Overview

Multiprocessor OS (Background and Review)
- How does it work? (Background)
- Scalability (Review)

Multiprocessor Hardware
- Contemporary and past systems (Intel, AMD, ARM, Oracle/Sun)
- Experimental (Intel, MS, Polaris)

OS Design for Multiprocessors
- Guidelines
- Design approaches
  - Divide and Conquer (Disco, Tesselation)
  - Reduce Sharing (K42, Corey, Linux, FlexSC, scalable commutativity)
  - No Sharing (Barrelfish, fos)
Multiprocessor OS
Uniprocessor OS

CPU

App2

OS

Run queue
Process control blocks
FS structs

OS data

Application data

App1
App2
App3
App4

Memory
Multiprocessor OS

CPU

App1
OS

CPU

App3
OS

CPU

App4
OS

CPU

App4
OS

Memory

OS data
Run queue
Process control blocks
FS structs

Application data
App1
App2
App3
App4
Multiprocessor OS

Key design challenges:
• Correctness of (shared) data structures
• Scalability (performance doesn’t suffer)
Correctness of Shared Data

Concurrency control

• Locks
• Semaphores
• Transactions
• Lock-free data structures

We know how to do this:

• In the application
• In the OS
Scalability

Speedup as more processors added

Ideal

$$S(N) = \frac{T_1}{T_N}$$
Scalability

Speedup as more processors added

Reality

\[ S(N) = \frac{T_1}{T_N} \]
Serialisation

Parallel Program

Processor 1

Parallel
Parallel
Parallel
Serial
Parallel

Processor 2

Parallel
Parallel
Parallel
Serial
Parallel

Processor 3

Parallel
Parallel
Parallel
Serial
Parallel

Program

Processor 1

Parallel
Parallel

Processor 2

Parallel
Parallel

Processor 3

Parallel
Parallel
Scalability and Serialisation

P1  P2  P3

P  P  P
P  P  P
P  P  P

P1  P2  P3

P  P  P
P  P  P
P  P  P

S  S  S

P  P  P
P  P  P
P  P  P

P1  P2  P3

P  P  P
S  S  S
S  S  S

P  P  P
P  P  P
P  P  P
Scalability and Serialisation

Remember Amdahl’s law

- Serial (non-parallel) portion: when application not running on all cores
- Serialisation prevents scalability

\[ T_1 = 1 = (1 - P) + P \]
\[ T_N = (1 - P) + \frac{P}{N} \]
\[ S(N) = \frac{\frac{T_1}{T_N}} = \frac{1}{(1 - P) + \frac{P}{N}} \]
\[ S(\infty) \rightarrow \frac{1}{(1 - P)} \]
Serialisation

Where does serialisation show up?
  • Application (e.g. access shared app data)
  • OS (e.g. performing syscall for app) How much time is spent in OS?

Sources of Serialisation

Locking (explicit serialisation)
  • Waiting for a lock \(\rightarrow\) stalls self
  • Lock implementation:
    • Atomic operations lock bus \(\rightarrow\) stalls everyone waiting for memory
    • Cache coherence traffic loads bus \(\rightarrow\) stalls others waiting for memory

Memory access (implicit)
  • Relatively high latency to memory \(\rightarrow\) stalls self

Cache (implicit)
  • Processor stalled while cache line is fetched or invalidated
  • Affected by latency of interconnect
  • Performance depends on data size (cache lines) and contention (number of cores)
More Cache-related Serialisation

False sharing
- Unrelated data structs share the same cache line
- Accessed from different processors
  ➔ Cache coherence traffic and delay

Cache line bouncing
- Shared R/W on many processors
- E.g.: bouncing due to locks: each processor spinning on a lock brings it into its own cache
  ➔ Cache coherence traffic and delay

Cache misses
- Potentially direct memory access ➔ stalls self
- When does cache miss occur?
  - Application accesses data for the first time, Application runs on new core
  - Cached memory has been evicted
    - Cache footprint too big, another app ran, OS ran
Multiprocessor Hardware
Multi-What?

