The Trouble with Hardware

Timothy Roscoe
Systems Group, ETH Zurich
November 30th 2017
Many thanks!

And all the ETH Systems Group!
The Gap.

For many commercially relevant workloads, cores spend much of their time in the OS.

**BUT:**

- Processor architects ignore OS designers
  - Simply don’t understand the OS problem
  - Cores rarely evaluated with >1 app running anyway
- HPC people try to remove the OS
  - And then blow the rest of their s/w development budget putting it back in a user library.
- and OS design people?
  - Complain among themselves and try and deal with it
  - Don't even try to influence hardware
SO, WHAT IS THE TROUBLE WITH HARDWARE?
Lies we teach our children

Figure 2.1  A modern computer system.
Lies we tell our children

systems, but all systems have a similar look and feel. Don’t worry about the complexity of this figure just now. We will get to its various details in stages throughout the course of the book.
A great way to frighten students

TI OMAP4460:
Kindle Fire 7"
Samsung Galaxy Nexus, etc.
A great way to frighten students

- pp. 3-63: Table of contents
- pp. 64-88: List of figures
- pp. 89-258: List of tables

30/11/2017
24.5 MMC/SD/SDIO Programming Guide

24.5.1 Low-Level Programming Models

24.5.1.1 Global Initialization

24.5.1.1.1 Surrounding Modules Global Initialization

This section identifies the requirements for initializing the surrounding modules when the module must be used for the first time after a device reset. This initialization of surrounding modules is based on the integration and environment of the MMC/SD/SDIO modules. For more information, see Section 24.3, MMC/SD/SDIO Integration, and Section 24.2, MMC/SD/SDIO Environment.

<table>
<thead>
<tr>
<th>Surrounding Modules</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRCM</td>
<td>Module interface and functional clocks must be enabled. See Chapter 3, Power, Reset, and Clock Management.</td>
</tr>
<tr>
<td>Control module</td>
<td>Module-specific pad mixing and configuration must be set in the control module. See Chapter 18, Control Module.</td>
</tr>
<tr>
<td>(optional) MPU INTC (or DSP INTC)</td>
<td>MPU INTC configuration must be done to enable the interrupts from the MMCHS module. See Chapter 17, Interrupt Controllers.</td>
</tr>
<tr>
<td>(optional) sDMA (or dDMA)</td>
<td>DMA configuration must be done to enable the module DMA channel requests. See Chapter 16, sDMA.</td>
</tr>
<tr>
<td>(optional) Interconnect</td>
<td>For more information about the interconnect configuration, see Chapter 13, Interconnect.</td>
</tr>
</tbody>
</table>

**NOTE:** The MPU/DSP INTC and the sDMA/dDMA configurations are necessary if the interrupt and DMA-based communication modes are used.

24.5.1.2 MMC/SD/SDIO Host Controller Initialization Flow

Table 24-23 shows the general boot process.

<table>
<thead>
<tr>
<th>Step</th>
<th>Access Type</th>
<th>Register/Bit Field/Programming Model</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize clocks.</td>
<td></td>
<td>See Section 24.5.1.1.2.1.</td>
<td></td>
</tr>
<tr>
<td>Software reset of the controller.</td>
<td></td>
<td>See Section 24.5.1.1.2.2.</td>
<td></td>
</tr>
<tr>
<td>Set module hardware capabilities.</td>
<td></td>
<td>See Section 24.5.1.1.2.3.</td>
<td></td>
</tr>
<tr>
<td>Set module idle and wake-up modes.</td>
<td></td>
<td>See Section 24.5.1.1.2.4.</td>
<td></td>
</tr>
</tbody>
</table>

24.5.1.1.2.1 Enable Interface and Functional clock for MMC Controller

Before any MMCHS register access, the MMCHS interface clock and functional clock in the PRCM module registers must be enabled. See Section 3.6.10.4, Clock Domain Module Attributes, in Chapter 3, Power, Reset, and Clock Management.

Table 24-22. Global Initialization of Surrounding Modules
Is this a computer?

Still programmed mostly as a classical distributed system.
Typical rack-scale architecture (in research, at least)

Q. Message latency in this network?
A. \(~ 0.7 \mu s, \) or \(~ 10 \) LLC misses.

\Rightarrow \text{ need to think of this as one big machine}
But it’s not typical.
But it’s not typical.
So, hardware is too complicated.

What clever techniques do OS developers use to mitigate this problem?

