SAND2024-05691C

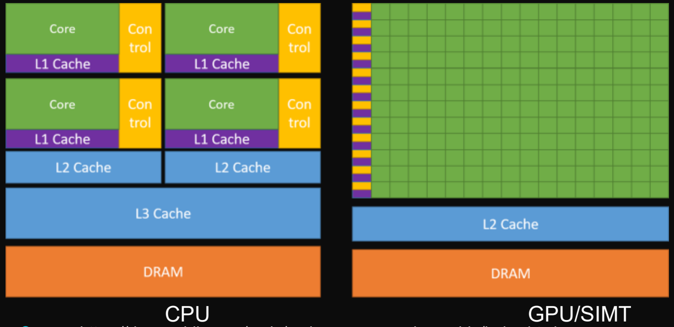
Integrating RISC-V SIMT and Scalar Cores: Loosely to Tightly Coupled

Presenter: Clay Hughes, Sandia National Labs

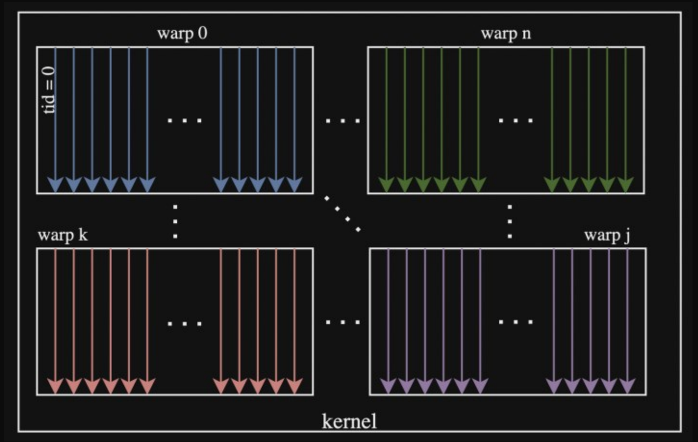
Authors: Sooraj Chetput, Anusuya Nallathambi

Introduction: SIMT Architectures

- Single Instruction Multiple Thread
- Computing model designed to handle highly parallel workloads.
- Highly optimized for overall throughput over thread latency.
- **Lockstep Synchronization:** threads within a warp (a collection of threads) execute same instruction simultaneously, all threads progress together.



Source: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html



Introduction: Vortex GPGPU

- Open-Source RISC-V GPU, written in SystemVerilog
- **Instruction Handling:** Treats RISC-V scalar instructions as standard SIMT instructions.
- **Simplified Code Conversion:** Code requires only recompilation, not a complete rewrite.
- Custom Instructions to Support a SIMT Stack
  - **WSPAWN:** Spawn a new warp
  - **SPLIT/JOIN:** Used to handle the immediate post dominator stack for managing control flow divergence
  - **TMC:** Thead Mask Control to control which threads within a warp are active.
  - **CSRR/CSRW**: **Special Instructions used to write to the control status registers.**

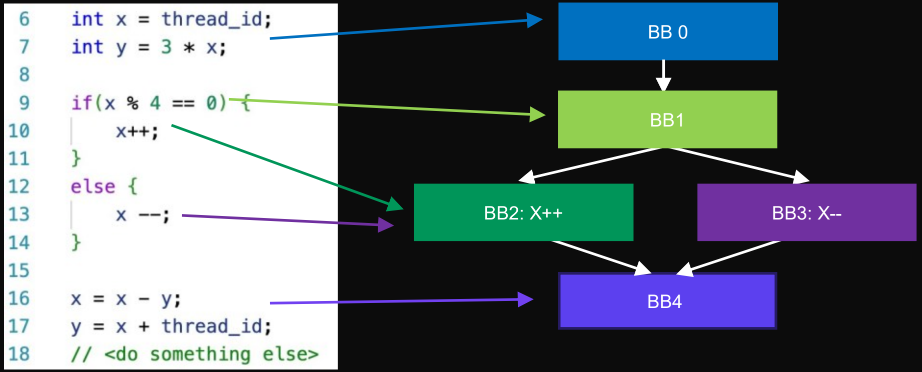
Source:

- **Vortex: An Open Source Reconfigurable RISC-V GPGPU Accelerator for Architecture Research**
  - *Authors: Fares Elsabbagh, Blaise Tine, Apurve Chawda, Will Gulian, Yaotian Feng, Da Eun Shim, Priyadarshini Roshan, Ethan Lyons, Lingjun Zhu, Sung Kyu Lim, Hyesoon Kim*
    Hot Chips 32 (2020)

