Emerging Topics in Computer Architecture: Programmable Accelerator, Solid-state Drive, and Security

Junghee Lee
Your Speaker

- **Education**
  - B.S. Seoul National University, 2000
  - M.S. Seoul National University, 2003
  - Ph.D. Georgia Institute of Technology, 2013

- **Work Experience**
  - Samsung Electronics, 2003-2008
  - Soteria Inc., 2013-present
  - Research scientist, 2013-present

- **Publications**
  - 10 Journal papers including 4 papers in ACM and IEEE transactions
  - 13 Conference papers, 2 of which were nominated for the best paper award
Research Experience

- Electronic system-level design (SoC/embedded system)
  - Electronic system-level model verification methodology
- Hardware-based load balancing (computer architecture)
- Networks-on-Chip (computer architecture)
  - Ring-based on-chip router architecture
  - Control and data packet segregation
- Programmable hardware accelerator (heterogeneous computer architecture)
- Solid-state drives (embedded system)
  - Preemptive garbage collection
  - Write cache design for an array of solid-state drives
- Hardware-assisted security (security)
Programmable Accelerator

- Introduction
- Execution Model
- Hardware Architecture
- Evaluation
- Conclusion
**Introduction**

**Massively Parallel Processing Array**

- **Single Core**
- **Multi Core**
- **Many Core**
- **Fusion**

*Powerful cores + H/W accelerator in a single die Ex) AMD Fusion*

*Programmable Hardware Accelerator Ex) GPGPU*
MPPA as Hardware Accelerator

Challenges

- Expressiveness
- Debugging
- Memory Hierarchy Design
## Related Works

<table>
<thead>
<tr>
<th></th>
<th>Expressiveness</th>
<th>Debugging</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPGPU AMD Fusion</td>
<td>SIMD</td>
<td>Multiple debuggers Event graph</td>
<td>Scratch-pad memory Cache</td>
</tr>
<tr>
<td>Tilera</td>
<td>Multi-threading</td>
<td>Multiple debuggers</td>
<td>Coherent cache</td>
</tr>
<tr>
<td>Rigel</td>
<td>Multi-threading</td>
<td>Not addressed</td>
<td>Software-managed cache</td>
</tr>
<tr>
<td>Ambric</td>
<td>Kahn process network</td>
<td>Formal model</td>
<td>Scratch-pad memory</td>
</tr>
<tr>
<td>Proposed MPPA</td>
<td>Event-driven model</td>
<td>Inter-module debug Intra-module debug</td>
<td>Scratch-pad memory Prefetching</td>
</tr>
</tbody>
</table>
Programmable Accelerator

- Introduction
- Execution Model
- Hardware Architecture
- Evaluation
- Conclusion
 Execution Model

Assembly

Structural

Object-oriented

Von Neuman Model

x86  MIPS  ARM

Multi-thread

MPI

SIMD

Von Neuman Model

x86  MIPS  ARM
Requirements

- **Decoupling**
  - The execution model should decouple the programming model and the execution model of the parallel hardware

- **Hardware perspective**
  - Low implementation overhead
  - Heterogeneity
  - Scalability

- **Software perspective**
  - Easy to program
  - Easy to debug
  - Performance
Event-driven Execution Model

- **Specification**
  - Module = \((b, P_i, P_o, C, F)\)
    - \(b\) = Behavior of module
    - \(P_i\) = Input ports
    - \(P_o\) = Output ports
    - \(C\) = Sensitivity list
  - Signal
  - Net = \((d, K)\)
    - \(d\) = Driver port
    - \(K\) = A set of sink ports

- **Semantics**
  - A module is triggered when any signal connected to \(C\) changes
  - Function calls and memory accesses are limited to within a module
  - Non-blocking write and block read
  - The specification can be modified during run-time
Programmable Accelerator

- Introduction
- Execution Model
- Hardware Architecture
- Evaluation
- Conclusion
MPPA Microarchitecture

