Exploring Tradeoffs between Programmability and Efficiency in Data-Parallel Accelerators

Yunsup Lee¹, Rimas Avizienis¹, Alex Bishara¹, Richard Xia¹, Derek Lockhart², Christopher Batten², Krste Asanovic¹

¹The Parallel Computing Lab, UC Berkeley
²Computer Systems Lab, Cornell University
DLP Kernels Dominate Many Computational Workloads

- Graphics Rendering
- Computer Vision
- Audio Processing
- Physical Simulation
DLP Accelerators are Getting Popular

Sandy Bridge

Knights Ferry

Fermi

Tegra
Important Metrics when Comparing DLP Accelerator Architectures

- Performance per Unit Area
- Energy per Task
- Flexibility (What can it run well?)
- Programmability (How hard is it to write code?)
Efficiency vs. Programmability: It’s a tradeoff
Maven Provides Both Greater Efficiency and Easier Programmability

- Maven/Vector-Thread
- Vector
- MIMD

Efficiency vs. Programmability

Regular DLP

Irregular DLP
Where does the GPU/SIMT fit in this picture?

Programmability vs. Efficiency

Regular DLP

Irregular DLP

Maven/Vector-Thread

Vector

GPU

SIMT?

MIMD

Yunsup Lee / UC Berkeley Par Lab
Outline

- Data-Parallel Architecture
  - Design Patterns
    - MIMD, Vector-SIMD, Subword-SIMD, SIMT, Maven/Vector-Thread
- Microarchitectural Components
- Evaluation Framework
- Evaluation Results
DLP Pattern #1: MIMD

Programmer’s Logical View

Vector-Vector Add

```
loop:
  load  a, a_ptr
  load  b, b_ptr
  add   c, a, b
  store c, c_ptr
  a_ptr++
  b_ptr++
  c_ptr++
  branch
```

Masked Filter

```
loop:
  load  a, a_ptr
  a_ptr++
  branch a = 0
  op0
  op1
  ...
  branch

FILTER OP
```
DLP Pattern #1: MIMD

Programmer’s Logical View

Examples: Tilera, Rigel

Typical Micro-architecture

Multi-threaded Cores
DLP Pattern #2: Vector-SIMD

Programmer’s Logical View

Vector-SIMD Arithmetic Instructions  Vector-SIMD Memory Instructions  Architectural Vector Register with 4 Elements

Vector-Vector Add

loop:
\[
\text{vload } A, a\_ptr \\
\text{vload } B, b\_ptr \\
\text{vadd } C, A, B \\
\text{vstore } C, c\_ptr \\
\text{a\_ptr++} \\
\text{b\_ptr++} \\
\text{c\_ptr++} \\
\text{branch} \\
\]

Masked Filter

loop:
\[
\text{vload } A, a\_ptr \\
\text{vset } F, A = 0 \\
\text{vop0 under flag } F \\
\text{vop1 under flag } F \\
\text{...} \\
\text{a\_ptr++} \\
\text{branch} \\
\]

Yunsup Lee / UC Berkeley Par Lab
DLP Pattern #2: Vector-SIMD

Programmer’s Logical View

Typical Micro-architecture

Examples:
T0
Cray-1
DLP Pattern #3: Subword-SIMD

Programmer’s Logical View

Subword-SIMD Arithmetic Commands

Full-Word Memory Commands

Execution Resources w/ Subword Ops

Architectural 64b Register w/ 8 Subwords

Memory

Typical Micro-architecture

Instruction Memory

SIMD Unit

Data Memory

Examples: AVX/SSE
DLP Pattern #4: GPU/SIMT

Microthread Block w/ 4 Microthreads

**Programmer’s Logical View**

**Vector-Vector Add**

- `a_ptr = a_base * thread_id`
- `b_ptr = b_base * thread_id`
- `c_ptr = c_base * thread_id`
- `load a, a_ptr`
- `load b, b_ptr`
- `add c, a, b`
- `store c, c_ptr`