1. The powerful high-level abstractions of C
2. Kernel **modules**
3. Server processes and **daemons**
4. Er...
5. That’s it.
The Barrelfish research OS

- Written from scratch, 2008-present
  - Industry help: Microsoft, HPE, Huawei, Oracle, Cisco, VMware, Xilinx, ARM, Cavium, Intel, ...
- Used for research and teaching
- Currently ~ 1m lines of code
  - MIT open source licence
  - ARMv7-A, ARMv8, x86_64, KNL
  - Previously: ARMv5, ia32, SCC, Beehive
  - See [www.barrelfish.org](http://www.barrelfish.org)
WHAT DID WE LEARN FROM BARRELFISH?
Learnings

We started trying to solve three challenges:

• **Scaling** to large core counts
• **Dynamic, heterogeneous** cores
• **Complex memory** hierarchies

We ended up identifying two big problems:

• Sheer **complexity** of hardware for software
• **Ossification** of hardware/software ecosystem
Hardware is changing faster than system software

- CAD systems make it easy to cut’n’paste
- High volumes for SoCs lead to diversity
- End of Dennard scaling $\Rightarrow$ specialism
System software is getting harder to change

• The OS retreats from most of the hardware
  – Heterogeneous cores?
  – Non-cache-coherent memory?
  – etc.

• OS can’t adapt to changing tradeoffs
  – “Least common denominator” tuning

• Today Linux is at best a small component of “that which manages the machine”
1\textsuperscript{st} TRY: LET’S REPRESENT THE MACHINE IN PROLOG
SKB – System Knowledge Base

- Basic OS service
  - Boots early
- Holds:
  - Hardware info
  - Runtime state
- Queried by:
  - OS services
  - Applications
- Rich semantic data model
Plenty of design options

- Knowledge-representation frameworks
- Database
- RDF
- Logic Programming, inference
- Description Logics
- Satisfiability Modulo Theories
- Constraint Satisfaction
- Optimization
- etc.

Initial choice: ECLiPse CLP solver: Prolog + constraint extensions (circa 2009!)

30/11/2017
A few SKB applications

• General name server / service registry
• Coordination service / lock manager
• Device management
  – Driver startup / hotplug
• PCIe bridge configuration
  – A surprisingly hard CSAT problem!
• Intra-machine routing
  – Efficient multicast tree construction
• Cache-aware thread placement
  – Used by e.g. databases for query planning
Example 1: PCIe bridge configuration

- Hierarchical allocation of physical address ranges
  - Natural power-of-two aligned
  - Three disjoint memory types
  - “Holes” in physical address space
  - Some devices can’t be moved
  - Odd constraints on some address mappings
  - “Quirks”
  - HotPlug
  - ...

Adrian Schüpbach, Simon Peter, Andrew Baumann

30/11/2017
How do others deal with this?

- Mostly, they don’t.

- Linux uses BIOS allocation and runs a fixup procedure
  - Configures missing devices if no bridge reprogramming needed
  - Otherwise fails

- Windows Vista, Server 2008: PCI Multi-Level Rebalance
  - Can move bridges to a place with bigger free space
  - Machine may appear to freeze for a few seconds

  - Recursive bottom-up algorithm to allocate resources

- People have published genetic algorithms for this problem (!)

- 25 years after PCI 1.0 standardization, no complete solution exists.
We coded it in constraint Prolog (CLP)

• Cleanly separate:
  1. “Ideal” allocation computation (in Prolog)
  2. Ad-hoc constraints (errata, quirks, etc.)
  3. Register read/write code (in C)

• A new quirk is 1-4 lines of portable Prolog

• CLP boots before devices
  – Runtime milliseconds vs. microseconds

30/11/2017

In 2012, we published a TOCS paper on how to configure a 20-year old hardware standard.
Example 2: Database thread placement

• Problem:
  – Place 4 threads of a join operator

• Parameters:
  – Selectivity of join
    • Low selectivity ⇒ share caches for locality
    • High selectivity ⇒ use more caches for speed
  – Inter-cache latency
  – Size of L1 cache
  – Size of (shared) L2 cache
Deployment suggestions on different machines

**AMD Magny Cours**

(a) No preference for cache-sharing

(b) No cache-sharing

For joins: depends on selectivity of the database.

30/11/2017
Deployment suggestions on different machines

Intel Nehalem - EX

(a) No preference for cache-sharing

(b) No cache-sharing

30/11/2017
Deployment suggestions on different machines

### AMD Barcelona

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2GB</td>
<td>2GB</td>
<td>2GB</td>
<td>2GB</td>
</tr>
</tbody>
</table>

#### (a) No preference for cache-sharing

#### (b) No cache-sharing
Deployment suggestions on different machines

AMD Shanghai

(a) No preference for cache-sharing

(b) No cache-sharing
OK...