- Identical core tiles
- Consists of uCPU, scratch-pad memory, and peripherals that support the execution model
- One core tile is designated to an execution engine

- Software running on a core tile
- Consists of scheduler, signal storage and interconnect directory
- Supports the execution model
- If necessary, it is split into multiple instances running on different core tiles
Core Tile Architecture

- **uCPU**
  - Generic small processor
  - Treated as a black box

- **Scratch Pad Memory**
  - For Current Module
  - Stores information about the modules
  - Stores the input data
  - Notifies the update event to the interconnect directory when the output is updated
  - Prefetches the code and data of the next module while the current module is running on uCPU
  - Counter-part prefetecher
  - Sends data to requester

- **For Next Module**
  - Stores the output data
  - Notifies the update event to the interconnect directory when the output is updated
  - Switches the context when the current module finishes and the next module is ready
  - Stores the input data

- **Context Manager**
  - Generic small processor
  - Treated as a black box
  - Stores information

- **Message Queue**
  - Handles the system messages

- **Network Interface**
  - NoC router
Execution Engine

- Most of its functionality is implemented in software while the hardware facilitates communication. Software implementation gives us flexibility in the number and location of the execution engine.
- One way to visualize our MPPA is to regard the execution engine as an event-driven simulation kernel.
- The execution engine interacts with modules running on other core tiles through messages.

<table>
<thead>
<tr>
<th>Type</th>
<th>From</th>
<th>To</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_FETCH_MODULE</td>
<td>Prefetcher</td>
<td>Scheduler</td>
<td>Request a new module</td>
</tr>
<tr>
<td>RES_FETCH_MODULE</td>
<td>Scheduler</td>
<td>Prefetcher</td>
<td>Module ID and list of input ports</td>
</tr>
<tr>
<td>MODULE_INSTANCE</td>
<td>Scheduler</td>
<td>Prefetcher</td>
<td>Code of the module</td>
</tr>
<tr>
<td>REQ_SIGNAL</td>
<td>Prefetcher</td>
<td>Interconnect</td>
<td>Port ID</td>
</tr>
<tr>
<td>RES_SIGNAL</td>
<td>Signal storage or a node</td>
<td>Prefetcher</td>
<td>Data</td>
</tr>
</tbody>
</table>

15/54
Components of Execution Engine

- **Scheduler**
  - Keeps track of the status and location of modules
  - Maintains three queues: wait, ready and run queue

- **Signal storage**
  - Stores signal values in the device memory
  - If a signal is updated but its value is still stored in the node, the signal storage invalidates its value and keeps the location of the latest value

- **Interconnect directory**
  - Keeps track of connectivity of signals and ports
  - Maintains the sensitivity list
Module-Level Prefetching

- Hides the overhead of the dynamic scheduling
- Prefetches the next module while the current module is running
Programmable Accelerator

- Introduction
- Execution Model
- Hardware Architecture
- Evaluation
- Conclusion
Benchmark

- Recognition, Synthesis and Mining (RMS) benchmark
- Fine-grained parallelism: dominated by short tasks
  - Small memory foot print
  - High run-time scheduling overhead
- Task-level parallelism: exhibits dependency
  - Hard to be implemented with GPGPU

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Min</th>
<th>Max</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Solve (FS)</td>
<td>26</td>
<td>646</td>
<td>336.00</td>
</tr>
<tr>
<td>Backward Solve (BS)</td>
<td>42</td>
<td>569</td>
<td>305.50</td>
</tr>
<tr>
<td>Cholesky Factorization (CF)</td>
<td>151</td>
<td>11800</td>
<td>789.35</td>
</tr>
<tr>
<td>Canny Edge Detection (CED)</td>
<td>330</td>
<td>5011</td>
<td>669.68</td>
</tr>
<tr>
<td>Binominal Tree (BT)</td>
<td>117</td>
<td>4506</td>
<td>462.71</td>
</tr>
<tr>
<td>Octree Partitioning (OP)</td>
<td>1441</td>
<td>6679</td>
<td>2678.70</td>
</tr>
<tr>
<td>Quick Sort (QS)</td>
<td>88</td>
<td>47027</td>
<td>683.70</td>
</tr>
</tbody>
</table>
## Simulator

