2024 IEEE 12th International Conference on Computer Science and Network Technology
ICCSNT 2024
Dalian, China October 19-20, 2024

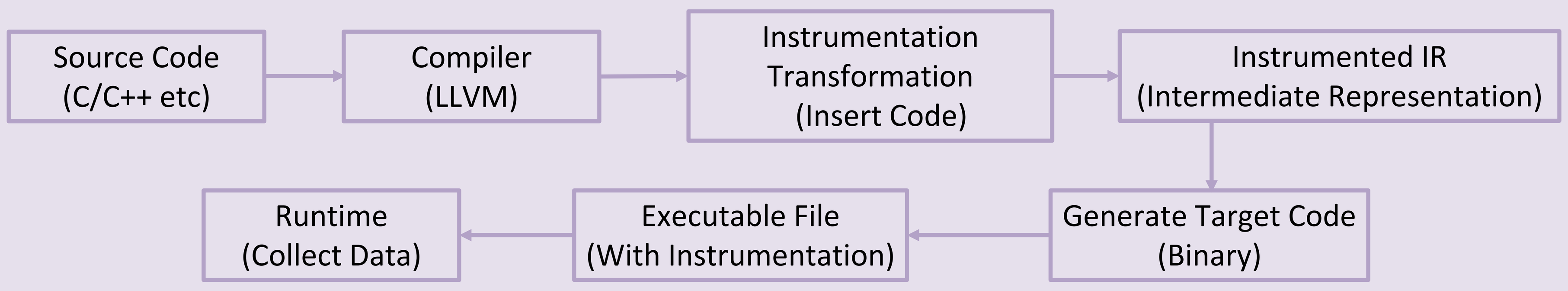
# Optimization Methods for Memory Access Monitoring Based on Hardware Page Protection

[1] **Yutian Fang**, National University of Defense Technology, Changsha, China
[2] **Ruibo Wang**, National University of Defense Technology, Changsha, China
[3] **Wenzhe Zhang**, National University of Defense Technology, Changsha, China

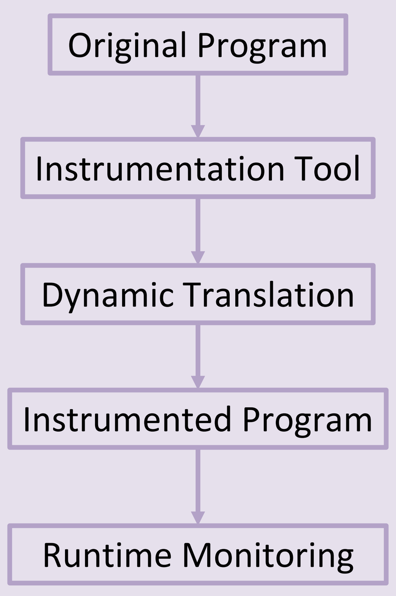
## Introduction

- Memory access monitoring refers to the monitoring and management of memory access behavior in computer systems.
- Software instrumentation is a technique for inserting additional code into the original program to facilitate the collection of runtime information, program debugging, monitoring, and analyzing performance.
- Memory access monitoring can also be implemented using hardware mechanisms provided by the processor, such as hardware page protection and memory protection keys.

