VAPRO: Performance Variance Detection and Diagnosis for Production-Run Parallel Applications

PPoPP22 @ Virtual, 2022.04.05

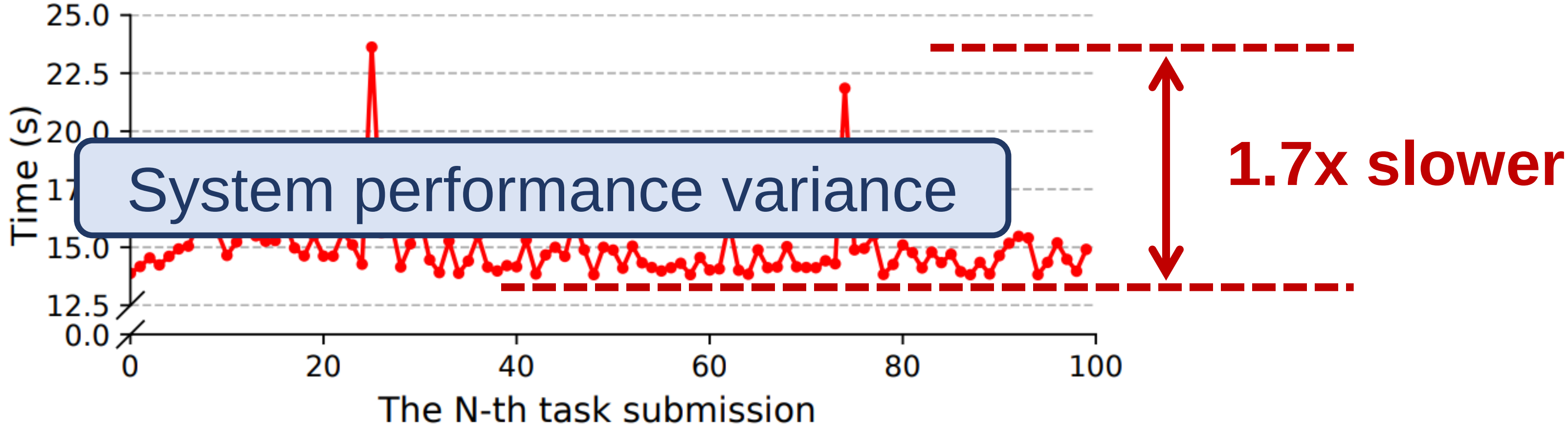
Liyan Zheng, Jidong Zhai, Xiongchao Tang,
Haojie Wang, Teng Yu, Yuyang jin,
Shuaiwen Leon Song, and Wenguang Chen

Tsinghua University

The University of Sydney

What happened to my program

Run a 256-process program (NPB-CG) on the same nodes 100 times



Consume more resources

Hard to understand its behavior

Bad nodes?

Optimization works?

Re-submission?

When, where, and why variance happens?

Existing approaches & limitations

1. Benchmark: easy but intrusive
2. Profile/trace: widely used but expert efforts required
3. Static analysis-based method: automatic but source code required and only for detection

- Ideal variance profilers for production environments

No source code | Diagnosis | Automatic

How to detect and diagnose variance without source code

Observation

- Fixed-workload fragment (FWF): fragments of a program execution with the same workload

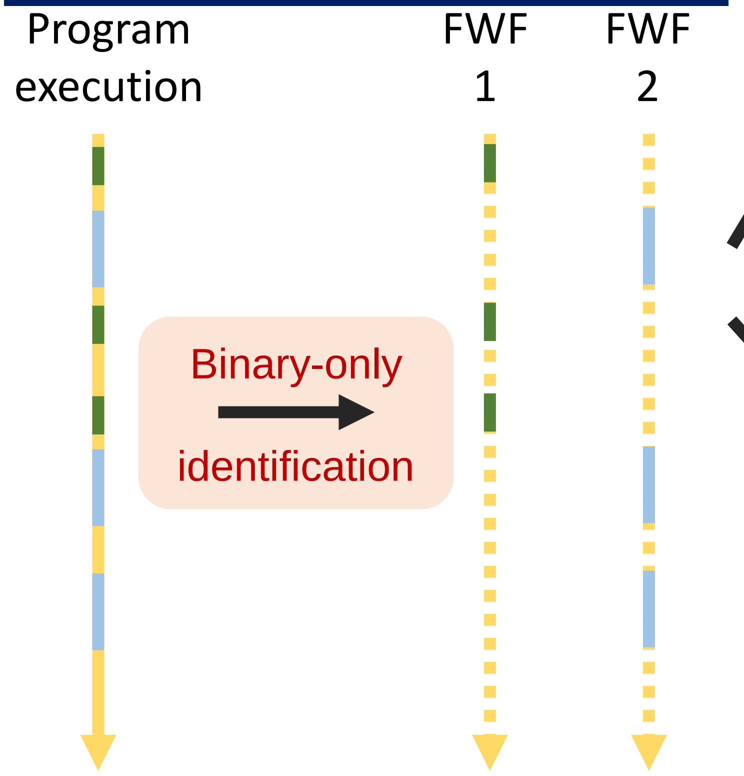
Performance = Workload / Time

Benchmark inside programs

Key idea: identify fixed-workload fragments and leverage them for variance detection and diagnosis

