The Case for Labeled von Neumann Architecture (LvNA)

Yungang Bao

April, 2017

Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS)
Agenda

- Background
- Challenges
- Opportunities
- Our Efforts
- Summary
We are in the Cloud Era

Datacenter processing accounts for 50~60%

User Perceived Latency

Datacenters: The Giant Game

• I claim there really are almost no companies in the world, just a handful, that are really investing in scaled public cloud infrastructure.

• We have something over a million servers in our data center infrastructure. Google is bigger than we are. Amazon is a little bit smaller. ... So the number of companies that really understand the network topology, the data center construction, the server requirements to build this public cloud infrastructure is very, very small.

—Steve Ballmer, Microsoft’s former CEO, 2013
Microsoft’s 15B USD Bet

15 billion dollar bet

Alibaba’s 3B USD Datacenter
Utilization is LOW

- Survey of Gartner/McKinsey\textsuperscript{[1,2]}: 6\%\textasciitilde12\%
- Amazon AWS Average CPU Utilization\textsuperscript{[3]}: 7\%\textasciitilde17\%

\textsuperscript{[1]} http://www.gartner.com/newsroom/id/1472714.
\textsuperscript{[3]} Huan Liu, A Measurement Study of Server Utilization in Public Clouds, 2011.
Sharing improves utilizations

Hardware
- CPU
- Memory
- Disk
- Network

Random app #2
CPU intensive job
Random MapReduce #1
Bigtable tablet server
Various other system services
File system chunks server
Scheduling system
Linux
Google’s Solutions

- **Batch-Workload Data Center**
  - Highly shared

**Software Optimization**

- Borg, cgroup,
  - backup request
- LXC, priority,
  - sync-backup-tasks

---

**Google Datacenters Utilization: (Jan-Mar, 2013)**

**Online Service**

- 30%

**Batch Workload**

- 75%

---

Why not increase to 75%?

An example: Memcached
- CPU: 30% → 70%
- Response time >10X, user experience↓
Challenges

• A Tradeoff between

Resource Utilization

User Experience
Response Time is Money

- Search
  Response time
  0.4s → 0.9s

- Ad revenue
  reduces by 20%

Google's Marissa Mayer: Speed wins

Summary: Marissa Mayer of Google gave a testimonial to speed. Her key insight for the crowd at the Web 2.0 Summit is that "slow and steady doesn't win the race." Speed is a huge component and big market driver of Web 2.0, she said.

In testing out the user interface for Google search, Mayer said that with more results for a query, users were spending less time on the site. It turned out that the cause wasn't just the paradox of choice—paralyzed by too many choices—but the fact that a page with 10 results was half a second faster than the page with 30 results. So, Google set about making the page with more results faster, and the rest is history.
Google’s Efforts in Software Stack

Borg,
Linux Container,
Cgroups,
Backup Requests,
Priority,
Sync-back-tasks,
……

Long Tail Latency

• Average latency of most requests is 60-70ms, but the tail latency can be 1800ms (~30X)

More Hardware Support Needed

Modern challenges in CPU design

- Isolating programs from each other on a shared server is hard
- As an industry, we do it poorly
  - Shared CPU scheduling
  - Shared caches
  - Shared network links
  - Shared disks
- More hardware support needed
- More innovation needed
von Neumann Bottleneck

John von Neumann

Von Neumann Bottleneck

John Backus
CPU-Memory Gap

- Memory Wall
- Increase memory hierarchy
Memory Hierarchy

- On-Core vs. Un-Core
Sharing -> Interference

• Cache sharing causes performance degradation

The sharing problem in Google

- **Dynamicity**: Different mixtures cause different performance degradation
- **Poor QoS**: Latency-critical workloads suffer from longer response time

---

[Yang et.al. ISCA ’13] Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computer

[Kambadur et.al SC’12] Measuring Interference Between Live Datacenter Applications
Hard to Predict

Google

- Unpredictable short jobs
- Test-and-debug

- B’s one-second burst cause A’s five-min degradation


S. Yang et al. Split-Level I/O Scheduling, SOSP, 2015.
• Background
• Challenges
• Opportunities
• Our Work
• Summary
Intel Resource Director Technology

