SuperTwin and API

Deliverable No: D4.5
Deliverable Title: SuperTwin and API
Deliverable Publish Date: 31 March 2024

Project Title: SparCity: An Optimization and Co-design Framework for Sparse Computation
Call ID: H2020-JTI-EuroHPC-2019-1
Project No: 956213
Project Duration: 36 months
Project Start Date: 1 April 2021
Contact: sparcity-project-group@ku.edu.tr

List of partners:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	Koç University	KU	Turkey
2	Sabancı University	SU	Turkey
3	Simula Research Laboratory AS	Simula	Norway
4	Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa	INESC-ID	Portugal
5	Ludwig-Maximilians-Universität München	LMU	Germany
6	Graphcore AS*	Graphcore	Norway

*until M21

CONTENTS

1 INTRODUCTION

The SparCity project is funded by EuroHPC JU (the European High Performance Computing Joint Undertaking) under the 2019 call of Extreme-Scale Computing and Data-Driven Technologies for research and innovation actions. SparCity aims to create a supercomputing framework that will provide efficient algorithms and coherent tools specifically designed for maximizing the performance and energy efficiency of sparse computations on emerging High-Performance Computing (HPC) systems, while also opening up new usage areas for sparse computations in data analytics and deep learning.

Sparse computations are commonly found at the heart of many important applications, but at the same time, it is challenging to achieve high performance when performing sparse computations. SparCity delivers a coherent collection of innovative algorithms and tools for enabling high efficiency of sparse computations on emerging hardware platforms. More specifically, the objectives of the project are:

- to develop a comprehensive application and data characterization mechanism for sparse computation based on the state-of-the-art analytical and machine-learning-based performance and energy models,
- to develop advanced node-level static and dynamic code optimizations designed for massive and heterogeneous parallel architectures with complex memory hierarchy for sparse computation,
- to devise topology-aware partitioning algorithms and communication optimizations to boost the efficiency of system-level parallelism,
- to create digital SuperTwins of supercomputers to evaluate and simulate what-if hardware scenarios,
- to demonstrate the effectiveness and usability of the SparCity framework by enhancing the computing scale and energy efficiency of challenging real-life applications.
- to deliver a robust, well-supported and documented SparCity framework into the hands of computational scientists, data analysts, and deep learning end-users from industry and academia.

1.1 OBJECTIVES OF THIS DELIVERABLE

The diversity within the data center and HPC ecosystems brings challenges in system design, performance engineering, and optimization, necessitating innovative models and approaches to skilfully steer inside a complicated computing environment. Performance variations caused by hardware capabilities and software factors such as load imbalances, CPU throttling, reduced frequency, shared resource contention, and network congestion can result in up to a 100% difference in execution times for the same computation and input data.[1] To find the root causes of these variations, one requires a comprehensive knowledge on the computational system generated via novel monitoring, profiling, and forecasting tools.

[1] Aksar. "E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems"". *Euro-Par 2021: Parallel Processing*. Springer International Publishing, 2021, pp. 70–85.

Various tools that can systematically collect and store information from performance metric sources have been developed.[2] However, they do not create a comprehensive *knowledge base* and most do not provide a live and/or automated analysis framework. Hence, there is a need for innovative tools that are capable of tracking every component within a single server or a full-scale cluster effectively leveraging kernel statistics and physical hardware performance counters. Capturing the hidden patterns and providing insights, digital twins, virtual models that mirror physical systems, can be an imminent solution to model, observe, and analyze the inherent performance variability in today's parallel and distributed systems. Crafting a virtual replica of a physical system, a twin can facilitate a meticulous exploration and analysis of system behavior, enabling researchers and engineers to optimize system configurations, simulate different scenarios, and predict performance under various conditions.

We propose SuperTwin, a major step to create a virtual modeling and simulation framework to untangle and comprehend the underlying complexities and interactions, thereby making judicious decision-making, enhanced system performance, and efficient resource management possible. Within SuperTwin, the *knowledge base* is deeply incorporated in almost every functionality, and contains the machine specification, topology, configuration parameters of the tools/frameworks used within, and historical job metadata linked to the sampled performance metrics.

While digital twin ontologies exist for various domains, such as industrial machines,[3] cities[4] and smart buildings[5] there is limited work on ontologies for a computational system. Moreover, existing twins collect data from stable sources, e.g., a thermostat, with a constant metric and frequency. On the contrary, SuperTwin tackles highly sensitive and volatile platforms, which impose a novel set of challenges to its design, such as being minimally disruptive in terms of performance and sustaining high accuracy with a limited number of performance-monitoring units (PMUs) – specialized components leveraging *programmable* registers dedicated to monitoring and recording performance-related events and metrics.

This deliverable describes the design and validation of SuperTwin , focusing on its potential to address the challenges posed by the ever-evolving complex landscape of computational systems. In addition, the API documentation is added as an Appendix. In this deliverable:

- To our knowledge, we propose the design of the first open-source digital twin framework, SuperTwin, tailored for data center and HPC servers.

- We derive an ontology as a guide to comprehending data center and HPC servers in an

[2]Brandt. *Lightweight Distributed Metric Service (LDMS): Run-time Resource Utilization Monitoring.* English. Tech. rep. SAND2013-6521C. Sandia National Lab. (SNL-CA), Livermore, CA (United States); Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), 2013. URL: https://www.osti.gov/biblio/1106397 (visited on 09/27/2021); Agelastos. *The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications.* English. Tech. rep. SAND2014-19868C. Sandia National Lab. (SNL-NM), Albuquerque, NM (United States); Sandia National Lab. (SNL-CA), Livermore, CA (United States), 2014. DOI: 10.1109/SC.2014.18. URL: https://www.osti.gov/biblio/1315267 (visited on 09/27/2021); Nagios. *Nagios*. https://www.nagios.org/. Accessed: 2022-12-12. 2022; Adhianto. "HPCTOOLKIT: Tools for Performance Analysis of Optimized Parallel Programs Http://Hpctoolkit.Org". *Concurr. Comput.: Pract. Exper.* 22.6 (2010), pp. 685–701. ISSN: 1532-0626; Roy. "PerfAugur: Robust diagnostics for performance anomalies in cloud services". *2015 IEEE 31st International Conference on Data Engineering*. 2015, pp. 1167–1178. DOI: 10.1109/ICDE.2015.7113365.

[3]Steinmetz. "Internet of Things Ontology for Digital Twin in Cyber Physical Systems". *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2018, pp. 154–159. DOI: 10.1109/SBESC.2018.00030.

[4]Deng. "A systematic review of a digital twin city: A new pattern of urban governance toward smart cities". *Journal of Management Science and Engineering* 6.2 (2021), pp. 125–134. ISSN: 2096-2320. DOI: https://doi.org/10.1016/j.jmse.2021.03.003.

[5]Qiuchen Lu et al. "Developing a dynamic digital twin at building and city levels: A case study of the West Cambridge campus". *Journal of Management in Engineering* 36 (2019). DOI: 10.1061/(ASCE)ME.1943-5479.0000763.

organized and intuitive manner. Based on this, SuperTwin constructs a *knowledge base* which is used to perform all the tasks on the linked performance data.

- To handle the architectural diversity and be as automated as possible, SuperTwin leverages an *abstraction layer*.
- Modern profiling capabilities such as live monitoring and cache-aware roofline analysis, are designed to provide real-time insights with minimal interference.
- We demonstrate the capabilities of SuperTwin on many microbenchmarks and SpMV kernels.

We have used a variety of high-end servers to assess the practical value of SuperTwin. While doing this, we have also observed and measured the overhead incurred, precision of measurements and the number of data points that can be collected per second. SuperTwin design is hardware-agnostic to let users employ it on various systems and allow easy inclusion of any device type, including GPUs. However, current GPU-monitoring methods only provide aggregated readings and lack the functionality to collect live time-series data. In the current design, SuperTwin presents aggregated information after a GPU kernel terminates and live information from NVML. While distributed system support is valuable, this deliverable exclusively focuses on multi and manycore single-node settings. Preliminary experiments with 4-8 nodes reveal no additional overhead in SuperTwin.

1.2 WORK PERFORMED

In this deliverable, we introduce the SuperTwin which is an open-source framework designed and implemented to generate a digital twin for multi and manycore systems. Leveraging its comprehensive Knowledge Base, which is built upon an HPC-specific ontology aimed at providing a systematic and intuitive guide for comprehending the performance of data centers or HPC servers, it rigorously manages telemetry samplers, databases, and visualization frameworks. The Knowledge Base is generated through an in-depth probing of the computational system. It enables the configuration and monitoring of performance metric samplers, the generation of real-time visualizations, the establishment of linked-data connections among acquired information, and the generation of queries for advanced analysis. With its Abstraction Layer, SuperTwin can be used as an automated tool for low-level profiling even on components, e.g., CPU sockets, from different vendors. It has modern profiling capabilities, including live cache-aware roofline modeling, crafted to provide real-time insights without impeding system performance. The capabilities of SuperTwin have been demonstrated on multiple multicore architectures using various microbenchmarks and a commonly used kernel, sparse-matrix vector multiplication (SpMV), which is often the bottleneck of many computations.

1.3 DEVIATIONS AND COUNTER MEASURES

There was no deviation from the work plan.

1.4 RESOURCES

All the source codes and documentation can be found at.[6] It is expected that SuperTwin will be further extended by us and external contributors to add new functionalities. In the worst case,

[6] https://github.com/sparcityeu/Digital-SuperTwin

we will maintain and regularly update the respective SparCity GitHub repository if/when bugs are reported.

2 SUPERTWIN AND API

To our knowledge, there exists no work on using digital twins to model HPC systems; the literature can be investigated in three contexts; monitoring frameworks, profiling methods, and digital twin ontologies. To systematically collect and analyze information from performance metric sources, several frameworks have been developed, e.g., LDMS,[7] HPC-Toolkit,[8] Ganglia,[9] Nagios,[10] and PerfAugur.[11] E2EWatch[12] specializes in system-wide monitoring using Linux metrics and focuses on anomaly classification and detection. ClusterCockpit,[13] a more recent tool, reports performance metrics from distributed systems to InfluxDB and offers monitoring dashboards and job history queries. However, these tools have limitations, such as supporting only preselected, a fixed set of metrics and lacking a comprehensive knowledge representation and linked-data capabilities.

Linked data is used in different branches of science for knowledge management, such as biology[14] and physics.[15] RDF (Resource Description Framework) is a standardized approach for organizing data as *triples*, a source node (referred to as a subject), an edge name (known as a predicate), and a target node (called an object). To enhance this structure, RDFs incorporate identifiers known as IRIs (Internationalized Resource Identifiers) for node descriptions and properties. JSON-LD, a serialization of RDF data, has unique *attributes*, on par with RDF, which makes it different from the JSON format. The most common attributes are `@context`, `@id` and `type`. With these, a JSON-LD describes the datatypes, and how to parse and process them. This allows the creation of large-scale twins of interconnected systems from their building blocks. For each domain, unique document structures, i.e., ontologies, are designed to keep static metadata. For liveness, new triples need to be continuously injected, which makes these structures impractical for managing time-series data as is.[16]

The current tools fall short of addressing modern system complexities. They typically lack support for both Linux (software) and hardware performance counter metrics, lack flexible configuration options, and are ill-equipped to create digital twins. This is problematic when dealing with performance variations caused by multiple factors, like a performance sampler on L1 cache bandwidth being unaware of system issues such as resource contention or thermal throttling. Thus, there is a gap in the literature where both ends should meet.

Performance Co-Pilot (PCP)[17] is a metric collection, transport, and storage tool that can be configured to sample every available metric counter on hardware and kernel, and energy usage of a system by Running Average Power Limit (RAPL)[18] and `perf` interfaces. It supports varying

[7]Brandt, *Lightweight Distributed Metric Service (LDMS)*; Agelastos, *The Lightweight Distributed Metric Service*.

[8]Adhianto, "HPCTOOLKIT: Tools for Performance Analysis of Optimized Parallel Programs Http://Hpctoolkit.Org".

[9]Ganglia. *Monitoring system*. 2022. URL: http://ganglia.sourceforge.net/ (visited on 12/12/2022).

[10]Nagios, *Nagios*.

