From Multicore to Manycore Architectures-
Current Trends and Operating System Issues

Prof. Dr.-Ing. Jörg Nolte
Lehrstuhl Verteilte Systeme/Betriebssysteme
BTU Cottbus
Introduction

- Multicore CPUs with up to 12 cores are mainstream these days.

- Manycore systems with 100 cores are already sold on the market (Tilera), experimental systems with 48 general purpose cores like the Intel SCC are available for research.

- Systems with more than 1000 cores seem possible in the near future.

- How will they look like?

- Clusters on a chip?

- Shared memory processors?

- What challenges pose these architectures to applications and operating systems?
Shared Memory or Message Passing?

- Conceptually duals with respect to general capabilities (Needham & Lauer 1978)

- Explicit message passing (MPI, mostly data shipping) dominates the HPC community

- Implicit message passing (i.e. function shipping like RPC, RMI) is favored by the distributed system community

- Shared memory (implicit data shipping) for everybody else

- The general acceptance for shared memory architectures is high

- ... but will memory subsystems scale to that numbers?
The Shared Memory Myth

- Numerous processes
- ... happily communicate and cooperate via shared memory
- ... which provides contention free access for everybody
- ... on each shared variable
The Impact of Memory Architectures

- Uniform Memory Access (UMA) Architectures

- ... are easy going for the programmer but simple solutions do not scale well

- Non-Uniform Memory Access (NUMA) Architectures dominate the market

- ... offer better scalability but require location awareness!

- For all practical purposes NUMA architectures are distributed memory systems that just pretend to be shared memory systems!
Small Scale UMA-Architecture

- Single DRAM controller
- Serious bottleneck for applications with low cache hit rate
- Placement of shared memory regions trivial
- ... page coloring suffices for page placement
- Processor affinity dependent on caches only
The Operating System Issue

- Legacy operating systems have been originally designed for single CPU systems (traditional Unix + clones) or small numbers of CPUs (e.g. Windows NT line)

- ... and expect cache coherent memory

- Scalability of OS services is limited by many factors

- ... lock contention on shared OS data structures, bad memory placement strategies, poorly implemented processor affinities and false sharing

- ... are notorious performance killers!

- A scalable MM subsystem is fundamental for applications with low cache hit rates!
Traditional Goals of UMA VM Management

- Use memory **efficiently** through shared memory regions and VM strategies
- Avoid redundant code by **shared text** and **shared libraries** whenever possible
- Provide a **consistent view** of shared memory regions for cooperating processes
- Speed up process creations by **lazy on demand paging** and **copy on write** mechanisms (COW)
- Quickly **resurrect** often accessed but temporarily non used memory mapped files
- **OS data structures** are designed and optimized for these purposes
Free BSD MM Data Structures

- address space management
  - vm_map
    - vm_map_entry
    - vm_map_entry
    - vm_map_entry
  - vm_pmap

- backing store mapping
  - vnode/object
  - shadow object chains
  - vm_page
    - ... to vm_page

- page replacement strategies
  - per frame mapping list
  - page daemon activity lists
  - vm_page
    - ... to vm_page

- multi level MMU tables
- MMU hardware
- TLB
The Cost of Sharing

- Gathering of VM statistics requires finding and heuristic examination of access bits in several page tables of those processes that share a page.

- Eviction of pages requires finding and invalidation of all page table entries of all processes that mapped the page to be evicted.

- ... and all cores where those processes have been running on need to invalidate the respective TLB entries (TLB shoot down e.g. via IPIs).

- Sharing has a substantial overhead, even on old fashioned single CPU systems!
Small Scale NUMA-Architecture

- one DRAM controller per CPU
- HT/Quick Path direct links between CPUs
- better scalability
- placement of shared memory regions tricky
- processor affinity depends on caches and DRAM controller
Large Scale NUMA-Architecture

- **hundreds** of small scale systems
- ... interconnected by a high speed fabric
- **plenty** of combined memory bandwidth
- ... but highly varying memory access latencies
- **complex placement!**
Figure 2. 512-Processor Dual “Fat-Tree” Interconnect Topology
Large Scale NUMA-Architecture
Quick Guide to NUMA Happiness

1. Avoid shared data and atomic operations, if you can!

2. Beware of false sharing and align shared variables to individual cache lines!

3. Use shared data for communication purposes mostly!

4. Copy often accessed variables to CPU-local memory!

5. Generally, program as if you are programming a distributed memory machine!
NUMA Placement Strategies

- Place private pages to the memory controller most close to the accessing CPU

- Place shared pages to the memory controller with the minimal communication path to all CPUs running processes sharing that page (somewhere in the “middle”, make NUMA imitate UMA by averaging access latencies)

- Page migration might be considered (either with COMA HW-support), or via copying of page frames to other memory locations and global remapping)

- Clever placement and affinity scheduling mitigate some of the NUMA peculiarities ...

- ... but where do you place pages shared by everybody (e.g. libc) or the MM related OS data structures?