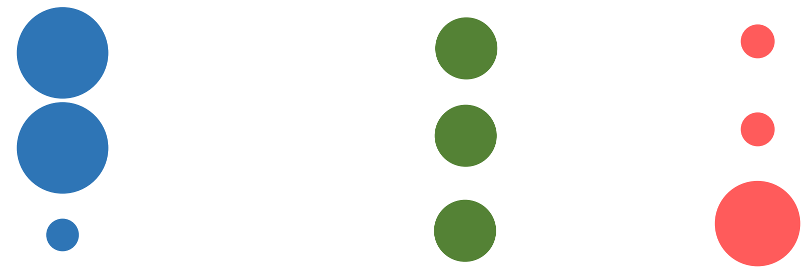
Vapro in a nutshell

(1) Find fixed-workload fragments



(2) Detect variance

Performance = Workload / Time



(3) Diagnose variance

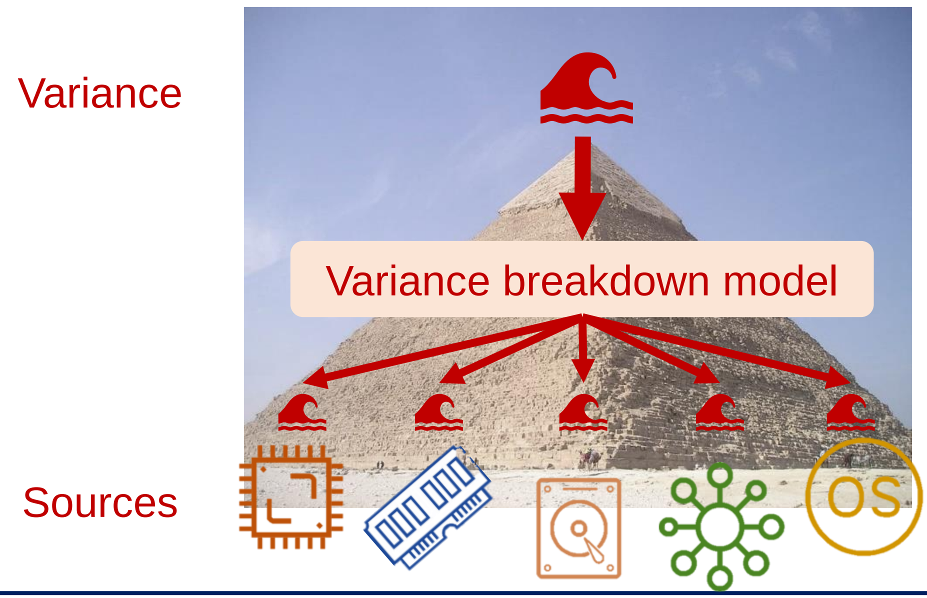
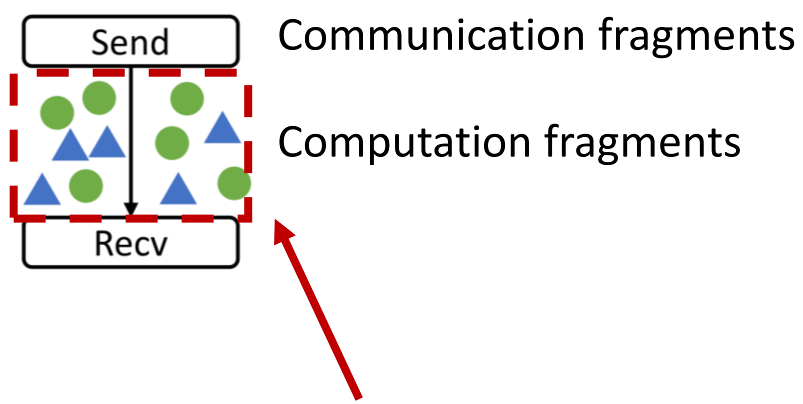


Image source (distributed with CC0): https://www.maxpixel.net/Egyptians-Egypt-Chephren-Pyramid-Culture-Gizeh-483

Fragment collection

- Fragment: an execution of a code snippet
  - Three types: computation/communication/IO fragments
- Splitting program executions by external function invocations
  - Communication, IO, and threading operations
  - Intercept external function invocations with dynamic linker

```
Send(…);
if (…) calc0();
else calc1();
Recv(…);
```

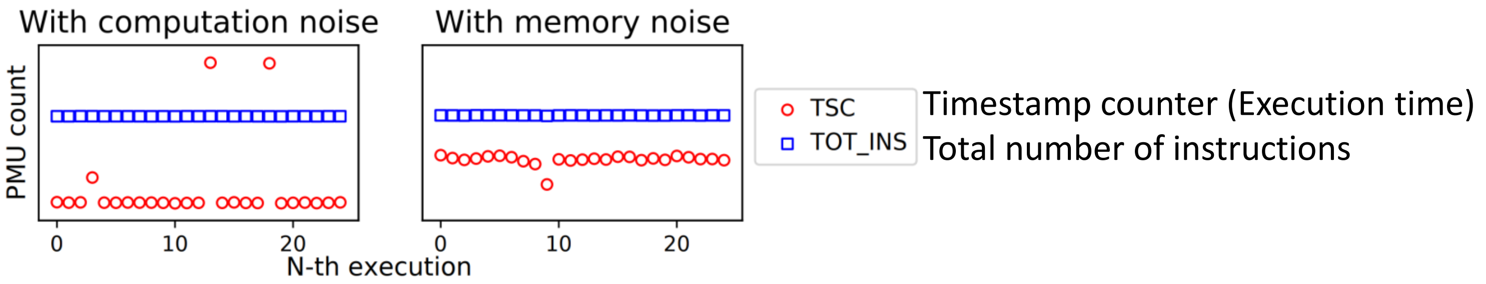


- **Challenge**: fragments with different workloads are mixed
  - How to represent workload when only binaries are available?

Workload identification – computation

- Workload measuring should remain stable under noises
- Identify computation workload by Performance Monitor Unit (PMU) metrics
    - TOT_INS is desired since it only counts the instructions of the specified process

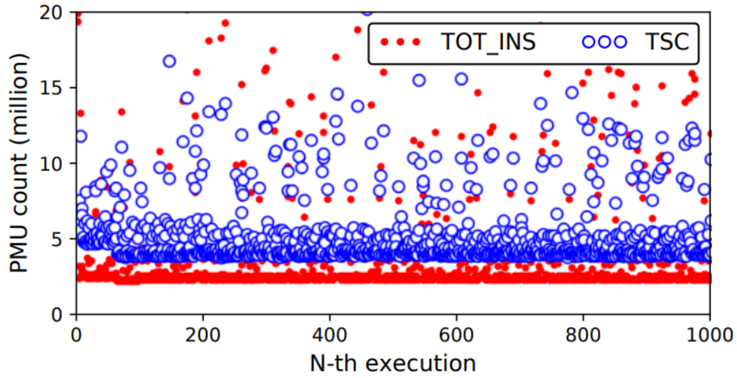
Fixed-workload computation fragments on Tianhe-2A



