Omnix: an accelerator-centric OS for omni-programmable systems

Rethinking the role of CPUs in modern computers

Mark Silberstein

EE Technion

Beyond CMOS: from devices to systems
June 5-6, 2017
From devices to systems

This talk

- Algorithm
- Language
- OS
- Architecture
- ISA
- Microarchitecture
- FU
- logic
- device
Beyond CMOS: total disruption!

Differing Levels of Disruption in Computing Stack

- Algorithm
- Language
- OS
- Architecture
- ISA
- Microarchitecture
- FU
- Logic
- Device

1. Level 1: New switch, 3D
2. 2: Adiabatic, Reversible, Unreliable Sw Cryogenic
3. 3: Approximate Stochastic
4. 4: Neuromorphic
5. 5: Quantum

Legend: No Disruption

From «IEEE rebooting computing»
What to do until the next revolution?

Performance

Birth of new technology

Today

New technology matured

??????????
What to do until the next revolution?

- **Performance**
- **Birth of new technology**
- **Today**
- **New technology matured**

**Hardware specialization and near-data accelerators**
Computer hardware: circa ~2017

Network I/O accelerator

GPU parallel accelerator

Storage I/O accelerator
Central Processing Units (CPUs) are no longer Central

- Network I/O accelerator
- Storage I/O accelerator
- GPU parallel accelerator

Programmability

Mark Silberstein, Technion
Omni-programmable system
Near-X-execution Units: NXUs

- Network I/O accelerator
- Near-Data Processing
- Storage I/O accelerator
- GPU parallel accelerator
  - Accelerated Processing
  - Near-Data Processing

Programmability

Mark Silberstein, Technion
NXUs break walls
NXUs break walls
jump over walls
But NXUs also build new walls!
Agenda

• The root cause of the programmability wall
• OmniX: accelerator-centric OS design
• Example: I/O OS services for GPUs
• Example: Near-data computing in network adapters
• Conclusions
Hard to maintain whole-application efficiency

Programming complexity

Number of NXUs

CPU
multi-core CPU
CPU+ GPU
CPU+ GPU+ Smart NIC
Hard to maintain whole-application efficiency

Programming complexity

Number of skillful developers

Number of NXUs

- CPU
- multi-core CPU
- CPU+ GPU
- CPU+ GPU+ Smart NIC
Hard to maintain whole-application efficiency

Programming complexity

Numer of skillful developers

Implications

Underutilized hardware
Poor application performance
Low efficiency
High costs

Number of NXUs

CPU
multi-core CPU
CPU+ GPU
CPU+ GPU+
Smart NIC
Example: image server

1. put: parse → contrast-enhance → store
2. get: parse → resize → store → marshal

Similar architecture used in Flickr
Example: image server

1. put: parse $\rightarrow$ contrast-enhance $\rightarrow$ store
2. get: parse $\rightarrow$ resize $\rightarrow$ store $\rightarrow$ marshal
Accelerating with NXUs

1. put: parse → contrast-enhance → store
2. get: parse → resize → store → marshal
Accelerating with NXUs

1. put: parse → contrast-enhance → store
2. get: parse → resize → store → marshal
Closer look at *get*

parse → **resize** → **store** → marshal

parse req

marshaled resp

resize img

store img

NIC

SSD
OS services run on CPUs

get: parse → resize → store → marshal

parse req

recv(req)

read(file,img)

write(file,img)

send(resp)

marshal resp

resize img

NIC

CPU

SSD
Result: offloading overheads dominate

get: \texttt{parse} \rightarrow \texttt{resize} \rightarrow \texttt{store} \rightarrow \texttt{marshal}

\texttt{recv(req)} \rightarrow \texttt{read(file,img)} \rightarrow \texttt{write(file,img)} \rightarrow \texttt{send(resp)}

\texttt{parse req} \rightarrow \texttt{marshal resp}
Result: offloading overheads dominate

get: parse → resize → store → marshal

set: parse req → recv(req) → read(file, img) → write(file, img) → send(resp)

- No sockets, isolation, transport layer...
- No files, protection...

Mark Silberstein, Technion
THE problem:
OS architecture is CPU-centric
OmniX: accelerator-centric OS architecture
Wouldn't it be lovely?

get: parse → resize → store → marshal

[Diagram showing the process flow with functions such as `nic_recv(req)`, `parse req`, `marshal resp`, `nic_send(resp)` on the left side, and `sdd_read(img)`, `resize img`, `sdd_write(img)` on the right side. The NIC and SSD devices are also shown.]
OmniX OS design principles
1. A single application OS

Each processor runs an optimized library of OS services
1. A single application OS

Each processor runs an optimized library of OS services
Challenges

- GPUs run hundreds of thousands of threads!
  → Require new OS Interface semantics
- GPUs have distinct NUMA properties
  → Require new locality-optimized OS design
- GPUs cannot run privileged code
  → Require CPU assistance / peripheral support
GPUfs: File system library for GPUs