- **In-house cycle-level simulator**
- **Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of core tiles</td>
<td>32</td>
</tr>
<tr>
<td>Memory access time</td>
<td>1 cycle for scratch-pad memory</td>
</tr>
<tr>
<td></td>
<td>100 cycles for device memory</td>
</tr>
<tr>
<td>Memory size</td>
<td>8 KB scratch-pad memory</td>
</tr>
<tr>
<td></td>
<td>32 MB device memory</td>
</tr>
<tr>
<td>Communication Delay</td>
<td>4 cycles per hop</td>
</tr>
</tbody>
</table>
Utilization

Core utilization

Benchmarks

FS  BS  CF  CED  BT  OP  QS

w/o prefetching  w/ prefetching
Conclusion

• A novel MPPA architecture is proposed that employs an event-driven execution model
  – Handles dependencies by dynamic scheduling
  – Hides dynamic scheduling overhead by module-level prefetching

• Future works
  – Supports applications that require larger memory footprint
  – Adjusts the number of execution engines dynamically
  – Supports inter-module debugging
Solid-state Drive

- Introduction
- Background and Motivation
- Semi-Preemptive Garbage Collection
- Evaluation
- Conclusion
High Performance Storage Systems

- Server centric services
  - File, web & media servers, transaction processing servers
- Enterprise-scale Storage Systems
  - Information technology focusing on storage, protection, retrieval of data in large-scale environments

High Performance Storage Systems

Storage Unit
Hard Disk Drive
Emergence of NAND Flash based SSD

• NAND Flash vs. Hard Disk Drives
  – Pros:
    • Semi-conductor technology, no mechanical parts
    • Offer lower access latencies
      – $\mu$s for SSDs vs. $ms$ for HDDs
    • Lower power consumption
    • Higher robustness to vibrations and temperature
  – Cons:
    • Limited lifetime
      – 10K - 1M erases per block
    • High cost
      – About 8X more expensive than current hard disks
    • Performance variability
Solid-state Drive

- Introduction
- Background and Motivation
- Semi-Preemptive Garbage Collection
- Evaluation
- Conclusion
Out-Of-Place Write

Logical-to-Physical Address Mapping Table

<table>
<thead>
<tr>
<th>LPN0</th>
<th>PPN1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPN1</td>
<td>PPN4</td>
</tr>
<tr>
<td>LPN2</td>
<td>PPN3</td>
</tr>
<tr>
<td>LPN3</td>
<td>PPN5</td>
</tr>
</tbody>
</table>

Physical Blocks

<table>
<thead>
<tr>
<th></th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>V</td>
<td>I</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>

- Write to LPN2
- Invalidate PPN2
- Write to PPN3
- Update table
Garbage Collection

Select Victim Block

Move Valid Pages

Erase Victim Block

Physical Blocks

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>E</td>
</tr>
<tr>
<td>P1</td>
<td>E</td>
</tr>
<tr>
<td>P2</td>
<td>E</td>
</tr>
<tr>
<td>P3</td>
<td>E</td>
</tr>
<tr>
<td>P4</td>
<td>V</td>
</tr>
<tr>
<td>P5</td>
<td>V</td>
</tr>
<tr>
<td>P6</td>
<td>V</td>
</tr>
<tr>
<td>P7</td>
<td>V</td>
</tr>
</tbody>
</table>

2 reads + 2 writes + 1 erase = 2 * 0.025 + 2 * 0.200 + 1.5 = 1.950 (ms) !!
Solid-state Drive

- Introduction
- Background and Motivation
- Semi-Preemptive Garbage Collection
- Evaluation
- Conclusion
Technique #1: Semi-Preemption