[11]Roy, "PerfAugur: Robust diagnostics for performance anomalies in cloud services".

[12]Aksar, "E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems"".

[13]*Cluster Cockpit*. https://www.clustercockpit.org/. Accessed on 30 Sep 2023.

[14]Xin. "Cross-linking BioThings APIs through JSON-LD to facilitate knowledge exploration". *BMC Bioinformatics* 19 (2018). DOI: 10.1186/s12859-018-2041-5.

[15]Xiaoli Chen. "CERN Analysis Preservation: A Novel Digital Library Service to Enable Reusable and Reproducible Research". *Research and Advanced Technology for Digital Libraries*. Springer International Publishing, 2016, pp. 347–356.

[16]Friedemann. "Linked Data Architecture for Assistance and Traceability in Smart Manufacturing". *MATEC Web of Conferences* 304 (2019), p. 04006. DOI: 10.1051/matecconf/201930404006.

[17]*Performance Co-Pilot*. https://pcp.io/. Accessed on 30 Sep 2023.

[18]Vincent M. Weaver et al. "Measuring Energy and Power with PAPI". *2012 41st International Conference on Parallel Processing Workshops*. 2012, pp. 262–268. DOI: 10.1109/ICPPW.2012.39.

sampling rates with negligible overhead, without the code compilation and instrumentation. SuperTwin leverages PCP to offer a robust and full-fledged analysis framework capable of enabling the creation of digital twins.

Digital twins for HPC systems differ from those for other physical entities due to the abundance of sensors, with each sensor, such as a hardware register or PMU, capable of reporting thousands of metrics through re-programming. Treating processes as unique components further adds to the heterogeneity within the HPC system. DTDL (Digital Twins Definition Language), a derivation of JSON-LD, consists of six metamodel classes that explain the context of digital twin components. These classes encompass *Interface*, *Telemetry*, *Properties*, *Commands*, *Relationship*, and various data schemes. In DTDL, each *Interface* represents a standalone (sub)twin, encompassing descriptions of its *Properties*, *Telemetry*, and *Relationships*. SuperTwin combines these components to hierarchically model an HPC system's structure, considering each component (e.g., node, socket, CPU, GPU, memory subsystem, etc.) as a distinct digital twin. The notion that each interface stands as an individual (sub)twin is a core principle extensively leveraged in SuperTwin.

The Roofline Model,[19] and its numerous variations,[20] including the Cache-Aware Roofline Model (CARM),[21] have emerged as invaluable tools to evaluate the computational capabilities of contemporary processors and pinpointing potential performance limitations.[22] SuperTwin incorporates CARM due to its ability to accurately characterize the entire system by considering all memory levels. However, the current literature primarily relies on a single tool, adCARM,[23] for CARM generation, which is tailored for Intel architectures, leaving a gap in support for AMD systems. In this work, an extension is introduced to support AMD systems under the SuperTwin framework. Furthermore, this work also addresses another gap in the area of Roofline modeling in general; real-time CARM visualization during execution. SuperTwin introduces the novel tool, the *live-CARM panel*, which takes performance-counter data and automatically calculates CARM-related metrics, displaying them in conjunction with other metrics to give users an immediate idea of how their application performs relative to architectural limits. This panel is a prime example of what can be achieved by leveraging all the capabilities of SuperTwin.

[19]Nan Ding and Samuel Williams. "An Instruction Roofline Model for GPUs". *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2019, pp. 7–18. doi: 10.1109/PMBS49563.2019.00007.

[20]Tuomas Koskela et al. "A novel multi-level integrated roofline model approach for performance characterization". *High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings 33*. Springer. 2018, pp. 226–245; Jee Whan Choi et al. "A roofline model of energy". *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE. 2013, pp. 661–672; Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. "Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores". *IEEE Transactions on Computers* 66.1 (2016), pp. 52–58.

[21]Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. "Cache-aware roofline model: Upgrading the loft". *IEEE Computer Architecture Letters* 13.1 (2013), pp. 21–24.

[22]Douglas Doerfler et al. "Applying the roofline performance model to the intel xeon phi knights landing processor". *High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P^3MA, VHPC, WOPSSS, Frankfurt, Germany, June 19–23, 2016, Revised Selected Papers 31*. Springer. 2016, pp. 339–353; Didem Unat et al. "ExaSAT: An exascale co-design tool for performance modeling". *The International Journal of High Performance Computing Applications* 29.2 (2015), pp. 209–232. doi: 10.1177/1094342014568690. url: https://doi.org/10.1177/1094342014568690.

[23]Diogo Marques et al. "Application-driven cache-aware roofline model". *Future Generation Computer Systems* 107 (2020), pp. 257–273.

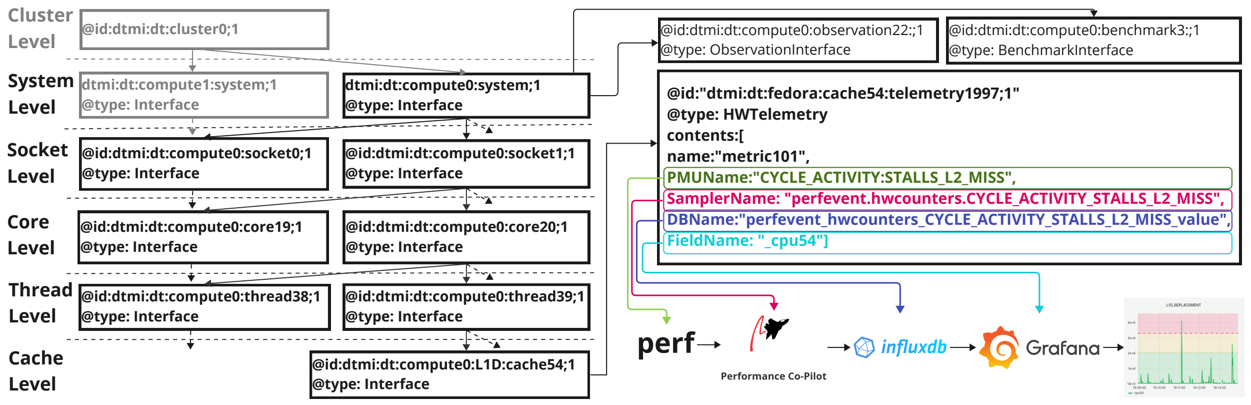

Cluster Level
@id:dtmi:dt:cluster0;1
System Level
dtmi:dt:compute1:system;1
@type: Interface
dtmi:dt:compute0:system;1
@type: Interface
Socket Level
@id:dtmi:dt:compute0:socket0;1
@type: Interface
@id:dtmi:dt:compute0:socket1;1
@type: Interface
Core Level
@id:dtmi:dt:compute0:core19;1
@type: Interface
@id:dtmi:dt:compute0:core20;1
@type: Interface
Thread Level
@id:dtmi:dt:compute0:thread38;1
@type: Interface
@id:dtmi:dt:compute0:thread39;1
@type: Interface
Cache Level
@id:dtmi:dt:compute0:L1D:cache54;1
@type: Interface
@id:dtmi:dt:compute0:observation22:;1
@type: ObservationInterface
@id:dtmi:dt:compute0:benchmark3:;1
@type: BenchmarkInterface
@id:"dtmi:dt:fedora:cache54:telemetry1997;1"
@type: HWTelemetry
contents:[
name:"metric101",
PMUName:"CYCLE_ACTIVITY:STALLS_L2_MISS",
SamplerName: "perfevent.hwcounters.CYCLE_ACTIVITY_STALLS_L2_MISS",
DBName:"perfevent_hwcounters_CYCLE_ACTIVITY_STALLS_L2_MISS_value",
FieldName: "_cpu54"]
perf
Performance Co-Pilot
influxdb
Grafana

Figure 1 *Knowledge Base of* SuperTwin*.*

3 SuperTwin: Digital Twins for HPC

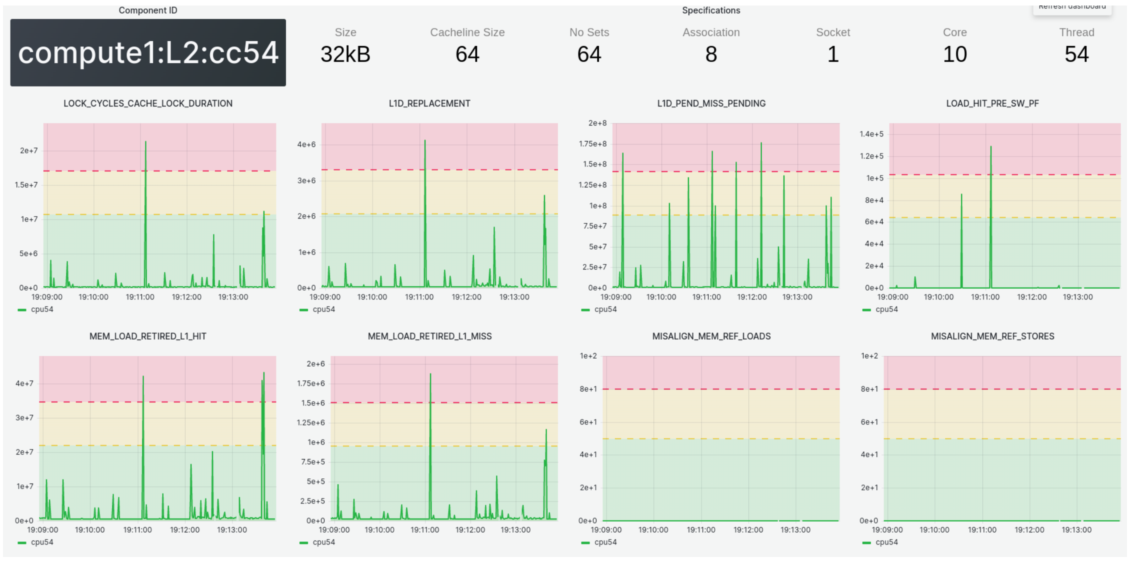
SuperTwin relies on a comprehensive knowledge base and linked-data capabilities. The Knowledge Base (KB), is used by each SuperTwin function as a parameter. It is dynamic, evolving to capture and link additional telemetry and metadata as they become available. This allows the twin to continue its operations in a live fashion without a procedural change and comprehend the factors influencing system performance in real-time. An example KB is shown in Fig. 1.

3.1 The Knowledge Base

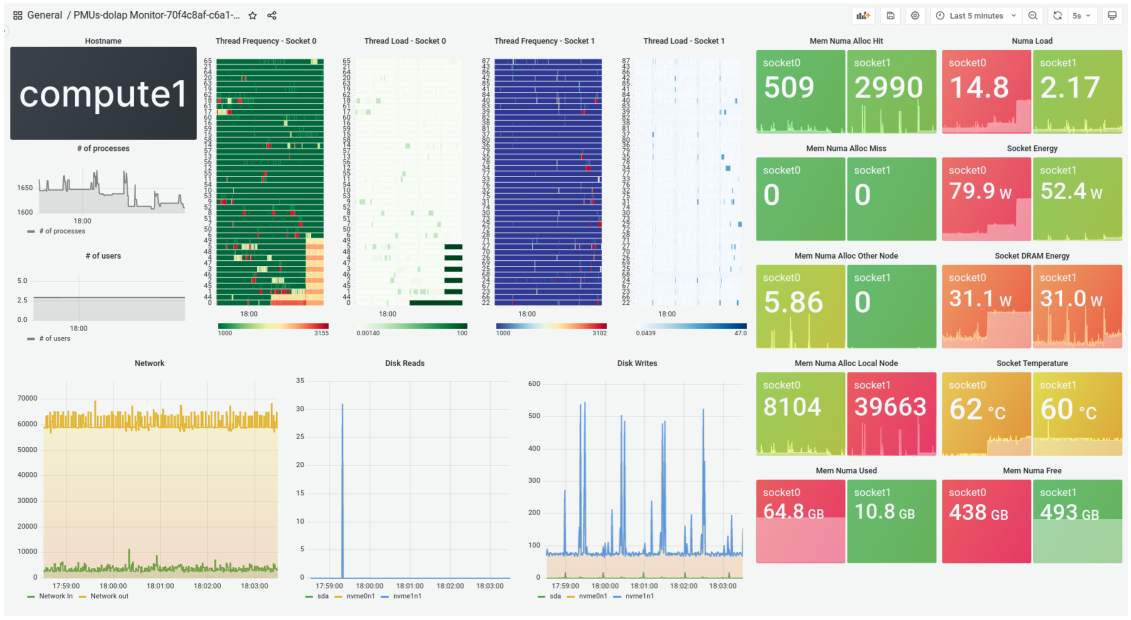
Capturing the target system and its component hierarchy, the KB can be parsed to acquire any information from topology to database parameters. There are two types of metrics to be sampled from an HPC system. The first type is *SWTelemetry*, i.e., software and system state-related metrics such as the number of processes, CPU, and memory load. These metrics are set to be *always sampled* with a low frequency. The second type is *HWTelemetry*, sampled from PMUs during kernel executions with high frequency. Sampling different metrics with varying frequencies yields a need for metadata associated with the host system's metadata. While time-series databases are tailored for telemetry data, they cannot keep much (linked) metadata. On the contrary, managing time-series data via a document database is impractical.[24] For this reason, SuperTwin's KB uses two types of databases with links between them. To this end, while InfluxDB stores the sampled SWTelemetry and HWTelemetry, MongoDB stores the knowledge base as JSON-LD extended with entries for each computation. To associate the computations with telemetry, pointers to InfluxDB are used to recall corresponding metrics.

