Rhythm: Harnessing Data Parallel Hardware for Server Workloads

Sandeep R. Agrawal
Jun Pang
David Tarjan

Valentin Pistol
John Tran
Alvin R. Lebeck

$Duke CS    #NVIDIA
Explosive Internet Growth

• Increasing web traffic and cloud service demands

• Example: Facebook
  • 900 million users
  • ~1 trillion page views a month or ~350,000 per second
  • 1.2 million photos per second
  • >100,000 servers

• How to best satisfy this demand?
  • Add more machines – More space and cooling costs
  • Improve existing machines to achieve higher throughput/Watt
SIMD Accelerator Efficiency

- Instruction Fetch + Decode as high as 30%-40% of core power!
- SIMT/SIMD amortizes fetch and decode costs
- NVIDIA Kepler and Intel Xeon Phi achieve > 5 GFlops/Watt
- Can we harness accelerator efficiency to increase throughput/Watt for server workloads?

$Hameed, et al. ISCA 2010

*Sartori, et al. HPCA 2012
Insight

• Many requests perform the same task(s), such as login or search query
• Delay some requests in order to “align” the execution of similar requests (a cohort)
• Execute cohort on a SIMD accelerator, trading response time for throughput and efficiency

• Motivated by Cohort Scheduling [Larus & Parkes ’02]
  • PacketShader [Han et al. ’10]
  • Memcached on GPUs [Hetherington et al. ’12]
Enabling Trends

• Memory bandwidth to accelerator
  • PCIe 4.0
  • SoC designs like Tegra K1 and AMD Fusion

• Network bandwidth
  • 100 Gbps Ethernet (IEEE 802.3bj)
  • 400 Gbps Ethernet Study Group (http://www.ieee802.org/3/400GSG/)

• High Throughput OS/DB Services
  • GPU-FS [Silberstein, et al. ASPLOS ’13]
  • Vector Interfaces [Vasudevan, et al. SOCC ’12]
  • Memcached
Server Design Space

- Ideal design
  - Throughput (Requests/second) $\geq$ an x86 core
  - Energy efficiency (Requests/Joule) $\geq$ an ARM core
Outline

- Motivation
- **Software Architecture**
- Implementation
- Evaluation
- Conclusion
Conventional Server Pipeline

- Requests processed individually
  - Thread per request (Apache)
  - Event driven execution (Nginx)
Rhythm

- Pipelined architecture for processing “cohorts” of requests on data parallel hardware
- Extends cohort scheduling and event-based staged servers
- **Cohort** – group of “similar” requests
  - Control flow similarity for SIMD accelerators
- Maximize throughput by stalling only on resource shortage
The Rhythm Pipeline

Clients

Inflight cohort

HTTP Request
Different colors denote different types

Clients
The Rhythm Pipeline

Clients

request accumulation

Reader

Inflight cohort

HTTP Request
Different colors denote different types

Clients
The Rhythm Pipeline

Clients → parse using HTTP spec → Reader → Parser → Inflight cohort → HTTP Request

Different colors denote different types
The Rhythm Pipeline

Clients

execute on host/device?

Inflight cohort

HTTP Request
Different colors denote different types
The Rhythm Pipeline

N backend, N+1 process stages

HTTP Request
Different colors denote different types

Inflight cohort
The Rhythm Pipeline

Clients → Reader → Parser → Dispatch → Process → DB → Process → Response → Clients

response, generation, send to clients

Inflight cohort

HTTP Request
Different colors denote different types
The Rhythm Pipeline

Clients → request accumulation

parse using HTTP spec

execute on host/device?