• In April 2016, Intel released Resource Director Technology (RDT) that support QoS
  – Cache Monitoring Technology (CMT)
  – Cache Allocation Technology (CAT) [HPCA’16]
  – Memory Bandwidth Monitoring (MBM)
NFV w/o CAT

- UC Berkeley’s Experimental results of CAT for network function virtualization (NFV)
- \textbf{w/o CAT} : throughput degrades by \textbf{51%}

http://span.cs.berkeley.edu
NFV w/ CAT

• **w/ CAT**: Throughput degrades by <2% when dedicating two ways to a specific NF.

http://span.cs.berkeley.edu
**Contention is Everywhere**

- HyperThread, LLC, DRAM, Network etc.

---

**Lo et al. Heracles: Improving Resource Efficiency at Scale, ISCA, 2015.**
Data Center Era 2010s

- Search, On-line shopping, Cloud computing,…
- Priority, Throughput, Latency, …
- QoS v.s. Utilization

Internet Era 1990s

- HTTP, FTP, VoIP, Stream Media, Game, …
- VoIP, Game, …: Latency-critical
- FTP, VoD, …: Bandwidth-sensitive
- Email: Best Effort
- QoS

Applications sharing infrastructure

Different Requirements

QoS Problem

Separate Online/Offline Service

1994, Integrated services
1998, Differentiated Services
2001, MPLS

Fine-grain solution
Labeling each packet
Labeled Networking

- **Fine-grain**: every packet has a label
- **Semantic Gap**: correlate labels with users’ demand
- **Propagation**: propagate labels in a whole network
- **DiffServ**: process packets differentiately based on labels

MPLS is widely used for VPN and QoS
Arch requires new interfaces

New, high-level interfaces are required to convey programmer and compiler knowledge to the hardware.

Crosscutting Interfaces
Current computer architectures define a set of interfaces that have evolved slowly for several decades. These interfaces—e.g., the Instruction Set Architecture and virtual memory—were defined when memory was at a premium, power was abundant, software infrastructures were limited, and there was little concern for security. Having stable interfaces has helped foster decades of evolutionary architectural innovations. We are now, however, at a technology crossroads, and these stable interfaces are a hindrance to many of the innovations discussed in this document.

Better Interfaces for High-Level Information. Current ISAs fail to provide an efficient means of expressing software intent or conveying critical high-level information to the hardware. For example, they have no way of specifying when a program requires energy efficiency, robust security, or a desired Quality of Service (QoS) level. Instead, current hardware must try to glean some of this information on its own—such as instruction-level parallelism or repeated branch outcome sequences—at great energy expense. New, higher-level interfaces are needed to encapsulate and convey programmer and compiler knowledge to the hardware, resulting in major efficiency gains and valuable new functionality.

Labeled Architecture?
The Computer as a Network

- Hardware components communicate via internal packets, e.g., PCIe packets, NoC packets, QPI packets.
Agenda

- Background
- Challenges
- Opportunities
- Our Work
- Summary
Labeled von Neumann Architecture (LvNA)

- **Fine-grain**: attach a label to each memory and I/O request
- **Semantic-Gap**: correlate labels with VM/Proc/Thread/Var
- **Propagation**: propagate labels in a whole machine
- **Programmable label control logic (CL)**: provide differentiated services based on different label-indexed rules

Goal

w/o loss of QoS

30%

75%
Labeling + CP → Priority Queues

- PriQ can achieve both utilization and QoS

\[ \bar{W} = \frac{\bar{R}}{1 - \rho} \]

\[ W_1 = \frac{\bar{R}}{1 - \rho_1} \]

\[ W_2 = \frac{\bar{R} + \rho_1 \bar{W}_1}{1 - \rho_1 - \rho_2} \]

Low-priority loads cannot affect \( \bar{W}_1 \)
Programmable Architecture for Resourcing-on-Demand (PARD)

Ma et. al, Supporting Differentiated Services in Computers via Programmable Architecture for Resourcing-on-Demand (PARD), *ASPLOS*, 2015
Challenges in Reconstruction

1. How to enforce labeling mechanism?
2. How to design control logics?
3. How to design programming interface?
Challenge #1
How to enable computer hardware to distinguish different applications?

