Control Flow Coalescing on a Hybrid Dataflow/von Neumann GPGPU

Dani Voitsechov    Yoav Etsion
Technion - Israel Institute of Technology
Electrical Engineering and Computer Science Departments
Massively Parallel Computing

• Massively parallel programming models (CUDA/OpenCL) are popular in HPC (High-Performance Computing)

• Programmer decomposes program into small tasks
  • Many instances of the same task are runnable at any given time
  • Decomposed code may yield fine-grain tasks that can be translated to data-flow graphs

• GPUs yield more FLOPS per Watt
  • Are integrated in systems from mobile to supercomputers

• But... GPGPUs still suffer from von-Neumann inefficiencies
von-Neumann Inefficiencies

• The von-Neumann model carries inherent inefficiencies

• Fetch/Decode/Issue each instruction
  • Even though most instructions come from loops

• Explicit storage needed for communicating values between instructions
  • Data travels back and forth between execution units and explicit storage

<table>
<thead>
<tr>
<th>Component</th>
<th>Inst. fetch</th>
<th>Pipeline registers</th>
<th>Data cache</th>
<th>Register file</th>
<th>Control</th>
<th>FU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power [%]</td>
<td>33%</td>
<td>22%</td>
<td>19%</td>
<td>10%</td>
<td>10%</td>
<td>6%</td>
</tr>
</tbody>
</table>

[Understanding Sources of Inefficiency in General-Purpose Chips, Hameed et al., ISCA10]
Alternative – Dataflow Based MT-CGRF

• Using Multi-Threaded Coarse Grain Reconfigurable Fabric (MT-CGRF)
  • Eliminates von-Neumann inefficiencies by using spatial computing (dataflow)
  • Increases utilization by using simultaneous multithreading

• To execute a CUDA kernel on a MT-CGRF
  • Transform the sequential code into a dataflow graph
  • Map and route the MT-CGRF
  • Stream multiple threads through the fabric
int temp1 = a[threadId] * b[threadId];
int temp2 = 5 * temp1;
if (temp2 > 255) {
temp2 = temp2 >> 3;
result[threadId] = temp2;
} else
result[threadId] = temp2;

<table>
<thead>
<tr>
<th>a</th>
<th>threadIdx</th>
<th>entry</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMM_5</td>
<td>S_LOAS1</td>
<td>S_LOAD2</td>
<td></td>
</tr>
<tr>
<td>ALU1_mul</td>
<td>ALU2_mul</td>
<td>JOIN1</td>
<td></td>
</tr>
<tr>
<td>IM_3</td>
<td>ALU4_ashl</td>
<td>ALU3_icmp</td>
<td>IMM_256</td>
</tr>
<tr>
<td>if_else</td>
<td>if_then</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S_SOTRE3</td>
<td>result</td>
<td></td>
<td>S_SOTRE4</td>
</tr>
</tbody>
</table>
**SGMF: Simultaneous Multithreading on CGRAs**

- **Single Graph Multiple Flows (SGMF)**
  - Control-dataflow graph (CDFG) is mapped to the grid
  - Maximal spatial parallelism within each thread (ILP)
  - Pipelining and dynamic dataflow achieve thread-level-parallelism (TLP)

- **But**
  - Limited to small kernels
  - When all control paths are mapped the grid cannot be fully utilized

[Single-Graph Multiple Flows: Energy Efficient Design Alternative for GPGPUs, Voitsechov and Etsion, ISCA14]
VGIW

- Vector Graph Instruction Word (VGIW)
  - Basic blocks are represented as compound graph instruction words (GIW)
  - GIWs concurrently execute for vectors of threads

- von Neumann control flow dynamically schedules VGIWs on the CGRA

- Threads are coalesced according to their control flow and are executed concurrently
Execution and Machine Model
VGIW – A Hybrid Solution

Code

kernel() <<8 threads>>
  Prologue //BB1
  if (threadIdx == either (1,3,7))
    func1() //BB2
  else
    func2() //BB3
    if (threadIdx == either (2,7))
      func3() //BB4
    else
      func4() //BB5
  Epilogue() //BB6

Control Flow

Tids 1-8
BB1

Tids 1,3,8
BB2

Tids 2,4-7
BB3

Tids 2,7
BB4

Tids 4-6
BB5

Tids 1-8
BB6
VGIW – A Hybrid Solution

Underutilized due to temporal (and spatial) division of the HW between the BBs

Underutilized due to spatial division of the HW between the BBs

Fully utilized
Machine Model

• Each thread may take a different control path
  • Control nodes update the thread vector of the next executed block for each thread
  • Threads that need to execute a block are dynamically coalesced into a thread vector
  • Thread vectors are stored in the Control-Vector-Table (CVT)

• Temporal Values may stay alive between blocks
  • Values that are alive in-between blocks are written to the Live-value-cache (LVC)
  • The LVC is accessed 10x less frequently than a GPGPU RF
Before executing BB1:
The MT-CGRF is configured with the graph of BB1.

