CS Brown Bag Lunch

Microarchitectural Mechanisms to Exploit Value Structure in SIMT Architectures

Christopher Batten

Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

Spring 2013
Motivating Trends in Computer Architecture

Trend 1
Power & energy constrain all systems

Trend 2
Transition to multicore processors

Trend 3
Inevitable end of Moore’s law

Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond
Pervasive Heterogeneous Specialization

Latency-Optimized Tile: few large out-of-order cores

Throughput-Optimized Tile: many small in-order cores

ASLA Tile: application specific loop accelerator

Fine-Grain SIMT Tile: flexible data-parallel accelerator
Projects Within the Batten Research Group

Data-Parallel Specialization
- Fine-Grain Single-Instruction Multiple-Thread Architectures
- XPC: Explicit-Parallel-Call Architectures

Domain-Specific Specialization
- Polymorphic Algorithm and Data-Structure Specialization
- Coarse-Grain Reconfigurable Accelerators

Vertically Driven Research Approach
- Python modeling framework
- FPGA prototypes/emulation
- Architecture test chips

Chip-Level Interconnection Networks
- Realistic On-Chip Networks
- Nanophotonic Networks
- Networks for Silicon Interposers
Agenda

Research Overview

Value Structure in SIMT Kernels

FG-SIMT Baseline Microarchitecture

FG-SIMT Compact Affine Execution

Evaluation
__global__ void vsadd_kernel( int y[], int a ) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    y[idx] = y[idx] + a;
}
...
void vsadd( int y[], int a, int n ) {
    // move data from CPU to GPU
    vsadd_kernel<<<32,n/32>>>( y, a );
    // move data from GPU to CPU
}
Fine-Grain SIMT Programming Model

```c
__kernel__ void vsadd_kernel( int y[], int a ) {
    int idx = fgsimt::init_kernel( y, a );
    y[idx] = y[idx] + a;
}

...  
void vsadd( int y[], int a, int n ) {
    // distribute work among control threads
    // x = base pointer for this control thread
    fgsimt::launch_kernel( n/32, &vsadd_kernel, x, a );
}
```
Fragment from Viterbi Application

___kernel__ void
calc_fwd_paths_kernel( ... ) {
    int idx = fgsimt::init_kernel( ... );
    ...

    // Inner loop
    for (int j = 0; j < vrate; j++)
        metric += bt_ptr[idx+j*STATES/2] ^ symbols[s*vrate+j];
    ...

    // More complicated array indexing
    m0 = old_error[idx] + metric;
    m1 = old_error[idx+STATES/2] + (max - metric);
    m2 = old_error[idx] + (max - metric);
    m3 = old_error[idx+STATES/2] + metric;
    ...

    // Data-dependent control flow
    new_error[2*idx] = decision0 ? m1 : m0;
    new_error[2*idx+1] = decision1 ? m3 : m2;
    ...
}
Control and Memory Access Structure

Regular Data Access
- Regular Control Flow
  ```cpp
  for ( i = 0; i < n; i++ )
  C[i] = A[i] + B[i];
  ```

Irregular Data Access
- Irregular Control Flow
  ```cpp
  for ( i = 0; i < n; i++ )
  C[i] = x * A[i] + B[2*i];
  ```

Regular Data Access
- Irregular Control Flow
  ```cpp
  for ( i = 0; i < n; i++ )
  x = ( A[i] > 0 ) ? y : z;
  C[i] = x * A[i] + B[i];
  ```

Irregular Data Access
- Irregular Control Flow
  ```cpp
  for ( i = 0; i < n; i++ )
  if ( A[i] > 0 )
    C[i] = x * A[i] + B[i];
  ```

Regular Data Access
- Regular Control Flow
  ```cpp
  for ( i = 0; i < n; i++ )
  E[C[i]] = D[A[i]] + B[i];
  ```

Irregular Data Access
- Irregular Control Flow
  ```cpp
  for ( i = 0; i < n; i++ )
    C[i] = true;
  ```
FG-SIMT Pseudo-Assembly Example

```c
__kernel__ void
ex_kernel( int y[], int a ) {
  int idx
  = fgsimt::init_kernel(y,a);
  y[idx] = y[idx] + a;
  if ( y[idx] > THRESHOLD )
    y[idx] = Y_MAX_VALUE;
}
```

```
ex_kernel:
  load R_a, M[A]
  load R_ybase, M[Y]
  add R_yptr, R_ybase, IDX
  load R_y, M[R_yptr]
  add R_y, R_y, R_a
  store R_y, M[R_yptr]
  branch R_y, THRESHOLD
  imm R_imm, Y_MAX_VALUE
  store R_imm, M[R_yptr]
  stop
```
Control Structure in FG-SIMT Kernels