Single Application

- Core
- Core
- Core

Shared Last Level Cache

- I/O Chipset
- Memory Ctrl

Disk
NIC

reality

Hypervisor

- APP0
- APP1
- APPn

Shared Last Level Cache

- I/O Chipset
- Memory Ctrl

Disk
NIC

expect
Propagate Labels in Datapath

Core -> ...

Shared Last Level Cache

I/O Chipset

Memory Controller

Disk

NIC
Propagate Labels in Datapath

Dev -> …
How to User Labels

- Core
- Shared Last Level Cache
- I/O
- Chipset
- Memory Controller
- Disk
- NIC

- DS-id
- Cache Partition

- Rate Limit
- Encryption
- Compress
- ...

Priority-based Scheduling
Challenge #2
How to design control logics for a diversity of hardware?

Control Logic (CL)
CL Design Choices

Table-based

- Simple to implement, Fast
- Inflexible

Processor-based

- Support advanced functionalities
- Complicated, slow
Table-based CL Design

Three Tables + Programming Interface + Interrupt Line

• **Three Control Table**: Parameter / Statistics / Trigger
• **A Programming Interface**: Control Tables R/W
• **A Interrupt Logic**: Send Interrupt when trigger condition meet
Integrate into HW Components

Cache Controller

Memory Controller

Common Control Logic Structure
Challenge #3

How to define/program resourcing-on-demand policy into hardware

Parameter Table

<table>
<thead>
<tr>
<th>DS-id</th>
<th>Param1</th>
<th>Param2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS-id1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS-id2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS-id3</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics Table

<table>
<thead>
<tr>
<th>DS-id</th>
<th>Stat1</th>
<th>Stat2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS-id1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS-id2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS-id3</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trigger Table

<table>
<thead>
<tr>
<th>DS-id</th>
<th>Cond-1</th>
<th>Action-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS-id1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS-id2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Programming Interface

Compare

Policy?
Platform Resource Manager (PRM)

- Augmented IPMI
- Connect all control logics (CLs)
- Run linux-based firmware
- Abstract CLs as files
Access Control Logics

Query Control Logic Info
- `cat /sys/cpa/cpa0/ident`
- `cat /sys/cpa/cpa0/type`

Query Parameters
- `cat /sys/cpa/cpa0/.../parameter/param1`

Setting Parameters
- `echo 10 > /sys/cpa/cpa0/.../parameter/param2`
1. Register trigger

`pardtrigger /dev/cpa0 -ldom=0 -action=0 -stats=miss_rate -cond=gt,30`

2. Prepare action scripts

Example 2: `/cpa0_ldom0_t0.sh`

```bash
1 #!/bin/sh
2 echo "<log message>>" > /log/triggers.log
3 cur_mask=cat /sys/cpa/.../waymask
4 miss_rate=cat /sys/cpa/.../miss_rate
5 capacity=cat /sys/cpa/.../capacity
6 target=update_mask(  
   $cur_mask, $miss_rate, $capacity)
7 echo $target > /sys/cpa/.../waymask
```

3. Install trigger action script

`echo "<cpa0_ldom0_t0.sh" > /sys/cpa/cpa0/ldoms/ldom0/triggers/0`
Implementation

- Full-system cycle-accurate simulator [Open Sourced *]
- FPGA prototype on Xilinx VC709 evaluation board
  - Microblaze version [Deprecated]
  - RISC-V version [Coming soon +]

* available at http://github.com/fsg-ict/PARD-gem5
+ check http://github.com/fsg-ict/PARD-fpga
Case 1: Add address mapping into CLs

- The whole server is partitioned into several sub-machines
Bare Metal Virtualization w/o Hypervisor

Web application (non-virtualized)

