Harmonica GPU

Objectives

- Detailed look at the implementation of a SIMT GPU
- Example of the type of information propagated down the pipeline
- Basis for the next assignment and the default project
Reading

- C. Kersey, “HARP Instruction Set Manual”
- CHDL Architecture Description
  - [https://github.com/cdkersey/harmonica2](https://github.com/cdkersey/harmonica2)
  - Note that this is under active development so the current code base/definitions may be changing

The Harmonica GPU

- An instance of the HARP family of ISAs
  - SIMT oriented RISC-like ISAs
- Parametric SIMT GPU
  - Datapath width, instruction encodings, #registers are configurable
- Lightweight, simplified pipeline with multi-warp execution
  - No thread blocks
  - Nested parallelism
- Designed as a memory side or CPU accelerator for data intensive computing
System Organization

- Parametric architecture
  - Key Parameters - #SMs, #lanes/SM, #registers, and data-widths.

Microarchitecture

- Customizable, multithreaded, SIMT soft core
  - Generated from an architecture specification
  - Supported by a generated HARP Tool assembler/linker/emulator
  - Small - ~1500 lines of C++
CAD Tool Chain: CHDL

- Vertically integrated C++ CAD environment for microarchitecture generation
- Strong emphasis on code reusability
- Same model for generating both gate level and system level simulations
- CAD tool interfaces
- v2 running in Altera FPGAs
- Export to synthesizable Verilog technology mapping and SPICE

A CHDL Program

In-memory Design Representation

<table>
<thead>
<tr>
<th>Nodes, node-implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
</tr>
<tr>
<td>Registers</td>
</tr>
<tr>
<td>Vectors</td>
</tr>
<tr>
<td>Modules</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Cores, caches, routers</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

C++ Program

- Elaboration
- Optimization
- Simulation
  - Coarse
  - Fine
  - SPICE
- Technology Mapping
Interrupt Handling

- Jump to handler code in lane 0
- Shadow register file for lane 0
- Option to run parallel handler – warp scale
Microarchitecture: Schedule

Microarchitecture: Fetch
Harmonica Instruction Format

Word Encoding

![Word Encoding Diagram]

Byte Encoding

![Byte Encoding Diagram]

Microarchitecture: PRF Access

![Microarchitecture Diagram]

\( L = \# \text{lanes} \)

- **pmask**
- **val0**
- **val1**

From previous stage

- **Predicate mask** <\( L \)>
- **Values of predicate registers** <\( L \)>
- **Values of predicate registers** <\( L \)>

(13)

(14)
Microarchitecture: GPR Access

Register File Organization
Microarchitecture: Cache Request

- Starter
- Fetch Unit
- Fetch
- Sched
- Int. Mem. Port
- PredRegs
- PLU
- Mul
- ALU
- Div
- Jmp
- Ld/St
- Switch
- Arbiter
- Splitter
- Exec
- GPRegs
- Int/FP
- GPRegs
- Data
- Mem
- Port
- Inst.
- WID
- lane
- R/W
- mask
- addr
- data

Microarchitecture: Cache Response

- Starter
- Fetch Unit
- Fetch
- Sched
- Int. Mem. Port
- PredRegs
- PLU
- Mul
- ALU
- Div
- Jmp
- Ld/St
- Switch
- Arbiter
- Splitter
- Exec
- GPRegs
- Int/FP
- GPRegs
- Data
- Mem
- Port
- Inst.
- WID
- lane
- Q

- Q – cache line number of words
Microarchitecture: Memory Request

Fetch N B-bit bytes from location addr (word addressed)

Microarchitecture: Memory Response

Unique ID to pair rq & rply
Load link store conditional
Success
Microarchitecture: FU Out

(21)

Microarchitecture: FU Out (2)

(22)
Microarchitecture: FU Out (3)

Microarchitecture: GPR Writeback
Microarchitecture: Next Instruction

Microarchitecture: Stalled Warps

Reconvergence stack managed in the branch unit (more later)

Warp wait at the barrier unit for all warps to reach a barrier

Warp wait at the LD/ST unit for memory request to be completed
Some Miscellaneous Observations

- Recirculating warp model of execution
  - Waiting warps distributed through the data path
    - Barriers, divergent warps, memory accesses
- Warps can spawn other warps \(\rightarrow\) nested parallelism
- Elementary scheduling, and fetch policies
- Supports kernel mode execution and interrupts (lane 0)

Summary

- Light weight GPU for embedded (e.g., in memory) applications
- RISC-like ISA with support for thread/warp spawning
- Synthesizable implementation for FPGA implementation
  - Can experiment with microarchitecture optimizations