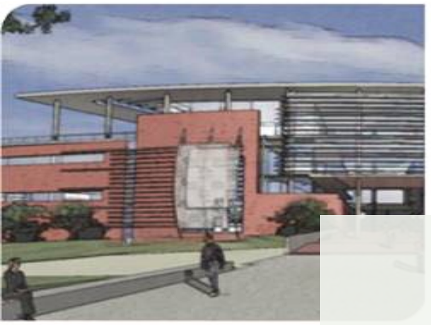
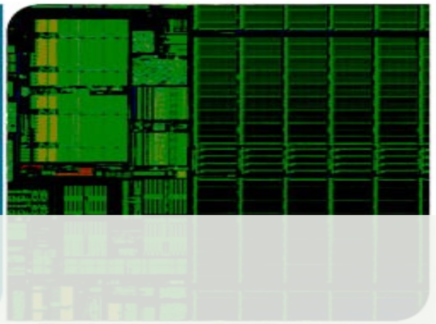
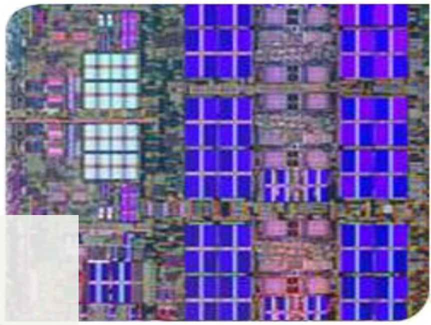
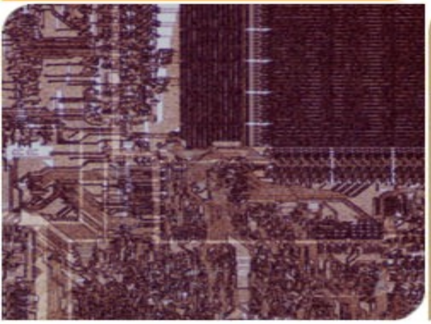




AOS-RISC-V: Towards Always-On Heap Memory Safety

Yonghae Kim, Anurag Kar, Siddant Singh, Ammar A. Ratnani (Georgia Institute of Technology), Jaekyu Lee (Arm Research), Hyesoon Kim (Georgia Institute of Technology)



Memory Safety is still Problematic!

- In history, memory safety vulnerabilities have been prevalent
  - E.g., buffer overflow, use-after-free
- Still account for almost 70% of vulnerabilities
  - According to security reports from Google[1] and Microsoft[2]

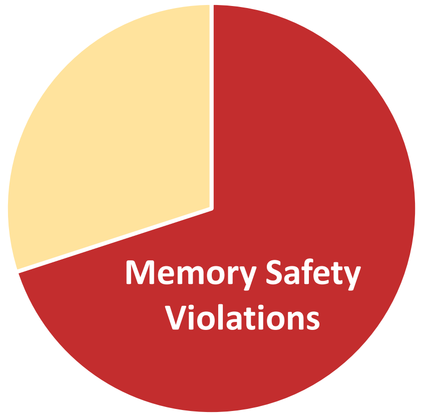
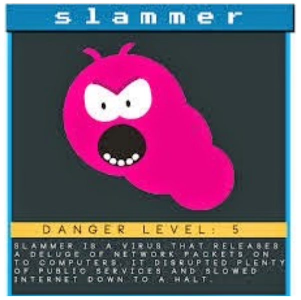


Fig. Root Cause of CVEs in 2006 – 2018.

Morris worm[3] (1988)


slammer
DANGER LEVEL: 5
SLAMMER IS A VIRUS THAT RELEASES A DELUGE OF NETWORK PACKETS ON TO COMPUTERS. IT DISRUPTED PLENTY OF PUBLIC SERVICES AND SLOWED INTERNET DOWN TO A HALT.

