Exploring Memory-Safety Techniques on a SIMT-style RISC-V GPGPU

Matthew Naylor
University of Cambridge
23 April 2024

CAPcelerate Project (DSbD)

Is memory safety an issue for GPGPUs?

```
__device__ void overread () {
    int data = 0xda1a;
    int secret = 0xc0de;
    int* ptr = &data;
    printf("%x\n", ptr[1]);
}
```

A study from AMD Research looked at 175 applications from standard benchmark suites and found 13 kernels with buffer overflows* **[Erb *et al.* 2017]**

*Incomplete: uses canaries, on global buffers only

Memory safety bugs have been found in several GPU applications and benchmark suites using Oclgrind* **[Price *et al.* 2015]**

* Slow: an interpreter for SPIR (LLVM IR variant)
* Finds data races, sync issues, as well as mem safety issues

Tool	Completeness	Overhead
Oclgrind	Good	Poor
cuda-memcheck	Good	Poor
clArmour	Poor	Good
GMOD	Poor	Good

Are these types of bugs exploitable?

Buffer overflows on GPUs can lead to:

- data corruption on the stack and heap
- control-flow hijacking
- code injection
- arbitrary code execution

[Di *et al.* 2016, Miele 2016, Park *et al.* 2021]

CHERI on GPGPUs?

CHERI on GPGPUs: Challenges

Problem 1: Adding CHERI to an existing GPU instruction set & toolchain would be a major effort (at least for a small research project).

Problem 2: Massively threaded GPUs depend on a small amount of architectural state per thread, which CHERI would potentially double.

Recent development: RISC-V GPUs

Simty: generalized SIMT execution on RISC-V

Caroline Collange
Inria
caroline.collange@inria.fr

ABSTRACT

We present Simty, a massively multi-threaded RISC-V processor core that acts as a proof of concept for dynamic inter-thread vectorization at the micro-architecture level. Simty runs groups of scalar threads executing SPMD code in lockstep, and assembles SIMD instructions dynamically across threads. Unlike existing SIMD or SIMT processors like GPUs or vector processors, Simty vectorizes scalar general-purpose binaries. It does not involve any instruction set extension or compiler change. Simty is described in synthesizable RTL. A FPGA prototype validates its scaling up to 2048 threads per core with 32-wide SIMD units. Simty provides an open platform for research on GPU micro-architecture, on hybrid CPU-GPU micro-architecture, or on heterogeneous platforms with throughput-optimized cores.

and programming languages to target GPU-like SIMT cores. Beside simplifying software layers, the unification of CPU and GPU instruction sets eases prototyping and debugging of parallel programs. We have implemented Simty in synthesizable VHDL and synthesized it for an FPGA target.

We present our approach to generalizing the SIMT model in Section 2, then describe Simty's microarchitecture is Section 3, and study an FPGA synthesis case Section 4.

2 CONTEXT

We introduce some background about the generalized SIMT model and clarify the terminology used in this paper.

2.1 General-purpose SIMT

"It does not involve any instruction set extension or compiler change" **[Collange 2017]**

What is SIMT?

Execute multiple hardware threads (a **warp**) in lockstep (where possible) exploiting regularity between them, e.g.

- control-flow regularity
- memory-access regularity
- value regularity

Seminal work on value regularity

Dynamic detection of uniform and affine vectors in GPGPU computations

Caroline Collange[1], David Defour[1] and Yao Zhang[2]

Seminal study **[Collange 2009]** reports:

- 27% of register reads yield the same value for each thread in a warp (**uniform vectors**)
- 44% yield values separated by a constant stride (**affine vectors**)

Where does value regularity come from?

In data-parallel programming, each thread typically:

- uses its **thread index** within a block
- and its **block index** within a grid

to determine which part of the input to read and which part of the output to write. For threads in a warp:

- the **thread index** is affine
- and the **block index** is uniform
- and uniform/affine vectors often propagate

CHERI on GPGPUs

Idea 1: Completely reuse existing CHERI ISA and toolchain in a SIMT-style RISC-V GPGPU.

- RISC-V also an existing target for Rust
- RISC-V also likely to support MTE at some point

We can reuse various existing CPU solutions to memory safety on GPGPUs

CHERI on GPGPUs

Idea 2: Exploit value regularity to reduce register storage cost of CHERI.

```
for (i = threadIdx.x; i < len; i += blockDim.x)
  sum += input[i];
```

- After compilation, each thread in the block may hold a different pointer into the `input` array
- But this pointer will likely have the same bounds, permissions, etc. in each thread

Which RISC-V GPGPU to use?

Should we build on top of **Simty** [Collange 2017] or **Vortex** [Tine *et al.* 2021]?

