Dynamic Inter-Thread Vectorization Architecture:
Extracting DLP from TLP

SBAC-PAD 2016
Los Angeles

Sajith Kalathingal, Sylvain Collange, Bharath Swamy, André Seznec
INRIA Rennes / IRISA
sylvain.collange@inria.fr
Dynamic inter-thread vectorization

- Idea: hardware aggregates SPMD threads together to assemble vector instructions

  SPMD Program
  
  add r1, r3
  mul r2, r1

  Threads
  
  add
  mul
  add
  mul
  add
  mul
  add
  mul

  Vector instructions

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

- 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: balance vector width (DLP) with multi-threading depth (TLP)
Outline

- Background
  - SMT
  - SIMT

- The DITVA microarchitecture
  - Threads, warps and instruction streams
  - Handling convergence and divergence
  - Register file and memory access units

- Performance evaluation
  - Instruction count
  - Performance
Simultaneous multi-threading (SMT)

- Multi-thread programming model
- **Time/space-multiplexing** of processing units

Hides latency thanks to explicit parallelism

Does not exploit **instruction redundancy** of SPMD programs
GPU execution model: SIMT

- Run SPMD threads in lockstep
- **Mutualize** fetch/decode, load-store units
  - Fetch 1 instruction on behalf of several threads
  - Read 1 memory location and broadcast to several registers

**SIMT**: Single Instruction, Multiple Threads

Improves Area/Power-efficiency by exploiting **instruction redundancy**

Needs software+hardware support to handle **thread divergence**
Goto considered harmful?

Control instructions in some CPU and GPU instruction sets

- Control flow structure is **explicit** on GPUs
  - Incompatible with general-purpose instruction sets!
Dynamic Inter-Thread Vectorization Architecture

The DITVA approach

- Start from an in-order SMT CPU
- Add dynamic inter-thread vectorization
- Run existing parallel programs compiled for general-purpose ISAs

Baseline: 4-thread 4-issue in-order with explicit SIMD

DITVA: 4-warp × 4-thread 4-issue
Outline

- Background
  - SMT
  - SIMT

- The DITVA microarchitecture
  - Threads, warps and instruction streams
  - Handling convergence and divergence
  - Register file and memory access units

- Performance evaluation
  - Instruction count
  - Performance
Logical thread organization

Programming model: SPMD with many threads

- Warps: groups of threads, statically assembled
  - Warps progress independently

- Instruction streams (ISs): threads that share the same PC in a warp
  - Set of threads that belong to IS represented as bit mask
  - IS count varies dynamically from 1 to n ISs per warp
Mapping threads to DITVA microarchitecture

- **Multi-stream front-end**
  - Follows instruction streams

- **SIMD back-end**
  - Each thread is bound to a fixed lane
  - Same instruction on all lanes
  - Per-lane predication for execution and register write-back
DITVA microarchitecture

- Baseline for experiments
- 4-issue in-order SMT
- Minimal additions to SMT
  - **PC comparators** for IS reconvergence detection
  - 2-level fetch steering policy
    - Select warp in round-robin
    - Select IS with lockstep-favoring policy
  - 1 instruction queue per IS
  - **Full SIMD** back-end
  - **Banked L1** data cache
Convergence detection at branch predict time

- Make a single branch prediction for all threads of 1 IS
  - Speculate that all threads of IS follow predicted direction
  - Prediction is “correct” if at least 1 thread takes the predicted direction
- Merge any 2 ISs that reach the same PC
  - Fetch policy makes it likely by steering ISs toward their convergence point
Fetch steering policy

Which IS to fetch from?

- Conditionals, loops
  - Order of code addresses
  - min(PC)
- Functions
  - Favor max nesting depth

<table>
<thead>
<tr>
<th>Source</th>
<th>Assembly</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(...)</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>{</td>
<td>p? br else</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>br endif</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td>else:</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>endif:</td>
<td></td>
</tr>
<tr>
<td>while(...)</td>
<td>start:</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>p? br start</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>f();</td>
<td>call f</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>void f()</td>
<td>f:</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

- Need to preserve forward progress!
- Hybrid with round-robin policy
  - Make sure each thread gets a minimum time share
Detect divergence when resolving branches
- When at least 1 thread takes a direction not predicted, fork a new IS
- Fix masks of younger instructions in pipeline and of original IS

Also handles misprediction
- Misprediction = divergence with no thread on the predicted path
- Abort the original IS, flush instructions with empty mask
Divergence at branch resolve time

- Detect divergence when resolving branches
  - When at least 1 thread takes a direction not predicted, fork a new IS
  - Fix masks of younger instructions in pipeline and of original IS

- Also handles branch misprediction
  - Misprediction = divergence with no thread on the predicted path
  - Abort the original IS, flush instructions with empty mask
Explicit vector instructions

- Direct explicit vector instructions (e.g. SSE, AVX) to SIMD units

- Need to allow both:
  - SIMD across threads for scalar instructions
  - SIMD within thread for vector instructions

- AVX registers swizzled in register file to support both access patterns
Memory access unit

Needs to support concurrent accesses from multiple threads

- Banked L1 cache
  - Based on physical address
  - Common access pattern: threads access the same offset on different pages
  - DITVA uses address hashing to avoid bank conflicts
- Per-lane TLB
  - No need to synchronize TLBs
Outline

- Background
  - SMT
  - SIMT
- The DITVA microarchitecture
  - Threads, warps and instruction streams
  - Handling convergence and divergence
  - Register file and memory access units
- Performance evaluation
  - Instruction count
  - Performance
Front-end pressure reduction

- DITVA reduces front-end utilization by fetching fewer instructions
- Simulation on PARSEC benchmarks:

-31% instructions fetched for 2-wide DITVA
-51% for 4-wide DITVA
**DITVA performance**

- **2-wide / 4-wide DITVA:** +37% and +55% performance over SMT 4 on SPMD workloads
- Some apps need at least 4 warps to hide latency
Conclusion: the missing link

- New range of architecture options between multi-core and GPUs
- Enables heterogeneous platforms with unified instruction set
Dynamic Inter-Thread Vectorization Architecture:
Extracting DLP from TLP

SBAC-PAD 2016
Los Angeles

Sajith Kalathingal, Sylvain Collange, Bharath Swamy, André Seznec
INRIA Rennes / IRISA
sylvain.collange@inria.fr