- Request
- Preemptive GC
- Non-Preemptive GC

- Read page x
- Write page x
- Erase a block

- Data transfer
- Meta data update

Time

GC

R_x
W_x
R_y
W_y
E

W_z

32/54
Technique #2: Merge

- \( R_y \) Request
- \( R_x \) Read page x
- \( W_x \) Write page x
- \( W_y \) Data transfer
- \( E \) Erase a block
- \( E \) Meta data update

Time

GC
Technique #3: Pipeline

- Request
- Read page x
- Write page x
- Erase a block
- Data transfer
- Meta data update

Time

GC

R_x  W_x  R_y  W_y  E
Level of Allowed Preemption

- **Drawback of PGC**
  - The completion time of GC is delayed
  - May incur lack of free blocks
  - Sometimes need to prohibit preemption

- **States of PGC**

<table>
<thead>
<tr>
<th></th>
<th>Garbage collection</th>
<th>Read requests</th>
<th>Write requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State 1</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>State 2</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>State 3</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Solid-state Drive

- Introduction
- Background and Motivation
- Semi-Preemptive Garbage Collection
- Evaluation
- Conclusion
Setup

- Simulator
  - MSR’s SSD simulator based on DiskSim
- Workloads

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Average request size (KB)</th>
<th>Read ratio (%)</th>
<th>Arrival rate (IOP/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write dominant</td>
<td>Financial</td>
<td>7.09</td>
<td>18.92</td>
</tr>
<tr>
<td></td>
<td>Cello</td>
<td>7.06</td>
<td>19.63</td>
</tr>
<tr>
<td>Read dominant</td>
<td>TPC-H</td>
<td>31.62</td>
<td>91.80</td>
</tr>
<tr>
<td></td>
<td>OpenMail</td>
<td>9.49</td>
<td>63.30</td>
</tr>
</tbody>
</table>
Performance Improvement for Realistic Workloads

- **Average Response Time**
  - Improvement of average response time by 6.5% and 66.6% for Financial and Cello.

- **Variance of Response Times**
  - Improvement of variance of response time by 49.8% and 83.3% for Financial and Cello.
Conclusions

• Solid state drives
  – Fast access speed
  – Performance variation ⇔ garbage collection

• Semi-preemptive garbage collection
  – Service incoming requests during GC

• Average response time and performance variation are reduced by up to 66.6% and 83.3%
Security

• Introduction
• Append-only Storage
• Use Cases
• Conclusion
• IoT (Internet of Things)
  - By 2015, 5 billion individuals will be connected to the Internet (source: GKP)
  - 100 billion uniquely identifiable objects will be connected to the Internet by 2020

• Big Data Visualization
  - Digital data is doubling every other year

• Cloud Computing and Mobile Computing

• Cybersecurity
  - New business models based on innovative thinking will be needed
Financial Impact

- Computer crimes cost firms who detect and verify incidents between $145 million and $730 million each year (NCSA Annual Worry Report)
- A company that experiences a computer outage lasting more than 10 days will never fully recover financially. 50 percent will be out of business within five years ("Disaster Recovery Planning: Managing Risk & Catastrophe in Information Systems" by Jon Toigo)
- 43% of lost or stolen data is valued at $5 million or more
- 43% of companies experiencing data disasters never reopen, and 29 percent close within two years (McGladrey and Pullen)
- It is estimated that 1 out of 500 data centers will have a severe disaster each year (McGladrey and Pullen)
Scope

- Network Security
  - Efficient
    - It can protect numerous hosts by securing only the perimeter
  - But not perfect
    - Although data centers are equipped with various network security techniques, it is estimated 1 out of 500 data centers will have a severe disaster each year (McGladrey and Pullen)
    - The ultimate goal is protecting hosts

- Host Security
  - Protect hosts directly
  - Compatibility issue
    - Heterogeneity of the hosts (different version and types of OS and different hardware)
  - Performance overhead
Hardware-assisted Security