- Simty was not reporting benchmark results
- Vortex was reporting poor benchmark results
- Neither were supporting shared local memory
- Neither were exploiting value regularity

We decided to develop our own: **SIMTight**.

What is SIMTight?

- Single RV32IMA**xCHERI** streaming multiprocessor
- 64 warps and 32 threads per warp (2048 threads)
- Fully **synthesisable** (high perf. density on FPGA)
- Ships with CUDA-like library and 14 benchmarks (**C++** and **Rust** versions of both)
- C++ benchmarks run **purecap** without modification

SIMTight exploits value regularity

SIMTight detects uniform and affine vectors in hardware and exploits them:

- **Register file and cache compression** (reducing on-chip storage)
- **Parallel affine and vector pipelines** (improving throughput)

- Entirely microarchitectural (no ISA or compiler changes)

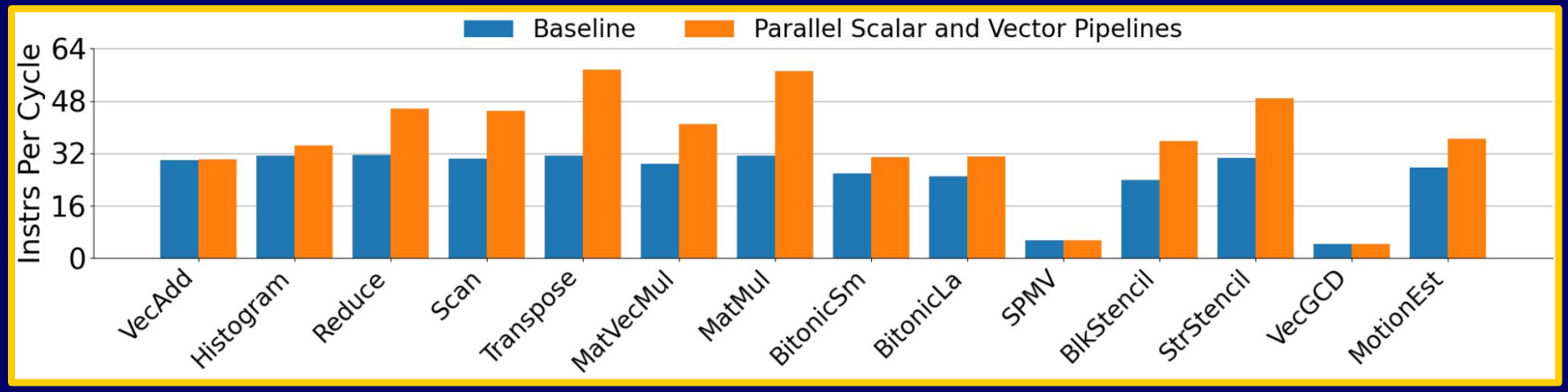
Results

Register file storage requirements

VRF Size (Vector Registers)	Total Storage (Kilobits)	Compression Ratio	Cycle Overhead	Main Memory Access Overhead
1024	1202	1 : 0.57	1.0%	0.0%
512	672	1 : 0.32	1.1%	1.3%
256	407	1 : 0.19	9.7%	47.9%

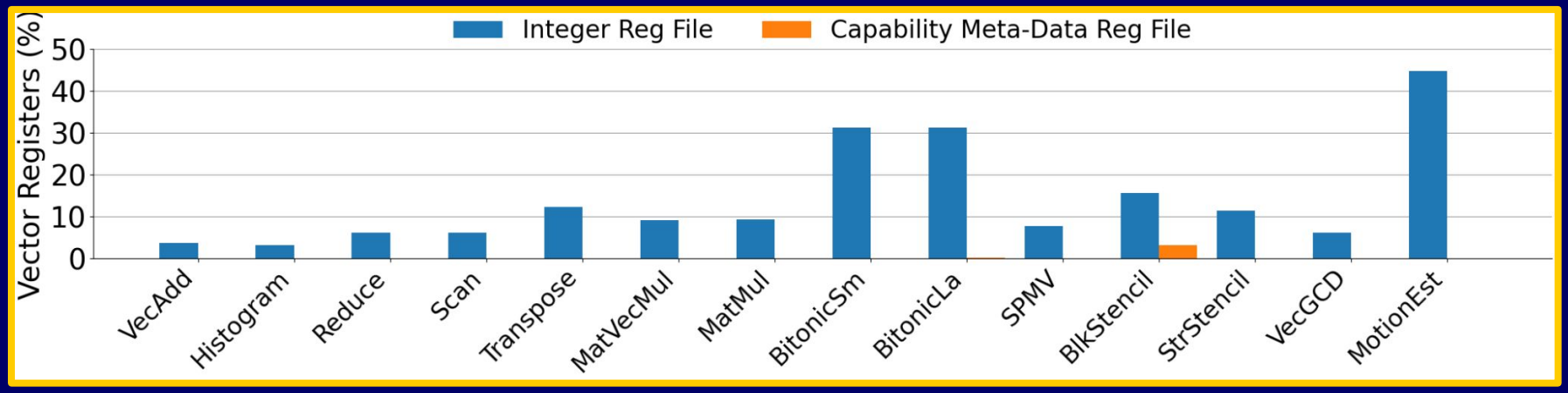
Reduces integer register file storage by 68%
(178KB per SM) for a geomean 1% cycle overhead