Problems with SIMT Architectures:
Control Flow Divergence

- Occurs when threads within a warp have different execution paths through the code
  - Happens if there are if statements or loops in the instruction streams
- Breaks the Lockstep Synchronization
- This is a problem for the SIMT model because the execution of threads within a warp becomes serialized

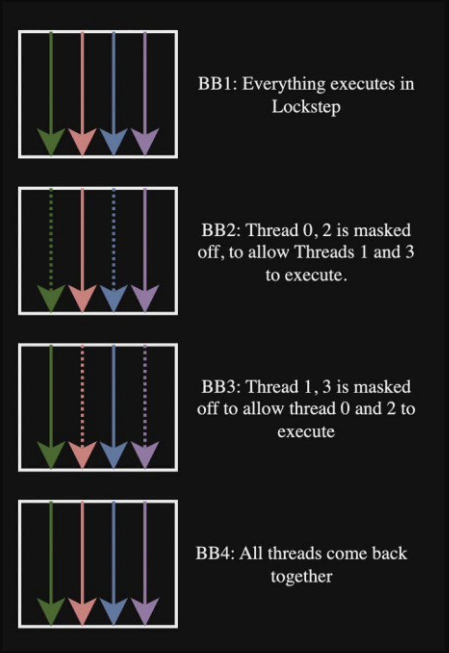
Existing Solution: Immediate Post Dominators



```
int x = thread_id;
int y = 3 * x;

if(x % 4 == 0) {
    x++;
}
else {
    x --;
}

x = x - y;
y = x + thread_id;
// <do something else>
```



BB1: Everything executes in Lockstep

BB2: Thread 0, 2 is masked off, to allow Threads 1 and 3 to execute.

BB3: Thread 1, 3 is masked off to allow thread 0 and 2 to execute

BB4: All threads come back together

- The above figure demonstrates how code translates to basic blocks (BB).
- We can say that BB4 immediately post dominates BB1
- Let's see what happens in the hardware if thread 0 and 2 decide to go through BB0 and threads 1 and 3 decide to go through BB3

Problem Statement

- **Control Flow Divergence:**
  - Leads to serialization of execution paths, increasing total execution time
  - **Post Dominators:** aligning execution by merging paths at a control point. This is implemented in the hardware using a stack.

Previously Proposed Solutions

- **Dynamic Warp Reshuffling:**
  - Proposed solution for maximizing SIMT lane utilization
  - Re-shuffling warps and rearranging threads based on Control Flow Divergence Patterns
  - Not used due to hardware overhead
- **2-way/N-way Lane Splitting:**
  - Suggests allowing parallel execution of divergent paths
  - Not used due to hardware overhead.

The idea behind our Solution

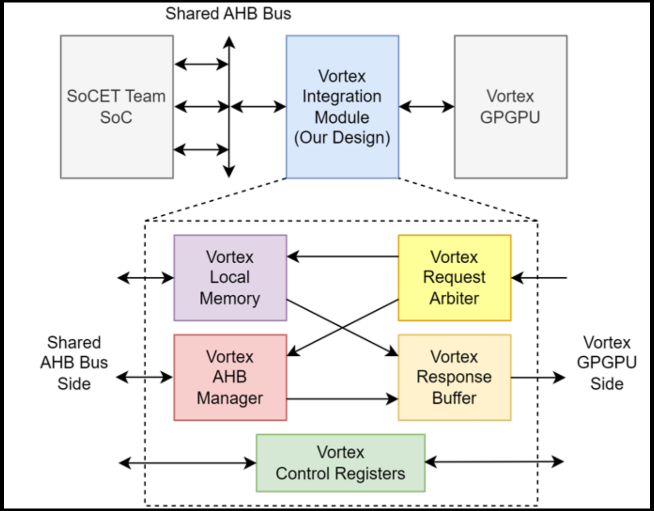
- We have 2 cores: one SIMT, one Scalar.
- Both cores run the same extended RISC-V ISA,
    - Thread transfer shouldn’t require any recompilation or translation

If we can send the divergent threads to the Scalar Core, and the less divergent ones to the SIMT Core, we can effectively reduce serialization on the SIMT Core.