- More candidate metrics: load/store/branch instructions, …

Workload identification – communication & IO

Fixed-workload communication fragments (MPI_Send) on Tianhe-2A



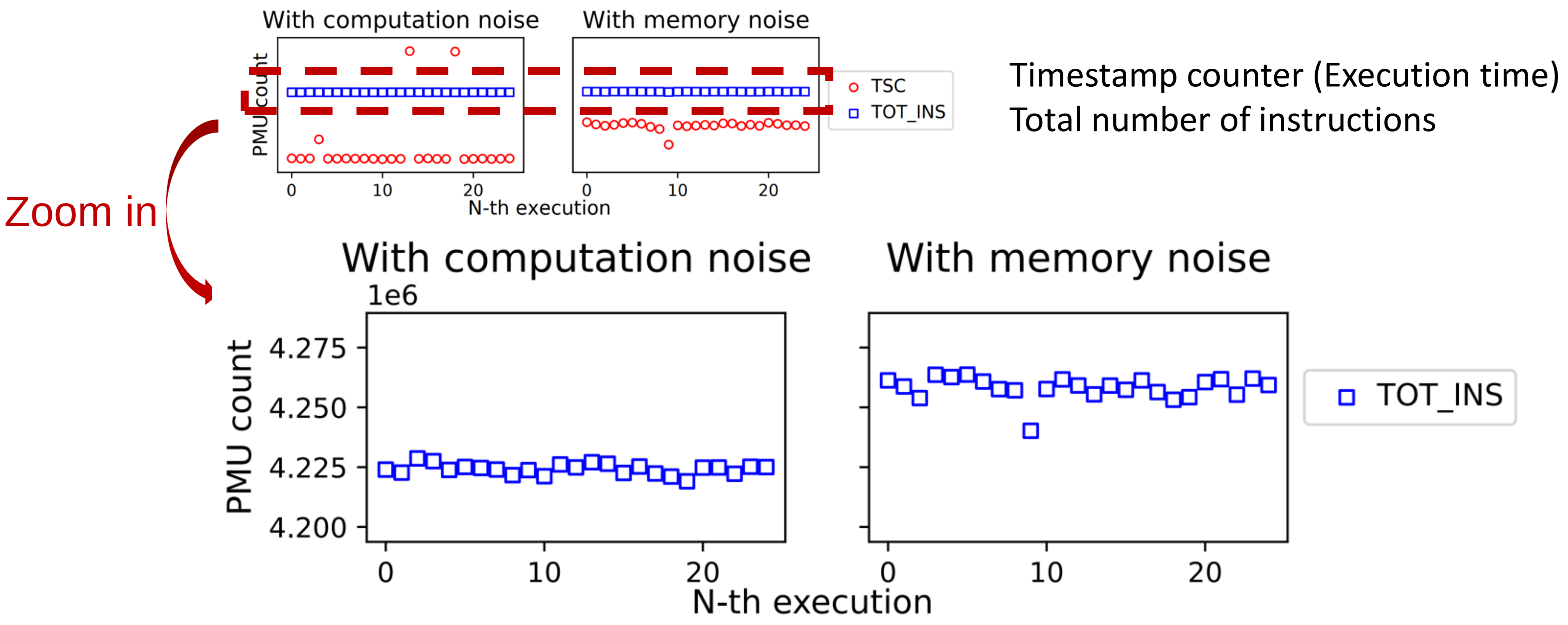
- PMU metrics are inconsistent with communication workload
- Use function arguments to identify workload
  - E.g., MPI_Send(buf, count, datatype, dest, tag, comm)

How to identify FWF according to workload?

FWF identification

- Communication/IO workload is well-described
- Computation: inherent error of PMU mechanism

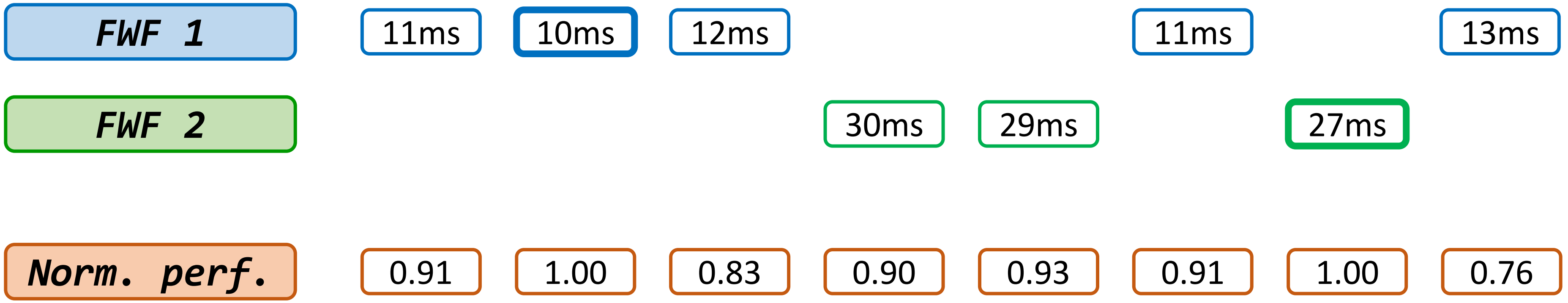
Fixed-workload computation fragments on Tianhe-2A



- Clustering workload metrics to identify FWF

Performance calculation

- Report application performance to users
- Normalize execution time to relative performance
  - Shortest time → relative performance 1
  - Longer time → smaller relative performance (0~1)



- Report performance of a period
  - Performance of a period is the weighted average of all FWFs in the period with execution time as weight