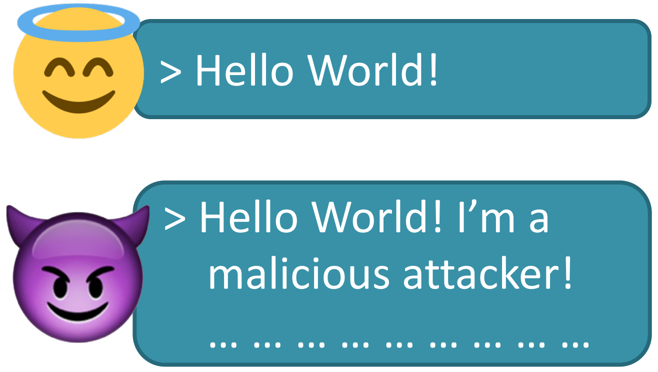
SQL Slammer[4] (2003)

Heartbleed[5] (2014)

[1] Google. 2017. Google Queue Hardening. https://security.googleblog.com/2019/ 05/queue-hardening-enhancements.html.
[2] Matt Miller. 2019. Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape. USENIX Association.
[3] https://newsroom.intel.com/editorials/lessons-from-the-first-computer-virus-the-morris-worm/
[4] https://en.wikipedia.org/wiki/Heartbleed
[5] https://virusinformaticosdericardoyjuliana.blogspot.com/2019/03/virus-sql-slammerzafiro.html

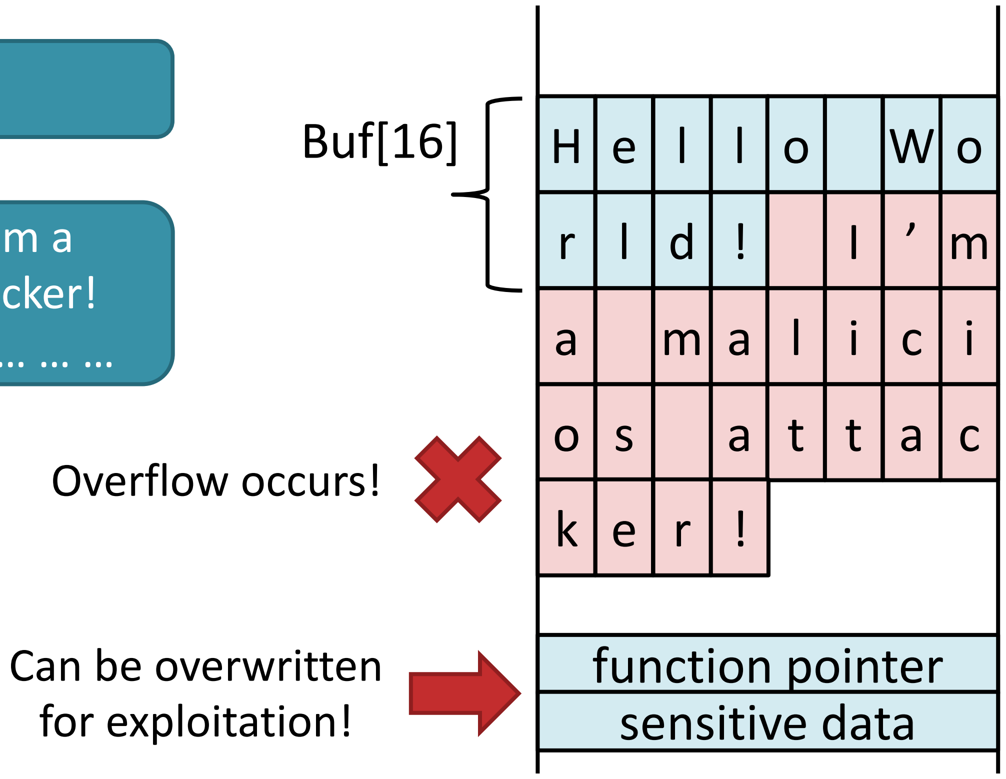
Buffer Overflow Example

```
int main(void) {
    char *buf = (char *) malloc(16);
    scanf("%s", buf);
    printf("buf: %s\n", buf);
    …
}
```



> Hello World!