Employing a tree-structured KB enables fully automated performance monitoring, anomaly detection and dashboards with meticulously selected metrics, tailoring various *views*. These views, namely (a) *Focus View*, (b) *Level View*, and (c) *Subtree View*, allow for a dynamic and versatile performance data exploration. Multiple views enable fine- and coarse-grain investigations into the component and system performance. Overall, SuperTwin can visualize data from different components and systems in tandem allowing for comprehensive analysis and comparison, further enriched by the inclusion of various views using Grafana visualization tool.

[24]Friedemann, "Linked Data Architecture for Assistance and Traceability in Smart Manufacturing"; Katarina Milenković. "Enabling Knowledge Management in Complex Industrial Processes Using Semantic Web Technology". English. *Proceedings of the 2019 International Conference on Theory and Applications in the Knowledge Economy*. 2019 International Conference on Theory and Applications in the Knowledge Economy, TAKE 2019 ; Conference date: 03-07-2019 Through 05-01-2020. 2019. URL: https://www.take-conference2019.com/.

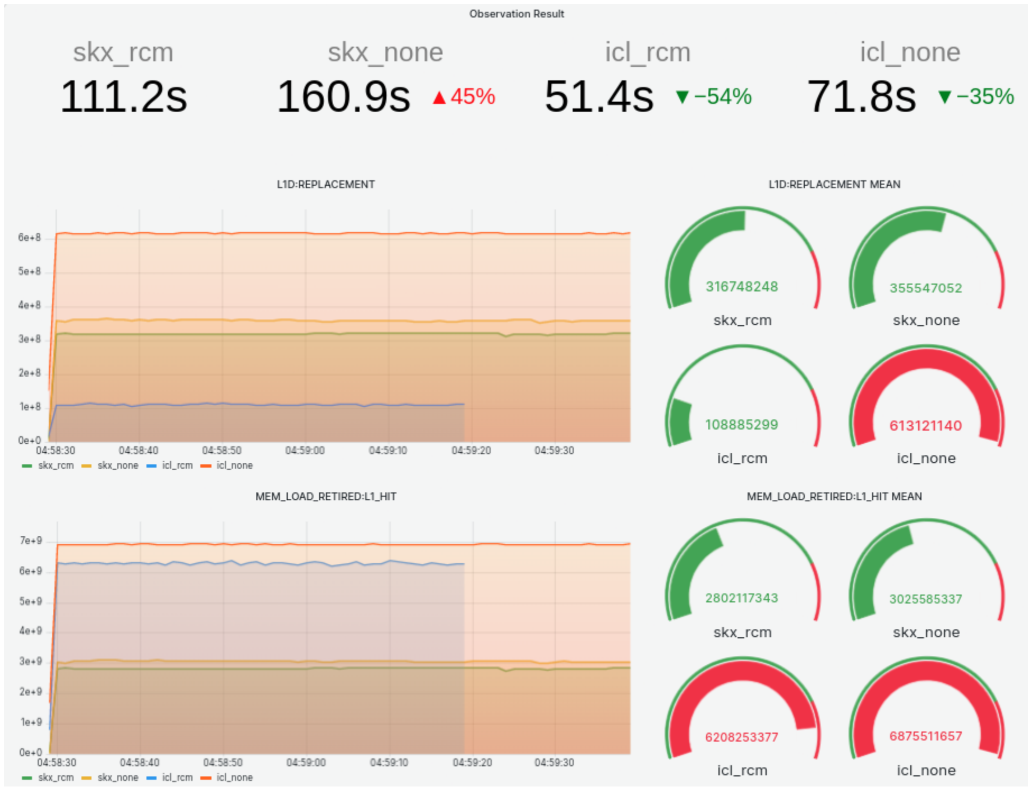


(a) *Focus view for an individual cache*

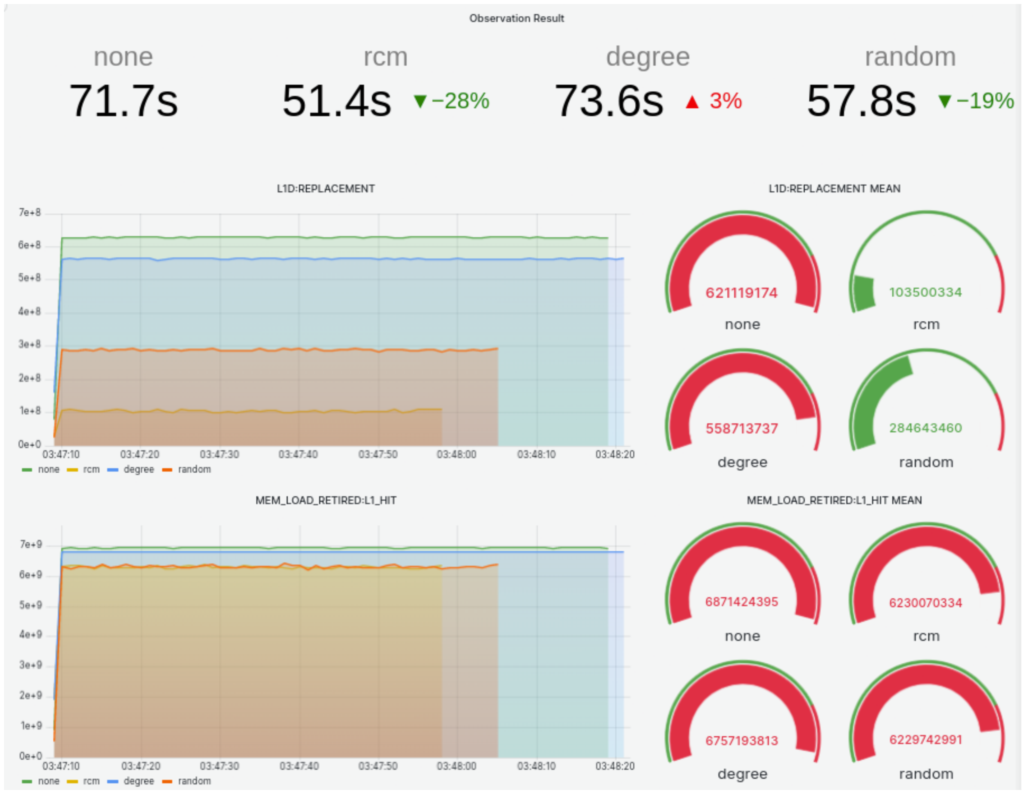


(b) *Subtree view for a node*

Figure 2 *Focus- and subtree-view dashboards, automatically generated by* SuperTwin .



(a) *Level-view for different matrix orderings.*



(b) *Level-view for different architectures.*

Figure 3 *Level-view dashboards, automatically generated by* SuperTwin *.*

- **The focus** (i.e., component) view offers a dashboard that visualizes active metrics from a single component, e.g., a socket, core, thread, network, disk, or process, providing a focused lens on individual element performance. This view can be extended to focus on the path from the root (whole system) to the focused component to investigate the root cause of anomalous behaviors or performance drawbacks. That is the path navigating from a component perspective to a more generalized system perspective is analyzed, aiding in tracing and isolating performance issues. An example focus-view dashboard is given in Fig. 2(a) for an individual cache.
- **The level** (i.e., type) view generates a dashboard that visualizes multiple instances from the same type, such as a group of threads, disks and even processes. This view allows the *isolation* of a single type, which corresponds to a level in the KB tree, treating them individually or in comparison to components within the same or a different system, whereas the linked-data capabilities enable the automatic visualization of component performance across different machines. For instance, the level-view dashboards for different processes running SpMV (each with a different reordering of the same matrix) and on two sockets running the SpMV code with two different orderings on the same matrix are given in Fig. 3(a) and Fig. 3(b), respectively.
- **The subtree** (i.e., (sub)system) view seeks to *zoom* into performance events, starting from an arbitrary node and extending to all connected leaf nodes, moving from a general perspective to a more specific one, i.e., from a single socket to all cores/caches. The detail intensifies as the path moves from the root (subsystem) to the leaf (components at the bottom of the KB hierarchy), facilitating a deeper dive into specific performance events and data. An example subtree-view dashboard for a single server is given in Fig. 2(d).

KB Lifecycle: The knowledge base is not a static object. It captures more about the system it represents as time passes by attaching new entries. To initialize the KB, SuperTwin uses its *probing tool*. To comprehensively capture the structural details of a system, including component specifications, inter/intra-relationships, and their associated performance metrics, a detailed probing is required. SuperTwin targets each hardware component that can be monitored, produce metrics or affect the overall system performance. Furthermore, it captures their relationships in a lightweight and adaptable fashion. SuperTwin's probing relies on widely available Linux tools to gather data. The system, network, and memory information are collected via `lshw`. The CPU, memory/cache topology metadata are collected by parsing `likwid-topology` from `likwid` tools[25] and `cpuid` instruction. When available, disk info is probed from `/sys/block/*/device` and `SMART`[26] utility. PMU information is collected with `libpfm4` library, which can recognize model-specific registers and their events of virtually every x86 and ARM processor available on the market. Upon probing available PMU metrics via `libpfm4`, and software telemetry via PCP, are filtered and mapped with the components.

In the initial KB, every single component that performs computation, communication, or I/O is represented with an `Interface`. Furthermore, each relationship among these components is encoded into these interfaces with a `Relationship`. The available metrics for the components are filtered and encoded as SWTelemetry and HWTelemetry. This makes precisely pinned executions and automated queries possible. To keep the KB dynamic and continuously link the system components to performance data, SuperTwin uses `Interfaces` and attaches their instances (i.e., entries) to KB. For instance, as mentioned above, processes are monitored via per-process kernel

[25]Thomas Röhl et al. "LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses". *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 2017, pp. 781–784. DOI: 10.1109/CLUSTER.2017.115.

[26]The Smartmontools Team. *Smartmontools*. Accessed on 5th October 2023. URL: https://www.smartmontools.org/.

metrics. JSON-LD interfaces are serialized with given parameters into a run-time object. Except for the `ProcessInterfaces`, all classes/interfaces have their values assigned as constants during the generation phase. In contrast, a `ProcessInterface` is re-instantiated each time it is invoked, reflecting the dynamic nature of processes. For performance events, SuperTwin has two other interface classes:

- `BenchmarkInterface`, and `BenchmarkResult` as a helper class, is designed to record benchmark results. SuperTwin is able to perform *Cache Aware Roofline Model* (CARM), *STREAM*[27] and *High Performance Conjugate Gradient*[28] (HPCG) benchmarks homogeneously using the `BenchmarkInterface`. It contains the source codes of these benchmarks in its codebase and similar to the probing phase, it first copies these codes to the target system. If required, based on the information in KB, SuperTwin first compiles the benchmarks on the target system using the benchmark's preferred compiler if it exists, e.g., `icc` or `gcc`. After the benchmark, SuperTwin parses the results and creates a `BenchmarkInterface` with the corresponding `BenchmarkResult`.
- `ObservationInterface` entries encode sampled hardware performance events and system metrics, executed commands, generated affinity, time and other relevant metadata. Using the parameters in KB, queries can be generated to automatically retrieve data through these entries. A basic `ObservationInterface` entry is shown in Listing 1. The queries automatically generated by SuperTwin to analyze the `BenchmarkEntry` in Listing 1 are given in Listing 2.