**Terminology:**
- core, die (chip), package (module, processor, CPU)

**Multiprocessor, SMP (Symmetric Multiprocessing)**
- >1 separate processors, connected by off-processor interconnect

**Multicore, CMP (Chip Multiprocessor)**
- >1 processing cores in a single die, connected by on-die interconnect

**Multithread, SMT (Simultaneous Multithreading)**
- >1 hardware threads in a single processing core

**Multicore + Multiprocessor**
- >1 multicore dies in a package (multi-chip module), on-processor interconnect
- >1 multicore processors, off-processor interconnect

**Manycore**
- Lots (>100) of cores
Contemporary Multiprocessor Hardware

Intel:
- Nehalem, Westmere: 10 core, QPI
- Sandy Bridge, Ivy Bridge: 5 core, ring bus, integrated GPU, L3, IO
- Haswell (Broadwell): 18+ core, ring bus, transactional memory, slices (EP)
- Skylake (SP): mesh architecture

AMD:
- K10 (Opteron: Barcelona, Magny Cours): 12 core, Hypertransport
- Bulldozer, Piledriver, Steamroller (Opteron, FX)
  - 16 core, Clustered Multithread: module with 2 integer cores
- Zen: on die NUMA: CPU Complex (CCX) (4 core, private L3)
- Zen 2: chiplets (2xCCX) chiplets, IO die (incl mem controller)

Oracle (Sun) UltraSparc T1,T2,T3,T4,T5 (Niagara), M5,M7
- T5: 16 cores, 8 threads/core (2 simultaneous), crossbar, 8 sockets,
- M8: 32 core, 8 threads, on chip network, 8 sockets, 5GHz

ARM Cortex A9, A15 MPCore, big.LITTLE, DynamIQ
- 4 -8 cores, big.LITTLE: A7 + A15, dynamIQ: A75 + A55
Experimental/Non-mainstream Multiprocessor Hardware

Microsoft Beehive
  • Ring bus, no cache coherence

Tilera (now Mellanox) Tile64, Tile-Gx
  • 100 cores, mesh network

Intel Polaris
  • 80 cores, mesh network

Intel SCC
  • 48 cores, mesh network, no cache coherency

Intel MIC (Multi Integrated Core)
  • Knight’s Corner/Landing - Xeon Phi
  • 60+ cores, ring bus/mesh
Interesting Properties of Multiprocessors

Scale and Structure
- How many cores and processors are there
- What kinds of cores and processors are there (homogeneous vs heterogeneous)

Memory Locality
- Where is the memory

Caches
- What is the cache architecture

Interconnect
- How are the cores and processors connected
- Access to IO, etc.

Communication
- How do cores and processors send messages to each other
- Interrupts
Scale and Structure

ARM Cortex A9 MPCore
- basic structure
- single die
- homogeneous cores

From http://www.arm.com/images/Cortex-A9-MP-core_Big.gif
Scale and Structure

Intel Nehalem – multiprocessor & multicore, homogeneous
Scale and Structure

**Tilera Tile64, Intel Polaris: manycore – simple, homogeneous**

From www.tilera.com/products/processors/TILE64
Scale and Structure

ARM big.LITTLE – multicore, semi-heterogeneous

From http://www.arm.com/images/Fig_1_Cortex-A15_CCI_Cortex-A7_System.jpg
Scale and Structure

Conventional big.LITTLE

- Quad Cortex-A53
- Octa Cortex-A53

DynamIQ big.LITTLE

- 1b+2L
- 1b+3L
- 1b+4L
- 1b+7L

From https://developer.arm.com/-/media/developer/Other%20Images/dynamiq-improvements-over-big-little.png
Scale and Structure

I.MX 6SoloX – multicore: Cortex-A + Cortex-M

Scale and Structure

NVIDIA Parker (Tegra X2) SOC
- Heterogeneity
- Application CPUs
- GPUs
- Management CPUs

From: https://elinux.org/Jetson_TX2
https://www.usenix.org/conference/osdi21/presentation/fri-keynote
Memory Locality

Cortex A9
Uniform Memory Access:
• same access to all memory
Memory Locality

NUMA (Non-Uniform Memory Access)
Cache

Hierarchy
• L1, L2, L3, ...