> Hello World! I'm a malicious attacker! … … … … … … … … …



Overview

- AOS-RISC-V: a RISC-V-based framework for memory safety
- Revisit the prior work, AOS[1]
  - Evaluated using the system emulation (SE) mode in gem5
- Build a full-system level framework in the ecosystem of RISC-V
  - HW extensions based on the BOOM core
  - Compiler support in LLVM
  - OS support in Linux OS
- Conduct performance evaluation

[1] Yonghae Kim, Jaekyu Lee, and Hyesoon Kim, Hardware-based Always-on Heap Memory Safety, MICRO 2020

Data-Pointer Signing

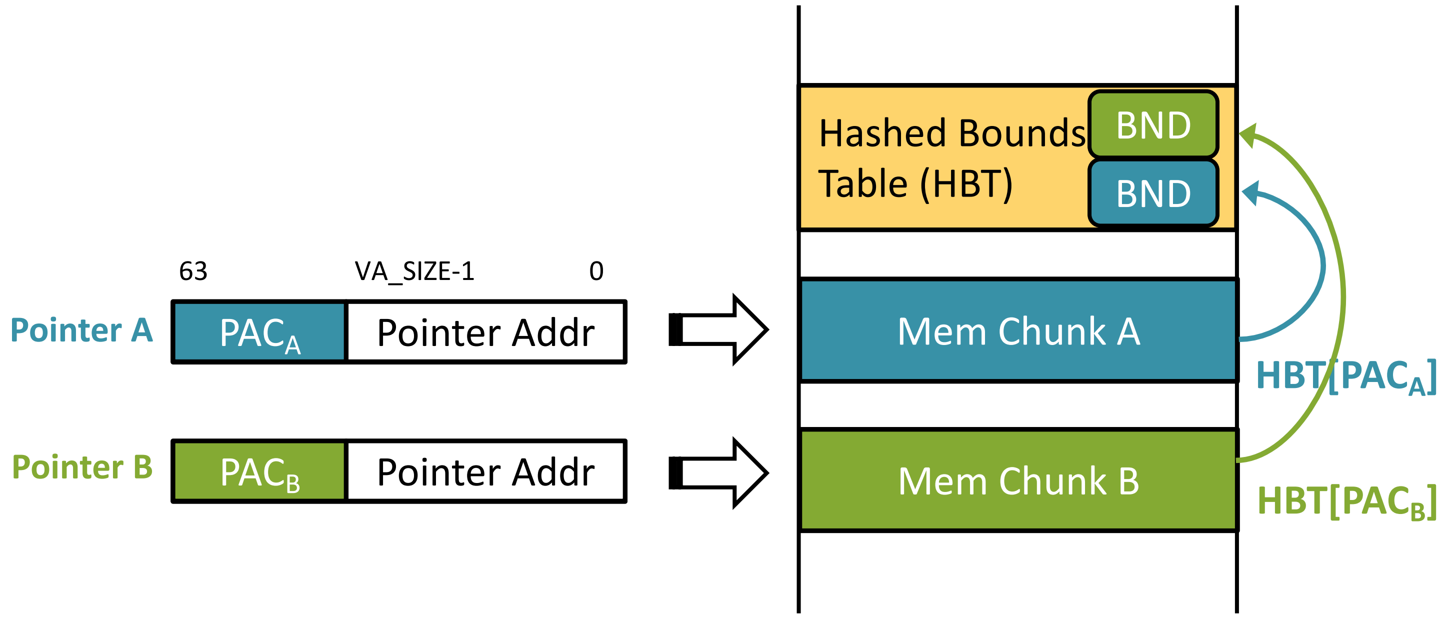
- Tags data pointers with key
  - Key: pointer authentication code (PAC) used in Arm PA
- Stores bounds metadata associated with the key

Mem Allocation

```
ptr = malloc(size);
pacma ptr, ptr, sp;      // Tag ptr
bndstr ptr, ptr, size;   // Store bounds
```