Performance DB: For long-term data management, thanks to its modular design, SuperTwin operates a *global performance database*, SuPerDB. Unlike local instances, SuPerDB employs cloud instances of MongoDB and InfluxDB. With a global performance database, SuperTwin aims to accumulate performance metrics from a wide array of systems to enhance architectural research and train robust machine learning models, particularly leveraging Large Language Models (LLMs) which can exploit the rich metadata collected to be trained as an assistant for performance engineering. The users of SuperTwin have the option to report their performance telemetry readings and the system's knowledge base to the performance database, alongside their local instances.

```
{
  "@type": "ObservationInterface",
  "@id": "278e26c2-3fd3-45e4-862b-5646dc9e7aa0",
  "displayName": "rcm_rma10_mt",
  "time": 48.667,
  "command": "./spmv -f rma10.mtx -o rcm -t 4",
  "modifier": "likwid-pin -q -c S0:0-1@S1:0-1",
  "no_threads": 4,
  "involved_threads": [0,1,22,23],
  "sampled_sw_metrics": ["kernel.percpu.cpu.idle", "mem.numa.alloc.hit", "mem.numa.alloc.
      miss"],
  "sampled_hw_metrics": ["RAPL_ENERGY_PKG", "INSTRUCTION_RETIRED", "FP_ARITH:
      SCALAR_DOUBLE", "MEM_LOAD_RETIRED:L1_HIT"],
  "dashboard": "http://localhost:3000/d/-PiOFZEVz/pmus-278e26c2-3fd3-45e4-862b-5646
      dc9e7aa0?time=1681499308500&time.window=17000"
}
```

Listing 1: An example `ObservationInterface` entry which is used to retrieve sampled metrics. A report is generated on the fly and added to the entry before appending to KB.

[27] John McCalpin. "Memory bandwidth and machine balance in high performance computers". *IEEE Technical Committee on Computer Architecture Newsletter* (1995), pp. 19–25.

[28] Jack Dongarra and Michael A Heroux. "Toward a new metric for ranking high performance computing systems". *Sandia Report, SAND2013-4744* 312 (2013), p. 150.

```
SELECT "_cpu0", "_cpu1", "_cpu22", "_cpu23" FROM "kernel_percpu_cpu_idle" WHERE tag="278
    e26c2-3fd3-45e4-862b-5646dc9e7aa0"
SELECT "_node0", "_node1" FROM "mem_numa_alloc_hit" WHERE tag="278e26c2-3fd3-45e4-862b
    -5646dc9e7aa0"
SELECT "_cpu0", "_cpu1", "_cpu22", "_cpu23" FROM "
    perfevent_hwcounters_fp_arith_scalar_double" WHERE tag="278e26c2-3fd3-45e4-862b-5646
    dc9e7aa0"
SELECT "_node0", "_node1" FROM "perfevent_hwcounters_RAPL_ENERGY_PKG" WHERE tag="278e26c2
    -3fd3-45e4-862b-5646dc9e7aa0"
```

Listing 2: Queries automatically generated by SuperTwin for the `BenchmarkInterface` entry given in Listing 1.

In SuPerDB, the `ObservationInterface` of SuperTwin evolves into two versions within the performance database context: `TS ObservationInterface` and `AGGObservationInterface`, where the latter statistically summarizes data using various aggregations, e.g., min, max, mean, to manage high data volumes. The users require a local SuperTwin instance to access SuPerDB, visualize performance data, and automatically generate dashboards and reports. Without SuperTwin, they can only download selected data for ML training. Future adaptations may include appending source code and binary executables to the collected metadata, facilitating the training of models that can optimize code and predict performance and potential inefficiencies.

3.2 ADDING COMPUTE DEVICES TO SUPERTWIN

The integration of a computing device, i.e., FPGA, GPU, etc., into the KB is handled similarly to other hardware components within a system. Initially, an in-depth probing of the target devices is done using widely available tools. For instance, in the case of Nvidia GPUs, this investigation uses `nvidia-smi` to find available GPUs, their models, bus and process information. `/sys/class/drm/` is used for NUMA location, and `DeviceQuery` for the hardware specifications such as the number of SMs, shared memory, and cache sizes. The latest GPUs lack the capability for real-time hardware telemetry reporting without modifications to the source code. To address this, we have employed `pcp-pmda-nvidia` for collecting `SWTelemetry`, essentially capturing every metric supported by NVML. Regarding `HWTelemetry`, we leveraged the approach used in benchmark executions. SuperTwin is tasked with creating a wrapper script for initiating the kernel launch and configuring `ncu` to record hardware performance events during runtime. Following the completion of these executions, SuperTwin analyzes the output from `ncu`, integrating these comprehensive performance metrics into the KB through the `ObservationInterface`. An example for (a subset of) an `Interface` encoding a GPU device in KB is given in Listing 3

```

"dtmi:dt:cn1:gpu0;1": {
    "@type": "Interface",
    "@id": "dtmi:dt:cn1:gpu0;1",
    "@context": "dtmi:dtdl:context;2",
    "contents": [
      {
        "@id": "dtmi:dt:cn1:gpu0:property0;1",
        "@type": "Property",
        "name": "model",
        "description": "NVIDIA Quadro GV100"
      },
      {
        "@id": "dtmi:dt:cn1:gpu0:property1;1",
        "@type": "Property",
        "name": "memory",
        "description": "34359 Mb"
      },
      {
        "@id": "dtmi:dt:cn1:gpu0:property12;1",
        "@type": "Property",
        "name": "numa node",
        "description": 0
      },
      {
        "@id": "dtmi:dt:cn1:gpu0:telemetry1337;1",
        "@type": "SWTelemetry",
        "name": "metric4",
        "SamplerName": "nvidia.memused",
        "DBName": "nvidia_memused",
        "fieldName": "_gpu0",
      },
      {
        "@id": "dtmi:dt:cn1:gpu0:telemetry1404;1",
        "@type": "HWTelemetry",
        "name": "metric137",
        "PMUName": "ncu",
        "SamplerName": "gpu__compute_memory_access
        _throughput",
        "DBName": "ncu_gpu__compute_memory_access
        _throughput",
        "FieldName": "_gpu0",
        "description": "Compute Memory Pipeline :
        throughput of internal activity within
        caches and DRAM",
      }}
```

Listing 3: An example `GPU Interface` entry which is used to monitor GPU devices on the system and profile kernel executions.

4 THE MECHANICS OF SUPERTWIN

SuperTwin is designed to run on a *host* that can be different than the *target* system. The host runs the SuperTwin daemon as well as the tools with heavy workloads, e.g., InfluxDB, MongoDB, and Grafana. The target only runs the PCP samplers and reports telemetry to the host when requested. In Figure 4, step ⓪ reads the environment variables such as the IP addresses of InfluxDB and MongoDB instances and Grafana token to the SuperTwin daemon. In step ①, the probing module is copied to the target system to generate a JSON file containing the system information which, in ②, is copied back to the host to generate the KB. The information collected from all the tools, components, and third-party tools SuperTwin manages is fused for KB generation. Once the KB is generated, it is inserted into MongoDB in step ③. Step ③ re-occurs every time KB changes or SuperTwin is restarted. When this phase is completed, the framework becomes fully functional using only this data structure.

In Figure 4, two SuperTwin scenarios are shown; the first is sampling software emitted metrics to monitor system state (Scenario A), and the other is capturing the hardware performance events during kernel execution. In step (A1), using KB, SuperTwin configures the PCP collectors and samples system-related metrics, such as CPU and memory usage, NUMA-related events, and energy spent. In (A3), a sampler on the target is requested for this telemetry. Since the query parameters are already encoded in KB, steps (A1) and (A2) can happen at the same time. That is the dashboards are already generated on the host when the target starts reporting.

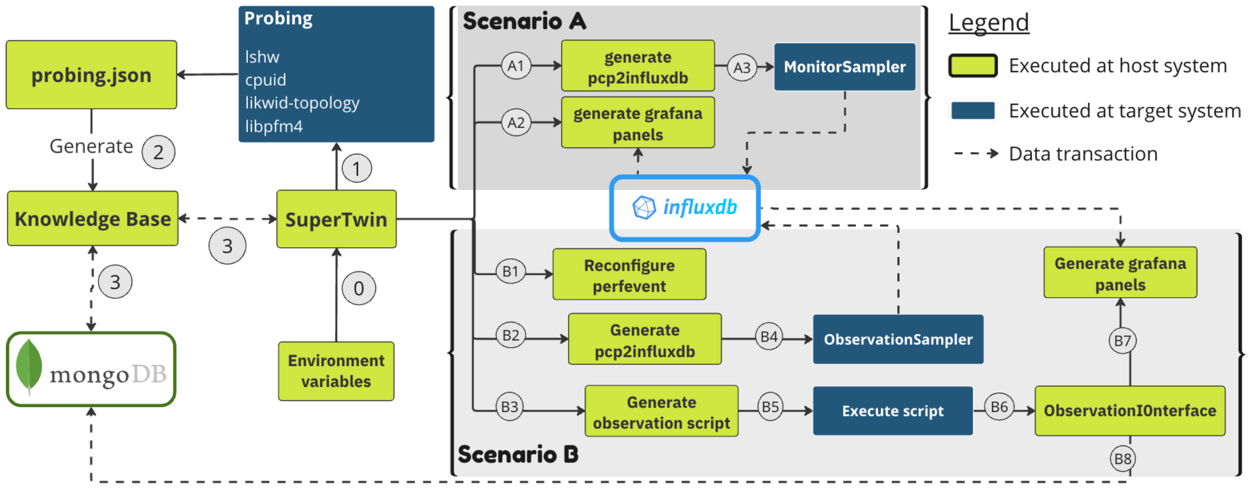


Figure 4 *Two scenarios within the* SuperTwin *framework*

In Scenario B, SuperTwin samples hardware events reported from the PMUs. In this case, it focuses on an execution on the target and the components on which the execution takes place. Therefore, SuperTwin requests an executable and its command-line parameters. Once these are provided, the PMUs are configured to report the requested metrics in step (B1). That is SuperTwin configures the sampler in the same way as step (A1). After the PMUs are configured, it generates a script to run the requested kernel on the target system. This script bounds the threads to the cores using one of the *balanced*, *compact*, *numa balanced*, *numa compact* strategies based on the probed target system topology. Then it samples performance events, executes the script to run a kernel on a target and stops the sampling as the kernel is halted. An `ObservationInterface` is generated to encode the execution metadata, collected metrics and the unique observation ID associated with the time-series data in InfluxDB. In step (B8), the `ObservationInterface` is appended to the system's KB. This `ObservationInterface` entry is later used to recall the performance data for visualization or analysis purposes.

