Simty: Generalized SIMT execution on RISC-V

CARRV 2017

Sylvain Collange
INRIA Rennes / IRISA
sylvain.collange@inria.fr
From CPU-GPU to heterogeneous multi-core

Yesterday (2000-2010)
- Homogeneous multi-core
- Discrete components

Today (2011-...)
Heterogeneous multi-core
- Physically unified
  CPU + GPU on the same chip
- Logically separated
  Different programming models, compilers, instruction sets

Tomorrow
- Unified programming models?
- Single instruction set?
From CPU-GPU to heterogeneous multi-core

- **Yesterday (2000-2010)**
  - Homogeneous multi-core
  - Discrete components

- **Today (2011-...)**
  Heterogeneous multi-core
  - **Physically unified**
    CPU + GPU on the same chip
  - **Logically separated**
    Different programming models, compilers, instruction sets

- **Tomorrow**
  - Unified programming models?
  - Single instruction set?
  - **Defining the general-purpose throughput-oriented core**
Outline

- Stateless dynamic vectorization
  - Functional view
  - Implementation options

- The Simty core
  - Design goals
  - Micro-architecture
The enabler: dynamic inter-thread vectorization

- **Idea:** Microarchitecture aggregates threads together to assemble vector instructions.

  - SPMD Program
    ```
    add r1, r3
    mul r2, r1
    ```

  - Threads
    ```
    add
    mul
    add
    mul
    add
    mul
    ```

  - Vector instructions
    ```
    add
    mul
    add
    mul
    add
    mul
    ```

- **Force threads to run in lockstep:**
  - Threads execute the same instruction at the same time (or do nothing)

- **Generalization of GPU's SIMT for general-purpose ISAs**

- **Benefits vs. static vectorization**
  - **Programmability:** Software sees only threads, not threads + vectors
  - **Portability:** Vector width is not exposed in the ISA
  - **Scalability:** + threads → larger vectors or more latency hiding or more cores
  - **Implementation simplicity:** Handling traps is straightforward
Goto considered harmful?

<table>
<thead>
<tr>
<th>RISC-V</th>
<th>NVIDIA</th>
<th>NVIDIA</th>
<th>Intel GMA</th>
<th>Intel GMA</th>
<th>AMD</th>
<th>AMD</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tesla Gen4</td>
<td>Fermi Gen4</td>
<td>SB Gen4</td>
<td>SB Gen4</td>
<td>R500 SB</td>
<td>R600 SB</td>
<td>Cayman SB</td>
</tr>
<tr>
<td>jal</td>
<td>bar</td>
<td>bar</td>
<td>jmp</td>
<td>jmp</td>
<td>jump</td>
<td>push</td>
<td>push</td>
</tr>
<tr>
<td>jalr</td>
<td>bra</td>
<td>jmpi</td>
<td>jmpi</td>
<td>jmpi</td>
<td>loop</td>
<td>push else</td>
<td>push else</td>
</tr>
<tr>
<td>bXX</td>
<td>brk</td>
<td>if</td>
<td>if</td>
<td>else</td>
<td>loop</td>
<td>pop</td>
<td>pop</td>
</tr>
<tr>
<td>ecall</td>
<td>brkpt</td>
<td>else</td>
<td>else</td>
<td>endif</td>
<td>endloop</td>
<td>pop wqm</td>
<td>push wqm</td>
</tr>
<tr>
<td>ebreak</td>
<td>cal</td>
<td>endif</td>
<td>endif</td>
<td>case</td>
<td>rep</td>
<td>else wqm</td>
<td>pop wqm</td>
</tr>
<tr>
<td>Xret</td>
<td>cont</td>
<td>do</td>
<td>while</td>
<td>break</td>
<td>cont</td>
<td>jump_any</td>
<td>else wqm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reinit</td>
<td>jump_any</td>
</tr>
<tr>
<td></td>
<td>kil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reinit wqm</td>
<td>jump_any</td>
</tr>
<tr>
<td></td>
<td>exit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loop_start</td>
<td>loop_start</td>
</tr>
<tr>
<td></td>
<td>brx</td>
<td>break</td>
<td></td>
<td></td>
<td></td>
<td>loop_start_no_al</td>
<td>loop_start</td>
</tr>
<tr>
<td></td>
<td>cal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loop_start_dx10</td>
<td>loop_start_dx10</td>
</tr>
<tr>
<td></td>
<td>cont</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loop_end</td>
<td>loop_end</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loop_continue</td>
<td>loop_continue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loop_break</td>
<td>loop_break</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>call</td>
<td>call</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>call_fs</td>
<td>call_fs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>return</td>
<td>return</td>
</tr>
<tr>
<td></td>
<td>pbk</td>
<td>push</td>
<td></td>
<td>msave</td>
<td>return</td>
<td>return_fs</td>
<td>return_fs</td>
</tr>
<tr>
<td></td>
<td>pret</td>
<td>pop</td>
<td></td>
<td>mrest</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ret</td>
<td></td>
<td></td>
<td>push</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td>pop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control transfer instructions in GPU instruction sets vs. RISC-V

- GPUs: control flow divergence and convergence is explicit
  - Incompatible with general-purpose instruction sets 😞
Stateless dynamic vectorization