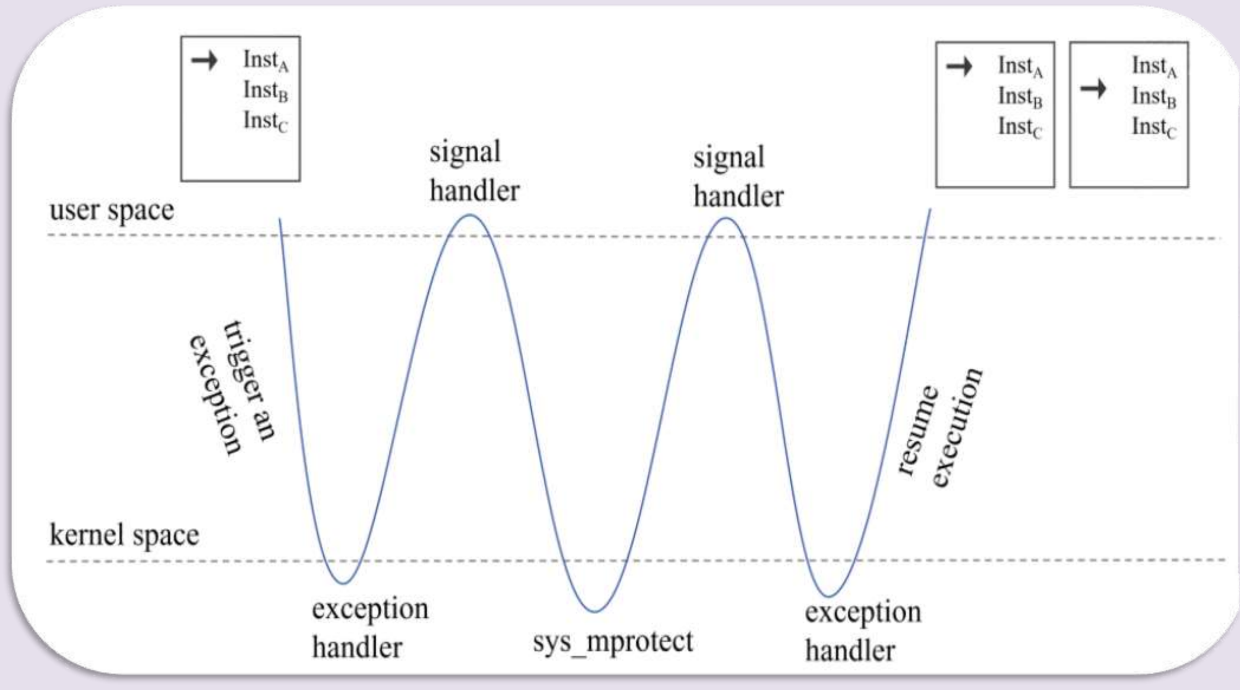
## Related works




Static Instrumentation



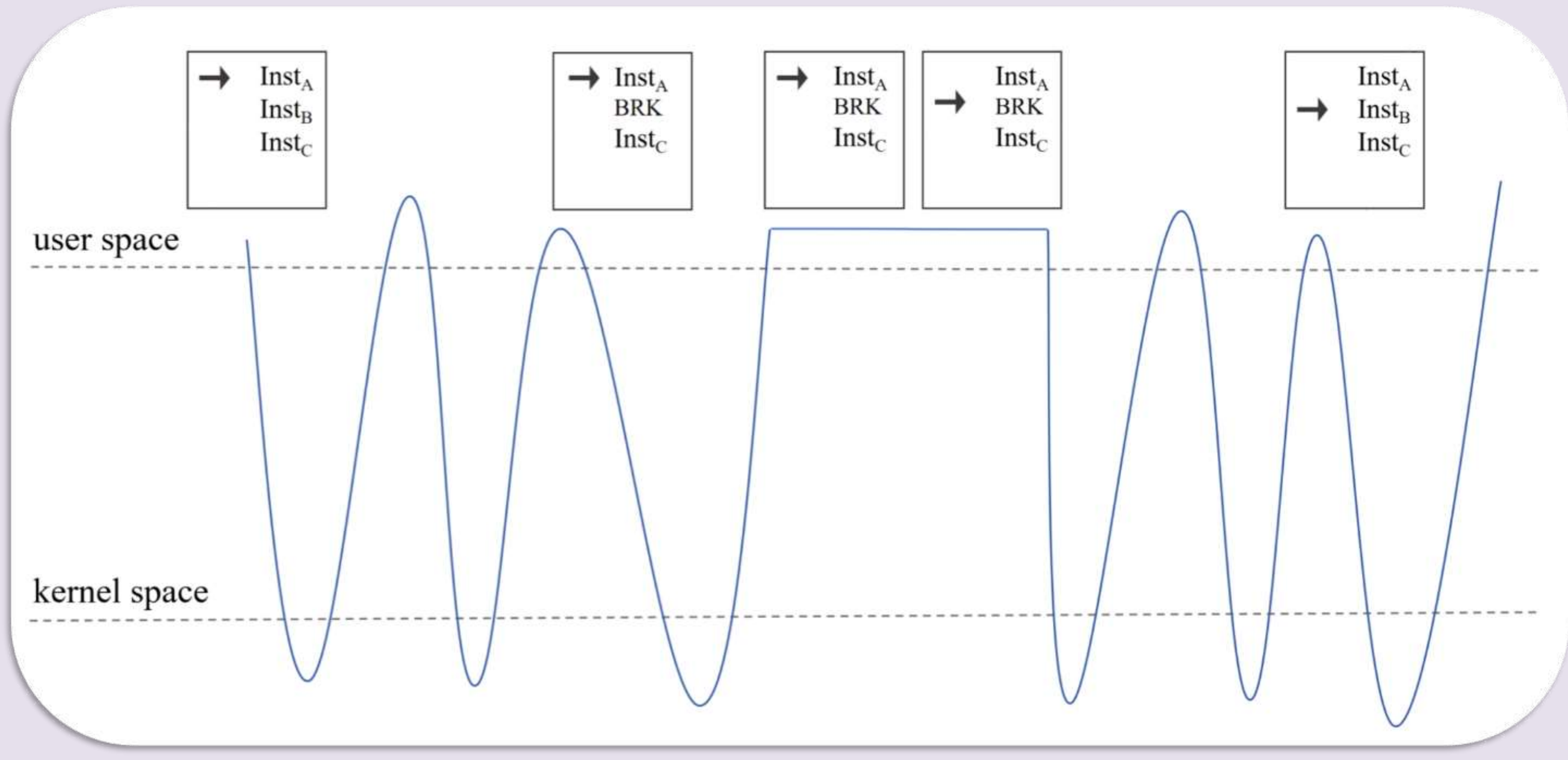

Dynamic Instrumentation




Hardware Page Protection

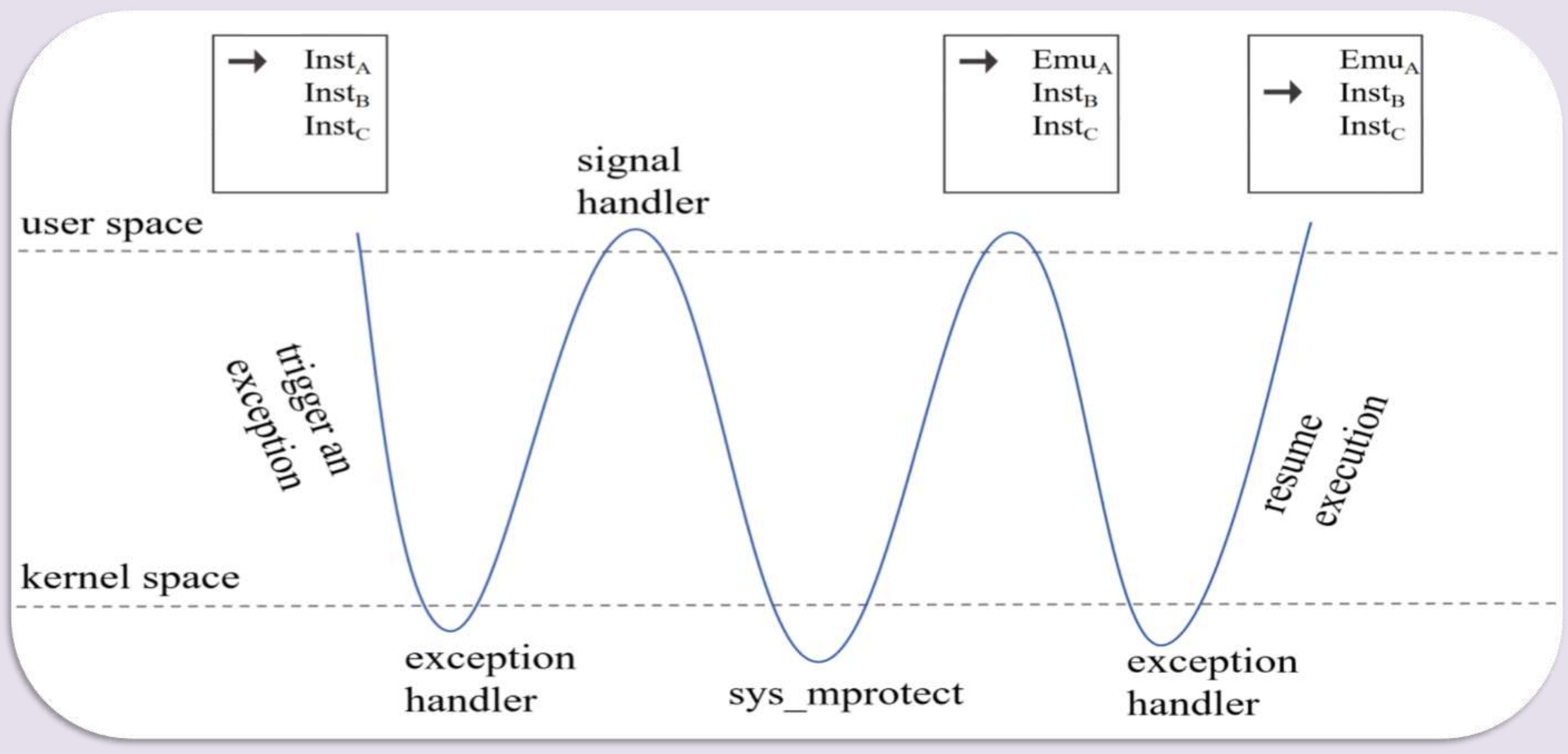
## Methods

- To achieve byte-grained memory access monitoring using hardware page protection, it is necessary to reapply access restrictions to memory page accessed by an instruction that triggers an exception as soon as instruction resumes execution after the exception.




Byte-Grained Memory Access Monitoring Based on Exception

- Emulating execution of memory access instructions directly in user mode can reduce an additional exception for each memory access operation, and exception handling introduces multiple context switches between kernel mode and user mode.