4.1 ABSTRACTION LAYER

To perform its actions and to effectively monitor PMU events on diverse target systems, each hosting CPUs across various vendors and micro-architectures, SuperTwin leverages an *Abstraction Layer*. The monitoring units and their reported events can significantly vary among different micro-architectures and from vendor to vendor. For instance, Intel has four general-purpose programmable counters/per-core to count performance events (eight if is not shared with a second thread in the core), whereas AMD has two internal counters, one for each sampling flag. Intel provides 62 sub-events corresponding to 12 events, each accompanied by mask values. Similarly, AMD offers support for events similar to Intel. As an example, similarities and differences of events for Intel Cascade and AMD Zen3 are listed in Table 1. A detailed comparison between Intel and AMD PMUs can be found in.[29]

Event	Intel Cascade	AMD Zen3
Energy	RAPL_ENERGY_PKG RAPL_ENERGY_DRAM	RAPL_ENERGY_PKG RAPL_ENERGY_DRAM
Retired Inst.	INSTRUCTIONS_RETIRED	RETIRED_INSTRUCTIONS
Tot. Mem. Op.	MEM_INST_RETIRED:ALL_LOADS + MEM_INST_RETIRED:ALL_STORES	LS_DISPATCH:STORE_DISPATCH+ LS_DISPATCH:LD_DISPATCH
L3 Hit	**Not Supported**	LONGEST_LAT_CACHE:MISS + LONGEST_LAT_CACHE:RETIRED

Table 1 *Intel vs. AMD PMU events: the same, similar, different, and exclusive event names for the same generic event, respectively.*

To facilitate the monitoring of PMU events in a platform-agnostic manner, an abstraction layer is implemented for SuperTwin. This layer effectively maps generic event names to concealed hardware-specific PMU event names, enhancing the system's versatility and ease of use. We have established a set of common events, such as L1_CACHE_DATA_MISS, FP_DIV_RETIRED, and RAPL_ENERGY_ PKG, that are *assumed to be* supported by all the commodity CPUs. The rest of the events are left to the user's discretion. For further flexibility and scalability, SuperTwin utilizes configuration files to establish a straightforward mapping of common events to corresponding hardware events. The structure of a configuration file is as follows:

```
[pmu_name | alias]
<generic_event>:<hardware_event_1> [op]
[op] : ((+|-|*|/)(<hw_event> | <const>)) [op]
```

Following the pattern delineated, it is possible to generate a configuration file for "any" hardware by specifying the events intended for monitoring. Upon registering the desired configuration files within SuperTwin, the application proceeds to configure the PCP of the target system using the registered configuration files when needed. Additionally, users can access event information in a CPU agnostic manner within the program using `pmu_util.get(...)` method. An example is given below;

```
>pmu_utils.get(HW_PMU_NAME, COMMON_EVENT_NAME)
>pmu_utils.get("skl", "TOTAL_MEMORY_OPERATIONS")
>[
  "MEM_INST_RETIRED:ALL_LOADS",
  "+",
  "MEM_INST_RETIRED:ALL_STORES"
 ]
```

Although this example belongs to the Intel CPU outlined in Table 2, SuperTwin's configuration mapping via its abstraction layer offers versatility. Users can create mapping files for a wide range of CPUs, including Intel, AMD, PowerPC, ARM, and others, as long as they are supported by the *libpfm4* library which is the core library that enables PCP to monitor PMU events in CPUs. As SuperTwin configures PCP on the target, it creates empty and *zero-overhead* dashboards

[29] Muhammad Aditya Sasongko et al. "Precise Event Sampling on AMD Versus Intel: Quantitative and Qualitative Comparison". *IEEE Transactions on Parallel and Distributed Systems* 34.5 (2023), pp. 1594–1608. ISSN: 1558-2183. DOI: 10.1109/TPDS.2023.3257105.

on Grafana, which are simply JSON files. Last, but not least, the abstraction layer seamlessly generates the formulas for the events the user is interested in. This changes from vendor to vendor as well as for every architecture even when the events are the same. An abstraction layer is necessary in modern tools to handle this diversity for performance profiling. An illustrative use case is presented in Section 5.4.

4.2 CACHE-AWARE ROOFLINE MODEL IN SUPERTWIN

4.2.1 MODEL CONSTRUCTION

For an intuitive visualization framework, SuperTwin supports the construction of a tailored CARM model for Intel and AMD microarchitectures. It is enriched with a set of custom microbenchmarks in x86 assembly, designed to experimentally assess the realistically attainable maximum performance of a given system, i.e., the sustainable bandwidth for different levels of memory hierarchy and the peak throughput of computational units.

In order to assess the different metrics necessary to construct the CARM roofs, such as bandwidth and peak flops, we rely on the Time Stamp Counter (TSC) to measure the number of clock cycles elapsed, detection of CPU operating frequency, and predefined amount of memory and compute operations contained in a specific microbenchmark executed. The microbenchmarks support various instruction set architecture (ISA) extensions, including scalar, SSE, AVX2 and AVX512, along with multithreaded measurements. This allows for further customization of SuperTwin's CARM plot based on the prevalent ISA extension or a specific thread count utilized in the tested applications.

Thanks to Knowledge Base, CARM microbenchmarks are automatically configured for a target system, taking into account cache sizes and available ISAs. To reduce the overheads associated with extensive benchmarking of all possible combinations of thread counts, SuperTwin generates a subset of the most representative thread counts for the microbenchmark executions. Finally, the KB is also used to store all the microbenchmarking results for each tested system, thus allowing for a re-construction of the CARM plot without the need to re-run all the microbenchmarks.

4.2.2 APPLICATION CHARACTERIZATION

Besides the construction of a CARM plot for a target system, SuperTwin also provides the CARM-based visualization of the application execution progress at run-time (live monitoring feature). This functionality is achieved by automatically configuring PMU events based on the underlying architecture of a system, in order to accurately calculate the live Arithmetic Intensity (AI) and live-GFLOPS of the system. These PMU-based metrics are sampled on a time-stamp basis and used to plot the application points in real time on the generated CARM for the target system. This generated panel is referred to in the framework as the live-CARM panel, which offers a unique feature of SuperTwin by delivering real-time feedback on a target system's utilization relative to architectural constraints determined by the already constructed CARM. This dynamic functionality is achieved through the formulation of specialized expressions based on hardware events, enabling the calculation of GFLOPS and Arithmetic Intensity (AI) tailored to diverse Intel and AMD microarchitectures.

The amount of GFLOPS is determined by mapping and adding all of the available floating-point operation events of the target system, using the PMU remapping capabilities of SuperTwin. As for the AI, this metric requires the already calculated GFLOPS, as well as the total amount of memory bytes transferred to/from the processing cores, which calculation varies across different generations of Intel and AMD systems. In general, they are inferred from the ratios of different floating point instructions (scalar, SSE, AVX2, AVX512), which are applied to the total amount of store and load events measured in the target system.

The live-CARM panel also automatically retrieves the micro-benchmarking results (to construct the CARM plot of the target system) from the Knowledge Base. By tightly coupling the application's live metrics with the CARM plot in the SuperTwin panel, we facilitate the observation of the relative performance of an application in real-time, when compared to the theoretical limits of the architecture it is running on. Furthermore, SuperTwin auto-generates graphs for any hardware metric configured by the user, which display the values of selected metrics for the different cores of the target machine, including a cumulative sum of the events across all cores.

5 EXPERIMENTAL RESULTS

In the host system, we used Grafana v9.4.7, InfluxDB 1.8, MongoDB 6.0.6. For micro-benchmarks, we used `likwid-bench` v5.2.2. Specifications of the target systems used in the experimental setting are presented in Table 2.

SKX		***ICL***	
OS	Ubuntu 20.04.3 LTS x86_64	OS	Linux Mint 21.1 x86_64
Kernel	5.15.0-73-generic	Kernel	5.15.0-56-generic
CPU	Intel Xeon Gold 6152 @3.7GHz x2 (44c/88t)	CPU	Intel i9-11900K @5.1GHz (8c/16t)
Arch	Skylake X	Arch	Ice Lake
Mem	1TB DDR4 @ 2666MHz	Mem	64GB DDR4 @ 2133MHz
Env.	pcp 5.3.6-1	Env.	pcp 5.3.6-1
CSL		***ZEN3***	
OS	CentOS Linux release 7.9.2009 (Core) x86_64	OS	Ubuntu 22.04.3 LTS x86_64
Kernel	3.10.0-1160.90.1.el7.x86_64	Kernel	6.2.0-33-generic
CPU	Intel Xeon Gold 6258R @2.7GHz (28c/56t)	CPU	AMD EPYC 7313 @3GHz (16c/32t)
Arch	Cascade Lake	Arch	Zen3
Mem	64GB DDR4 @ 3200 MHz	Mem	128GB DDR4 @ 2933 MHz
Env.	pcp 6.1.0-1	Env.	pcp 5.3.6-1

Table 2 *Specifications of platforms used in the experiments.*

5.1 THROUGHPUT AND ACCURACY

PCP performs sampling instead of recording performance events over time and reports the sum at the end. There is no buffer or queue mechanism to keep data points until their insertion into the DB. This theoretically can cause losses in data points, especially with high-frequency samplings if the DB insertion time is slower than the sampling time. Moreover, the sampled metrics are reported over a network, which presents another bottleneck to database throughput. We performed throughput experiments with high-frequency samplings using PCP and InfluxDB to ensure that sampled data points do not suffer heavy losses while they are inserted into the DB and determine an ideal sampling frequency to set for our framework.

Table 3 shows the throughput achieved with `pmdaperfevent`. Instead of sampling and reporting of operating system files, `pmda perfevent` samples PMUs, which may represent another limiting factor for reaching maximum throughput in high frequencies. As expected, we observed significant variation in losses. Besides the missing values, we also observed batched zero values in our DB when the frequency was high. Therefore, we also count the number of zeros inserted

Host	Freq.	#mt	Expected	Inserted	Zeros	%L	**%L+Z**	Tput	A.Tput
skx	2	4	7.04E+03	6.62E+03	0.00E+00	6.0	**6.0**	661.8	661.8
	2	5	8.80E+03	8.71E+03	0.00E+00	1.0	**1.0**	871.2	871.2
	2	6	1.06E+04	1.06E+04	0.00E+00	0.0	**0.0**	1056.0	1056.0
	8	4	2.82E+04	2.60E+04	5.84E+02	7.8	**9.8**	2597.8	2539.4
	8	5	3.52E+04	3.42E+04	7.72E+01	2.8	**3.0**	3423.2	3415.5
	8	6	4.22E+04	4.22E+04	0.00E+00	0.0	**0.0**	4224.0	4224.0
	32	4	1.13E+05	6.97E+04	3.04E+04	38.1	**65.1**	6969.6	3927.9
	32	5	1.41E+05	1.14E+05	5.32E+04	19.4	**57.2**	11352.0	6030.3
	32	6	1.69E+05	1.20E+05	5.02E+04	28.8	**58.5**	12027.8	7012.1

Host	Freq.	#mt	Expected	Inserted	Zeros	%L	**%L+Z**	Tput	A.Tput
icl	2	4	1.28E+03	1.25E+03	0.00E+00	2.0	**2.0**	125.4	125.4
	2	5	1.60E+03	1.60E+03	0.00E+00	0.0	**0.0**	160.0	160.0
	2	6	1.92E+03	1.92E+03	0.00E+00	0.0	**0.0**	192.0	192.0
	8	4	5.12E+03	4.97E+03	0.00E+00	3.0	**3.0**	496.6	496.6
	8	5	6.40E+03	6.22E+03	0.00E+00	2.8	**2.8**	622.4	622.4
	8	6	7.68E+03	7.68E+03	0.00E+00	0.0	**0.0**	768.0	768.0
	32	4	2.05E+04	2.00E+04	7.26E+03	2.2	**37.6**	2003.2	1277.6
	32	5	2.56E+04	2.50E+04	8.78E+03	2.4	**36.7**	2499.2	1621.2
	32	6	3.07E+04	3.00E+04	1.04E+04	2.3	**36.0**	3002.9	1965.9

Table 3 *Number of data points expected and observed at the host DB w.r.t. #metrics and sampling frequency.* (A.) Tput *inserted (actual) data points per second;* L%+Z *ratio of false zeros subtracted from inserted values to the expected value.*

into the DB. With `perfevent`, we sampled metrics that are highly unlikely to report zero, e.g., `UNHALTED_CORE_CYCLES`, `INSTRUCTION_RETIRED`, `UOPS_DISPATCHED` etc. Although, losses with relatively low frequencies are negligible, more than half of the data points are lost in transmission on `skx` and a third of data points are lost on `icl`. This is due to the correlation between the loss amount and the size of the domain; `skx` has 88 threads, therefore there are 88 data points in each report while this number is 16 for `icl`.