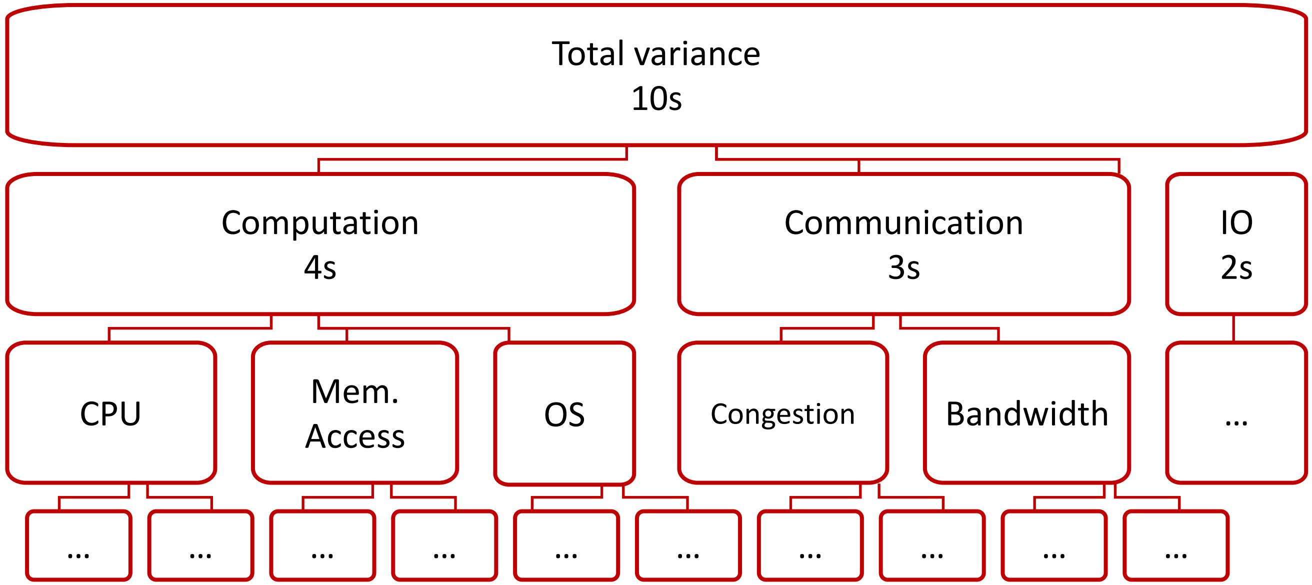
Diagnosis

- Challenge: infeasible to monitor all performance data
  - Hundreds of CPU PMU metrics
  - Software performance metrics

- Idea: can we divide variance into several factors?

Variance breakdown

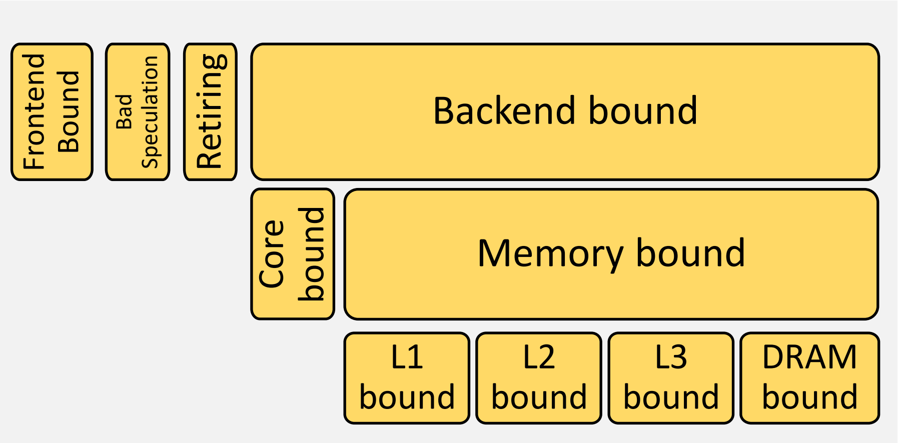
- Divide extra execution time caused by variance
  - Three factors: computation, communication, IO



- How about finer-grained factors
  - Such as detailed events in a CPU and an OS

Diagnosis – hardware PMU

- Similar structures in PMU of CPUs
- Top-down Microarchitecture Analysis (TMA) method[5]



1. Organizing functional blocks of pipeline into a tree hierarchy
2. Enables cycle-accounting for functional blocks
    - E.g.: $\text{Time of frontend bound} = \frac{IDQ\_UOPS\_NOT\_DELIVERED.CORE}{4 \times CPU\_CLK\_UNHALTED.THREAD}$ (for Intel Ivy Bridge)

- Breakdown of execution time

Can we break down variance?