Idea: per-thread PCs characterize thread state

**Code**

```plaintext
if(tid < 2) {
    if(tid == 0) {
        x = 2;
    } else {
        x = 3;
    }
} else {
    x = 3;
}
```

**Program Counters (PCs)**

<table>
<thead>
<tr>
<th>tid=</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Match</td>
<td>→ active</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No match</td>
<td>→ inactive</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Policy: MPC = \(\min(\text{PC}_i)\) inside deepest function

- Intuition: favor threads that are behind so they can catch up
- Earliest reconvergence with code laid out in reverse post order
Control transfer instruction or exception

- Match: execute instruction, update PC
- No match: discard instruction
Functional view

- Arithmetic instruction
  - Min(PC+1) = Min(PC)+1
  - No need to vote again

Match: execute instruction, update PC
No match: discard instruction, do not change PC
Implementation 1: reduction tree

Straightforward implementation of the functional view

- On every branch: compute Master PC from individual PCs
  - Reduction tree to compute $\max(\text{depth})-\min(\text{PCs})$
- On every instruction: compare Master PC with individual PCs
  - Row of address comparators
- Issues: area, energy overheads, extra branch resolution latency
Implementation 2: sorted context table

- Common case: few different PCs
- Order stable in time
- Keep Common PCs+activity masks in sorted heap

![Per-thread PCs and Sorted context table diagram]

- Branch = insertion in sorted context table
- Convergence = fusion of head entries when CPC₁ = CPC₂
- Activity mask is readily available
Outline

- Stateless dynamic vectorization
  - Functional view
  - Implementation options

- The Simty core
  - Design goals
  - Micro-architecture
Simty: illustrating the simplicity of SIMT

Proof of concept for dynamic inter-thread vectorization

- Focus on the core ideas → the RISC of dynamic vectorization
- Simple programming model
  - Many scalar threads
  - General-purpose RISC-V ISA
- Simple micro-architecture
  - Single-issue RISC pipeline
  - SIMD execution units
- Highly concurrent, scalable
  - Interleaved multi-threading to hide latency
  - **Dynamic vectorization** to increase execution throughput
  - Target: hundreds of threads per core
Simty implementation

- Written in synthesizable VHDL
- Runs the RISC-V instruction set (RV32I)
- Fully parametrizable SIMD width, multithreading depth
- 10-stage pipeline
Multiple warps

- Wide dynamic vectorization found counterproductive
  - Sensitive to control-flow and memory divergence
  - Threads that hit in the cache wait for threads that miss
  - Breaks latency hiding capability of interleaved multi-threading

- Two-level approach: partition threads into warps, vectorize inside warps
  - Standard approach on GPUs
Two-level context table

- Cache top 2 entries in the *Hot Context Table* register
  - Constant-time access to $\text{CPC}_i$, activity masks
  - In-band convergence detection

- Other entries in the *Cold Context Table*
  - Branch $\rightarrow$ incremental insertion in CCT
  - Out-of-band CCT sorting: inexpensive insertion sort in $O(n^2)$
  - If CCT sorting cannot catch up: degenerates into a stack (=GPUs)
Memory access patterns

In traditional vector processing

**Easy**
- Registers
- Memory
- Scalar load & broadcast
- Reduction & scalar store

**Hard**
- Registers
- Memory
- (Non-unit) strided load
- (Non-unit) strided store

**Easy**
- Registers
- Memory
- Unit-strided load
- Unit-strided store

**Hardest**
- Registers
- Memory
- Gather
- Scatter
Memory access patterns

With dynamic vectorization

Easy Registers
Memory

T_1 T_2 T_n

Scalar load & broadcast
Reduction & scalar store

Common case

Easy Registers
Memory

T_1 T_2 T_n

Unit-strided load
Unit-strided store

Hard Registers
Memory

T_1 T_2 T_n

(Non-unit) strided load
(Non-unit) strided store

Hardest Registers
Memory

T_1 T_2 T_n

Gather
Scatter

General case

→ Support the general case, optimize for the common case
Memory access unit

- Scalar and aligned unit-strided scenarios: single pass
- Complex accesses in multiple passes using replay
- Execution of a scatter/gather is interruptible
  - Allowed by multi-thread ISA
  - No need to rollback on TLB miss or exception

![Diagram of memory access unit]

1. Instruction Fetch
2. Broadcast
   - MPC = PC_0?
   - MPC = PC_1?
   - MPC = PC_n?
3. Memory
4. Update PC
5. Vote

No match or bank conflict: discard instruction, do not update PC
FPGA prototype

On Altera Cyclone IV

- Up to 2048 threads per core: 64 warps × 32 threads
- Sweet spot: 8x8 to 32x16

Latency hiding multithreading depth

Throughput SIMD width
Conclusion

- Stateless dynamic vectorization is implementable
- Unexpectedly inexpensive
  - Overhead amortized even for single-issue RISC without FPU
- Scalable
  - Parallelism in same class as state-of-the-art GPUs
- Minimal software impact
  - Standard scalar RISC-V instruction set, no proprietary extension
  - Reuse the RISC-V software infrastructure: gcc and LLVM backends
  - OS changes to manage ~10K threads?
- One step on the road to single-ISA heterogeneous CPU+GPU
Simty:
Generalized SIMT execution on RISC-V

CARRV 2017

Sylvain Collange
INRIA Rennes / IRISA
sylvain.collange@inria.fr