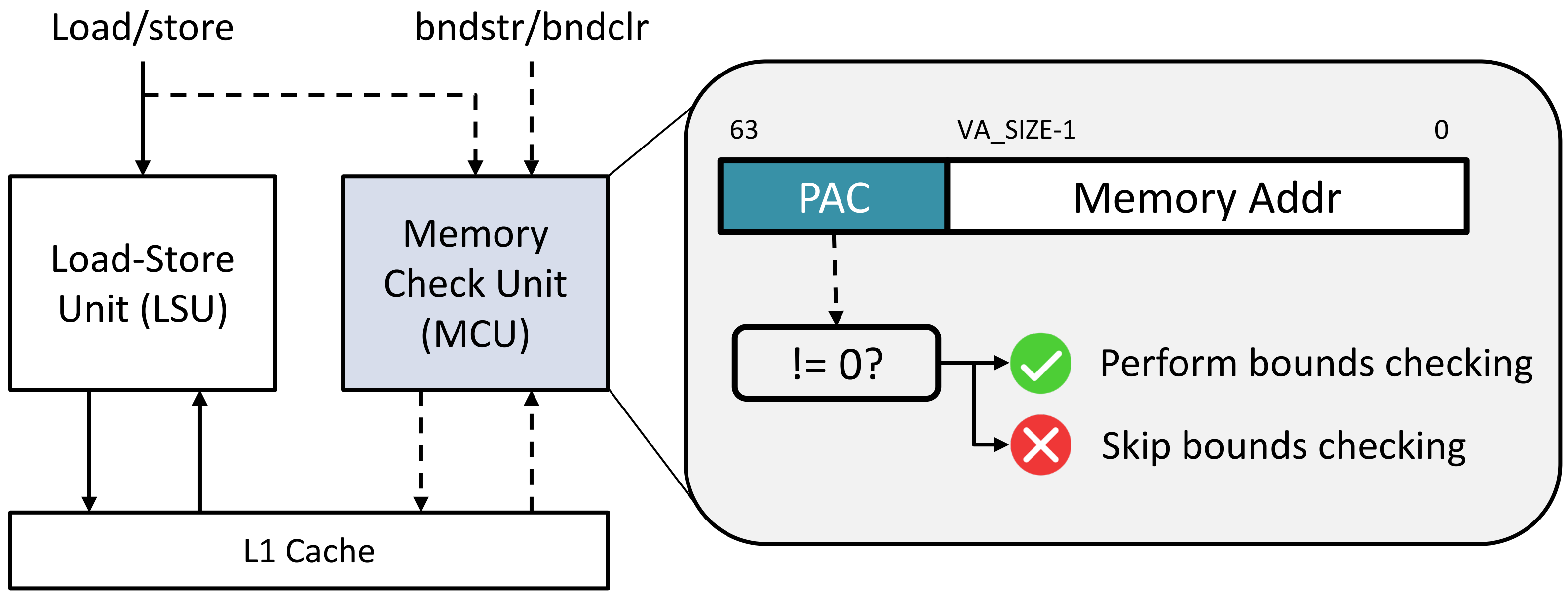
Mem Free

```
bndclr ptr, ptr;         // Clear bounds
xpacm ptr, ptr;          // Strip ptr
free(ptr);
```



Memory Check Unit (MCU)

- For load / store, performs selective bounds checking
- For bndstr / bndclr, stores bounds associated with a pointer



Compiler Support in LLVM

- Optimizer pass
  - Detect dynamic memory de-/allocations
  - Insert new intrinsic functions at LLVM IR level
- Backend pass
  - Replace intrinsic functions with new instructions

	Code Examples
C code	char *ptr = (char *) malloc(10);
LLVM IR code (frontend)	%3 = call noalias i8* @malloc(i64 10) #3 %4 = call i8* @llvm.aos.pacma.p0i8(i8* %3, i64 0) %5 = call i8* @llvm.aos.bndstr.p0i8(i8* %4, i64 10)
Assembly code (backend)	call malloc@plt pacma a0, a0, a1 bndstr a0, a0, a1