- Trusted Platform Module (TPM)
  - Key burnt in hardware
- Intel vPro
  - Trusted Execution Technology
    - Virtualization (TrustZone of ARM)
  - Identity Protection Technology
    - One-time password
- Monitoring
  - Copilot, RKRD, KI-Mon
    - Coprocessor-based
Security

- Introduction
- Append-only Storage
- Use Cases
- Conclusion
Elevator Pitch

Protect reference data from unauthorized modification by using Append-only Storage

- Write-only read many (WORM) devices: CD or DVD
Soteria Security Card (SSC)
Security

- Introduction
- Append-only Storage
- Use Cases
- Conclusion
Use Case #1: Log Protection

- **Using SSC**
  - Logs are stored in both the hard disk and SSC
  - Log integrity checker checks if the logs are contaminated by comparing those in the hard disk against those in SSC

- **Performance**
  - Performance degradation of the response time of the Apache web server is 0.88% employing a separate process to store logs
Current Practice

- Log protection techniques
  - Logging server
    - Vulnerabilities involved in collecting and transferring logs
  - Encryption
    - Encryption is secure only if the key is not revealed
    - According to the 2012 Verizon Data Breach report, 76% of data breaches exploited weak or stolen credentials
  - Hypervisor
    - Who protects hypervisor itself?

- Does this really happen?
  - According to a police officer in charge of cyber crime investigation,
    - some attackers delete their traces from logs, and
    - some attackers delete everything from the hard disk, which includes logs
Use Case #2: File Integrity Check

- **File integrity**
  - File modification is usually (if not always) a prerequisite or a result of malware
  - Therefore, file integrity checking is a powerful tool to find out the cause of attacks and malware

- **Using SSC**
  - The integrity information of files is stored in the hardware
  - By comparing against the stored integrity information, unauthorized modifications can be detected

- **Performance**
  - Since the file integrity checker is an off-line utility, the performance impact can be minimized by assigning a low priority
  - Malware detectors and integrity checkers detect malicious activities by comparing against some reference data
Conclusion

• Soteria Security Card:
  – Prevents reference data from unauthorized modification
  – Stored data cannot be modified or erased

• Use cases
  – Log protection
  – File integrity checking
  – File access monitoring
  – Non-repudiation
  – Medical record
  – Financial transaction
Thank you!
A Programmable Processing Array Architecture Supporting Dynamic Task Scheduling and Module-Level Prefetching

Junghhee Lee*, Hyung Gyu Lee*, Soonhoi Ha†, Jongman Kim*, and Chrysostomos Nicopolous‡
Example

- **Quick sort**
  - Pivot is selected
  - The given array is partitioned so that
    - The left segment should contain smaller elements than the pivot
    - The right segment should contain larger elements than the pivot
  - Recursively partition the left and right segments

- **Specifying quick sort**
  - Multi-threading
    - OK but hard to debug
  - SIMD
    - Inefficient due to input dependency
  - Kahn process network
    - Impossible due to the dynamic nature
Specify Quick Sort with Event-driven Model

- **Partition module**
  - \( b \) (behavior): select a pivot, partition the input array, instantiate another partition module if necessary
  - \( P_i \) (input port): input array and its position
  - \( P_o \) (output port): left and right segments and their position
  - \( C \) (sensitivity list): input array
  - \( P \) (prefetch list): input array

- **Collection module**
  - \( b \) (behavior): collect segments
  - \( P_i \) (input port): sorted segments and intermediate result
  - \( P_o \) (output port): final result and intermediate result
  - \( C \) (sensitivity list): sorted segments
  - \( P \) (prefetch list): sorted segments and intermediate result

\[ 
\text{Input array} \rightarrow \text{Partition} \rightarrow \text{Collection} \rightarrow \text{Final result} \]

\[ 
\downarrow \text{Partition} \rightarrow \ldots \rightarrow \text{Partition} \rightarrow \text{Collection} \rightarrow \text{Final result} \]

\[ 
\downarrow \text{Partition} \rightarrow \ldots \rightarrow \text{Partition} \rightarrow \text{Collection} \rightarrow \text{Final result} \]

\[ 
\downarrow \text{Partition} \rightarrow \ldots \rightarrow \text{Partition} \rightarrow \text{Collection} \rightarrow \text{Final result} \]
Illustrative Example