• We can do a better job of reasoning about hardware inside the OS.
• Simplifies programming
• Gives richer API
• Improves application performance

• But, it’s still a programming tool.
2\textsuperscript{ND} TRY: FORMALLY SPECIFY SEMANTICS OF HARDWARE
Current work!

• Describe formally the hardware 
  *as seen by software*

• Generate code, data, and proofs
  – Header files
  – SKB facts
  – Consistent (re)configuration code via 
    *Program Synthesis*

• Long-term goal: metrics for *tastefulness*
Key principles

1. Don’t **idealize** the hardware in any way.
2. Don’t **exclude** any “difficult” hardware.

*Embrace the mess, and stare into the abyss!*
A closer look at the OMAP4460
A closer look at the OMAP4460

6+ hetero. cores
A closer look at the OMAP4460

- 6+ hetero. cores
- shared + private memory

30/11/2017
A closer look at the OMAP4460

- 6+ hetero. cores
- shared + private memory
- 5+ Interconnects

30/11/2017
A closer look at the OMAP4460

- 6+ hetero. cores
- shared + private memory
- 5+ Interconnects
- Devices on different buses

30/11/2017
A closer look at the OMAP4460

6+ hetero. cores

shared + private memory

5+ Interconnects

Devices on different buses

interrupt subsystem

30/11/2017
There is no uniform view of the system from all cores
Writing correct software...

... means getting all this **right**

- C code is frequently wrong.
- Nice to generate this code, but from what?

- **Proving** software correct requires a **specification** of the hardware
  - But what would it look like?
  - What could be generated from it?
Model addresses as a *decoding net*

\[
\text{net}_s := [\mathbb{N}\ \text{is node}_s, \ \mathbb{N}..\ \mathbb{N}\ \text{are node}_s, \ ... \ ] \\
\text{node}_s := (\text{accept} [\text{block}_s, \ ... \ ])? (\text{map} [\text{map}_s, \ ... \ ])? (\text{over } \mathbb{N})? \\
\text{map}_s := \text{block}_s \text{ to } [\mathbb{N} \ (\text{at } \mathbb{N})?, \ ... \ ] \\
\text{block}_s := \mathbb{N} - \mathbb{N}
\]

Translates a block of addresses to a set of nodes at potential different addresses

Overlays another node (1-to-1 mapping)

Note: can be cyclic!
Representing address decoding

$V_{A9:0}$ is map [200003/12 to $P_{A9:0}$ at 800003]

$P_{A9:0}, P_{A9:1}$ are map [401383/12 to GPT at 0] over $L3$

$P_{DSP}$ is map [1d3e3/12 to GPT at 0] over $L3$

$V_{M3}, V_{M3}$ are over $LI_{M3}$

$RAM_{M3}$ is accept [550203/16]

$ROM_{M3}$ is accept [550003/14]

$MIF$ is map [0 – ffffffff to $L2_{M3}, 550003/14$ to $RAM_{M3}, 550203/16$ to $ROM_{M3}$]

$L3$ is map [490003/24 to $L4$ at 401003, 550003/12 to $MIF$] accept [800003/30]

$L4$ is map [490383/12 to GPT at 0]

$GPT$ is accept [0/12]

$L1_{M2}$ is map [028 to MIF]

$L2_{M3}$ is map [030 to $L3$ at 800003]

$V_{A9:1}$ is map [200003/12 to $P_{A9:1}$ at 800003]

$V_{DSP}$ is over $P_{DSP}$

Isabelle
Interrupts actually the same

---

SDMA is map [0 to SPIMap at 12 to INTC at 18 to NVIC0 at 18 to NVIC1 at 18,
1 to SPIMap at 13 to INTC at 19 to NVICM3:0 at 19 to NVICM3:1 at 19,
2 to SPIMap at 14 to NVIC0 at 20 to NVIC1 at 20,
3 to SPIMap at 15 to NVIC0 at 21 to NVIC1 at 21]

GPT5_int is map [0 to SPIMap at 41 to INTC at 41]

GIC is map [44 – 45 to IFA9:0 at 44, 46 – 47 to IFA9:1 at 46, …]

A90 is map [0 to IFA9:1 at 0]

T0 is map [0 to IFA9:0 at 29]

M3_MMU is map [0 to SPIMap at 100]

INTC, NVIC* are accept []

A91 is map [0 to IFA9:0 at 0]

T1 is map [0 to IFA9:1 at 29]

SPIMap is map [0 – 987 to GIC at 32]

IF* are accept [0 – 1020]
Can capture functionality of:

- Cores and DMA engines
- Caches (both physical and virtual)
- Firewalls
- MMUs and IOMMUs
- Lookup tables
- Interrupt controllers
- Virtualization hardware
What can you do?