<table>
<thead>
<tr>
<th>Price-Performance</th>
<th>IBM SoftLayer</th>
<th>AWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum requests per second (RPS)</td>
<td>21,765 RPS</td>
<td>16,079 RPS</td>
</tr>
<tr>
<td>Average requests per second</td>
<td>3,628 RPS</td>
<td>2,680 RPS</td>
</tr>
<tr>
<td>Cost per unit of work (RPS)</td>
<td>$46/average RPS</td>
<td>$115/average RPS</td>
</tr>
</tbody>
</table>

Messaging (network-intensive)

<table>
<thead>
<tr>
<th>Price-Performance</th>
<th>IBM SoftLayer</th>
<th>AWS EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total cost</td>
<td>$128,112</td>
<td>$225,179</td>
</tr>
<tr>
<td>Messages per second (MPS)</td>
<td>70,925 MPS</td>
<td>51,995 MPS</td>
</tr>
<tr>
<td>Cost per unit of work (MPS)</td>
<td>$1.81/MPS</td>
<td>$4.33/MPS</td>
</tr>
</tbody>
</table>

Bare-metal beats virt. by up to 40%
Address Mapping in DRAM CL

Cache Backend

Control Plane
- Control Tables
- MMU
  - params
  - stats

Data Plane
- AXI
- App UI
- DDR Controller

Other Signals
- AXI Slave
- AXI Master

Trigger Table
- DSid
- base
- limit
- priority

<table>
<thead>
<tr>
<th>DSid</th>
<th>base</th>
<th>limit</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x00000000</td>
<td>0x80000000</td>
<td>High</td>
</tr>
<tr>
<td>2</td>
<td>0x80000000</td>
<td>0xC0000000</td>
<td>Medium</td>
</tr>
<tr>
<td>...</td>
<td>......</td>
<td>......</td>
<td>...</td>
</tr>
</tbody>
</table>
CPU核与I/O控制平面设计

I/O Control Plane Design

<table>
<thead>
<tr>
<th>DSID</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>0x4000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phy. intr</th>
<th>DSID</th>
<th>Vir. intr</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

AXI4 request from CPU
AXI4.aruser = 3
AXI4.araddr = 0x60000000
AXI4.araddr = 0x60004000

Interrupt to CPU
(DSID = 2, Intr = 3)
Phy intr NO. = 5
Device interrupt

Device
Bare-Metal Virtualization without Hypervisor

**LDom#1** w/ Ethernet
**IP:** 192.168.1.124

- Configuring network interfaces...
- net eth0: Promiscuous mode disabled.
- net eth0: Promiscuous mode disabled.
- net eth0: Promiscuous mode disabled.

**LDom#2** w/ Ethernet, **ip:** 192.168.1.125
- download file from server

**LDom#3** w/o Ethernet
- check cpu&memory&kernel

**512MB memory, Linux-3.14.2**
Case 2: Cache Partitioning

- 4 Ldoms: 1 X 429.mcf + 3 X Attacker
- Allocate different LLC capacities
- Perf. degradation: 7% vs. 48%
Improve Utilization w/o Loss of QoS

CPU Utilization 4X

- Memcached: Tail Latency <1.5ms

Memcached Response Time

- solo
- w/ LLC Trigger
- shared

utilization 25%->100%

co-run with interference

w/ LLC Trigger
Labeled RISC-V

- Hardware – more exploration
- Software – better ecosystem
- Goal – establish the labeled RISC-V branch
Hardware - LvNA

- **Labeled von Neumann Architecture**
- **Extend PARD to all resources**

4. Software-defined control logic

1. Fine-grained object

2. Sematic association

3. Propagation

- Application
- Schedule Framework
- Compiler
- Runtime Library
- Operating System
- Hypervisor
- Hardware
Hardware - LvNA

- Labeled von Neumann Architecture
- Extend PARD to all resources
Hypervisor - NoHype

- Push the software hypervisor down to LvNA
- Remove run-time overhead
NoHype Example