Instruction throughput



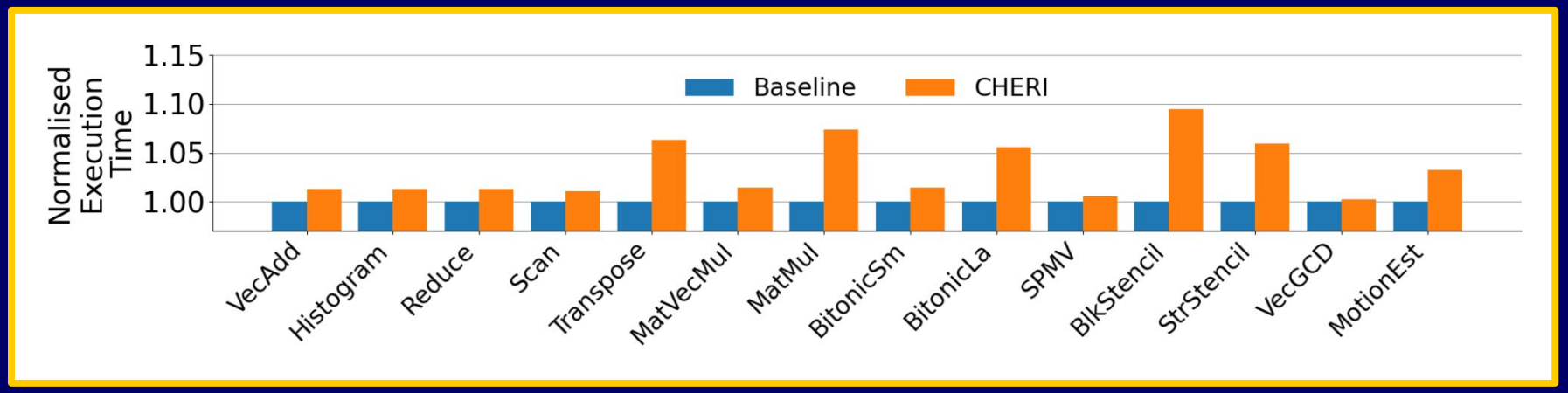
- Baseline IPC often approaches warp size (32)
- Parallel pipelines reduce runtime by 20% geomean at low hardware cost

Register file compression with CHERI



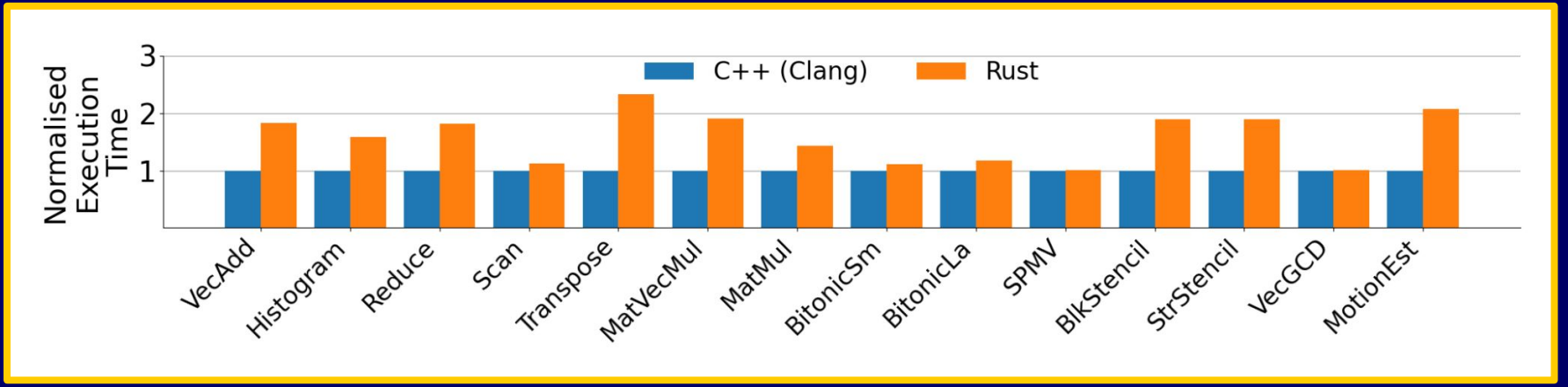
- Rarely need vectors of capability meta-data
- Compressing cap metadata reduces storage requirement of CHERI by 90%: 26KB per SM rather 270KB
- Storage overhead of CHERI is 31%

