SCALABILITY AND HETEROGENEITY

Nils Asmussen

Dresden, 12/06/2022
Commodity System with GPU

- Application
- Runtime
- OS Kernel
- Compute Kernel
- CC
- Interconnect
- Core
- GPU
- Mem
- HLSL
- OpenCL
- Driver

Scalability and Heterogeneity Slide 3 of 41
Current Trend

Scalability and Heterogeneity Slide 4 of 41
Why?

- More cores can (for some usecases) deliver more performance
- Specialization is the next step
- Cache coherency gets more expensive (performance, complexity and energy) with more (and heterogeneous) cores
Commodity Hardware
Non-Uniform Memory Access

- Core-to-RAM distance differs
- Various interconnect topologies: bus, star, ring, mesh, ...
- The good: all memory can be directly addressed
- The bad: different access latencies
- Consider placement of data
Measuring NUMA effects on:

Figure 3.1: Dell Precision T7500 System Overview
<table>
<thead>
<tr>
<th>Operation</th>
<th>Access</th>
<th>Time</th>
<th>NUMA Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>local</td>
<td>37.420s</td>
<td>1.000</td>
</tr>
<tr>
<td>read</td>
<td>remote</td>
<td>53.223s</td>
<td>1.422</td>
</tr>
<tr>
<td>write</td>
<td>local</td>
<td>23.555s</td>
<td>1.000</td>
</tr>
<tr>
<td>write</td>
<td>remote</td>
<td>23.976s</td>
<td>1.018</td>
</tr>
</tbody>
</table>
• fundamental options: migrate thread vs. migrate data
• use performance counters to decide
• dynamic management shows > 10% performance benefit compared to best static placement
Scalability and Heterogeneity

Slide 12 of 41
Research Prototypes
• Concept: multikernel, implementation: barrelfish
• Treat the machine as cores with a network
• “CPU driver” plus exokernel-ish structure
• No inter-core sharing at the lower levels
• Monitors coordinate system-wide state via replication and synchronization
Barrelfish


Scalability and Heterogeneity
• Based on Barrelfish
• Introduces abstractions for non-CC systems
• Takes advantage of CC, if possible
• Otherwise, data transfers via, e.g., DMA units
• Used to implement OS services (net, fs, ...)
Barrelfish + Cosh

Scalability and Heterogeneity
Barrelfish Scalability

- Driven by scalability issues of shared kernel designs and cache coherence
- This might not be a pressing issue today

Factored Operating System

- Idle Processor Core
- Application
- Fleet of File System Servers
- Fleet of Physical Memory Allocation Servers

• Idea: multiple Linux’s on one system
• Provide the illusion of an POSIX SMP system
• Kernels communicate to sync/exchange state
• Does not rely on global shared memory
• Distributed shared memory, if necessary
• Processes can migrate between kernels
Scalability and Heterogeneity
Popcorn Linux

Scalability and Heterogeneity Slide 22 of 41
• Idea: heterogeneous ISA systems need some kind of compiler support
• ISA-specific kernels: “satellite kernels”
• Provide uniform OS abstractions
• Memory management, scheduling
• Bootstrap: first kernel becomes coordinator, boots other cores
• Share-nothing, even on ccNUMA
• Processes cannot span across kernels
• Implementation based on Singularity
• Applications compiled into intermediate code
• 2nd stage compilation to native code of all available ISAs at install time
• Placement based on affinity hints
Coordinator kernel

x86

Satellite kernel

XScale Programmable Device
Our Own Work
M³ Approach – Software

Asmussen et al.: M3: A Hardware/OS Co-Design to Tame Heterogeneous Manycores, ASPLOS 2016

Scalability and Heterogeneity
Data Transfer Unit

- Supports memory access and message passing
- Provides a number of endpoints
- Each endpoint can be configured for:
  1. Accessing memory (contiguous range, byte granular)
  2. Receiving messages into a ringbuffer
  3. Sending messages to a receiving endpoint
- Configuration only by kernel, usage by application
- Direct reply on received messages
• M³: Microkernel-based system for het. manycores (or L4 ±1)

• Implemented from scratch

• Drivers, filesystems, . . . are implemented on top

• Kernel manages permissions, using capabilities

• DTU enforces permissions (communication, memory access)

• Kernel is independent of other tiles in the system
• M³ kernel manages user tiles in terms of *activities*
• An activity is comparable to a process
• Activity creation yields a activity cap., loading not done by kernel
• Library provides primitives like *fork* and *exec*
• Activities are used for *all* tiles:
  – Accelerators are not handled differently by the kernel
  – All activities can perform system calls
  – All activities can establish communication channels to others
  – …
• Used for all file-like objects
• Simple for accelerators, yet flexible for software
• Software uses POSIX-like API on top of the protocol
• Server configures client’s memory endpoint
• Client accesses data via DTU, without involving others
  • req(in) requests next input piece
  • req(out) requests next output piece
  • Receiving resp(n, 0) indicates EOF
### Example Implementations

#### Filesystem

- *m3fs* is an in-memory file system
- m3fs organizes the file’s data in extents
- Extent is contiguous region defined by start and length
- `req(in/out)` configures memory endpoint to next extent
Example Implementations

**Filesystem**
- *m3fs* is an in-memory file system
- *m3fs* organizes the file’s data in extents
- Extent is a contiguous region defined by start and length
- `req(in/out)` configures memory endpoint to next extent

**Pipe**
- M^3^ provides a server that offers UNIX-like pipes
- Data is exchanged via shared memory area
- Client’s memory EP is configured once for SHM
- Pipe server tells clients read/write positions within SHM area
• Accelerator works on scratchpad memory
• Input data needs to be loaded into scratchpad
• Result needs to be stored elsewhere
• Accelerator works on scratchpad memory
• Input data needs to be loaded into scratchpad
• Result needs to be stored elsewhere
Shell Integration

- $M^3$ allows to use accelerators from the shell:
  \[ \text{preproc} \mid \text{accel1} \mid \text{accel2} \rightarrow \text{output.dat} \]
- Shell connects the EPs according to stdin/stdout
- Accelerators work autonomously afterwards
- Requires about 30 additional lines in the shell
Prototype Platforms – FPGA

Asmussen et al.: Efficient and Scalable Core Multiplexing with M³v, ASPLOS 2022
Scalability and Heterogeneity

Slide 37 of 41
Prototype Platforms – gem5

- Modular platform for computer-system architecture research
- Supports various ISAs (x86, ARM, Alpha, …)
- Cycle-accurate simulation
- Has an out-of-order CPU model
- We built a DTU for gem5 and integrated accelerators
gem5 – Example Configuration

Scalability and Heterogeneity

Slide 39 of 41
Scalability and Heterogeneity Slide 40 of 41

This slide illustrates the components of a system architecture, focusing on scalability and heterogeneity. The diagram is divided into several sections:

- **Core and Mem**: These represent the core components of the system, possibly including processing and memory resources.
- **Interconnect**: Indicates the network or communication infrastructure connecting various parts of the system.
- **Srv**: Likely refers to server components, handling services and data exchange.
- **App**: Represents application layers, possibly responsible for user-facing operations.
- **Messages**: Indicates the flow of information or data messages between components.
- **Acc**: Could stand for access or additional services, enhancing the functionality of the system.

The diagram emphasizes the flow of data and messages, highlighting the integration of these components to achieve scalability and handle heterogeneity effectively.
• Various different approaches
• Not clear yet how to handle heterogeneity
• Memory will get heterogeneous as well (NVM)
• Reconfigurable hardware will emerge