Roadmap

The Vortex Tapeout

- **Integration Overview**
  - Integration follows a host-device CPU-GPU model on the AHB bus.
  - Enhances a RISC-V based microcontroller with parallel processing capabilities.
- **Software Interface**
  - Three control/status registers: Start, Start Address, and Status Register.
  - CPU independently handles tasks while GPU is active, enhancing multitasking.
-



The Vortex Tapeout Contd.

- **Results and Physical Design Insights**
  - We used flip-flops for memory due to PDK constraints.
  - Memory consumption is significant, impacting overall area efficiency.
  - The table below shows 4 configurations that we considered.
-

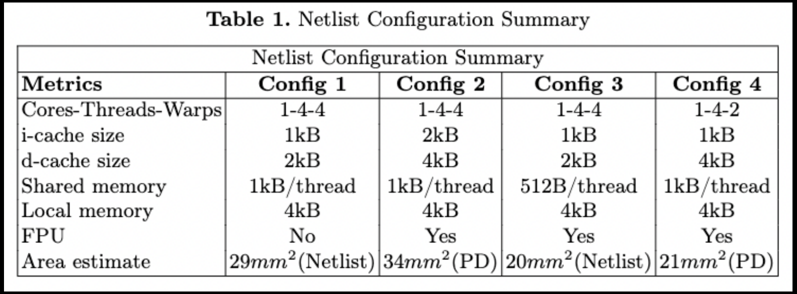
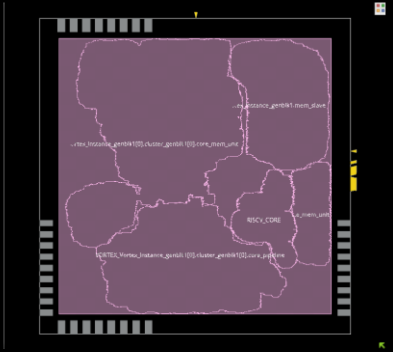
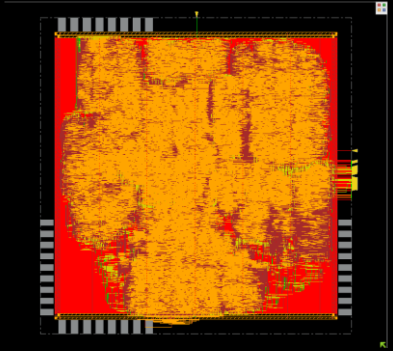


Table 1. Netlist Configuration Summary

Netlist Configuration Summary				
Metrics	**Config 1**	**Config 2**	**Config 3**	**Config 4**
Cores-Threads-Warps	1-4-4	1-4-4	1-4-4	1-4-2
i-cache size	1kB	2kB	1kB	1kB
d-cache size	2kB	4kB	2kB	4kB
Shared memory	1kB/thread	1kB/thread	512B/thread	1kB/thread
Local memory	4kB	4kB	4kB	4kB
FPU	No	Yes	Yes	Yes
Area estimate	$29mm^2$(Netlist)	$34mm^2$(PD)	$20mm^2$(Netlist)	$21mm^2$(PD)


...ex_instance_genblk1.mem_slave
...rtex_Instance_genblk1[0].cluster_genblk1[0].core_mem_unit
RISCV_CORE
...e_mem_unit
VORTEX_Vortex_Instance_genblk1[0].cluster_genblk1[0].core_pipeline

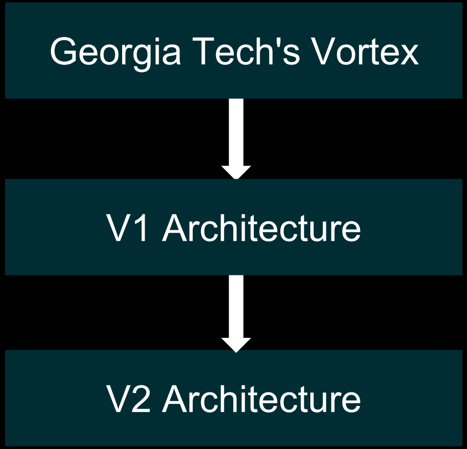
SoC+Vortex Amoeba View



SoC+Vortex Physical View

The SIMT-Scalar *V1* and *V2* Architectures for mitigating Control Flow Divergence

- **V1 architecture:** a simple in-order Scalar core created by parameterizing Vortex's SIMT core
- **V2 architecture:** Scalar core with logic to transfer threads from the SIMT core during runtime + simple not-taken branch predictor to reduce stalling of threads