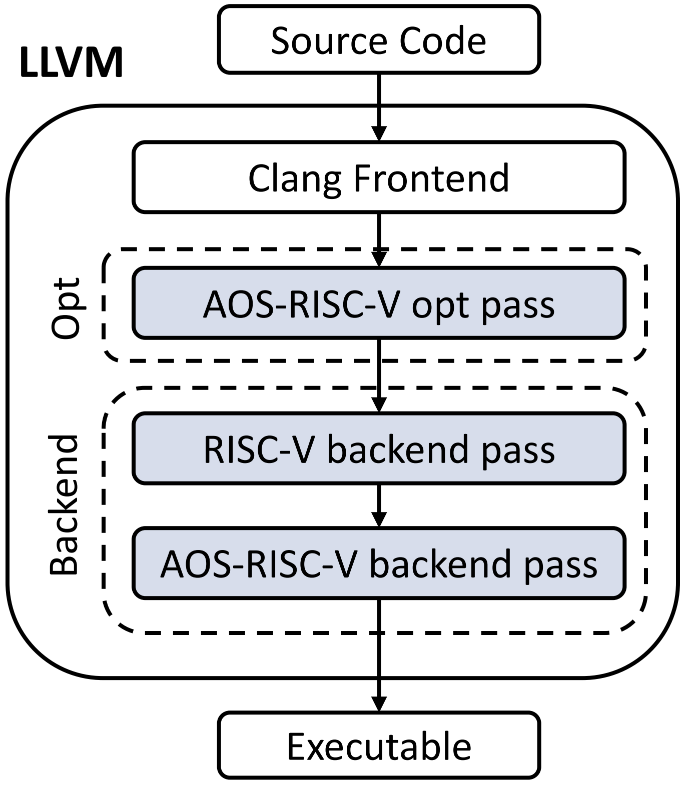


Fig. Modifications on LLVM in AOS-RISC-V

OS Support

- New control and status registers (CSRs)
  - Used to configure HW and obtain runtime statistics
  - Set and read by the kernel
- Process management
  - New fields added to the process structure in the Linux kernel
    - To hold configuration information

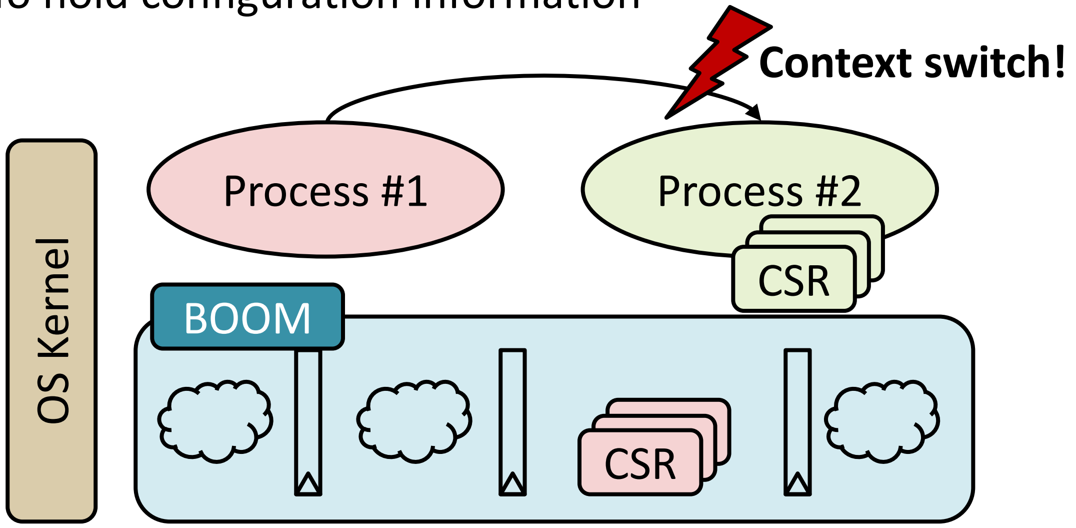


Table. New control and status registers in AOS-RISC-V

CSR Name	Description
enableAOS	Switch to enable AOS-RISC-V
baseAddrOfHBT	Base address of an HBT
numWaysOfHBT	Number of ways of an HBT
numBndstrFails	Number of bounds-store failures
numBndclrFails	Number of bounds-clear failures
numBndchkFails	Number of bounds-check failures