**Partition #1**

```
root@prn_core_bd:~# sh kszh 1
Download required files from server...
Connecting to 192.168.1.11 (192.168.1.11:60)
U-boot-s.bin 100% 1024.00 KiB 00:00:00
Connecting to 192.168.1.11 (192.168.1.11:60)
314.ub 100% 1224.00 KiB 00:00:00
Connecting to 192.168.1.11 (192.168.1.11:60)
system-mv-eth.dat 100% 32.00 KiB 00:00:00
Configure L10r for logic domain...
Copying boot using CDMA...
pl1 records in
pl1 records out
Copying kernel image using CDMA...
40K records in
40K records out
Copying device tree file using CDMA...
pl1 records in
pl1 records out
startup [dom]...
Run bootm 0x00400000 0x00000000 is about to startup system
root@prn_core_bd:~# ping 192.168.1.124
PING 192.168.1.124 [192.168.1.124]: 55 data bytes
64 bytes from 192.168.1.124: seq=0 ttl=64 time=0.598 ms
64 bytes from 192.168.1.124: seq=1 ttl=64 time=0.566 ms
64 bytes from 192.168.1.124: seq=2 ttl=64 time=0.570 ms
64 bytes from 192.168.1.124: seq=3 ttl=64 time=0.577 ms
64 bytes from 192.168.1.124: seq=4 ttl=64 time=0.552 ms
```

**Partition #2**

Linux-3.14.2

**Partition #3**

```
```

**Partition #4**

```
```
Operating System - Fine-grained labeling

- Add fine-grained label as context resource
  - Process
    - Process/container-level
    - Thread-level
  - Address space
    - Function-level
    - Object-level
  - Provide libraries
    - pthread_create_with_dsidd()
    - malloc_with_dsidd()

<table>
<thead>
<tr>
<th>dsid</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x8000</td>
<td>0xffff</td>
</tr>
<tr>
<td>3</td>
<td>0x2000</td>
<td>0x27ff</td>
</tr>
</tbody>
</table>
Compiler - collect QoS info. from prog

- Express QoS info. from source files
- Additional compilation results
  - Address space relative labeling info
    - Extra ELF sections for loader
- Resource requirement
  - QoS desc. file for schedule framework

```c
#define qos(10s) sort();
```

<table>
<thead>
<tr>
<th>dsid</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x8000</td>
<td>0xffff</td>
</tr>
<tr>
<td>3</td>
<td>0x2000</td>
<td>0x27ff</td>
</tr>
</tbody>
</table>

QoS Desc.
```
SLA = 10s
working set = 64KB
```
Sche. Framework - QoS resource schedule

- Expose QoS resources to schedule frameworks
- Integrate QoS resources into OpenStack [Finished]
Open Problems

- **Theory**: How does LvNA impact on RAM, PRAM, LogP models?
- **Hardware/Arch**: How to implement LvNA at in CPU, memory, storage, networking?
- **Programing Model and Compilers**: How to express users’ requirements and propagate to the hardware via labels? How to make compilers support labels?
- **OS/Hypervisor**: How to correlate labels with VMs, containers, processors, threads? How to abstract programming interfaces for labels?
- **Distributed systems**: How to correlate labels with distributed resources? How to manage distributed systems with label mechanisms?
- **Measurement/Audit**: How to leverage labels to gauge and audit resource usages?
Summary

- **QoS**: extremely important for improving utilization
- **LvNA**: a model of software-defined architecture
- **PARD**: a proof of concept of LvNA
Thanks
Overhead of Control Logic

Memory Controller: 10.1%  LLC: 3.5%
Extra Delay Analysis

- **Memory Controller**: CL significantly reduces queuing delay of high-priority requests by **5.6X**

- **Cache**: CL’s logic can be hidden in the pipeline of caches.
Cache CL: No Extra delay

- CL operations are hidden in the pipeline of a write Request

Receive Write Request → Access TagArray → Access LRU-History → Access DataArray → Update TagArray → Send Memory Request

- Lookup Parameter Table
- Update Statistics Table
- Enhanced LRU with Way-Partition
- Check Trigger Table