CHERI runtime and area overheads



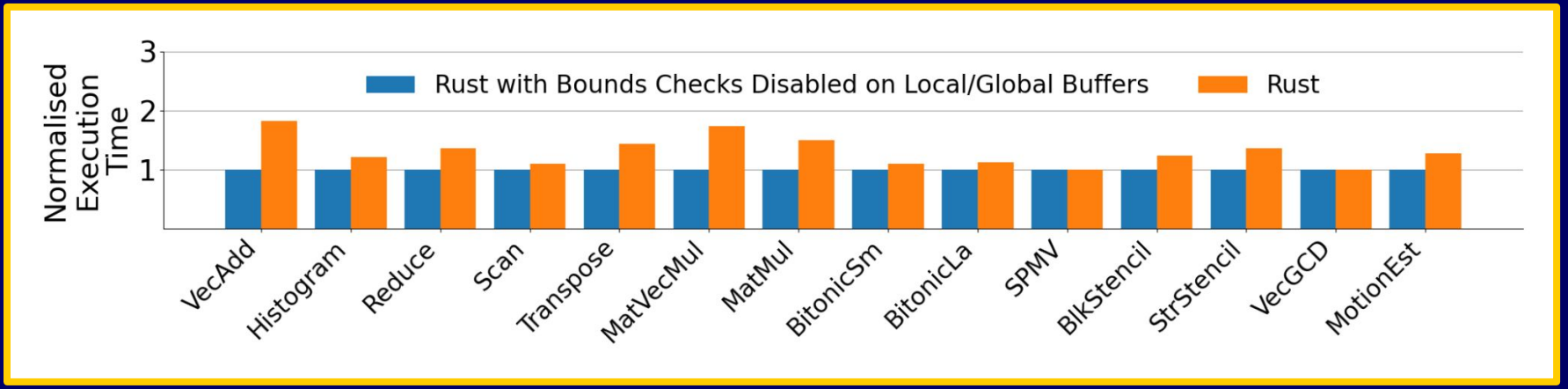
- Geomean 3% cycle overhead
- Geomean 5% wall-clock overhead
- Area cost ~ 1 pipelined divider per vector lane

Rust overheads



- Both compilers based on LLVM 18.1.3
- Geomean 52% cycle overhead
- (No area overhead, of course)

Rust bounds checking costs



Rust bounds checks on **local and global buffers** introduce a geomean 29% runtime overhead

Future Work

Logic area overhead of CHERI

This has been a challenge in SIMTight:

- Hard to avoid a bounds check per lane
- Not just one bounds check per lane, but two
- Bounds must be decompressed
- Sharing decompression cost not easy: uniform compressed bounds does not imply uniform decompressed bounds

MTE (Memory Tagging Extension)

- On CPUs, drawbacks of MTE occur due to large numbers of small buffers, leading to **colour reuse** and **fine granules** (4 tag bits per 16 bytes)
- But GPGPU code typically uses a small number of large buffers
- MTE's simple bounds check should have very low area cost per vector lane

Expressiveness

- Rust and MTE support memory safety within a compute kernel
- But other mechanisms are needed to isolate untrusted code, e.g.
  - multitasking of mutually distrusting compute kernels
  - calling untrusted code in a third-party library
- CHERI can do both, but so far unexplored

Closing remarks

With a RISC-V GPGPU, we can easily reuse various CPU memory-safety solutions.

On GPGPUs:

- **Runtime** overhead of CHERI is low, much lower than Rust
- **Storage** overhead of CHERI is much lower than expected
- **Logic area** overhead of CHERI is notable

CAPcelerate (EP/V000381/1)

Funded by the Digital Security by Design (DSbD) Programme delivered by UKRI to support the DSbD ecosystem.

References

[Erb et al. 2017] Dynamic buffer overflow detection for GPGPUs
[Price et al. 2015] Oclgrind: an extensible OpenCL device simulator
[Di et al. 2016] A Study of Overflow Vulnerabilities on GPUs
[Miele 2016] Buffer overflow vulnerabilities in CUDA: a preliminary analysis
[Park et al. 2021] Mind control attack: Undermining deep learning with GPU memory exploitation
[Collange 2017] Simty: generalized SIMT execution on RISC-V
[Collange 2009] Dynamic Detection of Uniform and Affine Vectors in GPGPU Computations
[Tine et al. 2021] Vortex: Extending the RISC-V ISA for GPGPU and 3D-Graphics