Good Control Structure

- Value Structure

- FG-SIMT Baseline

- FG-SIMT Affine

- Evaluation

ex_kernel:

\[
\begin{align*}
&\text{load } R_a, M[A] \\
&\text{load } R_ybase, M[Y] \\
&\text{add } R_{yptr}, R_ybase, \text{IDX} \\
&\text{load } R_y, M[R_{yptr}] \\
&\text{add } R_y, R_y, R_a \\
&\text{store } R_y, M[R_{yptr}] \\
&\text{branch } R_y, \text{THRESHOLD} \\
&\text{imm } R_{imm}, Y_{MAX\_VALUE} \\
&\text{store } R_{imm}, M[R_{yptr}] \\
&\text{stop}
\end{align*}
\]

μTs

Poor Control Structure (data dependent)
Memory Access Structure in FG-SIMT Kernels

**ex_kernel:**
- load R_a, M[A]
- load R_ybase, M[Y]
- add R_yptr, R_ybase, IDX
- load R_y, M[R_yptr]
- add R_y, R_y, R_a
- store R_y, M[R_yptr]
- branch R_y, THRESHOLD
- imm R_imm, Y_MAX_VALUE
- store R_imm, M[R_yptr]
- stop

**Good Memory Access Structure**
- "Unit-Stride" Load
- "Unit-Stride" Store

**Shared Load**

**Poor Memory Access Structure**
- (data dependent)
- Scatter
Value Structure in FG-SIMT Kernels

Affine Value Structure:
\[ V(i) = b + i \times s \]

Affine Arithmetic
\[
V_0(i) = b_0 + i \times s_0 \quad V_1(i) = b_1 + i \times s_1 \\
V_0(i) + V_1(i) = (b_0 + b_1) + i \times (s_0 + s_1)
\]

Affine branches and affine memory operations are also possible

ex_kernel:
```
load R_a, M[A]
load R_ybase, M[Y]
add R_yptr, R_ybase, IDX
load R_y, M[R_yptr]
add R_y, R_y, R_a
store R_y, M[R_yptr]
branch R_y, THRESHOLD
imm R_imm, Y_MAX_VALUE
store R_imm, M[R_yptr]
stop
```

- **Uniform values across threads**: If both inputs are uniform, we can execute the instruction once on the control processor.

- **Affine values across threads**: If inputs are affine/uniform, we can still potentially execute instruction once on the control processor.

- **Generic values across threads (i.e., no structure)**: We must ensure values are expanded and explicitly execute the instruction for each thread.
Characterizing Structure in FG-SIMT Kernels

- Vector-Vector Complex Multiply
- Viterbi Decoder
- Dense Matrix Multiplication
- Guassian Image Blur Under Mask
- Searching for Multiple Strings in Multiple Documents
- Breadth-First Search from Source to All Other Nodes
Agenda

Research Overview

Value Structure in SIMT Kernels

FG-SIMT Baseline Microarchitecture

FG-SIMT Compact Affine Execution

Evaluation
FG-SIMT Microarchitecture: Regular Control Flow

ex_kernel:
load  R_a, M[A]
load  R_ybase, M[Y]
add   R_yptr, R_ybase, IDX
load  R_y, M[R_yptr]
add   R_y, R_y, R_a
store R_y, M[R_yptr]
stop

Next iteration of hardware stripmined loop
FG-SIMT Microarchitecture: Irregular Control Flow

ex_kernel:
load R_a, M[A]
load R_ybase, M[Y]
add R_yptr, R_ybase, IDX
load R_y, M[R_yptr]
add R_y, R_y, R_a
store R_y, M[R_yptr]
branch R_y, THRESHOLD
imm R_imm, Y_MAX_VALUE
store R_imm, M[R_yptr]
stop

Finish pending SIMT fragment
FG-SIMT Detailed Microarchitecture

- Eight SIMT lanes
- Dynamic reconvergence
- Five vector functional units with support for chaining
- Multi-ported banked regfile with support for executing 32 threads at a time
- Shared load cache for kernel input parameters
- Memory coalescing to dynamically create wide accesses

Eight SIMT lanes
Dynamic reconvergence
Five vector functional units with support for chaining
Multi-ported banked regfile with support for executing 32 threads at a time
Shared load cache for kernel input parameters
Memory coalescing to dynamically create wide accesses
Agenda

Research Overview

Value Structure in SIMT Kernels

FG-SIMT Baseline Microarchitecture

FG-SIMT Compact Affine Execution

Evaluation
Compact Affine Execution: Affine Arithmetic

Add extra bits to track which values are uniform, affine, generic

Add affine SIMT register file and affine functional unit for affine arithmetic
Compact Affine Execution: Affine Memory Operations

ex_kernel:
load \( R_a, M[A] \)
load \( R_{ybase}, M[Y] \)
add \( R_{yptr}, R_{ybase}, IDX \)
load \( R_y, M[R_{yptr}] \)
store \( R_y, M[R_{yptr}] \)

