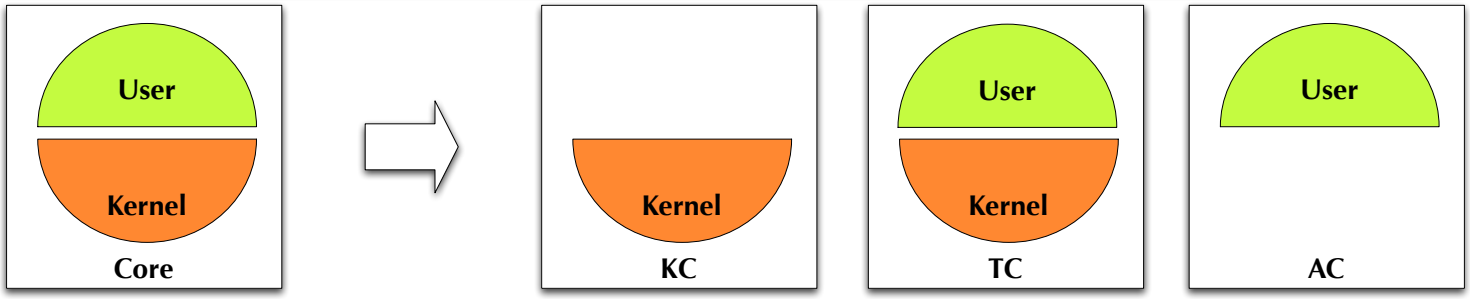
NIX is a novel OS designed for current manycore machines, which includes mechanisms to assign different roles to heterogeneous cores.

NIX includes a NUMA-aware memory allocator suited for new 64-bit x86 processors. The inherent flexibility for specializing cores of NIX makes it particularly suitable for the future heterogeneous multi-core chips. The core roles available in NIX are:

☛ ***Time- sharing Core (TC)***: a common core running kernel and user code in a time sharing fashion.

☛ ***Application Core (AC)***: a core running user code without any interrupt (even without clock interrupts)

☛ ***Kernel Core (KC)***: a core that only runs kernel code on demand. The cores communicate by sending *active messages* that include a function to be executed, together with its arguments.



Work in progress:

☛ *Role assignment to cores*: adding new core roles (e.g. XC), evaluation of core roles for different computing environments, automatic core provisioning and role assignment.

☛ *Scheduling:* quantitative evaluation of different scheduling policies (SMP, AMP, ACPI's proximity domain aware schedulers, etc.) for manycore machines.

☛ *Zero-copy:* design of a simple zero-copy I/O framework to avoid unnecessary data copies within data paths.

Early results

Benchmark: Build the NIX kernel (compile and link around 100 C and assembler source files in parallel) using a RAM disk. The figures show the time of 50 executions of the benchmark for different numbers of operational cores.
Machine: 32-core AMD K10 magny cours, 64 GB RAM.

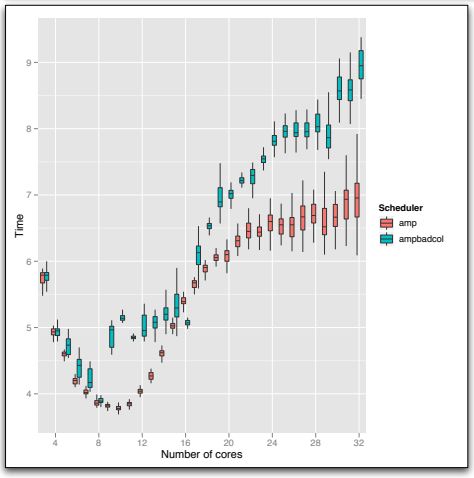


Figure 1: Results for an experiment comparing an AMP scheduler that selects cores according to the ACPI's proximity domain (*amp*) vs. an AMP scheduler that looks first for cores from a different ACPI's domain (*ampbadcol*).

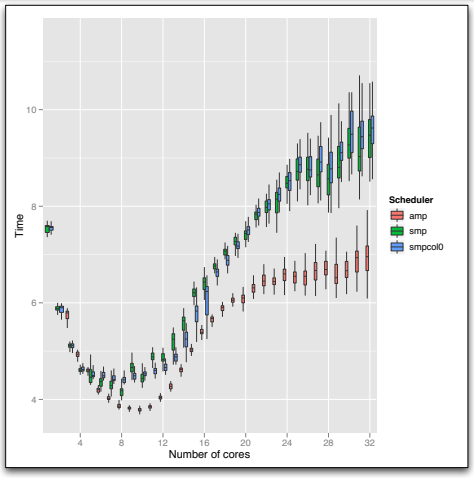


Figure 2: Results for an experiment comparing a SMP scheduler using memory from all ACPI's domains (*smp*), a SMP scheduler with all memory in ACPI's proximity domain zero (*smpcol0*), and the previous AMP scheduler (*amp*).

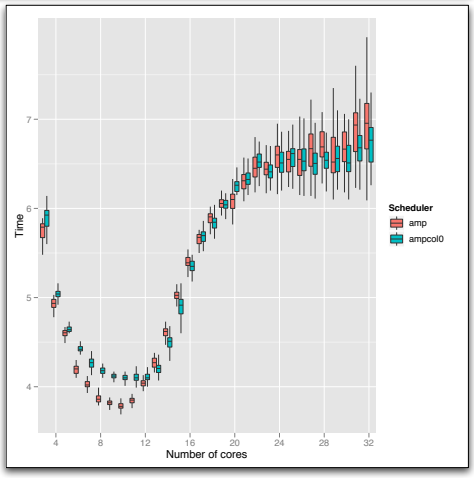


Figure 3: Results for an experiment comparing the previous AMP scheduler (*amp*) vs. an AMP scheduler with all the memory in ACPI's proximity domain zero (*ampcol0*).