Platform model

Theorem prover

Proofs

Correct compiler

Header files

Prolog facts

Runtime assertions

FPGA circuit

checking software

Program sketch of device

Program synthesis

Static h/w config

or

Code for dynamic config
Long-term goal

• A language for describing hardware platforms
  – c.f. ARM’s specification language
• Assemble descriptions of many devices
• Devise complexity metrics
• Create a style guide for hardware designers

What is “tasteful hardware design”?
THE BIGGER TROUBLE WITH HARDWARE
A deadly embrace

Commodity hardware is designed for current, conventional application workloads over Linux.

Academic research (and industrial innovation) in system software is constrained by available commodity hardware.
But hardware is easy to build

- Hardware is so complex and diverse because it’s so easy to build what you want
  - High-end CAD systems
  - Simulators and emulators, FPGAs
  - Rapid fabrication of boards and ASICs
- Big companies do
  - HPE’s The Machine
  - Oracle RAPID, SPARC M7
  - Amazon F1
  - Microsoft Catapult
  - Google TPU for Tensorflow
Challenge for research

Feasible hardware design space

Scope of most systems software research

Available COTS hardware

Specialized product hardware designs
SO, WHAT CAN WE DO?
What if we had...

- A hardware *research* platform for system software
  - Massively *overengineered* wrt. products
  - Highly *configurable* building block for rackscale
- Perhaps we can *actually build it* at ETH...
  - Logical next platform for our research
  - Seed to other universities for impact
Sketch: the basic building block

Large server-class SoC

High-end FPGA

SATA, PCIe, UART, USB

Lots of network bandwidth

Lots of DDR

Lots of DDR and/or HBM

Coherence

30/11/2017
Enzian v.1

Cavium ThunderX-1
48 x ARMv8-a processor

128 GB DDR4

2 x 40 Gb/s Ethernet

Xilinx VCU9P
UltraScale+ FPGA

8 lanes CCPI
10 GB/s

128 GB DDR4

2 x 100 Gb/s QSFP28

PCIe, USB, UART, etc.

PCIe, USB, UART, etc.
Enzian v.2
(November 2017)

**Cavium**
EBB88 Evaluation board

- **2 x 40 Gb/s Ethernet**
- Cavium ThunderX-1 48 x ARMv8-a processor
- 128 GB DDR4

**Xilinx**
VCU118 Evaluation board

- 24 lanes CCPI 30 GB/s
- Xilinx VCU9P UltraScale+ FPGA
- 128 GB DDR4

**Enzian v.2**
(November 2017)

**Cavium**
ThunderX-1 48 x ARMv8-a processor

- 128 GB DDR4

**Xilinx**
VCU118 Evaluation board

- 128 GB DDR4

23 Nov 2017
Enzian v.3 (2018)

Cavium ThunderX-1
48 x ARMv8-a processor

PCIe, USB, UART

CCPI 30 GB/s

I²C + GPIO

BMC

256 GB DDR4

Xilinx VCU9P
UltraScale+ FPGA

128 GB DDR4

64MB HMC

6x10GB/s

Pluggable

PCIe

NVMe

BMC

Single board

23 Nov 2017

AOS
All kinds of uses for this...

- Plug lots together for rack-scale computing
- Use the FPGA for data processing offload
- A better NetFPGA, or bump in the wire
- FPGA support infrastructure
- Sequester processors using the FPGA
- Runtime verification of program trace
- Experiment in scaling coherency etc.

You can probably think of more...

30/11/2017
Summary 1: Hardware is easy to build

- It is complex, diverse, and changes rapidly
- It is hard to program in C
- It has totally unspecified semantics
- OS researchers need to up our game

www.barrelfish.org
Summary 2:
COTS hardware is unrealistic

• All the product action uses custom hardware
• Vendors can build almost anything
• What to build is an economic question
  – ⇒ not something we can answer
• We need overengineered research platforms
  – Our goal should be to deliver options and techniques
Many thanks!