Quantify the time of TMA factors

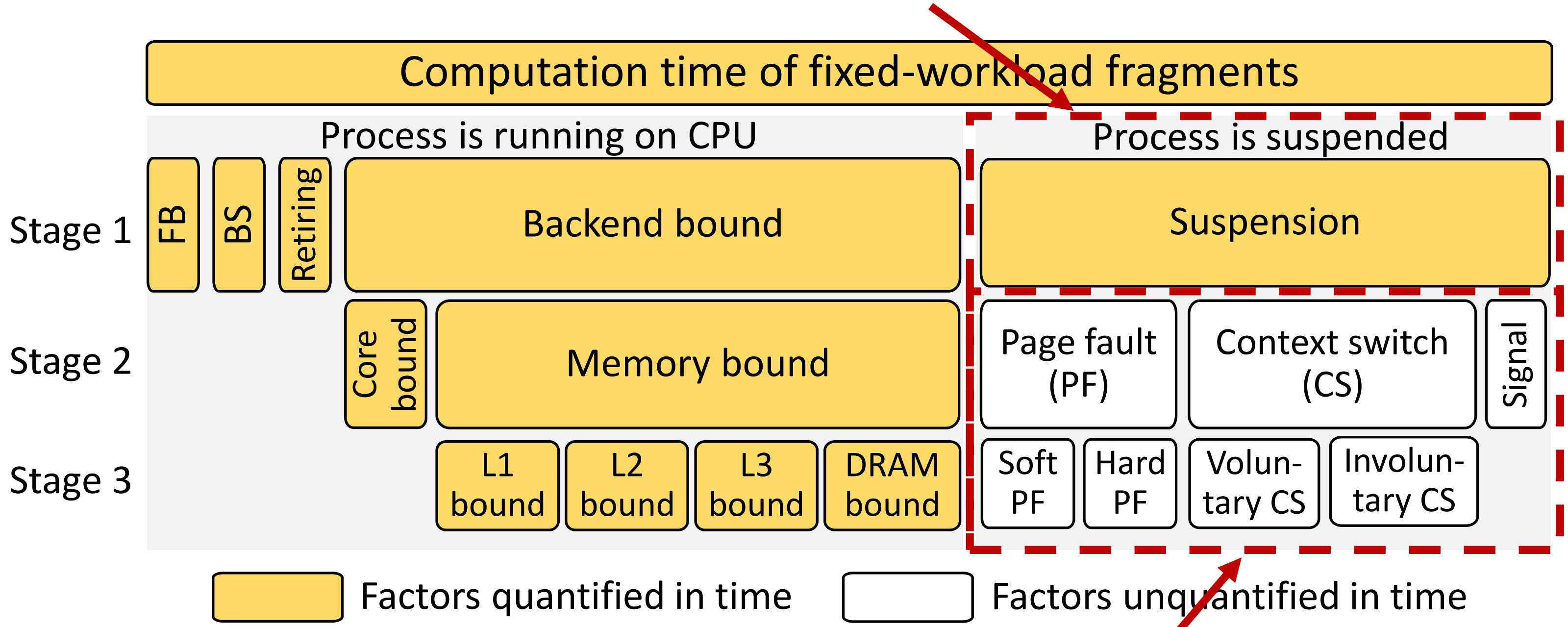
- Differentiate normal fragments and slow fragments
  - Time of execution → Time of variance

	Frontend Bound	Bad Speculation	Retiring	Backend bound
Slow fragment	1s	1s	1s	2s
Normal fragment	1s	1s	1s	1s
Differences				
Variance	0s	0s	0s	1s

- Pinpoint which reason causes the variance

Diagnosis – software metrics

- Can we apply this approach beyond TMA metrics?
- Variance breakdown model
  - Support both hardware and software factors



- **Problem**: Software factors are not quantified in time
  - Linux provides the count of events, instead of time

Quantify the time of software factors

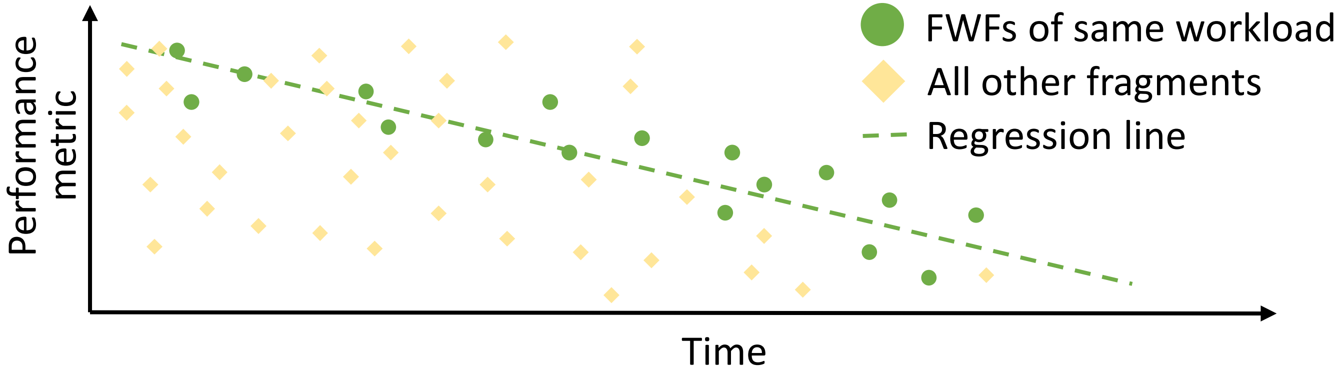
- We have a great number of fixed-workload fragments
  - Time & performance counters
  - Same workload

Statistical method for estimating the time of each factor

- Assume each metric has a linear impact on execution time

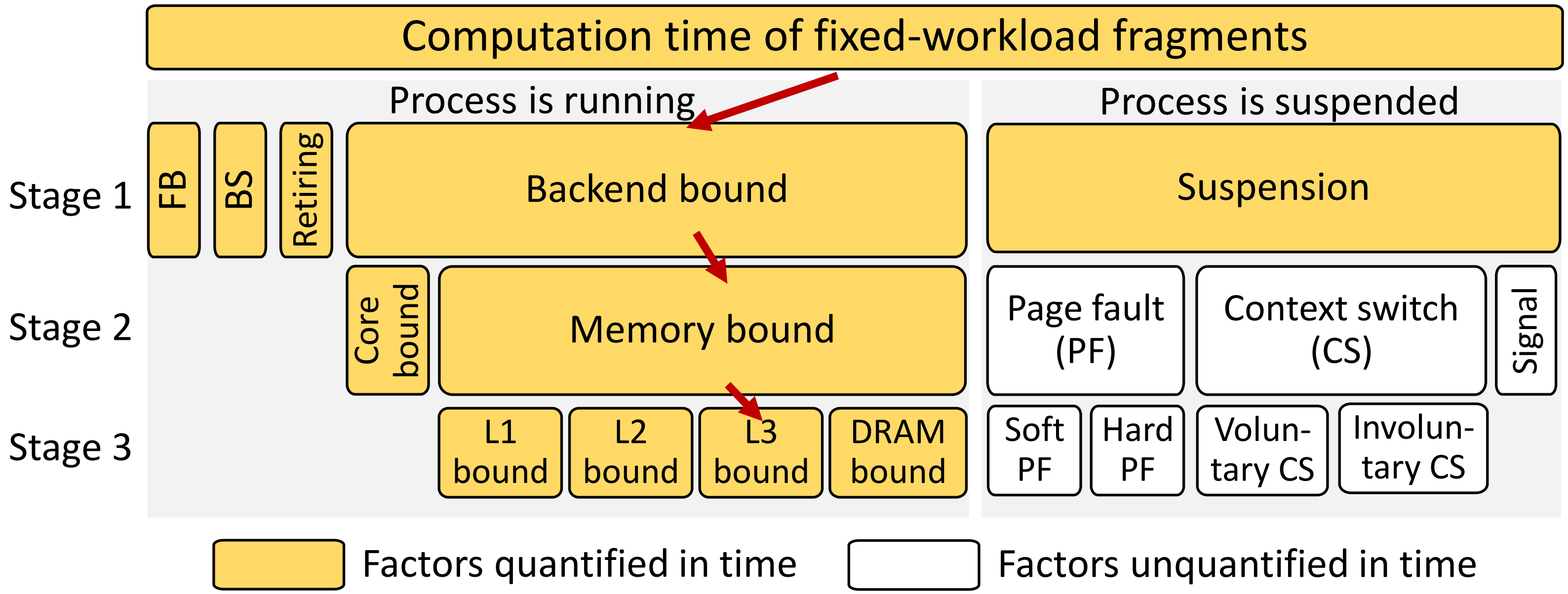
$$time_i = \beta_0 + \beta_1 Factor_{i1} + \cdots + \beta_n Factor_{in} + \epsilon_i$$