Optimization Method Based on Instruction Emulation

## Experiments

| Program | Runtime (sec) Based on exception | Runtime (sec) Based on instruction emulation | change |
|---|---|---|---|
| active-false | 6594.7 | 6013.7 | 8.8% |
| passive-false | 6780.5 | 6131.3 | 9.6% |
| Linux-scalability | 6300.2 | 5726.9 | 9.1% |
| threadtest | 5012.1 | 4566.0 | 8.9% |

Comparison of two methods

- In the four benchmarks, passive-false performs best after optimization, as it has a single memory access operation for each loop iteration during execution, making it very suitable as a benchmark for memory access monitoring.


Execution time reduction in Passive-false after optimization

## Conclusion

- To address the issues of coarse access control granularity and significant performance overhead in memory access monitoring scenarios based on hardware page protection mechanisms, we first proposed implementing byte-grained memory access monitoring using breakpoint exception. This approach overcomes the limitation of current hardware page protection mechanism, which only support page-grained access control.
- We further reduced additional context switching caused by breakpoint exception through emulating memory access instructions in page fault handling.
- When the proportion of memory accesses in the program remains relatively stable, the optimization effect is also relatively stable.
- As the number of memory accesses increases, the optimization effect based on instruction emulation also improves. In summary, implementing lightweight memory access monitoring is of significant importance in memory access-intensive programs.
- We use four benchmarks to test runtime of both approaches, and ultimately achieve an average of over 9% improvement in runtime performance.

## References

[1] M. Payer, K. Enrico, and R. G. Thomas. "Lightweight memory tracing." In 2013 USENIX Annual Technical Conference (USENIX ATC 13), pp. 115-126. 2013.
[2] S. Park, B. Madhuparna, and U. Alexandru. "DAOS: Data access-aware operating system." In Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing, pp. 4-15. 2022.
[3] K. Lu, W. Z. Zhang, X. P. Wang, M. Luján, and A. Nisbet. "Flexible page-level memory access monitoring based on virtualization hardware." ACM SIGPLAN Notices 52, no. 7 (2017): 201-213.
[4] S. S. Gong, A. Deniz, F. Pedro and M. Petros. "Snowboard: Finding kernel concurrency bugs through systematic inter-thread communication analysis." In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, pp. 66-83. 2021.
[5] H. J. Wang, J. D. Zhai, X. C. Tang, B. W. Yu, X. S. Ma, and W. G. Chen. "Spindle: Informed memory access monitoring." In 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18), pp. 561-574. 2018.
[6] Y. S. Q Zhang, K. Lu, Z. W. Wu, and W. Z. Zhang. "PMemTrace: Lightweight and Efficient Memory Access Monitoring for Persistent Memory." In Algorithms and Architectures for Parallel Processing: 22nd International Conference, ICA3PP 2022, Copenhagen, Denmark, October 10–12, 2022, Proceedings, pp. 81-97. Cham: Springer Nature Switzerland, 2023.
[7] K. Serebryany, B. Derek, P. Alexander, and V. Dmitriy. "{AddressSanitizer}: A fast address sanity checker." In 2012 USENIX annual technical conference (USENIX ATC 12), pp. 309-318. 2012.
[8] C. Lattner, and A. Vikram. "LLVM: A compilation framework for lifelong program analysis & transformation." In International symposium on code generation and optimization, 2004. CGO 2004., pp. 75-86. IEEE, 2004.
[9] C. K. Luk, R. Cohn, R. Muth, et al.Pin： building customized program analysis tools with dynamic instrumentation[J]. ACM Sigplan Notices, 2005, 40 (6) :190-200.
[10] M. Back, M. Charney, R. Cohn, et al.Analyzing parallel programs with Pin[J]. Computer, 2010, 43 (3) :34-41.
[11] K. Hzelwood, A. Klauser. A dynamic binary instrumentation engine for the ARM architecture[C]//Proc of International Conference on Compilers, Architecture and Synthesis for Embedded Systems.New York:ACM Press, 2006:261-270.
[12] C. Nachiketa, M. Srijoni, P. D. Partha; C. Amlan.PARALLELC-ASSIST:Productivity Accelerator Suite based on Dynamic Instrumentation[J].IEEE Access,2023,Vol.11: 1
[13] Z. W. Wu, K. Lu, A. Nisbet, W. Z. Zhang, and M. Luján. "PMThreads: Persistent memory threads harnessing versioned shadow copies." In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 623-637. 2020.
[14] E.D. Berger, K.S. McKinley, R.D. Blumofe, P.R. Wilson.View Correspondence (jump link).Hoard: A scalable memory allocator for multithreaded applications(Article)[J].SIGPLAN Notices (ACM SpecialInterest Group on Programming Languages),2000,Vol.35(11): 117-128.