Add direct access to SMU for affine memory accesses

Uniform  Affine (not uniform)  Generic
Compact Affine Execution: Without Divergence

ex_kernel:
load R_a, M[A]
load R_ybase, M[Y]
add R_y, M[R_y]
add R_y, R_y, R_a
store R_y, M[R_y]
branch R_y, THRESHOLD
imm R_imm, Y_MAX_VALUE
store R_imm, M[R_y]

Compact affine execution still possible after a branch if there is no divergence
Compact Affine Execution: Affine Branches

ex_kernel:

- load \( R_a, M[A] \)
- load \( R_ybase, M[Y] \)
- add \( R_yptr, R_ybase, IDX \)
- load \( R_y, M[R_y] \)
- add \( R_y, R_y, R_a \)
- store \( R_y, M[R_y] \)
- branch \( R_a, \text{THRESHOLD} \)
- imm \( R_{imm}, Y_{MAX\_VALUE} \)
- store \( R_{imm}, M[R_{yptr}] \)

If branch operands are uniform, then branch can be completely resolved in CP.
Compact Affine Execution: With Divergence

Compact affine execution still possible, but must expand out result
Store cannot now be executed compactly

ex_kernel:
load R_a, M[A]
load R_ybase, M[Y]
add R_y, R_ybase, IDX
load R_y, M[R_yptr]
add R_y, R_y, R_a
store R_y, M[R_yptr]
branch R_y, THRESHOLD
imm R_imm, Y_MAX_VALUE
store R_imm, M[R_yptr]
stop
Agenda

Research Overview
Value Structure in SIMT Kernels
FG-SIMT Baseline Microarchitecture
FG-SIMT Compact Affine Execution

Evaluation
Software/Hardware Evaluation Methodology

### Software Toolflow
- C++ App Using FG-SIMT API
- Native Compiler
  - Native Binary
- FG-SIMT Compiler
  - CEVS Binary
- FG-SIMT ISA Sim

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

### Results
- Cycle Count
- Area & Cycle Time
- Energy

TSMC 40nm Process with Synopsys CAD Toolflow
## Application Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs</td>
<td>Breadth-first search from source to every other node</td>
</tr>
<tr>
<td>bilat</td>
<td>Bilateral image filtering, optimized Taylor series for intensity</td>
</tr>
<tr>
<td>bsearch</td>
<td>Parallel binary searches in sorted linear array of key/value pairs</td>
</tr>
<tr>
<td>cmult</td>
<td>Vector-vector complex single-precision multiplication</td>
</tr>
<tr>
<td>conv</td>
<td>1D spatial convolution with large 20-element kernel</td>
</tr>
<tr>
<td>dither</td>
<td>Floyd-Steinberg image dithering from gray-scale to black-and-white</td>
</tr>
<tr>
<td>kmeans</td>
<td>KMeans clustering</td>
</tr>
<tr>
<td>mfilt</td>
<td>Apply Gaussian blur filter to gray-scale image under mask</td>
</tr>
<tr>
<td>rgb2cmyk</td>
<td>RGB-to-CMYK color conversion</td>
</tr>
<tr>
<td>rsort</td>
<td>Incremental radix sort of integers</td>
</tr>
<tr>
<td>sgemm</td>
<td>Dense single-precision matrix-matrix multiply</td>
</tr>
<tr>
<td>strsearch</td>
<td>Knuth-Morris-Pratt search for multiple strings in multiple docs</td>
</tr>
<tr>
<td>viterbi</td>
<td>Decode frames of convolutionally encoded data using Viterbi algo</td>
</tr>
</tbody>
</table>
FG-SIMT has comparable area to a multicore processor with equivalent floating-point and memory bandwidth resources.

Compact affine execution adds relatively little overhead.
FG-SIMT Cycle-Level Performance Results

- mcore
- fgsimt
- fgsimt+a
- fgsimt+ab
- fgsimt+abm

Speedup compared to baseline.

- cmult
- mfilter
- bsearch
- viterbi
- rsort
- dither
- strsearch
- rgb2cmyk
- conv
- kmeans
- bfs
- sgemm
- bilat
FG-SIMT Energy-Performance Results

![Graph showing the energy-performance results for various tasks. The x-axis represents Task/Second, while the y-axis represents Energy/Task. Different tasks are plotted as points on the graph, with labels such as cmult, dither, rgb2cmyk, bfs, strsearch, bilaterial, rsort, and viterbi. The graph illustrates the trade-off between energy and performance for these tasks.]
General-Purpose SIMT Microarchitecture
Exploiting Value Structure in General-Purpose SIMT
Take-Away Points

SIMT kernels contain **ample value structure** that is not exploited by current SIMT microarchitectures.

Compact affine execution of **affine arithmetic, branches, and memory operations** are a promising way to exploit value structure for improved performance and reduced energy.

For more information see our ISCA’13 paper.

Christopher Torng  Shreesha Srinath  Ji Kim  Christopher Batten  Derek Lockhart