Methodology

- Prototyped on top of the RISC-V BOOM core
- Using the Firesim (v1.13.6), launched Amazon EC2 F1 instances
  - FireSim: An open-source FPGA simulation platform
  - Each FPGA instance runs Linux kernel v.5.7-rc3

- In LLVM 9.0.1, compiled SPEC CPU2006 benchmarks
- Ran a subset of benchmarks using test inputs

Performance Evaluation

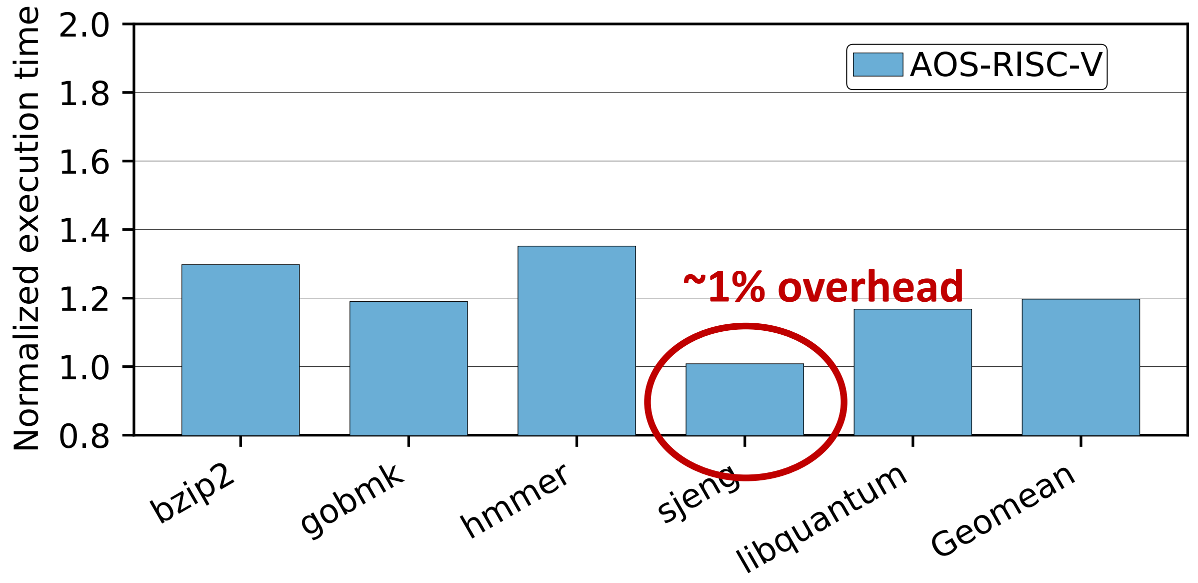


Fig. Execution time of AOS-RISC-V across SPEC 2006 workloads, normalized to the baseline

- 20% slowdown on average
- Main causes of performance overhead:
  - Increased cache port contentions due to additional memory accesses
  - Cache pollution due to extra bounds metadata