To verify the accuracy of PCP's `perf` readings while generating our performance models, we used `likwid-bench`,[30] which executes a pre-determined, fixed number of instruction streams and can report ground truth for performance events that took place afterwards. We executed kernels `sum`, `stream`, `triad`, `peakflops`, `ddot`, `daxpy` with SuperTwin while sampling performance, later parsed output of the `likwid-bench` kernels and compared with our readings. The relative errors acquired w.r.t. averaged kernel errors for different frequencies are reported in Figure 5 (positive and negative values represent overcounting and undercounting, respectively). The data volume (in bytes) is calculated as `(MEM_UOPS:LOADS + MEM_UOPS:STORES)` $\times$ 8 on `zen3` and `(MEM_INST_RETIRED:LOADS + MEM_INST_RETIRED:STORES)` on `skx` and `icl`. The #FLOPS is calculated as `RETIRED_SSE_AVX_FLOPS: ANY` on `zen3` and `FP_ARITH:SCALAR_DOUBLE` on `icl` and

[30] Thomas Röhl et al. "Overhead Analysis of Performance Counter Measurements". *2014 43rd International Conference on Parallel Processing Workshops*. 2014, pp. 176–185. DOI: 10.1109/ICPPW.2014.34.

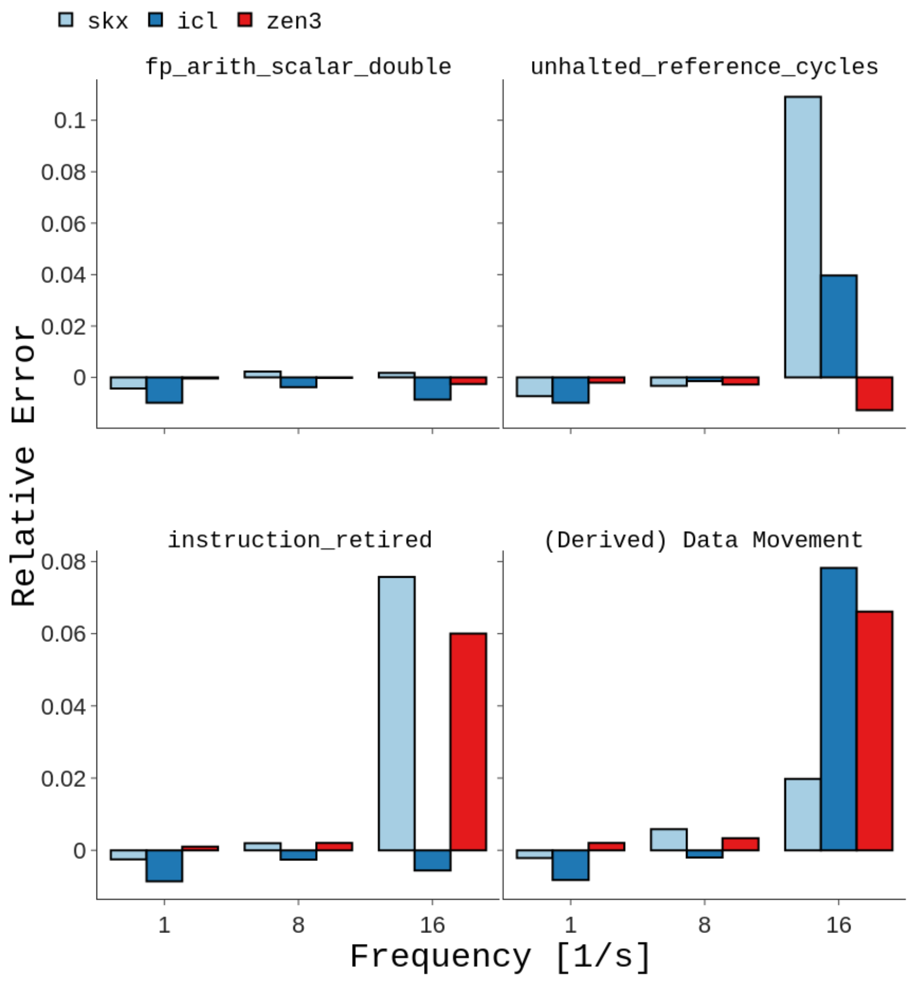


Figure 5 *Relative errors between sampled metrics and `likwid-bench` reported values.*

`skx`. We found that the measurements are accurate enough to profile executions and generate coherent performance models (e.g., live-CARM). The increased error rates may be due to losses in transmission, or the inherent noise in PMUs.[31]

5.2 RESOURCE USAGE OF SUPERTWIN

PCP employs multiple agents for metric shipment operations, and as the number of metrics and resolutions increase, remote system resource usage becomes a concern. We measured CPU and memory usage of individual PCP agents for various metric and sampling configurations on a high-capacity server (`skx`) with 2 sockets, 88 threads, 1 TB of RAM, and 4 disks. We conducted measurements over 10 minutes on an empty target system and averaged the results. Figure 6 shows results for sampling 50 metrics, comprising 15,937 data points at varying frequencies. The I/O use of PCP agents was negligible (< 1 KB) and excluded from the results. The host system had a 100Mbit cabled connection with the target system, whereas the disk performance was measured at 182 KB/s and 1.2 MB/s for 512B and 8K block-sized writes, respectively.

The PCP agents include `pmcd`, which manages other agents and reports their readings;

[31]Vincent Weaver, Dan Terpstra, and Shirley Moore. "Non-determinism and overcount on modern hardware performance counter implementations". 2013, pp. 215–224. DOI: 10.1109/ISPASS.2013.6557172.

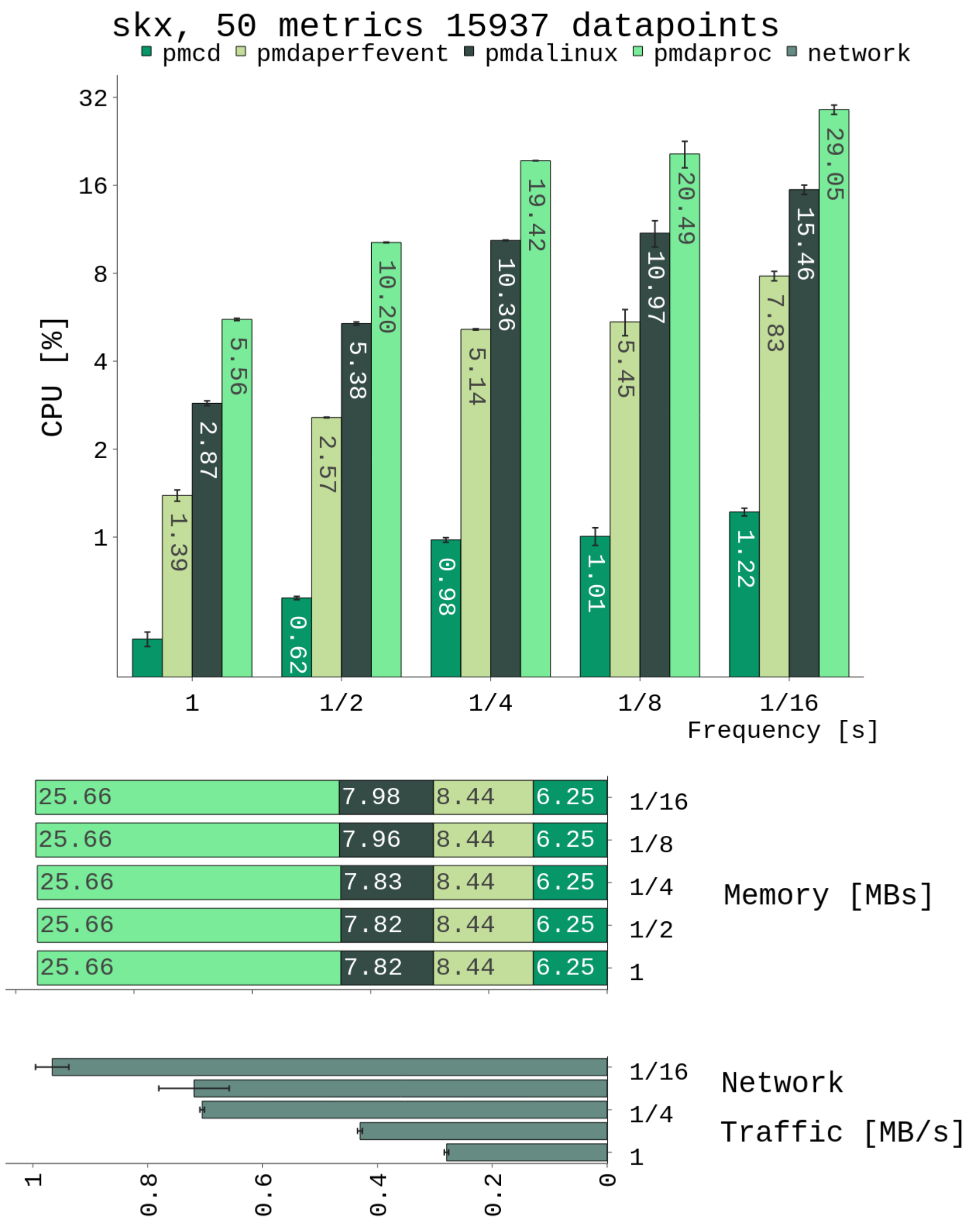


Figure 6 *System resource usage of metric shipment with kernel and PMU metrics on* skx.

perfevent, which samples PMU readings via Linux perf interface; pmdalinux, reporting software-sourced system state metrics like memory usage; and pmdaproc, which reports per-process metrics like I/O and memory usage. CPU usage measurements use the proc.psinfo.utime and proc.psinfo.stime, whereas memory measurements use the proc.psinfo.rss metric. Notably, regardless of the reported metrics or sampling frequency, all agents maintain constant memory usage. pmdaproc uses more memory due to a larger instance domain. Except for pmdaproc, all agents are efficient in resource usage. Overall, SuperTwin employs 0 per-process metrics and uses approximately 20 pmdalinux metrics, and 2 pmdaperfevent metrics at 1-second intervals.

CPU and network usage scale linearly with increased sampling frequency, showing consistent resource usage without deviations as seen in the error bars. However, one case in Figure 6

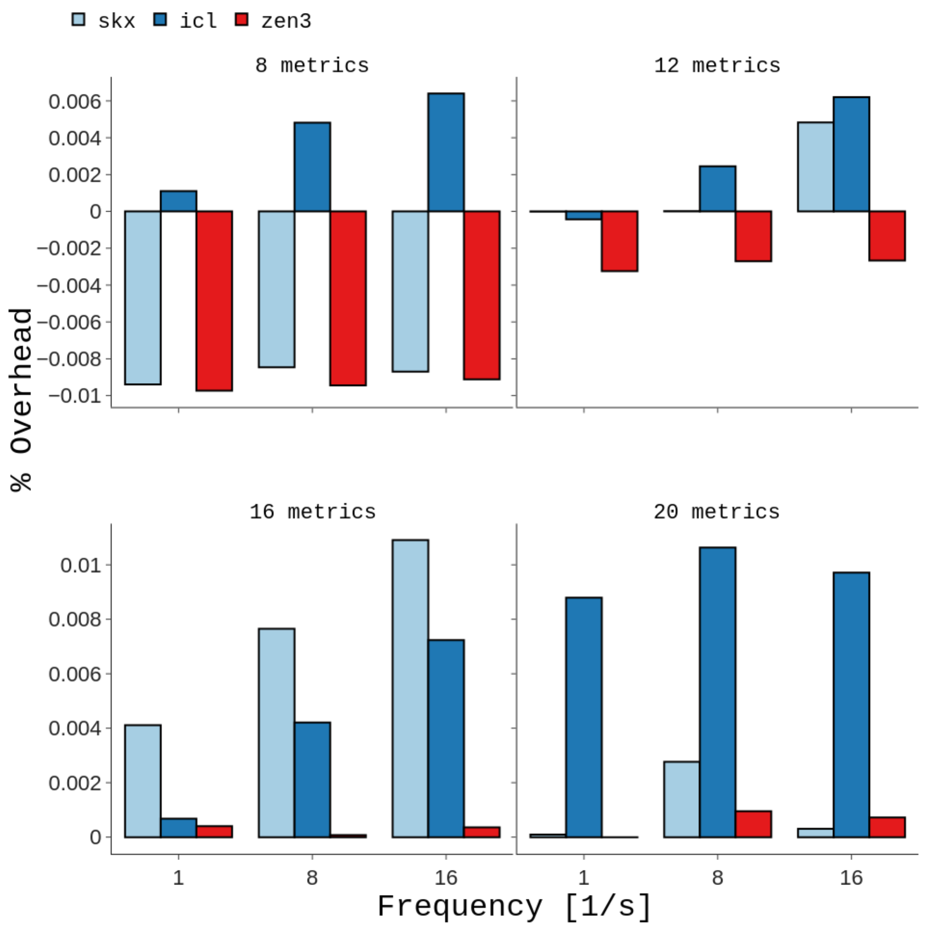


Figure 7 *Overhead caused by profiling six `likwid-bench` kernels (executions repeated 5 times, the run-times averaged).*

