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
  - 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++
Microarchitecture: Schedule

State | PC | AM | WID | Next_WID

Warp state such as divergent, user/kernel, etc
Program Counter
Activity Mask
Warp ID
ID of the next Warp if spawned

Microarchitecture: Fetch

State | PC | AM | WID | Next_WID | Instr

Warp state such as divergent, user/kernel, etc
Program Counter
Activity Mask
Warp ID
ID of the next Warp if spawned
Instruction
Harmonica Instruction Format

Word Encoding

Byte Encoding

Microarchitecture: PRF Access

L = \#lanes

From previous stage

Predicate mask <L>

Values of predicate registers <L>

Values of predicate registers <L>
Microarchitecture: GPR Access

Register File Organization

Similar Organization for the Predicate Register File
Microarchitecture: Cache Request

Microarchitecture: Cache Response

- 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

Microarchitecture: FU Out (2)
Microarchitecture: FU Out (3)

Microarchitecture: GPR Writeback
Microarchitecture: Next Instruction

- Microarchitecture: Next Instruction
- Starter
- Fetch Unit
- Write-Back Logic
- Write-Back Logic
- ALU
- GP Regs
- GP Regs
- Data Mem Port
- Data Mem Port

- boolean
- Target of warp to spawn
- State
- PC
- AM
- WID
- Next_WID

- Same as output of Schedule()

Microarchitecture: Stalled Warps

- Microarchitecture: Stalled Warps
- Reconvergence stack managed in the branch unit (more later)
- Warps wait at the barrier unit for all warps to reach a barrier
- Starter
- Fetch Unit
- Write-Back Logic
- Write-Back Logic
- All Regs
- All Regs
- Data Mem Port
- Data Mem Port

- Warps wait at the LD/ST unit for memory request to be completed

- boolean
- Target of warp to spawn

-(21)

-(22)
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 → nested parallelism

- Elementary scheduling, and fetch policies

- Supports kernel mode execution and interrupts (lane 0)