INFLUENTIAL OS RESEARCH

Multiprocessors

Jan Bierbaum
Tobias Stumpf

SS 2017
MULTIPROCESSOR ARCHITECTURES
SMP
MULTIPROCESSOR ARCHITECTURES
DISTRIBUTED SYSTEM
MULTIPROCESSOR ARCHITECTURES
CACHE PROBLEMATIC
1. read: lock=0  
2. read: lock=0  
3. write: lock=1  
4. write: lock=1  

initially: lock = 0

MUltiprocessor Architectures
Test-And-Set
MULTIPROCESSOR ARCHITECTURES
THE STANFORD FLASH MULTIPROCESSOR

Source: [KOH+94]
Partitioning
- run multiple independent OSes
- communicate like distributed systems
- e.g. Sun Enterprise10000, Digital's Galaxies OS

Large OS
- single OS controls all resources
- OS creates resource partitions, which can communicate
- e.g. Hive, Hurriance, Cellular-IRIX
Edouard Bugnion, Scott Devine, Kin-shuk Govil and Mendel Rosenblum
Disco Overview

Source: [BDGR97]
Disco Virtual Machine Monitor

- implemented as multi-threaded shared-memory program
- emulates CPU (MIPS R10000), MMU, I/O devices, network
- need to modify HAL to adapt an OS for Disco (optimisation: paravirtualisation)
- uniform address space for non-NUMA aware OSes
- virtual processors can time-share physical cores
- transparent page migration + replication to increase data locality
DISCO
MEMORY MANAGEMENT

Source: [BDGR97]
Pmake: multiprogrammed, short-lived, system and I/O intensive processes (software development)

Engineering: multiprogrammed, long running process (hardware development)

Raytrace: parallel applications (scientific computing)

Database: single memory intensive process (commercial database)
• OSes are designed for ccNUMA and SMP machines
• CPUs are treated as interchangeable parts
• OSes treat programmable devices like non-programmable I/O devices
• general purpose OSes are optimised for common hardware
• multiprocessor OSes for machines with several hundred processors and more than one terabyte of memory already exist
Present Age Research

Why are New OS Designs Necessary?

- Hardware diversity is increasing: GPUs, FPGAs, programmable controllers
- Need for specific hardware optimisations
- More than one ISA
- Cache coherency is given up, e.g. between CPUs and GPU/NICs
moving a process between different cores is difficult
lack of cache coherence
execution time depends on the core
PRESENT AGE RESEARCH
EXAMPLE — MULTICORE
The Multikernel: A New OS Architecture for Scalable Multicore Systems

Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach and Akhilesh Singhania

Idea

- new OS structure, trading multiprocessor system as a network of independent cores
- no inter-core sharing, provides traditional OS services as network services
THE MULTIKERNEL DESIGN PRINCIPLES

- keep OS structure hardware-neutral
- make all inter-core communication explicit
- regard state as replicated instead of shared

Source: [BBD+09]
- cache coherence simplifies memory view but is costly
- message passing less expensive than shared memory

Source: [BBD+09]
THE MULTIKERNEL
CACHE COHERENCE IS NOT A PANACEA

Pros
- simplifies the programmers life

Cons
- with increasing core count cache coherency protocols become expensive
- cache coherence restricts the ability to scale up to around 80 cores
- NICs, GPUs maintain no cache coherence with CPUs
• explicit network-like communication between two cores
• shared-memory areas only for message buffers
• applications can use shared memory
• separation provides resource isolation and management
• separate OS components
• easily adaptable to new hardware
• hardware independent message passing protocols
Process structure

- one process is represented by several dispatching objects
- one dispatching object per core available to process
- dispatching objects are scheduled by the local CPU driver
- communication is between dispatchers
Memory Management

- capability based system to decentralise resource allocation
- virtual memory management at user level
- most memory management operations need global coordination
Inter-core communication

- critical for a multikernel
- using message passing
- Barrelfish uses a user-level RPC mechanism
- memory regions are shared to transfer cache-line sized messages
- receiving uses polling
System Knowledge Base (SKB)

- information about the underlying hardware
- populated during boot (hardware discovery), runtime (measurements) and pre-asserted facts
- used for optimisation
THE MULTIKERNEL EVALUATION — TLB SHOOTDOWN

Source: [BBD+09]
THE MULTIKERNEL EVALUATION — MEMORY UNMAP

Source: [BBD+09]
THE MULTIKERNEL EVALUATION — COMPUTE-BOUND WORKLOADS

(a) OpenMP conjugate gradient (CG)  
(b) OpenMP 3D fast Fourier transform (FT)  
(c) OpenMP integer sort (IS)  

(d) SPLASH-2 Barnes-Hut  
(e) SPLASH-2 radiosity

Source: [BBD⁺09]
Edmund B. Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt

- OS for heterogeneous platforms
- single, uniform set of OS abstractions across different cores
- OS services (e.g. file system access) provided via remote message passing, cp. to distributed systems
- simplify application deployment and tuning
scheduler + memory manager + remote communication primitives
export a single, uniform set of OS abstractions across CPUs
Helios treats a NUMA architecture as a shared-nothing multiprocessor
kernel code is replicated instead of shared
boot-up satellite kernel launches other kernels and acts as coordinator
1. avoid unnecessary remote communication
2. require minimal hardware primitives
3. occupy minimal hardware resources
4. avoid unnecessary local IPC
- message-passing interface
- local/remote communication is transparent for the applications
- zero-copy protocol for local messages
- implementation for remote communication is hardware dependent
- during boot-up the controller creates point-to-point connections to all the satellite kernels
Source: [NHM+09]
• no resource sharing between two satellite kernels
• a process can exist only on one satellite kernel
• all threads of one process have to be executed on the same kernel
• Helios is based on a modified Singularity RDK
• Singularity application are type and memory safe
• memory protection is ensured by software isolation
• only one address space and all applications run with the highest privileges
HELIOS
NAMESPACE

- only component which needs a centralised control
- managed by the coordinator kernel
- applications make services available by registration
- a second application can bind to this service
- coordinator kernel sends a message to create a new channel
- de-registration by an explicit remove message or closing the channel to the namespace
• placement is performance critical
• Helios makes automatic placement decisions depending on an affinity metric
  Positive: hint that two components benefit from a fast message passing or the execution of one component on a specific core
  Neutral: no affect (standard case)
  Negative: expresses an interference between two components
HELIOS
PLACEMENT EXAMPLE

Coordinator kernel

x86

File System +10
Application

IDE Driver

Memory Manager
Scheduler
Namespace
Hardware abstraction layer
DMA

Satellite kernel

XScale Programmable Device

Net Stack +1
NIC Driver

Scheduler
Memory Manager
DMA
Hardware abstraction layer

Local channel
Remote channel stub

Source: [NHM*09]
**HELIOS**  
**EVALUATION — BENEFITS OF PERFORMANCE ISOLATION**

**Scheduling**

- Time (Seconds)

**Mail Server**

- Emails per second
- Instructions per cycle (IPC)

- No sat. kernel
- Sat. kernel

Source: [NHM⁺09]
THREE PAPERS IN ONE SLIDE

- **Disco**: virtualisation to partition the hardware; run different OSes which are not necessarily multiprocessor-aware
- **Barrelfish**: multikernel which treats the machine as a network of independent cores with minimal inter-core sharing of OS data
- **Helios**: single control kernel and several satellite kernels; smallest possible interference to provide scalability; affinity for simplified tuning


Helios: Heterogeneous multiprocessing with satellite kernels.