**Masked Filter**

- `a_ptr = a_base * thread_id`
- `load a, a_ptr`
- `branch a = 0`
- `op0`
- `op1`
- `...`
DLP Pattern #4: GPU/SIMT

Programmer’s Logical View

Microthread Block w/ 4 Microthreads

Typical Microarchitecture

Example: Fermi
DLP Pattern #5: Vector-Thread (VT)

Programmer’s Logical View

Vector and Scalar Control Flow Instructions

Vector and Scalar Memory Instructions

Vector-Vector Add

```
loop:
    vload A, a_ptr
    vload B, b_ptr
    vfetch ut_code
    vstore C, c_ptr
    a_ptr++
    b_ptr++
    c_ptr++
    branch
```

Masked Filter

```
loop:
    vload A, a_ptr
    vfetch ut_code
    a_ptr++
    branch
    ut_code:
    add c, a, b
    stop
```

Yunsup Lee / UC Berkeley Par Lab
DLP Pattern #5: Vector-Thread (VT)

Programmer’s Logical View

Examples: Scale, Maven

Typical Micro-architecture

Vector and Scalar Control Flow Instructions

Vector and Scalar Memory Instructions

Yunsup Lee / UC Berkeley Par Lab
Outline

- Data Parallel Architectural Design Patterns
- Microarchitectural Components
- Evaluation Framework
- Evaluation Results
Focus on the Tile

- MIMD Tile
  - On-Chip Network
  - Outer-Level Memory System
  - Mem Ctrl
- Vector Tile with Four Single-Lane Cores
- Vector Tile with One Four-Lane Core
Developed a library of parameterized synthesizable RTL components
- 32-bit integer multiplier, divider
- Single-precision floating-point add, multiply, divide, square root
5-stage Multi-threaded Scalar Core

- Change number of entries in register file (32, 64, 128, 256) to vary degree of multi-threading (1, 2, 4, 8 threads)
Vector Lanes

- Vector registers and ALUs
- Density-time Execution
- Replicate the lanes and execute in lock step for higher throughput
- Vector-SIMD: Flag Registers
Vector Issue Unit

- Vector-SIMD: VIU only handles scheduling, data dependent control done by flag registers
- Maven: VIU fetches instructions, PVFB handles uT branches and does control flow convergence
Vector Memory Unit

- VMU Handles unit stride, constant stride vector memory operations
- Vector-SIMD: VMU handles scatter, gather
- Maven: VMU handles μT loads and stores
Blocking, Non-blocking Caches

- Access Port Width
- Refill Port Width
- Cache Line Size
- Total Capacity
- Associativity

Only for Non-blocking Caches:
- # MSHR
- # secondary misses per MSHR
A Big Design Space …

- Number of entries in scalar register file
  - 32, 64, 128, 256 (1, 2, 4, 8 threads)
- Number of entries in vector register file
  - 32, 64, 128, 256
- Architecture of vector register file
  - 6r3w unified register file, 4x 2r1w banked register file
- Per-bank integer ALU
- Density time execution
- Pending Vector Fragment Buffer (PVFB)
  - FIFO, 1-stack, 2-stack
Outline

- Data Parallel Architectural Design Patterns
- Microarchitectural Components
- Evaluation Framework
- Evaluation Results
Programming Methodology

- Use GCC C++ Cross Compiler (which we ported)
- MIMD
  - Custom application-scheduled lightweight threading lib
- Vector-SIMD
  - Leverage built-in GCC vectorizer for mapping very simple regular DLP code
  - Use GCC’s inline assembly extensions for more complicated code
- Maven
  - Use C++ Macros with special library, which glues the control thread and microthreads
  - Automatic vector register allocation added to GCC
# Microbenchmarks & Application Kernels

## Microbenchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
<th>Irregularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>vvadd</td>
<td>1000 element FP vector-vector add</td>
<td>Regular</td>
</tr>
<tr>
<td>bsearch</td>
<td>1000 look-ups into a sorted array</td>
<td>Very Irregular</td>
</tr>
<tr>
<td>bsearch-cmv</td>
<td>inner-loop rewritten with cond. mov</td>
<td>Somewhat Irregular</td>
</tr>
</tbody>
</table>

## Application Kernels

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
<th>Irregularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>viterbi</td>
<td>Decode frames using Viterbi alg.</td>
<td>Regular</td>
</tr>
<tr>
<td>rsort</td>
<td>Radix sort on an array of integers</td>
<td>Slightly Irregular</td>
</tr>
<tr>
<td>kmeans</td>
<td>K-means clustering algorithm</td>
<td>Slightly Irregular</td>
</tr>
<tr>
<td>dither</td>
<td>Floyd-Steinberg dithering</td>
<td>Somewhat Irregular</td>
</tr>
<tr>
<td>physics</td>
<td>Newtonian physics simulation</td>
<td>Very Irregular</td>
</tr>
<tr>
<td>strsearch</td>
<td>Knuth-Morris-Pratt algorithm</td>
<td>Very Irregular</td>
</tr>
</tbody>
</table>
Evaluation Methodology

**Software Toolflow**
- C++ Application
- Cross Compiler
- Target Binary
- Target ISA Sim

**Hardware Toolflow**
- Verilog RTL
- Verilog Simulator
- Synthesis Place&Route
- Gate-Level Model
- Layout
- Verilog Simulator
- Switching Activity
- Power Analysis

**Results**
- Area & Cycle Time
- Cycle Count
- Power

TSMC 65nm Process with Synopsys CAD Toolflow

Yunsup Lee / UC Berkeley Par Lab
Three Example Layouts

- **MIMD Tile**: 1 Core x 4 Lanes
- **Maven Tile**: 4 Cores x 1 Lane

Yunsup Lee / UC Berkeley Par Lab
# Need Gate-level Activity for Accurate Energy Numbers

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Post Place&amp;Route Statistical (mW)</th>
<th>Simulated Gate-level Activity (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIMD 1</td>
<td>149</td>
<td>137-181</td>
</tr>
<tr>
<td>MIMD 2</td>
<td>216</td>
<td>130-247</td>
</tr>
<tr>
<td>MIMD 3</td>
<td>242</td>
<td>124-261</td>
</tr>
<tr>
<td>MIMD 4</td>
<td>299</td>
<td>221-298</td>
</tr>
<tr>
<td>Multi-core Vector-SIMD</td>
<td>396</td>
<td>213-331</td>
</tr>
<tr>
<td>Multi-lane Vector-SIMD</td>
<td>224</td>
<td>137-252</td>
</tr>
<tr>
<td>Multi-core Vector-Thread 1</td>
<td>428</td>
<td>162-318</td>
</tr>
<tr>
<td>Multi-core Vector-Thread 2</td>
<td>404</td>
<td>147-271</td>
</tr>
<tr>
<td>Multi-core Vector-Thread 3</td>
<td>445</td>
<td>172-298</td>
</tr>
<tr>
<td>Multi-core Vector-Thread 4</td>
<td>409</td>
<td>225-304</td>
</tr>
<tr>
<td>Multi-core Vector-Thread 5</td>
<td>410</td>
<td>168-300</td>
</tr>
<tr>
<td>Multi-lane Vector-Thread 1</td>
<td>205</td>
<td>111-167</td>
</tr>
<tr>
<td>Multi-lane Vector-Thread 2</td>
<td>223</td>
<td>118-173</td>
</tr>
</tbody>
</table>
Outline

- Data Parallel Architectural Design Patterns
- Microarchitectural Components
- Evaluation Framework
- Evaluation Results
Efficiency vs. Number of uTs running \textit{bsearch-cmv}

Normalized Tasks / Sec

Normalized Energy / Task

Energy / Task (\mu J)