N backend, N+1 process stages

response, generation, send to clients

Clients

HTTP Request

Different colors denote different types

Inflight cohort
Design goals

- Applicable to any SIMD hardware
- Utilize the most efficient computational resource
  - Dispatch cohort on host or accelerator
- Support deep pipelines
  - Arbitrary number of process and backend stages
- Support wide pipelines
  - Multiple instances of slowest stage to maximize throughput
- A pipeline stage implementation can be on host or accelerator
Optimizations

- Thread per request in cohort
- Transpose request, response buffers for coalescing
- Whitespace Padding in HTML Content and Headers
Prototype Implementation (GTX Titan)

1. Read requests from clients, launch warps
2. Read
3. Transpose
4. Dispatch
5. Process
6. Transpose
7. Write
8. Read
9. Transpose
10. Process
11. Process
12. Deinterleave client response
13. Write

HTTP Requests/Responses

Storage/Backend

Host

Accelerator

A request can access backend multiple times
Outline

• Motivation
• Software Architecture
• Implementation
• Evaluation
• Conclusion
Methodology

<table>
<thead>
<tr>
<th>Platform</th>
<th>GHz</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core i5</td>
<td>3.4</td>
<td>Core i5 3570, 22 nm, 4 cores (4 threads), 8GB DDR3 RAM, 1Gbps NIC</td>
</tr>
<tr>
<td>Core i7</td>
<td>3.4</td>
<td>Core i7 3770, 22 nm, 4 cores (8 threads), 16GB DDR3 RAM, 1Gbps NIC</td>
</tr>
<tr>
<td>ARM A9</td>
<td>1.2</td>
<td>OMAP 4460, 45 nm, Pandaboard, 2 cores, 1GB LPDDR2 RAM</td>
</tr>
<tr>
<td>Titan</td>
<td>0.8</td>
<td>GTX Titan, 28 nm, 14 Streaming Multiprocessors, 6GB GDDR5 Memory</td>
</tr>
</tbody>
</table>

- **SPECWeb Banking** (14 of 16 request types)
  - C version on x86 and ARM platforms
  - C+CUDA version on Titan platform (Rhythm)

- **Metrics** (weighted arithmetic mean)
  - Throughput
  - Power (Kill-A-Watt)
  - Latency
Future Server Platforms

• 1 Gbps NIC cannot sustain required throughputs
  • Max throughput 517Gbps raw, <100Gbps compressed
  • Requests generated locally on host, No responses sent

• x86 and ARM platforms
  • Backend emulated as function call

• Titan A
  • Backend emulated on host as separate thread

• Titan B
  • Backend emulated on device as function call

• Titan C
  • Titan B + No transpose for final response
Evaluation

Normalized ARM efficiency (Req/Joule)

Normalized i7 throughput (Req/Sec)

- 0.6x efficiency
- 1.1x throughput
- why?

- 1.2x efficiency
- 4.1x throughput

- 3.4x efficiency
- 8.2x throughput

Duke Computer Architecture
Titan A Limitations

Throughput achieved  Throughput capped by PCIE

PCIE 3.0 bandwidth < 12 Gbps
Latency

- **Titan A – 86 ms** (weighted average)
  - PCIE Stalls
- **Titan B – 24 ms** (weighted average)
  - 99th percentile latency for different request types is 18%-40% longer than the average
- **Titan C – 10 ms** (weighted average)
  - 99th percentile latency for different request types is 7%-34% longer than the average
  - Latency omits time for transpose of response
Scaling Many Core Processors

- Assume dynamic power of 1W per ARM core and 10W per x86 core
- Find #cores to match accelerator throughput
- To match Titan B’s throughput (232W dynamic power)

<table>
<thead>
<tr>
<th>Platform</th>
<th>#Cores</th>
<th>Power(W)</th>
<th>Available Uncore Power(W(%))</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>192</td>
<td>192</td>
<td>40 (21%)</td>
</tr>
<tr>
<td>x86</td>
<td>21</td>
<td>210</td>
<td>22 (10%)</td>
</tr>
</tbody>
</table>

- To match Titan C’s throughput (211W dynamic power)

<table>
<thead>
<tr>
<th>Platform</th>
<th>#Cores</th>
<th>Power(W)</th>
<th>Available Uncore Power(W(%))</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>385</td>
<td>385</td>
<td>-174 (-45%)</td>
</tr>
<tr>
<td>x86</td>
<td>41</td>
<td>410</td>
<td>-199 (-48%)</td>
</tr>
</tbody>
</table>

- Titan C has >170W to implement the transpose operation and still outperform the scaled systems.
Conclusion

- Web Server workloads amenable to SIMD accelerators
- Cohort scheduling using Rhythm to improve Requests/Joule
- Programming language/Runtime
  - reduce programmer effort to create/edit web pages
- Other workloads
- More SIMD platforms
  - Xeon Phi, ARM NEON, Tegra K1, AMD Fusion
Questions?