  - β_j is the impact of a single unit of Factor j on execution time
- With a regression on FWFs, we can quantify variance for each factor



Progressive variance diagnosis

- Variance is inclusive on the performance breakdown model
- Diagnose performance variance stage-by-stage
  - Locate coarse-grained factors first and drill down the hierarchy



- Small overhead: only collects performance counters for current stage

Evaluation

Platforms:

Tianhe-2A system in NSCC-Guangzhou **(for MPI programs)**

- Dual Xeon E5-2692(v2) (24 cores in total) and 64GB memory.

Gorgon cluster in Tsinghua **(for OpenMP programs)**

- Dual Xeon E5-2670(v3) (24 cores in total) and 128GB memory

Benchmarks:

MPI programs: CESM, AMG, and 7 programs from the NPB benchmark suite

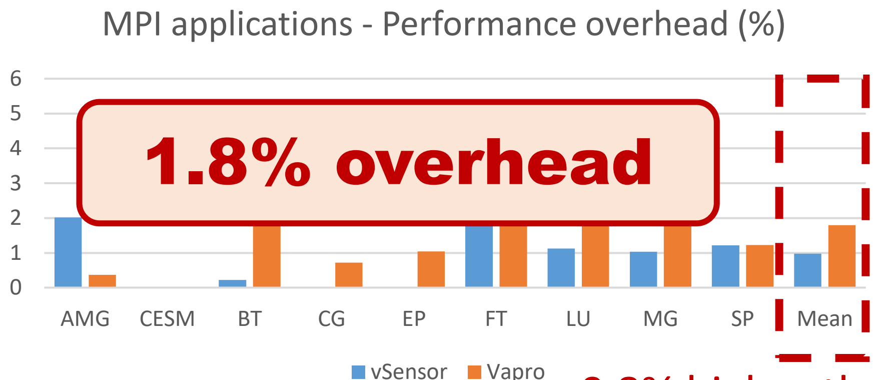
- Up to 2048 processes

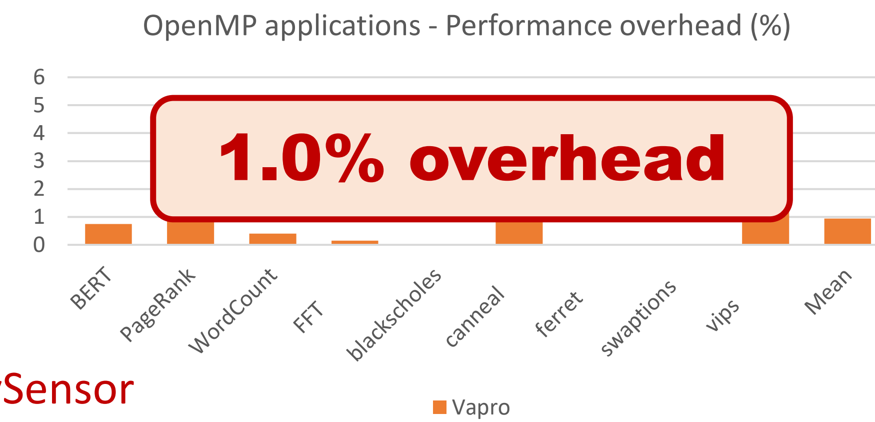
OpenMP programs: BERT, PageRank, WordCount, and 6 programs from the PARSEC benchmark suite

Baseline:

vSensor[1]: finds code snippets with fixed workload as benchmark **with compiler analysis**