- Even when communication paths are optimized, the memory bandwidth limitation remains!
Placement NUMA-Architecture

CPU 0

DRAM controller

address space a

CPU1

DRAM controller

address space b

memory mapped file

file image

disk
Placement NUMA-Architecture
Placement, NUMA-Architecture
Placement NUMA-Architecture

CPU 0

DRAM controller

address space a

CPU1

DRAM controller

address space b

memory mapped file

disk

file image

Sonntag, 25. September 2011
Placement NUMA-Architecture

CPU 0
- DRAM controller
- address space a

CPU 1
- DRAM controller
- address space b

disk
- memory mapped file
- file image

Sonntag, 25. September 2011
Placement NUMA-Architecture
Placement NUMA-Architecture
Placement NUMA-Architecture

CPU 0
DRAM controller
address space a

CPU 1
DRAM controller
address space b

memory mapped file
file image
disk
Placement NUMA-Architecture
Placement NUMA-Architecture
Migration NUMA-Architecture

CPU 0
- DRAM controller
- address space a

CPU1
- DRAM controller
- address space b

Memory mapped file
- file image
- disk
A Manycore Architecture

- some similarities to large scale NUMA
- ... but few memory controllers, rather small caches
- low combined memory bandwidth in relation to number of cores
- ... but high bandwidth, ultra-low latency networks built into the chip
Page Replication on Manycores

- Memory bandwidth is optimized, since heavily accessed pages can be replicated over all available memory controllers.

- Very effective for read-only or read-mostly pages.

- Requires effectively page-based DSM protocols to ensure consistency for pages that can be modified.

- Generally costly, but implementable with much lower overhead than traditional DSM systems because of on-chip ultra-low latency networks.

- Number of replicas generally low, since data is replicated over memory controllers not address spaces.
Page Replication
Page Replication

Diagram showing the relationship between CPUs, DRAM controllers, address spaces, memory-mapped files, and disk images.
Page Replication

CPU 0
DRAM controller
address space a

CPU1
DRAM controller
address space b

memory mapped file
file image
disk
Page Replication Reading Access

CPU 0
DRAM controller
address space a
replicated page frame

CPU 1
DRAM controller
address space b
memory mapped file

disk
file image

Sonntag, 25. September 2011
Page Replication Writing Access CPU1
Page Replication Strategies

- **Trivial** for read-only pages (i.e. shared libraries)
- Again **tricky** for pages, that can be **modified**
- **Modification rates** need to be determined
- ... and might **change** as well!
Granularity of Replicated Data

- **Coarse-grained** on the level of mapped regions

- ...e.g. **replicate everything** for readonly files and fetch pages from other memories if possible

- **Medium-grained** on page level, decision based on individual **access statistics** and modification rate

- **Fine-grained** on cache line level, decision **directly** based on access, typically implemented in hardware

- **Problem-oriented**, decision based on **application knowledge**, typically implemented in software
Hardware Cache Coherence or Not?

- Cache coherence is rather simple to implement on small systems.

- ... snooping on busses or broadcasts on small point-to-point networks (such as 4-way systems).

- Large scale systems require substantial hardware effort to implement distributed directories with rather large copy maps.

- ...typically leading to larger cache lines and more false sharing effects.

- Is this worth the effort?

- ... or are software based solutions the better choice?

- Ultra-fast on chip networks might pave new ways!
Memory Consistency Models

• A “zoo” of models: from sequential consistency to release consistency

• ... from hardware based cache coherence to page-based DSM

• Common denominator: some kind of replication control protocol

• ... that manages varying sets of replicas

• ... which are collectively addressed to be invalidated, updated or merged

• ... to implement some kind of shared view

• A group abstraction in combination with collective operations eases these tasks considerably!
TACO Topologies and Collections

• Template library that implements a **global object space** on top of a partitioned address space model (PGAS)

• **Global object pointers** and template based RMI-Mechanisms

• Method calls can be applied to entire **distributed object groups**

• Highly **concurrent**, unlike MPI’s collective communication mechanisms, processes **do not need** to participate explicitly in the same operation

• ... has been **ported** to various **cluster communication layers**

• ... and recently to the **Intel SCC**
Intel SCC, Overview

- experimental processor, no product!

- 24 tiles with 2 P54C cores each, 48 cores total, 533/800MHz clock

- L1 16+16, L2 256KiB, no cache coherence!

- mesh network built into the chip, 1.6GHz clock

- communication via 16KiB SRAM shared by cores on a tile
Intel Single Chip Cloud computer

• Cluster on a chip?

• Shared memory machine?

• The SCC is a hybrid system, that represents both aspects

• ... but always requires distributed software control with respect to caches!

• Ultra-low latency communication facilitates fast function shipping (RMI) between cores

• No cache coherence issues and improved cache utilization without cache thrashing on shared data!
TACO RMI/RMA Timing on SCC (800MHz)

- Always “receive from any”
- ... each receiver needs to poll 48 entries
- Advantage when writing and polling N_i and A_i have the “right” timing
- Time difference reflects extra cache line reads
Reduction: Parallel Visitor Pattern
An Example Tree Topology for 48 Objects

Sonntag, 25. September 2011
Map Operation: Infiniband vs SCC
Reduction: Infiniband Cluster vs SCC

![Graph showing the comparison between Infiniband Cluster and SCC with respect to group size and reduce completion time. The graph includes multiple lines representing different system topologies and configurations.]
Conclusion

• Manycore systems are hybrids between clusters on a chip and shared memory systems with a low combined memory bandwidth and small per core caches

• Replication of data over available memory controllers increases memory bandwidth effectively

• Flexible group abstractions in combination with ultra-Low-latency NOCs facilitate efficient software controlled consistency control protocols

• Function shipping (RMI) is likely to increase cache utilization and avoids cache thrashing effectively in the case of highly contended shared data

• Operating systems should be designed as distributed systems and provide flexible memory consistency models for legacy applications