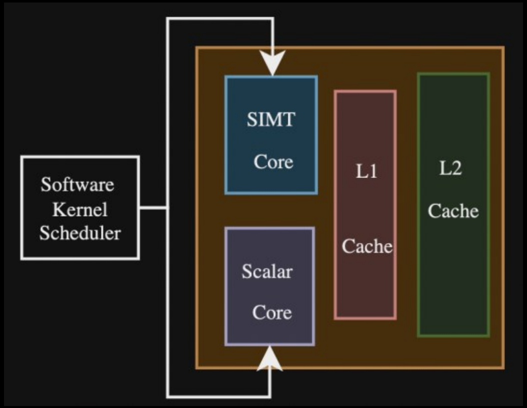


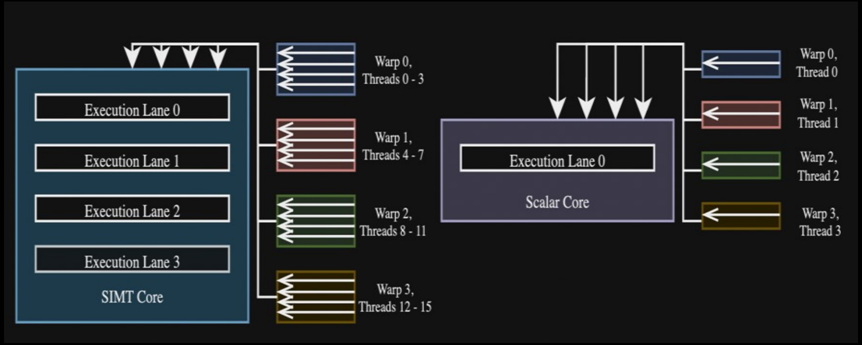
SIMT-Scalar V1 Architecture

- Kernel scheduler assigns threads separately to the cores according to priority.
- Priority is arbitrarily assigned by software.

- Low priority -> SIMT core (4 threads/warp) (4 warps)
- High priority -> scalar core (4 hyperthreaded threads)

- No Datapath hardware is shared between the SIMT and the Scalar core.
- L1 and L2 caches are shared
- Modified the Vortex GPGPU to implement this architecture
  - Scalar core is a parametrized single-lane version of the SIMT core





V1 Architecture – Simulations, Results

- Tested with simple vector add type kernels
- We ran a total of 20 threads on The V1 Arch and the baseline vortex.
- 4 different degrees of divergence
  - No divergence:
    - All threads Add
  - Min Divergence:
    - odd threads Add
    - even threads Subtract
  - Mid Divergence:
    - Odd threads: 4th bit == 1? Add else Subtract
    - Even threads: Subtract
  - High Divergence
    - Complete serialization of all the threads
    - (each thread takes its own path)