- ctrl
- reg
- mem
- fp
- int
- cp
- i$
- d$
- leak

Normalized Energy / Task

Normalized Tasks / Sec
Efficiency vs. Number of uTs running `bsearch-cmv`

- Faster
- Lower Energy

Normalized Energy / Task

Energy / Task (μJ)
Efficiency vs. Number of uTs running bsearch-cmv

Normalized Energy / Task

Normalized Tasks / Sec

Energy / Task (uJ)

r32

r64

ctrl
reg
mem
fp
int
cp
d$

leak
Efficiency vs. Number of uTs running bsearch-cmv
6r3w Vector Register File is Area Inefficient

MIMD Tile

Vector-Thread Tile

Normalized Area

1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

132
164
128
1256

ctrl
reg
mem
fp
int
cp
i$
d$

Efficiency vs. Number of uTs with Banking running \textit{bsearch-cmv}

![Graph showing normalized tasks per second and normalized energy per task.

- Normalized Tasks / Sec
- Normalized Energy / Task

- Labels: r32, r64, r128, r256
- Energy / Task (uJ)

- Categories: ctrl, reg, mem, fp, int, cp, i$, d$, leak

- Data points for mimd-c4, vt-c4v1, vt-c4v1+b]
Efficiency vs. Number of uTs with Per-Bank Integer ALU running bsearch-cmv

Normalized Tasks / Sec

Normalized Energy / Task

Energy / Task (uJ)

mimd-c4
vt-c4v1
vt-c4v1+b
vt-c4v1+bi
r32
r64
r128
r256
ctrl
reg
mem
fp
int
cp
i$
de$
leak
1.0 1.4 1.8 2.2 2.6

Yunsup Lee / UC B
Results running `bsearch` compared to `bsearch-cmv`

Results of Design Space Exploration
Apply Density-Time Execution
Convergence Scheme: 2-Stack PVFB

Yunsup Lee / UC Berkeley Par Lab
Results Running Application Kernels

Normalized Energy / Task

Normalized Tasks / Second

Normalized Tasks / Second / Area

viterbi rsort kmeans dither physics strsearch
Results Running Application Kernels

Performance

Normalized Tasks / Second

Performance per Unit Area

Normalized Tasks / Second / Area

Yunsup Lee / UC Berkeley Par Lab
Results Running Application Kernels

More Irregular

Normalized Energy / Task

Normalized Tasks / Second / Area

Yunsup Lee / UC Berkeley Par Lab
Multi-threading is not Effective on DLP Code

Normalized Energy / Task

Normalized Tasks / Second

Normalized Energy / Task

Normalized Tasks / Second / Area
Vector-SIMD is Faster and/or More Efficient than MIMD

No Vector-SIMD Implementation

Too hard to map
Maven Vector-Thread is More Efficient than Vector-SIMD

Normalized Energy / Task vs. Normalized Tasks / Second

Normalized Energy / Task vs. Normalized Tasks / Second / Area

viterbi rsort kmeans dither physics strsearch
Multi-Lane Tiles are More Efficient than Multi-Core Tiles
Comparing vector load/stores vs. uT load/stores running \textit{vvadd}
uT load/stores are Inefficient

9x Slower
5x More Energy
Memory Coalescing Helps, but Still Far Off

\[ \text{Normalized Tasks / Sec} \]

\[ \text{Normalized Energy / Task} \]

- \text{uT ld/st}
- \text{uT ld/st + mem coalescing}
- \text{vec ld/st}
Conclusions

- Vector architectures are more area and energy efficient than MIMD architectures on regular DLP and (surprisingly) on irregular DLP.

- The Maven vector-thread architecture is a promising alternative to traditional vector-SIMD architectures, providing greater efficiency and easier programmability.

- Using real RTL implementations and a standard ASIC toolflow is necessary to compare energy-optimized future architectures.

This work was supported in part by Microsoft (Award #024263) and Intel (Award #024894, equipment donations) funding and by matching funding from U.C. Discovery (Award #DIG07-10227).