Before executing BB1:
All threads execute the prologue \(\rightarrow\) Entire BB1’s thread vector is set to 1’s.

Before executing BB1:
No temporal values are alive \(\rightarrow\) The LVC is empty.
After the execution of BB1:
The control path has diverged, threads 1, 3, 8 are registered to execute BB2 and threads 2, 4-7 BB3

The prologue (BB1) has generated a *live-value* which is stored in the LVC

Before executing BB2: The MT-CGRF is configured with the graph of BB2.
After the execution of BB2:
threads 1,3,8 are registered to execute BB6 and threads 2,4-7 will now execute BB3

BB2 had generated a new live-value (LV2) for threads 1,3,8 which is stored in the LVC

Before executing BB3:
The MT-CGRF is configured with the graph of BB3.

After the execution of BB2: threads 1,3,8 are registered to execute BB6 and threads 2,4-7 will now execute BB3

BB2 had generated a new live-value (LV2) for threads 1,3,8 which is stored in the LVC
After the execution of BB3:
The control path had diverged again threads 4-6 are registered to execute BB4 and threads 2,7 will now execute BB4
BB3 had generated a new live-value (LV3) for threads 2,4-7 which is stored in the LVC

Before executing BB4:
The MT-CGRF is configured with the graph of BB4.

After the execution of BB3:
The control path had diverged again threads 4-6 are registered to execute BB4 and threads 2,7 will now execute BB4
**Execution Flow**

*Before executing BB5:*

The MT-CGRF is configured with the graph of BB5.

*After the execution of BB4:*

All threads but 4-6 is registered to execute BB6, threads 4-6 will now execute BB5.

BB4 has generated a new *live-value* (LV4) for threads 2,7 which is stored in the LVC.
After the execution of BB1-BB5 all the control path had converged back to the epilogue (BB6).

BB5 has generated a new live-value (LV5) for threads 4-6 which is stored in the LVC.

LV3 was used by BB4 and BB5 since it’s not a live any more the space in the LVC can be reclaimed.

Before executing BB6: The MT-CGRF is configured with the graph of BB6.

BB6}

Control Vector Table

<table>
<thead>
<tr>
<th>Tid</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BB2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BB3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BB4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BB5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BB6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Live Value Cache

<table>
<thead>
<tr>
<th>LV1</th>
<th>LV2</th>
<th>LV3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>1</td>
<td>3,4,6</td>
</tr>
<tr>
<td></td>
<td>2,7,4-6</td>
<td>2,4-7</td>
</tr>
</tbody>
</table>
Architecture
The VGIW Architecture Overview

• Coarse grained reconfigurable architecture

• Designed for massive multithreading

• A grid of computational and control units surrounded by LDST units on the perimeter

• The nodes on the grid are connected by an interconnect implemented by reconfigurable switches
Full Architecture

Legend

SCU—Special Compute Unit  CVT—Control Vector Table
SLU—Split/Len Unit  LVC—Low Value Cache
SBS—Basic Block Scheduler

Special Compute Unit (SCU)
Evaluation
Methodology

- The main HW blocks were Implemented in Verilog
- Synthesized to a 65nm process
  - Validate timing and connectivity
  - Estimate area and power consumption
- Cycle accurate simulations based on GPGPUSim
  - We Integrated synthesis results into the GPGPUSim/Wattch power model
- Benchmarks from Rodinia suite
  - CUDA kernels, compiled as VGIWs
System Configuration

- In the evaluation, a processor configuration similar to the Fermi is used
  - Clock: 1.4GHz
  - 32x compute tiles; 32x control tiles; 16x LD/ST tiles; 16x LVU tiles; 12x SCUs

- Fermi memory system
  - L1 cache: 64KB (SGMF does not have shared memory)
  - L2 cache: 768KB
  - Latencies as in default GPGPUSim Fermi configuration
Energy Efficiency

Energy efficiency vs. SGMF

Energy efficiency vs. Nvidia Fermi
Conclusions

• von-Neumann engines have inherent inefficiencies
  • Throughput computing can benefit from dataflow/spatial computing

• Scheduling blocks according to the von-Neumann semantics increases utilization of the dataflow fabric
  • Dynamic coalescing overcomes the control divergence problem

• VGIW can potentially achieve much better performance/power than current GPGPUs
  • On average x3 speedup and 33% energy saving
  • Need to tune the memory system

• Greatly motivates further research
  • Compilation, VGIW block granularity, memory coalescing on MT-CGRFs
Thank you!

Questions...?