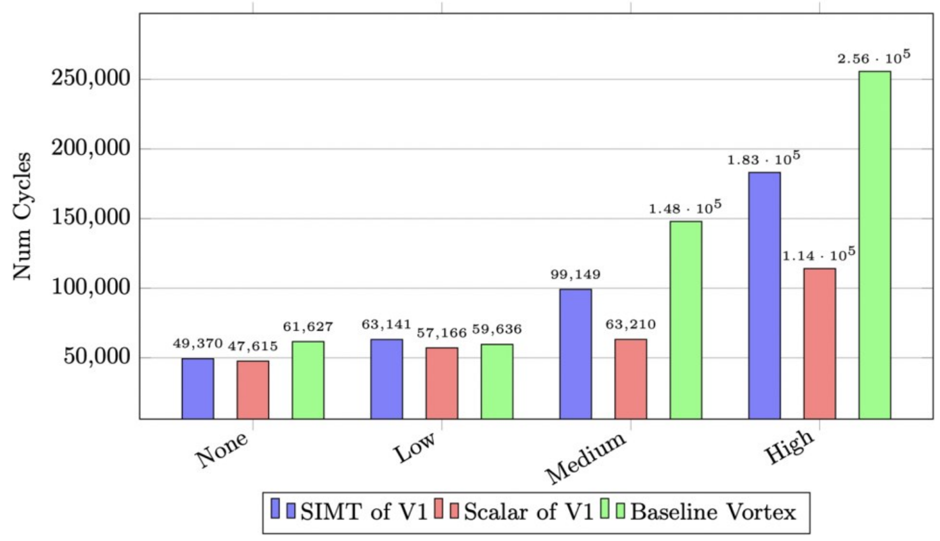


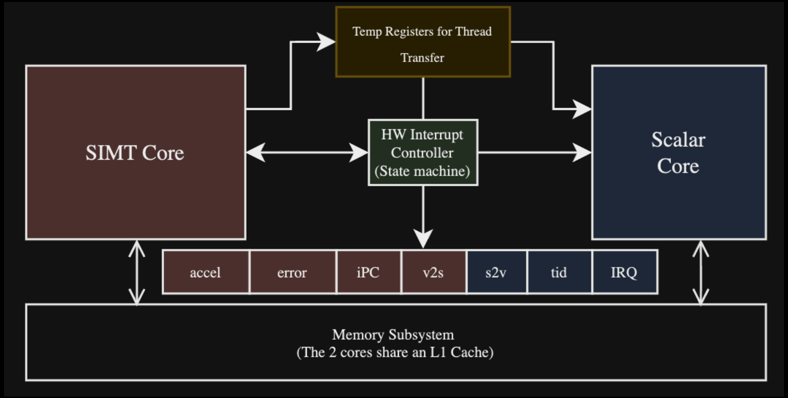
Fig. 10. Num Cycles Comparison for SIMT and Scalar Cores of V1 Architecture Across Divergence Levels vs. the Baseline Vortex.

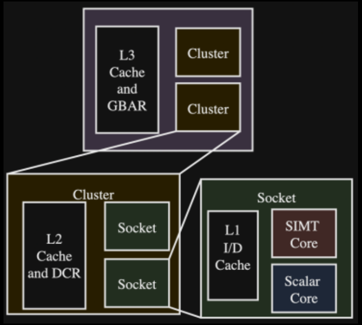
Discussion: V1 Architecture

- **Pros**
  - Simple design, ideal for proof-of-concept.
- **Cons**
  - We don’t know which threads are more divergent and thus should be marked as high priority before run-time. This process is implemented arbitrarily by the software currently.
  - The Scalar core is not optimized for branches (e.g. it stalls every time it runs into a branch).

SIMT-Scalar V2 Architecture

- **SIMT Core:** **4 Warps, 4 threads/warp**
- **Scalar Core:** **1 Thread, 1 Warp**
- **Coupling**
  - At the hardware interrupt controller level (shared control status registers)
- **Thread Priority Assignment**
  - Software still assigns a relative priority index to each thread (does not classify it as high priority or low priority like V1 did).
    - The Scalar core kernel scheduler uses this information to decide what thread to pull from the SIMT core.
- **Dynamic Switching:**
  - Introducing hardware interrupt controller to facilitate dynamic thread transfer from the SIMT to the scalar upon the scalar core's request
- **Memory Subsystem**
  - Shared L1 Instruction/Data cache for scalar and SIMT cores, Unified L2 cache
- Shared "status registers" inside of interrupt controller to allow communication between SIMT and scalar cores
  - Thread context is dumped into dedicated control status registers


Temp Registers for Thread Transfer
SIMT Core
HW Interrupt Controller (State machine)
Scalar Core
accel
error
iPC
v2s
s2v
tid
IRQ
Memory Subsystem
(The 2 cores share an L1 Cache)



V2 Arch - Discussion

- This still does not solve the problem of how to select the thread that is to be transferred in an intelligent way.
- All we did here is build a mechanism to transfer the thread from the SIMT to the Scalar core.
- Consider a case when none of the threads divergence
  - It would be best to execute the entire thing on the SIMT
- Consider when all the threads are completely serialized
  - It would be best to just execute the entire thing on the Scalar core.
- The development of the RTL is still under progress, and so is benchmarking.

Future Work

- Complete V2 Architecture and Benchmarking
- V3 Architecture: We can use the IPDOM analysis, and branch history to predict branch divergence. The threads that will cause the most serialization will be sent to the scalar core.
- V4 Architecture: We want to consider a cluster of M SIMT and N Scalar cores. This could be better for handling more complex divergence patterns.

Purdue SoCET Team

- The goal of Purdue SoCET (system-on-chip extension technologies) is to provide students hands on experience with a fully developed industry quality SoC design flow. Members of the group engage with RTL design, physical design, chip bringup, verification methods, an array of EDA tools and software development.
- Checkout the cool work the team does at the following link:
    - <insert QR code for this URL: https://engineering.purdue.edu/SoC-Team#about>
-