• Thanks!
Backup
Dynamic Power

- Dynamic Power = Power_{under\_load} − Power_{idle}
Evaluation – Dynamic power

- Normalized i7 throughput (Req/Sec)
- Normalized ARM efficiency (Req/Joule)

- Core i5
- Core i7
- Arm a9
- Titan A
- Titan B
- Titan C

- 0.5x efficiency
- 1.1x throughput
- Titan A

- 0.9x efficiency
- 4.1x throughput
- Why?

- 2.5x efficiency
- 8.2x throughput

- 1.1x throughput
- 0.9x efficiency
- Titan C

Duke Computer Architecture
Titan B – Dynamic Power

Why?

Normalized i7 throughput (Req/Sec)

Normalized ARM efficiency (Req/Joule)

post_transfer

add_payee

profile

post_payee

logout

bill_pay

account_summary

login

check_detail_html

transfer

change_profile

order_check

place_check_order

bill_pay_status_output

0.60
0.70
0.80
0.90
1.00
1.10
1.20
1.30

2.00
2.50
3.00
3.50
4.00
4.50
5.00
5.50

0.60
0.70
0.80
0.90
1.00
1.10
1.20
1.30

2.00
2.50
3.00
3.50
4.00
4.50
5.00
5.50

Duke Computer Architecture
## Titan B – Response Buffer Sizes

<table>
<thead>
<tr>
<th>Request type</th>
<th>Response size (KB)</th>
<th>Difference (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPECWeb</td>
<td>Rhythm</td>
</tr>
<tr>
<td>login</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>account_summary</td>
<td>17</td>
<td>32</td>
</tr>
<tr>
<td>add_payee</td>
<td>18</td>
<td>32</td>
</tr>
<tr>
<td>bill_pay</td>
<td>15</td>
<td>32</td>
</tr>
<tr>
<td>bill_pay_status_output</td>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>change_profile</td>
<td>29</td>
<td>32</td>
</tr>
<tr>
<td>check_detail_html</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>order_check</td>
<td>21</td>
<td>32</td>
</tr>
<tr>
<td>place_check_order</td>
<td>25</td>
<td>32</td>
</tr>
<tr>
<td>post_payee</td>
<td>34</td>
<td>64</td>
</tr>
<tr>
<td>post_transfer</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>profile</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>transfer</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>logout</td>
<td>46</td>
<td>64</td>
</tr>
</tbody>
</table>

- Well matched buffers give higher efficiency
Request similarity (SPECWeb Banking)
Rhythm Pipeline Control

• Event loop
  • Single-threaded, *epoll* based
  • Handle New connections
  • Backend responses
  • File system responses

• Callbacks
  • Linked list traversed on each iteration
  • Track stage completion via polling (no device interrupts)
  • Track stage transitions

• Data Structures
  • Cohort pool – static array
  • Session state – concurrent hash table
CUDA Specific Optimizations

- Transposes using shared memory
- Max butterfly reduction in shared memory to calculate HTML padding
- Constant memory to store static HTML content
- Store frequently used pointers in constant memory to reduce register pressure
Delay Requests?

- Can you really delay requests?
  - EcoDB [Lang & Patel CIDER ’09]
  - DreamWeaver [Meisner & Wenisch ASPLOS ’12]
Contributions

- *Rhythm*, a software architecture for high throughput SIMT-based servers
- Evaluation of future server platform architectures
- Prototype implementation of *Rhythm* on NVIDIA GPUs
- Standalone C and C+CUDA implementations of SPECWeb2009 Banking
The Rhythm Pipeline

- **Clients** request accumulation
- **Readers** parse using HTTP spec
- **Parsers** execute on host/device?
- **Dispatch** n backend, n+1 process stages
- **Process** response, generation, send to clients
- **Responses** Clients
The Rhythm Pipeline (SPECweb Banking)

- Maximize #inflight cohorts to improve throughput & efficiency
- A pipeline stage implementation can be on host or accelerator