INFLUENTIAL OS RESEARCH

Multiprocessors

Jan Bierbaum
Tobias Stumpf

SS 2018
Roadmap

Multiprocessor Architectures

Usage in the Old Days (mid 90s)

Disco

Present Age Research

The Multikernel

Helios

Three Papers in one Slide

References
MULTIPROCESSOR ARCHITECTURES

SMP
MULTIPROCESSOR ARCHITECTURES
NUMA
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
Partioning

- 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, Kinshuk Govil and Mendel Rosenblum
DISCO
VIRTUAL MACHINE MONITOR

- implemented as multi-threaded shared-memory program
- virtualises CPU (MIPS R10000), MMU, I/O devices, network
- HAL modified to adapt an OS for Disco (optimisation: paravirtualisation)
- near-uniform address space for non-NUMA aware OSes
- virtual processors can time-share physical cores
- COW disks and shared buffer cache
**DISCO**

**MEMORY MANAGEMENT**

Node 0

VCPU 0

Node 1

VCPU 1

Virtual Pages

Physical Pages

Machine Pages

Source: [BDGR97]
Physical Memory of VM 1

Physical Memory of VM 2

Machine Memory

Source: [BDGR97]
Setup

- FLASH processor, modeled in SimOS machine simulator
- 32 KB caches, 1 MB board-level cache
- memory latency: 300 ns local, 900 ns remote

Applications

- 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)
EVALUATION — VIRTUALISATION OVERHEAD

Source: [BDGR97]
DISCO
EVALUATION — DATA SHARING

Source: [BDG97]
EVALUATION — PAGE MIGRATION AND REPLICATION

![Diagram showing normalized execution time for different load conditions in Engineering and Raytrace.]
• modern OSes designed for ccNUMA and SMP machines
• CPUs as fully interchangeable parts
• OS treats programmable devices like non-programmable I/O devices → access via drivers
• general purpose OSes are optimised for common hardware and usage scenarios
• multiprocessor OSes for machines with several hundred processors and > 1 TB of memory already exist
• hardware diversity is increasing: GPUs, FPGAs, programmable controllers
• need for specific hardware optimisations (caches, interconnects, …)
• more than one ISA
• cache coherency given up, e.g. between CPU and GPU/NIC
• moving a process between different (types of) cores is difficult
• lack of cache coherence
• execution time depends on the core
PRESENT AGE RESEARCH

EXAMPLE — MULTICORE
PRESENT AGE RESEARCH
INTERCONNECT

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

Idea

• new OS structure, treating multiprocessor system as a network of independent cores
• traditional OS services become “network services”
1. explicit inter-core communication
2. hardware-neutral OS structure
3. replicated state instead of shared state
1. explicit inter-core communication
2. hardware-neutral OS structure
3. replicated state instead of shared state
• cache coherence simplifies memory view but has its price
• message passing less expensive than shared memory
Pros
  • simplifies the programmer’s life

Cons
  • with increasing core count cache coherency protocols become expensive
  • cache coherence restricts the ability to scale up the number of cores
  • NICs, GPUs maintain no cache coherence with CPUs
• separate OS components
• easily adaptable to new hardware
• hardware-independent message passing protocols
Process Structure

- each process is represented by one dispatcher object per core it may run on
- communication between dispatchers
- dispatching objects scheduled by the local CPU driver via upcall
Process Structure

- each process is represented by one dispatcher object per core it may run on
- communication between dispatchers
- dispatching objects scheduled by the local CPU driver via upcall

Communication

- critical for a Multikernel
- user-level RPC mechanism
- explicit network-like message passing
- local messages via CPU driver
- shared memory areas to transfer cache-line-sized message between cores
- receiving via polling...for some time (cp. spin lock)
Memory Management

- capability based system to decentralise resource allocation
- virtual memory management entirely at user level
- most memory management operations need global coordination
Memory Management

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

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

Latency (cycles × 1000)

- Broadcast
- Unicast
- Multicast
- NUMA-Aware Multicast

Cores

Source: [BBD+09]
The Multikernel Evaluation — End-to-end Memory Unmap

![Graph showing latency vs cores for Windows, Linux, and Barrelfish. The graph illustrates the performance comparison among the three operating systems as the number of cores increases. The x-axis represents the number of cores, ranging from 2 to 32, while the y-axis represents latency in cycles, scaled by 1000. The graph includes error bars to indicate variability. 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**

**Idea**

- OS for heterogeneous platforms
- simplify application writing, deploying and tuning
- single, uniform set of OS abstractions across different ISAs
- OS services (e.g. file system) provided via remote message passing, cp. distributed systems
Helios

Satellite Kernels

- manages “its” hardware: scheduler + memory manager + remote communication primitives
- Helios treats a NUMA architecture as a shared-nothing multiprocessor
- kernel code is replicated instead of shared
- boot-up kernel launches other satellite 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
Helios is based on a modified Singularity RDK
- Singularity application are type and memory safe
- memory protection is ensured by software isolation
- single address space per kernel, all applications run with the highest privileges
• message-passing interface
• local/remote communication is transparent for the applications
• zero-copy protocol (see Singularity [FAH+06]) for local messages
• hardware-dependent implementation for remote communication
• during boot-up the controller kernel establishes point-to-point connections between all satellite kernels
• no resource sharing between satellite kernels
• each process exists only on one satellite kernel
• no migration between kernels → initial placement is final
• managed by the coordinator kernel
• only component which uses a centralised control instance
• applications make services available via registration
• other application can bind to this service, i.e. request to use it
  → coordinator kernel forwards the message
  → service creates new, direct channel
• de-registration by an explicit remove message or closing the channel to the namespace
• placement is performance critical
• Helios makes automatic placement decisions considering a user-specified affinity metric

Positive  two components benefit from a fast message passing
or  component prefers execution on a specific ISA
Neutral  no affect (standard case)
Negative expresses an interference between two components
HELIOS
PLACEMENT EXAMPLE

Coordinator kernel

x86

Satellite kernel

XScale Programmable Device

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

Scheduling

Mail Server

Time (Seconds)

Emails per second

Instructions per cycle (IPC)

Source: [NHM+09]

No sat. kernel

Sat. kernel
Disco
- virtualisation to manage/partition the hardware
- run different OSes which are not necessarily multiprocessor-aware

Multikernel/“Barrelfish”
- machine as a network of independent cores
- minimal inter-core sharing of OS data: replication + agreement

Helios
- single control kernel and several satellite kernels
- affinity for simplified tuning
[BBD⁺09] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania.

The Multikernel: A new OS architecture for scalable multicore systems.


Disco: Running commodity operating systems on scalable multiprocessors.


[FAH⁺06] Manuel Fahndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, Jim Larus, and Steven Levi.

Language support for fast and reliable message-based communication in Singularity OS.


The Stanford FLASH Multiprocessor.


Helios: Heterogeneous multiprocessing with satellite kernels.