Sharing
• Private – per core
• Shared – all/some cores
• Partitioned – distributed and shared

Coherence
• No inconsistent values in caches
• At same level, at different levels
• Snooping, directory-based
Cache

**ARM Cortex A9 MPCore**

- **L1** – private, split, coherent, optimised MESI
- Optional **L2** – shared
- **DMA cache** coherent with **L1** (ACP)

From [http://www.arm.com/images/Cortex-A9-MP-core_Big.gif](http://www.arm.com/images/Cortex-A9-MP-core_Big.gif)
Cache

Core: L1, L2. Socket: L3. Cache coherent between sockets
Cache

Oracle Sparc T2 (Niagara 2)
- private L1
- partitioned L2
- all cores equal access to L2s
Cache

Intel MIC (Multi Integrated Core) (Knight’s Corner/Landing - Xeon Phi)

- Private L2
  - Tag Directory – info about addresses in other L2s
  - Send messages to other cores to access their L2
Cache

Intel SCC – no hardware cache coherence
Interconnect

Oracle Sparc T2

- Crossbar switch between cores and L2
- Cores have independent access to L2
- What does that mean for Software?
Interconnect

AMD Barcelona: network
Interconnect (Latency)
Interconnect (Bandwidth)

No direct link between node 0 and 7, 0 will do 2 hops to access 7.

No direct link between node 0 and 7, 0 will do 2 hops to access 7.

Node 6
Node 4
Node 5
Node 1

Node 2
Node 3
Node 7

From https://www.usenix.org/conference/atc15/technical-session/presentation/lepers
Interconnect

Tilera Tile64, Intel Polaris: Mesh network(s)

From www.tilera.com/products/processors/TILE64
Interconnect

Beehive

- Ring
- No hardware cache coherence
Interconnect

Intel MIC (Multi Integrated Core) (Knight’s Corner/Landing - Xeon Phi)

- Multiple rings
  - Directional
  - Data rings
  - Address rings
  - Coherence rings
Interconnect

Haswell EP Die Configurations

From http://www.anandtech.com/show/8423/intel-xeon-e5-version-3-up-to-18-haswell-ep-cores-/4
Cluster on Die (COD) Mode

- Supported on 1S & 2S SKUs with 2 Home Agents (10+ cores)
- In memory directory bits & directory cache used on 2S to reduce coherence traffic and cache-to-cache transfer latencies
- Targeted at NUMA optimized workloads where latency is more important than sharing across Caching Agents
  - Reduces average LLC hit and local memory latencies
  - HA sees most requests from reduced set of threads potentially offering higher effective memory bandwidth
- OS/VMM own NUMA and process affinity decisions
Interconnect

**Skylake SP**
- **Server**
- **Mesh**
  - Array of half-rings
- **Sub-NUMA clustering** (replacing CoD)
  - separate memory domains
- **Per core LLC slice**
  - Directory based coherency

From https://itpeernetwork.intel.com/intel-mesh-architecture-data-center/
Communication

Inter-processor interrupts
• Intel: through APIC
• ARM: SGI (software generated interrupts) through GIC – interrupt routing.
• Slower than cache coherency (10-100x)

Shared memory
• Rely on cache coherency
• Polling and atomic operations
• (ab)use cache lines for communication
Communication

Intel SCC – explicit message passing buffer
Communication

Device interrupts

• Interrupt affinity
• Route interrupts to specific cores
• ARM: GIC
  • Generic Interrupt Controller
• X86: APIC
  • Advanced Programmable Interrupt Controller

From:
- https://developer.arm.com/documentation/198123/0302/What-is-a-Generic-Interrupt-Controller-
- Towards Correct-by-Construction Interrupt Routing on Real Hardware, PLOS 2017
Summary

Scalability
- 100+ cores
- Amdahl’s law really kicks in

Heterogeneity
- Heterogeneous cores, memory, etc.
- Properties of similar systems may vary wildly (e.g. interconnect topology and latencies between different AMD platforms)

NUMA
- Also variable latencies due to topology and cache coherence

Cache coherence may not be possible
- Can’t use it for locking
- Shared data structures require explicit work

Computer is a distributed system
- Message passing
- Consistency and Synchronisation
- Fault tolerance