Partition 0
- uCPU
- Prefetcher
- Out Sig Q
- Msg Handler
- Interconnect Directory
- Signal Storage

Partition 1
- uCPU
- Prefetcher
- Out Sig Q
- Msg Handler
- Scheduler
- Collection
- Collection
- Collection

Partition 2
- uCPU
- Prefetcher
- Out Sig Q
- Msg Handler
- Collection
- Collection
- Collection

Partition 3
- uCPU
- Prefetcher
- Out Sig Q
- Msg Handler

Partition 4
- uCPU
- Prefetcher
- Out Sig Q
- Msg Handler

Partition 5
- uCPU
- Prefetcher
- Out Sig Q
- Msg Handler

Collection
Scalability

Core utilization vs. Number of core tiles

Execution time (cycles vs. Number of core tiles)

Util (1) and Util (3) vs. Execution time (1) and Execution time (3)
A Semi-Preemptive Garbage Collector for Solid State Drives

Junghee Lee, Youngjae Kim, Galen M. Shipman, Sarp Oral, Feiyi Wang, and Jongman Kim

Oak Ridge National Laboratory
Managed by UT-Battelle for the Department of Energy

Georgia Institute of Technology
Spider: A Large-scale Storage System

- **Jaguar**
  - Peta-scale computing machine
  - 25,000 nodes with 250,000 cores and over 300 TB memory

- **Spider storage system**
  - The largest center-wide Lustre-based file system
  - Over 10.7 PB of RAID 6 formatted capacity
    - 13,400 x 1 TB HDDs
    - 192 Lustre I/O servers
      - Over 3TB of memory (on Lustre I/O servers)
Pathological Behavior of SSDs

- Does GC have an impact on the foreground operations?
  - If so, we can observe sudden bandwidth drop
  - More drop with more write requests
  - More drop with more bursty workloads

- Experimental Setup
  - SSD devices
    - Intel (SLC) 64GB SSD
    - SuperTalent (MLC) 120GB SSD
  - I/O generator
    - Used *libaio* asynchronous I/O library for block-level testing
Bandwidth Drop for Write-Dominant Workloads

- Experiments
  - Measured bandwidth for 1MB by varying read-write ratio

Performance variability increases as we increase write-percentage of workloads.
Performance Variability for Bursty Workloads

- Experiments
  - Measured SSD write bandwidth for queue depth (qd) is 8 and 64

![Graphs showing performance variability for Intel SLC (SSD) and SuperTalent MLC (SSD)]

Performance variability increases as we increase the arrival-rate of requests (bursty workloads).
Lessons Learned

- From the empirical study, we learned:
  - Performance variability increases as the percentage of writes in workloads increases.
  - Performance variability increases with respect to the arrival rate of write requests.

- This is because:
  - Any incoming requests during the GC should wait until the on-going GC ends.
  - *GC is not preemptive*
Performance Improvements for Synthetic Workloads

- Varied four parameters: request size, inter-arrival time, sequentiality and read/write ratio
- Varied one at a time fixing others
Performance Improvement for Synthetic Workloads (con’t)

- **Bursty**
  - Inter-arrival time (ms)
  - Probability of sequential access

- **Random dominant**
  - Probability of read access

- **Write dominant**
  - Inter-arrival time (ms)
  - Probability of sequential access
Hardware-assisted Intrusion Detection by Preserving Reference Information Integrity

Junghee Lee, Chrysostomos Nicopoulos, Gi Hwan Oh, Sang-Won Lee, and Jongman Kim