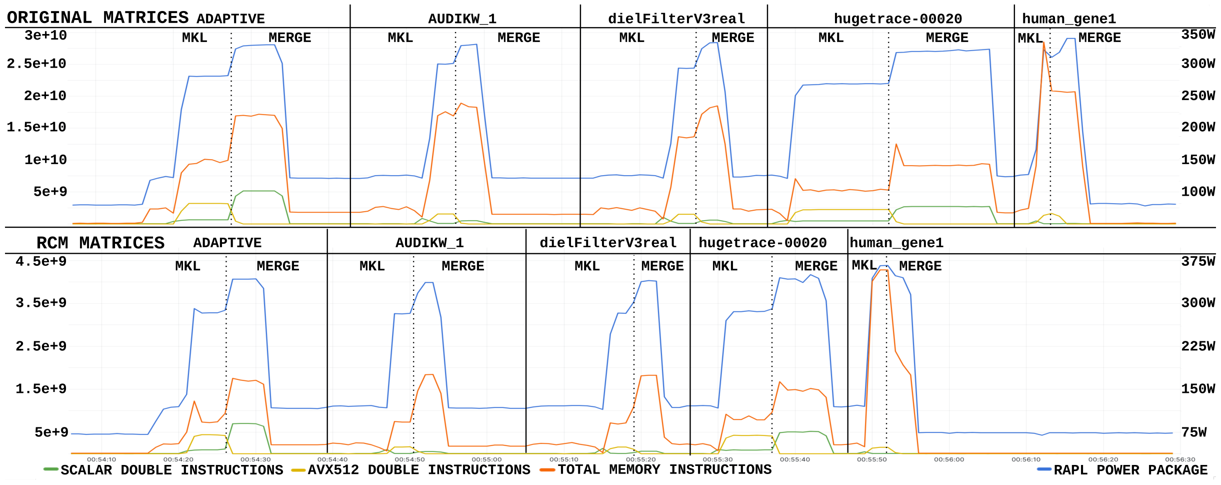


Figure 8 *Monitoring live performance events during SpMV execution on Intel CSL system*

reveals that the PCP framework does not scale perfectly for 4 and 8 reports per second, with varying network traffic. This behavior is observed in other skx measurements except for 10 metrics. The under-utilization of the network and CPU suggests that the framework may stall and fail to sample and report metrics as desired due to a lack of buffering and resending missing metrics. High-frequency sampling exacerbates this issue, leading to outdated or lost metrics during transmission, consistent with previous observations.

5.3 TIME OVERHEAD

During the hardware performance event samplings, both PCP run on the target system and performance monitoring registers are sampled. Therefore kernel run-time may be affected negatively. To measure the effect of sampling on a target system, we ran the same micro-benchmarks from previous tests with and without sampling and measured the change in their completion times. The overhead caused by sampling can be seen in Figure 7. Surprisingly, negative overheads are observed, which we explain as overhead added by sampling is smaller than the variance observed between different runs of the same kernel. This is understandable since the positive overheads are also measured at 0.01%. A similar negative overhead is also reported by[32] even in a much bigger distributed setting. However, a meaningful skew towards positive overhead is observed with increasing frequency.

5.4 MONITORING LIVE PERFORMANCE EVENTS

To showcase the live monitoring capabilities of SuperTwin, we execute two state-of-the-art algorithms for Sparse Matrix Vector Multiplication (SpMV), i.e., Intel MKL[33] and Merge,[34] on the Intel CSL system presented in Table 2. We selected five sparse matrices from the SuiteSparse collection,[35] as presented in Table 4, which cover a range of matrices from different scientific domains, characteristics, dimensions, and number of non-zero elements. Both SpMV algorithms are performed on the original (unaltered) matrices, as well as on their reordered versions using Reverse Cuthill-McKee (RCM).[36] For each combination of the sparse matrices, algorithms and reordering, the performance data is collected at runtime.

The obtained results are presented in Figure 8, when running the original (top part) and RCM-reordered (bottom part) matrices, and by subjecting each sparse matrix to the Intel MKL, followed by the Merge SpMV algorithm. For all cases, a set of PMU events were collected, these include `SCALAR_DOUBLE_INSTRUCTIONS`, `AVX512_ DOUBLE_INSTR.`, `TOTAL_MEMORY_INSTR.`, and `RAPL_POWER_PACKAGE`, with their evolution during the algorithm execution is depicted in Figure 8. As can be observed, there is a noticeable difference in the overall execution time required to process all five original (top) and reordered (bottom) matrices, where the reordered ones took about 22% less time for processing. This effect indicates the positive influence of reordering on improved data locality, which subsequently results in substantial performance improvements.

By focusing on the evolution of collected PMU events presented in Figure 8, one can observe that the `AVX512_DP_FP` events are only manifested during the Intel MKL execution, while the `SCALAR_DP_ FP` appear during the Merge algorithm runs. This is due to the ability of MKL SpMV implementation to take advantage of the Intel CPU's AVX512 capabilities, while Merge SpMV only

[32] Andrzej Nowak and Georgios Bitzes. *The overhead of profiling using PMU hardware counters*. 2014. DOI: 10.5281/zenodo.10800. URL: https://doi.org/10.5281/zenodo.10800.

[33] Endong Wang et al. "Intel Math Kernel Library". *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*. Springer International Publishing, 2014, pp. 167–188. DOI: 10.1007/978-3-319-06486-4_7. URL: https://doi.org/10.1007/978-3-319-06486-4_7.

[34] Duane Merrill and Michael Garland. "Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format". *Acm Sigplan Notices* 51.8 (2016), pp. 1–2.

[35] Tim Davis. *Sparse Matrix Collection*. Accessed on 5th October 2023. URL: https://sparse.tamu.edu/.

[36] Elizabeth Cuthill and James McKee. "Reducing the bandwidth of sparse symmetric matrices". *Proceedings of the 1969 24th national conference*. 1969, pp. 157–172.

Name	Group	Rows	Cols	Nnz
adaptive	DIMACS10	6,815,744	6,815,744	27,2M
audikw_1	GHS_psdef	943,695	943,695	77,7M
dielFilterV3real	Dziekonski	1,102,824	1,102,824	89,3M
hugetrace-00020	DIMACS10	16,002,413	16,002,413	48,0M
human_gene1	Belcastro	22,283	22,283	24,7M

Table 4 *Sparse matrices used in the experiment.*

exercised the scalar units (note the drop in AVX512 and the increase in scalar FP instructions at the vertical dashed lines, i.e., the points in time when MKL finishes and Merge starts its execution).

We can also observe that during the MKL execution, the measures for `RAPL_POWER_PACKAGE` and `TOTAL_MEMORY_INSTRUCTIONS` are lower than for Merge. This corroborates the fact that the codes using higher SIMD ISA may provoke reduced instruction counts when compared to their scalar counterparts (e.g., AVX512 load/store instructions involve 64-byte data transfer versus scalar memory instructions that operate on 8 bytes of data). This phenomenon, as well as data locality in different memory levels achieved with different algorithms and reordering, can provoke significant power consumption variations, as shown in Fig. 8.

5.5 LIVE-CARM FEATURE

To showcase the live-CARM feature in SuperTwin, we further analyze the performance differences between MKL and Merge SpMV algorithms, as well as three *likwid* benchmarks on the Intel CSL system (see Table 2).

SpMV Execution Profiling Figure 9 presents the live-CARM panel during the execution of both Intel MKL SpMV and Merge SpMV for the `hugetrace-00020` (see Table 4) in its original and RCM-reordered form. The live-CARM timestamps belonging to each execution phase are identified by the colored square that contains them, namely: *pink square* – Intel MKL; and *orange square* – Merge execution, while for both algorithms the *blue* and *green* squares denote the executions corresponding to the original and RCM-reordered matrix, respectively. As can be observed in the CARM plot, for each algorithm, the RCM reordering yielded higher performance, while we can also observe that the Intel MKL SpMV provides higher performance than the Merge SpMV (mainly due to its ability to exploit AVX512 SIMD capability). Furthermore, this study showcases how the Live-CARM dashboard can be used to make intuitive and insightful performance analyses across different applications and their execution phases during the run-time, as it allows pinpointing the data locality in different memory levels.

Benchmark Execution Profiling Live-CARM can also be used to profile benchmarks, by directly comparing the execution of a benchmark against the live-CARM roofs, i.e., the performance upper-bounds attainable on a target platform for different memory levels and compute units. This analysis provides a general idea on the ability of executed applications to fully exploit the capabilities of underlying hardware resources. For this purpose, various benchmarks from the *likwid* tool[37] (Triad, PeakFlops, and DDOT) were considered, with corresponding live-CARM reports presented in Figure 10.

The Triad benchmark (see orange points enclosed with green box) is a memory-bound benchmark with a theoretical AI of 0.625, which is accurately captured by the live-CARM in Fig. 10. As can be seen, the performance of this kernel approaches the L2 roof, but it is unable to surpass

[37]Röhl et al., "Overhead Analysis of Performance Counter Measurements".