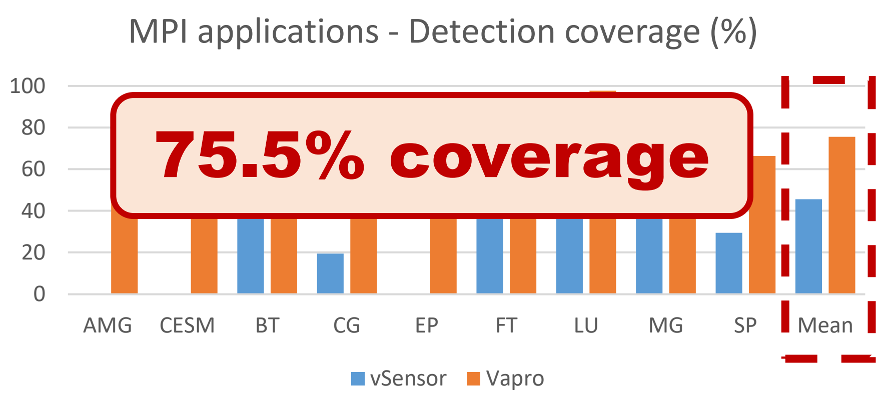
Basic results

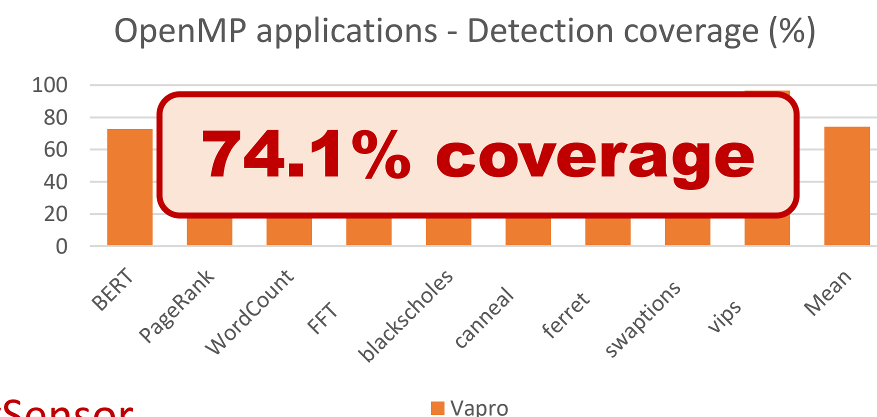




0.8% higher than vSensor
(which is based on static analysis)

- Detection coverage $= \dfrac{Time\ covered\ by\ FKFs}{Exuection\ time}$





30.8% higher than vSensor

Result visualization

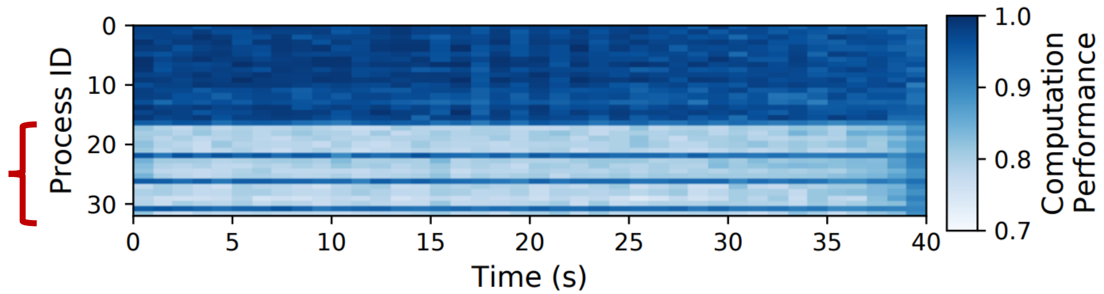
Light color blocks → bad performance

*An example result for computation performance, detected for a 128 processes program.

Case studies – a CPU cache bug

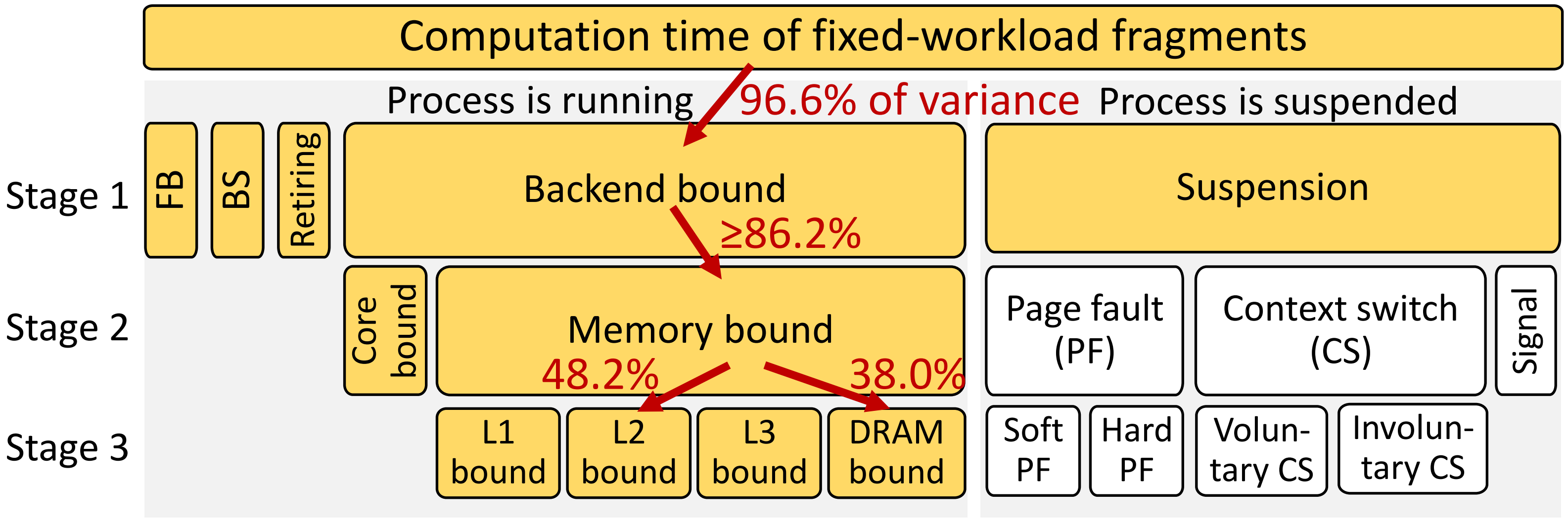
- HPL on dual Intel Xeon Gold 6140 processers
  - The benchmark of TOP500
  - Stable performance
- **Detection:** an abnormal run with 22.2% longer execution time

Variance only happens on Socket 1



Case studies – a CPU cache bug (cont.)