Runtime Statistics

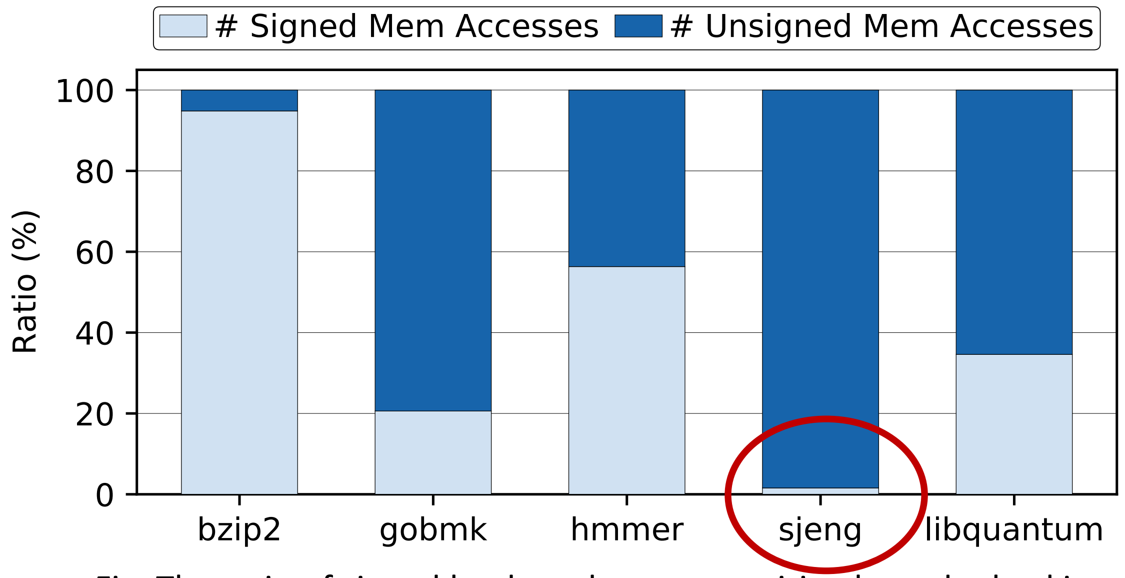


Fig. The ratio of signed loads and stores requiring bounds checking over the total memory accesses

Table. Number of additional signing and bounds instructions executed

Name	pacma	xpacm	bndstr	bndclr
bzip2	28	24	24	24
gobmk	4181	4172	4181	4172
hmmer	90138	90138	90138	90138
sjeng	4	0	4	0
libquantum	95	95	95	95

- In sjeng, only 1% of memory accesses required bounds checking
- bzip2 and hmmer exhibited high ratios of 95% and 56%, respectively
- hmmer was the most malloc-intensive application

Discussion and Future Work

- Comparison to AOS

	AOS w/ Bounds $	AOS w/o Bounds $	AOS-RISC-V w/o Bounds $
Performance Overhead	8.4%	21%	20%
Data fetch width	64 B	64 B	8 B
Input Set (SPEC 2006)	Reference	Reference	Test

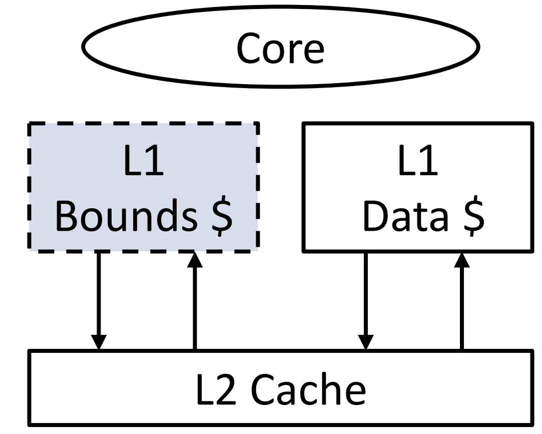


Fig. AOS architecture with a bounds cache

- Future work:
  - Dynamic bounds-table resizing
  - Exception handling for bounds-operation failure
  - Enhancing security guarantees

Conclusion

- We presented our prototype, AOS-RISC-V
  - A full-system level framework for heap memory safety
  - Protect against the most prevalent vulnerabilities
- In the RISC-V ecosystem, we implemented:
  - HW extensions based on the BOOM core
  - Compiler support in LLVM
  - OS support in Linux OS
- We conducted performance evaluation
  - Under Linux running on Amazon EC2 F1 instances
- To contribute to the RISC-V community. we open-source our framework:
  - https://github.com/yonghaekim/AOS-RISC-V

Thank you!

All questions are welcome.