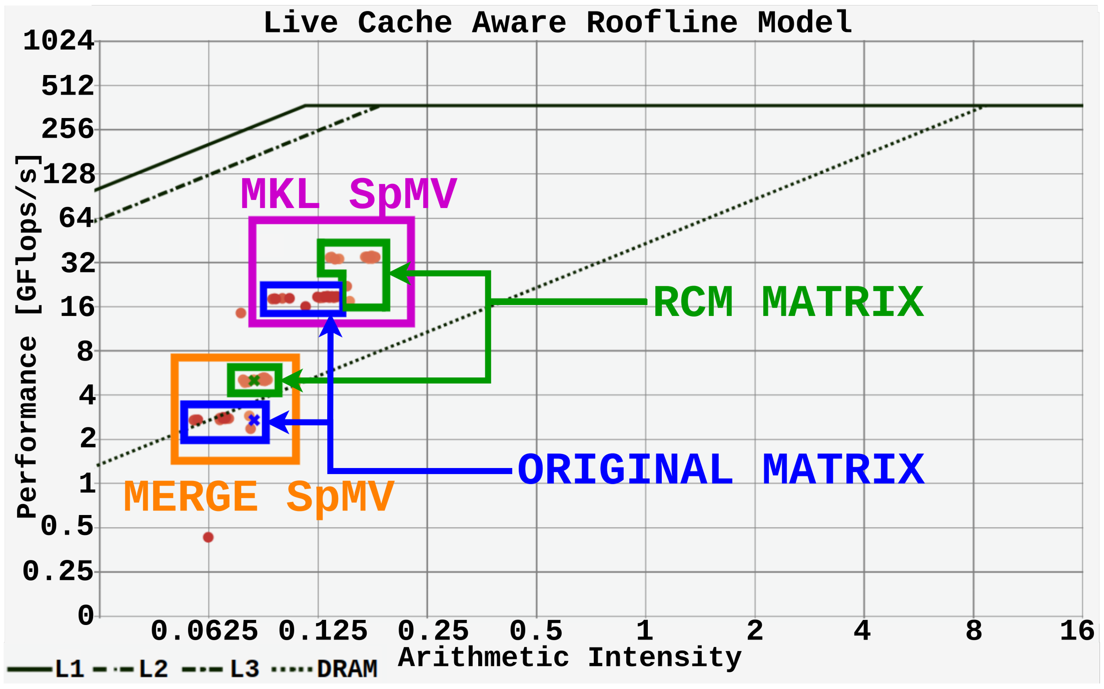


Figure 9 *Live-CARM during SpMV execution*

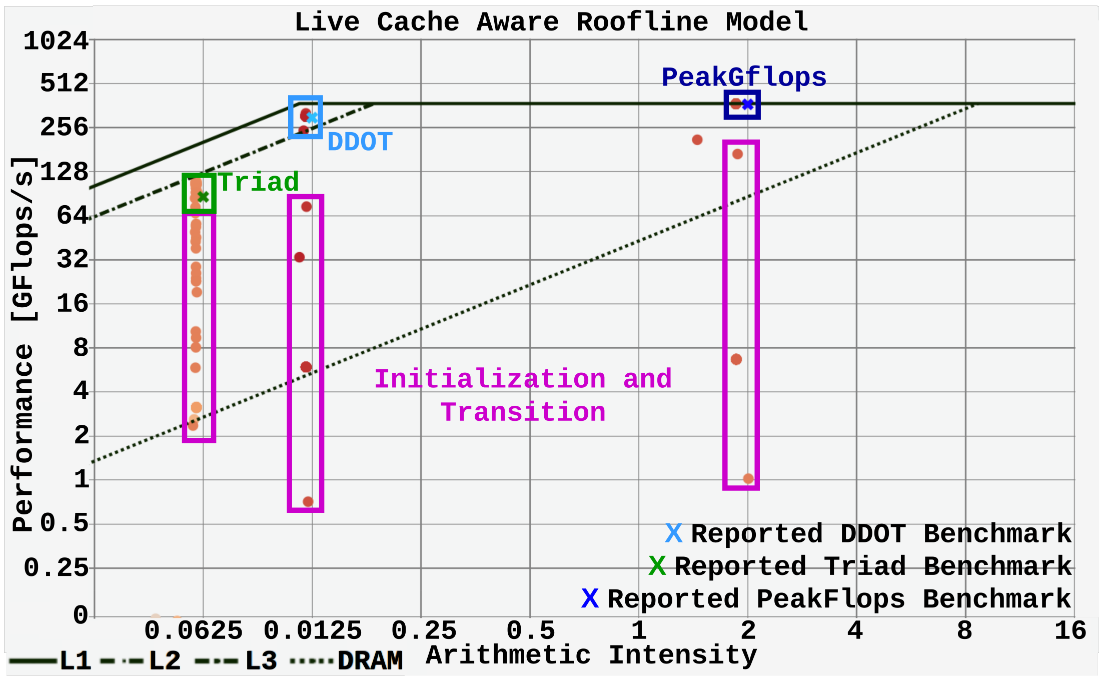


Figure 10 *Live-CARM during Likwid benchmarks execution*

it since the workload size does not fit in the 32Kb L1 cache. The PeakGflops benchmark (red dots enclosed with the dark blue box) is designed to reach the peak FP performance. With a theoretical AI of 2, this benchmark reports a performance very close to the one obtained with the CARM microbenchmarks (the application points aligned with the horizontal live-CARM roof in Fig. 10). Finally, similarly to Triad, the DDOT benchmark, is a memory-bound kernel that utilizes smaller problem sizes, thus able to fit in the L1 cache. As presented in Fig. 10 (see red dots with a light blue box), the theoretical DDOT AI of 0.125 is accurately captured by the live-CARM, with the performance surpassing the L2 roof, and approaching the maximum performance of the architecture.

6 CONCLUSION AND FUTURE WORK

In this deliverable, we present SuperTwin with an HPC-specific ontology, a knowledge base created on this ontology and proposed methods to parse the KB to detect performance variations/degradations in HPC environments. We demonstrated its lightweight remote performance profiling capabilities and presented its accuracy in the presence of transmission losses on high-frequency data over the network. Furthermore, it is equipped with the tools to compare performance metrics obtained from different systems which enables a heterogeneous performance analysis environment.

Overall, with SuperTwin, we aim to enhance the performance analysis and explainability landscape in multi-core architectures and facilitate architectural research. Future endeavours include gathering data from various systems and utilizing the dataset collected via SuPerfDB, the global performance database, for LLM training and building an AI tool for performance optimization. The design, as outlined in the deliverable, enables a straightforward extension of the framework from single-node servers to clusters. Based on the proposed design in this deliverable, we are on the verge of developing a cluster-level SuperTwin that encapsulates meticulous performance analysis and monitoring capabilities, in conjunction with communication telemetry and job-specific metadata emitted from HPC clusters.

7 CONTRIBUTIONS BY EACH PARTNER

SU, KU, and INESC-ID contributed to the design and implementation of SuperTwin, its components, API and documentation. Overall, the detailed design is a product of equal contributions by three partners. SU implemented the main components of the performance database.

8 DEVIATIONS (IF ANY)

There are no deviations.

REFERENCES

Adhianto. "HPCTOOLKIT: Tools for Performance Analysis of Optimized Parallel Programs Http://Hpctoolkit.Org". *Concurr. Comput.: Pract. Exper.* 22.6 (2010), pp. 685–701. ISSN: 1532-0626.

Agelastos. *The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications.* English. Tech. rep. SAND2014-19868C. Sandia National Lab. (SNL-NM), Albuquerque, NM (United States); Sandia National Lab. (SNL-CA), Livermore, CA (United States), 2014. DOI: 10.1109/SC.2014.18. URL: https://www.osti.gov/biblio/1315267 (visited on 09/27/2021).

Aksar. "E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems"". *Euro-Par 2021: Parallel Processing*. Springer International Publishing, 2021, pp. 70–85.

Brandt. *Lightweight Distributed Metric Service (LDMS): Run-time Resource Utilization Monitoring.* English. Tech. rep. SAND2013-6521C. Sandia National Lab. (SNL-CA), Livermore, CA (United States); Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), 2013. URL: https://www.osti.gov/biblio/1106397 (visited on 09/27/2021).

Chen, Xiaoli. "CERN Analysis Preservation: A Novel Digital Library Service to Enable Reusable and Reproducible Research". *Research and Advanced Technology for Digital Libraries*. Springer International Publishing, 2016, pp. 347–356.

Choi, Jee Whan et al. "A roofline model of energy". *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE. 2013, pp. 661–672.

Cluster Cockpit. https://www.clustercockpit.org/. Accessed on 30 Sep 2023.

Cuthill, Elizabeth and James McKee. "Reducing the bandwidth of sparse symmetric matrices". *Proceedings of the 1969 24th national conference*. 1969, pp. 157–172.

Davis, Tim. *Sparse Matrix Collection*. Accessed on 5th October 2023. URL: https://sparse.tamu.edu/.

Deng. "A systematic review of a digital twin city: A new pattern of urban governance toward smart cities". *Journal of Management Science and Engineering* 6.2 (2021), pp. 125–134. ISSN: 2096-2320. DOI: https://doi.org/10.1016/j.jmse.2021.03.003.

Ding, Nan and Samuel Williams. "An Instruction Roofline Model for GPUs". *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2019, pp. 7–18. DOI: 10.1109/PMBS49563.2019.00007.

Doerfler, Douglas et al. "Applying the roofline performance model to the intel xeon phi knights landing processor". *High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P^ 3MA, VHPC, WOPSSS, Frankfurt, Germany, June 19–23, 2016, Revised Selected Papers 31*. Springer. 2016, pp. 339–353.

Dongarra, Jack and Michael A Heroux. "Toward a new metric for ranking high performance computing systems". *Sandia Report, SAND2013-4744* 312 (2013), p. 150.

Friedemann. "Linked Data Architecture for Assistance and Traceability in Smart Manufacturing". *MATEC Web of Conferences* 304 (2019), p. 04006. DOI: 10.1051/matecconf/201930404006.

Ganglia. *Monitoring system*. 2022. URL: http://ganglia.sourceforge.net/ (visited on 12/12/2022).

Ilic, Aleksandar, Frederico Pratas, and Leonel Sousa. "Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores". *IEEE Transactions on Computers* 66.1 (2016), pp. 52–58.

— "Cache-aware roofline model: Upgrading the loft". *IEEE Computer Architecture Letters* 13.1 (2013), pp. 21–24.

Koskela, Tuomas et al. “A novel multi-level integrated roofline model approach for performance characterization”. *High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings 33*. Springer. 2018, pp. 226–245.

Lu, Qiuchen et al. “Developing a dynamic digital twin at building and city levels: A case study of the West Cambridge campus”. *Journal of Management in Engineering* 36 (2019). DOI: 10.1061/(ASCE)ME.1943-5479.0000763.

Marques, Diogo et al. “Application-driven cache-aware roofline model”. *Future Generation Computer Systems* 107 (2020), pp. 257–273.

McCalpin, John. “Memory bandwidth and machine balance in high performance computers”. *IEEE Technical Committee on Computer Architecture Newsletter* (1995), pp. 19–25.

Merrill, Duane and Michael Garland. “Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format”. *Acm Sigplan Notices* 51.8 (2016), pp. 1–2.

Milenković, Katarina. “Enabling Knowledge Management in Complex Industrial Processes Using Semantic Web Technology”. English. *Proceedings of the 2019 International Conference on Theory and Applications in the Knowledge Economy*. 2019 International Conference on Theory and Applications in the Knowledge Economy, TAKE 2019 ; Conference date: 03-07-2019 Through 05-01-2020. 2019. URL: https://www.take-conference2019.com/.

Nagios. *Nagios*. https://www.nagios.org/. Accessed: 2022-12-12. 2022.

Nowak, Andrzej and Georgios Bitzes. *The overhead of profiling using PMU hardware counters*. 2014. DOI: 10.5281/zenodo.10800. URL: https://doi.org/10.5281/zenodo.10800.

Performance Co-Pilot. https://pcp.io/. Accessed on 30 Sep 2023.

Röhl, Thomas et al. “LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses”. *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 2017, pp. 781–784. DOI: 10.1109/CLUSTER.2017.115.

Röhl, Thomas et al. “Overhead Analysis of Performance Counter Measurements”. *2014 43rd International Conference on Parallel Processing Workshops*. 2014, pp. 176–185. DOI: 10.1109/ICPPW.2014.34.

Roy. “PerfAugur: Robust diagnostics for performance anomalies in cloud services”. *2015 IEEE 31st International Conference on Data Engineering*. 2015, pp. 1167–1178. DOI: 10.1109/ICDE.2015.7113365.

Sasongko, Muhammad Aditya et al. “Precise Event Sampling on AMD Versus Intel: Quantitative and Qualitative Comparison”. *IEEE Transactions on Parallel and Distributed Systems* 34.5 (2023), pp. 1594–1608. ISSN: 1558-2183. DOI: 10.1109/TPDS.2023.3257105.

Steinmetz. “Internet of Things Ontology for Digital Twin in Cyber Physical Systems”. *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2018, pp. 154–159. DOI: 10.1109/SBESC.2018.00030.

Team, The Smartmontools. *Smartmontools*. Accessed on 5th October 2023. URL: https://www.smartmontools.org/.

Unat, Didem et al. “ExaSAT: An exascale co-design tool for performance modeling”. *The International Journal of High Performance Computing Applications* 29.2 (2015), pp. 209–232. DOI: 10.1177/1094342014568690. URL: https://doi.org/10.1177/1094342014568690.

Wang, Endong et al. “Intel Math Kernel Library”. *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*. Springer International Publishing, 2014, pp. 167–188. DOI: 10.1007/978-3-319-06486-4_7. URL: https://doi.org/10.1007/978-3-319-06486-4_7.

Weaver, Vincent, Dan Terpstra, and Shirley Moore. “Non-determinism and overcount on modern hardware performance counter implementations”. 2013, pp. 215–224. DOI: 10.1109/ISPASS.2013.6557172.

Weaver, Vincent M. et al. "Measuring Energy and Power with PAPI". *2012 41st International Conference on Parallel Processing Workshops*. 2012, pp. 262–268. doi: 10.1109/ICPPW.2012.39.
Xin. "Cross-linking BioThings APIs through JSON-LD to facilitate knowledge exploration". *BMC Bioinformatics* 19 (2018). doi: 10.1186/s12859-018-2041-5.

APPENDIX: API DOCUMENTATION

Due to the length of the API documentation, we are not including it in this document. You can find the API documentation through `https://people.sabanciuniv.edu/kaya/SuperTwin.pdf`.

8.1 history of changes

Version	Author(s)	Date	Comment
0.1	Kamer Kaya	22.03.2024	Initial version
0.2	Kamer Kaya	25.03.2024	Major revisions in subsections
0.2.1	Aleksandar Ilic	26.03.2024	Minor revisions and proofreading
0.2.2	Didem Unat	27.03.2024	Minor revisions and proofreading

Table 5 *Document History of Changes*