- An abnormal HPL run with 1.22x execution time
- **Diagnosis**

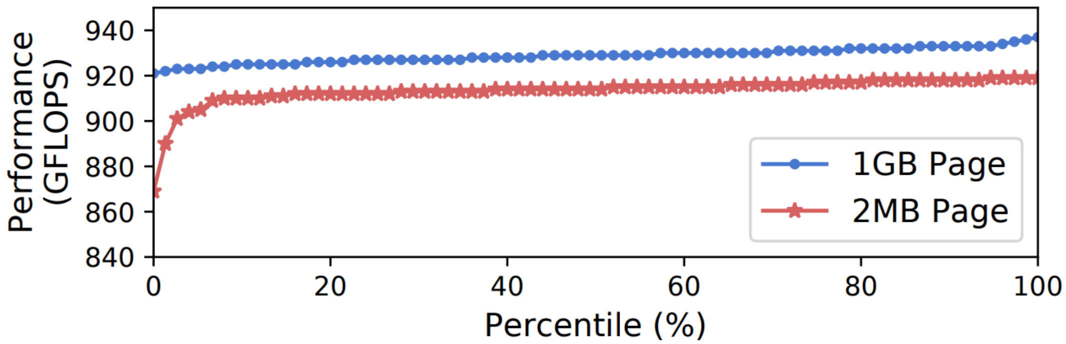


Extra cache misses and memory accesses impair the performance

Case studies – a CPU cache bug (cont.)

- **Validation:** an Intel processor hardware bug [3,4]
  - Makes data in the L2 cache evicted
  - Randomly generates significant slowdowns
- **Solution:** huge page mitigates this problem

Distribution of HPL performance with huge page



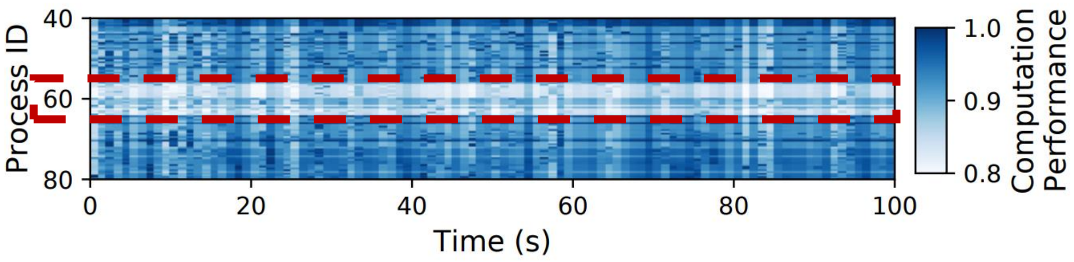
reduce standard derivation of execution time by 51.3%

- Vapro avoids time-consuming re-executions for diagnosing this non-deterministic problem

Case studies – memory problem

- **Detection:** processes on a node have low performance

128-process Nekbone on Tianhe-2A

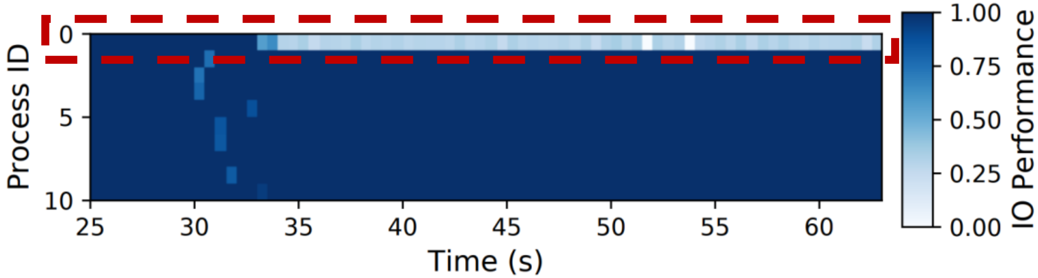


- **Diagnosis**: backend bound explains 97.2% of the slowdown
- **Validation:** memory bandwidth of the problematic node is 15.5% lower than others
- **Solution:** replacing this node yielding a 1.24× speedup

Case studies – IO performance variance

- RAxML: a phylogenetic analysis application
- **Detection:** 10 executions ranges from 41.1s to 68.0s
  - Both computation and communication is stable

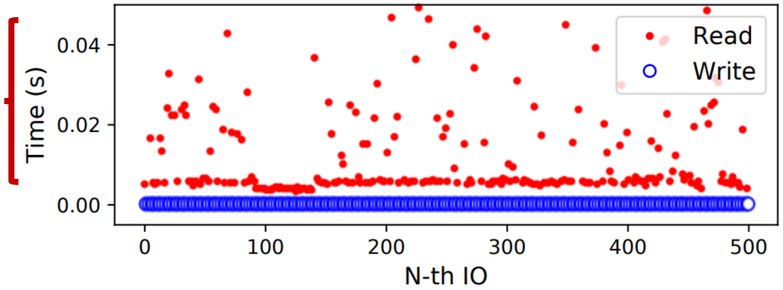
IO performance of 512-process RAxML on Tianhe-2A



Process 0 has significantly lower performance

Time of consecutive read and write operations fixed workload in RAxML

Significant variance for Read on distributed FS



- **Solution:** A simple file buffer yielding a 1.18x speedup and a 73.5% reduction in the standard deviation of overall execution time

Conclusion

The take away point:

Challenges		Techniques
No source code	☞	Workload identification
Diagnosis	☞	Variance breakdown

Vapro is a va riance pro filer that can detect and diagnose variance without source code.

More details in the paper

Q&A

Thank you!

PACMAN

pacman.cs.tsinghua.edu.cn

Part of References

[1] Xiongchao Tang, Jidong Zhai, Xuehai Qian, Bingsheng He, Wei Xue, and Wenguang Chen. 2018. vSensor: leveraging fixed-workload snippets of programs for performance variance detection. In Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming (PPoPP'18). 124–136.

[2] Jeffrey Vetter and Chris Chambreau. 2005. mpip: Lightweight, scalable mpi profiling.

[3] Intel. 2018. Addressing Potential DGEMM/HPL Perf Variability on 24-Core Intel Xeon Processor Scalable Family. White paper, number 606269, revision 1.0.

[4] John D McCalpin. 2018. HPL and DGEMM performance variability on the Xeon Platinum 8160 processor. In SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 225–237.

[5] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14). IEEE, 35–44.