ASPL0S13, TOCS14, CACM15, ISCA16, SYSTOR16

System-wide shared namespace

CUDA

CPUs

GPU1

GPU2

GPU3

open("shared_file")

open("shared_file")

write()

mmap()

Host File System

Persistent storage

Mark Silberstein, Technion
Transparent locality optimization

Distributed Page Cache

CPUs

GPU1

GPU2

GPU3

GPUsfs

Host File System

Mark Silberstein, Technion
GPUnet: Network library for GPUs

node0.technion.ac.il

GPU native server

socket(AF_INET, SOCK_STREAM);
listen(:2340)

Network

GPUnet

CPU client

socket(AF_INET, SOCK_STREAM);
connect("node0:2340")

GPUnet

GPU native client

socket(AF_INET, SOCK_STREAM);
connect("node0:2340")
Face verification server

CPU client (unmodified) via rsocket

GPU server (GPUnet)

memcached (unmodified) via rsocket

Infiniband

=?
Face verification: Different implementations

Throughput (KReq/sec)

Latency (μsec)

1 GPU (no GPUnet)

CPU 6 cores

1 GPU GPUnet

1.9x throughput
1/3x latency (500usec)
½ LOC
2. Exclude the **CPU** from control and data path
2. Exclude the CPU from control and data path
GPUrdma: direct control of network adapter from GPUs

ROSS16

GPU-to-GPU roundtrip latency via Infiniband

- GPUnet (CPU-mediated): 50 usec
- GPUrdma (NIC controled by GPU): 10 usec

CPU-less design: lower latency
GPUpipe: CPU-less network server

Image Similarity Search

CPU-less design: much better scaling
3. Support for near-data processing

- Network
- NXU
- OS Services
- GPU
- OS Services
- CPU
- Storage
- NXU
Goals

- Low, predictable latency
- Alleviate memory bandwidth bottlenecks
- High power efficiency
- Free CPU (and its caches) for other tasks

These goals are essential for data center applications
Smart Network Adapters

On-the-fly processing at wire speed!
Typical application scenario: sensor monitoring

- Receive measurements
- Update statistics
- Alert on **critical** events
- Retrieve statistics

Uses CPU only for rare events
NICA
Network application Acceleration

- Programming model
- In-NIC programming and development
- OS integration
High-level design
High-level design

Physical NIC hardware
Vendor I/O sandbox
NICA
App

Data
Control

Network stack
NICA
Application

Host OS

Mark Silberstein, Technion
High-level design

Physical NIC hardware
Vendor I/O sandbox
NICA
App
App

Data
Control

Host OS
Network stack
NICA
Application
Application

Mark Silberstein, Technion
Example: Tweet sentiment analysis

![Diagram of data flow and classification](image)

- **Physical NIC hardware**: Vendor I/O sandbox
  - **NICA**: Approx classifier
- **Host OS**: Network stack
  - **NICA**: Exact Classifier

Example: Tweet sentiment analysis

```
010101
```
Example: Tweeter sentiment analysis

Physical NIC hardware
Vendor I/O sandbox

NICA
Approx classifier

I love you

Data

Control

Host OS
Network stack

NICA

Exact Classifier
Example: Tweeter sentiment analysis
Example: Tweeter sentiment analysis

Physical NIC hardware
Vendor I/O sandbox
NICA
Approx classifier

Host OS
Network stack
NICA
Exact Classifier

Data
Control

Example:
Tweeter sentiment analysis
Example: Tweeter sentiment analysis
Example:
Tweeter sentiment analysis
Example: Tweeter sentiment analysis
Preliminary results

Thresholdikernel

Throughput [10^6 packets/sec]

packets to CPU [%]
CPU’s role

Do the setup
Then leave
CPU’s role

Do the setup
Then leave

• Not really:
  – Runs the main program
  – Performs privileged operations, handle exceptions
OmniX is an ongoing work in Accelerator Computer Systems Lab

- Haggai Eran, Amir Watad, Shai Bergman, Tanya Brokhman, Vasilis Dimistas, Lior Zeno, Maroun Tork, Meni Orenbach, Shai Vakhnin, Lev Rosenblit, Marina Minkin, Pavel Lifshits and Gabi Malka
Summary

- Future omni-programmable systems face programmability wall
- OmniX: Accelerator-centric Operating System design
- Operating System Services for on-NXU programs
- Omnix removes CPU from control and data pathes

More technical details in [HotOS17]
Summary

- Future omni-programmable systems face **programmability wall**
- OmniX: Accelerator-centric Operating System design
- Operating System Services for on-NXU programs
- Omnix removes CPU from control and data pathes

More technical details in [HotOS17]

Thank you!

mark@ee.technion.ac.il